University of California Berkeley

Computer Science 162

Operating Systems and Systems Programming

Course Reader for Spring 2012

Professors: Anthony D. Joseph
Ion Stoica

Contents:

Nachos for Java README

Welcome to Nachos for Java. We believe that working in Java rather than C++ will greatly simplify the development process by preventing bugs arising from memory management errors, and improving debugging support.

Getting Nachos:

Download nachos-java.tar.gz from the Projects section of the class homepage at:

        http://www-inst.EECS.Berkeley.EDU/~cs162/

Unpack it with these commands:

        gunzip -c nachos-java.tar.gz | tar xf -

Additional software:

Nachos requires the Java Devlopment Kit, version 1.5 or later. This is installed on all instructional machines in:
        /usr/sww/lang/jdk-1.5.0_05
To use this version of the JDK, be sure that
        /usr/sww/lang/jdk-1.5.0_05/bin
is on your PATH. (This should be the case for all class accounts already.)

If you are working at home, you will need to download the JDK. It is available from:
        http://java.sun.com/j2se/1.5/
Please DO NOT DOWNLOAD the JDK into your class account! Use the preinstalled version instead.

The build process for Nachos relies on GNU make. If you are running on one of the instructional machines, be sure you run 'gmake', as 'make' does not support all the features used. If you are running Linux, the two are equivalent. If you are running Windows, you will need to download and install a port. The most popular is the Cygnus toolkit, available at:

        http://sources.redhat.com/cygwin/mirrors.html

The Cygnus package includes ports of most common GNU utilities to Windows.

For project 2, you will need a MIPS cross compiler, which is a specially compiled GCC which will run on one architecture (e.g. Sparc) and produce files for the MIPS processor. These compilers are already installed on the instructional machines, and are available in the directory specified by the $ARCHDIR environment variable.

If you are working at home, you will need to get a cross-compiler for yourself. Cross-compilers for Linux and Win32 will be available from the CS162 Projects web page. Download the cross compiler distribution and unpack it with the following command:

        gunzip -c mips-x86-linux-xgcc.tar.gz | tar xf -

(Substitute the appropriate file name for mips-x86.linux-xgcc in the above command.) You need to add the mips-x86.linux-xgcc directory to your PATH, and set an environment variable ARCHDIR to point to this directory. (Again, this has already been done for you on the instructional machines.)

Compiling Nachos:

You should now have a directory called nachos, containing a Makefile, this README, and a number of subdirectories.

First, put the 'nachos/bin' directory on your PATH. This directory contains the script 'nachos', which simply runs the Nachos code.

To compile Nachos, go to the subdirectory for the project you wish to compile (I will assume 'proj1/' for Project 1 in my examples), and run:

        gmake

This will compile those portions of Nachos which are relevant to the project, and place the compiled .class files in the proj1/nachos directory.

You can now test Nachos from the proj1/ directory with:

        nachos

You should see output resembling the following:

```
  nachos 5.0j initializing... config interrupt timer elevators user-check grader
  *** thread 0 looped 0 times
  *** thread 1 looped 0 times
  *** thread 0 looped 1 times
  *** thread 1 looped 1 times
  *** thread 0 looped 2 times
  *** thread 1 looped 2 times
  *** thread 0 looped 3 times
  *** thread 1 looped 3 times
  *** thread 0 looped 4 times
  *** thread 1 looped 4 times
  Machine halting!

  Ticks: total 24750, kernel 24750, user 0
  Disk I/O: reads 0, writes 0
  Console I/O: reads 0, writes 0
  Paging: page faults 0, TLB misses 0
  Network I/O: received 0, sent 0
```

This is the correct output for the "bare bones" Nachos, without any of the features you will add during the projects.

If you are working on a project which runs user programs (projects 2-4), you will also need to compile the MIPS test programs with:

        gmake test

Command Line Arguments:

For a summary of the command line arguments, run:

