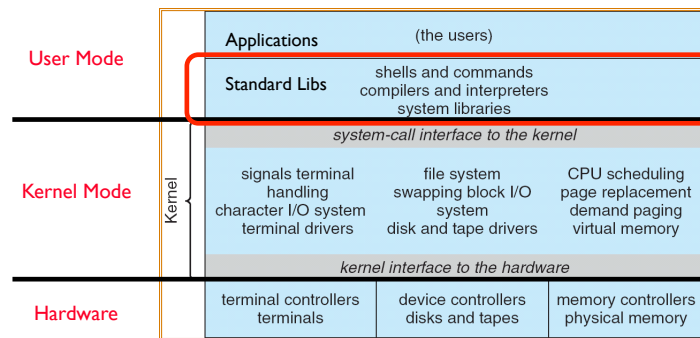


CSI 62 Operating Systems and Systems Programming Lecture 4

Introduction to I/O, Sockets, Networking

September 6th, 2017
Prof. Ion Stoica
<http://cs162.eecs.berkeley.edu>

Recall: UNIX System Structure



9/6/17

CS162 ©UCB Fall 2017

Lec 4.2

How Does the Kernel Provide Services?

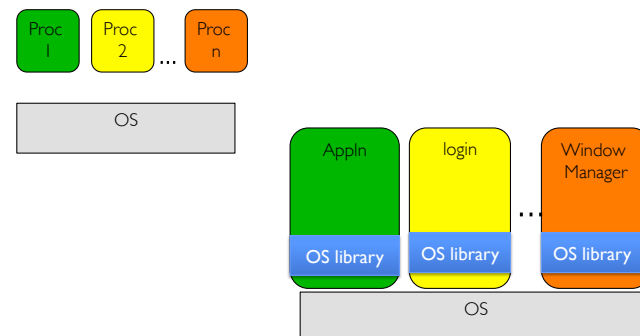
- You said that applications request services from the operating system via **syscall**, but ...
- I've been writing all sort of useful applications and I never ever saw a "**syscall**" !!!
- That's right.
- It was buried in the programming language runtime library (e.g., **libc.a**)
- ... Layering

