CS 162
Discussion Section
Week 2
Who am I?

Prashanth Mohan

http://www.cs.berkeley.edu/~prmohan
prashmohan@berkeley.edu

Office Hours: 11-12pm Tu W at 411 Soda Hall

Research:
Data Security
Tech for Sustainability
Administrivia

• Register with Piazza
• About the groups
• Meetings with TA for reviews.
• Get to know Nachos
What is an OS and What does it do?
Why do you want an OS?

• **Isolation**
  – Fault: “if my program crashes yours shouldn’t”
  – Performance: “if my program starts to do some massive computation, it shouldn’t starve yours from running”

• **Mediation (multiplexing/sharing + protection)**
  – Manage the sharing of hardware resources (CPU, NIC, RAM, disk, keyboard, sound card, etc)

• **Abstractions and Primitives**
  – Set of constructs and well-defined interfaces to simplify application development: “all the code you didn’t write” in order to implement your application
    • Because hardware changes faster than applications!
    • Because some concepts are useful across applications
Why bother with an OS?

• **User benefits**
  – Efficiency (cost and performance)
    • Share one computer across many users
    • Concurrent execution of multiple programs
  – Safety
    • OS protects programs from each other
    • OS fairly multiplexes resources across programs

• **Application benefits**
  – Simplicity
    • Sockets instead of ethernet cards
  – Portability
    • Device independence: tiCOM card or Intel card?
Why is concurrency hard?

What do you need to get concurrency working?
Concurrency

• Concurrency means multiple threads of computation can make progress, but possibly by sharing the same processor
  – Like doing homework while chatting on IM

• Why Concurrency?
  – Consider a web server: while it’s waiting for a response from one client, it could read a request for another client
  – Consider a browser: while it’s waiting for a response from a web server, it wants to react to mouse or keyboard input
Concurrency increases/enables responsiveness
Different levels of abstraction

- Threads
- Processes
- Symmetric multithreading
- Distributed systems (Single system image)
What is a thread?

A thread of execution is the smallest unit of processing that can be scheduled by an operating system. – Wikipedia circa Jan ‘12

What resources does a thread have?
Single and Multi Threading

<table>
<thead>
<tr>
<th>code</th>
<th>data</th>
<th>files</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>registers</td>
<td>stack</td>
<td></td>
</tr>
<tr>
<td>thread</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

single-threaded process

<table>
<thead>
<tr>
<th>code</th>
<th>data</th>
<th>files</th>
</tr>
</thead>
<tbody>
<tr>
<td>registers</td>
<td>registers</td>
<td>registers</td>
</tr>
<tr>
<td>stack</td>
<td>stack</td>
<td>stack</td>
</tr>
</tbody>
</table>

multithreaded process
Threading Issues

- Semantics of `fork()` and `exec()` system calls
- Thread cancellation
- Signal handling
- Thread pools
- Thread specific data
- Scheduler activations
How do you have multiple threads?

```
Code
Global Data
Heap
Global Data
Stack 2
Stack 1
```

Address Space
What is the difference between a Thread and a Process?
What is a Process?

Process Control Block

- process state
- process number
- program counter
- registers
- memory limits
- list of open files
  ...
Process Address Space

What is an address space?

What does this mean in terms of hardware?
Operating System Memory Management

Translation Map 1

Translation Map 2

Physical Address Space

Translation Map 1

Translation Map 2

Program 1
Virtual Address Space 1

Program 2
Virtual Address Space 2

Code
Data
Heap
Stack

Code
Data
Heap
Stack

Code
Data
Heap
Stack

Code
Data
Heap
Stack

OS code
OS data
OS heap & Stacks

OS code
OS data
OS heap & Stacks
When and how do you switch between processes?
Scheduling states

- new
- admitted
- interrupt
- exit
- terminated

- ready
- running
- waiting

- L/O or event completion
- scheduler dispatch
- L/O or event wait
Context Switching

The diagram illustrates the process of context switching in an operating system. When a process, say $P_0$, is executing, an interrupt or system call occurs, leading to the save of its state into the Process Control Block ($PCB_0$). The system then transitions to process $P_1$, which executes until another interrupt or system call occurs. This results in saving the state of $P_1$ into its $PCB_1$ and reloading the state of $P_0$ from its $PCB_0$. The process repeats until all processes have been executed.
An OS needs to mediate access to resources: how do we share the CPU?

• Strategy 1: force everyone to cooperate
  – a thread willingly gives up the CPU by calling yield() which calls into the scheduler, which context-switches to another thread
  – what if a thread never calls yield()?

• Strategy 2: use pre-emption
  – at timer interrupt, scheduler gains control and context switches as appropriate

Recall, an OS needs to mediate access to resources: how do we share the CPU?
From Lecture: Two Thread Yield

• Consider the following code blocks:
  "proc A() {
    B();
  }
  proc B() {
    while(TRUE) {
      yield();
    }
  }

• Suppose we have 2 threads:
  Threads S and T
Concurrent Programs

• Two threads, A and B, compete with each other
  – One tries to increment a shared counter
  – The other tries to decrement the counter

<table>
<thead>
<tr>
<th>Thread A</th>
<th>Thread B</th>
</tr>
</thead>
<tbody>
<tr>
<td>i = 0;</td>
<td>i = 0;</td>
</tr>
<tr>
<td>while (i &lt; 10)</td>
<td>while (i &gt; -10)</td>
</tr>
<tr>
<td>i = i + 1; i = i - 1;</td>
<td></td>
</tr>
<tr>
<td>printf(“A wins!”);</td>
<td>printf(“B wins!”);</td>
</tr>
</tbody>
</table>

• Assume that memory loads and stores are atomic, but incrementing and decrementing are \textit{not} atomic

• Who wins? Could be either

• Is it guaranteed that someone wins? Why or why not?

• What if both threads have their own CPU running at same speed? Is it guaranteed that it goes on forever?
Parallelism

• Parallelism means leveraging multiple processors to compute a result faster
  – Like dividing a pile of work among people

• Why Parallelism?
  – Because we actually have multiple CPUs!
  – Because matrix multiply goes so much faster!
How does OS do it?

- **Kernel**: The highly privileged code that carries out lowest level OS functions

- **Use multiple processes**, OS schedules them (i.e. multiplexes resources between them)
  - Each process has its own address space
  - Each process maintains a list of open files, open network connections ...

- **Use multiple threads within a process**, either OS or user schedules them
  - Threads share the process’s address space

- **Threads** are cheaper than processes and can more easily share state! But have no isolation.
Recall (61C): What happens during execution?

Execution sequence:
- Fetch Instruction at PC
- Decode
- Execute (possibly using registers)
- Write results to registers/mem
- PC = Next Instruction(PC)
- Repeat