        nachos -h

The commands are:

```
        -d <debug flags>
                Enable some debug flags, e.g. -d ti

        -h
                Print this help message.

        -s <seed>
                Specify the seed for the random number generator

        -x <program>
                Specify a program that UserKernel.run() should execute,
                instead of the value of the configuration variable
                Kernel.shellProgram

        -z
                print the copyright message

        -- <grader class>
                Specify an autograder class to use, instead of
                nachos.ag.AutoGrader

        -# <grader arguments>
                Specify the argument string to pass to the autograder.

        -[] <config file>
                Specifiy a config file to use, instead of nachos.conf


Nachos offers the following debug flags:

    c: COFF loader info
    i: HW interrupt controller info
    p: processor info
    m: disassembly
    M: more disassembly
    t: thread info
    a: process info (formerly "address space", hence a)

To use multiple debug flags, clump them all together. For example, to
monitor coff info and process info, run:

        nachos -d ac

nachos.conf:

When Nachos starts, it reads in nachos.conf from the current
directory.  It contains a bunch of keys and values, in the simple
format "key = value" with one key/value pair per line. To change the
default scheduler, default shell program, to change the amount of
memory the simulator provides, or to reduce network reliability, modify
this file.

Machine.stubFileSystem:
    Specifies whether the machine should provide a stub file system. A
    stub file system just provides direct access to the test directory.
    Since we're not doing the file system project, this should always
    be true.

Machine.processor:
    Specifies whether the machine should provide a MIPS processor. In
    the first project, we only run kernel code, so this is false. In
    the other projects it should be true.

Machine.console:
    Specifies whether the machine should provide a console. Again, the
    first project doesn't need it, but the rest of them do.

Machine.disk:
    Specifies whether the machine should provide a simulated disk. No
    file system project, so this should always be false.

ElevatorBank.allowElevatorGUI:
    Normally true. When we grade, this will be false, to prevent
    malicious students from running a GUI during grading.

NachosSecurityManager.fullySecure:
    Normally false. When we grade, this will be true, to enable
    additional security checks.

Kernel.kernel:
    Specifies what kernel class to dynmically load.  For proj1, this is
    nachos.threads.ThreadedKernel. For proj2, this should be
    nachos.userprog.UserKernel. For proj3, nachos.vm.VMKernel. For
    proj4, nachos.network.NetKernel.

Processor.usingTLB:
    Specifies whether the MIPS processor provides a page table
    interface or a TLB interface. In page table mode (proj2), the
    processor accesses an arbitrarily large kernel data structure to do
    address translation. In TLB mode (proj3 and proj4), the processor
    maintains a small TLB (4 entries).

Processor.numPhysPages:
    The number of pages of physical memory.  Each page is 1K. This is
    normally 64, but we can lower it in proj3 to see whether projects
    thrash or crash.

Documentation:

The JDK provides a command to create a set of HTML pages showing all
classes and methods in program. We will make these pages available on
the webpage, but you can create your own for your home machine by doing
the following (from the nachos/ directory):

        mkdir ../doc
        gmake doc

Troubleshooting:

If you receive an error about "class not found exception", it may be
because you have not set the CLASSPATH environment variable. Add the
following to your .cshrc:

        setenv CLASSPATH .

Credits:

Nachos was originally written by Wayne A. Christopher, Steven J.
Procter, and Thomas E. Anderson. It incorporates the SPIM simulator
written by John Ousterhout. Nachos was rewritten in Java by Daniel
Hettena.

Copyright:
```

```
JAVADOCPARAMS = -doctitle "Nachos 5.0 Java" -protected \
                -link http://java.sun.com/j2se/1.5.0/docs/api/

machine =       Lib Config Stats Machine TCB \
                Interrupt Timer \
                Processor TranslationEntry \
                SerialConsole StandardConsole \
                OpenFile OpenFileWithPosition ArrayFile FileSystem StubFileSystem \
                ElevatorBank ElevatorTest ElevatorGui \
                ElevatorControls ElevatorEvent ElevatorControllerInterface \
                RiderControls RiderEvent RiderInterface \
                Kernel Coff CoffSection \
                NetworkLink Packet MalformedPacketException

security =      Privilege NachosSecurityManager

ag =            AutoGrader BoatGrader

threads =       ThreadedKernel KThread Alarm \
                Scheduler ThreadQueue RoundRobinScheduler \
                Semaphore Lock Condition SynchList \
                Condition2 Communicator Rider ElevatorController \
                PriorityScheduler LotteryScheduler Boat

userprog =      UserKernel UThread UserProcess SynchConsole

vm =            VMKernel VMProcess

network =       NetKernel NetProcess PostOffice MailMessage

ALLDIRS = machine security ag threads userprog vm network

PACKAGES := $(patsubst %,nachos.%,$(ALLDIRS))

CLASSFILES := $(foreach dir,$(DIRS),$(patsubst %,nachos/$(dir)/%.class,$($(dir))))

.PHONY: all rmtemp clean doc hwdoc swdoc

all: $(CLASSFILES)

nachos/%.class: ../%.java
        javac -classpath . -d . -sourcepath ../.. -g $<

clean:
        rm -f */*/*.class

doc:
        mkdir -p ../doc
        javadoc $(JAVADOCPARAMS) -d ../doc -sourcepath .. $(PACKAGES)

test:
        cd ../test ; gmake

ag:     $(patsubst ../ag/%.java,nachos/ag/%.class,$(wildcard ../ag/*.java))
```

```
// PART OF THE MACHINE SIMULATION. DO NOT CHANGE.

package nachos.ag;

import nachos.machine.*;
import nachos.security.*;
import nachos.threads.*;

import java.util.Hashtable;
import java.util.StringTokenizer;

/**
 * The default autograder. Loads the kernel, and then tests it using
 * <tt>Kernel.selfTest()</tt>.
 */
public class AutoGrader {
    /**
     * Allocate a new autograder.
     */
    public AutoGrader() {
    }

    /**
     * Start this autograder. Extract the <tt>-#</tt> arguments, call
     * <tt>init()</tt>, load and initialize the kernel, and call
     * <tt>run()</tt>.
     *
     * @param   privilege       encapsulates privileged access to the Nachos
     *                          machine.
     */
    public void start(Privilege privilege) {
        Lib.assertTrue(this.privilege == null,
                   "start() called multiple times");
        this.privilege = privilege;

        String[] args = Machine.getCommandLineArguments();

        extractArguments(args);

        System.out.print(" grader");

        init();

        System.out.print("\n");

        kernel =
            (Kernel) Lib.constructObject(Config.getString("Kernel.kernel"));
        kernel.initialize(args);

        run();
    }

    private void extractArguments(String[] args) {
        String testArgsString = Config.getString("AutoGrader.testArgs");
        if (testArgsString == null) {
                testArgsString = "";
        }

        for (int i=0; i<args.length; ) {
            String arg = args[i++];
            if (arg.length() > 0 && arg.charAt(0) == '-') {
                if (arg.equals("-#")) {
                    Lib.assertTrue(i < args.length,
                                   "-# switch missing argument");
                    testArgsString = args[i++];
                }
            }
        }

        StringTokenizer st = new StringTokenizer(testArgsString, ",\n\t\f\r");

        while (st.hasMoreTokens()) {
            StringTokenizer pair = new StringTokenizer(st.nextToken(), "=");

            Lib.assertTrue(pair.hasMoreTokens(),
                       "test argument missing key");
            String key = pair.nextToken();

            Lib.assertTrue(pair.hasMoreTokens(),
                       "test argument missing value");
            String value = pair.nextToken();

            testArgs.put(key, value);
        }
    }

    String getStringArgument(String key) {
        String value = (String) testArgs.get(key);
        Lib.assertTrue(value != null,
                   "getStringArgument(" + key + ") failed to find key");
        return value;
    }

    int getIntegerArgument(String key) {
        try {
            return Integer.parseInt(getStringArgument(key));
        }
        catch (NumberFormatException e) {
            Lib.assertNotReached("getIntegerArgument(" + key + ") failed: " +
                                 "value is not an integer");
            return 0;
        }
    }

    boolean getBooleanArgument(String key) {
        String value = getStringArgument(key);

        if (value.equals("1") || value.toLowerCase().equals("true")) {
            return true;
        }
        else if (value.equals("0") || value.toLowerCase().equals("false")) {
            return false;
        }
        else {
            Lib.assertNotReached("getBooleanArgument(" + key + ") failed: " +
                                 "value is not a boolean");
            return false;
        }
    }

    long getTime() {
        return privilege.stats.totalTicks;
    }
```

```
    void targetLevel(int targetLevel) {
        this.targetLevel = targetLevel;
    }

    void level(int level) {
        this.level++;
        Lib.assertTrue(level == this.level,
                       "level() advanced more than one step: test jumped ahead");

        if (level == targetLevel)
            done();
    }

    private int level = 0, targetLevel = 0;

    void done() {
        System.out.print("\nsuccess\n");
        privilege.exit(162);
    }

    private Hashtable<String, String> testArgs =
        new Hashtable<String, String>();

    void init() {
    }

    void run() {
        kernel.selfTest();
        kernel.run();
        kernel.terminate();
    }

    Privilege privilege = null;
    Kernel kernel;

    /**
     * Notify the autograder that the specified thread is the idle thread.
     * <tt>KThread.createIdleThread()</tt> <i>must</i> call this method before
     * forking the idle thread.
     *
     * @param   idleThread      the idle thread.
     */
    public void setIdleThread(KThread idleThread) {
    }

    /**
     * Notify the autograder that the specified thread has moved to the ready
     * state. <tt>KThread.ready()</tt> <i>must</i> call this method before
     * returning.
     *
     * @param   thread  the thread that has been added to the ready set.
     */
    public void readyThread(KThread thread) {
    }

    /**
     * Notify the autograder that the specified thread is now running.
     * <tt>KThread.restoreState()</tt> <i>must</i> call this method before
     * returning.
     *
     * @param   thread  the thread that is now running.
     */
    public void runningThread(KThread thread) {
        privilege.tcb.associateThread(thread);
        currentThread = thread;
    }

    /**
     * Notify the autograder that the current thread has finished.
     * <tt>KThread.finish()</tt> <i>must</i> call this method before putting
     * the thread to sleep and scheduling its TCB to be destroyed.
     */
    public void finishingCurrentThread() {
        privilege.tcb.authorizeDestroy(currentThread);
    }

    /**
     * Notify the autograder that a timer interrupt occurred and was handled by
     * software if a timer interrupt handler was installed. Called by the
     * hardware timer.
     *
     * @param   privilege       proves the authenticity of this call.
     * @param   time    the actual time at which the timer interrupt was
     *                  issued.
     */
    public void timerInterrupt(Privilege privilege, long time) {
        Lib.assertTrue(privilege == this.privilege,
                       "security violation");
    }

    /**
     * Notify the autograder that a user program executed a syscall
     * instruction.
     *
     * @param   privilege       proves the authenticity of this call.
     * @return  <tt>true</tt> if the kernel exception handler should be called.
     */
    public boolean exceptionHandler(Privilege privilege) {
        Lib.assertTrue(privilege == this.privilege,
                       "security violation");
        return true;
    }

    /**
     * Notify the autograder that <tt>Processor.run()</tt> was invoked. This
     * can be used to simulate user programs.
     *
     * @param   privilege       proves the authenticity of this call.
     */
    public void runProcessor(Privilege privilege) {
        Lib.assertTrue(privilege == this.privilege,
                       "security violation");
    }

    /**
     * Notify the autograder that a COFF loader is being constructed for the
     * specified file. The autograder can use this to provide its own COFF
     * loader, or return <tt>null</tt> to use the default loader.
     *
     * @param   file    the executable file being loaded.
     * @return  a loader to use in loading the file, or <tt>null</tt> to use
     *          the default.
     */
    public Coff createLoader(OpenFile file) {
```

```
        return null;
    }

    /**
     * Request permission to send a packet. The autograder can use this to drop
     * packets very selectively.
     *
     * @param   privilege       proves the authenticity of this call.
     * @return  <tt>true</tt> if the packet should be sent.
     */
    public boolean canSendPacket(Privilege privilege) {
        Lib.assertTrue(privilege == this.privilege,
                       "security violation");
        return true;
    }

    /**
     * Request permission to receive a packet. The autograder can use this to
     * drop packets very selectively.
     *
     * @param   privilege       proves the authenticity of this call.
     * @return  <tt>true</tt> if the packet should be delivered to the kernel.
     */
    public boolean canReceivePacket(Privilege privilege) {
        Lib.assertTrue(privilege == this.privilege,
                       "security violation");
        return true;
    }

    private KThread currentThread;
}
```

```
package nachos.ag;

public class BoatGrader {

    /**
     * BoatGrader consists of functions to be called to show that
     * your solution is properly synchronized. This version simply
     * prints messages to standard out, so that you can watch it.
     * You cannot submit this file, as we will be using our own
     * version of it during grading.

     * Note that this file includes all possible variants of how
     * someone can get from one island to another. Inclusion in
     * this class does not imply that any of the indicated actions
     * are a good idea or even allowed.
     */

    /* ChildRowToMolokai should be called when a child pilots the boat
       from Oahu to Molokai */
    public void ChildRowToMolokai() {
        System.out.println("**Child rowing to Molokai.");
    }

    /* ChildRowToOahu should be called when a child pilots the boat
       from Molokai to Oahu*/
    public void ChildRowToOahu() {
        System.out.println("**Child rowing to Oahu.");
    }

    /* ChildRideToMolokai should be called when a child not piloting
       the boat disembarks on Molokai */
    public void ChildRideToMolokai() {
        System.out.println("**Child arrived on Molokai as a passenger.");
    }

    /* ChildRideToOahu should be called when a child not piloting
       the boat disembarks on Oahu */
    public void ChildRideToOahu() {
        System.out.println("**Child arrived on Oahu as a passenger.");
    }

    /* AdultRowToMolokai should be called when a adult pilots the boat
       from Oahu to Molokai */
    public void AdultRowToMolokai() {
        System.out.println("**Adult rowing to Molokai.");
    }

    /* AdultRowToOahu should be called when a adult pilots the boat
       from Molokai to Oahu */
    public void AdultRowToOahu() {
        System.out.println("**Adult rowing to Oahu.");
    }

    /* AdultRideToMolokai should be called when an adult not piloting
       the boat disembarks on Molokai */
    public void AdultRideToMolokai() {
        System.out.println("**Adult arrived on Molokai as a passenger.");
    }

    /* AdultRideToOahu should be called when an adult not piloting
       the boat disembarks on Oahu */
    public void AdultRideToOahu() {
        System.out.println("**Adult arrived on Oahu as a passenger.");
    }
}
```

```
#!/bin/sh

# Shell-script front-end to run Nachos.
# Simply sets terminal to a minimum of one byte to complete a read and
# disables character echo. Restores original terminal state on exit.

onexit () {
  stty $OLDSTTYSTATE
}

OLDSTTYSTATE=‘stty -g‘
trap onexit 0
stty -icanon min 1 -echo
java nachos.machine.Machine $*
```

```
// PART OF THE MACHINE SIMULATION. DO NOT CHANGE.

package nachos.machine;

/**
 * A read-only <tt>OpenFile</tt> backed by a byte array.
 */
public class ArrayFile extends OpenFileWithPosition {
    /**
     * Allocate a new <tt>ArrayFile</tt>.
     *
     * @param   array   the array backing this file.
     */
    public ArrayFile(byte[] array) {
        this.array = array;
    }

    public int length() {
        return array.length;
    }

    public void close() {
        array = null;
    }

    public int read(int position, byte[] buf, int offset, int length) {
        Lib.assertTrue(offset >= 0 && length >= 0 && offset+length <= buf.length);

        if (position < 0 || position >= array.length)
            return 0;

        length = Math.min(length, array.length-position);
        System.arraycopy(array, position, buf, offset, length);

        return length;
    }

    public int write(int position, byte[] buf, int offset, int length) {
        return 0;
    }

    private byte[] array;
}
```

```
// PART OF THE MACHINE SIMULATION. DO NOT CHANGE.

package nachos.machine;

import java.io.EOFException;

/**
 * A COFF (common object file format) loader.
 */
public class Coff {
    /**
     * Allocate a new Coff object.
     */
    protected Coff() {
        file = null;
        entryPoint = 0;
        sections = null;
    }

    /**
     * Load the COFF executable in the specified file.
     *
     * <p>
     * Notes:
     * <ol>
     * <li>If the constructor returns successfully, the file becomes the
     * property of this loader, and should not be accessed any further.
     * <li>The autograder expects this loader class to be used. Do not load
     * sections through any other mechanism.
     * <li>This loader will verify that the file is backed by a file system,
     * by asserting that read() operations take non-zero simulated time to
     * complete. Do not supply a file backed by a simulated cache (the primary
     * purpose of this restriction is to prevent sections from being loaded
     * instantaneously while handling page faults).
     * </ol>
     *
     * @param   file    the file containing the executable.
     * @exception       EOFException    if the executable is corrupt.
     */
    public Coff(OpenFile file) throws EOFException {
        this.file = file;

        Coff coff = Machine.autoGrader().createLoader(file);

        if (coff != null) {
            this.entryPoint = coff.entryPoint;
            this.sections = coff.sections;
        }
        else {
            byte[] headers = new byte[headerLength+aoutHeaderLength];

            if (file.length() < headers.length) {
                Lib.debug(dbgCoff, "\tfile is not executable");
                throw new EOFException();
            }

            Lib.strictReadFile(file, 0, headers, 0, headers.length);

            int magic = Lib.bytesToUnsignedShort(headers, 0);
            int numSections = Lib.bytesToUnsignedShort(headers, 2);
            int optionalHeaderLength = Lib.bytesToUnsignedShort(headers, 16);
            int flags = Lib.bytesToUnsignedShort(headers, 18);

            entryPoint = Lib.bytesToInt(headers, headerLength+16);

            if (magic != 0x0162) {
                Lib.debug(dbgCoff, "\tincorrect magic number");
                throw new EOFException();
            }
            if (numSections < 2 || numSections > 10) {
                Lib.debug(dbgCoff, "\tbad section count");
                throw new EOFException();
            }
            if ((flags & 0x0003) != 0x0003) {
                Lib.debug(dbgCoff, "\tbad header flags");
                throw new EOFException();
            }

            int offset = headerLength + optionalHeaderLength;

            sections = new CoffSection[numSections];
            for (int s=0; s<numSections; s++) {
                int sectionEntryOffset = offset + s*CoffSection.headerLength;
                try {
                    sections[s] =
                        new CoffSection(file, this, sectionEntryOffset);
                }
                catch (EOFException e) {
                    Lib.debug(dbgCoff, "\terror loading section " + s);
                    throw e;
                }
            }
        }
    }

    /**
     * Return the number of sections in the executable.
     *
     * @return  the number of sections in the executable.
     */
    public int getNumSections() {
        return sections.length;
    }

    /**
     * Return an object that can be used to access the specified section. Valid
     * section numbers include <tt>0</tt> through <tt>getNumSections() -
     * 1</tt>.
     *
     * @param   sectionNumber   the section to select.
     * @return  an object that can be used to access the specified section.
     */
    public CoffSection getSection(int sectionNumber) {
        Lib.assertTrue(sectionNumber >= 0 && sectionNumber < sections.length);

        return sections[sectionNumber];
    }

    /**
     * Return the program entry point. This is the value that to which the PC
     * register should be initialized to before running the program.
     *
     * @return  the program entry point.
     */
    public int getEntryPoint() {
```

```
        Lib.assertTrue(file != null);

        return entryPoint;
    }

    /**
     * Close the executable file and release any resources allocated by this
     * loader.
     */
    public void close() {
        file.close();

        sections = null;
    }

    private OpenFile file;

    /** The virtual address of the first instruction of the program. */
    protected int entryPoint;
    /** The sections in this COFF executable. */
    protected CoffSection sections[];

    private static final int headerLength = 20;
    private static final int aoutHeaderLength = 28;

    private static final char dbgCoff = 'c';
}
```

```
// PART OF THE MACHINE SIMULATION. DO NOT CHANGE.

package nachos.machine;

import nachos.security.*;

import java.io.EOFException;
import java.util.Arrays;

/**
 * A <tt>CoffSection</tt> manages a single section within a COFF executable.
 */
public class CoffSection {
    /**
     * Allocate a new COFF section with the specified parameters.
     *
     * @param   coff            the COFF object to which this section belongs.
     * @param   name            the COFF name of this section.
     * @param   executable      <tt>true</tt> if this section contains code.
     * @param   readOnly        <tt>true</tt> if this section is read-only.
     * @param   numPages        the number of virtual pages in this section.
     * @param   firstVPN        the first virtual page number used by this.
     */
    protected CoffSection(Coff coff, String name, boolean executable,
                          boolean readOnly, int numPages, int firstVPN) {
        this.coff = coff;
        this.name = name;
        this.executable = executable;
        this.readOnly = readOnly;
        this.numPages = numPages;
        this.firstVPN = firstVPN;

        file = null;
        size = 0;
        contentOffset = 0;
        initialized = true;
    }

    /**
     * Load a COFF section from an executable.
     *
     * @param   file            the file containing the executable.
     * @param   headerOffset    the offset of the section header in the
     *                          executable.
     *
     * @exception       EOFException    if an error occurs.
     */
    public CoffSection(OpenFile file, Coff coff,
                       int headerOffset) throws EOFException {
        this.file = file;
        this.coff = coff;

        Lib.assertTrue(headerOffset >= 0);
        if (headerOffset+headerLength > file.length()) {
            Lib.debug(dbgCoffSection, "\tsection header truncated");
            throw new EOFException();
        }

        byte[] buf = new byte[headerLength];
        Lib.strictReadFile(file, headerOffset, buf, 0, headerLength);

        name = Lib.bytesToString(buf, 0, 8);
        int vaddr = Lib.bytesToInt(buf, 12);
        size = Lib.bytesToInt(buf, 16);
        contentOffset = Lib.bytesToInt(buf, 20);
        int numRelocations = Lib.bytesToUnsignedShort(buf, 32);
        int flags = Lib.bytesToInt(buf, 36);

        if (numRelocations != 0) {
            Lib.debug(dbgCoffSection, "\tsection needs relocation");
            throw new EOFException();
        }

        switch (flags & 0x0FFF) {
        case 0x0020:
            executable = true;
            readOnly = true;
            initialized = true;
            break;
        case 0x0040:
            executable = false;
            readOnly = false;
            initialized = true;
            break;
        case 0x0080:
            executable = false;
            readOnly = false;
            initialized = false;
            break;
        case 0x0100:
            executable = false;
            readOnly = true;
            initialized = true;
            break;
        default:
            Lib.debug(dbgCoffSection, "\tinvalid section flags: " + flags);
            throw new EOFException();
        }

        if (vaddr%Processor.pageSize != 0 || size < 0 ||
            initialized && (contentOffset < 0 ||
                            contentOffset+size > file.length())) {
            Lib.debug(dbgCoffSection, "\tinvalid section addresses: " +
                      "vaddr=" + vaddr + " size=" + size +
                      " contentOffset=" + contentOffset);
            throw new EOFException();
        }

        numPages = Lib.divRoundUp(size, Processor.pageSize);
        firstVPN = vaddr / Processor.pageSize;
    }

    /**
     * Return the COFF object used to load this executable instance.
     *
     * @return  the COFF object corresponding to this section.
     */
    public Coff getCoff() {
        return coff;
    }

    /**
     * Return the name of this section.
     *
```

```
     * @return  the name of this section.
     */
    public String getName() {
        return name;
    }

    /**
     * Test whether this section is read-only.
     *
     * @return  <tt>true</tt> if this section should never be written.
     */
    public boolean isReadOnly() {
        return readOnly;
    }

    /**
     * Test whether this section is initialized. Loading a page from an
     * initialized section requires a disk access, while loading a page from an
     * uninitialized section requires only zero-filling the page.
     *
     * @return  <tt>true</tt> if this section contains initialized data in the
     *          executable.
     */
    public boolean isInitialzed() {
        return initialized;
    }

    /**
     * Return the length of this section in pages.
     *
     * @return  the number of pages in this section.
     */
    public int getLength() {
        return numPages;
    }

    /**
     * Return the first virtual page number used by this section.
     *
     * @return  the first virtual page number used by this section.
     */
    public int getFirstVPN() {
        return firstVPN;
    }

    /**
     * Load a page from this segment into physical memory.
     *
     * @param   spn     the page number within this segment.
     * @param   ppn     the physical page to load into.
     */
    public void loadPage(int spn, int ppn) {
        Lib.assertTrue(file != null);

        Lib.assertTrue(spn>=0 && spn<numPages);
        Lib.assertTrue(ppn>=0 && ppn<Machine.processor().getNumPhysPages());

        int pageSize = Processor.pageSize;
        byte[] memory = Machine.processor().getMemory();
        int paddr = ppn*pageSize;
        int faddr = contentOffset + spn*pageSize;
        int initlen;

        if (!initialized)
            initlen = 0;
        else if (spn == numPages-1)
            /** initlen = size % pageSize;
             *  Bug identified by Steven Schlansker 3/20/08
             *  Bug fix by Michael Rauser
             */
            initlen = (size==pageSize) ? pageSize : (size%pageSize);
        else
            initlen = pageSize;

        if (initlen > 0)
            Lib.strictReadFile(file, faddr, memory, paddr, initlen);

        Arrays.fill(memory, paddr+initlen, paddr+pageSize, (byte) 0);
    }

    /** The COFF object to which this section belongs. */
    protected Coff coff;
    /** The COFF name of this section. */
    protected String name;
    /** True if this section contains code. */
    protected boolean executable;
    /** True if this section is read-only. */
    protected boolean readOnly;
    /** True if this section contains initialized data. */
    protected boolean initialized;

    /** The number of virtual pages in this section. */
    protected int numPages;
    /** The first virtual page number used by this section. */
    protected int firstVPN;

    private OpenFile file;
    private int contentOffset, size;

    /** The length of a COFF section header. */
    public static final int headerLength = 40;

    private static final char dbgCoffSection = 'c';
}
```

```
// PART OF THE MACHINE SIMULATION. DO NOT CHANGE.

package nachos.machine;

import java.util.HashMap;
import java.io.File;
import java.io.FileReader;
import java.io.Reader;
import java.io.StreamTokenizer;

/**
 * Provides routines to access the Nachos configuration.
 */
public final class Config {
    /**
     * Load configuration information from the specified file. Must be called
     * before the Nachos security manager is installed.
     *
     * @param   fileName        the name of the file containing the
     *                          configuration to use.
     */
    public static void load(String fileName) {
        System.out.print(" config");

        Lib.assertTrue(!loaded);
        loaded = true;

        configFile = fileName;

        try {
            config = new HashMap<String, String>();

            File file = new File(configFile);
            Reader reader = new FileReader(file);
            StreamTokenizer s = new StreamTokenizer(reader);

            s.resetSyntax();
            s.whitespaceChars(0x00, 0x20);
            s.wordChars(0x21, 0xFF);
            s.eolIsSignificant(true);
            s.commentChar('#');
            s.quoteChar('"');

            int line = 1;

            s.nextToken();

            while (true) {
                if (s.ttype == StreamTokenizer.TT_EOF)
                    break;

                if (s.ttype == StreamTokenizer.TT_EOL) {
                    line++;
                    s.nextToken();
                    continue;
                }

                if (s.ttype != StreamTokenizer.TT_WORD)
                    loadError(line);

                String key = s.sval;

                if (s.nextToken() != StreamTokenizer.TT_WORD ||
                    !s.sval.equals("="))
                    loadError(line);

                if (s.nextToken() != StreamTokenizer.TT_WORD && s.ttype != '"')
                    loadError(line);

                String value = s.sval;

                // ignore everything after first string
                while (s.nextToken() != StreamTokenizer.TT_EOL &&
                       s.ttype != StreamTokenizer.TT_EOF);

                if (config.get(key) != null)
                    loadError(line);

                config.put(key, value);
                line++;
            }
        }
        catch (Throwable e) {
            System.err.println("Error loading " + configFile);
            System.exit(1);
        }
    }

    private static void loadError(int line) {
        System.err.println("Error in " + configFile + " line " + line);
        System.exit(1);
    }

    private static void configError(String message) {
        System.err.println("");
        System.err.println("Error in " + configFile + ": " + message);
        System.exit(1);
    }

    /**
     * Get the value of a key in <tt>nachos.conf</tt>.
     *
     * @param   key     the key to look up.
     * @return  the value of the specified key, or <tt>null</tt> if it is not
     *          present.
     */
    public static String getString(String key) {
        return (String) config.get(key);
    }

    /**
     * Get the value of a key in <tt>nachos.conf</tt>, returning the specified
     * default if the key does not exist.
     *
     * @param   key             the key to look up.
     * @param   defaultValue    the value to return if the key does not exist.
     * @return  the value of the specified key, or <tt>defaultValue</tt> if it
     *          is not present.
     */
    public static String getString(String key, String defaultValue) {
        String result = getString(key);

        if (result == null)
            return defaultValue;
```

```
    return result;
}

private static Integer requestInteger(String key) {
    try {
        String value = getString(key);
        if (value == null)
            return null;

        return new Integer(value);
    }
    catch (NumberFormatException e) {
        configError(key + " should be an integer");

        Lib.assertNotReached();
        return null;
    }
}

/**
 * Get the value of an integer key in <tt>nachos.conf</tt>.
 *
 * @param   key     the key to look up.
 * @return  the value of the specified key.
 */
public static int getInteger(String key) {
    Integer result = requestInteger(key);

    if (result == null)
        configError("missing int " + key);

    return result.intValue();
}

/**
 * Get the value of an integer key in <tt>nachos.conf</tt>, returning the
 * specified default if the key does not exist.
 *
 * @param   key             the key to look up.
 * @param   defaultValue    the value to return if the key does not exist.
 * @return  the value of the specified key, or <tt>defaultValue</tt> if the
 *          key does not exist.
 */
public static int getInteger(String key, int defaultValue) {
    Integer result = requestInteger(key);

    if (result == null)
        return defaultValue;

    return result.intValue();
}

private static Double requestDouble(String key) {
    try {
        String value = getString(key);
        if (value == null)
            return null;

        return new Double(value);
    }
    catch (NumberFormatException e) {
        configError(key + " should be a double");

        Lib.assertNotReached();
        return null;
    }
}

/**
 * Get the value of a double key in <tt>nachos.conf</tt>.
 *
 * @param   key     the key to look up.
 * @return  the value of the specified key.
 */
public static double getDouble(String key) {
    Double result = requestDouble(key);

    if (result == null)
        configError("missing double " + key);

    return result.doubleValue();
}

/**
 * Get the value of a double key in <tt>nachos.conf</tt>, returning the
 * specified default if the key does not exist.
 *
 * @param   key             the key to look up.
 * @param   defaultValue    the value to return if the key does not exist.
 * @return  the value of the specified key, or <tt>defaultValue</tt> if the
 *          key does not exist.
 */
public static double getDouble(String key, double defaultValue) {
    Double result = requestDouble(key);

    if (result == null)
        return defaultValue;

    return result.doubleValue();
}

private static Boolean requestBoolean(String key) {
    String value = getString(key);

    if (value == null)
        return null;

    if (value.equals("1") || value.toLowerCase().equals("true")) {
        return Boolean.TRUE;
    }
    else if (value.equals("0") || value.toLowerCase().equals("false")) {
        return Boolean.FALSE;
    }
    else {
        configError(key + " should be a boolean");

        Lib.assertNotReached();
        return null;
    }
}

/**
 * Get the value of a boolean key in <tt>nachos.conf</tt>.
```

```
     *
     * @param   key     the key to look up.
     * @return  the value of the specified key.
     */
    public static boolean getBoolean(String key) {
        Boolean result = requestBoolean(key);

        if (result == null)
            configError("missing boolean " + key);

        return result.booleanValue();
    }

    /**
     * Get the value of a boolean key in <tt>nachos.conf</tt>, returning the
     * specified default if the key does not exist.
     *
     * @param   key             the key to look up.
     * @param   defaultValue    the value to return if the key does not exist.
     * @return  the value of the specified key, or <tt>defaultValue</tt> if the
     *          key does not exist.
     */
    public static boolean getBoolean(String key, boolean defaultValue) {
        Boolean result = requestBoolean(key);

        if (result == null)
            return defaultValue;

        return result.booleanValue();
    }

    private static boolean loaded = false;
    private static String configFile;
    private static HashMap<String, String> config;
}
```

```
// PART OF THE MACHINE SIMULATION. DO NOT CHANGE.

package nachos.machine;

import nachos.security.*;
import nachos.threads.KThread;
import nachos.threads.Semaphore;

import java.util.Vector;
import java.util.LinkedList;
import java.util.Iterator;

/**
 * A bank of elevators.
 */
public final class ElevatorBank implements Runnable {
    /** Indicates an elevator intends to move down. */
    public static final int dirDown = -1;
    /** Indicates an elevator intends not to move. */
    public static final int dirNeither = 0;
    /** Indicates an elevator intends to move up. */
    public static final int dirUp = 1;

    /**
     * Allocate a new elevator bank.
     *
     * @param   privilege       encapsulates privileged access to the Nachos
     *                          machine.
     */
    public ElevatorBank(Privilege privilege) {
        System.out.print(" elevators");

        this.privilege = privilege;

        simulationStarted = false;
    }

    /**
     * Initialize this elevator bank with the specified number of elevators and
     * the specified number of floors. The software elevator controller must
     * also be specified. This elevator must not already be running a
     * simulation.
     *
     * @param   numElevators    the number of elevators in the bank.
     * @param   numFloors       the number of floors in the bank.
     * @param   controller      the elevator controller.
     */
    public void init(int numElevators, int numFloors,
                     ElevatorControllerInterface controller) {
        Lib.assertTrue(!simulationStarted);

        this.numElevators = numElevators;
        this.numFloors = numFloors;

        manager = new ElevatorManager(controller);

        elevators = new ElevatorState[numElevators];
        for (int i=0; i<numElevators; i++)
            elevators[i] = new ElevatorState(0);

        numRiders = 0;
        ridersVector = new Vector<RiderControls>();

        enableGui = false;
        gui = null;
    }

    /**
     * Add a rider to the simulation. This method must not be called after
     * <tt>run()</tt> is called.
     *
     * @param   rider   the rider to add.
     * @param   floor   the floor the rider will start on.
     * @param   stops   the array to pass to the rider's <tt>initialize()</tt>
     *                  method.
     * @return  the controls that will be given to the rider.
     */
    public RiderControls addRider(RiderInterface rider,
                                  int floor, int[] stops) {
        Lib.assertTrue(!simulationStarted);

        RiderControls controls = new RiderState(rider, floor, stops);
        ridersVector.addElement(controls);
        numRiders++;
        return controls;
    }

    /**
     * Create a GUI for this elevator bank.
     */
    public void enableGui() {
        Lib.assertTrue(!simulationStarted);
        Lib.assertTrue(Config.getBoolean("ElevatorBank.allowElevatorGUI"));

        enableGui = true;
    }

    /**
     * Run a simulation. Initialize all elevators and riders, and then
     * fork threads to each of their <tt>run()</tt> methods. Return when the
     * simulation is finished.
     */
    public void run() {
        Lib.assertTrue(!simulationStarted);
        simulationStarted = true;

        riders = new RiderState[numRiders];
        ridersVector.toArray(riders);

        if (enableGui) {
            privilege.doPrivileged(new Runnable() {
                public void run() { initGui(); }
            });
        }

        for (int i=0; i<numRiders; i++)
            riders[i].initialize();
        manager.initialize();

        for (int i=0; i<numRiders; i++)
            riders[i].run();
        manager.run();

        for (int i=0; i<numRiders; i++)
```

```
            riders[i].join();
        manager.join();

        simulationStarted = false;
    }

    private void initGui() {
        int[] numRidersPerFloor = new int[numFloors];
        for (int floor=0; floor<numFloors; floor++)
            numRidersPerFloor[floor] = 0;

        for (int rider=0; rider<numRiders; rider++)
            numRidersPerFloor[riders[rider].floor]++;

        gui = new ElevatorGui(numFloors, numElevators, numRidersPerFloor);
    }

    /**
     * Tests whether this module is working.
     */
    public static void selfTest() {
        new ElevatorTest().run();
    }

    void postRiderEvent(int event, int floor, int elevator) {
        int direction = dirNeither;
        if (elevator != -1) {
            Lib.assertTrue(elevator >= 0 && elevator < numElevators);
            direction = elevators[elevator].direction;
        }

        RiderEvent e = new RiderEvent(event, floor, elevator, direction);
        for (int i=0; i<numRiders; i++) {
            RiderState rider = riders[i];
            if ((rider.inElevator && rider.elevator == e.elevator) ||
                (!rider.inElevator && rider.floor == e.floor)) {
                rider.events.add(e);
                rider.schedule(1);
            }
        }
    }

    private class ElevatorManager implements ElevatorControls {
        ElevatorManager(ElevatorControllerInterface controller) {
            this.controller = controller;

            interrupt = new Runnable() { public void run() { interrupt(); }};
        }

        public int getNumFloors() {
            return numFloors;
        }

        public int getNumElevators() {
            return numElevators;
        }

        public void setInterruptHandler(Runnable handler) {
            this.handler = handler;
        }

        public void openDoors(int elevator) {
            Lib.assertTrue(elevator >= 0 && elevator < numElevators);
            postRiderEvent(RiderEvent.eventDoorsOpened,
                           elevators[elevator].openDoors(), elevator);

            if (gui != null) {
                if (elevators[elevator].direction == dirUp)
                    gui.clearUpButton(elevators[elevator].floor);
                else if (elevators[elevator].direction == dirDown)
                    gui.clearDownButton(elevators[elevator].floor);

                gui.openDoors(elevator);
            }
        }

        public void closeDoors(int elevator) {
            Lib.assertTrue(elevator >= 0 && elevator < numElevators);
            postRiderEvent(RiderEvent.eventDoorsClosed,
                           elevators[elevator].closeDoors(), elevator);

            if (gui != null)
                gui.closeDoors(elevator);
        }

        public boolean moveTo(int floor, int elevator) {
            Lib.assertTrue(floor >= 0 && floor < numFloors);
            Lib.assertTrue(elevator >= 0 && elevator < numElevators);

            if (!elevators[elevator].moveTo(floor))
                return false;

            schedule(Stats.ElevatorTicks);
            return true;
        }

        public int getFloor(int elevator) {
            Lib.assertTrue(elevator >= 0 && elevator < numElevators);
            return elevators[elevator].floor;
        }

        public void setDirectionDisplay(int elevator, int direction) {
            Lib.assertTrue(elevator >= 0 && elevator < numElevators);
            elevators[elevator].direction = direction;

            if (elevators[elevator].doorsOpen) {
                postRiderEvent(RiderEvent.eventDirectionChanged,
                               elevators[elevator].floor, elevator);
            }

            if (gui != null) {
                if (elevators[elevator].doorsOpen) {
                    if (direction == dirUp)
                        gui.clearUpButton(elevators[elevator].floor);
                    else if (direction == dirDown)
                        gui.clearDownButton(elevators[elevator].floor);
                }

                gui.setDirectionDisplay(elevator, direction);
            }
        }

        public void finish() {
            finished = true;
```

```
    Lib.assertTrue(KThread.currentThread() == thread);

    done.V();
    KThread.finish();
}

public ElevatorEvent getNextEvent() {
    if (events.isEmpty())
        return null;
    else
        return (ElevatorEvent) events.removeFirst();
}

void schedule(int when) {
    privilege.interrupt.schedule(when, "elevator", interrupt);
}

void postEvent(int event, int floor, int elevator, boolean schedule) {
    events.add(new ElevatorEvent(event, floor, elevator));

    if (schedule)
        schedule(1);
}

void interrupt() {
    for (int i=0; i<numElevators; i++) {
        if (elevators[i].atNextFloor()) {
            if (gui != null)
                gui.elevatorMoved(elevators[i].floor, i);

            if (elevators[i].atDestination()) {
                postEvent(ElevatorEvent.eventElevatorArrived,
                          elevators[i].destination, i, false);
            }
            else {
                elevators[i].nextETA += Stats.ElevatorTicks;
                privilege.interrupt.schedule(Stats.ElevatorTicks,
                                             "elevator",
                                             interrupt);
            }
        }
    }

    if (!finished && !events.isEmpty() && handler != null)
        handler.run();
}

void initialize() {
    controller.initialize(this);
}

void run() {
    thread = new KThread(controller);
    thread.setName("elevator controller");
    thread.fork();
}

void join() {
    postEvent(ElevatorEvent.eventRidersDone, -1, -1, true);
    done.P();
}
```

```
    ElevatorControllerInterface controller;
    Runnable interrupt;
    KThread thread;

    Runnable handler = null;
    LinkedList<ElevatorEvent> events = new LinkedList<ElevatorEvent>();
    Semaphore done = new Semaphore(0);
    boolean finished = false;
}

private class ElevatorState {
    ElevatorState(int floor) {
        this.floor = floor;
        destination = floor;
    }

    int openDoors() {
        Lib.assertTrue(!doorsOpen && !moving);
        doorsOpen = true;
        return floor;
    }

    int closeDoors() {
        Lib.assertTrue(doorsOpen);
        doorsOpen = false;
        return floor;
    }

    boolean moveTo(int newDestination) {
        Lib.assertTrue(!doorsOpen);

        if (!moving) {
            // can't move to current floor
            if (floor == newDestination)
                return false;

            destination = newDestination;
            nextETA = Machine.timer().getTime() + Stats.ElevatorTicks;

            moving = true;
            return true;
        }
        else {
            // moving, shouldn't be at destination
            Lib.assertTrue(floor != destination);

            // make sure it's ok to stop
            if ((destination > floor && newDestination <= floor) ||
                (destination < floor && newDestination >= floor))
                return false;

            destination = newDestination;
            return true;
        }
    }

    boolean enter(RiderState rider, int onFloor) {
        Lib.assertTrue(!riders.contains(rider));

        if (!doorsOpen || moving || onFloor != floor ||
            riders.size() == maxRiders)
```

```
            return false;

        riders.addElement(rider);
        return true;
    }

    boolean exit(RiderState rider, int onFloor) {
        Lib.assertTrue(riders.contains(rider));

        if (!doorsOpen || moving || onFloor != floor)
            return false;

        riders.removeElement(rider);
        return true;
    }

    boolean atNextFloor() {
        if (!moving || Machine.timer().getTime() < nextETA)
            return false;

        Lib.assertTrue(destination != floor);
        if (destination > floor)
            floor++;
        else
            floor--;

        for (Iterator i=riders.iterator(); i.hasNext(); ) {
            RiderState rider = (RiderState) i.next();

            rider.floor = floor;
        }

        return true;
    }

    boolean atDestination() {
        if (!moving || destination != floor)
            return false;

        moving = false;
        return true;
    }

    static final int maxRiders = 4;

    int floor, destination;
    long nextETA;

    boolean doorsOpen = false, moving = false;
    int direction = dirNeither;
    public Vector<RiderState> riders = new Vector<RiderState>();
}

private class RiderState implements RiderControls {
    RiderState(RiderInterface rider, int floor, int[] stops) {
        this.rider = rider;
        this.floor = floor;
        this.stops = stops;

        interrupt = new Runnable() { public void run() { interrupt(); }};
    }
```

```
public int getNumFloors() {
    return numFloors;
}

public int getNumElevators() {
    return numElevators;
}

public void setInterruptHandler(Runnable handler) {
    this.handler = handler;
}

public int getFloor() {
    return floor;
}

public int[] getFloors() {
    int[] array = new int[floors.size()];
    for (int i=0; i<array.length; i++)
        array[i] = ((Integer) floors.elementAt(i)).intValue();

    return array;
}

public int getDirectionDisplay(int elevator) {
    Lib.assertTrue(elevator >= 0 && elevator < numElevators);
    return elevators[elevator].direction;
}

public RiderEvent getNextEvent() {
    if (events.isEmpty())
        return null;
    else
        return (RiderEvent) events.removeFirst();
}

public boolean pressDirectionButton(boolean up) {
    if (up)
        return pressUpButton();
    else
        return pressDownButton();
}

public boolean pressUpButton() {
    Lib.assertTrue(!inElevator && floor < numFloors-1);

    for (int elevator=0; elevator<numElevators; elevator++) {
        if (elevators[elevator].doorsOpen &&
            elevators[elevator].direction == ElevatorBank.dirUp &&
            elevators[elevator].floor == floor)
            return false;
    }

    manager.postEvent(ElevatorEvent.eventUpButtonPressed,
                      floor, -1, true);

    if (gui != null)
        gui.pressUpButton(floor);

    return true;
}
```

```
    public boolean pressDownButton() {
        Lib.assertTrue(!inElevator && floor > 0);

        for (int elevator=0; elevator<numElevators; elevator++) {
            if (elevators[elevator].doorsOpen &&
                elevators[elevator].direction == ElevatorBank.dirDown &&
                elevators[elevator].floor == floor)
                return false;
        }

        manager.postEvent(ElevatorEvent.eventDownButtonPressed,
                          floor, -1, true);

        if (gui != null)
            gui.pressDownButton(floor);

        return true;
    }

    public boolean enterElevator(int elevator) {
        Lib.assertTrue(!inElevator &&
                   elevator >= 0 && elevator < numElevators);
        if (!elevators[elevator].enter(this, floor))
            return false;

        if (gui != null)
            gui.enterElevator(floor, elevator);

        inElevator = true;
        this.elevator = elevator;
        return true;
    }

    public boolean pressFloorButton(int floor) {
        Lib.assertTrue(inElevator && floor >= 0 && floor < numFloors);

        if (elevators[elevator].doorsOpen &&
            elevators[elevator].floor == floor)
            return false;

        manager.postEvent(ElevatorEvent.eventFloorButtonPressed,
                          floor, elevator, true);

        if (gui != null)
            gui.pressFloorButton(floor, elevator);

        return true;
    }

    public boolean exitElevator(int floor) {
        Lib.assertTrue(inElevator && floor >= 0 && floor < numFloors);

        if (!elevators[elevator].exit(this, floor))
            return false;

        inElevator = false;
        floors.add(new Integer(floor));

        if (gui != null)
            gui.exitElevator(floor, elevator);

        return true;
        }

        public void finish() {
            finished = true;

            int[] floors = getFloors();
            Lib.assertTrue(floors.length == stops.length);
            for (int i=0; i<floors.length; i++)
                Lib.assertTrue(floors[i] == stops[i]);

            Lib.assertTrue(KThread.currentThread() == thread);

            done.V();
            KThread.finish();
        }

        void schedule(int when) {
            privilege.interrupt.schedule(when, "rider", interrupt);
        }

        void interrupt() {
            if (!finished && !events.isEmpty() && handler != null)
                handler.run();
        }

        void initialize() {
            rider.initialize(this, stops);
        }

        void run() {
            thread = new KThread(rider);
            thread.setName("rider");
            thread.fork();
        }

        void join() {
            done.P();
        }

        RiderInterface rider;
        boolean inElevator = false, finished = false;
        int floor, elevator;
        int[] stops;
        Runnable interrupt, handler = null;
        LinkedList<RiderEvent> events = new LinkedList<RiderEvent>();
        Vector<Integer> floors = new Vector<Integer>();
        Semaphore done = new Semaphore(0);
        KThread thread;
    }

    private int numFloors, numElevators;
    private ElevatorManager manager;
    private ElevatorState[] elevators;

    private int numRiders;
    private Vector<RiderControls> ridersVector;
    private RiderState[] riders;

    private boolean simulationStarted, enableGui;
    private Privilege privilege;
    private ElevatorGui gui;
}
```

```
// PART OF THE MACHINE SIMULATION. DO NOT CHANGE.

package nachos.machine;

/**
 * A controller for all the elevators in an elevator bank. The controller
 * accesses the elevator bank through an instance of <tt>ElevatorControls</tt>.
 */
public interface ElevatorControllerInterface extends Runnable {
    /**
     * Initialize this elevator controller. The controller will access the
     * elevator bank through <i>controls</i>. This constructor should return
     * immediately after this controller is initialized, but not until the
     * interupt handler is set. The controller will start receiving events
     * after this method returns, but potentially before <tt>run()</tt> is
     * called.
     *
     * @param   controls        the controller's interface to the elevator
     *                          bank. The controller must not attempt to access
     *                          the elevator bank in <i>any</i> other way.
     */
    public void initialize(ElevatorControls controls);

    /**
     * Cause the controller to use the provided controls to receive and process
     * requests from riders. This method should not return, but instead should
     * call <tt>controls.finish()</tt> when the controller is finished.
     */
    public void run();

    /** The number of ticks doors should be held open before closing them. */
    public static final int timeDoorsOpen = 500;

    /** Indicates an elevator intends to move down. */
    public static final int dirDown = -1;
    /** Indicates an elevator intends not to move. */
    public static final int dirNeither = 0;
    /** Indicates an elevator intends to move up. */
    public static final int dirUp = 1;
}
```

```
// PART OF THE MACHINE SIMULATION. DO NOT CHANGE.

package nachos.machine;

/**
 * A set of controls that can be used by an elevator controller.
 */
public interface ElevatorControls {
    /**
     * Return the number of floors in the elevator bank. If <i>n</i> is the
     * number of floors in the bank, then the floors are numbered <i>0</i>
     * (the ground floor) through <i>n - 1</i> (the top floor).
     *
     * @return  the number of floors in the bank.
     */
    public int getNumFloors();

    /**
     * Return the number of elevators in the elevator bank. If <i>n</i> is the
     * number of elevators in the bank, then the elevators are numbered
     * <i>0</i> through <i>n - 1</i>.
     *
     * @return  the numbe rof elevators in the bank.
     */
    public int getNumElevators();

    /**
     * Set the elevator interrupt handler. This handler will be called when an
     * elevator event occurs, and when all the riders have reaced their
     * destinations.
     *
     * @param   handler the elevator interrupt handler.
     */
    public void setInterruptHandler(Runnable handler);

    /**
     * Open an elevator's doors.
     *
     * @param   elevator        which elevator's doors to open.
     */
    public void openDoors(int elevator);

    /**
     * Close an elevator's doors.
     *
     * @param   elevator        which elevator's doors to close.
     */
    public void closeDoors(int elevator);

    /**
     * Move an elevator to another floor. The elevator's doors must be closed.
     * If the elevator is already moving and cannot safely stop at the
     * specified floor because it has already passed or is about to pass the
     * floor, fails and returns <tt>false</tt>. If the elevator is already
     * stopped at the specified floor, returns <tt>false</tt>.
     *
     * @param   floor           the floor to move to.
     * @param   elevator        the elevator to move.
     * @return  <tt>true</tt> if the elevator's destination was changed.
     */
    public boolean moveTo(int floor, int elevator);

    /**
     * Return the current location of the elevator. If the elevator is in
     * motion, the returned value will be within one of the exact location.
     *
     * @param   elevator        the elevator to locate.
     * @return  the floor the elevator is on.
     */
    public int getFloor(int elevator);

    /**
     * Set which direction the elevator bank will show for this elevator's
     * display. The <i>direction</i> argument should be one of the <i>dir*</i>
     * constants in the <tt>ElevatorBank</tt> class.
     *
     * @param   elevator        the elevator whose direction display to set.
     * @param   direction       the direction to show (up, down, or neither).
     */
    public void setDirectionDisplay(int elevator, int direction);

    /**
     * Call when the elevator controller is finished.
     */
    public void finish();

    /**
     * Return the next event in the event queue. Note that there may be
     * multiple events pending when an elevator interrupt occurs, so this
     * method should be called repeatedly until it returns <tt>null</tt>.
     *
     * @return  the next event, or <tt>null</tt> if no further events are
     * currently pending.
     */
    public ElevatorEvent getNextEvent();
}
```

```
// PART OF THE MACHINE SIMULATION. DO NOT CHANGE.

package nachos.machine;

/**
 * An event that affects elevator software.
 */
public final class ElevatorEvent {
    public ElevatorEvent(int event, int floor, int elevator) {
        this.event = event;
        this.floor = floor;
        this.elevator = elevator;
    }

    /** The event identifier. Refer to the <i>event*</i> constants. */
    public final int event;
    /** The floor pertaining to the event, or -1 if not applicable. */
    public final int floor;
    /** The elevator pertaining to the event, or -1 if not applicable. */
    public final int elevator;

    /** An up button was pressed. */
    public static final int eventUpButtonPressed = 0;
    /** A down button was pressed. */
    public static final int eventDownButtonPressed = 1;
    /** A floor button was pressed inside an elevator. */
    public static final int eventFloorButtonPressed = 2;
    /** An elevator has arrived and stopped at its destination floor. */
    public static final int eventElevatorArrived = 3;
    /** All riders have finished; the elevator controller should terminate. */
    public static final int eventRidersDone = 4;
}
```

```
// PART OF THE MACHINE SIMULATION. DO NOT CHANGE.

package nachos.machine;

import java.awt.Frame;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.Panel;
import java.awt.ScrollPane;
import java.awt.Canvas;

import java.awt.GridLayout;
import java.awt.FlowLayout;
import java.awt.Insets;

import java.awt.Rectangle;
import java.awt.Graphics;
import java.awt.Color;

/**
 * A graphical visualization for the <tt>ElevatorBank</tt> class.
 */
public final class ElevatorGui extends Frame {
    private final static int w=90, h=75;

    private int numFloors, numElevators;

    private ElevatorShaft[] elevators;
    private Floor[] floors;

    private int totalWidth, totalHeight;

    ElevatorGui(int numFloors, int numElevators, int[] numRidersPerFloor) {
        this.numFloors = numFloors;
        this.numElevators = numElevators;

        totalWidth = w*(numElevators+1);
        totalHeight = h*numFloors;

        setTitle("Elevator Bank");

        Panel floorPanel = new Panel(new GridLayout(numFloors, 1, 0, 0));

        floors = new Floor[numFloors];
        for (int i=numFloors-1; i>=0; i--) {
            floors[i] = new Floor(i, numRidersPerFloor[i]);
            floorPanel.add(floors[i]);
        }

        Panel panel = new Panel(new GridLayout(1, numElevators+1, 0, 0));

        panel.add(floorPanel);

        elevators = new ElevatorShaft[numElevators];
        for (int i=0; i<numElevators; i++) {
            elevators[i] = new ElevatorShaft(i);
            panel.add(elevators[i]);
        }

        add(panel);
        pack();

        setVisible(true);

        repaint();
    }

    void openDoors(int elevator) {
        elevators[elevator].openDoors();
    }

    void closeDoors(int elevator) {
        elevators[elevator].closeDoors();
    }

    void setDirectionDisplay(int elevator, int direction) {
        elevators[elevator].setDirectionDisplay(direction);
    }

    void pressUpButton(int floor) {
        floors[floor].pressUpButton();
    }

    void clearUpButton(int floor) {
        floors[floor].clearUpButton();
    }

    void pressDownButton(int floor) {
        floors[floor].pressDownButton();
    }

    void clearDownButton(int floor) {
        floors[floor].clearDownButton();
    }

    void enterElevator(int floor, int elevator) {
        floors[floor].removeRider();
        elevators[elevator].addRider();
    }

    void pressFloorButton(int floor, int elevator) {
        elevators[elevator].pressFloorButton(floor);
    }

    void exitElevator(int floor, int elevator) {
        elevators[elevator].removeRider();
        floors[floor].addRider();
    }

    void elevatorMoved(int floor, int elevator) {
        elevators[elevator].elevatorMoved(floor);
    }

    private void paintRider(Graphics g, int x, int y, int r) {
        g.setColor(Color.yellow);

        g.fillOval(x-r, y-r, 2*r, 2*r);

        g.setColor(Color.black);

        g.fillOval(x-r/2, y-r/2, r/3, r/3);
        g.fillOval(x+r/4, y-r/2, r/3, r/3);

        g.drawArc(x-r/2, y-r/2, r, r, 210, 120);
```

```
    }

    private void paintRiders(Graphics g, int x, int y, int w, int h, int n) {
        int r = 8, t = 20;

        int xn = w/t;
        int yn = h/t;

        int x0 = x + (w-xn*t)/2 + t/2;
        int y0 = y + h - t/2;

        for (int j=0; j<yn; j++) {
            for (int i=0; i<xn; i++) {
                if (n-- > 0)
                    paintRider(g, x0 + i*t, y0 - j*t, r);
            }
        }
    }

    private class Floor extends Canvas {
        int floor, numRiders;

        boolean upSet = false;
        boolean downSet = false;

        Floor(int floor, int numRiders) {
            this.floor = floor;
            this.numRiders = numRiders;

            setBackground(Color.black);
        }

        public Dimension getPreferredSize() {
            return new Dimension(w, h);
        }

        public Dimension getMinimumSize() {
            return getPreferredSize();
        }

        public Dimension getMaximumSize() {
            return getPreferredSize();
        }

        public void repaint() {
            super.repaint();