9/6/17

CS162 ©UCB Fall 2017

Lec 4.3

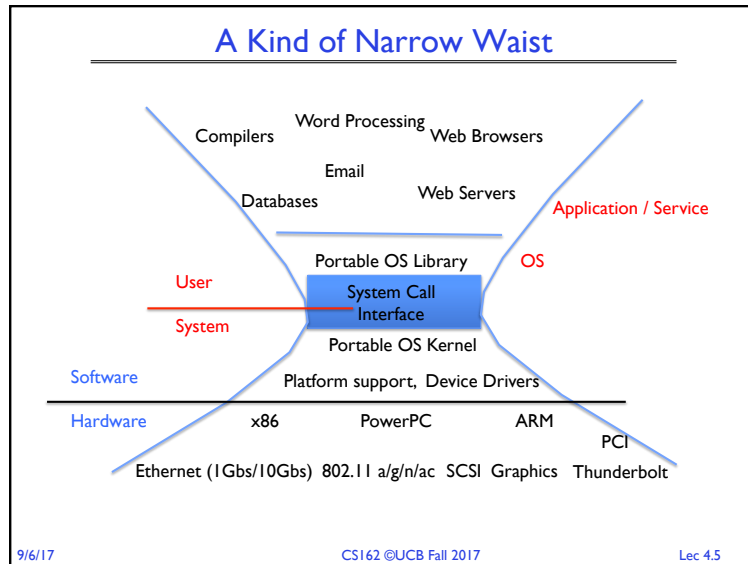
OS Run-Time Library



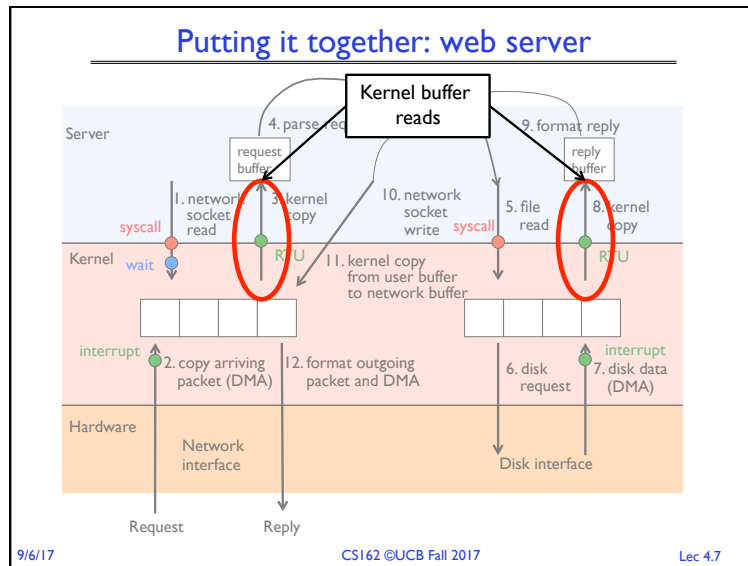
9/6/17

CS162 ©UCB Fall 2017

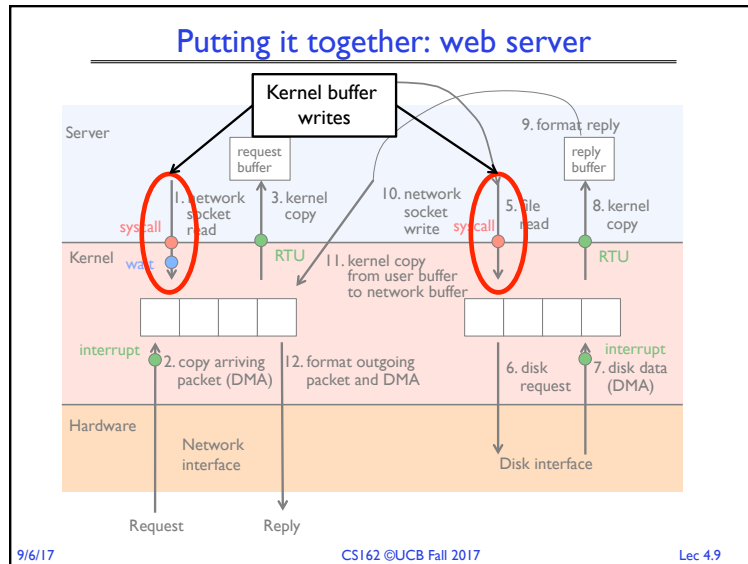
Lec 4.4



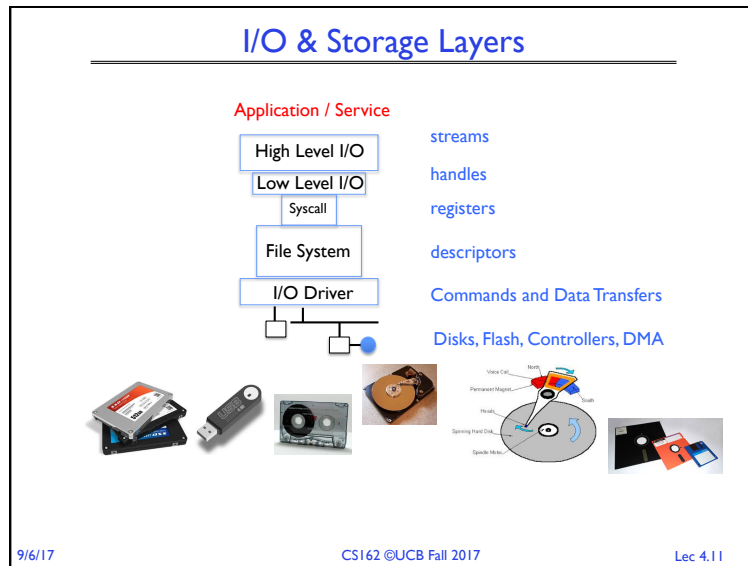
- ### Key Unix I/O Design Concepts
- Uniformity
 - file operations, device I/O, and interprocess communication through `open`, `read/write`, `close`
 - Allows simple composition of programs
 - » `find | grep | wc ...`
 - Open before use
 - Provides opportunity for access control and arbitration
 - Sets up the underlying machinery, i.e., data structures
 - Byte-oriented
 - Even if blocks are transferred, addressing is in bytes
 - Kernel buffered reads
 - Streaming and block devices looks the same
 - read blocks process, yielding processor to other task
- 9/6/17 CS162 @UCB Fall 2017 Lec 4.6



