Lecturer: Dr. Michael J. May
Kinneret College

Signals, Syscalls, I/O Basics, High and low, Drivers, Sockets

5 December 2024
Lecture 5

Slides adapted from John Kubiatowicz (UC Berkeley)

Concept Review

Segments	Process context	Interrupt context	Interrupt handler
Kernel Stack	Process Control Block	Scheduler	Simultaneous Multi Threading (SMT)
	fork	pid_t	

operations
Socket
high
Support
Drivers
Read
kernel
ssizet
IO
processes
API
Streams
Basic
standard
File
fp
like
sizet
networks
c
Level
error
writewfd
Write
void
include
stream
device
outfile
char
Sockets
Driver
Files
const
BUFSIZE
rmax
stdioh
data
system
Low
descriptors
nreadrfd
wbuf
UNIX
bytes
abstraction
restrict
filedes

Topics for Today

- Signals
- Syscalls
- Basic Support for I/O (drivers, etc.)
  - Files and Streams
  - Low Level
- I/O and Drivers
- Sockets and networks

Signals: Simple Messaging

- Signal: Like a software interrupt
  - Simple (integer) message you can send a process
  - Interrupts normal execution
- Sent by:
  - One process to another (syscall or via `kill` program)
  - OS to a process (due to event)

- Sample signals:

SIGINT	Interrupt from keyboard (CTRL+C)
SIGQUIT	Quit from keyboard (CTRL+\)
SIGCONT	Continue a program
SIGKILL	Kill signal
SIGTTIN	Background process tries to read from STDIN

Signals – infloop.c

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

void signal_callback_handler(int signum)
{
  printf("Caught signal %d - phew!\n",signum);
  exit(1);
}

int main() {
  signal(SIGINT, signal_callback_handler);
  while (1) {}
}
```

How do you stop this?

Look at `top`

Some more signals – signals-sample.c

```c
void signal_handler(int sig) {
  switch (sig) {
    case SIGINT:
      printf("Received SIGINT (Interrupt from keyboard), signal number: %d\n", sig);
      break;
    case SIGTERM:
      printf("Received SIGTERM (Termination signal), signal number: %d\n", sig);
      break;
    case SIGQUIT:
      printf("Received SIGQUIT (Quit from keyboard), signal number: %d\n", sig);
      break;
    default:
      printf("Received signal number: %d\n", sig);
  }
}
```

signals-sample.c