            if (TCB.isNachosThread()) {
                try {
                    Thread.sleep(100);
                }
                catch (InterruptedException e) {
                }
            }
        }

        void pressUpButton() {
            if (!upSet) {
                upSet = true;
                repaint();
            }
        }

        void pressDownButton() {
            if (!downSet) {
                downSet = true;
                repaint();
            }
        }

        void clearUpButton() {
            if (upSet) {
                upSet = false;
                repaint();
            }
        }

        void clearDownButton() {
            if (downSet) {
                downSet = false;
                repaint();
            }
        }

        void addRider() {
            numRiders++;

            repaint();
        }

        void removeRider() {
            numRiders--;

            repaint();
        }

        public void paint(Graphics g) {
            g.setColor(Color.lightGray);
            g.drawLine(0, 0, w, 0);

            paintRiders(g, 0, 5, 3*w/4, h-10, numRiders);

            paintButtons(g);
        }

        private void paintButtons(Graphics g) {
            int s = 3*w/4;

            int x1 = s+w/32;
            int x2 = w-w/32;
            int y1 = h/8;
            int y2 = h-h/8;

            g.setColor(Color.darkGray);
            g.drawRect(x1, y1, x2-x1, y2-y1);
            g.setColor(Color.lightGray);
            g.fillRect(x1+1, y1+1, x2-x1-2, y2-y1-2);

            int r = Math.min((x2-x1)/3, (y2-y1)/6);
            int xc = (x1+x2)/2;
            int yc1 = (y1+y2)/2 - (3*r/2);
            int yc2 = (y1+y2)/2 + (3*r/2);

            g.setColor(Color.red);
```

```
        if (floor < numFloors-1) {
            if (upSet)
                g.fillOval(xc-r, yc1-r, 2*r, 2*r);
            else
                g.drawOval(xc-r, yc1-r, 2*r, 2*r);
        }

        if (floor > 0) {
            if (downSet)
                g.fillOval(xc-r, yc2-r, 2*r, 2*r);
            else
                g.drawOval(xc-r, yc2-r, 2*r, 2*r);
        }
    }
}

private class ElevatorShaft extends Canvas {
    ElevatorShaft(int elevator) {
        this.elevator = elevator;

        floorsSet = new boolean[numFloors];
        for (int i=0; i<numFloors; i++)
            floorsSet[i] = false;

        setBackground(Color.black);
    }

    public Dimension getPreferredSize() {
        return new Dimension(w, h*numFloors);
    }

    public Dimension getMinimumSize() {
        return getPreferredSize();
    }

    public Dimension getMaximumSize() {
        return getPreferredSize();
    }

    private void repaintElevator() {
        repaint(s, h*(numFloors-1-Math.max(floor, prevFloor)), w-2*s, h*2);

        try {
            Thread.sleep(100);
        }
        catch (InterruptedException e) {
        }
    }

    void openDoors() {
        doorsOpen = true;

        repaintElevator();
    }

    void closeDoors() {
        doorsOpen = false;

        repaintElevator();
    }

    void setDirectionDisplay(int direction) {
        this.direction = direction;

        repaintElevator();
    }

    void pressFloorButton(int floor) {
        if (!floorsSet[floor]) {
            floorsSet[floor] = true;

            repaintElevator();
        }
    }

    void elevatorMoved(int floor) {
        prevFloor = this.floor;
        this.floor = floor;

        floorsSet[floor] = false;

        repaintElevator();
    }

    void addRider() {
        numRiders++;

        repaintElevator();
    }

    void removeRider() {
        numRiders--;

        repaintElevator();
    }

    public void paint(Graphics g) {
        g.setColor(Color.lightGray);

        if (g.hitClip(0, 0, s, h*numFloors)) {
            g.drawLine(0, 0, 0, h*numFloors);
            g.drawLine(s-1, 0, s-1, h*numFloors);
            for (int y=0; y<h*numFloors-s; y+=s)
                g.drawLine(0, y, s-1, y+s-1);
        }

        if (g.hitClip(w-s, 0, s, h*numFloors)) {
            g.drawLine(w-s, 0, w-s, h*numFloors);
            g.drawLine(w-1, 0, w-1, h*numFloors);
            for (int y=0; y<h*numFloors-s; y+=s)
                g.drawLine(w-s, y, w-1, y+s-1);
        }

        // rectangle containing direction display area
        Rectangle d = new Rectangle(s*3/2, h*(numFloors-1-floor),
                                    w-3*s, w/3-s);

        // unit of measurement in direction rect (12ux4u)
        int u = d.width/12;

        // draw elevator, fill riders
        Rectangle e = new Rectangle(d.x, d.y+d.height,
                                    d.width, h-d.height-u);
```

```
            g.drawRect(e.x, e.y, e.width, e.height);
            paintRiders(g, e.x, e.y, e.width, e.height, numRiders);

            g.setColor(Color.lightGray);

            // draw doors...
            if (doorsOpen) {
                g.drawLine(e.x+2*s, e.y, e.x+2*s, e.y+e.height);
                for (int y=0; y<e.height-2*s; y+=2*s)
                    g.drawLine(e.x, e.y+y, e.x+2*s, e.y+y+2*s);

                g.drawLine(e.x+e.width-2*s, e.y,
                           e.x+e.width-2*s, e.y+e.height);
                for (int y=0; y<e.height-2*s; y+=2*s)
                    g.drawLine(e.x+e.width-2*s, e.y+y, e.x+e.width, e.y+y+2*s);
            }
            else {
                for (int x=0; x<e.width; x+=2*s)
                    g.drawLine(e.x+x, e.y, e.x+x, e.y+e.height);
            }

            g.setColor(Color.yellow);

            int[] xUp = { d.x + u*6, d.x + u*8, d.x + u*7 };
            int[] yUp = { d.y + u*3, d.y + u*3, d.y + u*1 };

            int[] xDown = { d.x + u*4, d.x + u*6, d.x + u*5 };
            int[] yDown = { d.y + u*1, d.y + u*1, d.y + u*3 };

            // draw arrows
            if (direction == ElevatorBank.dirUp)
                g.fillPolygon(xUp, yUp, 3);
            else
                g.drawPolygon(xUp, yUp, 3);

            if (direction == ElevatorBank.dirDown)
                g.fillPolygon(xDown, yDown, 3);
            else
                g.drawPolygon(xDown, yDown, 3);
        }

        private static final int s = 5;

        private boolean doorsOpen = false;
        private int floor = 0, prevFloor = 0, numRiders = 0;
        private int direction = ElevatorBank.dirNeither;

        private int elevator;

        private boolean floorsSet[];
    }
}
```

```
// PART OF THE MACHINE SIMULATION. DO NOT CHANGE.

package nachos.machine;

import nachos.security.*;
import nachos.threads.KThread;
import nachos.threads.Semaphore;

/**
 * Tests the <tt>ElevatorBank</tt> module, using a single elevator and a single
 * rider.
 */
public final class ElevatorTest {
    /**
     * Allocate a new <tt>ElevatorTest</tt> object.
     */
    public ElevatorTest() {
    }

    /**
     * Run a test on <tt>Machine.bank()</tt>.
     */
    public void run() {
        Machine.bank().init(1, 2, new ElevatorController());

        int[] stops = { 1 };

        Machine.bank().addRider(new Rider(), 0, stops);

        Machine.bank().run();
    }

    private class ElevatorController implements ElevatorControllerInterface {
        public void initialize(ElevatorControls controls) {
            this.controls = controls;

            eventWait = new Semaphore(0);

            controls.setInterruptHandler(new Runnable() {
                    public void run() { interrupt(); }
                });
        }

        public void run() {
            ElevatorEvent e;

            Lib.assertTrue(controls.getFloor(0) == 0);

            e = getNextEvent();
            Lib.assertTrue(e.event == ElevatorEvent.eventUpButtonPressed &&
                       e.floor == 0);

            controls.setDirectionDisplay(0, dirUp);
            controls.openDoors(0);

            e = getNextEvent();
            Lib.assertTrue(e.event == ElevatorEvent.eventFloorButtonPressed &&
                       e.floor == 1);

            controls.closeDoors(0);
            controls.moveTo(1, 0);

            e = getNextEvent();
            Lib.assertTrue(e.event == ElevatorEvent.eventElevatorArrived &&
                       e.floor == 1 &&
                       e.elevator == 0);

            controls.openDoors(0);

            e = getNextEvent();
            Lib.assertTrue(e.event == ElevatorEvent.eventRidersDone);

            controls.finish();
            Lib.assertNotReached();
        }

        private void interrupt() {
            eventWait.V();
        }

        private ElevatorEvent getNextEvent() {
            ElevatorEvent event;
            while (true) {
                if ((event = controls.getNextEvent()) != null)
                    break;

                eventWait.P();
            }
            return event;
        }

        private ElevatorControls controls;
        private Semaphore eventWait;
    }

    private class Rider implements RiderInterface {
        public void initialize(RiderControls controls, int[] stops) {
            this.controls = controls;
            Lib.assertTrue(stops.length == 1 && stops[0] == 1);

            eventWait = new Semaphore(0);

            controls.setInterruptHandler(new Runnable() {
                    public void run() { interrupt(); }
                });
        }

        public void run() {
            RiderEvent e;

            Lib.assertTrue(controls.getFloor() == 0);

            controls.pressUpButton();

            e = getNextEvent();
            Lib.assertTrue(e.event == RiderEvent.eventDoorsOpened &&
                       e.floor == 0 &&
                       e.elevator == 0);
            Lib.assertTrue(controls.getDirectionDisplay(0) == dirUp);

            Lib.assertTrue(controls.enterElevator(0));
            controls.pressFloorButton(1);

            e = getNextEvent();
```

```
            Lib.assertTrue(e.event == RiderEvent.eventDoorsClosed &&
                       e.floor == 0 &&
                       e.elevator == 0);

            e = getNextEvent();
            Lib.assertTrue(e.event == RiderEvent.eventDoorsOpened &&
                       e.floor == 1 &&
                       e.elevator == 0);

            Lib.assertTrue(controls.exitElevator(1));

            controls.finish();
            Lib.assertNotReached();
        }

        private void interrupt() {
            eventWait.V();
        }

        private RiderEvent getNextEvent() {
            RiderEvent event;
            while (true) {
                if ((event = controls.getNextEvent()) != null)
                    break;

                eventWait.P();
            }
            return event;
        }

        private RiderControls controls;
        private Semaphore eventWait;
    }
}
```

```
// PART OF THE MACHINE SIMULATION. DO NOT CHANGE.

package nachos.machine;

/**
 * A file system that allows the user to create, open, and delete files.
 */
public interface FileSystem {
    /**
     * Atomically open a file, optionally creating it if it does not
     * already exist. If the file does not
     * already exist and <tt>create</tt> is <tt>false</tt>, returns
     * <tt>null</tt>. If the file does not already exist and <tt>create</tt>
     * is <tt>true</tt>, creates the file with zero length. If the file already
     * exists, opens the file without changing it in any way.
     *
     * @param   name            the name of the file to open.
     * @param   create          <tt>true</tt> to create the file if it does not
     *                          already exist.
     * @return  an <tt>OpenFile</tt> representing a new instance of the opened
     *          file, or <tt>null</tt> if the file could not be opened.
     */
    public OpenFile open(String name, boolean create);

    /**
     * Atomically remove an existing file. After a file is removed, it cannot
     * be opened until it is created again with <tt>open</tt>. If the file is
     * already open, it is up to the implementation to decide whether the file
     * can still be accessed or if it is deleted immediately.
     *
     * @param   name    the name of the file to remove.
     * @return  <tt>true</tt> if the file was successfully removed.
     */
    public boolean remove(String name);
}
```

```
// PART OF THE MACHINE SIMULATION. DO NOT CHANGE.

package nachos.machine;

import nachos.security.*;

import java.util.TreeSet;
import java.util.Iterator;
import java.util.SortedSet;

/**
 * The <tt>Interrupt</tt> class emulates low-level interrupt hardware. The
 * hardware provides a method (<tt>setStatus()</tt>) to enable or disable
 * interrupts.
 *
 * <p>
 * In order to emulate the hardware, we need to keep track of all pending
 * interrupts the hardware devices would cause, and when they are supposed to
 * occur.
 *
 * <p>
 * This module also keeps track of simulated time. Time advances only when the
 * following occur:
 * <ul>
 * <li>interrupts are enabled, when they were previously disabled
 * <li>a MIPS instruction is executed
 * </ul>
 *
 * <p>
 * As a result, unlike real hardware, interrupts (including time-slice context
 * switches) cannot occur just anywhere in the code where interrupts are
 * enabled, but rather only at those places in the code where simulated time
 * advances (so that it becomes time for the hardware simulation to invoke an
 * interrupt handler).
 *
 * <p>
 * This means that incorrectly synchronized code may work fine on this hardware
 * simulation (even with randomized time slices), but it wouldn't work on real
 * hardware. But even though Nachos can't always detect when your program
 * would fail in real life, you should still write properly synchronized code.
 */
public final class Interrupt {
    /**
     * Allocate a new interrupt controller.
     *
     * @param   privilege       encapsulates privileged access to the Nachos
     *                          machine.
     */
    public Interrupt(Privilege privilege) {
        System.out.print(" interrupt");

        this.privilege = privilege;
        privilege.interrupt = new InterruptPrivilege();

        enabled = false;
        pending = new TreeSet<PendingInterrupt>();
    }

    /**
     * Enable interrupts. This method has the same effect as
     * <tt>setStatus(true)</tt>.
     */
    public void enable() {
        setStatus(true);
    }

    /**
     * Disable interrupts and return the old interrupt state. This method has
     * the same effect as <tt>setStatus(false)</tt>.
     *
     * @return  <tt>true</tt> if interrupts were enabled.
     */
    public boolean disable() {
        return setStatus(false);
    }

    /**
     * Restore interrupts to the specified status. This method has the same
     * effect as <tt>setStatus(<i>status</i>)</tt>.
     *
     * @param   status  <tt>true</tt> to enable interrupts.
     */
    public void restore(boolean status) {
        setStatus(status);
    }

    /**
     * Set the interrupt status to be enabled (<tt>true</tt>) or disabled
     * (<tt>false</tt>) and return the previous status. If the interrupt
     * status changes from disabled to enabled, the simulated time is advanced.
     *
     * @param   status          <tt>true</tt> to enable interrupts.
     * @return                  <tt>true</tt> if interrupts were enabled.
     */
    public boolean setStatus(boolean status) {
        boolean oldStatus = enabled;
        enabled = status;

        if (oldStatus == false && status == true)
            tick(true);

        return oldStatus;
    }

    /**
     * Tests whether interrupts are enabled.
     *
     * @return  <tt>true</tt> if interrupts are enabled.
     */
    public boolean enabled() {
        return enabled;
    }

    /**
     * Tests whether interrupts are disabled.
     *
     * @return <tt>true</tt> if interrupts are disabled.
     */
    public boolean disabled() {
        return !enabled;
    }

    private void schedule(long when, String type, Runnable handler) {
        Lib.assertTrue(when>0);
```

```
    long time = privilege.stats.totalTicks + when;
    PendingInterrupt toOccur = new PendingInterrupt(time, type, handler);

    Lib.debug(dbgInt,
              "Scheduling the " + type +
              " interrupt handler at time = " + time);

    pending.add(toOccur);
}

private void tick(boolean inKernelMode) {
    Stats stats = privilege.stats;

    if (inKernelMode) {
        stats.kernelTicks += Stats.KernelTick;
        stats.totalTicks += Stats.KernelTick;
    }
    else {
        stats.userTicks += Stats.UserTick;
        stats.totalTicks += Stats.UserTick;
    }

    if (Lib.test(dbgInt))
        System.out.println("== Tick " + stats.totalTicks + " ==");

    enabled = false;
    checkIfDue();
    enabled = true;
}

private void checkIfDue() {
    long time = privilege.stats.totalTicks;

    Lib.assertTrue(disabled());

    if (Lib.test(dbgInt))
        print();

    if (pending.isEmpty())
        return;

    if (((PendingInterrupt) pending.first()).time > time)
        return;

    Lib.debug(dbgInt, "Invoking interrupt handlers at time = " + time);

    while (!pending.isEmpty() &&
           ((PendingInterrupt) pending.first()).time <= time) {
        PendingInterrupt next = (PendingInterrupt) pending.first();
        pending.remove(next);

        Lib.assertTrue(next.time <= time);

        if (privilege.processor != null)
            privilege.processor.flushPipe();

        Lib.debug(dbgInt, "  " + next.type);

        next.handler.run();
    }

    Lib.debug(dbgInt, "  (end of list)");
}

private void print() {
    System.out.println("Time: " + privilege.stats.totalTicks
                       + ", interrupts " + (enabled ? "on" : "off"));
    System.out.println("Pending interrupts:");

    for (Iterator i=pending.iterator(); i.hasNext(); ) {
        PendingInterrupt toOccur = (PendingInterrupt) i.next();
        System.out.println("  " + toOccur.type +
                           ", scheduled at " + toOccur.time);
    }

    System.out.println("  (end of list)");
}

private class PendingInterrupt implements Comparable {
    PendingInterrupt(long time, String type, Runnable handler) {
        this.time = time;
        this.type = type;
        this.handler = handler;
        this.id = numPendingInterruptsCreated++;
    }

    public int compareTo(Object o) {
        PendingInterrupt toOccur = (PendingInterrupt) o;

        // can't return 0 for unequal objects, so check all fields
        if (time < toOccur.time)
            return -1;
        else if (time > toOccur.time)
            return 1;
        else if (id < toOccur.id)
            return -1;
        else if (id > toOccur.id)
            return 1;
        else
            return 0;
    }

    long time;
    String type;
    Runnable handler;

    private long id;
}

private long numPendingInterruptsCreated = 0;

private Privilege privilege;

private boolean enabled;
private TreeSet<PendingInterrupt> pending;

private static final char dbgInt = 'i';

private class InterruptPrivilege implements Privilege.InterruptPrivilege {
    public void schedule(long when, String type, Runnable handler) {
        Interrupt.this.schedule(when, type, handler);
    }
```

```
        public void tick(boolean inKernelMode) {
            Interrupt.this.tick(inKernelMode);
        }
    }
}
```

```
// PART OF THE MACHINE SIMULATION. DO NOT CHANGE.

package nachos.machine;

/**
 * An OS kernel.
 */
public abstract class Kernel {
    /** Globally accessible reference to the kernel. */
    public static Kernel kernel = null;

    /**
     * Allocate a new kernel.
     */
    public Kernel() {
        // make sure only one kernel is created
        Lib.assertTrue(kernel == null);
        kernel = this;
    }

    /**
     * Initialize this kernel.
     */
    public abstract void initialize(String[] args);

    /**
     * Test that this module works.
     *
     * <b>Warning:</b> this method will not be invoked by the autograder when
     * we grade your projects. You should perform all initialization in
     * <tt>initialize()</tt>.
     */
    public abstract void selfTest();

    /**
     * Begin executing user programs, if applicable.
     */
    public abstract void run();

    /**
     * Terminate this kernel. Never returns.
     */
    public abstract void terminate();
}
```

```
// PART OF THE MACHINE SIMULATION. DO NOT CHANGE.

package nachos.machine;

import java.lang.reflect.Constructor;
import java.lang.reflect.Method;
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;
import java.security.PrivilegedAction;
import java.util.Random;

/**
 * Thrown when an assertion fails.
 */
class AssertionFailureError extends Error {
    AssertionFailureError() {
        super();
    }

    AssertionFailureError(String message) {
        super(message);
    }
}

/**
 * Provides miscellaneous library routines.
 */
public final class Lib {
    /**
     * Prevent instantiation.
     */
    private Lib() {
    }

    private static Random random = null;

    /**
     * Seed the random number generater. May only be called once.
     *
     * @param   randomSeed      the seed for the random number generator.
     */
    public static void seedRandom(long randomSeed) {
        assertTrue(random == null);
        random = new Random(randomSeed);
    }

    /**
     * Return a random integer between 0 and <i>range - 1</i>. Must not be
     * called before <tt>seedRandom()</tt> seeds the random number generator.
     *
     * @param   range   a positive value specifying the number of possible
     *                  return values.
     * @return  a random integer in the specified range.
     */
    public static int random(int range) {
        assertTrue(range > 0);
        return random.nextInt(range);
    }

    /**
     * Return a random double between 0.0 (inclusive) and 1.0 (exclusive).
     *
     * @return  a random double between 0.0 and 1.0.
     */
    public static double random() {
        return random.nextDouble();
    }

    /**
     * Asserts that <i>expression</i> is <tt>true</tt>. If not, then Nachos
     * exits with an error message.
     *
     * @param   expression      the expression to assert.
     */
    public static void assertTrue(boolean expression) {
        if (!expression)
            throw new AssertionFailureError();
    }

    /**
     * Asserts that <i>expression</i> is <tt>true</tt>. If not, then Nachos
     * exits with the specified error message.
     *
     * @param   expression      the expression to assert.
     * @param   message         the error message.
     */
    public static void assertTrue(boolean expression, String message) {
        if (!expression)
            throw new AssertionFailureError(message);
    }

    /**
     * Asserts that this call is never made. Same as <tt>assertTrue(false)</tt>.
     */
    public static void assertNotReached() {
        assertTrue(false);
    }

    /**
     * Asserts that this call is never made, with the specified error messsage.
     * Same as <tt>assertTrue(false, message)</tt>.
     *
     * @param   message the error message.
     */
    public static void assertNotReached(String message) {
        assertTrue(false, message);
    }

    /**
     * Print <i>message</i> if <i>flag</i> was enabled on the command line. To
     * specify which flags to enable, use the -d command line option. For
     * example, to enable flags a, c, and e, do the following:
     *
     * <p>
     * <pre>nachos -d ace</pre>
     *
     * <p>
     * Nachos uses several debugging flags already, but you are encouraged to
     * add your own.
     *
     * @param   flag    the debug flag that must be set to print this message.
     * @param   message the debug message.
     */
    public static void debug(char flag, String message) {
```

```
    if (test(flag))
        System.out.println(message);
}

/**
 * Tests if <i>flag</i> was enabled on the command line.
 *
 * @param   flag    the debug flag to test.
 *
 * @return  <tt>true</tt> if this flag was enabled on the command line.
 */
public static boolean test(char flag) {
    if (debugFlags == null)
        return false;
    else if (debugFlags[(int) '+'])
        return true;
    else if (flag >= 0 && flag < 0x80 && debugFlags[(int) flag])
        return true;
    else
        return false;
}

/**
 * Enable all the debug flags in <i>flagsString</i>.
 *
 * @param   flagsString     the flags to enable.
 */
public static void enableDebugFlags(String flagsString) {
    if (debugFlags == null)
        debugFlags = new boolean[0x80];

    char[] newFlags = flagsString.toCharArray();
    for (int i=0; i<newFlags.length; i++) {
        char c = newFlags[i];
        if (c >= 0 && c < 0x80)
            debugFlags[(int) c] = true;
    }
}

/** Debug flags specified on the command line. */
private static boolean debugFlags[];

/**
 * Read a file, verifying that the requested number of bytes is read, and
 * verifying that the read operation took a non-zero amount of time.
 *
 * @param   file            the file to read.
 * @param   position        the file offset at which to start reading.
 * @param   buf             the buffer in which to store the data.
 * @param   offset          the buffer offset at which storing begins.
 * @param   length          the number of bytes to read.
 */
public static void strictReadFile(OpenFile file, int position,
                                  byte[] buf, int offset, int length) {
    long startTime = Machine.timer().getTime();
    assertTrue(file.read(position, buf, offset, length) == length);
    long finishTime = Machine.timer().getTime();
    assertTrue(finishTime>startTime);
}

/**
 * Load an entire file into memory.
 *
 * @param   file    the file to load.
 * @return  an array containing the contents of the entire file, or
 *          <tt>null</tt> if an error occurred.
 */
public static byte[] loadFile(OpenFile file) {
    int startOffset = file.tell();

    int length = file.length();
    if (length < 0)
        return null;

    byte[] data = new byte[length];

    file.seek(0);
    int amount = file.read(data, 0, length);
    file.seek(startOffset);

    if (amount == length)
        return data;
    else
        return null;
}

/**
 * Take a read-only snapshot of a file.
 *
 * @param   file    the file to take a snapshot of.
 * @return  a read-only snapshot of the file.
 */
public static OpenFile cloneFile(OpenFile file) {
    OpenFile clone = new ArrayFile(loadFile(file));

    clone.seek(file.tell());

    return clone;
}

/**
 * Convert a short into its little-endian byte string representation.
 *
 * @param   array   the array in which to store the byte string.
 * @param   offset  the offset in the array where the string will start.
 * @param   value   the value to convert.
 */
public static void bytesFromShort(byte[] array, int offset, short value) {
    array[offset+0] = (byte) ((value>>0)&0xFF);
    array[offset+1] = (byte) ((value>>8)&0xFF);
}

/**
 * Convert an int into its little-endian byte string representation.
 *
 * @param   array   the array in which to store the byte string.
 * @param   offset  the offset in the array where the string will start.
 * @param   value   the value to convert.
 */
public static void bytesFromInt(byte[] array, int offset, int value) {
    array[offset+0] = (byte) ((value>>0) &0xFF);
    array[offset+1] = (byte) ((value>>8) &0xFF);
    array[offset+2] = (byte) ((value>>16)&0xFF);
    array[offset+3] = (byte) ((value>>24)&0xFF);
```

```
}

/**
 * Convert an int into its little-endian byte string representation, and
 * return an array containing it.
 *
 * @param   value   the value to convert.
 * @return  an array containing the byte string.
 */
public static byte[] bytesFromInt(int value) {
    byte[] array = new byte[4];
    bytesFromInt(array, 0, value);
    return array;
}

/**
 * Convert an int into a little-endian byte string representation of the
 * specified length.
 *
 * @param   array   the array in which to store the byte string.
 * @param   offset  the offset in the array where the string will start.
 * @param   length  the number of bytes to store (must be 1, 2, or 4).
 * @param   value   the value to convert.
 */
public static void bytesFromInt(byte[] array, int offset,
                                int length, int value) {
    assertTrue(length==1 || length==2 || length==4);

    switch (length) {
    case 1:
        array[offset] = (byte) value;
        break;
    case 2:
        bytesFromShort(array, offset, (short) value);
        break;
    case 4:
        bytesFromInt(array, offset, value);
        break;
    }
}

/**
 * Convert to a short from its little-endian byte string representation.
 *
 * @param   array   the array containing the byte string.
 * @param   offset  the offset of the byte string in the array.
 * @return  the corresponding short value.
 */
public static short bytesToShort(byte[] array, int offset) {
    return (short) ((((short) array[offset+0] & 0xFF) << 0) |
                    (((short) array[offset+1] & 0xFF) << 8));
}

/**
 * Convert to an unsigned short from its little-endian byte string
 * representation.
 *
 * @param   array   the array containing the byte string.
 * @param   offset  the offset of the byte string in the array.
 * @return  the corresponding short value.
 */
public static int bytesToUnsignedShort(byte[] array, int offset) {
    return (((int) bytesToShort(array, offset)) & 0xFFFF);
}

/**
 * Convert to an int from its little-endian byte string representation.
 *
 * @param   array   the array containing the byte string.
 * @param   offset  the offset of the byte string in the array.
 * @return  the corresponding int value.
 */
public static int bytesToInt(byte[] array, int offset) {
    return (int) ((((int) array[offset+0] & 0xFF) << 0)  |
                  (((int) array[offset+1] & 0xFF) << 8)  |
                  (((int) array[offset+2] & 0xFF) << 16) |
                  (((int) array[offset+3] & 0xFF) << 24));
}

/**
 * Convert to an int from a little-endian byte string representation of the
 * specified length.
 *
 * @param   array   the array containing the byte string.
 * @param   offset  the offset of the byte string in the array.
 * @param   length  the length of the byte string.
 * @return  the corresponding value.
 */
public static int bytesToInt(byte[] array, int offset, int length) {
    assertTrue(length==1 || length==2 || length==4);

    switch (length) {
    case 1:
        return array[offset];
    case 2:
        return bytesToShort(array, offset);
    case 4:
        return bytesToInt(array, offset);
    default:
        return -1;
    }
}

/**
 * Convert to a string from a possibly null-terminated array of bytes.
 *
 * @param   array   the array containing the byte string.
 * @param   offset  the offset of the byte string in the array.
 * @param   length  the maximum length of the byte string.
 * @return  a string containing the specified bytes, up to and not
 *          including the null-terminator (if present).
 */
public static String bytesToString(byte[] array, int offset, int length) {
    int i;
    for (i=0; i<length; i++) {
        if (array[offset+i] == 0)
            break;
    }

    return new String(array, offset, i);
}

/** Mask out and shift a bit substring.
 *
```

```
     * @param   bits    the bit string.
     * @param   lowest  the first bit of the substring within the string.
     * @param   size    the number of bits in the substring.
     * @return  the substring.
     */
    public static int extract(int bits, int lowest, int size) {
        if (size == 32)
            return (bits >> lowest);
        else
            return ((bits >> lowest) & ((1<<size)-1));
    }

    /** Mask out and shift a bit substring.
     *
     * @param   bits    the bit string.
     * @param   lowest  the first bit of the substring within the string.
     * @param   size    the number of bits in the substring.
     * @return  the substring.
     */
    public static long extract(long bits, int lowest, int size) {
        if (size == 64)
            return (bits >> lowest);
        else
            return ((bits >> lowest) & ((1L<<size)-1));
    }

    /** Mask out and shift a bit substring; then sign extend the substring.
     *
     * @param   bits    the bit string.
     * @param   lowest  the first bit of the substring within the string.
     * @param   size    the number of bits in the substring.
     * @return  the substring, sign-extended.
     */
    public static int extend(int bits, int lowest, int size) {
        int extra = 32 - (lowest+size);
        return ((extract(bits, lowest, size) << extra) >> extra);
    }

    /** Test if a bit is set in a bit string.
     *
     * @param   flag    the flag to test.
     * @param   bits    the bit string.
     * @return  <tt>true</tt> if <tt>(bits & flag)</tt> is non-zero.
     */
    public static boolean test(long flag, long bits) {
        return ((bits & flag) != 0);
    }

    /**
     * Creates a padded upper-case string representation of the integer
     * argument in base 16.
     *
     * @param   i       an integer.
     * @return  a padded upper-case string representation in base 16.
     */
    public static String toHexString(int i) {
        return toHexString(i, 8);
    }

    /**
     * Creates a padded upper-case string representation of the integer
     * argument in base 16, padding to at most the specified number of digits.
     *
     * @param   i       an integer.
     * @param   pad     the minimum number of hex digits to pad to.
     * @return  a padded upper-case string representation in base 16.
     */
    public static String toHexString(int i, int pad) {
        String result = Integer.toHexString(i).toUpperCase();
        while (result.length() < pad)
            result = "0" + result;
        return result;
    }

    /**
     * Divide two non-negative integers, round the quotient up to the nearest
     * integer, and return it.
     *
     * @param   a       the numerator.
     * @param   b       the denominator.
     * @return  <tt>ceiling(a / b)</tt>.
     */
    public static int divRoundUp(int a, int b) {
        assertTrue(a >= 0 && b > 0);

        return ((a + (b-1)) / b);
    }

    /**
     * Load and return the named class, or return <tt>null</tt> if the class
     * could not be loaded.
     *
     * @param   className       the name of the class to load.
     * @return  the loaded class, or <tt>null</tt> if an error occurred.
     */
    public static Class tryLoadClass(String className) {
        try {
            return ClassLoader.getSystemClassLoader().loadClass(className);
        }
        catch (Throwable e) {
            return null;
        }
    }

    /**
     * Load and return the named class, terminating Nachos on any error.
     *
     * @param   className       the name of the class to load.
     * @return  the loaded class.
     */
    public static Class loadClass(String className) {
        try {
            return ClassLoader.getSystemClassLoader().loadClass(className);
        }
        catch (Throwable e) {
            Machine.terminate(e);
            return null;
        }
    }

    /**
     * Create and return a new instance of the named class, using the
     * constructor that takes no arguments.
     *
```

```
     * @param   className       the name of the class to instantiate.
     * @return  a new instance of the class.
     */
    public static Object constructObject(String className) {
        try {
            // kamil - workaround for Java 1.4
            // Thanks to Ka-Hing Cheung for the suggestion.
            // Fixed for Java 1.5 by geels
            Class[] param_types = new Class[0];
            Object[] params = new Object[0];
            return loadClass(className).getConstructor(param_types).newInstance(params
);
        }
        catch (Throwable e) {
            Machine.terminate(e);
            return null;
        }
    }

    /**
     * Verify that the specified class extends or implements the specified
     * superclass.
     *
     * @param   cls     the descendant class.
     * @param   superCls        the ancestor class.
     */
    public static void checkDerivation(Class<?> cls, Class<?> superCls) {
        Lib.assertTrue(superCls.isAssignableFrom(cls));
    }

    /**
     * Verifies that the specified class is public and not abstract, and that a
     * constructor with the specified signature exists and is public.
     *
     * @param   cls     the class containing the constructor.
     * @param   parameterTypes  the list of parameters.
     */
    public static void checkConstructor(Class cls, Class[] parameterTypes) {
        try {
            Lib.assertTrue(Modifier.isPublic(cls.getModifiers()) &&
                           !Modifier.isAbstract(cls.getModifiers()));
            Constructor constructor = cls.getConstructor(parameterTypes);
            Lib.assertTrue(Modifier.isPublic(constructor.getModifiers()));
        }
        catch (Exception e) {
            Lib.assertNotReached();
        }
    }

    /**
     * Verifies that the specified class is public, and that a non-static
     * method with the specified name and signature exists, is public, and
     * returns the specified type.
     *
     * @param   cls     the class containing the non-static method.
     * @param   methodName      the name of the non-static method.
     * @param   parameterTypes  the list of parameters.
     * @param   returnType      the required return type.
     */
    public static void checkMethod(Class cls, String methodName,
                                   Class[] parameterTypes, Class returnType) {
        try {
            Lib.assertTrue(Modifier.isPublic(cls.getModifiers()));
            Method method = cls.getMethod(methodName, parameterTypes);
            Lib.assertTrue(Modifier.isPublic(method.getModifiers()) &&
                           !Modifier.isStatic(method.getModifiers()));
            Lib.assertTrue(method.getReturnType() == returnType);
        }
        catch (Exception e) {
            Lib.assertNotReached();
        }
    }

    /**
     * Verifies that the specified class is public, and that a static method
     * with the specified name and signature exists, is public, and returns the
     * specified type.
     *
     * @param   cls     the class containing the static method.
     * @param   methodName      the name of the static method.
     * @param   parameterTypes  the list of parameters.
     * @param   returnType      the required return type.
     */
    public static void checkStaticMethod(Class cls, String methodName,
                                         Class[] parameterTypes,
                                         Class returnType) {
        try {
            Lib.assertTrue(Modifier.isPublic(cls.getModifiers()));
            Method method = cls.getMethod(methodName, parameterTypes);
            Lib.assertTrue(Modifier.isPublic(method.getModifiers()) &&
                           Modifier.isStatic(method.getModifiers()));
            Lib.assertTrue(method.getReturnType() == returnType);
        }
        catch (Exception e) {
            Lib.assertNotReached();
        }
    }

    /**
     * Verifies that the specified class is public, and that a non-static field
     * with the specified name and type exists, is public, and is not final.
     *
     * @param   cls     the class containing the field.
     * @param   fieldName       the name of the field.
     * @param   fieldType       the required type.
     */
    public static void checkField(Class cls, String fieldName,
                                  Class fieldType) {
        try {
            Lib.assertTrue(Modifier.isPublic(cls.getModifiers()));
            Field field = cls.getField(fieldName);
            Lib.assertTrue(field.getType() == fieldType);
            Lib.assertTrue(Modifier.isPublic(field.getModifiers()) &&
                           !Modifier.isStatic(field.getModifiers()) &&
                           !Modifier.isFinal(field.getModifiers()));
        }
        catch (Exception e) {
            Lib.assertNotReached();
        }
    }

    /**
     * Verifies that the specified class is public, and that a static field
     * with the specified name and type exists and is public.
```

```
     *
     * @param   cls     the class containing the static field.
     * @param   fieldName       the name of the static field.
     * @param   fieldType       the required type.
     */
    public static void checkStaticField(Class cls, String fieldName,
                                        Class fieldType) {
        try {
            Lib.assertTrue(Modifier.isPublic(cls.getModifiers()));
            Field field = cls.getField(fieldName);
            Lib.assertTrue(field.getType() == fieldType);
            Lib.assertTrue(Modifier.isPublic(field.getModifiers()) &&
                       Modifier.isStatic(field.getModifiers()));
        }
        catch (Exception e) {
            Lib.assertNotReached();
        }
    }
}
```

```
// PART OF THE MACHINE SIMULATION. DO NOT CHANGE.

package nachos.machine;

import nachos.security.*;
import nachos.ag.*;

import java.io.File;

/**
 * The master class of the simulated machine. Processes command line arguments,
 * constructs all simulated hardware devices, and starts the grader.
 */
public final class Machine {
    /**
     * Nachos main entry point.
     *
     * @param   args    the command line arguments.
     */
    public static void main(final String[] args) {
        System.out.print("nachos 5.0j initializing...");

        Lib.assertTrue(Machine.args == null);
        Machine.args = args;

        processArgs();

        Config.load(configFileName);

        // get the current directory (.)
        baseDirectory = new File(new File("").getAbsolutePath());
        // get the nachos directory (./nachos)
        nachosDirectory = new File(baseDirectory, "nachos");

        String testDirectoryName =
            Config.getString("FileSystem.testDirectory");

        // get the test directory
        if (testDirectoryName != null) {
            testDirectory = new File(testDirectoryName);
        }
        else {
            // use ../test
            testDirectory = new File(baseDirectory.getParentFile(), "test");
        }

        securityManager = new NachosSecurityManager(testDirectory);
        privilege = securityManager.getPrivilege();

        privilege.machine = new MachinePrivilege();

        TCB.givePrivilege(privilege);
        privilege.stats = stats;

        securityManager.enable();
        createDevices();
        checkUserClasses();

        autoGrader = (AutoGrader) Lib.constructObject(autoGraderClassName);

        new TCB().start(new Runnable() {
            public void run() { autoGrader.start(privilege); }
        });
    }

    /**
     * Yield to non-Nachos threads. Use in non-preemptive JVM's to give
     * non-Nachos threads a chance to run.
     */
    public static void yield() {
        Thread.yield();
    }

    /**
     * Terminate Nachos. Same as <tt>TCB.die()</tt>.
     */
    public static void terminate() {
        TCB.die();
    }

    /**
     * Terminate Nachos as the result of an unhandled exception or error.
     *
     * @param   e       the exception or error.
     */
    public static void terminate(Throwable e) {
        if (e instanceof ThreadDeath)
            throw (ThreadDeath) e;

        e.printStackTrace();
        terminate();
    }

    /**
     * Print stats, and terminate Nachos.
     */
    public static void halt() {
        System.out.print("Machine halting!\n\n");
        stats.print();
        terminate();
    }

    /**
     * Return an array containing all command line arguments.
     *
     * @return  the command line arguments passed to Nachos.
     */
    public static String[] getCommandLineArguments() {
        String[] result = new String[args.length];

        System.arraycopy(args, 0, result, 0, args.length);

        return result;
    }

    private static void processArgs() {
        for (int i=0; i<args.length; ) {
            String arg = args[i++];
            if (arg.length() > 0 && arg.charAt(0) == '-') {
                if (arg.equals("-d")) {
                    Lib.assertTrue(i < args.length, "switch without argument");
                    Lib.enableDebugFlags(args[i++]);
                }
                else if (arg.equals("-h")) {
```

```
                System.out.print(help);
                System.exit(1);
            }
            else if (arg.equals("-m")) {
                Lib.assertTrue(i < args.length, "switch without argument");
                try {
                    numPhysPages = Integer.parseInt(args[i++]);
                }
                catch (NumberFormatException e) {
                    Lib.assertNotReached("bad value for -m switch");
                }
            }
            else if (arg.equals("-s")) {
                Lib.assertTrue(i < args.length, "switch without argument");
                try {
                    randomSeed = Long.parseLong(args[i++]);
                }
                catch (NumberFormatException e) {
                    Lib.assertNotReached("bad value for -s switch");
                }
            }
            else if (arg.equals("-x")) {
                Lib.assertTrue(i < args.length, "switch without argument");
                shellProgramName = args[i++];
            }
            else if (arg.equals("-z")) {
                System.out.print(copyright);
                System.exit(1);
            }
            // these switches are reserved for the autograder
            else if (arg.equals("-[]")) {
                Lib.assertTrue(i < args.length, "switch without argument");
                configFileName = args[i++];
            }
            else if (arg.equals("--")) {
                Lib.assertTrue(i < args.length, "switch without argument");
                autoGraderClassName = args[i++];
            }
        }
    }

    Lib.seedRandom(randomSeed);
}

private static void createDevices() {
    interrupt = new Interrupt(privilege);
    timer = new Timer(privilege);

    if (Config.getBoolean("Machine.bank"))
        bank = new ElevatorBank(privilege);

    if (Config.getBoolean("Machine.processor")) {
        if (numPhysPages == -1)
            numPhysPages = Config.getInteger("Processor.numPhysPages");
        processor = new Processor(privilege, numPhysPages);
    }

    if (Config.getBoolean("Machine.console"))
        console = new StandardConsole(privilege);

    if (Config.getBoolean("Machine.stubFileSystem"))
        stubFileSystem = new StubFileSystem(privilege, testDirectory);

    if (Config.getBoolean("Machine.networkLink"))
        networkLink = new NetworkLink(privilege);
}

private static void checkUserClasses() {
    System.out.print(" user-check");

    Class aclsInt = (new int[0]).getClass();
    Class clsObject = Lib.loadClass("java.lang.Object");
    Class clsRunnable = Lib.loadClass("java.lang.Runnable");
    Class clsString = Lib.loadClass("java.lang.String");

    Class clsKernel = Lib.loadClass("nachos.machine.Kernel");
    Class clsFileSystem = Lib.loadClass("nachos.machine.FileSystem");
    Class clsRiderControls = Lib.loadClass("nachos.machine.RiderControls");
    Class clsElevatorControls =
        Lib.loadClass("nachos.machine.ElevatorControls");
    Class clsRiderInterface =
        Lib.loadClass("nachos.machine.RiderInterface");
    Class clsElevatorControllerInterface =
        Lib.loadClass("nachos.machine.ElevatorControllerInterface");

    Class clsAlarm = Lib.loadClass("nachos.threads.Alarm");
    Class clsThreadedKernel =
        Lib.loadClass("nachos.threads.ThreadedKernel");
    Class clsKThread = Lib.loadClass("nachos.threads.KThread");
    Class clsCommunicator = Lib.loadClass("nachos.threads.Communicator");
    Class clsSemaphore = Lib.loadClass("nachos.threads.Semaphore");
    Class clsLock = Lib.loadClass("nachos.threads.Lock");
    Class clsCondition = Lib.loadClass("nachos.threads.Condition");
    Class clsCondition2 = Lib.loadClass("nachos.threads.Condition2");
    Class clsRider = Lib.loadClass("nachos.threads.Rider");
    Class clsElevatorController =
        Lib.loadClass("nachos.threads.ElevatorController");

    Lib.checkDerivation(clsThreadedKernel, clsKernel);

    Lib.checkStaticField(clsThreadedKernel, "alarm", clsAlarm);
    Lib.checkStaticField(clsThreadedKernel, "fileSystem", clsFileSystem);

    Lib.checkMethod(clsAlarm, "waitUntil", new Class[] { long.class },
                    void.class);

    Lib.checkConstructor(clsKThread, new Class[] { });
    Lib.checkConstructor(clsKThread, new Class[] { clsRunnable });

    Lib.checkStaticMethod(clsKThread, "currentThread", new Class[] {},
                          clsKThread);
    Lib.checkStaticMethod(clsKThread, "finish", new Class[] {},
                          void.class);
    Lib.checkStaticMethod(clsKThread, "yield", new Class[] {}, void.class);
    Lib.checkStaticMethod(clsKThread, "sleep", new Class[] {}, void.class);

    Lib.checkMethod(clsKThread, "setTarget", new Class[]{ clsRunnable },
                    clsKThread);
    Lib.checkMethod(clsKThread, "setName", new Class[] { clsString },
                    clsKThread);
    Lib.checkMethod(clsKThread, "getName", new Class[] { }, clsString);
    Lib.checkMethod(clsKThread, "fork", new Class[] { }, void.class);
    Lib.checkMethod(clsKThread, "ready", new Class[] { }, void.class);
    Lib.checkMethod(clsKThread, "join", new Class[] { }, void.class);
```

```
    Lib.checkField(clsKThread, "schedulingState", clsObject);

    Lib.checkConstructor(clsCommunicator, new Class[] {});
    Lib.checkMethod(clsCommunicator, "speak", new Class[] { int.class },
                    void.class);
    Lib.checkMethod(clsCommunicator, "listen", new Class[] { }, int.class);

    Lib.checkConstructor(clsSemaphore, new Class[] { int.class });
    Lib.checkMethod(clsSemaphore, "P", new Class[] { }, void.class);
    Lib.checkMethod(clsSemaphore, "V", new Class[] { }, void.class);

    Lib.checkConstructor(clsLock, new Class[] { });
    Lib.checkMethod(clsLock, "acquire", new Class[] { }, void.class);
    Lib.checkMethod(clsLock, "release", new Class[] { }, void.class);
    Lib.checkMethod(clsLock, "isHeldByCurrentThread", new Class[]{ },
                    boolean.class);

    Lib.checkConstructor(clsCondition, new Class[] { clsLock });
    Lib.checkConstructor(clsCondition2, new Class[] { clsLock });

    Lib.checkMethod(clsCondition, "sleep", new Class[] { }, void.class);
    Lib.checkMethod(clsCondition, "wake", new Class[] { }, void.class);
    Lib.checkMethod(clsCondition, "wakeAll", new Class[] { }, void.class);
    Lib.checkMethod(clsCondition2, "sleep", new Class[] { }, void.class);
    Lib.checkMethod(clsCondition2, "wake", new Class[] { }, void.class);
    Lib.checkMethod(clsCondition2, "wakeAll", new Class[] { }, void.class);

    Lib.checkDerivation(clsRider, clsRiderInterface);

    Lib.checkConstructor(clsRider, new Class[] { });
    Lib.checkMethod(clsRider, "initialize",
                    new Class[] { clsRiderControls, aclsInt }, void.class);

    Lib.checkDerivation(clsElevatorController,
                        clsElevatorControllerInterface);

    Lib.checkConstructor(clsElevatorController, new Class[] { });
    Lib.checkMethod(clsElevatorController, "initialize",
                    new Class[] { clsElevatorControls }, void.class);
}

/**
 * Prevent instantiation.
 */
private Machine() {
}

/**
 * Return the hardware interrupt manager.
 *
 * @return  the hardware interrupt manager.
 */
public static Interrupt interrupt() { return interrupt; }

/**
 * Return the hardware timer.
 *
 * @return  the hardware timer.
 */
public static Timer timer() { return timer; }

/**
 * Return the hardware elevator bank.
 *
 * @return  the hardware elevator bank, or <tt>null</tt> if it is not
 *          present.
 */
public static ElevatorBank bank() { return bank; }

/**
 * Return the MIPS processor.
 *
 * @return  the MIPS processor, or <tt>null</tt> if it is not present.
 */
public static Processor processor() { return processor; }

/**
 * Return the hardware console.
 *
 * @return  the hardware console, or <tt>null</tt> if it is not present.
 */
public static SerialConsole console() { return console; }

/**
 * Return the stub filesystem.
 *
 * @return  the stub file system, or <tt>null</tt> if it is not present.
 */
public static FileSystem stubFileSystem() { return stubFileSystem; }

/**
 * Return the network link.
 *
 * @return  the network link,  or <tt>null</tt> if it is not present.
 */
public static NetworkLink networkLink() { return networkLink; }

/**
 * Return the autograder.
 *
 * @return  the autograder.
 */
public static AutoGrader autoGrader() { return autoGrader; }

private static Interrupt interrupt = null;
private static Timer timer = null;
private static ElevatorBank bank = null;
private static Processor processor = null;
private static SerialConsole console = null;
private static FileSystem stubFileSystem = null;
private static NetworkLink networkLink = null;
private static AutoGrader autoGrader = null;

private static String autoGraderClassName = "nachos.ag.AutoGrader";

/**
 * Return the name of the shell program that a user-programming kernel
 * must run. Make sure <tt>UserKernel.run()</tt> <i>always</i> uses this
 * method to decide which program to run.
 *
 * @return  the name of the shell program to run.
 */
public static String getShellProgramName() {
```

```
    if (shellProgramName == null)
        shellProgramName = Config.getString("Kernel.shellProgram");

    Lib.assertTrue(shellProgramName != null);
    return shellProgramName;
}

private static String shellProgramName = null;

/**
 * Return the name of the process class that the kernel should use. In
 * the multi-programming project, returns
 * <tt>nachos.userprog.UserProcess</tt>. In the VM project, returns
 * <tt>nachos.vm.VMProcess</tt>. In the networking project, returns
 * <tt>nachos.network.NetProcess</tt>.
 *
 * @return  the name of the process class that the kernel should use.
 *
 * @see     nachos.userprog.UserKernel#run
 * @see     nachos.userprog.UserProcess
 * @see     nachos.vm.VMProcess
 * @see     nachos.network.NetProcess
 */
public static String getProcessClassName() {
    if (processClassName == null)
        processClassName = Config.getString("Kernel.processClassName");

    Lib.assertTrue(processClassName != null);
    return processClassName;
}

private static String processClassName = null;

private static NachosSecurityManager securityManager;
private static Privilege privilege;

private static String[] args = null;

private static Stats stats = new Stats();

private static int numPhysPages = -1;
private static long randomSeed = 0;

private static File baseDirectory, nachosDirectory, testDirectory;
private static String configFileName = "nachos.conf";

private static final String help =
    "\n" +
    "Options:\n" +
    "\n" +
    "\t-d <debug flags>\n" +
    "\t\tEnable some debug flags, e.g. -d ti\n" +
    "\n" +
    "\t-h\n" +
    "\t\tPrint this help message.\n" +
    "\n" +
    "\t-m <pages>\n" +
    "\t\tSpecify how many physical pages of memory to simulate.\n" +
    "\n" +
    "\t-s <seed>\n" +
    "\t\tSpecify the seed for the random number generator (seed is a\n" +
    "\t\tlong).\n" +
        "\n" +
        "\t-x <program>\n" +
        "\t\tSpecify a program that UserKernel.run() should execute,\n" +
        "\t\tinstead of the value of the configuration variable\n" +
        "\t\tKernel.shellProgram\n" +
        "\n" +
        "\t-z\n" +
        "\t\tprint the copyright message\n" +
        "\n" +
        "\t-- <grader class>\n" +
        "\t\tSpecify an autograder class to use, instead of\n" +
        "\t\tnachos.ag.AutoGrader\n" +
        "\n" +
        "\t-# <grader arguments>\n" +
        "\t\tSpecify the argument string to pass to the autograder.\n" +
        "\n" +
        "\t-[] <config file>\n" +
        "\t\tSpecifiy a config file to use, instead of nachos.conf\n" +
        ""
        ;

    private static final String copyright = "\n"
        + "Copyright 1992-2001 The Regents of the University of California.\n"
        + "All rights reserved.\n"
        + "\n"
        + "Permission to use, copy, modify, and distribute this software and\n"
        + "its documentation for any purpose, without fee, and without\n"
        + "written agreement is hereby granted, provided that the above\n"
        + "copyright notice and the following two paragraphs appear in all\n"
        + "copies of this software.\n"
        + "\n"
        + "IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY\n"
        + "PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL\n"
        + "DAMAGES ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS\n"
        + "DOCUMENTATION, EVEN IF THE UNIVERSITY OF CALIFORNIA HAS BEEN\n"
        + "ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.\n"
        + "\n"
        + "THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY\n"
        + "WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES\n"
        + "OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.  THE\n"
        + "SOFTWARE PROVIDED HEREUNDER IS ON AN \"AS IS\" BASIS, AND THE\n"
        + "UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO PROVIDE\n"
        + "MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.\n"
        ;

    private static class MachinePrivilege
        implements Privilege.MachinePrivilege {
        public void setConsole(SerialConsole console) {
            Machine.console = console;
        }
    }

    // dummy variables to make javac smarter
    private static Coff dummy1 = null;
}
```

```
// PART OF THE MACHINE SIMULATION. DO NOT CHANGE.

package nachos.machine;

/**
 * Thrown when a malformed packet is processed.
 */
public class MalformedPacketException extends Exception {
    /**
     * Allocate a new <tt>MalformedPacketException</tt>.
     */
    public MalformedPacketException() {
    }
}
```

```
// PART OF THE MACHINE SIMULATION. DO NOT CHANGE.

package nachos.machine;

import nachos.security.*;

import java.io.IOException;
import java.net.DatagramSocket;
import java.net.DatagramPacket;
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.net.SocketException;

/**
 * A full-duplex network link. Provides ordered, unreliable delivery of
 * limited-size packets to other machines on the network. Packets are
 * guaranteed to be uncorrupted as well.
 *
 * <p>
 * Recall the general layering of network protocols:
 * <ul>
 * <li>Session/Transport
 * <li>Network
 * <li>Link
 * <li>Physical
 * </ul>
 *
 * <p>
 * The physical layer provides a bit stream interface to the link layer. This
 * layer is very hardware-dependent.
 *
 * <p>
 * The link layer uses the physical layer to provide a packet interface to the
 * network layer. The link layer generally provides unreliable delivery of
 * limited-size packets, but guarantees that packets will not arrive out of
 * order. Some links protect against packet corruption as well. The ethernet
 * protocol is an example of a link layer.
 *
 * <p>
 * The network layer exists to connect multiple networks together into an
 * internet. The network layer provides globally unique addresses. Routers
 * (a.k.a. gateways) move packets across networks at this layer. The network
 * layer provides unordered, unreliable delivery of limited-size uncorrupted
 * packets to any machine on the same internet. The most commonly used network
 * layer protocol is IP (Internet Protocol), which is used to connect the
 * Internet.
 *
 * <p>
 * The session/transport layer provides a byte-stream interface to the
 * application. This means that the transport layer must deliver uncorrupted
 * bytes to the application, in the same order they were sent. Byte-streams
 * must be connected and disconnected, and exist between ports, not machines.
 *
 * <p>
 * This class provides a link layer abstraction. Since we do not allow
 * different Nachos networks to communicate with one another, there is no need
 * for a network layer in Nachos. This should simplify your design for the
 * session/transport layer, since you can assume packets never arrive out of
 * order.
 */
public class NetworkLink {
    /**
     * Allocate a new network link.
     *
     * <p>
     * <tt>nachos.conf</tt> specifies the reliability of the network. The
     * reliability, between 0 and 1, is the probability that any particular
     * packet will not get dropped by the network.
     *
     * @param   privilege       encapsulates privileged access to the Nachos
     *                          machine.
     */
    public NetworkLink(Privilege privilege) {
        System.out.print(" network");

        this.privilege = privilege;

        try {
            localHost = InetAddress.getLocalHost();
        }
        catch (UnknownHostException e) {
            localHost = null;
        }

        Lib.assertTrue(localHost != null);

        reliability = Config.getDouble("NetworkLink.reliability");
        Lib.assertTrue(reliability > 0 && reliability <= 1.0);

        socket = null;

        for (linkAddress=0;linkAddress<Packet.linkAddressLimit;linkAddress++) {
            try {
                socket = new DatagramSocket(portBase + linkAddress, localHost);
                break;
            }
            catch (SocketException e) {
            }
        }

        if (socket == null) {
            System.out.println("");
            System.out.println("Unable to acquire a link address!");
            Lib.assertNotReached();
        }

        System.out.print("(" + linkAddress + ")");

        receiveInterrupt = new Runnable() {
                public void run() { receiveInterrupt(); }
            };

        sendInterrupt = new Runnable() {
                public void run() { sendInterrupt(); }
            };

        scheduleReceiveInterrupt();

        Thread receiveThread = new Thread(new Runnable() {
                public void run() { receiveLoop(); }
            });

        receiveThread.start();
    }
```

```
/**
 * Returns the address of this network link.
 *
 * @return  the address of this network link.
 */
public int getLinkAddress() {
    return linkAddress;
}

/**
 * Set this link's receive and send interrupt handlers.
 *
 * <p>
 * The receive interrupt handler is called every time a packet arrives
 * and can be read using <tt>receive()</tt>.
 *
 * <p>
 * The send interrupt handler is called every time a packet sent with
 * <tt>send()</tt> is finished being sent. This means that another
 * packet can be sent.
 *
 * @param   receiveInterruptHandler the callback to call when a packet
 *                                  arrives.
 * @param   sendInterruptHandler    the callback to call when another
 *                                  packet can be sent.
 */
public void setInterruptHandlers(Runnable receiveInterruptHandler,
                                 Runnable sendInterruptHandler) {
    this.receiveInterruptHandler = receiveInterruptHandler;
    this.sendInterruptHandler = sendInterruptHandler;
}

private void scheduleReceiveInterrupt() {
    privilege.interrupt.schedule(Stats.NetworkTime, "network recv",
                                 receiveInterrupt);
}

private synchronized void receiveInterrupt() {
    Lib.assertTrue(incomingPacket == null);

    if (incomingBytes != null) {
        if (Machine.autoGrader().canReceivePacket(privilege)) {
            try {
                incomingPacket = new Packet(incomingBytes);

                privilege.stats.numPacketsReceived++;
            }
            catch (MalformedPacketException e) {
            }
        }

        incomingBytes = null;
        notify();

        if (incomingPacket == null)
            scheduleReceiveInterrupt();
        else if (receiveInterruptHandler != null)
            receiveInterruptHandler.run();
    }
    else {
        scheduleReceiveInterrupt();
    }
}

/**
 * Return the next packet received.
 *
 * @return  the next packet received, or <tt>null</tt> if no packet is
 *          available.
 */
public Packet receive() {
    Packet p = incomingPacket;

    if (incomingPacket != null) {
        incomingPacket = null;
        scheduleReceiveInterrupt();
    }

    return p;
}

private void receiveLoop() {
    while (true) {
        synchronized(this) {
            while (incomingBytes != null) {
                try {
                    wait();
                }
                catch (InterruptedException e) {
                }
            }
        }

        byte[] packetBytes;

        try {
            byte[] buffer = new byte[Packet.maxPacketLength];

            DatagramPacket dp = new DatagramPacket(buffer, buffer.length);

            socket.receive(dp);

            packetBytes = new byte[dp.getLength()];

            System.arraycopy(buffer,0, packetBytes,0, packetBytes.length);
        }
        catch (IOException e) {
            return;
        }

        synchronized(this) {
            incomingBytes = packetBytes;
        }
    }
}

private void scheduleSendInterrupt() {
    privilege.interrupt.schedule(Stats.NetworkTime, "network send",
                                 sendInterrupt);
}

private void sendInterrupt() {
    Lib.assertTrue(outgoingPacket != null);
```

```