- ### Key Unix I/O Design Concepts
- Uniformity
 - file operations, device I/O, and interprocess communication through `open`, `read/write`, `close`
 - Allows simple composition of programs
 - » `find | grep | wc ...`
 - Open before use
 - Provides opportunity for access control and arbitration
 - Sets up the underlying machinery, i.e., data structures
 - Byte-oriented
 - Even if blocks are transferred, addressing is in bytes
 - Kernel buffered reads
 - Streaming and block devices looks the same
 - read blocks process, yielding processor to other task
 - Kernel buffered writes
 - Completion of out-going transfer decoupled from the application, allowing it to continue
- 9/6/17 CS162 @UCB Fall 2017 Lec 4.8



- ### Key Unix I/O Design Concepts
- **Uniformity**
 - file operations, device I/O, and interprocess communication through **open, read/write, close**
 - Allows simple composition of programs
 - » `find | grep | wc ...`
 - **Open before use**
 - Provides opportunity for access control and arbitration
 - Sets up the underlying machinery, i.e., data structures
 - **Byte-oriented**
 - Even if blocks are transferred, addressing is in bytes
 - **Kernel buffered reads**
 - Streaming and block devices looks the same
 - read blocks process, yielding processor to other task
 - **Kernel buffered writes**
 - Completion of out-going transfer decoupled from the application, allowing it to continue
 - **Explicit close**
- The diagram is dated 9/6/17, CS162 ©UCB Fall 2017, Lec 4.10.



- ### The File System Abstraction
- **High-level idea**
 - Files live in hierarchical namespace of filenames
 - **File**
 - Named collection of data in a file system
 - File data
 - » Text, binary, linearized objects
 - File Metadata: information about the file
 - » Size, Modification Time, Owner, Security info
 - » Basis for access control
 - **Directory**
 - “Folder” containing files & Directories
 - Hierarchical (graphical) naming
 - » Path through the directory graph
 - » Uniquely identifies a file or directory
 - `/home/ff/cs162/public_html/fa17/index.html`
 - Links and Volumes (later)
- The diagram is dated 9/6/17, CS162 ©UCB Fall 2017, Lec 4.12.

C High-Level File API – Streams (review)

- Operate on “streams” - sequence of bytes, whether text or data, with a position



```
#include <stdio.h>
FILE *fopen( const char *filename, const char *mode );
int fclose( FILE *fp );
```

Mode	Text	Binary	Descriptions
r		rb	Open existing file for reading
w		wb	Open for writing; created if does not exist
a		ab	Open for appending; created if does not exist
r+		rb+	Open existing file for reading & writing.
w+		wb+	Open for reading & writing; truncated to zero if exists, create otherwise
a+		ab+	Open for reading & writing. Created if does not exist. Read from beginning, write as append

Don't forget to flush

9/6/17

CS162 ©UCB Fall 2017

Lec 4.13

Connecting Processes, Filesystem, and Users

- Process has a ‘current working directory’
- Absolute Paths
 - /home/ff/cs162
- Relative paths
 - index.html, ./index.html - current WD
 - ../index.html - parent of current WD
 - ~, ~cs162 - home directory

9/6/17

CS162 ©UCB Fall 2017

Lec 4.14

C API Standard Streams

- Three predefined streams are opened implicitly when a program is executed
 - FILE *stdin - normal source of input, can be redirected
 - FILE *stdout - normal source of output, can be redirected
 - FILE *stderr - diagnostics and errors, can be redirected
- STDIN / STDOUT enable composition in Unix
 - Recall: Use of pipe symbols connects STDOUT and STDIN
 - » find | grep | wc ...

9/6/17

CS162 ©UCB Fall 2017

Lec 4.15

C high level File API – Stream Ops

```
#include <stdio.h>
// character oriented
int fputc( int c, FILE *fp ); // rtn c or EOF on err
int fputs( const char *s, FILE *fp ); // rtn >0 or EOF

int fgetc( FILE *fp );
char *fgets( char *buf, int n, FILE *fp );
```

DESCRIPTION

The `fgets()` function reads at most one less than the number of characters specified by `size` from the given `stream` and stores them in the string `str`. Reading stops when a newline character is found, at end-of-file or error. The newline, if any, is retained. If any characters are read and there is no error, a `'\0'` character is appended to end the string.

9/6/17

CS162 ©UCB Fall 2017

Lec 4.16

C high level File API – Stream Ops

```
#include <stdio.h>
// character oriented
int fputc( int c, FILE *fp ); // rtn c or EOF on err
int fputs( const char *s, FILE *fp ); // rtn >0 or EOF

int fgetc( FILE * fp );
char *fgets( char *buf, int n, FILE *fp );

// block oriented
size_t fread(void *ptr, size_t size_of_elements,
             size_t number_of_elements, FILE *a_file);

size_t fwrite(const void *ptr, size_t size_of_elements,
             size_t number_of_elements, FILE *a_file);
```

9/6/17

CS162 ©UCB Fall 2017

Lec 4.17

C high level File API – Stream Ops

```
#include <stdio.h>
// character oriented
int fputc( int c, FILE *fp ); // rtn c or EOF on err
int fputs( const char *s, FILE *fp ); // rtn >0 or EOF

int fgetc( FILE * fp );
char *fgets( char *buf, int n, FILE *fp );

// block oriented
size_t fread(void *ptr, size_t size_of_elements,
             size_t number_of_elements, FILE *a_file);

size_t fwrite(const void *ptr, size_t size_of_elements,
             size_t number_of_elements, FILE *a_file);

// formatted
int fprintf(FILE *restrict stream, const char *restrict format,
...);
int fscanf(FILE *restrict stream, const char *restrict format,
...);
```

9/6/17

CS162 ©UCB Fall 2017

Lec 4.18

Example Code

```
#include <stdio.h>
#define BUFLLEN 256
FILE *outfile;
char mybuf(BUFLLEN);
int storetofile( FILE *outfile, char *instring; ) {
    char *instring;

    outfile = fopen("/usr/homes/testing/tokens", "w+");
    if (!outfile)
        return (-1); // Error!
    while (1) {
        instring = fgets(mybuf, BUFLLEN, stdin); // catches overrun!

        // Check for error or end of file (^D)
        if (!instring || strlen(instring)==0) break;

        // Write string to output file, exit on error
        if (fputs(instring, outfile)< 0) break;
    }
    fclose(outfile); // Flushes from userspace
}
```

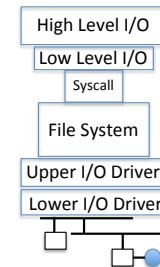
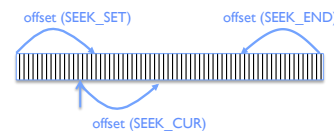
9/6/17

CS162 ©UCB Fall 2017

Lec 4.19

C Stream API positioning

```
int fseek(FILE *stream, long int offset, int whence);
long int ftell (FILE *stream)
void rewind (FILE *stream)
```



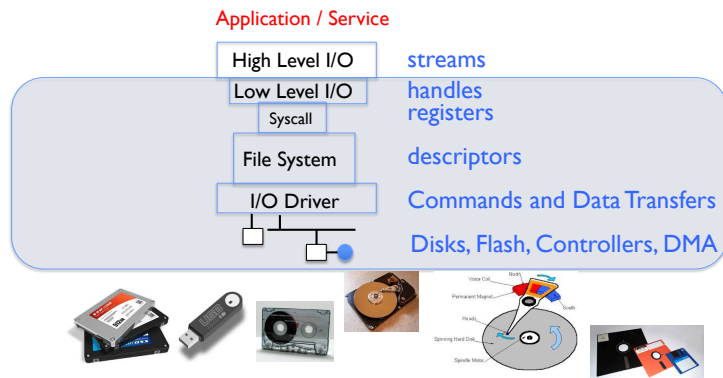
- Preserves high level abstraction of uniform stream of objects
- Adds buffering for performance

9/6/17

CS162 ©UCB Fall 2017

Lec 4.20

What's below the surface ??



9/6/17

CS162 ©UCB Fall 2017

Lec 4.21

C Low level I/O

- Operations on File Descriptors – as OS object representing the state of a file
 - User has a “handle” on the descriptor

```
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
```

```
int open (const char *filename, int flags [, mode_t mode])
int creat (const char *filename, mode_t mode)
int close (int filedes)
```

Bit vector of:

- Access modes (Rd,Wr,...)
- Open Flags (Create,...)
- Operating modes (Appends,...)