    // randomly drop packets, according to its reliability
    if (Machine.autoGrader().canSendPacket(privilege) &&
        Lib.random() <= reliability) {
        // ok, no drop
        privilege.doPrivileged(new Runnable() {
                public void run() { sendPacket(); }
            });
    }
    else {
        outgoingPacket = null;
    }

    if (sendInterruptHandler != null)
        sendInterruptHandler.run();
}

private void sendPacket() {
    Packet p = outgoingPacket;
    outgoingPacket = null;

    try {
        socket.send(new DatagramPacket(p.packetBytes, p.packetBytes.length,
                                       localHost, portBase+p.dstLink));

        privilege.stats.numPacketsSent++;
    }
    catch (IOException e) {
    }
}

/**
 * Send another packet. If a packet is already being sent, the result is
 * not defined.
 *
 * @param   pkt     the packet to send.
 */
public void send(Packet pkt) {
    if (outgoingPacket == null)
        scheduleSendInterrupt();

    outgoingPacket = pkt;
}

private static final int hash;
private static final int portBase;

/**
 * The address of the network to which are attached all network links in
 * this JVM. This is a hash on the account name of the JVM running this
 * Nachos instance. It is used to help prevent packets from other users
 * from accidentally interfering with this network.
 */
public static final byte networkID;

static {
    hash = System.getProperty("user.name").hashCode();
    portBase = 0x4E41 + Math.abs(hash%0x4E41);
    networkID  = (byte) (hash/0x4E41);
}

private Privilege privilege;

    private Runnable receiveInterrupt;
    private Runnable sendInterrupt;

    private Runnable receiveInterruptHandler = null;
    private Runnable sendInterruptHandler = null;

    private InetAddress localHost;
    private DatagramSocket socket;

    private byte linkAddress;
    private double reliability;

    private byte[] incomingBytes = null;
    private Packet incomingPacket = null;
    private Packet outgoingPacket = null;

    private boolean sendBusy = false;
}
```

```
// PART OF THE MACHINE SIMULATION. DO NOT CHANGE.

package nachos.machine;

import java.io.EOFException;

/**
 * A file that supports reading, writing, and seeking.
 */
public class OpenFile {
    /**
     * Allocate a new <tt>OpenFile</tt> object with the specified name on the
     * specified file system.
     *
     * @param   fileSystem      the file system to which this file belongs.
     * @param   name            the name of the file, on that file system.
     */
    public OpenFile(FileSystem fileSystem, String name) {
        this.fileSystem = fileSystem;
        this.name = name;
    }

    /**
     * Allocate a new unnamed <tt>OpenFile</tt> that is not associated with any
     * file system.
     */
    public OpenFile() {
        this(null, "unnamed");
    }

    /**
     * Get the file system to which this file belongs.
     *
     * @return  the file system to which this file belongs.
     */
    public FileSystem getFileSystem() {
        return fileSystem;
    }

    /**
     * Get the name of this open file.
     *
     * @return  the name of this open file.
     */
    public String getName() {
        return name;
    }

    /**
     * Read this file starting at the specified position and return the number
     * of bytes successfully read. If no bytes were read because of a fatal
     * error, returns -1
     *
     * @param   pos     the offset in the file at which to start reading.
     * @param   buf     the buffer to store the bytes in.
     * @param   offset  the offset in the buffer to start storing bytes.
     * @param   length  the number of bytes to read.
     * @return  the actual number of bytes successfully read, or -1 on failure.
     */
    public int read(int pos, byte[] buf, int offset, int length) {
        return -1;
    }

    /**
     * Write this file starting at the specified position and return the number
     * of bytes successfully written. If no bytes were written because of a
     * fatal error, returns -1.
     *
     * @param   pos     the offset in the file at which to start writing.
     * @param   buf     the buffer to get the bytes from.
     * @param   offset  the offset in the buffer to start getting.
     * @param   length  the number of bytes to write.
     * @return  the actual number of bytes successfully written, or -1 on
     *          failure.
     */
    public int write(int pos, byte[] buf, int offset, int length) {
        return -1;
    }

    /**
     * Get the length of this file.
     *
     * @return  the length of this file, or -1 if this file has no length.
     */
    public int length() {
        return -1;
    }

    /**
     * Close this file and release any associated system resources.
     */
    public void close() {
    }

    /**
     * Set the value of the current file pointer.
     */
    public void seek(int pos) {
    }

    /**
     * Get the value of the current file pointer, or -1 if this file has no
     * pointer.
     */
    public int tell() {
        return -1;
    }

    /**
     * Read this file starting at the current file pointer and return the
     * number of bytes successfully read. Advances the file pointer by this
     * amount. If no bytes could be* read because of a fatal error, returns -1.
     *
     * @param   buf     the buffer to store the bytes in.
     * @param   offset  the offset in the buffer to start storing bytes.
     * @param   length  the number of bytes to read.
     * @return  the actual number of bytes successfully read, or -1 on failure.
     */
    public int read(byte[] buf, int offset, int length) {
        return -1;
    }

    /**
     * Write this file starting at the current file pointer and return the
```

```
     * number of bytes successfully written. Advances the file pointer by this
     * amount. If no bytes could be written because of a fatal error, returns
     * -1.
     *
     * @param   buf     the buffer to get the bytes from.
     * @param   offset  the offset in the buffer to start getting.
     * @param   length  the number of bytes to write.
     * @return  the actual number of bytes successfully written, or -1 on
     *          failure.
     */
    public int write(byte[] buf, int offset, int length) {
        return -1;
    }

    private FileSystem fileSystem;
    private String name;
}
```

```
// PART OF THE MACHINE SIMULATION. DO NOT CHANGE.

package nachos.machine;

/**
 * An <tt>OpenFile</tt> that maintains a current file position.
 */
public abstract class OpenFileWithPosition extends OpenFile {
    /**
     * Allocate a new <tt>OpenFileWithPosition</tt> with the specified name on
     * the specified file system.
     *
     * @param   fileSystem      the file system to which this file belongs.
     * @param   name            the name of the file, on that file system.
     */
    public OpenFileWithPosition(FileSystem fileSystem, String name) {
        super(fileSystem, name);
    }

    /**
     * Allocate a new unnamed <tt>OpenFileWithPosition</tt> that is not
     * associated with any file system.
     */
    public OpenFileWithPosition() {
        super();
    }

    public void seek(int position) {
        this.position = position;
    }

    public int tell() {
        return position;
    }

    public int read(byte[] buf, int offset, int length) {
        int amount = read(position, buf, offset, length);
        if (amount == -1)
            return -1;

        position += amount;
        return amount;
    }

    public int write(byte[] buf, int offset, int length) {
        int amount = write(position, buf, offset, length);
        if (amount == -1)
            return -1;

        position += amount;
        return amount;
    }

    /**
     * The current value of the file pointer.
     */
    protected int position = 0;
}
```

```
// PART OF THE MACHINE SIMULATION. DO NOT CHANGE.

package nachos.machine;

/**
 * A link-layer packet.
 *
 * @see nachos.machine.NetworkLink
 */
public class Packet {
    /**
     * Allocate a new packet to be sent, using the specified parameters.
     *
     * @param   dstLink         the destination link address.
     * @param   srcLink         the source link address.
     * @param   contents        the contents of the packet.
     */
    public Packet(int dstLink, int srcLink, byte[] contents)
        throws MalformedPacketException {
        // make sure the paramters are valid
        if (dstLink < 0 || dstLink >= linkAddressLimit ||
            srcLink < 0 || srcLink >= linkAddressLimit ||
            contents.length > maxContentsLength)
            throw new MalformedPacketException();

        this.dstLink = dstLink;
        this.srcLink = srcLink;
        this.contents = contents;

        packetBytes = new byte[headerLength + contents.length];

        packetBytes[0] = NetworkLink.networkID;
        packetBytes[1] = (byte) dstLink;
        packetBytes[2] = (byte) srcLink;
        packetBytes[3] = (byte) contents.length;

        // if java had subarrays, i'd use them. but System.arraycopy is ok...
        System.arraycopy(contents, 0, packetBytes, headerLength,
                         contents.length);
    }

    /**
     * Allocate a new packet using the specified array of bytes received from
     * the network.
     *
     * @param   packetBytes     the bytes making up this packet.
     */
    public Packet(byte[] packetBytes) throws MalformedPacketException {
        this.packetBytes = packetBytes;

        // make sure we have a valid header
        if (packetBytes.length < headerLength ||
            packetBytes[0] != NetworkLink.networkID ||
            packetBytes[1] < 0 || packetBytes[1] >= linkAddressLimit ||
            packetBytes[2] < 0 || packetBytes[2] >= linkAddressLimit ||
            packetBytes[3] < 0 || packetBytes[3] > packetBytes.length-4)
            throw new MalformedPacketException();

        dstLink = packetBytes[1];
        srcLink = packetBytes[2];

        contents = new byte[packetBytes[3]];
        System.arraycopy(packetBytes, headerLength, contents, 0,
                         contents.length);
    }

    /** This packet, as an array of bytes that can be sent on a network. */
    public byte[] packetBytes;
    /** The address of the destination link of this packet. */
    public int dstLink;
    /** The address of the source link of this packet. */
    public int srcLink;
    /** The contents of this packet, excluding the link-layer header. */
    public byte[] contents;

    /**
     * The number of bytes in a link-layer packet header. The header is
     * formatted as follows:
     *
     * <table>
     * <tr><td>offset</td><td>size</td><td>value</td></tr>
     * <tr><td>0</td><td>1</td><td>network ID (collision detecting)</td></tr>
     * <tr><td>1</td><td>1</td><td>destination link address</td></tr>
     * <tr><td>2</td><td>1</td><td>source link address</td></tr>
     * <tr><td>3</td><td>1</td><td>length of contents</td></tr>
     * </table>
     */
    public static final int headerLength = 4;
    /**
     * The maximum length, in bytes, of a packet that can be sent or received
     * on the network.
     */
    public static final int maxPacketLength = 32;
    /**
     * The maximum number of content bytes (not including the header). Note
     * that this is just <tt>maxPacketLength - headerLength</tt>.
     */
    public static final int maxContentsLength = maxPacketLength - headerLength;

    /**
     * The upper limit on Nachos link addresses. All link addresses fall
     * between <tt>0</tt> and <tt>linkAddressLimit - 1</tt>.
     */
    public static final int linkAddressLimit = 128;
}
```

```
// PART OF THE MACHINE SIMULATION. DO NOT CHANGE.

package nachos.machine;

import nachos.security.*;

/**
 * The <tt>Processor</tt> class simulates a MIPS processor that supports a
 * subset of the R3000 instruction set. Specifically, the processor lacks all
 * coprocessor support, and can only execute in user mode. Address translation
 * information is accessed via the API. The API also allows a kernel to set an
 * exception handler to be called on any user mode exception.
 *
 * <p>
 * The <tt>Processor</tt> API is re-entrant, so a single simulated processor
 * can be shared by multiple user threads.
 *
 * <p>
 * An instance of a <tt>Processor</tt> also includes pages of physical memory
 * accessible to user programs, the size of which is fixed by the constructor.
 */
public final class Processor {
    /**
     * Allocate a new MIPS processor, with the specified amount of memory.
     *
     * @param   privilege       encapsulates privileged access to the Nachos
     *                          machine.
     * @param   numPhysPages    the number of pages of physical memory to
     *                          attach.
     */
    public Processor(Privilege privilege, int numPhysPages) {
        System.out.print(" processor");

        this.privilege = privilege;
        privilege.processor = new ProcessorPrivilege();

        Class<?> clsKernel = Lib.loadClass(Config.getString("Kernel.kernel"));
        Class<?> clsVMKernel = Lib.tryLoadClass("nachos.vm.VMKernel");

        usingTLB =
            (clsVMKernel != null && clsVMKernel.isAssignableFrom(clsKernel));

        this.numPhysPages = numPhysPages;

        for (int i=0; i<numUserRegisters; i++)
            registers[i] = 0;

        mainMemory = new byte[pageSize * numPhysPages];

        if (usingTLB) {
            translations = new TranslationEntry[tlbSize];
            for (int i=0; i<tlbSize; i++)
                translations[i] = new TranslationEntry();
        }
        else {
            translations = null;
        }
    }

    /**
     * Set the exception handler, called whenever a user exception occurs.
     *
     * <p>
     * When the exception handler is called, interrupts will be enabled, and
     * the CPU cause register will specify the cause of the exception (see the
     * <tt>exception<i>*</i></tt> constants).
     *
     * @param   exceptionHandler        the kernel exception handler.
     */
    public void setExceptionHandler(Runnable exceptionHandler) {
        this.exceptionHandler = exceptionHandler;
    }

    /**
     * Get the exception handler, set by the last call to
     * <tt>setExceptionHandler()</tt>.
     *
     * @return  the exception handler.
     */
    public Runnable getExceptionHandler() {
        return exceptionHandler;
    }

    /**
     * Start executing instructions at the current PC. Never returns.
     */
    public void run() {
        Lib.debug(dbgProcessor, "starting program in current thread");

        registers[regNextPC] = registers[regPC] + 4;

        Machine.autoGrader().runProcessor(privilege);

        Instruction inst = new Instruction();

        while (true) {
            try {
                inst.run();
            }
            catch (MipsException e) {
                e.handle();
            }

            privilege.interrupt.tick(false);
        }
    }

    /**
     * Read and return the contents of the specified CPU register.
     *
     * @param   number  the register to read.
     * @return  the value of the register.
     */
    public int readRegister(int number) {
        Lib.assertTrue(number >= 0 && number < numUserRegisters);

        return registers[number];
    }

    /**
     * Write the specified value into the specified CPU register.
     *
     * @param   number  the register to write.
     * @param   value   the value to write.
```

```
     */
    public void writeRegister(int number, int value) {
        Lib.assertTrue(number >= 0 && number < numUserRegisters);

        if (number != 0)
            registers[number] = value;
    }

    /**
     * Test whether this processor uses a software-managed TLB, or single-level
     * paging.
     *
     * <p>
     * If <tt>false</tt>, this processor directly supports single-level paging;
     * use <tt>setPageTable()</tt>.
     *
     * <p>
     * If <tt>true</tt>, this processor has a software-managed TLB;
     * use <tt>getTLBSize()</tt>, <tt>readTLBEntry()</tt>, and
     * <tt>writeTLBEntry()</tt>.
     *
     * <p>
     * Using a method associated with the wrong address translation mechanism
     * will result in an assertion failure.
     *
     * @return  <tt>true</tt> if this processor has a software-managed TLB.
     */
    public boolean hasTLB() {
        return usingTLB;
    }

    /**
     * Get the current page table, set by the last call to setPageTable().
     *
     * @return  the current page table.
     */
    public TranslationEntry[] getPageTable() {
        Lib.assertTrue(!usingTLB);

        return translations;
    }

    /**
     * Set the page table pointer. All further address translations will use
     * the specified page table. The size of the current address space will be
     * determined from the length of the page table array.
     *
     * @param   pageTable       the page table to use.
     */
    public void setPageTable(TranslationEntry[] pageTable) {
        Lib.assertTrue(!usingTLB);

        this.translations = pageTable;
    }

    /**
     * Return the number of entries in this processor's TLB.
     *
     * @return  the number of entries in this processor's TLB.
     */
    public int getTLBSize() {
        Lib.assertTrue(usingTLB);

        return tlbSize;
    }

    /**
     * Returns the specified TLB entry.
     *
     * @param   number  the index into the TLB.
     * @return  the contents of the specified TLB entry.
     */
    public TranslationEntry readTLBEntry(int number) {
        Lib.assertTrue(usingTLB);
        Lib.assertTrue(number >= 0 && number < tlbSize);

        return new TranslationEntry(translations[number]);
    }

    /**
     * Fill the specified TLB entry.
     *
     * <p>
     * The TLB is fully associative, so the location of an entry within the TLB
     * does not affect anything.
     *
     * @param   number  the index into the TLB.
     * @param   entry   the new contents of the TLB entry.
     */
    public void writeTLBEntry(int number, TranslationEntry entry) {
        Lib.assertTrue(usingTLB);
        Lib.assertTrue(number >= 0 && number < tlbSize);

        translations[number] = new TranslationEntry(entry);
    }

    /**
     * Return the number of pages of physical memory attached to this simulated
     * processor.
     *
     * @return  the number of pages of physical memory.
     */
    public int getNumPhysPages() {
        return numPhysPages;
    }

    /**
     * Return a reference to the physical memory array. The size of this array
     * is <tt>pageSize * getNumPhysPages()</tt>.
     *
     * @return  the main memory array.
     */
    public byte[] getMemory() {
        return mainMemory;
    }

    /**
     * Concatenate a page number and an offset into an address.
     *
     * @param   page    the page number. Must be between <tt>0</tt> and
     *                  <tt>(2<sup>32</sup> / pageSize) - 1</tt>.
     * @param   offset  the offset within the page. Must be between <tt>0</tt>
     *                  and
     *                  <tt>pageSize - 1</tt>.
```

```
     * @return  a 32-bit address consisting of the specified page and offset.
     */
    public static int makeAddress(int page, int offset) {
        Lib.assertTrue(page >= 0 && page < maxPages);
        Lib.assertTrue(offset >= 0 && offset < pageSize);

        return (page * pageSize) | offset;
    }

    /**
     * Extract the page number component from a 32-bit address.
     *
     * @param   address the 32-bit address.
     * @return  the page number component of the address.
     */
    public static int pageFromAddress(int address) {
        return (int) (((long) address & 0xFFFFFFFFL) / pageSize);
    }

    /**
     * Extract the offset component from an address.
     *
     * @param   address the 32-bit address.
     * @return  the offset component of the address.
     */
    public static int offsetFromAddress(int address) {
        return (int) (((long) address & 0xFFFFFFFFL) % pageSize);
    }

    private void finishLoad() {
        delayedLoad(0, 0, 0);
    }

    /**
     * Translate a virtual address into a physical address, using either a
     * page table or a TLB. Check for alignment, make sure the virtual page is
     * valid, make sure a read-only page is not being written, make sure the
     * resulting physical page is valid, and then return the resulting physical
     * address.
     *
     * @param   vaddr   the virtual address to translate.
     * @param   size    the size of the memory reference (must be 1, 2, or 4).
     * @param   writing <tt>true</tt> if the memory reference is a write.
     * @return          the physical address.
     * @exception       MipsException   if a translation error occurred.
     */
    private int translate(int vaddr, int size, boolean writing)
        throws MipsException {
        if (Lib.test(dbgProcessor))
            System.out.println("\ttranslate vaddr=0x" + Lib.toHexString(vaddr)
                               + (writing ? ", write" : ", read..."));

        // check alignment
        if ((vaddr & (size-1)) != 0) {
            Lib.debug(dbgProcessor, "\t\talignment error");
            throw new MipsException(exceptionAddressError, vaddr);
        }

        // calculate virtual page number and offset from the virtual address
        int vpn = pageFromAddress(vaddr);
        int offset = offsetFromAddress(vaddr);

        TranslationEntry entry = null;

        // if not using a TLB, then the vpn is an index into the table
        if (!usingTLB) {
            if (translations == null || vpn >= translations.length ||
                translations[vpn] == null ||
                !translations[vpn].valid) {
                privilege.stats.numPageFaults++;
                Lib.debug(dbgProcessor, "\t\tpage fault");
                throw new MipsException(exceptionPageFault, vaddr);
            }

            entry = translations[vpn];
        }
        // else, look through all TLB entries for matching vpn
        else {
            for (int i=0; i<tlbSize; i++) {
                if (translations[i].valid && translations[i].vpn == vpn) {
                    entry = translations[i];
                    break;
                }
            }
            if (entry == null) {
                privilege.stats.numTLBMisses++;
                Lib.debug(dbgProcessor, "\t\tTLB miss");
                throw new MipsException(exceptionTLBMiss, vaddr);
            }
        }

        // check if trying to write a read-only page
        if (entry.readOnly && writing) {
            Lib.debug(dbgProcessor, "\t\tread-only exception");
            throw new MipsException(exceptionReadOnly, vaddr);
        }

        // check if physical page number is out of range
        int ppn = entry.ppn;
        if (ppn < 0 || ppn >= numPhysPages) {
            Lib.debug(dbgProcessor, "\t\tbad ppn");
            throw new MipsException(exceptionBusError, vaddr);
        }

        // set used and dirty bits as appropriate
        entry.used = true;
        if (writing)
            entry.dirty = true;

        int paddr = (ppn*pageSize) + offset;

        if (Lib.test(dbgProcessor))
            System.out.println("\t\tpaddr=0x" + Lib.toHexString(paddr));
        return paddr;
    }

    /**
     * Read </i>size</i> (1, 2, or 4) bytes of virtual memory at <i>vaddr</i>,
     * and return the result.
     *
     * @param   vaddr   the virtual address to read from.
     * @param   size    the number of bytes to read (1, 2, or 4).
     * @return          the value read.
     * @exception       MipsException   if a translation error occurred.
```

```
     */
    private int readMem(int vaddr, int size) throws MipsException {
        if (Lib.test(dbgProcessor))
            System.out.println("\treadMem vaddr=0x" + Lib.toHexString(vaddr)
                               + ", size=" + size);

        Lib.assertTrue(size==1 || size==2 || size==4);

        int value = Lib.bytesToInt(mainMemory, translate(vaddr, size, false),
                                   size);

        if (Lib.test(dbgProcessor))
            System.out.println("\t\tvalue read=0x" +
                               Lib.toHexString(value, size*2));

        return value;
    }

    /**
     * Write <i>value</i> to </i>size</i> (1, 2, or 4) bytes of virtual memory
     * starting at <i>vaddr</i>.
     *
     * @param   vaddr   the virtual address to write to.
     * @param   size    the number of bytes to write (1, 2, or 4).
     * @param   value   the value to store.
     * @exception       MipsException   if a translation error occurred.
     */
    private void writeMem(int vaddr, int size, int value)
        throws MipsException {
        if (Lib.test(dbgProcessor))
            System.out.println("\twriteMem vaddr=0x" + Lib.toHexString(vaddr)
                               + ", size=" + size + ", value=0x"
                               + Lib.toHexString(value, size*2));

        Lib.assertTrue(size==1 || size==2 || size==4);

        Lib.bytesFromInt(mainMemory, translate(vaddr, size, true), size,
                         value);
    }

    /**
     * Complete the in progress delayed load and scheduled a new one.
     *
     * @param   nextLoadTarget  the target register of the new load.
     * @param   nextLoadValue   the value to be loaded into the new target.
     * @param   nextLoadMask    the mask specifying which bits in the new
     *                          target are to be overwritten. If a bit in
     *                          <tt>nextLoadMask</tt> is 0, then the
     *                          corresponding bit of register
     *                          <tt>nextLoadTarget</tt> will not be written.
     */
    private void delayedLoad(int nextLoadTarget, int nextLoadValue,
                             int nextLoadMask) {
        // complete previous delayed load, if not modifying r0
        if (loadTarget != 0) {
            int savedBits = registers[loadTarget] & ~loadMask;
            int newBits = loadValue & loadMask;
            registers[loadTarget] = savedBits | newBits;
        }

        // schedule next load
        loadTarget = nextLoadTarget;
        loadValue = nextLoadValue;
        loadMask = nextLoadMask;
    }

    /**
     * Advance the PC to the next instruction.
     *
     * <p>
     * Transfer the contents of the nextPC register into the PC register, and
     * then add 4 to the value in the nextPC register. Same as
     * <tt>advancePC(readRegister(regNextPC)+4)</tt>.
     *
     * <p>
     * Use after handling a syscall exception so that the processor will move
     * on to the next instruction.
     */
    public void advancePC() {
        advancePC(registers[regNextPC]+4);
    }

    /**
     * Transfer the contents of the nextPC register into the PC register, and
     * then write the nextPC register.
     *
     * @param   nextPC  the new value of the nextPC register.
     */
    private void advancePC(int nextPC) {
        registers[regPC] = registers[regNextPC];
        registers[regNextPC] = nextPC;
    }

    /** Caused by a syscall instruction. */
    public static final int exceptionSyscall = 0;
    /** Caused by an access to an invalid virtual page. */
    public static final int exceptionPageFault = 1;
    /** Caused by an access to a virtual page not mapped by any TLB entry. */
    public static final int exceptionTLBMiss = 2;
    /** Caused by a write access to a read-only virtual page. */
    public static final int exceptionReadOnly = 3;
    /** Caused by an access to an invalid physical page. */
    public static final int exceptionBusError = 4;
    /** Caused by an access to a misaligned virtual address. */
    public static final int exceptionAddressError = 5;
    /** Caused by an overflow by a signed operation. */
    public static final int exceptionOverflow = 6;
    /** Caused by an attempt to execute an illegal instruction. */
    public static final int exceptionIllegalInstruction = 7;

    /** The names of the CPU exceptions. */
    public static final String exceptionNames[] = {
        "syscall      ",
        "page fault   ",
        "TLB miss     ",
        "read-only    ",
        "bus error    ",
        "address error",
        "overflow     ",
        "illegal inst "
    };

    /** Index of return value register 0. */
    public static final int regV0 = 2;
```

```
    /** Index of return value register 1. */
    public static final int regV1 = 3;
    /** Index of argument register 0. */
    public static final int regA0 = 4;
    /** Index of argument register 1. */
    public static final int regA1 = 5;
    /** Index of argument register 2. */
    public static final int regA2 = 6;
    /** Index of argument register 3. */
    public static final int regA3 = 7;
    /** Index of the stack pointer register. */
    public static final int regSP = 29;
    /** Index of the return address register. */
    public static final int regRA = 31;
    /** Index of the low register, used for multiplication and division. */
    public static final int regLo = 32;
    /** Index of the high register, used for multiplication and division. */
    public static final int regHi = 33;
    /** Index of the program counter register. */
    public static final int regPC = 34;
    /** Index of the next program counter register. */
    public static final int regNextPC = 35;
    /** Index of the exception cause register. */
    public static final int regCause = 36;
    /** Index of the exception bad virtual address register. */
    public static final int regBadVAddr = 37;

    /** The total number of software-accessible CPU registers. */
    public static final int numUserRegisters = 38;

    /** Provides privilege to this processor. */
    private Privilege privilege;

    /** MIPS registers accessible to the kernel. */
    private int registers[] = new int[numUserRegisters];

    /** The registered target of the delayed load currently in progress. */
    private int loadTarget = 0;
    /** The bits to be modified by the delayed load currently in progress. */
    private int loadMask;
    /** The value to be loaded by the delayed load currently in progress. */
    private int loadValue;

    /** <tt>true</tt> if using a software-managed TLB. */
    private boolean usingTLB;
    /** Number of TLB entries. */
    private int tlbSize = 4;
    /**
     * Either an associative or direct-mapped set of translation entries,
     * depending on whether there is a TLB.
     */
    private TranslationEntry[] translations;

    /** Size of a page, in bytes. */
    public static final int pageSize = 0x400;
    /** Number of pages in a 32-bit address space. */
    public static final int maxPages = (int) (0x100000000L / pageSize);
    /** Number of physical pages in memory. */
    private int numPhysPages;
    /** Main memory for user programs. */
    private byte[] mainMemory;

    /** The kernel exception handler, called on every user exception. */
    private Runnable exceptionHandler = null;

    private static final char dbgProcessor = 'p';
    private static final char dbgDisassemble = 'm';
    private static final char dbgFullDisassemble = 'M';

    private class ProcessorPrivilege implements Privilege.ProcessorPrivilege {
        public void flushPipe() {
            finishLoad();
        }
    }

    private class MipsException extends Exception {
        public MipsException(int cause) {
            Lib.assertTrue(cause >= 0 && cause < exceptionNames.length);

            this.cause = cause;
        }

        public MipsException(int cause, int badVAddr) {
            this(cause);

            hasBadVAddr = true;
            this.badVAddr = badVAddr;
        }

        public void handle() {
            writeRegister(regCause, cause);

            if (hasBadVAddr)
                writeRegister(regBadVAddr, badVAddr);

            if (Lib.test(dbgDisassemble) || Lib.test(dbgFullDisassemble))
                System.out.println("exception: " + exceptionNames[cause]);

            finishLoad();

            Lib.assertTrue(exceptionHandler != null);

            // autograder might not want kernel to know about this exception
            if (!Machine.autoGrader().exceptionHandler(privilege))
                return;

            exceptionHandler.run();
        }

        private boolean hasBadVAddr = false;
        private int cause, badVAddr;
    }

    private class Instruction {
        public void run() throws MipsException {
            // hopefully this looks familiar to 152 students?
            fetch();
            decode();
            execute();
            writeBack();
        }

        private boolean test(int flag) {
            return Lib.test(flag, flags);
```

```
    }

    private void fetch() throws MipsException {
        if ((Lib.test(dbgDisassemble) && !Lib.test(dbgProcessor)) ||
            Lib.test(dbgFullDisassemble))
            System.out.print("PC=0x" + Lib.toHexString(registers[regPC])
                             + "\t");

        value = readMem(registers[regPC], 4);
    }

    private void decode() {
        op = Lib.extract(value, 26, 6);
        rs = Lib.extract(value, 21, 5);
        rt = Lib.extract(value, 16, 5);
        rd = Lib.extract(value, 11, 5);
        sh = Lib.extract(value, 6, 5);
        func = Lib.extract(value, 0, 6);
        target = Lib.extract(value, 0, 26);
        imm = Lib.extend(value, 0, 16);

        Mips info;
        switch (op) {
        case 0:
            info = Mips.specialtable[func];
            break;
        case 1:
            info = Mips.regimmtable[rt];
            break;
        default:
            info = Mips.optable[op];
            break;
        }

        operation = info.operation;
        name = info.name;
        format = info.format;
        flags = info.flags;

        mask = 0xFFFFFFFF;
        branch = true;

        // get memory access size
        if (test(Mips.SIZEB))
            size = 1;
        else if (test(Mips.SIZEH))
            size = 2;
        else if (test(Mips.SIZEW))
            size = 4;
        else
            size = 0;

        // get nextPC
        nextPC = registers[regNextPC]+4;

        // get dstReg
        if (test(Mips.DSTRA))
            dstReg = regRA;
        else if (format == Mips.IFMT)
            dstReg = rt;
        else if (format == Mips.RFMT)
            dstReg = rd;
        else
            dstReg = -1;

        // get jtarget
        if (format == Mips.RFMT)
            jtarget = registers[rs];
        else if (format == Mips.IFMT)
            jtarget = registers[regNextPC] + (imm<<2);
        else if (format == Mips.JFMT)
            jtarget = (registers[regNextPC]&0xF0000000) | (target<<2);
        else
            jtarget = -1;

        // get imm
        if (test(Mips.UNSIGNED)) {
            imm &= 0xFFFF;
        }

        // get addr
        addr = registers[rs] + imm;

        // get src1
        if (test(Mips.SRC1SH))
            src1 = sh;
        else
            src1 = registers[rs];

        // get src2
        if (test(Mips.SRC2IMM))
            src2 = imm;
        else
            src2 = registers[rt];

        if (test(Mips.UNSIGNED)) {
            src1 &= 0xFFFFFFFFL;
            src2 &= 0xFFFFFFFFL;
        }

        if (Lib.test(dbgDisassemble) || Lib.test(dbgFullDisassemble))
            print();
    }

    private void print() {
        if (Lib.test(dbgDisassemble) && Lib.test(dbgProcessor) &&
            !Lib.test(dbgFullDisassemble))
            System.out.print("PC=0x" + Lib.toHexString(registers[regPC])
                             + "\t");

        if (operation == Mips.INVALID) {
            System.out.print("invalid: op=" + Lib.toHexString(op, 2) +
                             " rs=" + Lib.toHexString(rs, 2) +
                             " rt=" + Lib.toHexString(rt, 2) +
                             " rd=" + Lib.toHexString(rd, 2) +
                             " sh=" + Lib.toHexString(sh, 2) +
                             " func=" + Lib.toHexString(func, 2) +
                             "\n");
            return;
        }

        int spaceIndex = name.indexOf(' ');
        Lib.assertTrue(spaceIndex!=-1 && spaceIndex==name.lastIndexOf(' '));
```

```
        String instname = name.substring(0, spaceIndex);
        char[] args = name.substring(spaceIndex+1).toCharArray();

        System.out.print(instname + "\t");

        int minCharsPrinted = 0, maxCharsPrinted = 0;

        for (int i=0; i<args.length; i++) {
            switch (args[i]) {
            case Mips.RS:
                System.out.print("$" + rs);
                minCharsPrinted += 2;
                maxCharsPrinted += 3;

                if (Lib.test(dbgFullDisassemble)) {
                    System.out.print("#0x" +
                                     Lib.toHexString(registers[rs]));
                    minCharsPrinted += 11;
                    maxCharsPrinted += 11;
                }
                break;
            case Mips.RT:
                System.out.print("$" + rt);
                minCharsPrinted += 2;
                maxCharsPrinted += 3;

                if (Lib.test(dbgFullDisassemble) &&
                    (i!=0 || !test(Mips.DST)) &&
                    !test(Mips.DELAYEDLOAD)) {
                    System.out.print("#0x" +
                                     Lib.toHexString(registers[rt]));
                    minCharsPrinted += 11;
                    maxCharsPrinted += 11;
                }
                break;
            case Mips.RETURNADDRESS:
                if (rd == 31)
                    continue;
            case Mips.RD:
                System.out.print("$" + rd);
                minCharsPrinted += 2;
                maxCharsPrinted += 3;
                break;
            case Mips.IMM:
                System.out.print(imm);
                minCharsPrinted += 1;
                maxCharsPrinted += 6;
                break;
            case Mips.SHIFTAMOUNT:
                System.out.print(sh);
                minCharsPrinted += 1;
                maxCharsPrinted += 2;
                break;
            case Mips.ADDR:
                System.out.print(imm + "($" + rs);
                minCharsPrinted += 4;
                maxCharsPrinted += 5;

                if (Lib.test(dbgFullDisassemble)) {
                    System.out.print("#0x" +
                                     Lib.toHexString(registers[rs]));
                    minCharsPrinted += 11;
                    maxCharsPrinted += 11;
                }

                System.out.print(")");
                break;
            case Mips.TARGET:
                System.out.print("0x" + Lib.toHexString(jtarget));
                minCharsPrinted += 10;
                maxCharsPrinted += 10;
                break;
            default:
                Lib.assertTrue(false);
            }
            if (i+1 < args.length) {
                System.out.print(", ");
                minCharsPrinted += 2;
                maxCharsPrinted += 2;
            }
            else {
                // most separation possible is tsi, 5+1+1=7,
                // thankfully less than 8 (makes this possible)
                Lib.assertTrue(maxCharsPrinted-minCharsPrinted < 8);
                // longest string is stj, which is 40-42 chars w/ -d M;
                // go for 48
                while ((minCharsPrinted%8) != 0) {
                    System.out.print(" ");
                    minCharsPrinted++;
                    maxCharsPrinted++;
                }
                while (minCharsPrinted < 48) {
                    System.out.print("\t");
                    minCharsPrinted += 8;
                }
            }
        }

        if (Lib.test(dbgDisassemble) && Lib.test(dbgProcessor) &&
            !Lib.test(dbgFullDisassemble))
            System.out.print("\n");
    }

    private void execute() throws MipsException {
        int value;
        int preserved;

        switch (operation) {
        case Mips.ADD:
            dst = src1 + src2;
            break;
        case Mips.SUB:
            dst = src1 - src2;
            break;
        case Mips.MULT:
            dst = src1 * src2;
            registers[regLo] = (int) Lib.extract(dst, 0, 32);
            registers[regHi] = (int) Lib.extract(dst, 32, 32);
            break;
        case Mips.DIV:
            try {
                registers[regLo] = (int) (src1 / src2);
                registers[regHi] = (int) (src1 % src2);
                if (registers[regLo]*src2 + registers[regHi] != src1)
```

```
                throw new ArithmeticException();
            }
            catch (ArithmeticException e) {
                throw new MipsException(exceptionOverflow);
            }
            break;

        case Mips.SLL:
            dst = src2 << (src1&0x1F);
            break;
        case Mips.SRA:
            dst = src2 >> (src1&0x1F);
            break;
        case Mips.SRL:
            dst = src2 >>> (src1&0x1F);
            break;

        case Mips.SLT:
            dst = (src1<src2) ? 1 : 0;
            break;

        case Mips.AND:
            dst = src1 & src2;
            break;
        case Mips.OR:
            dst = src1 | src2;
            break;
        case Mips.NOR:
            dst = ~(src1 | src2);
            break;
        case Mips.XOR:
            dst = src1 ^ src2;
            break;
        case Mips.LUI:
            dst = imm << 16;
            break;

        case Mips.BEQ:
            branch = (src1 == src2);
            break;
        case Mips.BNE:
            branch = (src1 != src2);
            break;
        case Mips.BGEZ:
            branch = (src1 >= 0);
            break;
        case Mips.BGTZ:
            branch = (src1 > 0);
            break;
        case Mips.BLEZ:
            branch = (src1 <= 0);
            break;
        case Mips.BLTZ:
            branch = (src1 < 0);
            break;

        case Mips.JUMP:
            break;

        case Mips.MFLO:
            dst = registers[regLo];
            break;
        case Mips.MFHI:
            dst = registers[regHi];
            break;
        case Mips.MTLO:
            registers[regLo] = (int) src1;
            break;
        case Mips.MTHI:
            registers[regHi] = (int) src1;
            break;

        case Mips.SYSCALL:
            throw new MipsException(exceptionSyscall);

        case Mips.LOAD:
            value = readMem(addr, size);

            if (!test(Mips.UNSIGNED))
                dst = Lib.extend(value, 0, size*8);
            else
                dst = value;

            break;

        case Mips.LWL:
            value = readMem(addr&~0x3, 4);

            // LWL shifts the input left so the addressed byte is highest
            preserved = (3-(addr&0x3))*8;   // number of bits to preserve
            mask = -1 << preserved;         // preserved bits are 0 in mask
            dst = value << preserved;       // shift input to correct place
            addr &= ~0x3;

            break;

        case Mips.LWR:
            value = readMem(addr&~0x3, 4);

            // LWR shifts the input right so the addressed byte is lowest
            preserved = (addr&0x3)*8;       // number of bits to preserve
            mask = -1 >>> preserved;        // preserved bits are 0 in mask
            dst = value >>> preserved;      // shift input to correct place
            addr &= ~0x3;

            break;

        case Mips.STORE:
            writeMem(addr, size, (int) src2);
            break;

        case Mips.SWL:
            value = readMem(addr&~0x3, 4);

            // SWL shifts highest order byte into the addressed position
            preserved = (3-(addr&0x3))*8;
            mask = -1 >>> preserved;
            dst = src2 >>> preserved;

            // merge values
            dst = (dst & mask) | (value & ~mask);

            writeMem(addr&~0x3, 4, (int) dst);
            break;
```

```
        case Mips.SWR:
            value = readMem(addr&~0x3, 4);

            // SWR shifts the lowest order byte into the addressed position
            preserved = (addr&0x3)*8;
            mask = -1 << preserved;
            dst = src2 << preserved;

            // merge values
            dst = (dst & mask) | (value & ~mask);

            writeMem(addr&~0x3, 4, (int) dst);
            break;

        case Mips.UNIMPL:
            System.err.println("Warning: encountered unimplemented inst");

        case Mips.INVALID:
            throw new MipsException(exceptionIllegalInstruction);

        default:
            Lib.assertNotReached();
        }
    }

    private void writeBack() throws MipsException {
        // if instruction is signed, but carry bit !+ sign bit, throw
        if (test(Mips.OVERFLOW) && Lib.test(dst,31) != Lib.test(dst,32))
            throw new MipsException(exceptionOverflow);

        if (test(Mips.DELAYEDLOAD))
            delayedLoad(dstReg, (int) dst, mask);
        else
            finishLoad();

        if (test(Mips.LINK))
            dst = nextPC;

        if (test(Mips.DST) && dstReg != 0)
            registers[dstReg] = (int) dst;

        if ((test(Mips.DST) || test(Mips.DELAYEDLOAD)) && dstReg != 0) {
            if (Lib.test(dbgFullDisassemble)) {
                System.out.print("#0x" + Lib.toHexString((int) dst));
                if (test(Mips.DELAYEDLOAD))
                    System.out.print(" (delayed load)");
            }
        }

        if (test(Mips.BRANCH) && branch) {
            nextPC = jtarget;
        }

        advancePC(nextPC);

        if ((Lib.test(dbgDisassemble) && !Lib.test(dbgProcessor)) ||
            Lib.test(dbgFullDisassemble))
            System.out.print("\n");
    }

    // state used to execute a single instruction
        int value, op, rs, rt, rd, sh, func, target, imm;
        int operation, format, flags;
        String name;

        int size;
        int addr, nextPC, jtarget, dstReg;
        long src1, src2, dst;
        int mask;
        boolean branch;
    }

    private static class Mips {
        Mips() {
        }

        Mips(int operation, String name) {
            this.operation = operation;
            this.name = name;
        }

        Mips(int operation, String name, int format, int flags) {
            this(operation, name);
            this.format = format;
            this.flags = flags;
        }

        int operation = INVALID;
        String name = "invalid ";
        int format;
        int flags;

        // operation types
        static final int
            INVALID     = 0,
            UNIMPL      = 1,
            ADD         = 2,
            SUB         = 3,
            MULT        = 4,
            DIV         = 5,
            SLL         = 6,
            SRA         = 7,
            SRL         = 8,
            SLT         = 9,
            AND         = 10,
            OR          = 11,
            NOR         = 12,
            XOR         = 13,
            LUI         = 14,
            MFLO        = 21,
            MFHI        = 22,
            MTLO        = 23,
            MTHI        = 24,
            JUMP        = 25,
            BEQ         = 26,
            BNE         = 27,
            BLEZ        = 28,
            BGTZ        = 29,
            BLTZ        = 30,
            BGEZ        = 31,
            SYSCALL     = 32,
            LOAD        = 33,
            LWL         = 36,
```

```
    LWR         = 37,
    STORE       = 38,
    SWL         = 39,
    SWR         = 40,
    MAX         = 40;

static final int
    IFMT = 1,
    JFMT = 2,
    RFMT = 3;

static final int
    DST         = 0x00000001,
    DSTRA       = 0x00000002,
    OVERFLOW    = 0x00000004,
    SRC1SH      = 0x00000008,
    SRC2IMM     = 0x00000010,
    UNSIGNED    = 0x00000020,
    LINK        = 0x00000040,
    DELAYEDLOAD = 0x00000080,
    SIZEB       = 0x00000100,
    SIZEH       = 0x00000200,
    SIZEW       = 0x00000400,
    BRANCH      = 0x00000800;

static final char
    RS                  = 's',
    RT                  = 't',
    RD                  = 'd',
    IMM                 = 'i',
    SHIFTAMOUNT         = 'h',
    ADDR                = 'a',  // imm(rs)
    TARGET              = 'j',
    RETURNADDRESS       = 'r';  // rd, or none if rd=31; can't be last

static final Mips[] optable = {
    new Mips(),                                             // special
    new Mips(),                                             // reg-imm
    new Mips(JUMP,      "j j",          JFMT, BRANCH),
    new Mips(JUMP,      "jal j",        JFMT, BRANCH|LINK|DST|DSTRA),
    new Mips(BEQ,       "beq stj",      IFMT, BRANCH),
    new Mips(BNE,       "bne stj",      IFMT, BRANCH),
    new Mips(BLEZ,      "blez sj",      IFMT, BRANCH),
    new Mips(BGTZ,      "bgtz sj",      IFMT, BRANCH),
    new Mips(ADD,       "addi tsi",     IFMT, DST|SRC2IMM|OVERFLOW),
    new Mips(ADD,       "addiu tsi",    IFMT, DST|SRC2IMM),
    new Mips(SLT,       "slti tsi",     IFMT, DST|SRC2IMM),
    new Mips(SLT,       "sltiu tsi",    IFMT, DST|SRC2IMM|UNSIGNED),
    new Mips(AND,       "andi tsi",     IFMT, DST|SRC2IMM|UNSIGNED),
    new Mips(OR,        "ori tsi",      IFMT, DST|SRC2IMM|UNSIGNED),
    new Mips(XOR,       "xori tsi",     IFMT, DST|SRC2IMM|UNSIGNED),
    new Mips(LUI,       "lui ti",       IFMT, DST|SRC2IMM|UNSIGNED),
    new Mips(),
    new Mips(),
    new Mips(),
    new Mips(),
    new Mips(BEQ,       "beql stj",     IFMT, BRANCH),
    new Mips(BNE,       "bnel stj",     IFMT, BRANCH),
    new Mips(BLEZ,      "blezl sj",     IFMT, BRANCH),
    new Mips(BGTZ,      "bgtzl sj",     IFMT, BRANCH),
    new Mips(),
    new Mips(),
    new Mips(),
    new Mips(),
    new Mips(),
    new Mips(),
    new Mips(),
    new Mips(LOAD,      "lb ta",        IFMT, DELAYEDLOAD|SIZEB),
    new Mips(LOAD,      "lh ta",        IFMT, DELAYEDLOAD|SIZEH),
    new Mips(LWL,       "lwl ta",       IFMT, DELAYEDLOAD),
    new Mips(LOAD,      "lw ta",        IFMT, DELAYEDLOAD|SIZEW),
    new Mips(LOAD,      "lbu ta",    IFMT, DELAYEDLOAD|SIZEB|UNSIGNED),
    new Mips(LOAD,      "lhu ta",    IFMT, DELAYEDLOAD|SIZEH|UNSIGNED),
    new Mips(LWR,       "lwr ta",       IFMT, DELAYEDLOAD),
    new Mips(),
    new Mips(STORE,     "sb ta",        IFMT, SIZEB),
    new Mips(STORE,     "sh ta",        IFMT, SIZEH),
    new Mips(SWL,       "swl ta",       IFMT, 0),
    new Mips(STORE,     "sw ta",        IFMT, SIZEW),
    new Mips(),
    new Mips(),
    new Mips(SWR,       "swr ta",       IFMT, 0),
    new Mips(),
    new Mips(UNIMPL,    "ll "),
    new Mips(),
    new Mips(),
    new Mips(),
    new Mips(),
    new Mips(),
    new Mips(),
    new Mips(),
    new Mips(UNIMPL,    "sc "),
    new Mips(),
    new Mips(),
    new Mips(),
    new Mips(),
    new Mips(),
    new Mips(),
    new Mips(),
};

static final Mips[] specialtable = {
    new Mips(SLL,       "sll dth",      RFMT, DST|SRC1SH),
    new Mips(),
    new Mips(SRL,       "srl dth",      RFMT, DST|SRC1SH),
    new Mips(SRA,       "sra dth",      RFMT, DST|SRC1SH),
    new Mips(SLL,       "sllv dts",     RFMT, DST),
    new Mips(),
    new Mips(SRL,       "srlv dts",     RFMT, DST),
    new Mips(SRA,       "srav dts",     RFMT, DST),
    new Mips(JUMP,      "jr s",         RFMT, BRANCH),
    new Mips(JUMP,      "jalr rs",      RFMT, BRANCH|LINK|DST),
    new Mips(),
    new Mips(),
    new Mips(SYSCALL,   "syscall "),
    new Mips(UNIMPL,    "break "),
    new Mips(),
    new Mips(UNIMPL,    "sync "),
    new Mips(MFHI,      "mfhi d",       RFMT, DST),
    new Mips(MTHI,      "mthi s",       RFMT, 0),
    new Mips(MFLO,      "mflo d",       RFMT, DST),
    new Mips(MTLO,      "mtlo s",       RFMT, 0),
    new Mips(),
```

```
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(MULT,      "mult st",      RFMT, 0),
        new Mips(MULT,      "multu st",     RFMT, UNSIGNED),
        new Mips(DIV,       "div st",       RFMT, 0),
        new Mips(DIV,       "divu st",      RFMT, UNSIGNED),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(ADD,       "add dst",      RFMT, DST|OVERFLOW),
        new Mips(ADD,       "addu dst",     RFMT, DST),
        new Mips(SUB,       "sub dst",      RFMT, DST|OVERFLOW),
        new Mips(SUB,       "subu dst",     RFMT, DST),
        new Mips(AND,       "and dst",      RFMT, DST),
        new Mips(OR,        "or dst",       RFMT, DST),
        new Mips(XOR,       "xor dst",      RFMT, DST),
        new Mips(NOR,       "nor dst",      RFMT, DST),
        new Mips(),
        new Mips(),
        new Mips(SLT,       "slt dst",      RFMT, DST),
        new Mips(SLT,       "sltu dst",     RFMT, DST|UNSIGNED),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
    };

    static final Mips[] regimmtable = {
        new Mips(BLTZ,      "bltz sj",      IFMT, BRANCH),
        new Mips(BGEZ,      "bgez sj",      IFMT, BRANCH),
        new Mips(BLTZ,      "bltzl sj",     IFMT, BRANCH),
        new Mips(BGEZ,      "bgezl sj",     IFMT, BRANCH),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(BLTZ,      "bltzal sj",    IFMT, BRANCH|LINK|DST|DSTRA),
        new Mips(BGEZ,      "bgezal sj",    IFMT, BRANCH|LINK|DST|DSTRA),
        new Mips(BLTZ,      "bltzlal sj",   IFMT, BRANCH|LINK|DST|DSTRA),
        new Mips(BGEZ,      "bgezlal sj",   IFMT, BRANCH|LINK|DST|DSTRA),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips(),
        new Mips()
    };
  }
}
```

```
// PART OF THE MACHINE SIMULATION. DO NOT CHANGE.

package nachos.machine;

/**
 * A set of controls that can be used by a rider controller. Each rider uses a
 * distinct <tt>RiderControls</tt> object.
 */
public interface RiderControls {
    /**
     * Return the number of floors in the elevator bank. If <i>n</i> is the
     * number of floors in the bank, then the floors are numbered <i>0</i>
     * (the ground floor) through <i>n - 1</i> (the top floor).
     *
     * @return  the number of floors in the bank.
     */
    public int getNumFloors();

    /**
     * Return the number of elevators in the elevator bank. If <i>n</i> is the
     * number of elevators in the bank, then the elevators are numbered
     * <i>0</i> through <i>n - 1</i>.
     *
     * @return  the numbe rof elevators in the bank.
     */
    public int getNumElevators();

    /**
     * Set the rider's interrupt handler. This handler will be called when the
     * rider observes an event.
     *
     * @param   handler the rider's interrupt handler.
     */
    public void setInterruptHandler(Runnable handler);

    /**
     * Return the current location of the rider. If the rider is in motion,
     * the returned value will be within one of the exact location.
     *
     * @return  the floor the rider is on.
     */
    public int getFloor();

    /**
     * Return an array specifying the sequence of floors at which this rider
     * has successfully exited an elevator. This array naturally does not
     * count the floor the rider started on, nor does it count floors where
     * the rider is in the elevator and does not exit.
     *
     * @return  an array specifying the floors this rider has visited.
     */
    public int[] getFloors();

    /**
     * Return the indicated direction of the specified elevator, set by
     * <tt>ElevatorControls.setDirectionDisplay()</tt>.
     *
     * @param   elevator        the elevator to check the direction of.
     * @return  the displayed direction for the elevator.
     *
     * @see     nachos.machine.ElevatorControls#setDirectionDisplay
     */
    public int getDirectionDisplay(int elevator);

    /**
     * Press a direction button. If <tt>up</tt> is <tt>true</tt>, invoke
     * <tt>pressUpButton()</tt>; otherwise, invoke <tt>pressDownButton()</tt>.
     *
     * @param   up      <tt>true</tt> to press the up button, <tt>false</tt> to
     *                  press the down button.
     * @return  the return value of <tt>pressUpButton()</tt> or of
     *          <tt>pressDownButton()</tt>.
     */
    public boolean pressDirectionButton(boolean up);

    /**
     * Press the up button. The rider must not be on the top floor and must not
     * be inside an elevator. If an elevator is on the same floor as this
     * rider, has the doors open, and says it is going up, does nothing and
     * returns <tt>false</tt>.
     *
     * @return  <tt>true</tt> if the button event was sent to the elevator
     *          controller.
     */
    public boolean pressUpButton();

    /**
     * Press the down button. The rider must not be on the bottom floor and
     * must not be inside an elevator. If an elevator is on the same floor as
     * as this rider, has the doors open, and says it is going down, does
     * nothing and returns <tt>false</tt>.
     *
     * @return  <tt>true</tt> if the button event was sent to the elevator
     *          controller.
     */
    public boolean pressDownButton();

    /**
     * Enter an elevator. A rider cannot enter an elevator if its doors are not
     * open at the same floor, or if the elevator is full. The rider must not
     * already be in an elevator.
     *
     * @param   elevator        the elevator to enter.
     * @return  <tt>true</tt> if the rider successfully entered the elevator.
     */
    public boolean enterElevator(int elevator);

    /**
     * Press a floor button. The rider must be inside an elevator. If the
     * elevator already has its doors open on <tt>floor</tt>, does nothing and
     * returns <tt>false</tt>.
     *
     * @param   floor   the button to press.
     * @return  <tt>true</tt> if the button event was sent to the elevator
     *          controller.
     */
    public boolean pressFloorButton(int floor);

    /**
     * Exit the elevator. A rider cannot exit the elevator if its doors are not
     * open on the requested floor. The rider must already be in an elevator.
     *
     * @param   floor   the floor to exit on.
     * @return  <tt>true</tt> if the rider successfully got off the elevator.
```

```
     */
    public boolean exitElevator(int floor);

    /**
     * Call when the rider is finished.
     */
    public void finish();

    /**
     * Return the next event in the event queue. Note that there may be
     * multiple events pending when a rider interrupt occurs, so this
     * method should be called repeatedly until it returns <tt>null</tt>.
     *
     * @return  the next event, or <tt>null</tt> if no further events are
     * currently pending.
     */
    public RiderEvent getNextEvent();
}
```

```
// PART OF THE MACHINE SIMULATION. DO NOT CHANGE.

package nachos.machine;

/**
 * An event that affects rider software. If a rider is outside the elevators,
 * it will only receive events on the same floor as the rider. If a rider is
 * inside an elevator, it will only receive events pertaining to that elevator.
 */
public final class RiderEvent {
    public RiderEvent(int event, int floor, int elevator, int direction) {
        this.event = event;
        this.floor = floor;
        this.elevator = elevator;
        this.direction = direction;
    }

    /** The event identifier. Refer to the <i>event*</i> constants. */
    public final int event;
    /** The floor pertaining to the event, or -1 if not applicable. */
    public final int floor;
    /** The elevator pertaining to the event, or -1 if not applicable. */
    public final int elevator;
    /** The direction display of the elevator (neither if not applicable). */
    public final int direction;

    /** An elevator's doors have opened. */
    public static final int eventDoorsOpened = 0;
    /** An elevator's doors were open and its direction display changed. */
    public static final int eventDirectionChanged = 1;
    /** An elevator's doors have closed. */
    public static final int eventDoorsClosed = 2;
}
```

```
// PART OF THE MACHINE SIMULATION. DO NOT CHANGE.

package nachos.machine;

/**
 * A single rider. Each rider accesses the elevator bank through an
 * instance of <tt>RiderControls</tt>.
 */
public interface RiderInterface extends Runnable {
    /**
     * Initialize this rider. The rider will access the elevator bank through
     * <i>controls</i>, and the rider will make stops at different floors as
     * specified in <i>stops</i>. This method should return immediately after
     * this rider is initialized, but not until the interrupt handler is
     * set. The rider will start receiving events after this method returns,
     * potentially before <tt>run()</tt> is called.
     *
     * @param   controls        the rider's interface to the elevator bank. The
     *                          rider must not attempt to access the elevator
     *                          bank in <i>any</i> other way.
     * @param   stops           an array of stops the rider should make; see
     *                          below.
     */
    public void initialize(RiderControls controls, int[] stops);

    /**
     * Cause the rider to use the provided controls to make the stops specified
     * in the constructor. The rider should stop at each of the floors in
     * <i>stops</i>, an array of floor numbers. The rider should <i>only</i>
     * make the specified stops.
     *
     * <p>
     * For example, suppose the rider uses <i>controls</i> to determine that
     * it is initially on floor 1, and suppose the stops array contains two
     * elements: { 0, 2 }. Then the rider should get on an elevator, get off
     * on floor 0, get on an elevator, and get off on floor 2, pushing buttons
     * as necessary.
     *
     * <p>
     * This method should not return, but instead should call
     * <tt>controls.finish()</tt> when the rider is finished.
     */
    public void run();

    /** Indicates an elevator intends to move down. */
    public static final int dirDown = -1;
    /** Indicates an elevator intends not to move. */
    public static final int dirNeither = 0;
    /** Indicates an elevator intends to move up. */
    public static final int dirUp = 1;
}
```

```
// PART OF THE MACHINE SIMULATION. DO NOT CHANGE.

package nachos.machine;

import nachos.security.*;

/**
 * A serial console can be used to send and receive characters. Only one
 * character may be sent at a time, and only one character may be received at a
 * time.
 */

public interface SerialConsole {
    /**
     * Set this console's receive and send interrupt handlers.
     *
     * <p>
     * The receive interrupt handler is called every time another byte arrives
     * and can be read using <tt>readByte()</tt>.
     *
     * <p>
     * The send interrupt handler is called every time a byte sent with
     * <tt>writeByte()</tt> is finished being sent. This means that another
     * byte can be sent.
     *
     * @param   receiveInterruptHandler the callback to call when a byte
     *                                  arrives.
     * @param   sendInterruptHandler    the callback to call when another byte
     *                                  can be sent.
     */
    public void setInterruptHandlers(Runnable receiveInterruptHandler,
                                     Runnable sendInterruptHandler);

    /**
     * Return the next unsigned byte received (in the range <tt>0</tt> through
     * <tt>255</tt>).
     *
     * @return  the next byte read, or -1 if no byte is available.
     */
    public int  readByte();

    /**
     * Send another byte. If a byte is already being sent, the result is not
     * defined.
     *
     * @param   value   the byte to be sent (the upper 24 bits are ignored).
     */
    public void writeByte(int value);
}
```

```
// PART OF THE MACHINE SIMULATION. DO NOT CHANGE.

package nachos.machine;

import nachos.security.*;

import java.io.IOException;

/**
 * A text-based console that uses System.in and System.out.
 */
public class StandardConsole implements SerialConsole {
    /**
     * Allocate a new standard console.
     *
     * @param   privilege       encapsulates privileged access to the Nachos
     *                          machine.
     */
    public StandardConsole(Privilege privilege) {
        System.out.print(" console");

        this.privilege = privilege;

        receiveInterrupt = new Runnable() {
                public void run() { receiveInterrupt(); }
            };

        sendInterrupt = new Runnable() {
                public void run() { sendInterrupt(); }
            };

        scheduleReceiveInterrupt();
    }

    public final void setInterruptHandlers(Runnable receiveInterruptHandler,
                                           Runnable sendInterruptHandler) {
        this.receiveInterruptHandler = receiveInterruptHandler;
        this.sendInterruptHandler = sendInterruptHandler;
    }

    private void scheduleReceiveInterrupt() {
        privilege.interrupt.schedule(Stats.ConsoleTime, "console read",
                                     receiveInterrupt);
    }

    /**
     * Attempt to read a byte from the object backing this console.
     *
     * @return  the byte read, or -1 of no data is available.
     */
    protected int in() {
        try {
            if (System.in.available() <= 0)
                return -1;

            return System.in.read();
        }
        catch (IOException e) {
            return -1;
        }
    }

    private int translateCharacter(int c) {