Bit vector of Permission Bits:

- User|Group|Other X R|W|X

http://www.gnu.org/software/libc/manual/html_node/Opening-and-Closing-Files.html

9/6/17

CS162 ©UCB Fall 2017

Lec 4.22

C Low Level: standard descriptors

```
#include <unistd.h>
```

```
STDIN_FILENO - macro has value 0
STDOUT_FILENO - macro has value 1
STDERR_FILENO - macro has value 2
```

```
int fileno (FILE *stream)
```

```
FILE * fdopen (int filedes, const char *opentype)
```

- Crossing levels: File descriptors vs. streams
- Don't mix them!

9/6/17

CS162 ©UCB Fall 2017

Lec 4.23

C Low Level Operations

```
ssize_t read (int filedes, void *buffer, size_t maxsize)
- returns bytes read, 0 => EOF, -1 => error
ssize_t write (int filedes, const void *buffer, size_t size)
- returns bytes written
```

```
off_t lseek (int filedes, off_t offset, int whence)
```

```
int fsync (int filedes) - wait for i/o to finish
void sync (void) - wait for ALL to finish
```

- When write returns, data is on its way to disk and can be read, but it may not actually be permanent!

9/6/17

CS162 ©UCB Fall 2017

Lec 4.24

And lots more !

- TTYs versus files
- Memory mapped files
- File Locking
- Asynchronous I/O
- Generic I/O Control Operations
- Duplicating descriptors

```
int dup2 (int old, int new)
int dup (int old)
```

9/6/17

CS162 ©UCB Fall 2017

Lec 4.25

Another example: lowio-std.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>

#define BUFSIZE 1024

int main(int argc, char *argv[])
{
    char buf[BUFSIZE];
    ssize_t writelen = write(STDOUT_FILENO, "I am a process.\n", 16);

    ssize_t readlen = read(STDIN_FILENO, buf, BUFSIZE);

    ssize_t strlen = snprintf(buf, BUFSIZE, "Got %zd chars\n", readlen);

    writelen = strlen < BUFSIZE ? strlen : BUFSIZE;
    write(STDOUT_FILENO, buf, writelen);