        // translate win32 0x0D 0x0A sequence to single newline
        if (c == 0x0A && prevCarriageReturn) {
            prevCarriageReturn = false;
            return -1;
        }
        prevCarriageReturn = (c == 0x0D);

        // invalid if non-ASCII
        if (c >= 0x80)
            return -1;
        // backspace characters
        else if (c == 0x04 || c == 0x08 || c == 0x19 || c == 0x1B || c == 0x7F)
            return '\b';
        // if normal ASCII range, nothing to do
        else if (c >= 0x20)
            return c;
        // newline characters
        else if (c == 0x0A || c == 0x0D)
            return '\n';
        // everything else is invalid
        else
            return -1;
    }


    private void receiveInterrupt() {
        Lib.assertTrue(incomingKey == -1);

        incomingKey = translateCharacter(in());
        if (incomingKey == -1) {
            scheduleReceiveInterrupt();
        }
        else {
            privilege.stats.numConsoleReads++;

            if (receiveInterruptHandler != null)
                receiveInterruptHandler.run();
        }
    }

    public final int readByte() {
        int key = incomingKey;

        if (incomingKey != -1) {
            incomingKey = -1;
            scheduleReceiveInterrupt();
        }

        return key;
    }

    private void scheduleSendInterrupt() {
        privilege.interrupt.schedule(Stats.ConsoleTime, "console write",
                                     sendInterrupt);
    }

    /**
     * Write a byte to the object backing this console.
     *
     * @param   value   the byte to write.
     */
```

```
    protected void out(int value) {
        System.out.write(value);
        System.out.flush();
    }

    private void sendInterrupt() {
        Lib.assertTrue(outgoingKey != -1);

        out(outgoingKey);
        outgoingKey = -1;

        privilege.stats.numConsoleWrites++;

        if (sendInterruptHandler != null)
            sendInterruptHandler.run();
    }

    public final void writeByte(int value) {
        if (outgoingKey == -1)
            scheduleSendInterrupt();

        outgoingKey = value&0xFF;
    }

    private Privilege privilege = null;

    private Runnable receiveInterrupt;
    private Runnable sendInterrupt;

    private Runnable receiveInterruptHandler = null;
    private Runnable sendInterruptHandler = null;

    private int incomingKey = -1;
    private int outgoingKey = -1;

    private boolean prevCarriageReturn = false;
}
```

```
// PART OF THE MACHINE SIMULATION. DO NOT CHANGE.

package nachos.machine;

import nachos.machine.*;

/**
 * An object that maintains Nachos runtime statistics.
 */
public final class Stats {
    /**
     * Allocate a new statistics object.
     */
    public Stats() {
    }

    /**
     * Print out the statistics in this object.
     */
    public void print() {
        System.out.println("Ticks: total " + totalTicks
                           + ", kernel " + kernelTicks
                           + ", user " + userTicks);
        System.out.println("Disk I/O: reads " + numDiskReads
                           + ", writes " + numDiskWrites);
        System.out.println("Console I/O: reads " + numConsoleReads
                           + ", writes " + numConsoleWrites);
        System.out.println("Paging: page faults " + numPageFaults
                           + ", TLB misses " + numTLBMisses);
        System.out.println("Network I/O: received " + numPacketsReceived
                           + ", sent " + numPacketsSent);
    }

    /**
     * The total amount of simulated time that has passed since Nachos
     * started.
     */
    public long totalTicks = 0;
    /**
     * The total amount of simulated time that Nachos has spent in kernel mode.
     */
    public long kernelTicks = 0;
    /**
     * The total amount of simulated time that Nachos has spent in user mode.
     */
    public long userTicks = 0;

    /** The total number of sectors Nachos has read from the simulated disk.*/
    public int numDiskReads = 0;
    /** The total number of sectors Nachos has written to the simulated disk.*/
    public int numDiskWrites = 0;
    /** The total number of characters Nachos has read from the console. */
    public int numConsoleReads = 0;
    /** The total number of characters Nachos has written to the console. */
    public int numConsoleWrites = 0;
    /** The total number of page faults that have occurred. */
    public int numPageFaults = 0;
    /** The total number of TLB misses that have occurred. */
    public int numTLBMisses = 0;
    /** The total number of packets Nachos has sent to the network. */
    public int numPacketsSent = 0;
    /** The total number of packets Nachos has received from the network. */
    public int numPacketsReceived = 0;

    /**
     * The amount to advance simulated time after each user instructions is
     * executed.
     */
    public static final int UserTick = 1;
    /**
     * The amount to advance simulated time after each interrupt enable.
     */
    public static final int KernelTick = 10;
    /**
     * The amount of simulated time required to rotate the disk 360 degrees.
     */
    public static final int RotationTime = 500;
    /**
     * The amount of simulated time required for the disk to seek.
     */
    public static final int SeekTime = 500;
    /**
     * The amount of simulated time required for the console to handle a
     * character.
     */
    public static final int ConsoleTime = 100;
    /**
     * The amount of simulated time required for the network to handle a
     * packet.
     */
    public static final int NetworkTime = 100;
    /**
     * The mean amount of simulated time between timer interrupts.
     */
    public static final int TimerTicks = 500;
    /**
     * The amount of simulated time required for an elevator to move a floor.
     */
    public static final int ElevatorTicks = 2000;
}
```

```
// PART OF THE MACHINE SIMULATION. DO NOT CHANGE.

package nachos.machine;

import nachos.security.*;
import nachos.threads.*;

import java.io.File;
import java.io.RandomAccessFile;
import java.io.IOException;

/**
 * This class implements a file system that redirects all requests to the host
 * operating system's file system.
 */
public class StubFileSystem implements FileSystem {
    /**
     * Allocate a new stub file system.
     *
     * @param   privilege       encapsulates privileged access to the Nachos
     *                          machine.
     * @param   directory       the root directory of the stub file system.
     */
    public StubFileSystem(Privilege privilege, File directory) {
        this.privilege = privilege;
        this.directory = directory;
    }

    public OpenFile open(String name, boolean truncate) {
        if (!checkName(name))
            return null;

        delay();

        try {
            return new StubOpenFile(name, truncate);
        }
        catch (IOException e) {
            return null;
        }
    }

    public boolean remove(String name) {
        if (!checkName(name))
            return false;

        delay();

        FileRemover fr = new FileRemover(new File(directory, name));
        privilege.doPrivileged(fr);
        return fr.successful;
    }

    private class FileRemover implements Runnable {
        public FileRemover(File f) {
            this.f = f;
        }

        public void run() {
            successful = f.delete();
        }
        public boolean successful = false;
        private File f;
    }

    private void delay() {
        long time = Machine.timer().getTime();
        int amount = 1000;
        ThreadedKernel.alarm.waitUntil(amount);
        Lib.assertTrue(Machine.timer().getTime() >= time+amount);
    }

    private class StubOpenFile extends OpenFileWithPosition {
        StubOpenFile(final String name, final boolean truncate)
            throws IOException {
            super(StubFileSystem.this, name);

            final File f = new File(directory, name);

            if (openCount == maxOpenFiles)
                throw new IOException();

            privilege.doPrivileged(new Runnable() {
                public void run() { getRandomAccessFile(f, truncate); }
            });

            if (file == null)
                throw new IOException();

            open = true;
            openCount++;
        }

        private void getRandomAccessFile(File f, boolean truncate) {
            try {
                if (!truncate && !f.exists())
                    return;

                file = new RandomAccessFile(f, "rw");

                if (truncate)
                    file.setLength(0);
            }
            catch (IOException e) {
            }
        }

        public int read(int pos, byte[] buf, int offset, int length) {
            if (!open)
                return -1;

            try {
                delay();

                file.seek(pos);
                return Math.max(0, file.read(buf, offset, length));
            }
            catch (IOException e) {
                return -1;
            }
        }

        public int write(int pos, byte[] buf, int offset, int length) {
```

```
        if (!open)
            return -1;

        try {
            delay();

            file.seek(pos);
            file.write(buf, offset, length);
            return length;
        }
        catch (IOException e) {
            return -1;
        }
    }

    public int length() {
        try {
            return (int) file.length();
        }
        catch (IOException e) {
            return -1;
        }
    }

    public void close() {
        if (open) {
            open = false;
            openCount--;
        }

        try {
            file.close();
        }
        catch (IOException e) {
        }
    }

    private RandomAccessFile file = null;
    private boolean open = false;
}

private int openCount = 0;
private static final int maxOpenFiles = 16;

private Privilege privilege;
private File directory;

private static boolean checkName(String name) {
    char[] chars = name.toCharArray();

    for (int i=0; i<chars.length; i++) {
        if (chars[i] < 0 || chars[i] >= allowedFileNameCharacters.length)
            return false;
        if (!allowedFileNameCharacters[(int) chars[i]])
            return false;
    }
    return true;
}

private static boolean[] allowedFileNameCharacters = new boolean[0x80];

private static void reject(char c) {
        allowedFileNameCharacters[c] = false;
    }

    private static void allow(char c) {
        allowedFileNameCharacters[c] = true;
    }

    private static void reject(char first, char last) {
        for (char c=first; c<=last; c++)
            allowedFileNameCharacters[c] = false;
    }

    private static void allow(char first, char last) {
        for (char c=first; c<=last; c++)
            allowedFileNameCharacters[c] = true;
    }

    static {
        reject((char) 0x00, (char) 0x7F);

        allow('A', 'Z');
        allow('a', 'z');
        allow('0', '9');

        allow('-');
        allow('_');
        allow('.');
        allow(',');
    }
}
```

```
// PART OF THE MACHINE SIMULATION. DO NOT CHANGE.

package nachos.machine;

import nachos.security.*;
import nachos.threads.KThread;

import java.util.Vector;
import java.security.PrivilegedAction;

/**
 * A TCB simulates the low-level details necessary to create, context-switch,
 * and destroy Nachos threads. Each TCB controls an underlying JVM Thread
 * object.
 *
 * <p>
 * Do not use any methods in <tt>java.lang.Thread</tt>, as they are not
 * compatible with the TCB API. Most <tt>Thread</tt> methods will either crash
 * Nachos or have no useful effect.
 *
 * <p>
 * Do not use the <i>synchronized</i> keyword <b>anywhere</b> in your code.
 * It's against the rules, <i>and</i> it can easily deadlock nachos.
 */
public final class TCB {
    /**
     * Allocate a new TCB.
     */
    public TCB() {
    }

    /**
     * Give the TCB class the necessary privilege to create threads. This is
     * necessary, because unlike other machine classes that need privilege, we
     * want the kernel to be able to create TCB objects on its own.
     *
     * @param   privilege       encapsulates privileged access to the Nachos
     *                          machine.
     */
    public static void givePrivilege(Privilege privilege) {
        TCB.privilege = privilege;
        privilege.tcb = new TCBPrivilege();
    }

    /**
     * Causes the thread represented by this TCB to begin execution. The
     * specified target is run in the thread.
     */
    public void start(Runnable target) {
        /* We will not use synchronization here, because we're assuming that
         * either this is the first call to start(), or we're being called in
         * the context of another TCB. Since we only allow one TCB to run at a
         * time, no synchronization is necessary.
         *
         * The only way this assumption could be broken is if one of our
         * non-Nachos threads used the TCB code.
         */

        /* Make sure this TCB has not already been started. If done is false,
         * then destroy() has not yet set javaThread back to null, so we can
         * use javaThread as a reliable indicator of whether or not start() has
         * already been invoked.
         */
        Lib.assertTrue(javaThread == null && !done);

        /* Make sure there aren't too many running TCBs already. This
         * limitation exists in an effort to prevent wild thread usage.
         */
        Lib.assertTrue(runningThreads.size() < maxThreads);

        isFirstTCB = (currentTCB == null);

        /* Probably unnecessary sanity check: if this is not the first TCB, we
         * make sure that the current thread is bound to the current TCB. This
         * check can only fail if non-Nachos threads invoke start().
         */
        if (!isFirstTCB)
            Lib.assertTrue(currentTCB.javaThread == Thread.currentThread());

        /* At this point all checks are complete, so we go ahead and start the
         * TCB. Whether or not this is the first TCB, it gets added to
         * runningThreads, and we save the target closure.
         */
        runningThreads.add(this);

        this.target = target;

        if (!isFirstTCB) {
            /* If this is not the first TCB, we have to make a new Java thread
             * to run it. Creating Java threads is a privileged operation.
             */
            tcbTarget = new Runnable() {
                    public void run() { threadroot(); }
                };

            privilege.doPrivileged(new Runnable() {
                    public void run() { javaThread = new Thread(tcbTarget); }
                });

            /* The Java thread hasn't yet started, but we need to get it
             * blocking in yield(). We do this by temporarily turning off the
             * current TCB, starting the new Java thread, and waiting for it
             * to wake us up from threadroot(). Once the new TCB wakes us up,
             * it's safe to context switch to the new TCB.
             */
            currentTCB.running = false;

            this.javaThread.start();
            currentTCB.waitForInterrupt();
        }
        else {
            /* This is the first TCB, so we don't need to make a new Java
             * thread to run it; we just steal the current Java thread.
             */
            javaThread = Thread.currentThread();

            /* All we have to do now is invoke threadroot() directly. */
            threadroot();
        }
    }

    /**
     * Return the TCB of the currently running thread.
     */
```

```
public static TCB currentTCB() {
    return currentTCB;
}

/**
 * Context switch between the current TCB and this TCB. This TCB will
 * become the new current TCB. It is acceptable for this TCB to be the
 * current TCB.
 */
public void contextSwitch() {
    /* Probably unnecessary sanity check: we make sure that the current
     * thread is bound to the current TCB. This check can only fail if
     * non-Nachos threads invoke start().
     */
    Lib.assertTrue(currentTCB.javaThread == Thread.currentThread());

    // make sure AutoGrader.runningThread() called associateThread()
    Lib.assertTrue(currentTCB.associated);
    currentTCB.associated = false;

    // can't switch from a TCB to itself
    if (this == currentTCB)
        return;

    /* There are some synchronization concerns here. As soon as we wake up
     * the next thread, we cannot assume anything about static variables,
     * or about any TCB's state. Therefore, before waking up the next
     * thread, we must latch the value of currentTCB, and set its running
     * flag to false (so that, in case we get interrupted before we call
     * yield(), the interrupt will set the running flag and yield() won't
     * block).
     */

    TCB previous = currentTCB;
    previous.running = false;

    this.interrupt();
    previous.yield();
}

/**
 * Destroy this TCB. This TCB must not be in use by the current thread.
 * This TCB must also have been authorized to be destroyed by the
 * autograder.
 */
public void destroy() {
    // make sure the current TCB is correct
    Lib.assertTrue(currentTCB != null &&
               currentTCB.javaThread == Thread.currentThread());
    // can't destroy current thread
    Lib.assertTrue(this != currentTCB);
    // thread must have started but not be destroyed yet
    Lib.assertTrue(javaThread != null && !done);

    // ensure AutoGrader.finishingCurrentThread() called authorizeDestroy()
    Lib.assertTrue(nachosThread == toBeDestroyed);
    toBeDestroyed = null;

    this.done = true;
    currentTCB.running = false;

    this.interrupt();
    currentTCB.waitForInterrupt();

    this.javaThread = null;
}

/**
 * Destroy all TCBs and exit Nachos. Same as <tt>Machine.terminate()</tt>.
 */
public static void die() {
    privilege.exit(0);
}

/**
 * Test if the current JVM thread belongs to a Nachos TCB. The AWT event
 * dispatcher is an example of a non-Nachos thread.
 *
 * @return  <tt>true</tt> if the current JVM thread is a Nachos thread.
 */
public static boolean isNachosThread() {
    return (currentTCB != null &&
            Thread.currentThread() == currentTCB.javaThread);
}

private void threadroot() {
    // this should be running the current thread
    Lib.assertTrue(javaThread == Thread.currentThread());

    if (!isFirstTCB) {
        /* start() is waiting for us to wake it up, signalling that it's OK
         * to context switch to us. We leave the running flag false so that
         * we'll still run if a context switch happens before we go to
         * sleep. All we have to do is wake up the current TCB and then
         * wait to get woken up by contextSwitch() or destroy().
         */

        currentTCB.interrupt();
        this.yield();
    }
    else {
        /* start() called us directly, so we just need to initialize
         * a couple things.
         */

        currentTCB = this;
        running = true;
    }

    try {
        target.run();

        // no way out of here without going throw one of the catch blocks
        Lib.assertNotReached();
    }
    catch (ThreadDeath e) {
        // make sure this TCB is being destroyed properly
        if (!done) {
            System.out.print("\nTCB terminated improperly!\n");
            privilege.exit(1);
        }

        runningThreads.removeElement(this);
        if (runningThreads.isEmpty())
```

```
            privilege.exit(0);
        }
        catch (Throwable e) {
            System.out.print("\n");
            e.printStackTrace();

            runningThreads.removeElement(this);
            if (runningThreads.isEmpty())
                privilege.exit(1);
            else
                die();
        }
    }

    /**
     * Invoked by threadroot() and by contextSwitch() when it is necessary to
     * wait for another TCB to context switch to this TCB. Since this TCB
     * might get destroyed instead, we check the <tt>done</tt> flag after
     * waking up. If it is set, the TCB that woke us up is waiting for an
     * acknowledgement in destroy(). Otherwise, we just set the current TCB to
     * this TCB and return.
     */
    private void yield() {
        waitForInterrupt();

        if (done) {
            currentTCB.interrupt();
            throw new ThreadDeath();
        }

        currentTCB = this;
    }

    /**
     * Waits on the monitor bound to this TCB until its <tt>running</tt> flag
     * is set to <tt>true</tt>. <tt>waitForInterrupt()</tt> is used whenever a
     * TCB needs to go to wait for its turn to run. This includes the ping-pong
     * process of starting and destroying TCBs, as well as in context switching
     * from this TCB to another. We don't rely on <tt>currentTCB</tt>, since it
     * is updated by <tt>contextSwitch()</tt> before we get called.
     */
    private synchronized void waitForInterrupt() {
        while (!running) {
            try { wait(); }
            catch (InterruptedException e) { }
        }
    }

    /**
     * Wake up this TCB by setting its <tt>running</tt> flag to <tt>true</tt>
     * and signalling the monitor bound to it. Used in the ping-pong process of
     * starting and destroying TCBs, as well as in context switching to this
     * TCB.
     */
    private synchronized void interrupt() {
        running = true;
        notify();
    }

    private void associateThread(KThread thread) {
        // make sure AutoGrader.runningThread() gets called only once per
        // context switch
        Lib.assertTrue(!associated);
        associated = true;

        Lib.assertTrue(thread != null);

        if (nachosThread != null)
            Lib.assertTrue(thread == nachosThread);
        else
            nachosThread = thread;
    }

    private static void authorizeDestroy(KThread thread) {
        // make sure AutoGrader.finishingThread() gets called only once per
        // destroy
        Lib.assertTrue(toBeDestroyed == null);
        toBeDestroyed = thread;
    }

    /**
     * The maximum number of started, non-destroyed TCB's that can be in
     * existence.
     */
    public static final int maxThreads = 250;

    /**
     * A reference to the currently running TCB. It is initialized to
     * <tt>null</tt> when the <tt>TCB</tt> class is loaded, and then the first
     * invocation of <tt>start(Runnable)</tt> assigns <tt>currentTCB</tt> a
     * reference to the first TCB. After that, only <tt>yield()</tt> can
     * change <tt>currentTCB</tt> to the current TCB, and only after
     * <tt>waitForInterrupt()</tt> returns.
     *
     * <p>
     * Note that <tt>currentTCB.javaThread</tt> will not be the current thread
     * if the current thread is not bound to a TCB (this includes the threads
     * created for the hardware simulation).
     */
    private static TCB currentTCB = null;

    /**
     * A vector containing all <i>running</i> TCB objects. It is initialized to
     * an empty vector when the <tt>TCB</tt> class is loaded. TCB objects are
     * added only in <tt>start(Runnable)</tt>, which can only be invoked once
     * on each TCB object. TCB objects are removed only in each of the
     * <tt>catch</tt> clauses of <tt>threadroot()</tt>, one of which is always
     * invoked on thread termination. The maximum number of threads in
     * <tt>runningThreads</tt> is limited to <tt>maxThreads</tt> by
     * <tt>start(Runnable)</tt>. If <tt>threadroot()</tt> drops the number of
     * TCB objects in <tt>runningThreads</tt> to zero, Nachos exits, so once
     * the first TCB is created, this vector is basically never empty.
     */
    private static Vector<TCB> runningThreads = new Vector<TCB>();

    private static Privilege privilege;
    private static KThread toBeDestroyed = null;

    /**
     * <tt>true</tt> if and only if this TCB is the first TCB to start, the one
     * started in <tt>Machine.main(String[])</tt>. Initialized by
     * <tt>start(Runnable)</tt>, on the basis of whether <tt>currentTCB</tt>
     * has been initialized.
     */
```

```
    private boolean isFirstTCB;

    /**
     * A reference to the Java thread bound to this TCB. It is initially
     * <tt>null</tt>, assigned to a Java thread in <tt>start(Runnable)</tt>,
     * and set to <tt>null</tt> again in <tt>destroy()</tt>.
     */
    private Thread javaThread = null;

    /**
     * <tt>true</tt> if and only if the Java thread bound to this TCB ought to
     * be running. This is an entirely different condition from membership in
     * <tt>runningThreads</tt>, which contains all TCB objects that have
     * started and have not terminated. <tt>running</tt> is only <tt>true</tt>
     * when the associated Java thread ought to run ASAP. When starting or
     * destroying a TCB, this is temporarily true for a thread other than that
     * of the current TCB.
     */
    private boolean running = false;

    /**
     * Set to <tt>true</tt> by <tt>destroy()</tt>, so that when
     * <tt>waitForInterrupt()</tt> returns in the doomed TCB, <tt>yield()</tt>
     * will know that the current TCB is doomed.
     */
    private boolean done = false;

    private KThread nachosThread = null;
    private boolean associated = false;
    private Runnable target;
    private Runnable tcbTarget;

    private static class TCBPrivilege implements Privilege.TCBPrivilege {
        public void associateThread(KThread thread) {
            Lib.assertTrue(currentTCB != null);
            currentTCB.associateThread(thread);
        }
        public void authorizeDestroy(KThread thread) {
            TCB.authorizeDestroy(thread);
        }
    }
}
```

```
// PART OF THE MACHINE SIMULATION. DO NOT CHANGE.

package nachos.machine;

import nachos.security.*;

/**
 * A hardware timer generates a CPU timer interrupt approximately every 500
 * clock ticks. This means that it can be used for implementing time-slicing,
 * or for having a thread go to sleep for a specific period of time.
 *
 * The <tt>Timer</tt> class emulates a hardware timer by scheduling a timer
 * interrupt to occur every time approximately 500 clock ticks pass. There is
 * a small degree of randomness here, so interrupts do not occur exactly every
 * 500 ticks.
 */
public final class Timer {
    /**
     * Allocate a new timer.
     *
     * @param   privilege       encapsulates privileged access to the Nachos
     *                          machine.
     */
    public Timer(Privilege privilege) {
        System.out.print(" timer");

        this.privilege = privilege;

        timerInterrupt = new Runnable() {
                public void run() { timerInterrupt(); }
            };

        autoGraderInterrupt = new Runnable() {
                public void run() {
                    Machine.autoGrader().timerInterrupt(Timer.this.privilege,
                                                        lastTimerInterrupt);
                }
            };

        scheduleInterrupt();
    }

    /**
     * Set the callback to use as a timer interrupt handler. The timer
     * interrupt handler will be called approximately every 500 clock ticks.
     *
     * @param   handler         the timer interrupt handler.
     */
    public void setInterruptHandler(Runnable handler) {
        this.handler = handler;
    }

    /**
     * Get the current time.
     *
     * @return  the number of clock ticks since Nachos started.
     */
    public long getTime() {
        return privilege.stats.totalTicks;
    }

    private void timerInterrupt() {
        scheduleInterrupt();
        scheduleAutoGraderInterrupt();

        lastTimerInterrupt = getTime();

        if (handler != null)
            handler.run();
    }

    private void scheduleInterrupt() {
        int delay = Stats.TimerTicks;
        delay += Lib.random(delay/10) - (delay/20);

        privilege.interrupt.schedule(delay, "timer", timerInterrupt);
    }

    private void scheduleAutoGraderInterrupt() {
        privilege.interrupt.schedule(1, "timerAG", autoGraderInterrupt);
    }

    private long lastTimerInterrupt;
    private Runnable timerInterrupt;
    private Runnable autoGraderInterrupt;

    private Privilege privilege;
    private Runnable handler = null;
}
```

```
// PART OF THE MACHINE SIMULATION. DO NOT CHANGE.

package nachos.machine;

import nachos.machine.*;

/**
 * A single translation between a virtual page and a physical page.
 */
public final class TranslationEntry {
    /**
     * Allocate a new invalid translation entry.
     */
    public TranslationEntry() {
        valid = false;
    }

    /**
     * Allocate a new translation entry with the specified initial state.
     *
     * @param   vpn             the virtual page numben.
     * @param   ppn             the physical page number.
     * @param   valid           the valid bit.
     * @param   readOnly        the read-only bit.
     * @param   used            the used bit.
     * @param   dirty           the dirty bit.
     */
    public TranslationEntry(int vpn, int ppn, boolean valid, boolean readOnly,
                            boolean used, boolean dirty) {
        this.vpn = vpn;
        this.ppn = ppn;
        this.valid = valid;
        this.readOnly = readOnly;
        this.used = used;
        this.dirty = dirty;
    }

    /**
     * Allocate a new translation entry, copying the contents of an existing
     * one.
     *
     * @param   entry   the translation entry to copy.
     */
    public TranslationEntry(TranslationEntry entry) {
        vpn = entry.vpn;
        ppn = entry.ppn;
        valid = entry.valid;
        readOnly = entry.readOnly;
        used = entry.used;
        dirty = entry.dirty;
    }

    /** The virtual page number. */
    public int vpn;

    /** The physical page number. */
    public int ppn;

    /**
     * If this flag is <tt>false</tt>, this translation entry is ignored.
     */
    public boolean valid;

    /**
     * If this flag is <tt>true</tt>, the user pprogram is not allowed to
     * modify the contents of this virtual page.
     */
    public boolean readOnly;

    /**
     * This flag is set to <tt>true</tt> every time the page is read or written
     * by a user program.
     */
    public boolean used;

    /**
     * This flag is set to <tt>true</tt> every time the page is written by a
     * user program.
     */
    public boolean dirty;
}
```

```
package nachos.network;

import nachos.machine.*;

/**
 * A mail message. Includes a packet header, a mail header, and the actual
 * payload.
 *
 * @see nachos.machine.Packet
 */
public class MailMessage {
    /**
     * Allocate a new mail message to be sent, using the specified parameters.
     *
     * @param   dstLink         the destination link address.
     * @param   dstPort         the destination port.
     * @param   srcLink         the source link address.
     * @param   srcPort         the source port.
     * @param   contents        the contents of the packet.
     */
    public MailMessage(int dstLink, int dstPort, int srcLink, int srcPort,
                       byte[] contents) throws MalformedPacketException {
        // make sure the paramters are valid
        if (dstPort < 0 || dstPort >= portLimit ||
            srcPort < 0 || srcPort >= portLimit ||
            contents.length > maxContentsLength)
            throw new MalformedPacketException();

        this.dstPort = (byte) dstPort;
        this.srcPort = (byte) srcPort;
        this.contents = contents;

        byte[] packetContents = new byte[headerLength + contents.length];

        packetContents[0] = (byte) dstPort;
        packetContents[1] = (byte) srcPort;

        System.arraycopy(contents, 0, packetContents, headerLength,
                         contents.length);

        packet = new Packet(dstLink, srcLink, packetContents);
    }

    /**
     * Allocate a new mail message using the specified packet from the network.
     *
     * @param   packet  the packet containg the mail message.
     */
    public MailMessage(Packet packet) throws MalformedPacketException {
        this.packet = packet;

        // make sure we have a valid header
        if (packet.contents.length < headerLength ||
            packet.contents[0] < 0 || packet.contents[0] >= portLimit ||
            packet.contents[1] < 0 || packet.contents[1] >= portLimit)
            throw new MalformedPacketException();

        dstPort = packet.contents[0];
        srcPort = packet.contents[1];

        contents = new byte[packet.contents.length - headerLength];
        System.arraycopy(packet.contents, headerLength, contents, 0,
                         contents.length);
    }

    /**
     * Return a string representation of the message headers.
     */
    public String toString() {
        return "from (" + packet.srcLink + ":" + srcPort +
            ") to (" + packet.dstLink + ":" + dstPort +
            "), " + contents.length + " bytes";
    }

    /** This message, as a packet that can be sent through a network link. */
    public Packet packet;
    /** The port used by this message on the destination machine. */
    public int dstPort;
    /** The port used by this message on the source machine. */
    public int srcPort;
    /** The contents of this message, excluding the mail message header. */
    public byte[] contents;

    /**
     * The number of bytes in a mail header. The header is formatted as
     * follows:
     *
     * <table>
     * <tr><td>offset</td><td>size</td><td>value</td></tr>
     * <tr><td>0</td><td>1</td><td>destination port</td></tr>
     * <tr><td>1</td><td>1</td><td>source port</td></tr>
     * </table>
     */
    public static final int headerLength = 2;

    /** Maximum payload (real data) that can be included in a single mesage. */
    public static final int maxContentsLength =
        Packet.maxContentsLength - headerLength;

    /**
     * The upper limit on mail ports. All ports fall between <tt>0</tt> and
     * <tt>portLimit - 1</tt>.
     */
    public static final int portLimit = 128;
}
```

```
package nachos.network;

import nachos.machine.*;
import nachos.threads.*;
import nachos.userprog.*;
import nachos.vm.*;
import nachos.network.*;

/**
 * A kernel with network support.
 */
public class NetKernel extends VMKernel {
    /**
     * Allocate a new networking kernel.
     */
    public NetKernel() {
        super();
    }

    /**
     * Initialize this kernel.
     */
    public void initialize(String[] args) {
        super.initialize(args);

        postOffice = new PostOffice();
    }

    /**
     * Test the network. Create a server thread that listens for pings on port
     * 1 and sends replies. Then ping one or two hosts. Note that this test
     * assumes that the network is reliable (i.e. that the network's
     * reliability is 1.0).
     */
    public void selfTest() {
        super.selfTest();

        KThread serverThread = new KThread(new Runnable() {
                public void run() { pingServer(); }
            });

        serverThread.fork();

        System.out.println("Press any key to start the network test...");
        console.readByte(true);

        int local = Machine.networkLink().getLinkAddress();

        // ping this machine first
        ping(local);

        // if we're 0 or 1, ping the opposite
        if (local <= 1)
            ping(1-local);
    }

    private void ping(int dstLink) {
        int srcLink = Machine.networkLink().getLinkAddress();

        System.out.println("PING " + dstLink + " from " + srcLink);

        long startTime = Machine.timer().getTime();

        MailMessage ping;

        try {
            ping = new MailMessage(dstLink, 1,
                                   Machine.networkLink().getLinkAddress(), 0,
                                   new byte[0]);
        }
        catch (MalformedPacketException e) {
            Lib.assertNotReached();
            return;
        }

        postOffice.send(ping);

        MailMessage ack = postOffice.receive(0);

        long endTime = Machine.timer().getTime();

        System.out.println("time=" + (endTime-startTime) + " ticks");
    }

    private void pingServer() {
        while (true) {
            MailMessage ping = postOffice.receive(1);

            MailMessage ack;

            try {
                ack = new MailMessage(ping.packet.srcLink, ping.srcPort,
                                      ping.packet.dstLink, ping.dstPort,
                                      ping.contents);
            }
            catch (MalformedPacketException e) {
                // should never happen...
                continue;
            }

            postOffice.send(ack);
        }
    }

    /**
     * Start running user programs.
     */
    public void run() {
        super.run();
    }

    /**
     * Terminate this kernel. Never returns.
     */
    public void terminate() {
        super.terminate();
    }

    private PostOffice postOffice;

    // dummy variables to make javac smarter
    private static NetProcess dummy1 = null;
}
```

```
package nachos.network;

import nachos.machine.*;
import nachos.threads.*;
import nachos.userprog.*;
import nachos.vm.*;

/**
 * A <tt>VMProcess</tt> that supports networking syscalls.
 */
public class NetProcess extends VMProcess {
    /**
     * Allocate a new process.
     */
    public NetProcess() {
        super();
    }

    private static final int
        syscallConnect = 11,
        syscallAccept = 12;

    /**
     * Handle a syscall exception. Called by <tt>handleException()</tt>. The
     * <i>syscall</i> argument identifies which syscall the user executed:
     *
     * <table>
     * <tr><td>syscall#</td><td>syscall prototype</td></tr>
     * <tr><td>11</td><td><tt>int  connect(int host, int port);</tt></td></tr>
     * <tr><td>12</td><td><tt>int  accept(int port);</tt></td></tr>
     * </table>
     *
     * @param   syscall the syscall number.
     * @param   a0      the first syscall argument.
     * @param   a1      the second syscall argument.
     * @param   a2      the third syscall argument.
     * @param   a3      the fourth syscall argument.
     * @return  the value to be returned to the user.
     */
    public int handleSyscall(int syscall, int a0, int a1, int a2, int a3) {
        switch (syscall) {
        default:
            return super.handleSyscall(syscall, a0, a1, a2, a3);
        }
    }
}
```

```
package nachos.network;

import nachos.machine.*;
import nachos.threads.*;

/**
 * A collection of message queues, one for each local port. A
 * <tt>PostOffice</tt> interacts directly with the network hardware. Because
 * of the network hardware, we are guaranteed that messages will never be
 * corrupted, but they might get lost.
 *
 * <p>
 * The post office uses a "postal worker" thread to wait for messages to arrive
 * from the network and to place them in the appropriate queues. This cannot
 * be done in the receive interrupt handler because each queue (implemented
 * with a <tt>SynchList</tt>) is protected by a lock.
 */
public class PostOffice {
    /**
     * Allocate a new post office, using an array of <tt>SynchList</tt>s.
     * Register the interrupt handlers with the network hardware and start the
     * "postal worker" thread.
     */
    public PostOffice() {
        messageReceived = new Semaphore(0);
        messageSent = new Semaphore(0);
        sendLock = new Lock();

        queues = new SynchList[MailMessage.portLimit];
        for (int i=0; i<queues.length; i++)
            queues[i] = new SynchList();

        Runnable receiveHandler = new Runnable() {
            public void run() { receiveInterrupt(); }
        };
        Runnable sendHandler = new Runnable() {
            public void run() { sendInterrupt(); }
        };
        Machine.networkLink().setInterruptHandlers(receiveHandler,
                                                   sendHandler);

        KThread t = new KThread(new Runnable() {
                public void run() { postalDelivery(); }
            });

        t.fork();
    }

    /**
     * Retrieve a message on the specified port, waiting if necessary.
     *
     * @param   port    the port on which to wait for a message.
     *
     * @return  the message received.
     */
    public MailMessage receive(int port) {
        Lib.assertTrue(port >= 0 && port < queues.length);

        Lib.debug(dbgNet, "waiting for mail on port " + port);

        MailMessage mail = (MailMessage) queues[port].removeFirst();

        if (Lib.test(dbgNet))
            System.out.println("got mail on port " + port + ": " + mail);

        return mail;
    }

    /**
     * Wait for incoming messages, and then put them in the correct mailbox.
     */
    private void postalDelivery() {
        while (true) {
            messageReceived.P();

            Packet p = Machine.networkLink().receive();

            MailMessage mail;

            try {
                mail = new MailMessage(p);
            }
            catch (MalformedPacketException e) {
                continue;
            }

            if (Lib.test(dbgNet))
                System.out.println("delivering mail to port " + mail.dstPort
                                   + ": " + mail);

            // atomically add message to the mailbox and wake a waiting thread
            queues[mail.dstPort].add(mail);
        }
    }

    /**
     * Called when a packet has arrived and can be dequeued from the network
     * link.
     */
    private void receiveInterrupt() {
        messageReceived.V();
    }

    /**
     * Send a message to a mailbox on a remote machine.
     */
    public void send(MailMessage mail) {
        if (Lib.test(dbgNet))
            System.out.println("sending mail: " + mail);

        sendLock.acquire();

        Machine.networkLink().send(mail.packet);
        messageSent.P();

        sendLock.release();
    }

    /**
     * Called when a packet has been sent and another can be queued to the
     * network link. Note that this is called even if the previous packet was
     * dropped.
     */
    private void sendInterrupt() {
```

```
        messageSent.V();
    }

    private SynchList[] queues;
    private Semaphore messageReceived;  // V'd when a message can be dequeued
    private Semaphore messageSent;      // V'd when a message can be queued
    private Lock sendLock;

    private static final char dbgNet = 'n';
}
```

```
DIRS = threads machine security ag

include ../Makefile
```

```
Machine.stubFileSystem = false
Machine.processor = false
Machine.console = false
Machine.disk = false
Machine.bank = false
Machine.networkLink = false
ElevatorBank.allowElevatorGUI = true
NachosSecurityManager.fullySecure = false
ThreadedKernel.scheduler = nachos.threads.RoundRobinScheduler #nachos.threads.Priority
Scheduler
Kernel.kernel = nachos.threads.ThreadedKernel
```

```
DIRS = userprog threads machine security ag

include ../Makefile
```

```
Machine.stubFileSystem = true
Machine.processor = true
Machine.console = true
Machine.disk = false
Machine.bank = false
Machine.networkLink = false
Processor.usingTLB = false
Processor.numPhysPages = 64
ElevatorBank.allowElevatorGUI = false
NachosSecurityManager.fullySecure = false
ThreadedKernel.scheduler = nachos.threads.RoundRobinScheduler #nachos.threads.LotteryS
cheduler
Kernel.shellProgram = halt.coff #sh.coff
Kernel.processClassName = nachos.userprog.UserProcess
Kernel.kernel = nachos.userprog.UserKernel
```

```
DIRS = vm userprog threads machine security ag

include ../Makefile
```

```
Machine.stubFileSystem = true
Machine.processor = true
Machine.console = true
Machine.disk = false
Machine.bank = false
Machine.networkLink = false
Processor.usingTLB = true
Processor.numPhysPages = 16
ElevatorBank.allowElevatorGUI = false
NachosSecurityManager.fullySecure = false
ThreadedKernel.scheduler = nachos.threads.RoundRobinScheduler
Kernel.shellProgram = sh.coff
Kernel.processClassName = nachos.vm.VMProcess
Kernel.kernel = nachos.vm.VMKernel
```

```
DIRS = network vm userprog threads machine security ag

include ../Makefile
```

```
Machine.stubFileSystem = true
Machine.processor = true
Machine.console = true
Machine.disk = false
Machine.bank = false
Machine.networkLink = true
Processor.usingTLB = true
Processor.variableTLB = true
Processor.numPhysPages = 16
ElevatorBank.allowElevatorGUI = false
NetworkLink.reliability = 1.0                   # use 0.9 when you're ready
NachosSecurityManager.fullySecure = false
ThreadedKernel.scheduler = nachos.threads.RoundRobinScheduler
Kernel.shellProgram = sh.coff
Kernel.processClassName = nachos.network.NetProcess
Kernel.kernel = nachos.network.NetKernel
```

```
// PART OF THE MACHINE SIMULATION. DO NOT CHANGE.

package nachos.security;

import nachos.machine.*;

import java.io.File;
import java.security.Permission;
import java.io.FilePermission;
import java.util.PropertyPermission;
import java.net.NetPermission;
import java.awt.AWTPermission;
import java.security.PrivilegedAction;
import java.security.PrivilegedExceptionAction;
import java.security.PrivilegedActionException;

/**
 * Protects the environment from malicious Nachos code.
 */
public class NachosSecurityManager extends SecurityManager {
    /**
     * Allocate a new Nachos security manager.
     *
     * @param   testDirectory   the directory usable by the stub file system.
     */
    public NachosSecurityManager(File testDirectory) {
        this.testDirectory = testDirectory;

        fullySecure = Config.getBoolean("NachosSecurityManager.fullySecure");
    }

    /**
     * Return a privilege object for this security manager. This security
     * manager must not be the active security manager.
     *
     * @return  a privilege object for this security manager.
     */
    public Privilege getPrivilege() {
        Lib.assertTrue(this != System.getSecurityManager());

        return new PrivilegeProvider();
    }

    /**
     * Install this security manager.
     */
    public void enable() {
        Lib.assertTrue(this != System.getSecurityManager());

        doPrivileged(new Runnable() {
            public void run() {
                System.setSecurityManager(NachosSecurityManager.this);
            }
        });
    }

    private class PrivilegeProvider extends Privilege {
        public void doPrivileged(Runnable action) {
            NachosSecurityManager.this.doPrivileged(action);
        }

        public Object doPrivileged(PrivilegedAction action) {
            return NachosSecurityManager.this.doPrivileged(action);
        }

        public Object doPrivileged(PrivilegedExceptionAction action)
            throws PrivilegedActionException {
            return NachosSecurityManager.this.doPrivileged(action);
        }

        public void exit(int exitStatus) {
            invokeExitNotificationHandlers();
            NachosSecurityManager.this.exit(exitStatus);
        }
    }

    private void enablePrivilege() {
        if (privilegeCount == 0) {
            Lib.assertTrue(privileged == null);
            privileged = Thread.currentThread();
            privilegeCount++;
        }
        else {
            Lib.assertTrue(privileged == Thread.currentThread());
            privilegeCount++;
        }
    }

    private void rethrow(Throwable e) {
        disablePrivilege();

        if (e instanceof RuntimeException)
            throw (RuntimeException) e;
        else if (e instanceof Error)
            throw (Error) e;
        else
            Lib.assertNotReached();
    }

    private void disablePrivilege() {
        Lib.assertTrue(privileged != null && privilegeCount > 0);
        privilegeCount--;
        if (privilegeCount == 0)
            privileged = null;
    }

    private void forcePrivilege() {
        privileged = Thread.currentThread();
        privilegeCount = 1;
    }

    private void exit(int exitStatus) {
        forcePrivilege();
        System.exit(exitStatus);
    }

    private boolean isPrivileged() {
        // the autograder does not allow non-Nachos threads to be created, so..
        if (!TCB.isNachosThread())
            return true;

        return (privileged == Thread.currentThread());
    }
```

```
    private void doPrivileged(final Runnable action) {
        doPrivileged(new PrivilegedAction() {
            public Object run() { action.run(); return null; }
        });
    }

    private Object doPrivileged(PrivilegedAction action) {
        Object result = null;
        enablePrivilege();
        try {
            result = action.run();
        }
        catch (Throwable e) {
            rethrow(e);
        }
        disablePrivilege();
        return result;
    }

    private Object doPrivileged(PrivilegedExceptionAction action)
        throws PrivilegedActionException {
        Object result = null;
        enablePrivilege();
        try {
            result = action.run();
        }
        catch (Exception e) {
            throw new PrivilegedActionException(e);
        }
        catch (Throwable e) {
            rethrow(e);
        }
        disablePrivilege();
        return result;
    }

    private void no() {
        throw new SecurityException();
    }

    private void no(Permission perm) {
        System.err.println("\n\nLacked permission: " + perm);
        throw new SecurityException();
    }

    /**
     * Check the specified permission. Some operations are permissible while
     * not grading. These operations are regulated here.
     *
     * @param   perm    the permission to check.
     */
    public void checkPermission(Permission perm) {
        String name = perm.getName();

        // some permissions are strictly forbidden
        if (perm instanceof RuntimePermission) {
            // no creating class loaders
            if (name.equals("createClassLoader"))
                no(perm);
        }

        // allow the AWT mess when not grading
        if (!fullySecure) {
            if (perm instanceof NetPermission) {
                // might be needed to load awt stuff
                if (name.equals("specifyStreamHandler"))
                    return;
            }

            if (perm instanceof RuntimePermission) {
                // might need to load libawt
                if (name.startsWith("loadLibrary.")) {
                    String lib = name.substring("loadLibrary.".length());
                    if (lib.equals("awt")) {
                        Lib.debug(dbgSecurity, "\tdynamically linking " + lib);
                        return;
                    }
                }
            }

            if (perm instanceof AWTPermission) {
                // permit AWT stuff
                if (name.equals("accessEventQueue"))
                    return;
            }
        }

        // some are always allowed
        if (perm instanceof PropertyPermission) {
            // allowed to read properties
            if (perm.getActions().equals("read"))
                return;
        }

        // some require some more checking
        if (perm instanceof FilePermission) {
            if (perm.getActions().equals("read")) {
                // the test directory can only be read with privilege
                if (isPrivileged())
                    return;

                enablePrivilege();

                // not allowed to read test directory directly w/out privilege
                try {
                    File f = new File(name);
                    if (f.isFile()) {
                        File p = f.getParentFile();
                        if (p != null) {
                            if (p.equals(testDirectory))
                                no(perm);
                        }
                    }
                }
                catch (Throwable e) {
                    rethrow(e);
                }

                disablePrivilege();
                return;
            }
            else if (perm.getActions().equals("write") ||
                     perm.getActions().equals("delete")) {
                // only allowed to write test diretory, and only with privilege
```

```
            verifyPrivilege();

            try {
                File f = new File(name);
                if (f.isFile()) {
                    File p = f.getParentFile();
                    if (p != null && p.equals(testDirectory))
                        return;
                }
            }
            catch (Throwable e) {
                no(perm);
            }
        }
        else if (perm.getActions().equals("execute")) {
            // only allowed to execute with privilege, and if there's a net
            verifyPrivilege();

            if (Machine.networkLink() == null)
                no(perm);
        }
        else {
            no(perm);
        }
    }

    // default to requiring privilege
    verifyPrivilege(perm);
}

/**
 * Called by the <tt>java.lang.Thread</tt> constructor to determine a
 * thread group for a child thread of the current thread. The caller must
 * be privileged in order to successfully create the thread.
 *
 * @return  a thread group for the new thread, or <tt>null</tt> to use the
 *          current thread's thread group.
 */
public ThreadGroup getThreadGroup() {
    verifyPrivilege();
    return null;
}

/**
 * Verify that the caller is privileged.
 */
public void verifyPrivilege() {
    if (!isPrivileged())
        no();
}

/**
 * Verify that the caller is privileged, so as to check the specified
 * permission.
 *
 * @param   perm    the permission being checked.
 */
public void verifyPrivilege(Permission perm) {
    if (!isPrivileged())
        no(perm);
}
```

```
    private File testDirectory;
    private boolean fullySecure;

    private Thread privileged = null;
    private int privilegeCount = 0;

    private static final char dbgSecurity = 'S';
}
```

```
// PART OF THE MACHINE SIMULATION. DO NOT CHANGE.

package nachos.security;

import nachos.machine.*;
import nachos.threads.KThread;

import java.util.LinkedList;
import java.util.Iterator;
import java.security.PrivilegedAction;
import java.security.PrivilegedExceptionAction;
import java.security.PrivilegedActionException;

/**
 * A capability that allows privileged access to the Nachos machine.
 *
 * <p>
 * Some privileged operations are guarded by the Nachos security manager:
 * <ol>
 * <li>creating threads
 * <li>writing/deleting files in the test directory
 * <li>exit with specific status code
 * </ol>
 * These operations can only be performed through <tt>doPrivileged()</tt>.
 *
 * <p>
 * Some privileged operations require a capability:
 * <ol>
 * <li>scheduling interrupts
 * <li>advancing the simulated time
 * <li>accessing machine statistics
 * <li>installing a console
 * <li>flushing the simulated processor's pipeline
 * <li>approving TCB operations
 * </ol>
 * These operations can be directly performed using a <tt>Privilege</tt>
 * object.
 *
 * <p>
 * The Nachos kernel should <i>never</i> be able to directly perform any of
 * these privileged operations. If you have discovered a loophole somewhere,
 * notify someone.
 */
public abstract class Privilege {
    /**
     * Allocate a new <tt>Privilege</tt> object. Note that this object in
     * itself does not encapsulate privileged access until the machine devices
     * fill it in.
     */
    public Privilege() {
    }

    /**
     * Perform the specified action with privilege.
     *
     * @param   action  the action to perform.
     */
    public abstract void doPrivileged(Runnable action);

    /**
     * Perform the specified <tt>PrivilegedAction</tt> with privilege.
     *
     * @param   action  the action to perform.
     * @return  the return value of the action.
     */
    public abstract Object doPrivileged(PrivilegedAction action);

    /**
     * Perform the specified <tt>PrivilegedExceptionAction</tt> with privilege.
     *
     * @param   action  the action to perform.
     * @return  the return value of the action.
     */
    public abstract Object doPrivileged(PrivilegedExceptionAction action)
        throws PrivilegedActionException;

    /**
     * Exit Nachos with the specified status.
     *
     * @param   exitStatus      the exit status of the Nachos process.
     */
    public abstract void exit(int exitStatus);

    /**
     * Add an <tt>exit()</tt> notification handler. The handler will be invoked
     * by exit().
     *
     * @param   handler the notification handler.
     */
    public void addExitNotificationHandler(Runnable handler) {
        exitNotificationHandlers.add(handler);
    }

    /**
     * Invoke each <tt>exit()</tt> notification handler added by
     * <tt>addExitNotificationHandler()</tt>. Called by <tt>exit()</tt>.
     */
    protected void invokeExitNotificationHandlers() {
        for (Iterator i=exitNotificationHandlers.iterator(); i.hasNext(); ) {
            try {
                ((Runnable) i.next()).run();
            }
            catch (Throwable e) {
                System.out.println("exit() notification handler failed");
            }
        }
    }

    private LinkedList<Runnable> exitNotificationHandlers =
        new LinkedList<Runnable>();

    /** Nachos runtime statistics. */
    public Stats stats = null;

    /** Provides access to some private <tt>Machine</tt> methods. */
    public MachinePrivilege machine = null;
    /** Provides access to some private <tt>Interrupt</tt> methods. */
    public InterruptPrivilege interrupt = null;
    /** Provides access to some private <tt>Processor</tt> methods. */
    public ProcessorPrivilege processor = null;
    /** Provides access to some private <tt>TCB</tt> methods. */
    public TCBPrivilege tcb = null;

    /**
```

```
 * An interface that provides access to some private <tt>Machine</tt>
 * methods.
 */
public interface MachinePrivilege {
    /**
     * Install a hardware console.
     *
     * @param       console the new hardware console.
     */
    public void setConsole(SerialConsole console);
}

/**
 * An interface that provides access to some private <tt>Interrupt</tt>
 * methods.
 */
public interface InterruptPrivilege {
    /**
     * Schedule an interrupt to occur at some time in the future.
     *
     * @param       when    the number of ticks until the interrupt should
     *                      occur.
     * @param       type    a name for the type of interrupt being
     *                      scheduled.
     * @param       handler the interrupt handler to call.
     */
    public void schedule(long when, String type, Runnable handler);

    /**
     * Advance the simulated time.
     *
     * @param inKernelMode  <tt>true</tt> if the current thread is running kernel
     *              code, <tt>false</tt> if the current thread is running
     *              MIPS user code.
     */
    public void tick(boolean inKernelMode);
}

/**
 * An interface that provides access to some private <tt>Processor</tt>
 * methods.
 */
public interface ProcessorPrivilege {
    /**
     * Flush the processor pipeline in preparation for switching to kernel
     * mode.
     */
    public void flushPipe();
}

/**
 * An interface that provides access to some private <tt>TCB</tt> methods.
 */
public interface TCBPrivilege {
    /**
     * Associate the current TCB with the specified <tt>KThread</tt>.
     * <tt>AutoGrader.runningThread()</tt> <i>must</i> call this method
     * before returning.
     *
     * @param       thread  the current thread.
     */
    public void associateThread(KThread thread);

    /**
     * Authorize the TCB associated with the specified thread to be
     * destroyed.
     *
     * @param       thread  the thread whose TCB is about to be destroyed.
     */
    public void authorizeDestroy(KThread thread);
}
}
```

```
# GNU Makefile for building user programs to run on top of Nachos
#
# Things to be aware of:
#
#    The value of the ARCHDIR environment variable must be set before using
#       this makefile. If you are using an instructional machine, this should
#       be automatic. However, if you are not using an instructional machine,
#       you need to point ARCHDIR at the cross-compiler directory, e.g.
#               setenv ARCHDIR ../mips-x86.win32-xgcc

# you need to point to the right executables
GCCDIR = $(ARCHDIR)/mips-

ASFLAGS = -mips1
CPPFLAGS =
CFLAGS = -O2 -B$(GCCDIR) -G 0 -Wa,-mips1 -nostdlib -ffreestanding
LDFLAGS = -s -T script -N -warn-common -warn-constructors -warn-multiple-gp

CC = $(GCCDIR)gcc
AS = $(GCCDIR)as
LD = $(GCCDIR)ld
CPP = $(GCCDIR)cpp
AR = $(GCCDIR)ar
RANLIB = $(GCCDIR)ranlib

STDLIB_H = stdio.h stdlib.h ag.h
STDLIB_C = stdio.c stdlib.c
STDLIB_O = start.o stdio.o stdlib.o

LIB = assert atoi printf readline stdio strncmp strcat strcmp strcpy strlen memcpy mem
set
NLIB = libnachos.a

TARGETS = halt sh matmult sort echo cat cp mv rm #chat chatserver

.SECONDARY: $(patsubst %.c,%.o,$(wildcard *.c))

all: $(patsubst %,%.coff,$(TARGETS))

ag: grade-file.coff grade-exec.coff grade-mini.coff grade-dumb.coff

clean:
        rm -f strt.s *.o *.coff $(NLIB)

agclean: clean
        rm -f f1-* f2-*

$(NLIB): $(patsubst %,$(NLIB)(%.o),$(LIB)) start.o
        $(RANLIB) $(NLIB)

start.o: start.s syscall.h
        $(CPP) $(CPPFLAGS) start.s > strt.s
        $(AS) $(ASFLAGS) -o start.o strt.s
        rm strt.s

%.o: %.c *.h
        $(CC) $(CFLAGS) -c $<

%.coff: %.o $(NLIB)
        $(LD) $(LDFLAGS) -o $@ $< start.o -lnachos
```

```
#include "stdio.h"
#include "stdlib.h"

void __assert(char* file, int line) {
  printf("\nAssertion failed: line %d file %s\n", line, file);
  exit(1);
}
```

```
#include "stdlib.h"

int atoi(const char *s) {
  int result=0, sign=1;

  if (*s == -1) {
    sign = -1;
    s++;
  }

  while (*s >= '0' && *s <= '9')
    result = result*10 + (*(s++)-'0');

  return result*sign;
}
```

```
#include "syscall.h"
#include "stdio.h"
#include "stdlib.h"

#define BUFSIZE 1024

char buf[BUFSIZE];

int main(int argc, char** argv)
{
  int fd, amount;

  if (argc!=2) {
    printf("Usage: cat <file>\n");
    return 1;
  }

  fd = open(argv[1]);
  if (fd==-1) {
    printf("Unable to open %s\n", argv[1]);
    return 1;
  }

  while ((amount = read(fd, buf, BUFSIZE))>0) {
    write(1, buf, amount);
  }

  close(fd);

  return 0;
}
```

```
#include "syscall.h"
#include "stdio.h"
#include "stdlib.h"

#define BUFSIZE 1024

char buf[BUFSIZE];

int main(int argc, char** argv)
{
  int src, dst, amount;

  if (argc!=3) {
    printf("Usage: cp <src> <dst>\n");
    return 1;
  }

  src = open(argv[1]);
  if (src==-1) {
    printf("Unable to open %s\n", argv[1]);
    return 1;
  }

  creat(argv[2]);
  dst = open(argv[2]);
  if (dst==-1) {
    printf("Unable to create %s\n", argv[2]);
    return 1;
  }

  while ((amount = read(src, buf, BUFSIZE))>0) {
    write(dst, buf, amount);
  }

  close(src);
  close(dst);

  return 0;
}
```

```
#include "stdio.h"
#include "stdlib.h"

int main(int argc, char** argv)
{
  int i;

  printf("%d arguments\n", argc);

  for (i=0; i<argc; i++)
    printf("arg %d: %s\n", i, argv[i]);

  return 0;
}
```

```
/* halt.c
 *      Simple program to test whether running a user program works.
 *
 *      Just do a "syscall" that shuts down the OS.
 *
 *      NOTE: for some reason, user programs with global data structures
 *      sometimes haven't worked in the Nachos environment.  So be careful
 *      out there!  One option is to allocate data structures as
 *      automatics within a procedure, but if you do this, you have to
 *      be careful to allocate a big enough stack to hold the automatics!
 */

#include "syscall.h"

int
main()
{
    halt();
    /* not reached */
}
```

```
/* matmult.c
 *    Test program to do matrix multiplication on large arrays.
 *
 *    Intended to stress virtual memory system. Should return 7220 if Dim==20
 */

#include "syscall.h"

#define Dim     20      /* sum total of the arrays doesn't fit in
                         * physical memory
                         */

int A[Dim][Dim];
int B[Dim][Dim];
int C[Dim][Dim];

int
main()
{
    int i, j, k;

    for (i = 0; i < Dim; i++)           /* first initialize the matrices */
        for (j = 0; j < Dim; j++) {
             A[i][j] = i;
             B[i][j] = j;
             C[i][j] = 0;
        }

    for (i = 0; i < Dim; i++)           /* then multiply them together */
        for (j = 0; j < Dim; j++)
            for (k = 0; k < Dim; k++)
                 C[i][j] += A[i][k] * B[k][j];

    printf("C[%d][%d] = %d\n", Dim-1, Dim-1, C[Dim-1][Dim-1]);
    return (C[Dim-1][Dim-1]);           /* and then we're done */
}
```

```
#include "stdlib.h"

void *memcpy(void *s1, const void *s2, unsigned n) {
  int i;

  for (i=0; i<n; i++)
    ((char*)s1)[i] = ((char*)s2)[i];

  return s1;
}
```

```
#include "stdlib.h"

void *memset(void *s, int c, unsigned int n) {
  int i;

  for (i=0; i<n; i++)
    ((char*)s)[i] = (char) c;

  return s;
}
```

```
#include "syscall.h"
#include "stdio.h"
#include "stdlib.h"

#define BUFSIZE 1024

char buf[BUFSIZE];

int main(int argc, char** argv)
{
  int src, dst, amount;

  if (argc!=3) {
    printf("Usage: cp <src> <dst>\n");
    return 1;
  }

  src = open(argv[1]);
  if (src==-1) {
    printf("Open to open %s\n", argv[1]);
    return 1;
  }

  creat(argv[2]);
  dst = open(argv[2]);
  if (dst==-1) {
    printf("Unable to create %s\n", argv[2]);
    return 1;
  }

  while ((amount = read(src, buf, BUFSIZE))>0) {
    write(dst, buf, amount);
  }

  close(src);
  close(dst);
  unlink(argv[1]);

  return 0;
}
```

```
#include "stdio.h"
#include "stdlib.h"

static char digittoascii(unsigned n, int uppercase) {
    assert(n<36);

    if (n<=9)
        return '0'+n;
    else if (uppercase)
        return 'A'+n-10;
    else
        return 'a'+n-10;
}

static int charprint(char **s, char c) {
    *((*s)++) = c;

    return 1;
}

static int mcharprint(char **s, char* chars, int length) {
    memcpy(*s, chars, length);
    *s += length;

    return length;
}

static int integerprint(char **s, int n, unsigned base, int min, int zpad, int upper)
{
    char buf[32];
    int i=32, digit, len=0;

    assert(base>=2 && base < 36);

    if (min>32)
        min=32;

    if (n==0) {
        for (i=1; i<min; i++)
            len += charprint(s, zpad ? '0' : ' ');

        len += charprint(s, '0');
        return len;
    }

    if (n<0) {
        len += charprint(s, '-');
        n *= -1;
    }

    while (n!=0) {
        digit = n%base;
        n /= base;

        if (digit<0)
            digit *= -1;

        buf[--i] = digittoascii(digit, upper);
    }

    while (i>32-min)
        buf[--i] = zpad ? '0' : ' ';

    len += mcharprint(s, &buf[i], 32-i);
    return len;
}

static int stringprint(char **s, char *string) {
    return mcharprint(s, string, strlen(string));
}

static int _vsprintf(char *s, char *format, va_list ap) {
    int min,zpad,len=0,regular=0;
    char *temp;

    /* process format string */
    while (*format != 0) {
        /* if switch, process */
        if (*format == '%') {
            if (regular > 0) {
                len += mcharprint(&s, format-regular, regular);
                regular = 0;
            }
            format++;
            /* bug: '-' here will potentially screw things up */
            assert(*format != '-');

            min=zpad=0;

            if (*format == '0')
                zpad=1;

            min = atoi(format);

            temp = format;
            while (*temp >= '0' && *temp <= '9')
                temp++;

            switch (*(temp++)) {

            case 'c':
                len += charprint(&s, va_arg(ap, int));
                break;

            case 'd':
                len += integerprint(&s, va_arg(ap, int), 10, min, zpad, 0);
                break;

            case 'x':
                len += integerprint(&s, va_arg(ap, int), 16, min, zpad, 0);
                break;

            case 'X':
                len += integerprint(&s, va_arg(ap, int), 16, min, zpad, 1);
                break;

            case 's':
                len += stringprint(&s, (char*) va_arg(ap, int));
                break;

            default:
                len += charprint(&s, '%');
                temp = format;
            }
```

```
            format = temp;
        }
        else {
            regular++;
            format++;
        }
    }

    if (regular > 0) {
        len += mcharprint(&s, format-regular, regular);
        regular = 0;
    }

    *s = 0;

    return len;
}

void vsprintf(char *s, char *format, va_list ap) {
    _vsprintf(s, format, ap);
}

static char vfprintfbuf[256];

void vfprintf(int fd, char *format, va_list ap) {
    int len = _vsprintf(vfprintfbuf, format, ap);
    assert(len < sizeof(vfprintfbuf));
    write(fd, vfprintfbuf, len);
}

void vprintf(char *format, va_list ap) {
    vfprintf(stdout, format, ap);
}

void sprintf(char *s, char *format, ...) {
    va_list ap;
    va_start(ap,format);

    vsprintf(s, format, ap);

    va_end(ap);
}

void fprintf(int fd, char *format, ...) {
    va_list ap;
    va_start(ap,format);

    vfprintf(fd, format, ap);

    va_end(ap);
}

void printf(char *format, ...) {
    va_list ap;
    va_start(ap,format);

    vprintf(format, ap);

    va_end(ap);
}
```

```
#include "stdio.h"
#include "stdlib.h"

void readline(char *s, int maxlength) {
  int i = 0;

  while (1) {
    char c = getch();
    /* if end of line, finish up */
    if (c == '\n') {
      putchar('\n');
      s[i] = 0;
      return;
    }
    /* else if backspace... */
    else if (c == '\b') {
      /* if nothing to delete, beep */
      if (i == 0) {
        beep();
      }
      /* else delete it */
      else {
        printf("\b \b");
        i--;
      }
    }
    /* else if bad character or no room for more, beep */
    else if (c < 0x20 || i+1 == maxlength) {
      beep();
    }
    /* else add the character */
    else {
      s[i++] = c;
      putchar(c);
    }
  }
}
```

```
#include "syscall.h"
#include "stdio.h"
#include "stdlib.h"

int main(int argc, char** argv)
{
  if (argc!=2) {
    printf("Usage: rm <file>\n");
    return 1;
  }

  if (unlink(argv[1]) != 0) {
      printf("Unable to remove %s\n", argv[1]);
      return 1;
  }

  return 0;
}
```

```
OUTPUT_FORMAT("ecoff-littlemips")
SEARCH_DIR(.)
ENTRY(__start)

SECTIONS {
  .text         0               : { *(.text) }
  .rdata        BLOCK(0x400)    : { *(.rdata) }
  .data         BLOCK(0x400)    : { *(.data) }
  .sbss         BLOCK(0x400)    : { *(.sbss) }
  .bss          BLOCK(0x400)    : { *(.bss) }
  .scommon      BLOCK(0x400)    : { *(.scommon) }
}
```

```
#include "stdio.h"
#include "stdlib.h"

#define BUFFERSIZE      64

#define MAXARGSIZE      16
#define MAXARGS         16

/**
 * tokenizeCommand
 *
 * Splits the specified command line into tokens, creating a token array with a maximu
m
 * of maxTokens entries, using storage to hold the tokens. The storage array should be
 as
 * long as the command line.
 *
 * Whitespace (spaces, tabs, newlines) separate tokens, unless
 * enclosed in double quotes. Any character can be quoted by preceeding
 * it with a backslash. Quotes must be terminated.
 *
 * Returns the number of tokens, or -1 on error.
 */
static int tokenizeCommand(char* command, int maxTokens, char *tokens[], char* storage
) {
    const int quotingCharacter = 0x00000001;
    const int quotingString = 0x00000002;
    const int startedArg = 0x00000004;

    int state = 0;
    int numTokens = 0;

    char c;

    assert(maxTokens > 0);

    while ((c = *(command++)) != '\0') {
        if (state & quotingCharacter) {
            switch (c) {
            case 't':
                c = '\t';
                break;
            case 'n':
                c = '\n';
                break;
            }
            *(storage++) = c;
            state &= ~quotingCharacter;
        }
        else if (state & quotingString) {
            switch (c) {
            case '\\':
                state |= quotingCharacter;
                break;
            case '"':
                state &= ~quotingString;
                break;
            default:
                *(storage++) = c;
                break;
            }
        }
        else {
            switch (c) {
            case ' ':
            case '\t':
            case '\n':
                if (state & startedArg) {
                    *(storage++) = '\0';
                    state &= ~startedArg;
                }
                break;
            default:
                if (!(state & startedArg)) {
                    if (numTokens == maxTokens) {
                        return -1;
                    }
                    tokens[numTokens++] = storage;
                    state |= startedArg;
                }

                switch (c) {
                case '\\':
                    state |= quotingCharacter;
                    break;
                case '"':
                    state |= quotingString;
                    break;
                default:
                    *(storage++) = c;
                    break;
                }
            }
        }
    }

    if (state & quotingCharacter) {
        printf("Unmatched \\.\n");
        return -1;
    }

    if (state & quotingString) {
        printf("Unmatched \".\n");
        return -1;
    }

    if (state & startedArg) {
        *(storage++) = '\0';
    }

    return numTokens;
}

void runline(char* line) {
    int pid, background, status;

    char args[BUFFERSIZE], prog[BUFFERSIZE];
    char *argv[MAXARGS];

    int argc = tokenizeCommand(line, MAXARGS, argv, args);
    if (argc <= 0)
        return;

    if (argc > 0 && strcmp(argv[argc-1], "&") == 0) {
```

```
    argc--;
    background = 1;
}
else {
    background = 0;
}

if (argc > 0) {
    if (strcmp(argv[0], "exit")==0) {
        if (argc == 1) {
            exit(0);
        }
        else if (argc == 2) {
            exit(atoi(argv[1]));
        }
        else {
            printf("exit: Expression Syntax.\n");
            return;
        }
    }
    else if (strcmp(argv[0], "halt")==0) {
        if (argc == 1) {
            halt();
            printf("Not the root process!\n");
        }
        else {
            printf("halt: Expression Syntax.\n");
        }
        return;
    }
    else if (strcmp(argv[0], "join")==0) {
        if (argc == 2) {
            pid = atoi(argv[1]);
        }
        else {
            printf("join: Expression Syntax.\n");
            return;
        }
    }
    else {
        strcpy(prog, argv[0]);
        strcat(prog, ".coff");

        pid = exec(prog, argc, argv);
        if (pid == -1) {
            printf("%s: exec failed.\n", argv[0]);
            return;
        }
    }

    if (!background) {
        switch (join(pid, &status)) {
        case -1:
            printf("join: Invalid process ID.\n");
            break;
        case 0:
            printf("\n[%d] Unhandled exception\n", pid);
            break;
        case 1:
            printf("\n[%d] Done (%d)\n", pid, status);
            break;
        }
        else {
            printf("\n[%d]\n", pid);
        }
    }
}

int main(int argc, char *argv[]) {
    char prompt[] = "nachos% ";

    char buffer[BUFFERSIZE];

    while (1) {
        printf("%s", prompt);

        readline(buffer, BUFFERSIZE);

        runline(buffer);
    }
}
```

```
/* sort.c
 *    Test program to sort a large number of integers.
 *
 *    Intention is to stress virtual memory system. To increase the memory
 *    usage of this program, simply increase SORTSHIFT. The size of the array
 *    is (SORTSIZE)(2^(SORTSHIFT+2)).
 */

#include "syscall.h"

/* size of physical memory; with code, we'll run out of space! */
#define SORTSIZE        256
#define SORTSHIFT       0

int array[SORTSIZE<<SORTSHIFT];

#define A(i)    (array[(i)<<SORTSHIFT])

void swap(int* x, int* y)
{
  int temp = *x;
  *x = *y;
  *y = temp;
}

int
main()
{
  int i, j;

  /* first initialize the array, in reverse sorted order */
  for (i=0; i<SORTSIZE; i++)
    A(i) = (SORTSIZE-1)-i;

  /* then sort! */
  for (i=0; i<SORTSIZE-1; i++) {
    for (j=i; j<SORTSIZE; j++) {
      if (A(i) > A(j))
        swap(&A(i), &A(j));
    }
  }

  /* and last, verify */
  for (i=0; i<SORTSIZE; i++) {
    if (A(i) != i)
      return 1;
  }

  /* if successful, return 0 */
  return 0;
}
```

```
/* Start.s
 *      Assembly language assist for user programs running on top of Nachos.
 *
 *      Since we don't want to pull in the entire C library, we define
 *      what we need for a user program here, namely Start and the system
 *      calls.
 */

#define START_S
#include "syscall.h"

        .text
        .align  2

/* ------------------------------------------------------------
 * __start
 *      Initialize running a C program, by calling "main".
 * ------------------------------------------------------------
 */

        .globl  __start
        .ent    __start
__start:
        jal     main
        addu    $4,$2,$0
        jal     exit     /* if we return from main, exit(return value) */
        .end    __start

        .globl  __main
        .ent    __main
__main:
        jr      $31
        .end    __main

/* ------------------------------------------------------------
 * System call stubs:
 *      Assembly language assist to make system calls to the Nachos kernel.
 *      There is one stub per system call, that places the code for the
 *      system call into register r2, and leaves the arguments to the
 *      system call alone (in other words, arg1 is in r4, arg2 is
 *      in r5, arg3 is in r6, arg4 is in r7)
 *
 *      The return value is in r2. This follows the standard C calling
 *      convention on the MIPS.
 * ------------------------------------------------------------
 */

#define SYSCALLSTUB(name, number) \
        .globl  name            ; \
        .ent    name            ; \
name:                           ; \
        addiu   $2,$0,number    ; \
        syscall                 ; \
        j       $31             ; \
        .end    name

        SYSCALLSTUB(halt, syscallHalt)
        SYSCALLSTUB(exit, syscallExit)
        SYSCALLSTUB(exec, syscallExec)
        SYSCALLSTUB(join, syscallJoin)
        SYSCALLSTUB(creat, syscallCreate)
        SYSCALLSTUB(open, syscallOpen)
        SYSCALLSTUB(read, syscallRead)
        SYSCALLSTUB(write, syscallWrite)
        SYSCALLSTUB(close, syscallClose)
        SYSCALLSTUB(unlink, syscallUnlink)
        SYSCALLSTUB(mmap, syscallMmap)
        SYSCALLSTUB(connect, syscallConnect)
        SYSCALLSTUB(accept, syscallAccept)
```

```
/* stdarg.h for GNU.
   Note that the type used in va_arg is supposed to match the
   actual type **after default promotions**.
   Thus, va_arg (..., short) is not valid.  */

#ifndef _STDARG_H
#ifndef _ANSI_STDARG_H_
#ifndef __need___va_list
#define _STDARG_H
#define _ANSI_STDARG_H_
#endif /* not __need___va_list */
#undef __need___va_list

#ifdef __clipper__
#include "va-clipper.h"
#else
#ifdef __m88k__
#include "va-m88k.h"
#else
#ifdef __i860__
#include "va-i860.h"
#else
#ifdef __hppa__
#include "va-pa.h"
#else
#ifdef __mips__
#include "va-mips.h"
#else
#ifdef __sparc__
#include "va-sparc.h"
#else
#ifdef __i960__
#include "va-i960.h"
#else
#ifdef __alpha__
#include "va-alpha.h"
#else
#if defined (__H8300__) || defined (__H8300H__) || defined (__H8300S__)
#include "va-h8300.h"
#else
#if defined (__PPC__) && (defined (_CALL_SYSV) || defined (_WIN32))
#include "va-ppc.h"
#else
#ifdef __arc__
#include "va-arc.h"
#else
#ifdef __M32R__
#include "va-m32r.h"
#else
#ifdef __sh__
#include "va-sh.h"
#else
#ifdef __mn10300__
#include "va-mn10300.h"
#else
#ifdef __mn10200__
#include "va-mn10200.h"
#else
#ifdef __v850__
#include "va-v850.h"
#else
/* Define __gnuc_va_list.  */

#ifndef __GNUC_VA_LIST
#define __GNUC_VA_LIST
#if defined(__svr4__) || defined(_AIX) || defined(_M_UNIX) || defined(__NetBSD__)
typedef char *__gnuc_va_list;
#else
typedef void *__gnuc_va_list;
#endif
#endif

/* Define the standard macros for the user,
   if this invocation was from the user program.  */
#ifdef _STDARG_H

/* Amount of space required in an argument list for an arg of type TYPE.
   TYPE may alternatively be an expression whose type is used.  */

#if defined(sysV68)
#define __va_rounded_size(TYPE)  \
  (((sizeof (TYPE) + sizeof (short) - 1) / sizeof (short)) * sizeof (short))
#else
#define __va_rounded_size(TYPE)  \
  (((sizeof (TYPE) + sizeof (int) - 1) / sizeof (int)) * sizeof (int))
#endif

#define va_start(AP, LASTARG)                                           \
 (AP = ((__gnuc_va_list) __builtin_next_arg (LASTARG)))

#undef va_end
void va_end (__gnuc_va_list);           /* Defined in libgcc.a */
#define va_end(AP)      ((void)0)

/* We cast to void * and then to TYPE * because this avoids
   a warning about increasing the alignment requirement.  */

#if (defined (__arm__) && ! defined (__ARMEB__)) || defined (__i386__) || defined (__i
860__) || defined (__ns32000__) || defined (__vax__)
/* This is for little-endian machines; small args are padded upward.  */
#define va_arg(AP, TYPE)                                                \
 (AP = (__gnuc_va_list) ((char *) (AP) + __va_rounded_size (TYPE)),     \
  *((TYPE *) (void *) ((char *) (AP) - __va_rounded_size (TYPE))))
#else /* big-endian */
/* This is for big-endian machines; small args are padded downward.  */
#define va_arg(AP, TYPE)                                                \
 (AP = (__gnuc_va_list) ((char *) (AP) + __va_rounded_size (TYPE)),     \
  *((TYPE *) (void *) ((char *) (AP)                                    \
                       - ((sizeof (TYPE) < __va_rounded_size (char)     \
                           ? sizeof (TYPE) : __va_rounded_size (TYPE))))))
#endif /* big-endian */

/* Copy __gnuc_va_list into another variable of this type.  */
#define __va_copy(dest, src) (dest) = (src)

#endif /* _STDARG_H */

#endif /* not v850 */
#endif /* not mn10200 */
#endif /* not mn10300 */
#endif /* not sh */
#endif /* not m32r */
#endif /* not arc */
```

```
#endif /* not powerpc with V.4 calling sequence */
#endif /* not h8300 */
#endif /* not alpha */
#endif /* not i960 */
#endif /* not sparc */
#endif /* not mips */
#endif /* not hppa */
#endif /* not i860 */
#endif /* not m88k */
#endif /* not clipper */

#ifdef _STDARG_H
/* Define va_list, if desired, from __gnuc_va_list. */
/* We deliberately do not define va_list when called from
   stdio.h, because ANSI C says that stdio.h is not supposed to define
   va_list.  stdio.h needs to have access to that data type,
   but must not use that name.  It should use the name __gnuc_va_list,
   which is safe because it is reserved for the implementation.  */

#ifdef _HIDDEN_VA_LIST  /* On OSF1, this means varargs.h is "half-loaded".  */
#undef _VA_LIST
#endif

#ifdef _BSD_VA_LIST
#undef _BSD_VA_LIST
#endif

#if defined(__svr4__) || (defined(_SCO_DS) && !defined(__VA_LIST))
/* SVR4.2 uses _VA_LIST for an internal alias for va_list,
   so we must avoid testing it and setting it here.
   SVR4 uses _VA_LIST as a flag in stdarg.h, but we should
   have no conflict with that.  */
#ifndef _VA_LIST_
#define _VA_LIST_
#ifdef __i860__
#ifndef _VA_LIST
#define _VA_LIST va_list
#endif
#endif /* __i860__ */
typedef __gnuc_va_list va_list;
#ifdef _SCO_DS
#define __VA_LIST
#endif
#endif /* _VA_LIST_ */
#else /* not __svr4__ || _SCO_DS */

/* The macro _VA_LIST_ is the same thing used by this file in Ultrix.
   But on BSD NET2 we must not test or define or undef it.
   (Note that the comments in NET 2's ansi.h
   are incorrect for _VA_LIST_--see stdio.h!)  */
#if !defined (_VA_LIST_) || defined (__BSD_NET2__) || defined (____386BSD____) || defi
ned (__bsdi__) || defined (__sequent__) || defined (__FreeBSD__) || defined(WINNT)
/* The macro _VA_LIST_DEFINED is used in Windows NT 3.5  */
#ifndef _VA_LIST_DEFINED
/* The macro _VA_LIST is used in SCO Unix 3.2.  */
#ifndef _VA_LIST
/* The macro _VA_LIST_T_H is used in the Bull dpx2  */
#ifndef _VA_LIST_T_H
typedef __gnuc_va_list va_list;
#endif /* not _VA_LIST_T_H */
#endif /* not _VA_LIST */
#endif /* not _VA_LIST_DEFINED */
#if !(defined (__BSD_NET2__) || defined (____386BSD____) || defined (__bsdi__) || defi
ned (__sequent__) || defined (__FreeBSD__))
#define _VA_LIST_
#endif
#ifndef _VA_LIST
#define _VA_LIST
#endif
#ifndef _VA_LIST_DEFINED
#define _VA_LIST_DEFINED
#endif
#ifndef _VA_LIST_T_H
#define _VA_LIST_T_H
#endif

#endif /* not _VA_LIST_, except on certain systems */

#endif /* not __svr4__ */

#endif /* _STDARG_H */

#endif /* not _ANSI_STDARG_H_ */
#endif /* not _STDARG_H */
```

```
#include "stdio.h"
#include "stdlib.h"

int fgetc(int fd) {
    unsigned char c;

    while (read(fd, &c, 1) != 1);

    return c;
}

void fputc(char c, int fd) {
    write(fd, &c, 1);
}

void fputs(const char *s, int fd) {
    write(fd, (char*) s, strlen(s));
}
```

```
/*----------------------------------------------------------
 * stdio.h
 *
 * Header file for standard I/O routines.
 *----------------------------------------------------------*/

#ifndef STDIO_H
#define STDIO_H

#include "syscall.h"
#include "stdarg.h"

typedef int             FILE;
#define stdin           fdStandardInput
#define stdout          fdStandardOutput

int  fgetc(FILE stream);
void readline(char *s, int maxlength);
int  tryreadline(char *s, char c, int maxlength);

#define getc(stream)    fgetc(stream)
#define getchar()       getc(stdin)
#define getch()         getchar()

void fputc(char c, FILE stream);
void fputs(const char *s, FILE stream);

#define puts(s)         fputs(s,stdout)
#define putc(c,stream)  fputc(c,stream)
#define putchar(c)      putc(c,stdout)
#define beep()          putchar(0x07)

void vsprintf(char *s, char *format, va_list ap);
void vfprintf(FILE f, char *format, va_list ap);
void vprintf(char *format, va_list ap);
void sprintf(char *s, char *format, ...);
void fprintf(FILE f, char *format, ...);
void printf(char *format, ...);

#endif // STDIO_H
```

```
#include "stdlib.h"
```

```
/*-----------------------------------------------------------
 * stdlib.h
 *
 * Header file for standard library functions.
 *-----------------------------------------------------------*/

#ifndef STDLIB_H
#define STDLIB_H

#include "syscall.h"

#define null    0L
#define true    1
#define false   0

#define min(a,b)  (((a) < (b)) ? (a) : (b))
#define max(a,b)  (((a) > (b)) ? (a) : (b))

#define divRoundDown(n,s)  ((n) / (s))
#define divRoundUp(n,s)    (((n) / (s)) + ((((n) % (s)) > 0) ? 1 : 0))

#define assert(_EX)     ((_EX) ? (void) 0 : __assert(__FILE__, __LINE__))
void __assert(char* file, int line);

#define assertNotReached()      assert(false)

void *memcpy(void *s1, const void *s2, unsigned int n);
void *memset(void *s, int c, unsigned int n);

unsigned int strlen(const char *str);
char *strcpy(char *dst, const char *src);
int strcmp(const char *a, const char *b);
int strncmp(const char *a, const char *b, int n);

int atoi(const char *s);

#endif // STDLIB_H
```

```
#include "stdlib.h"

/* concatenates s2 to the end of s1 and returns s1 */
char *strcat(char *s1, const char *s2) {
    char* result = s1;

    while (*s1 != 0)
        s1++;

    do {
        *(s1++) = *(s2);
    }
    while (*(s2++) != 0);

    return result;
}
```

```
#include "stdlib.h"

/* lexicographically compares a and b */
int strcmp(const char* a, const char* b) {
  do {
    if (*a < *b)
      return -1;
    if (*a > *b)
      return 1;
  }
  while (*(a++) != 0 && *(b++) != 0);

  return 0;
}
```

```
#include "stdlib.h"

/* copies src to dst, returning dst */
char *strcpy(char *dst, const char *src) {
  int n=0;
  char *result = dst;

  do {
    *(dst++) = *src;
    n++;
  }
  while (*(src++) != 0);

  return result;
}
```

```
#include "stdlib.h"

/* returns the length of the character string str, not including the null-terminator *
/
unsigned strlen(const char *str) {
  int result=0;

  while (*(str++) != 0)
    result++;

  return result;
}
```

```
#include "stdlib.h"

/* lexicographically compares a and b up to n chars */
int strncmp(const char* a, const char* b, int n)
{
  assert(n > 0);

  do {
    if (*a < *b)
      return -1;
    if (*a > *b)
      return 1;
    n--;
    a++;
    b++;
  }
  while (n > 0);

  return 0;
}
```

```
/**
 * The Nachos system call interface. These are Nachos kernel operations that
 * can be invoked from user programs using the syscall instruction.
 *
 * This interface is derived from the UNIX syscalls. This information is
 * largely copied from the UNIX man pages.
 */

#ifndef SYSCALL_H
#define SYSCALL_H

/**
 * System call codes, passed in $r0 to tell the kernel which system call to do.
 */
#define syscallHalt             0
#define syscallExit             1
#define syscallExec             2
#define syscallJoin             3
#define syscallCreate           4
#define syscallOpen             5
#define syscallRead             6
#define syscallWrite            7
#define syscallClose            8
#define syscallUnlink           9
#define syscallMmap             10
#define syscallConnect          11
#define syscallAccept           12

/* Don't want the assembler to see C code, but start.s includes syscall.h. */
#ifndef START_S

/* When a process is created, two streams are already open. File descriptor 0
 * refers to keyboard input (UNIX stdin), and file descriptor 1 refers to
 * display output (UNIX stdout). File descriptor 0 can be read, and file
 * descriptor 1 can be written, without previous calls to open().
 */
#define fdStandardInput         0
#define fdStandardOutput        1

/* The system call interface. These are the operations the Nachos kernel needs
 * to support, to be able to run user programs.
 *
 * Each of these is invoked by a user program by simply calling the procedure;
 * an assembly language stub stores the syscall code (see above) into $r0 and
 * executes a syscall instruction. The kernel exception handler is then
 * invoked.
 */

/* Halt the Nachos machine by calling Machine.halt(). Only the root process
 * (the first process, executed by UserKernel.run()) should be allowed to
 * execute this syscall. Any other process should ignore the syscall and return
 * immediately.
 */
void halt();

/* PROCESS MANAGEMENT SYSCALLS: exit(), exec(), join() */

/**
 * Terminate the current process immediately. Any open file descriptors
 * belonging to the process are closed. Any children of the process no longer
 * have a parent process.
 *
 * status is returned to the parent process as this process's exit status and
 * can be collected using the join syscall. A process exiting normally should
 * (but is not required to) set status to 0.
 *
 * exit() never returns.
 */
void exit(int status);

/**
 * Execute the program stored in the specified file, with the specified
 * arguments, in a new child process. The child process has a new unique
 * process ID, and starts with stdin opened as file descriptor 0, and stdout
 * opened as file descriptor 1.
 *
 * file is a null-terminated string that specifies the name of the file
 * containing the executable. Note that this string must include the ".coff"
 * extension.
 *
 * argc specifies the number of arguments to pass to the child process. This
 * number must be non-negative.
 *
 * argv is an array of pointers to null-terminated strings that represent the
 * arguments to pass to the child process. argv[0] points to the first
 * argument, and argv[argc-1] points to the last argument.
 *
 * exec() returns the child process's process ID, which can be passed to
 * join(). On error, returns -1.
 */
int exec(char *file, int argc, char *argv[]);

/**
 * Suspend execution of the current process until the child process specified
 * by the processID argument has exited. If the child has already exited by the
 * time of the call, returns immediately. When the current process resumes, it
 * disowns the child process, so that join() cannot be used on that process
 * again.
 *
 * processID is the process ID of the child process, returned by exec().
 *
 * status points to an integer where the exit status of the child process will
 * be stored. This is the value the child passed to exit(). If the child exited
 * because of an unhandled exception, the value stored is not defined.
 *
 * If the child exited normally, returns 1. If the child exited as a result of
 * an unhandled exception, returns 0. If processID does not refer to a child
 * process of the current process, returns -1.
 */
int join(int processID, int *status);

/* FILE MANAGEMENT SYSCALLS: creat, open, read, write, close, unlink
 *
 * A file descriptor is a small, non-negative integer that refers to a file on
 * disk or to a stream (such as console input, console output, and network
 * connections). A file descriptor can be passed to read() and write() to
 * read/write the corresponding file/stream. A file descriptor can also be
 * passed to close() to release the file descriptor and any associated
 * resources.
 */

/**
 * Attempt to open the named disk file, creating it if it does not exist,
 * and return a file descriptor that can be used to access the file.
```

```
 *
 * Note that creat() can only be used to create files on disk; creat() will
 * never return a file descriptor referring to a stream.
 *
 * Returns the new file descriptor, or -1 if an error occurred.
 */
int creat(char *name);

/**
 * Attempt to open the named file and return a file descriptor.
 *
 * Note that open() can only be used to open files on disk; open() will never
 * return a file descriptor referring to a stream.
 *
 * Returns the new file descriptor, or -1 if an error occurred.
 */
int open(char *name);

/**
 * Attempt to read up to count bytes into buffer from the file or stream
 * referred to by fileDescriptor.
 *
 * On success, the number of bytes read is returned. If the file descriptor
 * refers to a file on disk, the file position is advanced by this number.
 *
 * It is not necessarily an error if this number is smaller than the number of
 * bytes requested. If the file descriptor refers to a file on disk, this
 * indicates that the end of the file has been reached. If the file descriptor
 * refers to a stream, this indicates that the fewer bytes are actually
 * available right now than were requested, but more bytes may become available
 * in the future. Note that read() never waits for a stream to have more data;
 * it always returns as much as possible immediately.
 *
 * On error, -1 is returned, and the new file position is undefined. This can
 * happen if fileDescriptor is invalid, if part of the buffer is read-only or
 * invalid, or if a network stream has been terminated by the remote host and
 * no more data is available.
 */
int read(int fileDescriptor, void *buffer, int count);

/**
 * Attempt to write up to count bytes from buffer to the file or stream
 * referred to by fileDescriptor. write() can return before the bytes are
 * actually flushed to the file or stream. A write to a stream can block,
 * however, if kernel queues are temporarily full.
 *
 * On success, the number of bytes written is returned (zero indicates nothing
 * was written), and the file position is advanced by this number. It IS an
 * error if this number is smaller than the number of bytes requested. For
 * disk files, this indicates that the disk is full. For streams, this
 * indicates the stream was terminated by the remote host before all the data
 * was transferred.
 *
 * On error, -1 is returned, and the new file position is undefined. This can
 * happen if fileDescriptor is invalid, if part of the buffer is invalid, or
 * if a network stream has already been terminated by the remote host.
 */
int write(int fileDescriptor, void *buffer, int count);

/**
 * Close a file descriptor, so that it no longer refers to any file or stream
 * and may be reused.
 *
 * If the file descriptor refers to a file, all data written to it by write()
 * will be flushed to disk before close() returns.
 * If the file descriptor refers to a stream, all data written to it by write()
 * will eventually be flushed (unless the stream is terminated remotely), but
 * not necessarily before close() returns.
 *
 * The resources associated with the file descriptor are released. If the
 * descriptor is the last reference to a disk file which has been removed using
 * unlink, the file is deleted (this detail is handled by the file system
 * implementation).
 *
 * Returns 0 on success, or -1 if an error occurred.
 */
int close(int fileDescriptor);

/**
 * Delete a file from the file system. If no processes have the file open, the
 * file is deleted immediately and the space it was using is made available for
 * reuse.
 *
 * If any processes still have the file open, the file will remain in existence
 * until the last file descriptor referring to it is closed. However, creat()
 * and open() will not be able to return new file descriptors for the file
 * until it is deleted.
 *
 * Returns 0 on success, or -1 if an error occurred.
 */
int unlink(char *name);

/**
 * Map the file referenced by fileDescriptor into memory at address. The file
 * may be as large as 0x7FFFFFFF bytes.
 *
 * To maintain consistency, further calls to read() and write() on this file
 * descriptor will fail (returning -1) until the file descriptor is closed.
 *
 * When the file descriptor is closed, all remaining dirty pages of the map
 * will be flushed to disk and the map will be removed.
 *
 * Returns the length of the file on success, or -1 if an error occurred.
 */
int mmap(int fileDescriptor, char *address);

/**
 * Attempt to initiate a new connection to the specified port on the specified
 * remote host, and return a new file descriptor referring to the connection.
 * connect() does not give up if the remote host does not respond immediately.
 *
 * Returns the new file descriptor, or -1 if an error occurred.
 */
int connect(int host, int port);

/**
 * Attempt to accept a single connection on the specified local port and return
 * a file descriptor referring to the connection.
 *
 * If any connection requests are pending on the port, one request is dequeued
 * and an acknowledgement is sent to the remote host (so that its connect()
 * call can return). Since the remote host will never cancel a connection
 * request, there is no need for accept() to wait for the remote host to
 * confirm the connection (i.e. a 2-way handshake is sufficient; TCP's 3-way
```

```
 * handshake is unnecessary).
 *
 * If no connection requests are pending, returns -1 immediately.
 *
 * In either case, accept() returns without waiting for a remote host.
 *
 * Returns a new file descriptor referring to the connection, or -1 if an error
 * occurred.
 */
int accept(int port);

#endif /* START_S */

#endif /* SYSCALL_H */
```

```
/* ----------------------------------------- */
/*           VARARGS  for MIPS/GNU CC        */
/*                                           */
/*                                           */
/*                                           */
/*                                           */
/* ----------------------------------------- */


/* These macros implement varargs for GNU C--either traditional or ANSI.  */

/* Define __gnuc_va_list.  */

#ifndef __GNUC_VA_LIST
#define __GNUC_VA_LIST
#if defined (__mips_eabi) && ! defined (__mips_soft_float) && ! defined (__mips_single
_float)

typedef struct {
  /* Pointer to FP regs.  */
  char *__fp_regs;
  /* Number of FP regs remaining.  */
  int __fp_left;
  /* Pointer to GP regs followed by stack parameters.  */
  char *__gp_regs;
} __gnuc_va_list;

#else /* ! (defined (__mips_eabi) && ! defined (__mips_soft_float) && ! defined (__mip
s_single_float)) */

typedef char * __gnuc_va_list;

#endif /* ! (defined (__mips_eabi) && ! defined (__mips_soft_float) && ! defined (__mi
ps_single_float)) */
#endif /* not __GNUC_VA_LIST */

/* If this is for internal libc use, don't define anything but
   __gnuc_va_list.  */
#if defined (_STDARG_H) || defined (_VARARGS_H)

#ifndef _VA_MIPS_H_ENUM
#define _VA_MIPS_H_ENUM
enum {
  __no_type_class = -1,
  __void_type_class,
  __integer_type_class,
  __char_type_class,
  __enumeral_type_class,
  __boolean_type_class,
  __pointer_type_class,
  __reference_type_class,
  __offset_type_class,
  __real_type_class,
  __complex_type_class,
  __function_type_class,
  __method_type_class,
  __record_type_class,
  __union_type_class,
  __array_type_class,
  __string_type_class,
  __set_type_class,
  __file_type_class,
  __lang_type_class
};
#endif

/* In GCC version 2, we want an ellipsis at the end of the declaration
   of the argument list.  GCC version 1 can't parse it.  */

#if __GNUC__ > 1
#define __va_ellipsis ...
#else
#define __va_ellipsis
#endif

#ifdef __mips64
#define __va_rounded_size(__TYPE)  \
  (((sizeof (__TYPE) + 8 - 1) / 8) * 8)
#else
#define __va_rounded_size(__TYPE)  \
  (((sizeof (__TYPE) + sizeof (int) - 1) / sizeof (int)) * sizeof (int))
#endif

#ifdef __mips64
#define __va_reg_size 8
#else
#define __va_reg_size 4
#endif

/* Get definitions for _MIPS_SIM_ABI64 etc.  */
#ifdef _MIPS_SIM
#include <sgidefs.h>
#endif

#ifdef _STDARG_H
#if defined (__mips_eabi)
#if ! defined (__mips_soft_float) && ! defined (__mips_single_float)
#ifdef __mips64
#define va_start(__AP, __LASTARG)                                        \
  (__AP.__gp_regs = ((char *) __builtin_next_arg (__LASTARG)             \
                     - (__builtin_args_info (2) < 8                      \
                        ? (8 - __builtin_args_info (2)) * __va_reg_size  \
                        : 0)),                                           \
   __AP.__fp_left = 8 - __builtin_args_info (3),                         \
   __AP.__fp_regs = __AP.__gp_regs - __AP.__fp_left * __va_reg_size)
#else /* ! defined (__mips64) */
#define va_start(__AP, __LASTARG)                                        \
  (__AP.__gp_regs = ((char *) __builtin_next_arg (__LASTARG)             \
                     - (__builtin_args_info (2) < 8                      \
                        ? (8 - __builtin_args_info (2)) * __va_reg_size  \
                        : 0)),                                           \
   __AP.__fp_left = (8 - __builtin_args_info (3)) / 2,                   \
   __AP.__fp_regs = __AP.__gp_regs - __AP.__fp_left * 8,                 \
   __AP.__fp_regs = (char *) ((int) __AP.__fp_regs & -8))
#endif /* ! defined (__mips64) */
#else /* ! (! defined (__mips_soft_float) && ! defined (__mips_single_float) ) */
#define va_start(__AP, __LASTARG)                                        \
  (__AP = ((__gnuc_va_list) __builtin_next_arg (__LASTARG)               \
           - (__builtin_args_info (2) >= 8 ? 0                           \
              : (8 - __builtin_args_info (2)) * __va_reg_size)))
#endif /* ! (! defined (__mips_soft_float) && ! defined (__mips_single_float) ) */
#else /* ! defined (__mips_eabi) */
#define va_start(__AP, __LASTARG) \
  (__AP = (__gnuc_va_list) __builtin_next_arg (__LASTARG))
```

```
#endif /* ! (defined (__mips_eabi) && ! defined (__mips_soft_float) && ! defined (__mi
ps_single_float)) */
#else /* ! _STDARG_H */
#define va_alist  __builtin_va_alist
#ifdef __mips64
/* This assumes that 'long long int' is always a 64 bit type.  */
#define va_dcl    long long int __builtin_va_alist; __va_ellipsis
#else
#define va_dcl    int __builtin_va_alist; __va_ellipsis
#endif
#if defined (__mips_eabi)
#if ! defined (__mips_soft_float) && ! defined (__mips_single_float)
#ifdef __mips64
#define va_start(__AP)                                          \
  (__AP.__gp_regs = ((char *) __builtin_next_arg ()             \
                     - (__builtin_args_info (2) < 8             \
                        ? (8 - __builtin_args_info (2)) * __va_reg_size \
                        : __va_reg_size)),                      \
   __AP.__fp_left = 8 - __builtin_args_info (3),                \
   __AP.__fp_regs = __AP.__gp_regs - __AP.__fp_left * __va_reg_size)
#else /* ! defined (__mips64) */
#define va_start(__AP)                                          \
  (__AP.__gp_regs = ((char *) __builtin_next_arg ()             \
                     - (__builtin_args_info (2) < 8             \
                        ? (8 - __builtin_args_info (2)) * __va_reg_size \
                        : __va_reg_size)),                      \
   __AP.__fp_left = (8 - __builtin_args_info (3)) / 2,          \
   __AP.__fp_regs = __AP.__gp_regs - __AP.__fp_left * 8,        \
   __AP.__fp_regs = (char *) ((int) __AP.__fp_regs & -8))
#endif /* ! defined (__mips64) */
#else /* ! (! defined (__mips_soft_float) && ! defined (__mips_single_float)) */
#define va_start(__AP)                                          \
  (__AP = ((__gnuc_va_list) __builtin_next_arg ()               \
           - (__builtin_args_info (2) >= 8 ? __va_reg_size      \
              : (8 - __builtin_args_info (2)) * __va_reg_size)))
#endif /* ! (! defined (__mips_soft_float) && ! defined (__mips_single_float)) */
/* Need alternate code for _MIPS_SIM_ABI64.  */
#elif defined(_MIPS_SIM) && (_MIPS_SIM == _MIPS_SIM_ABI64 || _MIPS_SIM == _MIPS_SIM_NA
BI32)
#define va_start(__AP)                                          \
  (__AP = (__gnuc_va_list) __builtin_next_arg ()                \
   + (__builtin_args_info (2) >= 8 ? -8 : 0))
#else
#define va_start(__AP)  __AP = (char *) &__builtin_va_alist
#endif
#endif /* ! _STDARG_H */

#ifndef va_end
void va_end (__gnuc_va_list);           /* Defined in libgcc.a */
#endif
#define va_end(__AP)    ((void)0)

#if defined (__mips_eabi)

#if ! defined (__mips_soft_float) && ! defined (__mips_single_float)
#ifdef __mips64
#define __va_next_addr(__AP, __type)                                    \
  ((__builtin_classify_type (*(__type *) 0) == __real_type_class        \
    && __AP.__fp_left > 0)                                              \
   ? (--__AP.__fp_left, (__AP.__fp_regs += 8) - 8)                      \
   : (__AP.__gp_regs += __va_reg_size) - __va_reg_size)
#else
#define __va_next_addr(__AP, __type)                                    \
  ((__builtin_classify_type (*(__type *) 0) == __real_type_class        \
    && __AP.__fp_left > 0)                                              \
   ? (--__AP.__fp_left, (__AP.__fp_regs += 8) - 8)                      \
   : (((__builtin_classify_type (* (__type *) 0) < __record_type_class  \
        && __alignof__ (__type) > 4)                                    \
       ? __AP.__gp_regs = (char *) (((int) __AP.__gp_regs + 8 - 1) & -8) \
       : (char *) 0),                                                   \
      (__builtin_classify_type (* (__type *) 0) >= __record_type_class  \
       ? (__AP.__gp_regs += __va_reg_size) - __va_reg_size              \
       : ((__AP.__gp_regs += __va_rounded_size (__type))                \
          - __va_rounded_size (__type)))))
#endif
#else /* ! (! defined (__mips_soft_float) && ! defined (__mips_single_float)) */
#ifdef __mips64
#define __va_next_addr(__AP, __type)                                    \
  ((__AP += __va_reg_size) - __va_reg_size)
#else
#define __va_next_addr(__AP, __type)                                    \
  (((__builtin_classify_type (* (__type *) 0) < __record_type_class     \
     && __alignof__ (__type) > 4)                                       \
    ? __AP = (char *) (((int) __AP + 8 - 1) & -8)                       \
    : (char *) 0),                                                      \
   (__builtin_classify_type (* (__type *) 0) >= __record_type_class     \
    ? (__AP += __va_reg_size) - __va_reg_size                           \
    : ((__AP += __va_rounded_size (__type))                             \
       - __va_rounded_size (__type))))
#endif
#endif /* ! (! defined (__mips_soft_float) && ! defined (__mips_single_float)) */

#ifdef __MIPSEB__
#define va_arg(__AP, __type)                                            \
  ((__va_rounded_size (__type) <= __va_reg_size)                        \
   ? *(__type *) (void *) (__va_next_addr (__AP, __type)                \
                           + __va_reg_size                              \
                           - sizeof (__type))                           \
   : (__builtin_classify_type (*(__type *) 0) >= __record_type_class    \
      ? **(__type **) (void *) (__va_next_addr (__AP, __type)           \
                                + __va_reg_size                         \
                                - sizeof (char *))                      \
      : *(__type *) (void *) __va_next_addr (__AP, __type)))
#else
#define va_arg(__AP, __type)                                            \
  ((__va_rounded_size (__type) <= __va_reg_size)                        \
   ? *(__type *) (void *) __va_next_addr (__AP, __type)         \
   : (__builtin_classify_type (* (__type *) 0) >= __record_type_class   \
      ? **(__type **) (void *) __va_next_addr (__AP, __type)            \
      : *(__type *) (void *) __va_next_addr (__AP, __type)))
#endif

#else /* ! defined (__mips_eabi) */

/* We cast to void * and then to TYPE * because this avoids
   a warning about increasing the alignment requirement.  */
/* The __mips64 cases are reversed from the 32 bit cases, because the standard
   32 bit calling convention left-aligns all parameters smaller than a word,
   whereas the __mips64 calling convention does not (and hence they are
   right aligned).  */
#ifdef __mips64
#ifdef __MIPSEB__
#define va_arg(__AP, __type)                                    \
  ((__type *) (void *) (__AP = (char *) ((((__PTRDIFF_TYPE__)__AP + 8 - 1) & -8) \
```

```
                                      + __va_rounded_size (__type))))[-1]
#else
#define va_arg(__AP, __type)                                          \
  ((__AP = (char *) ((((__PTRDIFF_TYPE__)__AP + 8 - 1) & -8)    \
                     + __va_rounded_size (__type))),            \
   *(__type *) (void *) (__AP - __va_rounded_size (__type)))
#endif

#else /* not __mips64 */

#ifdef __MIPSEB__
/* For big-endian machines.  */
#define va_arg(__AP, __type)                                          \
  ((__AP = (char *) ((__alignof__ (__type) > 4                \
                      ? ((int)__AP + 8 - 1) & -8              \
                      : ((int)__AP + 4 - 1) & -4)             \
                     + __va_rounded_size (__type))),            \
   *(__type *) (void *) (__AP - __va_rounded_size (__type)))
#else
/* For little-endian machines.  */
#define va_arg(__AP, __type)                                                \
  ((__type *) (void *) (__AP = (char *) ((__alignof__(__type) > 4           \
                                       ? ((int)__AP + 8 - 1) & -8       \
                                       : ((int)__AP + 4 - 1) & -4)      \
                                      + __va_rounded_size(__type))))[-1]
#endif
#endif
#endif /* ! defined (__mips_eabi)  */

/* Copy __gnuc_va_list into another variable of this type.  */
#define __va_copy(dest, src) (dest) = (src)

#endif /* defined (_STDARG_H) || defined (_VARARGS_H) */
```

```
package nachos.threads;

import nachos.machine.*;

/**
 * Uses the hardware timer to provide preemption, and to allow threads to sleep
 * until a certain time.
 */
public class Alarm {
    /**
     * Allocate a new Alarm. Set the machine's timer interrupt handler to this
     * alarm's callback.
     *
     * <p><b>Note</b>: Nachos will not function correctly with more than one
     * alarm.
     */
    public Alarm() {
        Machine.timer().setInterruptHandler(new Runnable() {
                public void run() { timerInterrupt(); }
            });
    }

    /**
     * The timer interrupt handler. This is called by the machine's timer
     * periodically (approximately every 500 clock ticks). Causes the current
     * thread to yield, forcing a context switch if there is another thread
     * that should be run.
     */
    public void timerInterrupt() {
        KThread.currentThread().yield();
    }

    /**
     * Put the current thread to sleep for at least <i>x</i> ticks,
     * waking it up in the timer interrupt handler. The thread must be
     * woken up (placed in the scheduler ready set) during the first timer
     * interrupt where
     *
     * <p><blockquote>
     * (current time) >= (WaitUntil called time)+(x)
     * </blockquote>
     *
     * @param   x       the minimum number of clock ticks to wait.
     *
     * @see     nachos.machine.Timer#getTime()
     */
    public void waitUntil(long x) {
        // for now, cheat just to get something working (busy waiting is bad)
        long wakeTime = Machine.timer().getTime() + x;
        while (wakeTime > Machine.timer().getTime())
            KThread.yield();
    }
}
```

```
package nachos.threads;
import nachos.ag.BoatGrader;

public class Boat
{
    static BoatGrader bg;

    public static void selfTest()
    {
        BoatGrader b = new BoatGrader();

        System.out.println("\n ***Testing Boats with only 2 children***");
        begin(0, 2, b);

//      System.out.println("\n ***Testing Boats with 2 children, 1 adult***");
//      begin(1, 2, b);

//      System.out.println("\n ***Testing Boats with 3 children, 3 adults***");
//      begin(3, 3, b);
    }

    public static void begin( int adults, int children, BoatGrader b )
    {
        // Store the externally generated autograder in a class
        // variable to be accessible by children.
        bg = b;

        // Instantiate global variables here

        // Create threads here. See section 3.4 of the Nachos for Java
        // Walkthrough linked from the projects page.

        Runnable r = new Runnable() {
            public void run() {
                SampleItinerary();
            }
        };
        KThread t = new KThread(r);
        t.setName("Sample Boat Thread");
        t.fork();

    }

    static void AdultItinerary()
    {
        /* This is where you should put your solutions. Make calls
           to the BoatGrader to show that it is synchronized. For
           example:
               bg.AdultRowToMolokai();
           indicates that an adult has rowed the boat across to Molokai
        */
    }

    static void ChildItinerary()
    {
    }

    static void SampleItinerary()
    {
        // Please note that this isn't a valid solution (you can't fit
        // all of them on the boat). Please also note that you may not
        // have a single thread calculate a solution and then just play
        // it back at the autograder -- you will be caught.
        System.out.println("\n ***Everyone piles on the boat and goes to Molokai***");
        bg.AdultRowToMolokai();
        bg.ChildRideToMolokai();
        bg.AdultRideToMolokai();
        bg.ChildRideToMolokai();
    }

}
```

```
package nachos.threads;

import nachos.machine.*;

/**
 * A <i>communicator</i> allows threads to synchronously exchange 32-bit
 * messages. Multiple threads can be waiting to <i>speak</i>,
 * and multiple threads can be waiting to <i>listen</i>. But there should never
 * be a time when both a speaker and a listener are waiting, because the two
 * threads can be paired off at this point.
 */
public class Communicator {
    /**
     * Allocate a new communicator.
     */
    public Communicator() {
    }

    /**
     * Wait for a thread to listen through this communicator, and then transfer
     * <i>word</i> to the listener.
     *
     * <p>
     * Does not return until this thread is paired up with a listening thread.
     * Exactly one listener should receive <i>word</i>.
     *
     * @param   word    the integer to transfer.
     */
    public void speak(int word) {
    }

    /**
     * Wait for a thread to speak through this communicator, and then return
     * the <i>word</i> that thread passed to <tt>speak()</tt>.
     *
     * @return  the integer transferred.
     */
    public int listen() {
        return 0;
    }
}
```

```
package nachos.threads;

import nachos.machine.*;

import java.util.LinkedList;

/**
 * An implementation of condition variables built upon semaphores.
 *
 * <p>
 * A condition variable is a synchronization primitive that does not have
 * a value (unlike a semaphore or a lock), but threads may still be queued.
 *
 * <p><ul>
 *
 * <li><tt>sleep()</tt>: atomically release the lock and relinkquish the CPU
 * until woken; then reacquire the lock.
 *
 * <li><tt>wake()</tt>: wake up a single thread sleeping in this condition
 * variable, if possible.
 *
 * <li><tt>wakeAll()</tt>: wake up all threads sleeping inn this condition
 * variable.
 *
 * </ul>
 *
 * <p>
 * Every condition variable is associated with some lock. Multiple condition
 * variables may be associated with the same lock. All three condition variable
 * operations can only be used while holding the associated lock.
 *
 * <p>
 * In Nachos, condition variables are summed to obey <i>Mesa-style</i>
 * semantics. When a <tt>wake()</tt> or <tt>wakeAll()</tt> wakes up another
 * thread, the woken thread is simply put on the ready list, and it is the
 * responsibility of the woken thread to reacquire the lock (this reacquire is
 * taken core of in <tt>sleep()</tt>).
 *
 * <p>
 * By contrast, some implementations of condition variables obey
 * <i>Hoare-style</i> semantics, where the thread that calls <tt>wake()</tt>
 * gives up the lock and the CPU to the woken thread, which runs immediately
 * and gives the lock and CPU back to the waker when the woken thread exits the
 * critical section.
 *
 * <p>
 * The consequence of using Mesa-style semantics is that some other thread
 * can acquire the lock and change data structures, before the woken thread
 * gets a chance to run. The advance to Mesa-style semantics is that it is a
 * lot easier to implement.
 */
public class Condition {
    /**
     * Allocate a new condition variable.
     *
     * @param   conditionLock   the lock associated with this condition
     *                          variable. The current thread must hold this
     *                          lock whenever it uses <tt>sleep()</tt>,
     *                          <tt>wake()</tt>, or <tt>wakeAll()</tt>.
     */
    public Condition(Lock conditionLock) {
        this.conditionLock = conditionLock;
        waitQueue = new LinkedList<Semaphore>();
    }

    /**
     * Atomically release the associated lock and go to sleep on this condition
     * variable until another thread wakes it using <tt>wake()</tt>. The
     * current thread must hold the associated lock. The thread will
     * automatically reacquire the lock before <tt>sleep()</tt> returns.
     *
     * <p>
     * This implementation uses semaphores to implement this, by allocating a
     * semaphore for each waiting thread. The waker will <tt>V()</tt> this
     * semaphore, so thre is no chance the sleeper will miss the wake-up, even
     * though the lock is released before caling <tt>P()</tt>.
     */
    public void sleep() {
        Lib.assertTrue(conditionLock.isHeldByCurrentThread());

        Semaphore waiter = new Semaphore(0);
        waitQueue.add(waiter);

        conditionLock.release();
        waiter.P();
        conditionLock.acquire();
    }

    /**
     * Wake up at most one thread sleeping on this condition variable. The
     * current thread must hold the associated lock.
     */
    public void wake() {
        Lib.assertTrue(conditionLock.isHeldByCurrentThread());

        if (!waitQueue.isEmpty())
            ((Semaphore) waitQueue.removeFirst()).V();
    }

    /**
     * Wake up all threads sleeping on this condition variable. The current
     * thread must hold the associated lock.
     */
    public void wakeAll() {
        Lib.assertTrue(conditionLock.isHeldByCurrentThread());

        while (!waitQueue.isEmpty())
            wake();
    }

    private Lock conditionLock;
    private LinkedList<Semaphore> waitQueue;
}
```

```
package nachos.threads;

import nachos.machine.*;

/**
 * An implementation of condition variables that disables interrupt()s for
 * synchronization.
 *
 * <p>
 * You must implement this.
 *
 * @see nachos.threads.Condition
 */
public class Condition2 {
    /**
     * Allocate a new condition variable.
     *
     * @param   conditionLock   the lock associated with this condition
     *                          variable. The current thread must hold this
     *                          lock whenever it uses <tt>sleep()</tt>,
     *                          <tt>wake()</tt>, or <tt>wakeAll()</tt>.
     */
    public Condition2(Lock conditionLock) {
        this.conditionLock = conditionLock;
    }

    /**
     * Atomically release the associated lock and go to sleep on this condition
     * variable until another thread wakes it using <tt>wake()</tt>. The
     * current thread must hold the associated lock. The thread will
     * automatically reacquire the lock before <tt>sleep()</tt> returns.
     */
    public void sleep() {
        Lib.assertTrue(conditionLock.isHeldByCurrentThread());

        conditionLock.release();

        conditionLock.acquire();
    }

    /**
     * Wake up at most one thread sleeping on this condition variable. The
     * current thread must hold the associated lock.
     */
    public void wake() {
        Lib.assertTrue(conditionLock.isHeldByCurrentThread());
    }

    /**
     * Wake up all threads sleeping on this condition variable. The current
     * thread must hold the associated lock.
     */
    public void wakeAll() {
        Lib.assertTrue(conditionLock.isHeldByCurrentThread());
    }

    private Lock conditionLock;
}
```

```
package nachos.threads;

import nachos.machine.*;

/**
 * A controller for all the elevators in an elevator bank. The controller
 * accesses the elevator bank through an instance of <tt>ElevatorControls</tt>.
 */
public class ElevatorController implements ElevatorControllerInterface {
    /**
     * Allocate a new elevator controller.
     */
    public ElevatorController() {
    }

    /**
     * Initialize this elevator controller. The controller will access the
     * elevator bank through <i>controls</i>. This constructor should return
     * immediately after this controller is initialized, but not until the
     * interupt handler is set. The controller will start receiving events
     * after this method returns, but potentially before <tt>run()</tt> is
     * called.
     *
     * @param   controls        the controller's interface to the elevator
     *                          bank. The controler must not attempt to access
     *                          the elevator bank in <i>any</i> other way.
     */
    public void initialize(ElevatorControls controls) {
    }

    /**
     * Cause the controller to use the provided controls to receive and process
     * requests from riders. This method should not return, but instead should
     * call <tt>controls.finish()</tt> when the controller is finished.
     */
    public void run() {
    }
}
```

```
package nachos.threads;

import nachos.machine.*;

/**
 * A KThread is a thread that can be used to execute Nachos kernel code. Nachos
 * allows multiple threads to run concurrently.
 *
 * To create a new thread of execution, first declare a class that implements
 * the <tt>Runnable</tt> interface. That class then implements the <tt>run</tt>
 * method. An instance of the class can then be allocated, passed as an
 * argument when creating <tt>KThread</tt>, and forked. For example, a thread
 * that computes pi could be written as follows:
 *
 * <p><blockquote><pre>
 * class PiRun implements Runnable {
 *     public void run() {
 *         // compute pi
 *         ...
 *     }
 * }
 * </pre></blockquote>
 * <p>The following code would then create a thread and start it running:
 *
 * <p><blockquote><pre>
 * PiRun p = new PiRun();
 * new KThread(p).fork();
 * </pre></blockquote>
 */
public class KThread {
    /**
     * Get the current thread.
     *
     * @return  the current thread.
     */
    public static KThread currentThread() {
        Lib.assertTrue(currentThread != null);
        return currentThread;
    }

    /**
     * Allocate a new <tt>KThread</tt>. If this is the first <tt>KThread</tt>,
     * create an idle thread as well.
     */
    public KThread() {
        if (currentThread != null) {
            tcb = new TCB();
        }
        else {
            readyQueue = ThreadedKernel.scheduler.newThreadQueue(false);
            readyQueue.acquire(this);

            currentThread = this;
            tcb = TCB.currentTCB();
            name = "main";
            restoreState();

            createIdleThread();
        }
    }

    /**
     * Allocate a new KThread.
     *
     * @param   target  the object whose <tt>run</tt> method is called.
     */
    public KThread(Runnable target) {
        this();
        this.target = target;
    }

    /**
     * Set the target of this thread.
     *
     * @param   target  the object whose <tt>run</tt> method is called.
     * @return  this thread.
     */
    public KThread setTarget(Runnable target) {
        Lib.assertTrue(status == statusNew);

        this.target = target;
        return this;
    }

    /**
     * Set the name of this thread. This name is used for debugging purposes
     * only.
     *
     * @param   name    the name to give to this thread.
     * @return  this thread.
     */
    public KThread setName(String name) {
        this.name = name;
        return this;
    }

    /**
     * Get the name of this thread. This name is used for debugging purposes
     * only.
     *
     * @return  the name given to this thread.
     */
    public String getName() {
        return name;
    }

    /**
     * Get the full name of this thread. This includes its name along with its
     * numerical ID. This name is used for debugging purposes only.
     *
     * @return  the full name given to this thread.
     */
    public String toString() {
        return (name + " (#" + id + ")");
    }

    /**
     * Deterministically and consistently compare this thread to another
     * thread.
     */
    public int compareTo(Object o) {
        KThread thread = (KThread) o;

        if (id < thread.id)
```

```
        return -1;
    else if (id > thread.id)
        return 1;
    else
        return 0;
}

/**
 * Causes this thread to begin execution. The result is that two threads
 * are running concurrently: the current thread (which returns from the
 * call to the <tt>fork</tt> method) and the other thread (which executes
 * its target's <tt>run</tt> method).
 */
public void fork() {
    Lib.assertTrue(status == statusNew);
    Lib.assertTrue(target != null);

    Lib.debug(dbgThread,
              "Forking thread: " + toString() + " Runnable: " + target);

    boolean intStatus = Machine.interrupt().disable();

    tcb.start(new Runnable() {
            public void run() {
                runThread();
            }
        });

    ready();

    Machine.interrupt().restore(intStatus);
}

private void runThread() {
    begin();
    target.run();
    finish();
}

private void begin() {
    Lib.debug(dbgThread, "Beginning thread: " + toString());

    Lib.assertTrue(this == currentThread);

    restoreState();

    Machine.interrupt().enable();
}

/**
 * Finish the current thread and schedule it to be destroyed when it is
 * safe to do so. This method is automatically called when a thread's
 * <tt>run</tt> method returns, but it may also be called directly.
 *
 * The current thread cannot be immediately destroyed because its stack and
 * other execution state are still in use. Instead, this thread will be
 * destroyed automatically by the next thread to run, when it is safe to
 * delete this thread.
 */
public static void finish() {
    Lib.debug(dbgThread, "Finishing thread: " + currentThread.toString());

    Machine.interrupt().disable();

    Machine.autoGrader().finishingCurrentThread();

    Lib.assertTrue(toBeDestroyed == null);
    toBeDestroyed = currentThread;


    currentThread.status = statusFinished;

    sleep();
}

/**
 * Relinquish the CPU if any other thread is ready to run. If so, put the
 * current thread on the ready queue, so that it will eventually be
 * rescheuled.
 *
 * <p>
 * Returns immediately if no other thread is ready to run. Otherwise
 * returns when the current thread is chosen to run again by
 * <tt>readyQueue.nextThread()</tt>.
 *
 * <p>
 * Interrupts are disabled, so that the current thread can atomically add
 * itself to the ready queue and switch to the next thread. On return,
 * restores interrupts to the previous state, in case <tt>yield()</tt> was
 * called with interrupts disabled.
 */
public static void yield() {
    Lib.debug(dbgThread, "Yielding thread: " + currentThread.toString());

    Lib.assertTrue(currentThread.status == statusRunning);

    boolean intStatus = Machine.interrupt().disable();

    currentThread.ready();

    runNextThread();

    Machine.interrupt().restore(intStatus);
}

/**
 * Relinquish the CPU, because the current thread has either finished or it
 * is blocked. This thread must be the current thread.
 *
 * <p>
 * If the current thread is blocked (on a synchronization primitive, i.e.
 * a <tt>Semaphore</tt>, <tt>Lock</tt>, or <tt>Condition</tt>), eventually
 * some thread will wake this thread up, putting it back on the ready queue
 * so that it can be rescheduled. Otherwise, <tt>finish()</tt> should have
 * scheduled this thread to be destroyed by the next thread to run.
 */
public static void sleep() {
    Lib.debug(dbgThread, "Sleeping thread: " + currentThread.toString());

    Lib.assertTrue(Machine.interrupt().disabled());

    if (currentThread.status != statusFinished)
        currentThread.status = statusBlocked;
```

```
    runNextThread();
}

/**
 * Moves this thread to the ready state and adds this to the scheduler's
 * ready queue.
 */
public void ready() {
    Lib.debug(dbgThread, "Ready thread: " + toString());

    Lib.assertTrue(Machine.interrupt().disabled());
    Lib.assertTrue(status != statusReady);

    status = statusReady;
    if (this != idleThread)
        readyQueue.waitForAccess(this);

    Machine.autoGrader().readyThread(this);
}

/**
 * Waits for this thread to finish. If this thread is already finished,
 * return immediately. This method must only be called once; the second
 * call is not guaranteed to return. This thread must not be the current
 * thread.
 */
public void join() {
    Lib.debug(dbgThread, "Joining to thread: " + toString());

    Lib.assertTrue(this != currentThread);

}

/**
 * Create the idle thread. Whenever there are no threads ready to be run,
 * and <tt>runNextThread()</tt> is called, it will run the idle thread. The
 * idle thread must never block, and it will only be allowed to run when
 * all other threads are blocked.
 *
 * <p>