    exit(0);
}
```

9/6/17

CS162 ©UCB Fall 2017

Lec 4.26

Administrivia

- Waitlist was closed Friday
 - Unfortunately, no concurrent enrollments will be processed
- Recommendation: Read assigned readings *before* lecture
- Group sign up with the autograder this week
 - Get finding groups ASAP – deadline Friday 9/8 at 11:59PM
 - 4 people in a group!
- TA *preference* signup form due Monday 9/11 at 11:59PM
 - Everyone in a group must have the same TA!
 - » Preference given to same section
 - Participation: Get to know your TA!

9/6/17

CS162 ©UCB Fall 2017

Lec 4.27

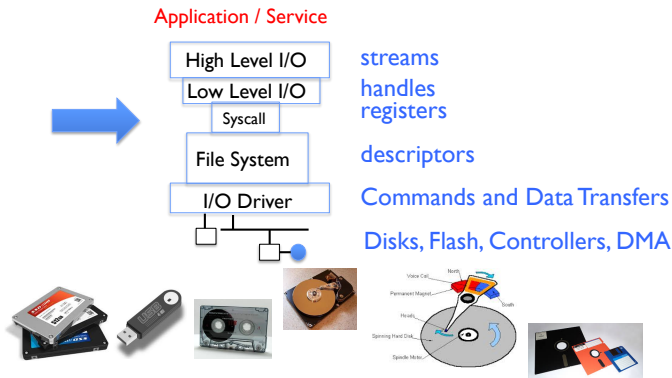
BREAK

9/6/17

CS162 ©UCB Fall 2017

Lec 4.28

What's below the surface ??



9/6/17

CS162 @UCB Fall 2017

Lec 4.29

Recall: SYSCALL

syscalls.kernelgrok.com

Linux Syscall Reference

#	Name	eax	ebx	ecx	edx	esi	edi	Definition
0	sys_restart_syscall	0x00	-	-	-	-	-	kernel/signal.c:2058
1	sys_exit	0x01	int error_code	-	-	-	-	kernel/signal.c:1046
2	sys_fork	0x02	struct pt_regs *	-	-	-	-	arch/alpha/kernel/entry.S:716
3	sys_read	0x03	unsigned int fd	char_user *buf	size_t count	-	-	fs/read_write.c:1361
4	sys_write	0x04	unsigned int fd	const char_user *buf	size_t count	-	-	fs/read_write.c:408
5	sys_open	0x05	const char_user *filename	int flags	int mode	-	-	fs/open.c:900
6	sys_close	0x06	unsigned int fd	-	-	-	-	fs/open.c:969
7	sys_waitpid	0x07	pid_t pid	int __user *stat_addr	int options	-	-	kernel/sys.c:1771
8	sys_creat	0x08	const char_user *pathname	int mode	-	-	-	fs/open.c:933
9	sys_link	0x09	const char_user *oldname	const char_user *newname	-	-	-	fs/namei.c:2120

Generated from Linux kernel 2.6.35.4 using [Exuberant Ctags](#), [Python](#), and [DataTables](#).
Project on [GitHub](#). Hosted on [GitHub Pages](#).

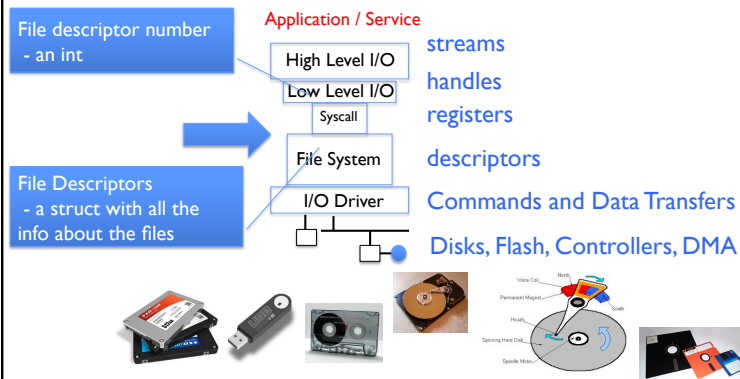
- Low level lib parameters are set up in registers and syscall instruction is issued
 - A type of synchronous exception that enters well-defined entry points into kernel

9/6/17

CS162 @UCB Fall 2017

Lec 4.30

What's below the surface ??



9/6/17

CS162 @UCB Fall 2017

Lec 4.31

Internal OS File Descriptor

- Internal Data Structure describing everything about the file
 - Where it resides
 - Its status
 - How to access it

- Pointer:
`struct file *file`

```

746 struct file {
747     union {
748         struct list_head f_u_list;
749         struct f_u_list f_u_list;
750     };
751     struct path f_path;
752     struct f_dentry f_dentry;
753 #define f_inode f_dentry.f_inode; /* cast
754     const struct file_operations *f_op;
755
756     /*
757     * Protects f_ep_links, f_flags.
758     * Must not be taken from IRQ context.
759     */
760     spinlock_t f_lock;
761     atomic_long_t f_count;
762     unsigned int f_flags;
763     fmode_t f_mode;
764     struct mutex f_pos_lock;
765     loff_t f_pos;
766     struct fown_struct f_owner;
767     const struct cred *f_cred;
768     struct file_ra_state f_ra;
769
770     u64 f_version;
771 #ifdef CONFIG_SECURITY
772     void *f_security;
773 #endif
774     /* needed for tty driver, and maybe others */
775     void *private_data;
776
777 #ifdef CONFIG_EPOLL
778     /* Used by fs/eventpoll.c to link all the hooks
779     struct list_head f_ep_links;
780     struct list_head _tfile_llink;
781 #endif /* #ifdef CONFIG_EPOLL */
782     struct address_space *f_mapping;
783     ...attribute__((aligned(4))); /* Test something weird
784 }
    
```

9/6/17

CS162 @UCB Fall 2017

Lec 4.32

File System: from syscall to driver

In fs/read_write.c