 * Note that <tt>ready()</tt> never adds the idle thread to the ready set.
 */
private static void createIdleThread() {
    Lib.assertTrue(idleThread == null);

    idleThread = new KThread(new Runnable() {
        public void run() { while (true) yield(); }
    });
    idleThread.setName("idle");

    Machine.autoGrader().setIdleThread(idleThread);

    idleThread.fork();
}

/**
 * Determine the next thread to run, then dispatch the CPU to the thread
 * using <tt>run()</tt>.
 */
private static void runNextThread() {
    KThread nextThread = readyQueue.nextThread();
    if (nextThread == null)
        nextThread = idleThread;

    nextThread.run();
}

/**
 * Dispatch the CPU to this thread. Save the state of the current thread,
 * switch to the new thread by calling <tt>TCB.contextSwitch()</tt>, and
 * load the state of the new thread. The new thread becomes the current
 * thread.
 *
 * <p>
 * If the new thread and the old thread are the same, this method must
 * still call <tt>saveState()</tt>, <tt>contextSwitch()</tt>, and
 * <tt>restoreState()</tt>.
 *
 * <p>
 * The state of the previously running thread must already have been
 * changed from running to blocked or ready (depending on whether the
 * thread is sleeping or yielding).
 *
 * @param   finishing       <tt>true</tt> if the current thread is
 *                          finished, and should be destroyed by the new
 *                          thread.
 */
private void run() {
    Lib.assertTrue(Machine.interrupt().disabled());

    Machine.yield();

    currentThread.saveState();

    Lib.debug(dbgThread, "Switching from: " + currentThread.toString()
              + " to: " + toString());

    currentThread = this;

    tcb.contextSwitch();

    currentThread.restoreState();
}

/**
 * Prepare this thread to be run. Set <tt>status</tt> to
 * <tt>statusRunning</tt> and check <tt>toBeDestroyed</tt>.
 */
protected void restoreState() {
    Lib.debug(dbgThread, "Running thread: " + currentThread.toString());

    Lib.assertTrue(Machine.interrupt().disabled());
    Lib.assertTrue(this == currentThread);
    Lib.assertTrue(tcb == TCB.currentTCB());

    Machine.autoGrader().runningThread(this);

    status = statusRunning;

    if (toBeDestroyed != null) {
        toBeDestroyed.tcb.destroy();
        toBeDestroyed.tcb = null;
        toBeDestroyed = null;
    }
```

```
    }

    /**
     * Prepare this thread to give up the processor. Kernel threads do not
     * need to do anything here.
     */
    protected void saveState() {
        Lib.assertTrue(Machine.interrupt().disabled());
        Lib.assertTrue(this == currentThread);
    }

    private static class PingTest implements Runnable {
        PingTest(int which) {
            this.which = which;
        }

        public void run() {
            for (int i=0; i<5; i++) {
                System.out.println("*** thread " + which + " looped "
                                   + i + " times");
                currentThread.yield();
            }
        }

        private int which;
    }

    /**
     * Tests whether this module is working.
     */
    public static void selfTest() {
        Lib.debug(dbgThread, "Enter KThread.selfTest");

        new KThread(new PingTest(1)).setName("forked thread").fork();
        new PingTest(0).run();
    }

    private static final char dbgThread = 't';

    /**
     * Additional state used by schedulers.
     *
     * @see     nachos.threads.PriorityScheduler.ThreadState
     */
    public Object schedulingState = null;

    private static final int statusNew = 0;
    private static final int statusReady = 1;
    private static final int statusRunning = 2;
    private static final int statusBlocked = 3;
    private static final int statusFinished = 4;

    /**
     * The status of this thread. A thread can either be new (not yet forked),
     * ready (on the ready queue but not running), running, or blocked (not
     * on the ready queue and not running).
     */
    private int status = statusNew;
    private String name = "(unnamed thread)";
    private Runnable target;
    private TCB tcb;

    /**
     * Unique identifer for this thread. Used to deterministically compare
     * threads.
     */
    private int id = numCreated++;
    /** Number of times the KThread constructor was called. */
    private static int numCreated = 0;

    private static ThreadQueue readyQueue = null;
    private static KThread currentThread = null;
    private static KThread toBeDestroyed = null;
    private static KThread idleThread = null;
}
```

```
package nachos.threads;

import nachos.machine.*;

/**
 * A <tt>Lock</tt> is a synchronization primitive that has two states,
 * <i>busy</i> and <i>free</i>. There are only two operations allowed on a
 * lock:
 *
 * <ul>
 * <li><tt>acquire()</tt>: atomically wait until the lock is <i>free</i> and
 * then set it to <i>busy</i>.
 * <li><tt>release()</tt>: set the lock to be <i>free</i>, waking up one
 * waiting thread if possible.
 * </ul>
 *
 * <p>
 * Also, only the thread that acquired a lock may release it. As with
 * semaphores, the API does not allow you to read the lock state (because the
 * value could change immediately after you read it).
 */
public class Lock {
    /**
     * Allocate a new lock. The lock will initially be <i>free</i>.
     */
    public Lock() {
    }

    /**
     * Atomically acquire this lock. The current thread must not already hold
     * this lock.
     */
    public void acquire() {
        Lib.assertTrue(!isHeldByCurrentThread());

        boolean intStatus = Machine.interrupt().disable();
        KThread thread = KThread.currentThread();

        if (lockHolder != null) {
            waitQueue.waitForAccess(thread);
            KThread.sleep();
        }
        else {
            waitQueue.acquire(thread);
            lockHolder = thread;
        }

        Lib.assertTrue(lockHolder == thread);

        Machine.interrupt().restore(intStatus);
    }

    /**
     * Atomically release this lock, allowing other threads to acquire it.
     */
    public void release() {
        Lib.assertTrue(isHeldByCurrentThread());

        boolean intStatus = Machine.interrupt().disable();

        if ((lockHolder = waitQueue.nextThread()) != null)
            lockHolder.ready();

        Machine.interrupt().restore(intStatus);
    }

    /**
     * Test if the current thread holds this lock.
     *
     * @return  true if the current thread holds this lock.
     */
    public boolean isHeldByCurrentThread() {
        return (lockHolder == KThread.currentThread());
    }

    private KThread lockHolder = null;
    private ThreadQueue waitQueue =
        ThreadedKernel.scheduler.newThreadQueue(true);
}
```

```
package nachos.threads;

import nachos.machine.*;

import java.util.TreeSet;
import java.util.HashSet;
import java.util.Iterator;

/**
 * A scheduler that chooses threads using a lottery.
 *
 * <p>
 * A lottery scheduler associates a number of tickets with each thread. When a
 * thread needs to be dequeued, a random lottery is held, among all the tickets
 * of all the threads waiting to be dequeued. The thread that holds the winning
 * ticket is chosen.
 *
 * <p>
 * Note that a lottery scheduler must be able to handle a lot of tickets
 * (sometimes billions), so it is not acceptable to maintain state for every
 * ticket.
 *
 * <p>
 * A lottery scheduler must partially solve the priority inversion problem; in
 * particular, tickets must be transferred through locks, and through joins.
 * Unlike a priority scheduler, these tickets add (as opposed to just taking
 * the maximum).
 */
public class LotteryScheduler extends PriorityScheduler {
    /**
     * Allocate a new lottery scheduler.
     */
    public LotteryScheduler() {
    }

    /**
     * Allocate a new lottery thread queue.
     *
     * @param   transferPriority        <tt>true</tt> if this queue should
     *                                  transfer tickets from waiting threads
     *                                  to the owning thread.
     * @return  a new lottery thread queue.
     */
    public ThreadQueue newThreadQueue(boolean transferPriority) {
        // implement me
        return null;
    }
}
```

```
package nachos.threads;

import nachos.machine.*;

import java.util.TreeSet;
import java.util.HashSet;
import java.util.Iterator;

/**
 * A scheduler that chooses threads based on their priorities.
 *
 * <p>
 * A priority scheduler associates a priority with each thread. The next thread
 * to be dequeued is always a thread with priority no less than any other
 * waiting thread's priority. Like a round-robin scheduler, the thread that is
 * dequeued is, among all the threads of the same (highest) priority, the
 * thread that has been waiting longest.
 *
 * <p>
 * Essentially, a priority scheduler gives access in a round-robin fassion to
 * all the highest-priority threads, and ignores all other threads. This has
 * the potential to
 * starve a thread if there's always a thread waiting with higher priority.
 *
 * <p>
 * A priority scheduler must partially solve the priority inversion problem; in
 * particular, priority must be donated through locks, and through joins.
 */
public class PriorityScheduler extends Scheduler {
    /**
     * Allocate a new priority scheduler.
     */
    public PriorityScheduler() {
    }

    /**
     * Allocate a new priority thread queue.
     *
     * @param   transferPriority        <tt>true</tt> if this queue should
     *                                  transfer priority from waiting threads
     *                                  to the owning thread.
     * @return  a new priority thread queue.
     */
    public ThreadQueue newThreadQueue(boolean transferPriority) {
        return new PriorityQueue(transferPriority);
    }

    public int getPriority(KThread thread) {
        Lib.assertTrue(Machine.interrupt().disabled());

        return getThreadState(thread).getPriority();
    }

    public int getEffectivePriority(KThread thread) {
        Lib.assertTrue(Machine.interrupt().disabled());

        return getThreadState(thread).getEffectivePriority();
    }

    public void setPriority(KThread thread, int priority) {
        Lib.assertTrue(Machine.interrupt().disabled());

        Lib.assertTrue(priority >= priorityMinimum &&
                   priority <= priorityMaximum);

        getThreadState(thread).setPriority(priority);
    }

    public boolean increasePriority() {
        boolean intStatus = Machine.interrupt().disable();

        KThread thread = KThread.currentThread();

        int priority = getPriority(thread);
        if (priority == priorityMaximum)
            return false;

        setPriority(thread, priority+1);

        Machine.interrupt().restore(intStatus);
        return true;
    }

    public boolean decreasePriority() {
        boolean intStatus = Machine.interrupt().disable();

        KThread thread = KThread.currentThread();

        int priority = getPriority(thread);
        if (priority == priorityMinimum)
            return false;

        setPriority(thread, priority-1);

        Machine.interrupt().restore(intStatus);
        return true;
    }

    /**
     * The default priority for a new thread. Do not change this value.
     */
    public static final int priorityDefault = 1;
    /**
     * The minimum priority that a thread can have. Do not change this value.
     */
    public static final int priorityMinimum = 0;
    /**
     * The maximum priority that a thread can have. Do not change this value.
     */
    public static final int priorityMaximum = 7;

    /**
     * Return the scheduling state of the specified thread.
     *
     * @param   thread  the thread whose scheduling state to return.
     * @return  the scheduling state of the specified thread.
     */
    protected ThreadState getThreadState(KThread thread) {
        if (thread.schedulingState == null)
            thread.schedulingState = new ThreadState(thread);

        return (ThreadState) thread.schedulingState;
    }
```

```
/**
 * A <tt>ThreadQueue</tt> that sorts threads by priority.
 */
protected class PriorityQueue extends ThreadQueue {
    PriorityQueue(boolean transferPriority) {
        this.transferPriority = transferPriority;
    }

    public void waitForAccess(KThread thread) {
        Lib.assertTrue(Machine.interrupt().disabled());
        getThreadState(thread).waitForAccess(this);
    }

    public void acquire(KThread thread) {
        Lib.assertTrue(Machine.interrupt().disabled());
        getThreadState(thread).acquire(this);
    }

    public KThread nextThread() {
        Lib.assertTrue(Machine.interrupt().disabled());
        // implement me
        return null;
    }

    /**
     * Return the next thread that <tt>nextThread()</tt> would return,
     * without modifying the state of this queue.
     *
     * @return      the next thread that <tt>nextThread()</tt> would
     *              return.
     */
    protected ThreadState pickNextThread() {
        // implement me
        return null;
    }

    public void print() {
        Lib.assertTrue(Machine.interrupt().disabled());
        // implement me (if you want)
    }

    /**
     * <tt>true</tt> if this queue should transfer priority from waiting
     * threads to the owning thread.
     */
    public boolean transferPriority;
}

/**
 * The scheduling state of a thread. This should include the thread's
 * priority, its effective priority, any objects it owns, and the queue
 * it's waiting for, if any.
 *
 * @see     nachos.threads.KThread#schedulingState
 */
protected class ThreadState {
    /**
     * Allocate a new <tt>ThreadState</tt> object and associate it with the
     * specified thread.
     *
     * @param       thread  the thread this state belongs to.
     */
    public ThreadState(KThread thread) {
        this.thread = thread;

        setPriority(priorityDefault);
    }

    /**
     * Return the priority of the associated thread.
     *
     * @return      the priority of the associated thread.
     */
    public int getPriority() {
        return priority;
    }

    /**
     * Return the effective priority of the associated thread.
     *
     * @return      the effective priority of the associated thread.
     */
    public int getEffectivePriority() {
        // implement me
        return priority;
    }

    /**
     * Set the priority of the associated thread to the specified value.
     *
     * @param       priority        the new priority.
     */
    public void setPriority(int priority) {
        if (this.priority == priority)
            return;

        this.priority = priority;

        // implement me
    }

    /**
     * Called when <tt>waitForAccess(thread)</tt> (where <tt>thread</tt> is
     * the associated thread) is invoked on the specified priority queue.
     * The associated thread is therefore waiting for access to the
     * resource guarded by <tt>waitQueue</tt>. This method is only called
     * if the associated thread cannot immediately obtain access.
     *
     * @param       waitQueue       the queue that the associated thread is
     *                              now waiting on.
     *
     * @see nachos.threads.ThreadQueue#waitForAccess
     */
    public void waitForAccess(PriorityQueue waitQueue) {
        // implement me
    }

    /**
     * Called when the associated thread has acquired access to whatever is
     * guarded by <tt>waitQueue</tt>. This can occur either as a result of
     * <tt>acquire(thread)</tt> being invoked on <tt>waitQueue</tt> (where
     * <tt>thread</tt> is the associated thread), or as a result of
     * <tt>nextThread()</tt> being invoked on <tt>waitQueue</tt>.
     *
```

```
         * @see nachos.threads.ThreadQueue#acquire
         * @see nachos.threads.ThreadQueue#nextThread
         */
        public void acquire(PriorityQueue waitQueue) {
            // implement me
        }

        /** The thread with which this object is associated. */
        protected KThread thread;
        /** The priority of the associated thread. */
        protected int priority;
    }
}
```

```
package nachos.threads;

import nachos.machine.*;

/**
 * A single rider. Each rider accesses the elevator bank through an
 * instance of <tt>RiderControls</tt>.
 */
public class Rider implements RiderInterface {
    /**
     * Allocate a new rider.
     */
    public Rider() {
    }

    /**
     * Initialize this rider. The rider will access the elevator bank through
     * <i>controls</i>, and the rider will make stops at different floors as
     * specified in <i>stops</i>. This method should return immediately after
     * this rider is initialized, but not until the interrupt handler is
     * set. The rider will start receiving events after this method returns,
     * potentially before <tt>run()</tt> is called.
     *
     * @param   controls        the rider's interface to the elevator bank. The
     *                          rider must not attempt to access the elevator
     *                          bank in <i>any</i> other way.
     * @param   stops           an array of stops the rider should make; see
     *                          below.
     */
    public void initialize(RiderControls controls, int[] stops) {
    }

    /**
     * Cause the rider to use the provided controls to make the stops specified
     * in the constructor. The rider should stop at each of the floors in
     * <i>stops</i>, an array of floor numbers. The rider should <i>only</i>
     * make the specified stops.
     *
     * <p>
     * For example, suppose the rider uses <i>controls</i> to determine that
     * it is initially on floor 1, and suppose the stops array contains two
     * elements: { 0, 2 }. Then the rider should get on an elevator, get off
     * on floor 0, get on an elevator, and get off on floor 2, pushing buttons
     * as necessary.
     *
     * <p>
     * This method should not return, but instead should call
     * <tt>controls.finish()</tt> when the rider is finished.
     */
    public void run() {
    }
}
```

```
package nachos.threads;

import nachos.machine.*;

import java.util.LinkedList;
import java.util.Iterator;

/**
 * A round-robin scheduler tracks waiting threads in FIFO queues, implemented
 * with linked lists. When a thread begins waiting for access, it is appended
 * to the end of a list. The next thread to receive access is always the first
 * thread in the list. This causes access to be given on a first-come
 * first-serve basis.
 */
public class RoundRobinScheduler extends Scheduler {
    /**
     * Allocate a new round-robin scheduler.
     */
    public RoundRobinScheduler() {
    }

    /**
     * Allocate a new FIFO thread queue.
     *
     * @param   transferPriority        ignored. Round robin schedulers have
     *                                  no priority.
     * @return  a new FIFO thread queue.
     */
    public ThreadQueue newThreadQueue(boolean transferPriority) {
        return new FifoQueue();
    }

    private class FifoQueue extends ThreadQueue {
        /**
         * Add a thread to the end of the wait queue.
         *
         * @param       thread  the thread to append to the queue.
         */
        public void waitForAccess(KThread thread) {
            Lib.assertTrue(Machine.interrupt().disabled());

            waitQueue.add(thread);
        }

        /**
         * Remove a thread from the beginning of the queue.
         *
         * @return      the first thread on the queue, or <tt>null</tt> if the
         *              queue is
         *              empty.
         */
        public KThread nextThread() {
            Lib.assertTrue(Machine.interrupt().disabled());

            if (waitQueue.isEmpty())
                return null;

            return (KThread) waitQueue.removeFirst();
        }

        /**
         * The specified thread has received exclusive access, without using
         * <tt>waitForAccess()</tt> or <tt>nextThread()</tt>. Assert that no
         * threads are waiting for access.
         */
        public void acquire(KThread thread) {
            Lib.assertTrue(Machine.interrupt().disabled());

            Lib.assertTrue(waitQueue.isEmpty());
        }

        /**
         * Print out the contents of the queue.
         */
        public void print() {
            Lib.assertTrue(Machine.interrupt().disabled());

            for (Iterator i=waitQueue.iterator(); i.hasNext(); )
                System.out.print((KThread) i.next() + " ");
        }

        private LinkedList<KThread> waitQueue = new LinkedList<KThread>();
    }
}
```

```
package nachos.threads;

import nachos.machine.*;

/**
 * Coordinates a group of thread queues of the same kind.
 *
 * @see nachos.threads.ThreadQueue
 */
public abstract class Scheduler {
    /**
     * Allocate a new scheduler.
     */
    public Scheduler() {
    }

    /**
     * Allocate a new thread queue. If <i>transferPriority</i> is
     * <tt>true</tt>, then threads waiting on the new queue will transfer their
     * "priority" to the thread that has access to whatever is being guarded by
     * the queue. This is the mechanism used to partially solve priority
     * inversion.
     *
     * <p>
     * If there is no definite thread that can be said to have "access" (as in
     * the case of semaphores and condition variables), this parameter should
     * be <tt>false</tt>, indicating that no priority should be transferred.
     *
     * <p>
     * The processor is a special case. There is clearly no purpose to donating
     * priority to a thread that already has the processor. When the processor
     * wait queue is created, this parameter should be <tt>false</tt>.
     *
     * <p>
     * Otherwise, it is beneficial to donate priority. For example, a lock has
     * a definite owner (the thread that holds the lock), and a lock is always
     * released by the same thread that acquired it, so it is possible to help
     * a high priority thread waiting for a lock by donating its priority to
     * the thread holding the lock. Therefore, a queue for a lock should be
     * created with this parameter set to <tt>true</tt>.
     *
     * <p>
     * Similarly, when a thread is asleep in <tt>join()</tt> waiting for the
     * target thread to finish, the sleeping thread should donate its priority
     * to the target thread. Therefore, a join queue should be created with
     * this parameter set to <tt>true</tt>.
     *
     * @param   transferPriority        <tt>true</tt> if the thread that has
     *                                  access should receive priority from the
     *                                  threads that are waiting on this queue.
     * @return  a new thread queue.
     */
    public abstract ThreadQueue newThreadQueue(boolean transferPriority);

    /**
     * Get the priority of the specified thread. Must be called with
     * interrupts disabled.
     *
     * @param   thread  the thread to get the priority of.
     * @return  the thread's priority.
     */
    public int getPriority(KThread thread) {
        Lib.assertTrue(Machine.interrupt().disabled());
        return 0;
    }

    /**
     * Get the priority of the current thread. Equivalent to
     * <tt>getPriority(KThread.currentThread())</tt>.
     *
     * @return  the current thread's priority.
     */
    public int getPriority() {
        return getPriority(KThread.currentThread());
    }

    /**
     * Get the effective priority of the specified thread. Must be called with
     * interrupts disabled.
     *
     * <p>
     * The effective priority of a thread is the priority of a thread after
     * taking into account priority donations.
     *
     * <p>
     * For a priority scheduler, this is the maximum of the thread's priority
     * and the priorities of all other threads waiting for the thread through a
     * lock or a join.
     *
     * <p>
     * For a lottery scheduler, this is the sum of the thread's tickets and the
     * tickets of all other threads waiting for the thread through a lock or a
     * join.
     *
     * @param   thread  the thread to get the effective priority of.
     * @return  the thread's effective priority.
     */
    public int getEffectivePriority(KThread thread) {
        Lib.assertTrue(Machine.interrupt().disabled());
        return 0;
    }

    /**
     * Get the effective priority of the current thread. Equivalent to
     * <tt>getEffectivePriority(KThread.currentThread())</tt>.
     *
     * @return  the current thread's priority.
     */
    public int getEffectivePriority() {
        return getEffectivePriority(KThread.currentThread());
    }

    /**
     * Set the priority of the specified thread. Must be called with interrupts
     * disabled.
     *
     * @param   thread  the thread to set the priority of.
     * @param   priority        the new priority.
     */
    public void setPriority(KThread thread, int priority) {
        Lib.assertTrue(Machine.interrupt().disabled());
    }

    /**
```

```
     * Set the priority of the current thread. Equivalent to
     * <tt>setPriority(KThread.currentThread(), priority)</tt>.
     *
     * @param   priority        the new priority.
     */
    public void setPriority(int priority) {
        setPriority(KThread.currentThread(), priority);
    }

    /**
     * If possible, raise the priority of the current thread in some
     * scheduler-dependent way.
     *
     * @return  <tt>true</tt> if the scheduler was able to increase the current
     *          thread's
     *          priority.
     */
    public boolean increasePriority() {
        return false;
    }

    /**
     * If possible, lower the priority of the current thread user in some
     * scheduler-dependent way, preferably by the same amount as would a call
     * to <tt>increasePriority()</tt>.
     *
     * @return  <tt>true</tt> if the scheduler was able to decrease the current
     *          thread's priority.
     */
    public boolean decreasePriority() {
        return false;
    }
}
```

```
package nachos.threads;

import nachos.machine.*;

/**
 * A <tt>Semaphore</tt> is a synchronization primitive with an unsigned value.
 * A semaphore has only two operations:
 *
 * <ul>
 * <li><tt>P()</tt>: waits until the semaphore's value is greater than zero,
 * then decrements it.
 * <li><tt>V()</tt>: increments the semaphore's value, and wakes up one thread
 * waiting in <tt>P()</tt> if possible.
 * </ul>
 *
 * <p>
 * Note that this API does not allow a thread to read the value of the
 * semaphore directly. Even if you did read the value, the only thing you would
 * know is what the value used to be. You don't know what the value is now,
 * because by the time you get the value, a context switch might have occurred,
 * and some other thread might have called <tt>P()</tt> or <tt>V()</tt>, so the
 * true value might now be different.
 */
public class Semaphore {
    /**
     * Allocate a new semaphore.
     *
     * @param   initialValue    the initial value of this semaphore.
     */
    public Semaphore(int initialValue) {
        value = initialValue;
    }

    /**
     * Atomically wait for this semaphore to become non-zero and decrement it.
     */
    public void P() {
        boolean intStatus = Machine.interrupt().disable();

        if (value == 0) {
            waitQueue.waitForAccess(KThread.currentThread());
            KThread.sleep();
        }
        else {
            value--;
        }

        Machine.interrupt().restore(intStatus);
    }

    /**
     * Atomically increment this semaphore and wake up at most one other thread
     * sleeping on this semaphore.
     */
    public void V() {
        boolean intStatus = Machine.interrupt().disable();

        KThread thread = waitQueue.nextThread();
        if (thread != null) {
            thread.ready();
        }
        else {
            value++;
        }

        Machine.interrupt().restore(intStatus);
    }

    private static class PingTest implements Runnable {
        PingTest(Semaphore ping, Semaphore pong) {
            this.ping = ping;
            this.pong = pong;
        }

        public void run() {
            for (int i=0; i<10; i++) {
                ping.P();
                pong.V();
            }
        }

        private Semaphore ping;
        private Semaphore pong;
    }

    /**
     * Test if this module is working.
     */
    public static void selfTest() {
        Semaphore ping = new Semaphore(0);
        Semaphore pong = new Semaphore(0);

        new KThread(new PingTest(ping, pong)).setName("ping").fork();

        for (int i=0; i<10; i++) {
            ping.V();
            pong.P();
        }
    }

    private int value;
    private ThreadQueue waitQueue =
        ThreadedKernel.scheduler.newThreadQueue(false);
}
```

```
package nachos.threads;

import java.util.LinkedList;
import nachos.machine.*;
import nachos.threads.*;

/**
 * A synchronized queue.
 */
public class SynchList {
    /**
     * Allocate a new synchronized queue.
     */
    public SynchList() {
        list = new LinkedList<Object>();
        lock = new Lock();
        listEmpty = new Condition(lock);
    }

    /**
     * Add the specified object to the end of the queue. If another thread is
     * waiting in <tt>removeFirst()</tt>, it is woken up.
     *
     * @param   o       the object to add. Must not be <tt>null</tt>.
     */
    public void add(Object o) {
        Lib.assertTrue(o != null);

        lock.acquire();
        list.add(o);
        listEmpty.wake();
        lock.release();
    }

    /**
     * Remove an object from the front of the queue, blocking until the queue
     * is non-empty if necessary.
     *
     * @return  the element removed from the front of the queue.
     */
    public Object removeFirst() {
        Object o;

        lock.acquire();
        while (list.isEmpty())
            listEmpty.sleep();
        o = list.removeFirst();
        lock.release();

        return o;
    }

    private static class PingTest implements Runnable {
        PingTest(SynchList ping, SynchList pong) {
            this.ping = ping;
            this.pong = pong;
        }

        public void run() {
            for (int i=0; i<10; i++)
                pong.add(ping.removeFirst());
        }

        private SynchList ping;
        private SynchList pong;
    }

    /**
     * Test that this module is working.
     */
    public static void selfTest() {
        SynchList ping = new SynchList();
        SynchList pong = new SynchList();

        new KThread(new PingTest(ping, pong)).setName("ping").fork();

        for (int i=0; i<10; i++) {
            Integer o = new Integer(i);
            ping.add(o);
            Lib.assertTrue(pong.removeFirst() == o);
        }
    }

    private LinkedList<Object> list;
    private Lock lock;
    private Condition listEmpty;
}
```

```
package nachos.threads;

/**
 * Schedules access to some sort of resource with limited access constraints. A
 * thread queue can be used to share this limited access among multiple
 * threads.
 *
 * <p>
 * Examples of limited access in Nachos include:
 *
 * <ol>
 * <li>the right for a thread to use the processor. Only one thread may run on
 * the processor at a time.
 *
 * <li>the right for a thread to acquire a specific lock. A lock may be held by
 * only one thread at a time.
 *
 * <li>the right for a thread to return from <tt>Semaphore.P()</tt> when the
 * semaphore is 0. When another thread calls <tt>Semaphore.V()</tt>, only one
 * thread waiting in <tt>Semaphore.P()</tt> can be awakened.
 *
 * <li>the right for a thread to be woken while sleeping on a condition
 * variable. When another thread calls <tt>Condition.wake()</tt>, only one
 * thread sleeping on the condition variable can be awakened.
 *
 * <li>the right for a thread to return from <tt>KThread.join()</tt>. Threads
 * are not allowed to return from <tt>join()</tt> until the target thread has
 * finished.
 * </ol>
 *
 * All these cases involve limited access because, for each of them, it is not
 * necessarily possible (or correct) for all the threads to have simultaneous
 * access. Some of these cases involve concrete resources (e.g. the processor,
 * or a lock); others are more abstract (e.g. waiting on semaphores, condition
 * variables, or join).
 *
 * <p>
 * All thread queue methods must be invoked with <b>interrupts disabled</b>.
 */
public abstract class ThreadQueue {
    /**
     * Notify this thread queue that the specified thread is waiting for
     * access. This method should only be called if the thread cannot
     * immediately obtain access (e.g. if the thread wants to acquire a lock
     * but another thread already holds the lock).
     *
     * <p>
     * A thread must not simultaneously wait for access to multiple resources.
     * For example, a thread waiting for a lock must not also be waiting to run
     * on the processor; if a thread is waiting for a lock it should be
     * sleeping.
     *
     * <p>
     * However, depending on the specific objects, it may be acceptable for a
     * thread to wait for access to one object while having access to another.
     * For example, a thread may attempt to acquire a lock while holding
     * another lock. Note, though, that the processor cannot be held while
     * waiting for access to anything else.
     *
     * @param   thread  the thread waiting for access.
     */
    public abstract void waitForAccess(KThread thread);

    /**
     * Notify this thread queue that another thread can receive access. Choose
     * and return the next thread to receive access, or <tt>null</tt> if there
     * are no threads waiting.
     *
     * <p>
     * If the limited access object transfers priority, and if there are other
     * threads waiting for access, then they will donate priority to the
     * returned thread.
     *
     * @return  the next thread to receive access, or <tt>null</tt> if there
     *          are no threads waiting.
     */
    public abstract KThread nextThread();

    /**
     * Notify this thread queue that a thread has received access, without
     * going through <tt>request()</tt> and <tt>nextThread()</tt>. For example,
     * if a thread acquires a lock that no other threads are waiting for, it
     * should call this method.
     *
     * <p>
     * This method should not be called for a thread returned from
     * <tt>nextThread()</tt>.
     *
     * @param   thread  the thread that has received access, but was not
     *                  returned from <tt>nextThread()</tt>.
     */
    public abstract void acquire(KThread thread);

    /**
     * Print out all the threads waiting for access, in no particular order.
     */
    public abstract void print();
}
```

```
package nachos.threads;

import nachos.machine.*;

/**
 * A multi-threaded OS kernel.
 */
public class ThreadedKernel extends Kernel {
    /**
     * Allocate a new multi-threaded kernel.
     */
    public ThreadedKernel() {
        super();
    }

    /**
     * Initialize this kernel. Creates a scheduler, the first thread, and an
     * alarm, and enables interrupts. Creates a file system if necessary.
     */
    public void initialize(String[] args) {
        // set scheduler
        String schedulerName = Config.getString("ThreadedKernel.scheduler");
        scheduler = (Scheduler) Lib.constructObject(schedulerName);

        // set fileSystem
        String fileSystemName = Config.getString("ThreadedKernel.fileSystem");
        if (fileSystemName != null)
            fileSystem = (FileSystem) Lib.constructObject(fileSystemName);
        else if (Machine.stubFileSystem() != null)
            fileSystem = Machine.stubFileSystem();
        else
            fileSystem = null;

        // start threading
        new KThread(null);

        alarm  = new Alarm();

        Machine.interrupt().enable();
    }

    /**
     * Test this kernel. Test the <tt>KThread</tt>, <tt>Semaphore</tt>,
     * <tt>SynchList</tt>, and <tt>ElevatorBank</tt> classes. Note that the
     * autograder never calls this method, so it is safe to put additional
     * tests here.
     */
    public void selfTest() {
        KThread.selfTest();
        Semaphore.selfTest();
        SynchList.selfTest();
        if (Machine.bank() != null) {
            ElevatorBank.selfTest();
        }
    }

    /**
     * A threaded kernel does not run user programs, so this method does
     * nothing.
     */
    public void run() {
    }

    /**
     * Terminate this kernel. Never returns.
     */
    public void terminate() {
        Machine.halt();
    }

    /** Globally accessible reference to the scheduler. */
    public static Scheduler scheduler = null;
    /** Globally accessible reference to the alarm. */
    public static Alarm alarm = null;
    /** Globally accessible reference to the file system. */
    public static FileSystem fileSystem = null;

    // dummy variables to make javac smarter
    private static RoundRobinScheduler dummy1 = null;
    private static PriorityScheduler dummy2 = null;
    private static LotteryScheduler dummy3 = null;
    private static Condition2 dummy4 = null;
    private static Communicator dummy5 = null;
    private static Rider dummy6 = null;
    private static ElevatorController dummy7 = null;
}
```

```
package nachos.userprog;

import nachos.machine.*;
import nachos.threads.*;
import nachos.userprog.*;

/**
 * Provides a simple, synchronized interface to the machine's console. The
 * interface can also be accessed through <tt>OpenFile</tt> objects.
 */
public class SynchConsole {
    /**
     * Allocate a new <tt>SynchConsole</tt>.
     *
     * @param   console the underlying serial console to use.
     */
    public SynchConsole(SerialConsole console) {
        this.console = console;

        Runnable receiveHandler = new Runnable() {
            public void run() { receiveInterrupt(); }
        };
        Runnable sendHandler = new Runnable() {
            public void run() { sendInterrupt(); }
        };
        console.setInterruptHandlers(receiveHandler, sendHandler);
    }

    /**
     * Return the next unsigned byte received (in the range <tt>0</tt> through
     * <tt>255</tt>). If a byte has not arrived at, blocks until a byte
     * arrives, or returns immediately, depending on the value of <i>block</i>.
     *
     * @param   block   <tt>true</tt> if <tt>readByte()</tt> should wait for a
     *                  byte if none is available.
     * @return  the next byte read, or -1 if <tt>block</tt> was <tt>false</tt>
     *          and no byte was available.
     */
    public int readByte(boolean block) {
        int value;
        boolean intStatus = Machine.interrupt().disable();
        readLock.acquire();

        if (block || charAvailable) {
            charAvailable = false;
            readWait.P();

            value = console.readByte();
            Lib.assertTrue(value != -1);
        }
        else {
            value = -1;
        }

        readLock.release();
        Machine.interrupt().restore(intStatus);
        return value;
    }

    /**
     * Return an <tt>OpenFile</tt> that can be used to read this as a file.
     *
     * @return  a file that can read this console.
     */
    public OpenFile openForReading() {
        return new File(true, false);
    }

    private void receiveInterrupt() {
        charAvailable = true;
        readWait.V();
    }

    /**
     * Send a byte. Blocks until the send is complete.
     *
     * @param   value   the byte to be sent (the upper 24 bits are ignored).
     */
    public void writeByte(int value) {
        writeLock.acquire();
        console.writeByte(value);
        writeWait.P();
        writeLock.release();
    }

    /**
     * Return an <tt>OpenFile</tt> that can be used to write this as a file.
     *
     * @return  a file that can write this console.
     */
    public OpenFile openForWriting() {
        return new File(false, true);
    }

    private void sendInterrupt() {
        writeWait.V();
    }

    private boolean charAvailable = false;

    private SerialConsole console;
    private Lock readLock = new Lock();
    private Lock writeLock = new Lock();
    private Semaphore readWait = new Semaphore(0);
    private Semaphore writeWait = new Semaphore(0);

    private class File extends OpenFile {
        File(boolean canRead, boolean canWrite) {
            super(null, "SynchConsole");

            this.canRead = canRead;
            this.canWrite = canWrite;
        }

        public void close() {
            canRead = canWrite = false;
        }

        public int read(byte[] buf, int offset, int length) {
            if (!canRead)
                return 0;

            int i;
            for (i=0; i<length; i++) {
```

```
                int value = SynchConsole.this.readByte(false);
                if (value == -1)
                    break;

                buf[offset+i] = (byte) value;
            }

            return i;
        }

        public int write(byte[] buf, int offset, int length) {
            if (!canWrite)
                return 0;

            for (int i=0; i<length; i++)
                SynchConsole.this.writeByte(buf[offset+i]);

            return length;
        }

        private boolean canRead, canWrite;
    }
}
```

```
package nachos.userprog;

import nachos.machine.*;
import nachos.threads.*;
import nachos.userprog.*;

/**
 * A UThread is KThread that can execute user program code inside a user
 * process, in addition to Nachos kernel code.
 */
public class UThread extends KThread {
    /**
     * Allocate a new UThread.
     */
    public UThread(UserProcess process) {
        super();

        setTarget(new Runnable() {
                public void run() {
                    runProgram();
                }
            });

        this.process = process;
    }

    private void runProgram() {
        process.initRegisters();
        process.restoreState();

        Machine.processor().run();

        Lib.assertNotReached();
    }

    /**
     * Save state before giving up the processor to another thread.
     */
    protected void saveState() {
        process.saveState();

        for (int i=0; i<Processor.numUserRegisters; i++)
            userRegisters[i] = Machine.processor().readRegister(i);

        super.saveState();
    }

    /**
     * Restore state before receiving the processor again.
     */
    protected void restoreState() {
        super.restoreState();

        for (int i=0; i<Processor.numUserRegisters; i++)
            Machine.processor().writeRegister(i, userRegisters[i]);

        process.restoreState();
    }

    /**
     * Storage for the user register set.
     *
     * <p>
     * A thread capable of running user code actually has <i>two</i> sets of
     * CPU registers: one for its state while executing user code, and one for
     * its state while executing kernel code. While this thread is not running,
     * its user state is stored here.
     */
    public int userRegisters[] = new int[Processor.numUserRegisters];

    /**
     * The process to which this thread belongs.
     */
    public UserProcess process;
}
```

```
package nachos.userprog;

import nachos.machine.*;
import nachos.threads.*;
import nachos.userprog.*;

/**
 * A kernel that can support multiple user processes.
 */
public class UserKernel extends ThreadedKernel {
    /**
     * Allocate a new user kernel.
     */
    public UserKernel() {
        super();
    }

    /**
     * Initialize this kernel. Creates a synchronized console and sets the
     * processor's exception handler.
     */
    public void initialize(String[] args) {
        super.initialize(args);

        console = new SynchConsole(Machine.console());

        Machine.processor().setExceptionHandler(new Runnable() {
                public void run() { exceptionHandler(); }
            });
    }

    /**
     * Test the console device.
     */
    public void selfTest() {
        super.selfTest();

        System.out.println("Testing the console device. Typed characters");
        System.out.println("will be echoed until q is typed.");

        char c;

        do {
            c = (char) console.readByte(true);
            console.writeByte(c);
        }
        while (c != 'q');

        System.out.println("");
    }

    /**
     * Returns the current process.
     *
     * @return  the current process, or <tt>null</tt> if no process is current.
     */
    public static UserProcess currentProcess() {
        if (!(KThread.currentThread() instanceof UThread))
            return null;

        return ((UThread) KThread.currentThread()).process;
    }

    /**
     * The exception handler. This handler is called by the processor whenever
     * a user instruction causes a processor exception.
     *
     * <p>
     * When the exception handler is invoked, interrupts are enabled, and the
     * processor's cause register contains an integer identifying the cause of
     * the exception (see the <tt>exceptionZZZ</tt> constants in the
     * <tt>Processor</tt> class). If the exception involves a bad virtual
     * address (e.g. page fault, TLB miss, read-only, bus error, or address
     * error), the processor's BadVAddr register identifies the virtual address
     * that caused the exception.
     */
    public void exceptionHandler() {
        Lib.assertTrue(KThread.currentThread() instanceof UThread);

        UserProcess process = ((UThread) KThread.currentThread()).process;
        int cause = Machine.processor().readRegister(Processor.regCause);
        process.handleException(cause);
    }

    /**
     * Start running user programs, by creating a process and running a shell
     * program in it. The name of the shell program it must run is returned by
     * <tt>Machine.getShellProgramName()</tt>.
     *
     * @see     nachos.machine.Machine#getShellProgramName
     */
    public void run() {
        super.run();

        UserProcess process = UserProcess.newUserProcess();

        String shellProgram = Machine.getShellProgramName();
        Lib.assertTrue(process.execute(shellProgram, new String[] { }));

        KThread.currentThread().finish();
    }

    /**
     * Terminate this kernel. Never returns.
     */
    public void terminate() {
        super.terminate();
    }

    /** Globally accessible reference to the synchronized console. */
    public static SynchConsole console;

    // dummy variables to make javac smarter
    private static Coff dummy1 = null;
}
```

```
package nachos.userprog;

import nachos.machine.*;
import nachos.threads.*;
import nachos.userprog.*;

import java.io.EOFException;

/**
 * Encapsulates the state of a user process that is not contained in its
 * user thread (or threads). This includes its address translation state, a
 * file table, and information about the program being executed.
 *
 * <p>
 * This class is extended by other classes to support additional functionality
 * (such as additional syscalls).
 *
 * @see nachos.vm.VMProcess
 * @see nachos.network.NetProcess
 */
public class UserProcess {
    /**
     * Allocate a new process.
     */
    public UserProcess() {
        int numPhysPages = Machine.processor().getNumPhysPages();
        pageTable = new TranslationEntry[numPhysPages];
        for (int i=0; i<numPhysPages; i++)
            pageTable[i] = new TranslationEntry(i,i, true,false,false,false);
    }

    /**
     * Allocate and return a new process of the correct class. The class name
     * is specified by the <tt>nachos.conf</tt> key
     * <tt>Kernel.processClassName</tt>.
     *
     * @return  a new process of the correct class.
     */
    public static UserProcess newUserProcess() {
        return (UserProcess)Lib.constructObject(Machine.getProcessClassName());
    }

    /**
     * Execute the specified program with the specified arguments. Attempts to
     * load the program, and then forks a thread to run it.
     *
     * @param   name    the name of the file containing the executable.
     * @param   args    the arguments to pass to the executable.
     * @return  <tt>true</tt> if the program was successfully executed.
     */
    public boolean execute(String name, String[] args) {
        if (!load(name, args))
            return false;

        new UThread(this).setName(name).fork();

        return true;
    }

    /**
     * Save the state of this process in preparation for a context switch.
     * Called by <tt>UThread.saveState()</tt>.
     */
    public void saveState() {
    }

    /**
     * Restore the state of this process after a context switch. Called by
     * <tt>UThread.restoreState()</tt>.
     */
    public void restoreState() {
        Machine.processor().setPageTable(pageTable);
    }

    /**
     * Read a null-terminated string from this process's virtual memory. Read
     * at most <tt>maxLength + 1</tt> bytes from the specified address, search
     * for the null terminator, and convert it to a <tt>java.lang.String</tt>,
     * without including the null terminator. If no null terminator is found,
     * returns <tt>null</tt>.
     *
     * @param   vaddr   the starting virtual address of the null-terminated
     *                  string.
     * @param   maxLength       the maximum number of characters in the string,
     *                          not including the null terminator.
     * @return  the string read, or <tt>null</tt> if no null terminator was
     *          found.
     */
    public String readVirtualMemoryString(int vaddr, int maxLength) {
        Lib.assertTrue(maxLength >= 0);

        byte[] bytes = new byte[maxLength+1];

        int bytesRead = readVirtualMemory(vaddr, bytes);

        for (int length=0; length<bytesRead; length++) {
            if (bytes[length] == 0)
                return new String(bytes, 0, length);
        }

        return null;
    }

    /**
     * Transfer data from this process's virtual memory to all of the specified
     * array. Same as <tt>readVirtualMemory(vaddr, data, 0, data.length)</tt>.
     *
     * @param   vaddr   the first byte of virtual memory to read.
     * @param   data    the array where the data will be stored.
     * @return  the number of bytes successfully transferred.
     */
    public int readVirtualMemory(int vaddr, byte[] data) {
        return readVirtualMemory(vaddr, data, 0, data.length);
    }

    /**
     * Transfer data from this process's virtual memory to the specified array.
     * This method handles address translation details. This method must
     * <i>not</i> destroy the current process if an error occurs, but instead
     * should return the number of bytes successfully copied (or zero if no
     * data could be copied).
     *
     * @param   vaddr   the first byte of virtual memory to read.
     * @param   data    the array where the data will be stored.
```

```
     * @param   offset  the first byte to write in the array.
     * @param   length  the number of bytes to transfer from virtual memory to
     *                  the array.
     * @return  the number of bytes successfully transferred.
     */
    public int readVirtualMemory(int vaddr, byte[] data, int offset,
                                 int length) {
        Lib.assertTrue(offset >= 0 && length >= 0 && offset+length <= data.length);

        byte[] memory = Machine.processor().getMemory();

        // for now, just assume that virtual addresses equal physical addresses
        if (vaddr < 0 || vaddr >= memory.length)
            return 0;

        int amount = Math.min(length, memory.length-vaddr);
        System.arraycopy(memory, vaddr, data, offset, amount);

        return amount;
    }

    /**
     * Transfer all data from the specified array to this process's virtual
     * memory.
     * Same as <tt>writeVirtualMemory(vaddr, data, 0, data.length)</tt>.
     *
     * @param   vaddr   the first byte of virtual memory to write.
     * @param   data    the array containing the data to transfer.
     * @return  the number of bytes successfully transferred.
     */
    public int writeVirtualMemory(int vaddr, byte[] data) {
        return writeVirtualMemory(vaddr, data, 0, data.length);
    }

    /**
     * Transfer data from the specified array to this process's virtual memory.
     * This method handles address translation details. This method must
     * <i>not</i> destroy the current process if an error occurs, but instead
     * should return the number of bytes successfully copied (or zero if no
     * data could be copied).
     *
     * @param   vaddr   the first byte of virtual memory to write.
     * @param   data    the array containing the data to transfer.
     * @param   offset  the first byte to transfer from the array.
     * @param   length  the number of bytes to transfer from the array to
     *                  virtual memory.
     * @return  the number of bytes successfully transferred.
     */
    public int writeVirtualMemory(int vaddr, byte[] data, int offset,
                                  int length) {
        Lib.assertTrue(offset >= 0 && length >= 0 && offset+length <= data.length);

        byte[] memory = Machine.processor().getMemory();

        // for now, just assume that virtual addresses equal physical addresses
        if (vaddr < 0 || vaddr >= memory.length)
            return 0;

        int amount = Math.min(length, memory.length-vaddr);
        System.arraycopy(data, offset, memory, vaddr, amount);

        return amount;
    }

    /**
     * Load the executable with the specified name into this process, and
     * prepare to pass it the specified arguments. Opens the executable, reads
     * its header information, and copies sections and arguments into this
     * process's virtual memory.
     *
     * @param   name    the name of the file containing the executable.
     * @param   args    the arguments to pass to the executable.
     * @return  <tt>true</tt> if the executable was successfully loaded.
     */
    private boolean load(String name, String[] args) {
        Lib.debug(dbgProcess, "UserProcess.load(\"" + name + "\")");

        OpenFile executable = ThreadedKernel.fileSystem.open(name, false);
        if (executable == null) {
            Lib.debug(dbgProcess, "\topen failed");
            return false;
        }

        try {
            coff = new Coff(executable);
        }
        catch (EOFException e) {
            executable.close();
            Lib.debug(dbgProcess, "\tcoff load failed");
            return false;
        }

        // make sure the sections are contiguous and start at page 0
        numPages = 0;
        for (int s=0; s<coff.getNumSections(); s++) {
            CoffSection section = coff.getSection(s);
            if (section.getFirstVPN() != numPages) {
                coff.close();
                Lib.debug(dbgProcess, "\tfragmented executable");
                return false;
            }
            numPages += section.getLength();
        }

        // make sure the argv array will fit in one page
        byte[][] argv = new byte[args.length][];
        int argsSize = 0;
        for (int i=0; i<args.length; i++) {
            argv[i] = args[i].getBytes();
            // 4 bytes for argv[] pointer; then string plus one for null byte
            argsSize += 4 + argv[i].length + 1;
        }
        if (argsSize > pageSize) {
            coff.close();
            Lib.debug(dbgProcess, "\targuments too long");
            return false;
        }

        // program counter initially points at the program entry point
        initialPC = coff.getEntryPoint();

        // next comes the stack; stack pointer initially points to top of it
        numPages += stackPages;
        initialSP = numPages*pageSize;
```

```
    // and finally reserve 1 page for arguments
    numPages++;

    if (!loadSections())
        return false;

    // store arguments in last page
    int entryOffset = (numPages-1)*pageSize;
    int stringOffset = entryOffset + args.length*4;

    this.argc = args.length;
    this.argv = entryOffset;

    for (int i=0; i<argv.length; i++) {
        byte[] stringOffsetBytes = Lib.bytesFromInt(stringOffset);
        Lib.assertTrue(writeVirtualMemory(entryOffset,stringOffsetBytes) == 4);
        entryOffset += 4;
        Lib.assertTrue(writeVirtualMemory(stringOffset, argv[i]) ==
                   argv[i].length);
        stringOffset += argv[i].length;
        Lib.assertTrue(writeVirtualMemory(stringOffset,new byte[] { 0 }) == 1);
        stringOffset += 1;
    }

    return true;
}

/**
 * Allocates memory for this process, and loads the COFF sections into
 * memory. If this returns successfully, the process will definitely be
 * run (this is the last step in process initialization that can fail).
 *
 * @return  <tt>true</tt> if the sections were successfully loaded.
 */
protected boolean loadSections() {
    if (numPages > Machine.processor().getNumPhysPages()) {
        coff.close();
        Lib.debug(dbgProcess, "\tinsufficient physical memory");
        return false;
    }

    // load sections
    for (int s=0; s<coff.getNumSections(); s++) {
        CoffSection section = coff.getSection(s);

        Lib.debug(dbgProcess, "\tinitializing " + section.getName()
                  + " section (" + section.getLength() + " pages)");

        for (int i=0; i<section.getLength(); i++) {
            int vpn = section.getFirstVPN()+i;

            // for now, just assume virtual addresses=physical addresses
            section.loadPage(i, vpn);
        }
    }

    return true;
}

/**
 * Release any resources allocated by <tt>loadSections()</tt>.
 */
protected void unloadSections() {
}

/**
 * Initialize the processor's registers in preparation for running the
 * program loaded into this process. Set the PC register to point at the
 * start function, set the stack pointer register to point at the top of
 * the stack, set the A0 and A1 registers to argc and argv, respectively,
 * and initialize all other registers to 0.
 */
public void initRegisters() {
    Processor processor = Machine.processor();

    // by default, everything's 0
    for (int i=0; i<processor.numUserRegisters; i++)
        processor.writeRegister(i, 0);

    // initialize PC and SP according
    processor.writeRegister(Processor.regPC, initialPC);
    processor.writeRegister(Processor.regSP, initialSP);

    // initialize the first two argument registers to argc and argv
    processor.writeRegister(Processor.regA0, argc);
    processor.writeRegister(Processor.regA1, argv);
}

/**
 * Handle the halt() system call.
 */
private int handleHalt() {

    Machine.halt();

    Lib.assertNotReached("Machine.halt() did not halt machine!");
    return 0;
}


private static final int
    syscallHalt = 0,
    syscallExit = 1,
    syscallExec = 2,
    syscallJoin = 3,
    syscallCreate = 4,
    syscallOpen = 5,
    syscallRead = 6,
    syscallWrite = 7,
    syscallClose = 8,
    syscallUnlink = 9;

/**
 * Handle a syscall exception. Called by <tt>handleException()</tt>. The
 * <i>syscall</i> argument identifies which syscall the user executed:
 *
 * <table>
 * <tr><td>syscall#</td><td>syscall prototype</td></tr>
 * <tr><td>0</td><td><tt>void halt();</tt></td></tr>
 * <tr><td>1</td><td><tt>void exit(int status);</tt></td></tr>
 * <tr><td>2</td><td><tt>int  exec(char *name, int argc, char **argv);
 *                                                              </tt></td></tr>
 * <tr><td>3</td><td><tt>int  join(int pid, int *status);</tt></td></tr>
```

```
     * <tr><td>4</td><td><tt>int  creat(char *name);</tt></td></tr>
     * <tr><td>5</td><td><tt>int  open(char *name);</tt></td></tr>
     * <tr><td>6</td><td><tt>int  read(int fd, char *buffer, int size);
     *                                                </tt></td></tr>
     * <tr><td>7</td><td><tt>int  write(int fd, char *buffer, int size);
     *                                                </tt></td></tr>
     * <tr><td>8</td><td><tt>int  close(int fd);</tt></td></tr>
     * <tr><td>9</td><td><tt>int  unlink(char *name);</tt></td></tr>
     * </table>
     *
     * @param   syscall the syscall number.
     * @param   a0      the first syscall argument.
     * @param   a1      the second syscall argument.
     * @param   a2      the third syscall argument.
     * @param   a3      the fourth syscall argument.
     * @return  the value to be returned to the user.
     */
    public int handleSyscall(int syscall, int a0, int a1, int a2, int a3) {
        switch (syscall) {
        case syscallHalt:
            return handleHalt();


        default:
            Lib.debug(dbgProcess, "Unknown syscall " + syscall);
            Lib.assertNotReached("Unknown system call!");
        }
        return 0;
    }

    /**
     * Handle a user exception. Called by
     * <tt>UserKernel.exceptionHandler()</tt>. The
     * <i>cause</i> argument identifies which exception occurred; see the
     * <tt>Processor.exceptionZZZ</tt> constants.
     *
     * @param   cause   the user exception that occurred.
     */
    public void handleException(int cause) {
        Processor processor = Machine.processor();

        switch (cause) {
        case Processor.exceptionSyscall:
            int result = handleSyscall(processor.readRegister(Processor.regV0),
                                       processor.readRegister(Processor.regA0),
                                       processor.readRegister(Processor.regA1),
                                       processor.readRegister(Processor.regA2),
                                       processor.readRegister(Processor.regA3)
                                       );
            processor.writeRegister(Processor.regV0, result);
            processor.advancePC();
            break;

        default:
            Lib.debug(dbgProcess, "Unexpected exception: " +
                      Processor.exceptionNames[cause]);
            Lib.assertNotReached("Unexpected exception");
        }
    }

    /** The program being run by this process. */
    protected Coff coff;

    /** This process's page table. */
    protected TranslationEntry[] pageTable;
    /** The number of contiguous pages occupied by the program. */
    protected int numPages;

    /** The number of pages in the program's stack. */
    protected final int stackPages = 8;

    private int initialPC, initialSP;
    private int argc, argv;

    private static final int pageSize = Processor.pageSize;
    private static final char dbgProcess = 'a';
}
```

```
package nachos.vm;

import nachos.machine.*;
import nachos.threads.*;
import nachos.userprog.*;
import nachos.vm.*;

/**
 * A kernel that can support multiple demand-paging user processes.
 */
public class VMKernel extends UserKernel {
    /**
     * Allocate a new VM kernel.
     */
    public VMKernel() {
        super();
    }

    /**
     * Initialize this kernel.
     */
    public void initialize(String[] args) {
        super.initialize(args);
    }

    /**
     * Test this kernel.
     */
    public void selfTest() {
        super.selfTest();
    }

    /**
     * Start running user programs.
     */
    public void run() {
        super.run();
    }

    /**
     * Terminate this kernel. Never returns.
     */
    public void terminate() {
        super.terminate();
    }

    // dummy variables to make javac smarter
    private static VMProcess dummy1 = null;

    private static final char dbgVM = 'v';
}
```

```
package nachos.vm;

import nachos.machine.*;
import nachos.threads.*;
import nachos.userprog.*;
import nachos.vm.*;

/**
 * A <tt>UserProcess</tt> that supports demand-paging.
 */
public class VMProcess extends UserProcess {
    /**
     * Allocate a new process.
     */
    public VMProcess() {
        super();
    }

    /**
     * Save the state of this process in preparation for a context switch.
     * Called by <tt>UThread.saveState()</tt>.
     */
    public void saveState() {
        super.saveState();
    }

    /**
     * Restore the state of this process after a context switch. Called by
     * <tt>UThread.restoreState()</tt>.
     */
    public void restoreState() {
        super.restoreState();
    }

    /**
     * Initializes page tables for this process so that the executable can be
     * demand-paged.
     *
     * @return  <tt>true</tt> if successful.
     */
    protected boolean loadSections() {
        return super.loadSections();
    }

    /**
     * Release any resources allocated by <tt>loadSections()</tt>.
     */
    protected void unloadSections() {
        super.unloadSections();
    }

    /**
     * Handle a user exception. Called by
     * <tt>UserKernel.exceptionHandler()</tt>. The
     * <i>cause</i> argument identifies which exception occurred; see the
     * <tt>Processor.exceptionZZZ</tt> constants.
     *
     * @param   cause   the user exception that occurred.
     */
    public void handleException(int cause) {
        Processor processor = Machine.processor();
        switch (cause) {
        default:
            super.handleException(cause);
            break;
        }
    }

    private static final int pageSize = Processor.pageSize;
    private static final char dbgProcess = 'a';
    private static final char dbgVM = 'v';
}
```