```

ssize_t vfs_read(struct file *file, char __user *buf, size_t count, loff_t *pos)
{
    ssize_t ret;
    if (!(file->f_mode & FMODE_READ)) return -EBADF;
    if (!file->f_op || (!file->f_op->read && !file->f_op->aio_read))
        return -EINVAL;
    if (unlikely(!access_ok(VERIFY_WRITE, buf, count))) return -EFAULT;
    ret = rw_verify_area(READ, file, pos, count);
    if (ret >= 0) {
        count = ret;
        if (file->f_op->read)
            ret = file->f_op->read(file, buf, count, pos);
        else
            ret = do_sync_read(file, buf, count, pos);
        if (ret > 0) {
            fsnotify_access(file->f_path.dentry);
            add_rchar(current, ret);
        }
        inc_syscr(current);
    }
    return ret;
}

```

9/6/17

CS162 @UCB Fall 2017

Lec 4.33

Lower Level Driver

- Associated with particular hardware device
- Registers / Unregisters itself with the kernel
- Handler functions for each of the file operations

```

struct file_operations {
    struct module *owner;
    loff_t (*llseek) (struct file *, loff_t, int);
    ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
    ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
    ssize_t (*aio_read) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
    ssize_t (*aio_write) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
    int (*readdir) (struct file *, void *, filldir_t);
    unsigned int (*poll) (struct file *, struct poll_table_struct *);
    int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
    int (*mmap) (struct file *, struct vm_area_struct *);
    int (*open) (struct inode *, struct file *);
    int (*flush) (struct file *, fl_owner_t id);
    int (*release) (struct inode *, struct file *);
    int (*fsync) (struct file *, struct dentry *, int datasync);
    int (*fasync) (int, struct file *, int);
    int (*flock) (struct file *, int, struct file_lock *);
    [...]
};

```

9/6/17

CS162 @UCB Fall 2017

Lec 4.34

Recall: Device Drivers

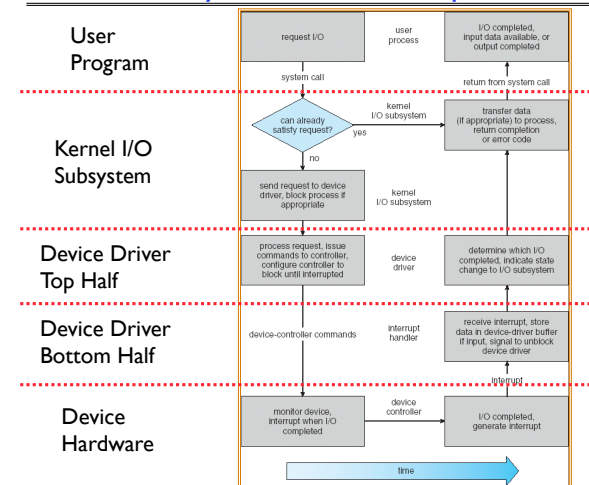
- **Device Driver:** Device-specific code in the kernel that interacts directly with the device hardware
 - Supports a standard, internal interface
 - Same kernel I/O system can interact easily with different device drivers
 - Special device-specific configuration supported with the `ioctl()` system call
- Device Drivers typically divided into two pieces:
 - Top half: accessed in call path from system calls
 - » implements a set of **standard, cross-device calls** like `open()`, `close()`, `read()`, `write()`, `ioctl()`, `strategy()`
 - » This is the kernel's interface to the device driver
 - » Top half will *start* I/O to device, may put thread to sleep until finished
 - Bottom half: run as interrupt routine
 - » Gets input or transfers next block of output
 - » May wake sleeping threads if I/O now complete

9/6/17

CS162 @UCB Fall 2017

Lec 4.35

Life Cycle of An I/O Request

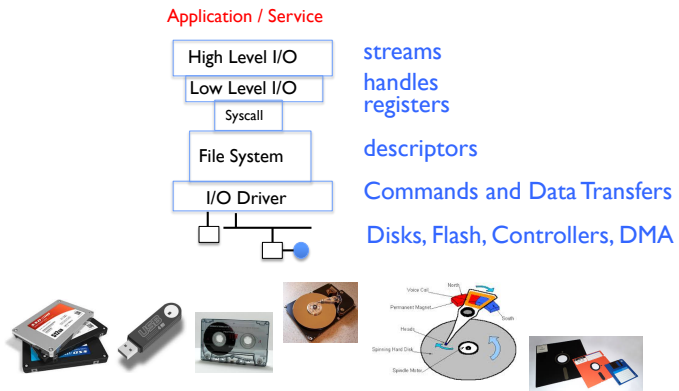


9/6/17

CS162 @UCB Fall 2017

Lec 4.36

So what happens when you fgetc?



9/6/17

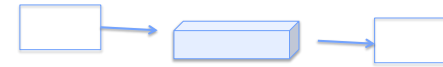
CS162 ©UCB Fall 2017

Lec 4.37

Communication between processes

- Can we view files as communication channels?