A Guide to Nachos 5.0j

Dan Hettena

Rick Cox

rick@rescomp.berkeley.edu

We have ported the Nachos instructional operating system [1 (http://http.cs.berkeley.edu/~tea/nachos/nachos.ps)] to Java, and in the process of doing so, many details changed (hopefully for the better). [2 (http://www.cs.duke.edu/~narten/110/nachos/main/main.html)] remains an excellent resource for learning about the C++ versions of Nachos, but an update is necessary to account for the differences between the Java version and the C++ versions.

We attempt to describe Nachos 5.0j in the same way that [2 (http://www.cs.duke.edu/~narten/110/nachos/main/main.html)] described previous versions of Nachos, except that we defer some of the details to the Javadoc-generated documentation. We do not claim any originality in this documentation, and freely offer any deserved credit to Narten.

Table of Contents

1. Nachos and the Java Port

The Nachos instructional operating system, developed at Berkeley, was first tested on guinea pig students in 1992 [1 (http://http.cs.berkeley.edu/~tea/nachos/nachos.ps)]. The authors intended it to be a simple, yet realistic, project for undergraduate operating systems classes. Nachos is now in wide use.

The original Nachos, written in a subset of C++ (with a little assembly), ran as a regular UNIX process. It simulated the hardware devices of a simple computer: it had a timer, a console, a MIPS R3000 processor, a disk, and a network link. In order to achieve reasonable performance, the operating system kernel ran natively, while user processes ran on the simulated processor. Because it was simulated, multiple Nachos instances could run on the same physical computer.

1.1. Why Java?

Despite the success of Nachos, there are good reasons to believe that it would be more useful in Java:

- Java is much simpler than C++. It is not necessary to restrict Nachos to a subset of the language; students can understand the whole language.
- Java is type-safe. C++ is not type-safe; it is possible for a C++ program to perform a legal operation (e.g. writing off the end of an array) such that the operation of the program can no longer be described in terms of the C++ language. This turns out to be a major problem; some project groups are unable to debug their projects within the alotted time, primarily because of bugs not at all related to operating systems concepts.
- It is much more reasonable to machine-grade a Java project than a C++ project.
- Many undergraduate data structures classes, including the one at Berkeley, now use Java, not C++; students know Java well.
- Java is relatively portable. Nachos 4.0 uses unportable assembly to support multithreading. Adding a new target to Nachos 4.0 required writing a bit of additional code for the port.

1.2. Will it work?

One of the first concerns many people have about Java is its speed. It is an undebatable fact that Java programs run slower than their C++ equivalents. This statement can be misleading, though:

- Compiling is a significant part of the Nachos 4.0 debug cycle. Because javac compiles as much as it can everytime it is invoked, Nachos 5.0j actually compiles faster than Nachos 4.0 (running on a local disk partition with no optimizations enabled).
- Generating large files on network partitions further slows down the debug cycle. Nachos 5.0j's .class files are significantly smaller than Nachos 4.0's .o files, even when compiling with -Os. This is in part due to C++ templates, which, without a smart compiler or careful management, get very big.
- Type-safe languages are widely known to make debugging cycles more effective.

Another common concern is that writing an operating system in a type-safe language is unrealistic. In short, it *is* unrealistic, but not as unrealistic as you might think. Two aspects of real operating systems are lost by using Java, but neither are critical:

- Since the JVM provides threads for Nachos 5.0j, the context switch code is no longer exposed. In Nachos 4.0, students could read the assembly code used to switch between threads. But, as mentioned above, this posed a portability problem.
- The kernel can allocate kernel memory without releasing it; the garbage collector will release it. In Linux, this would be similar to removing all calls to `kfree`. This, however, is conceptually one of the simplest forms of resource allocation within the kernel (there's a lot more to Linux than `kmalloc` and `kfree`). The Nachos kernel must still directly manage the allocation of physical pages among processes, and must close files when processes exit, for example.

2. Nachos Machine

Nachos simulates a real CPU and harware devices, including interrupts and memory management. The Java package nachos.machine provides this simulation.

2.1. Configuring Nachos

The nachos simulation is configured for the various projects using the `nachos.conf` file (for the most part, this file is equivelant to the BIOS or OpenFirmware configuration of modern PCs or Macintoshes). It specifies which hardware devices to include in the simulation as well as which Nachos kernel to use. The project directories include appropriate configurations, and, where neccessary, the project handouts document any changes to this file required to complete the project.

2.2. Boot Process

The nachos boot process is similar to that of a real machine. An instance of the nachos.machine.Machine class is created to begin booting. The hardware (Machine object) first initializes the devices including the interrupt controller, timer, elevator controller, MIPS processor, console, and file system.

The Machine object then hands control to the particular AutoGrader in use, an action equivelant to loading the bootstrap code from the boot sector of the disk. It is the AutoGrader that creates a Nachos kernel, starting the operating system. Students need not worry about this step in the boot process - the interesting part begins with the kernel.

A Nachos kernel is just a subclass of nachos.machine.Kernel. For instance, the thread project uses nachos.threads.ThreadedKernel (and later projects inherit from ThreadedKernel).

2.3. Nachos Hardware Devices

The Nachos machine simulation includes several hardware devices. Some would be found in most modern computers (e.g. the network interface), while others (such as the elevator controller) are unique to Nachos. Most classes in the `machine` directory are part of the hardware simulation, while all classes outside that directory are part of the Nachos operating system.

2.3.1. Interrupt Management

The nachos.machine.Interrupt class simulates interrupts by maintaining an event queue together with a simulated clock. As the clock ticks, the event queue is examined to find events scheduled to take place now. The interrupt controller is returned by `Machine.interrupt()`.

The clock is maintained entirely in software and ticks only under the following conditions:

- Every time interrupts are re-enabled (i.e. only when interrupts are disabled and get enabled again), the clock advances 10 ticks. Nachos code frequently disables and restores interrupts for mutual exclusion purposes by making explicit calls to `disable()` and `restore()`.
- Whenever the MIPS simulator executes one instruction, the clock advances one tick.

> **Note:** Nachos C++ users: Nachos C++ allowed the simulated time to be advanced to that of the next interrupt whenever the ready list is empty. This provides a small performance gain, but it creates unnatural interaction between the kernel and the hardware, and it is unnecessary (a normal OS uses an idle thread, and this is exactly what Nachos does now).

Whenever the clock advances, the event queue is examined and any pending interrupt events are serviced by invoking the device event handler associated with the event. Note that this handler is *not* an interrupt handler (a.k.a. interrupt service routine). Interrupt handlers are part of software, while device event handlers are part of the hardware simulation. A device event handler will *invoke* the software interrupt handler for the device, as we will see later. For this reason, the Interrupt class disables interrupts before calling a device event handler.

Caution

Due to a bug in the current release of Nachos, *only the timer interrupt handler may cause a context switch* (the problem is that a few device event handlers are not reentrant; in order for an interrupt handler to be allowed to do a context switch, the device event handler that invoked it must be reentrant). All interrupt handlers besides the timer interrupt handler must not directly or indirectly cause a context switch before returning, or deadlock may occur. However, you probably won't even want to context switch in any other interrupt handler anyway, so this should not be a problem.

The Interrupt class accomplishes the above through three methods. These methods are only accessible to hardware simulation devices.

- `schedule()` takes a time and a device event handler as arguments, and schedules the specified handler to be called at the specified time.
- `tick()` advances the time by 1 tick or 10 ticks, depending on whether Nachos is in user mode or kernel mode. It is called by `setStatus()` whenever interrupts go from being disabled to being enabled, and also by `Processor.run()` after each user instruction is executed.
- `checkIfDue()` invokes event handlers for queued events until no more events are due to occur. It is invoked by `tick()`.

The Interrupt class also simulates the hardware interface to enable and disable interrupts (see the Javadoc for Interrupt).

The remainder of the hardware devices present in Nachos depend on the Interrupt device. No hardware devices in Nachos create threads, thus, the only time the code in the device classes execute is due to a function call by the running KThread or due to an interrupt handler executed by the Interrupt object.

2.3.2. Timer

Nachos provides an instance of a Timer to simulate a real-time clock, generating interrupts at regular intervals. It is implemented using the event driven interrupt mechanism described above. `Machine.timer()` returns a reference to this timer.

Timer supports only two operations:

- `getTime()` returns the number of ticks since Nachos started.
- `setInterruptHandler()` sets the timer interrupt handler, which is invoked by the simulated timer approximately every `Stats.TimerTicks` ticks.

The timer can be used to provide preemption. Note however that the timer interrupts do not always occur at exactly the same intervals. Do not rely on timer interrupts being equally spaced; instead, use `getTime()`.

2.3.3. Serial Console

Nachos provides three classes of I/O devices with read/write interfaces, of which the simplest is the serial console. The serial console, specified by the SerialConsole class, simulates the behavior of a serial port. It provides byte-wide read and write primitives that never block. The machine's serial console is returned by `Machine.console()`.

The read operation tests if a byte of data is ready to be returned. If so, it returns the byte immediately, and otherwise it returns -1. When another byte of data is received, a receive interrupt occurs. Only one byte can be queued at a time, so it is not possible for two receive interrupts to occur without an intervening read operation.

The write operation starts transmitting a byte of data and returns immediately. When the transmission is complete and another byte can be sent, a send interrupt occurs. If two writes occur without an intervening send interrupt, the actual data transmitted is undefined (so the kernel should always wait for a send interrupt first).

Note that the receive interrupt handler and send interrupt handler are provided by the kernel, by calling `setInterruptHandlers()`.

Implementation note: in a normal Nachos session, the serial console is implemented by class StandardConsole, which uses stdin and stdout. It schedules a read device event every `Stats.ConsoleTime` ticks to poll stdin for another byte of data. If a byte is present, it stores it and invokes the receive interrupt handler.

2.3.4. Disk

The file systems project has not yet been ported, so the disk has not been tested.

2.3.5. Network Link

Separate Nachos instances running on the same real-life machine can communicate with each other over a network, using the NetworkLink class. An instance of this class is returned by `Machine.networkLink()`.

The network link's interface is similar to the serial console's interface, except that instead of receiving and sending bytes at a time, the network link receives and sends packets at a time. Packets are instances of the Packet class.

Each network link has a *link address*, a number that uniquely identifies the link on the network. The link address is returned by `getLinkAddress()`.

A packet consists of a header and some data bytes. The header specifies the link address of the machine sending the packet (the source link address), the link address of the machine to which the packet is being sent (the destination link address), and the number of bytes of data contained in the packet. The data bytes are not analyzed by the network hardware, while the header is. When a link transmits a packet, it transmits it only to the link specified in the destination link address field of the header. Note that the source address can be forged.

The remainder of the interface to NetworkLink is equivalent to that of SerialConsole. The kernel can check for a packet by calling `receive()`, which returns `null` if no packet is available. Whenever a packet arrives, a receive interrupt is generated. The kernel can send a packet by calling `send()`, but it must wait for a send interrupt before attempting to send another packet.

3. Threads and Scheduling

Nachos provides a kernel threading package, allowing multiple tasks to run concurrently (see nachos.threads.ThreadedKernel and nachos.threads.KThread). Once the user-prcesses are implmented (phase 2), some threads may be running the MIPS processor simulation. As the scheduler and thread package are concerned, there is no difference between a thread running the MIPS simulation and one running just kernel Java code.

3.1. Thread Package

All Nachos threads are instances of nachos.threads.KThread (threads capable of running user-level MIPS code are a subclass of KThread, nachos.userprog.UThread). A nachos.machine.TCB object is contained by each KThread and provides low-level support for context switches, thread creation, thread destruction, and thread yield.

Every KThread has a `status` member that tracks the state of the thread. Certain KThread methods will fail (with a `Lib.assert()`) if called on threads in the wrong state; check the KThread Javadoc for details.

`statusNew`

A newly created, yet to be forked thread.

`statusReady`

A thread waiting for access to the CPU. `KThread.ready()` will add the thread to the ready queue and set the status to `statusReady`.

`statusRunning`

The thread currently using the CPU. `KThread.restoreState()` is responsible for setting `status` to `statusRunning`, and is called by `KThread.runNextThread()`.

`statusBlocked`

A thread which is asleep (as set by `KThread.sleep()`), waiting on some resource besides the CPU.

`statusFinished`

A thread scheduled for destruction. Use `KThread.finish()` to set this status.

Internally, Nachos implements threading using a Java thread for each TCB. The Java threads are synchronized by the TCBs such that exactly one is running at any given time. This provides the illusion of context switches saving state for the current thread and loading the saved state of the new thread. This detail, however, is only important for use with debuggers (which will show multiple Java threads), as the behavior is equivelant to a context switch on a real processor.

3.2. Scheduler

A sub-class (specified in the `nachos.conf`) of the abstract base class nachos.threads.Scheduler is responsible for scheduling threads for all limited resources, be it the CPU, a synchronization construct like a lock, or even a thread join operation. For each resource a nachos.threads.ThreadQueue is created by `Scheduler.newThreadQueue()`. The implementation of the resource (e.g. nachos.threads.Semaphore class) is responsible for adding KThreads to the ThreadQueue (`ThreadQueue.waitForAccess()`) and requesting the ThreadQueue return the next thread (`ThreadQueue.nextThread()`). Thus, all scheduling decisions (including those regarding the CPU's ready queue) reduce to the selection of the next thread by the ThreadQueue objects [1].

Various phases of the project will require modifications to the scheduler base class. The nachos.threads.RoundRobinScheduler is the default, and implements a fully functional (though naive) FIFO scheduler. Phase 1 of the projects requires the student to complete the nachos.threads.PriorityScheduler; for phase 2, students complete nachos.threads.LotteryScheduler.

3.3. Creating the First Thread

Upto the point where the Kernel is created, the boot process is fairly easy to follow - Nachos is just making Java objects, same as any other Java program. Also like any other single-threaded Java program, Nachos code is executing on the initial Java thread created automaticaly for it by Java. `ThreadedKernel.initialize()` has the task of starting threading:

```
public void initialize(String[] args) {
      ...
      // start threading
      new KThread(null);
      ...
}
```

The first clue that something special is happening should be that the new KThread object created is not stored in a variable inside `initialize()`. The constructor for KThread follows the following procedure the first time it is called:

1. Create the ready queue (`ThreadedKernel.scheduler.newThreadQueue()`).
2. Allocate the CPU to the new KThread object being created (`readyQueue.acquire(this)`).
3. Set `KThread.currentThread` to the new KThread being made.
4. Set the TCB object of the new KThread to `TCB.currentTCB()`. In doing so, the currently running Java thread is assigned to the new KThread object being created.
5. Change the `status` of the new KThread from the default (`statusNew`) to `statusRunning`. This bypasses the `statusReady` state.
6. Create an idle thread.
    a. Make another new KThread, with the target set to an infinite `yield()` loop.
    b. Fork the idle thread off from the main thread.

After this procedure, there are two KThread objects, each with a TCB object (one for the main thread, and one for the idle thread). The main thread is not special - the scheduler treats it exactly like any other KThread. The main thread can create other threads, it can die, it can block. The Nachos session will not end until all KThreads finish, regardless of whether the main thread is alive.

For the most part the idle thread is also a normal thread, which can be contexted switched like any other. The only difference is it will never be added to the ready queue (`KThread.ready()` has an explicit check for the idle thread). Instead, if `readyQueue.nextThread()` returns `null`, the thread system will switch to the idle thread.

Note: While the Nachos idle thread does nothing but `yield()` forever, some systems use the idle thread to do work. One common use is zeroing memory to prepare it for reallocation.

3.4. Creating More Threads

Creating subsequent threads is much simpler. As described in the KThread Javadoc, a new KThread is created, passing the constructor a Runnable object. Then, `fork()` is called:

```
KThread newThread = new KThread(myRunnable);
...
newThread.fork();
```

This sequence results in the new thread being placed on the ready queue. The currently running thread does not immediatly yield, however.

3.4.1. Java Anonymous Classes in Nachos

The Nachos source is relatively clean, using only basic Java, with the exception of the use of anonymous classes to replicate the functionality of function pointers in C++. The following code illustrates the use of an anonymous class to create a new KThread object which, when forked, will execute the `myFunction()` method of the encolosing object.

```
Runnable myRunnable = new Runnable() {
public void run() {
    myFunction();
}
    };
KThread newThread = new KThread(myRunnable);
```

This code creates a new object of type Runnable inside the context of the enclosing object. Since `myRunnable` has no method `myFunction()`, executing `myRunnable.run()` will cause Java to look in the enclosing class for a `myFunction()` method.

3.5. On Thread Death

All threads have some resources allocated to them which are neccessary for the thread to run (e.g. the TCB object). Since the thread itself cannot deallocate these resources while it is running, it leaves a virtual will asking the next thread which runs to deallocate its resources. This is implemented in

`KThread.finish()`, which sets `KThread.toBeDestroyed` to the currently running thread. It then sets current thread's `status` field to `statusFinished` and calls `sleep()`.

Since the thread is not waiting on a ThreadQueue object, its sleep will be permanent (that is, Nachos will never try to wake the thread). This scheme does, however, require that after every context switch, the newly running thread must check `toBeDestroyed`.

> **Note:** In the C++ version of Nachos, thread death was complicated by the explicit memory deallocation required, combined with dangling references that still pointing to the thread after death (for example, most thread `join()` implementations requires some reference to the thread). In Java, the garbage collector is responsible for noticing when these references are detached, significantly simplifying the thread finishing process.

4. The Nachos Simulated MIPS Machine

Nachos simulates a machine with a processor that roughly approximates the MIPS architecture. In addition, an event-driven simulated clock provides a mechanism to schedule events and execute them at a later time. This is a building block for classes that simulate various hardware devices: a timer, an elevator bank, a console, a disk, and a network link.

The simulated MIPS processor can execute arbitrary programs. One simply loads instructions into the processor's memory, initializes registers (including the program counter, regPC) and then tells the processor to start executing instructions. The processor then fetches the instruction that regPC points at, decodes it, and executes it. The process is repeated indefinitely, until either an instruction causes an exception or a hardware interrupt is generated. When an exception or interrupt takes place, execution of MIPS instructions is suspended, and a Nachos interrupt service routine is invoked to deal with the condition.

Conceptually, Nachos has two modes of execution, one of which is the MIPS simulator. Nachos executes user-level processes by loading them into the simulator's memory, initializing the simulator's registers and then running the simulator. User-programs can only access the memory associated with the simulated processor. The second mode corresponds to the Nachos "kernel". The kernel executes when Nachos first starts up, or when a user-program executes an instruction that causes an exception (e.g., illegal instruction, page fault, system call, etc.). In kernel mode, Nachos executes the way normal Java programs execute. That is, the statements corresponding to the Nachos source code are executed, and the memory accessed corresponds to the memory assigned to Nachos variables.

4.1. Processor Components

The Nachos/MIPS processor is implemented by the Processor class, an instance of which is created when Nachos first starts up. The Processor class exports a number of public methods and fields that the Nachos kernel accesses directly. In the following, we describe some of the important variables of the Processor class; describing their role helps explain what the simulated hardware does.

The processor provides registers and physical memory, and supports virtual memory. It provides operations to run the machine and to examine and modify its current state. When Nachos first starts up, it creates an instance of the Processor class and makes it available through `Machine.processor()`. The following aspects of the processor are accessible to the Nachos kernel:

Registers

> The processor's registers are accessible through `readRegister()` and `writeRegister()`. The registers include MIPS registers 0 through 31, the low and high registers used for multiplication and division, the program counter and next program counter registers (two are necessary because of branch delay slots), a register specifying the cause of the most recent exception, and a register specifying the virtual memory address associated with the most recent exception. Recall that the stack pointer register and return address registers are general MIPS registers (specifically, they are registers 29 and 31, respectively). Recall also that r0 is always 0 and cannot be modified.

Physical memory

> Memory is byte-addressable and organized into 1-kilobyte pages, the same size as disk sectors. A reference to the main memory array is returned by `getMemory()`. Memory corresponding to physical address `m` can be accessed in Nachos at `Machine.processor().getMemory()[m]`. The number of pages of physical memory is returned by `getNumPhysPages()`.

Virtual memory

> The processor supports VM through either a single linear page table or a software-managed TLB (but not both). The mode of address translation is actually used is determined by `nachos.conf`, and is returned by `hasTLB()`. If the processor does not have a TLB, the kernel can tell it what page table to use by calling `setPageTable()`. If the processor does have a TLB, the kernel can query the size of the TLB by calling `getTLBSize()`, and the kernel can read and write TLB entries by calling `readTLBEntry()` and `writeTLBEntry()`.

Exceptions

> When the processor attempts to execute an instruction and it results in an exception, the kernel exception handler is invoked. The kernel must tell the processor where this exception handler is by invoking `setExceptionHandler()`. If the exception resulted from a syscall instruction, it is the kernel's responsibility to advance the PC register, which it should do by calling `advancePC()`.

At this point, we know enough about the Processor class to explain how it executes arbitrary user programs. First, we load the program's instructions into the processor's physical memory (i.e. the array returned by `getMemory()`). Next, we initialize the processor's page table and registers. Finally, we invoke `run()`, which begins the fetch-execute cycle for the processor.

`run()` causes the processor to enter an infinite fetch-execute loop. This method should only be called after the registers and memory have been properly initialized. Each iteration of the loop does three things:

1. It attempts to run an instruction. This should be very familiar to students who have studied the generic 5-stage MIPS pipeline. Note that when an exception occurs, the pipline is aborted.
    a. The 32-bit instruction is fetched from memory, by reading the word of virtual memory pointed to by the PC register. Reading virtual memory can cause an exception.
    b. The instruction is decoded by looking at its 6-bit `op` field and looking up the meaning of the instruction in one of three tables.
    c. The instruction is executed, and data memory reads and writes occur. An exception can occur if an arithmetic error occurs, if the instruction is invalid, if the instruction was a syscall, or if a memory operand could not be accessed.
    d. The registers are modified to reflect the completion of the instruction.
2. If an exception occurred, handle it. The cause of the exception is written to the cause register, and if the exception involved a bad virtual address, this address is written to the bad virtual address register. If a delayed load is in progress, it is completed. Finally, the kernel's exception handler is invoked.
3. It advances the simulated clock (the clock, used to simulate interrupts, is discussed in the following section).

Note that from a user-level process's perspective, exceptions take place in the same way as if the program were executing on a bare machine; an exception handler is invoked to deal with the problem. However, from our perspective, the kernel's exception handler is actually called via a normal procedure call by the simulated processor.

The processor provides three methods we have not discussed yet: `makeAddress()`, `offsetFromAddress()`, and `pageFromAddress()`. These are utility procedures that help the kernel go between virtual addresses and virtual-page/offset pairs.

4.2. Address Translation

The simulated processor supports one of two address translation modes: linear page tables, or a software-managed TLB. While the former is simpler to program, the latter more closely corresponds to what current machines support.

In both cases, when translating an address, the processor breaks the 32-bit virtual address into a virtual page number (VPN) and a page offset. Since the processor's page size is 1KB, the offset is 10 bits wide and the VPN is 22 bits wide. The processor then translates the virtual page number into a translation entry.

Each translation entry (see the TranslationEntry class) contains six fields: a valid bit, a read-only bit, a used bit, a dirty bit, a 22-bit VPN, and a 22-bit physical page number (PPN). The valid bit and read-only bit are set by the kernel and read by the processor. The used and dirty bits are set by the processor, and read and cleared by the kernel.

4.2.1. Linear Page Tables

When in linear page table mode, the processor uses the VPN to index into an array of translation entries. This array is specified by calling `setPageTable()`. If, in translating a VPN, the VPN is greater than or equal to the length of the page table, or the VPN is within range but the corresponding translation entry's valid bit is clear, then a page fault occurs.

In general, each user process will have its own private page table. Thus, each process switch requires calling `setPageTable()`. On a real machine, the page table pointer would be stored in a special processor register.

4.2.2. Software-Managed TLB

When in TLB mode, the processor maintains a small array of translation entries that the kernel can read/write using `readTLBEntry()` and `writeTLBEntry()`. On each address translation, the processor searches the entire TLB for the first entry whose VPN matches.

5. User-Level Processes

Nachos runs each user program in its own private address space. Nachos can run any COFF MIPS binaries that meet a few restrictions. Most notably, the code must only make system calls that Nachos understands. Also, the code must not use any floating point instructions, because the Nachos MIPS simulator does not support coprocessors.

5.1. Loading COFF Binaries

COFF (Common Object File Format) binaries contain a lot of information, but very little of it is actually

relevent to Nachos programs. Further, Nachos provides a COFF loader class, nachos.machine.Coff, that abstracts away most of the details. But a few details are still important.

A COFF binary is broken into one or more *sections*. A section is a contiguous chunk of virtual memory, all the bytes of which have similar attributes (code vs. data, read-only vs. read-write, initialized vs. uninitialized). When Nachos loads a program, it creates a new processor, and then copies each section into the program's virtual memory, at some start address specified by the section. A COFF binary also specifies an initial value for the PC register. The kernel must initialize this register, as well as the stack pointer, and then instruct the processor to start executing the program.

The Coff constructor takes one argument, an OpenFile referring to the MIPS binary file. If there is any error parsing the headers of the specified binary, an EOFException is thrown. Note that if this constructor succeeds, the file belongs to the Coff object; it should not be closed or accessed anymore, except through Coff operations.

There are four Coff methods:

- `getNumSections()` returns the number of sections in this binary.
- `getSection()` takes a section number, between `0` and `getNumSections() - 1`, and returns a CoffSection object representing the section. This class is described below.
- `getEntryPoint()` returns the value with which to initialize the program counter.
- `close()` releases any resources allocated by the loader. This includes closing the file passed to the constructor.

The CoffSection class allows Nachos to access a single section within a COFF executable. Note that while the MIPS cross-compiler generates a variety of sections, the only important distinction to the Nachos kernel is that some sections are read-only (i.e. the program should never write to any byte in the section), while some sections are read-write (i.e. non-`const` data). There are four methods for accessing COFF sections:

- `getFirstVPN()` returns the first virtual page number occupied by the section.
- `getLength()` returns the number of pages occupied by the section. This section therefore occupies pages `getFirstVPN()` through `getFirstVPN() + getLength() - 1`. Sections should never overlap.
- `isReadOnly()` returns true if and only if the section is read-only (i.e. it only contains code or constant data).
- `loadPage()` reads a page of the section into main memory. It takes two arguments, the page within the section to load (in the range `0` through `getLength() - 1`) and the physical page of memory to write.

5.2. Starting a Process

The kernel starts a process in two steps. First, it calls `UserProcess.newUserProcess()` to instantiate a process of the appropriate class. This is necessary because the process class changes as more functionality is added to each process. Second, it calls `execute()` to load and execute the program, passing the name of the file containing the binary and an array of arguments.

`execute()` in turn takes two steps. It first loads the program into the process's address space by calling `load()`. It then forks a new thread, which initializes the processor's registers and address translation information and then calls `Machine.processor().run()` to start executing user code.

`load()` opens the executable's file, instantiates a COFF loader to process it, verifies that the sections are contiguously placed in virtual memory, verifies that the arguments will fit within a single page, calculates the size of the program in pages (including the stack and arguments), calls `loadSections()` to actually load the contents of each section, and finally writes the command line arguments to virtual memory.

`load()` lays out the program in virtual memory as follows: first, starting at virtual address 0, the sections of the executable occupy a contiguous region of virtual memory. Next comes the stack, the size of which is determined by the variable `stackPages`. Finally, one page is reserved for command line arguments (that `argv` array).

`loadSections()` allocates physical memory for the program and initializes its page table, and then loads sections to physical memory (though for the VM project, this loading is done lazily, delayed until pages are demanded). This is separated from the rest of `load()` because the loading mechanism depends on the details of the paging system.

In the code you are given, Nachos assumes that only a single process can exist at any given time. Therefore, `loadSections()` assumes that no one else is using physical memory, and it initializes its page table so as to map virtual memory addresses directly to physical memory addresses, without any translation (i.e. virtual address `n` maps to physical address `n`).

The method `initRegisters()` zeros out the processor's registers, and then initializes the program counter, the stack pointer, and the two argument registers (which hold `argc` and `argv`) with the values computed by `load()`. `initRegisters()` is called exactly once by the thread forked in `execute()`.

5.3. User Threads

User threads (that is, kernel threads that will be used to run user code) require additional state. Specifically, whenever a user thread starts running, it must restore the processor's registers, and possibly restore some address translation information as well. Right before a context switch, a user thread needs to save the processor's registers.

To accomplish this, there is a new thread class, UThread, that extends KThread. It is necessary to know which process, if any, the current thread belongs to. Therefore each UThread is bound to a single process.

UThread overrides `saveState()` and `restoreState()` from KThread so as to save/restore the additional information. These methods deal only with the user register set, and then direct the current process to deal with process-level state (i.e. address translation information). This separation makes it possible to allow multiple threads to run within a single process.

5.4. System Calls and Exception Handling

User programs invoke system calls by executing the MIPS `syscall` instruction, which causes the Nachos kernel exception handler to be invoked (with the cause register set to `Processor.exceptionSyscall`). The kernel must first tell the processor where the exception handler is by calling `Machine.processor().setExceptionHandler()`.

The default Kernel exception handler, `UserKernel.exceptionHandler()`, reads the value of the processor's cause register, determines the current process, and invokes `handleException` on the current process, passing the cause of the exception as an argument. Again, for a syscall, this value will be `Processor.exceptionSyscall`.

The `syscall` instruction indicates a system call is requested, but doesn't indicate which system call to perform. By convention, user programs place the value indicating the particular system call desried into MIPS register `r2` (the first return register, `v0`) before executing the `syscall` instruction. Arguments to the system call, when necessary, are passed in MIPS registers `r4` through `r7` (i.e. the argument registers, `a0 ... a3`), following the standard C procedure call convention. Function return values, including system call return values, are expected to be in register `r2` (`v0`) on return.

> **Note:** When accessing user memory from within the exception handler (or within Nachos in general), user-level addresses cannot be referenced directly. Recall that user-level processes execute in their own private address spaces, which the kernel cannot reference directly. Use `readVirtualMemory()`, `readVirtualMemoryString()`, and `writeVirtualMemory()` to make use of pointer arguments to syscalls.

Notes

1. The ThreadQueue object representing the ready queue is stored in the static variable `KThread.readyQueue`.