```
write(wfd, wbuf, wlen);
```



```
n = read(rfd, rbuf, rmax);
```

- Producer and Consumer of a file may be distinct processes
 - May be separated in time (or not)
- However, what if data written once and consumed once?
 - Don't we want something more like a queue?
 - Can still look like File I/O!

9/6/17

CS162 ©UCB Fall 2017

Lec 4.38

Communication Across the world looks like file IO

```
write(wfd, wbuf, wlen);
```



```
n = read(rfd, rbuf, rmax);
```

- Connected queues over the Internet
 - But what's the analog of open?
 - What is the namespace?
 - How are they connected in time?

9/6/17

CS162 ©UCB Fall 2017

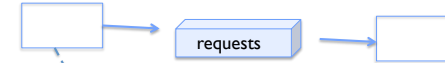
Lec 4.39

Request Response Protocol

Client (issues requests)

Server (performs operations)

```
write(rqfd, rqbuf, buflen);
```



```
n = read(rfd, rbuf, rmax);
```

wait

service request

```
write(wfd, respbuf, len);
```



```
n = read(resfd, resbuf, resmax);
```

9/6/17

CS162 ©UCB Fall 2017

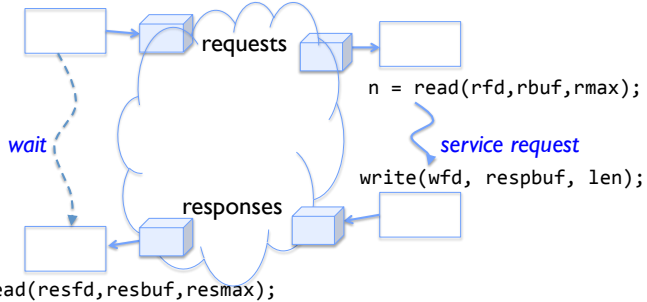
Lec 4.40

Request Response Protocol

Client (issues requests)

Server (performs operations)

```
write(rqfd, rqbuf, buflen);
```

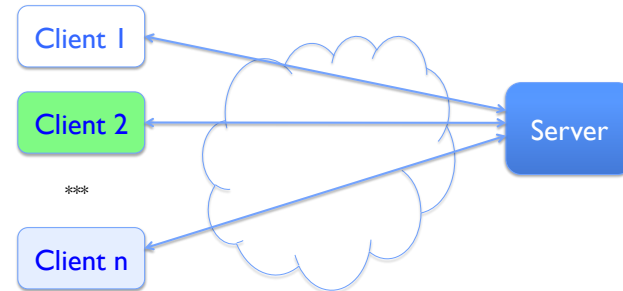


9/6/17

CS162 ©UCB Fall 2017

Lec 4.41

Client-Server Models



- File servers, web, FTP, Databases, ...
- Many clients accessing a common server

9/6/17

CS162 ©UCB Fall 2017

Lec 4.42

Conclusion (I)

- System Call Interface is "narrow waist" between user programs and kernel
- Streaming IO: modeled as a stream of bytes
 - Most streaming I/O functions start with "f" (like "fread")
 - Data buffered automatically by C-library functions
- Low-level I/O:
 - File descriptors are integers
 - Low-level I/O supported directly at system call level
- **STDIN / STDOUT** enable composition in Unix
 - Use of pipe symbols connects **STDOUT** and **STDIN**
 - » `find | grep | wc ...`

9/6/17

CS162 ©UCB Fall 2017

Lec 4.43

Conclusion (II)

- Device Driver: Device-specific code in the kernel that interacts directly with the device hardware
 - Supports a standard, internal interface
 - Same kernel I/O system can interact easily with different device drivers
- File abstraction works for inter-processes communication (local or Internet)

9/6/17

CS162 ©UCB Fall 2017

Lec 4.44