```c
int main() {
    if (signal(SIGINT, signal_handler) == SIG_ERR) {
        perror("Error registering SIGINT handler");
        exit(1);
    }
    if (signal(SIGTERM, signal_handler) == SIG_ERR) {
        perror("Error registering SIGTERM handler");
        exit(1);
    }
    if (signal(SIGQUIT, signal_handler) == SIG_ERR) {
        perror("Error registering SIGQUIT handler");
        exit(1);
    }
     printf("Running... Press Ctrl+C to send SIGINT, or send SIGTERM or 
SIGQUIT to this process.\n");
    while (1) {
        sleep(1);
    }
    return 0;
}
```

An aside about System Calls (syscalls)

What is a syscall?

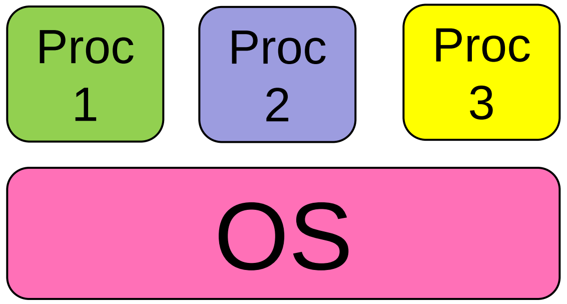
- Applications request services from the operating system via a ***syscall***, but …
- I’ve been writing applications and never saw a “syscall” in code?

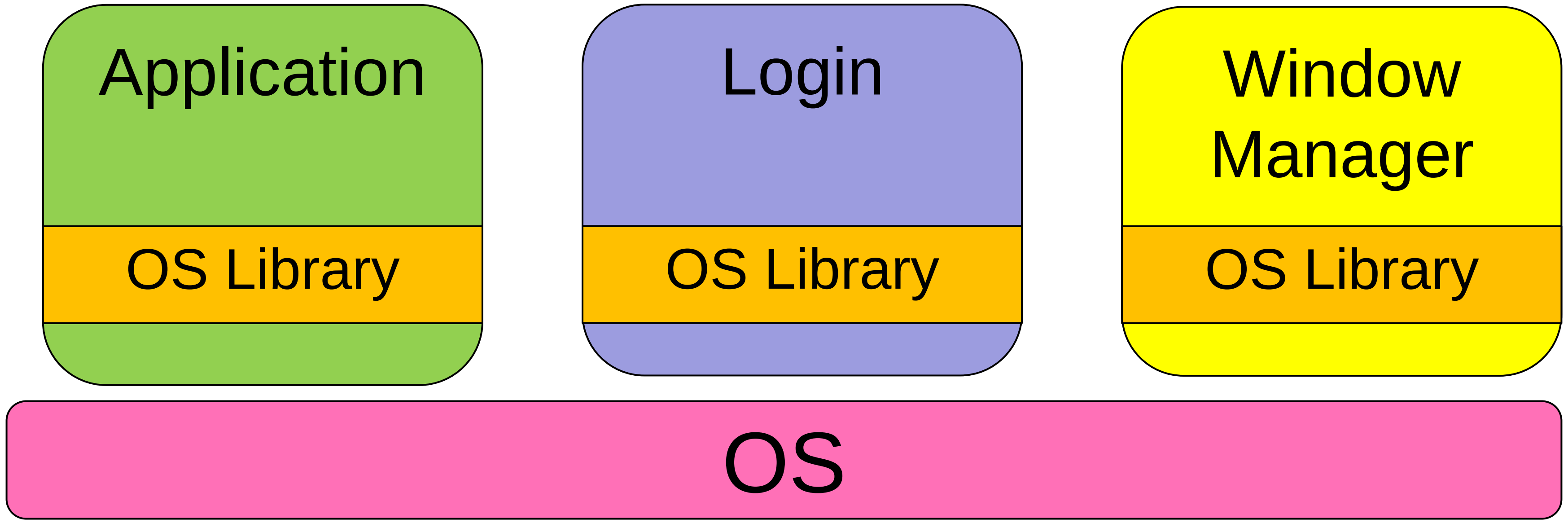
Why?

Syscalls are buried in the programming language runtime library (ex. `libc.a`)

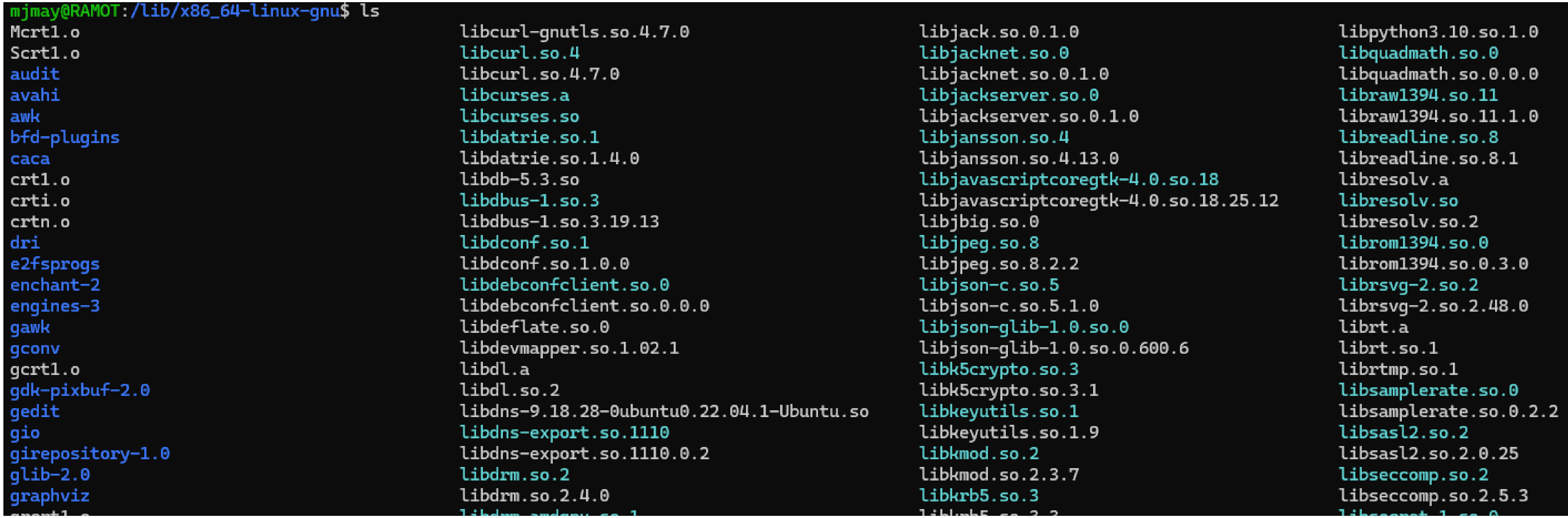
Layering at work

OS run-time library



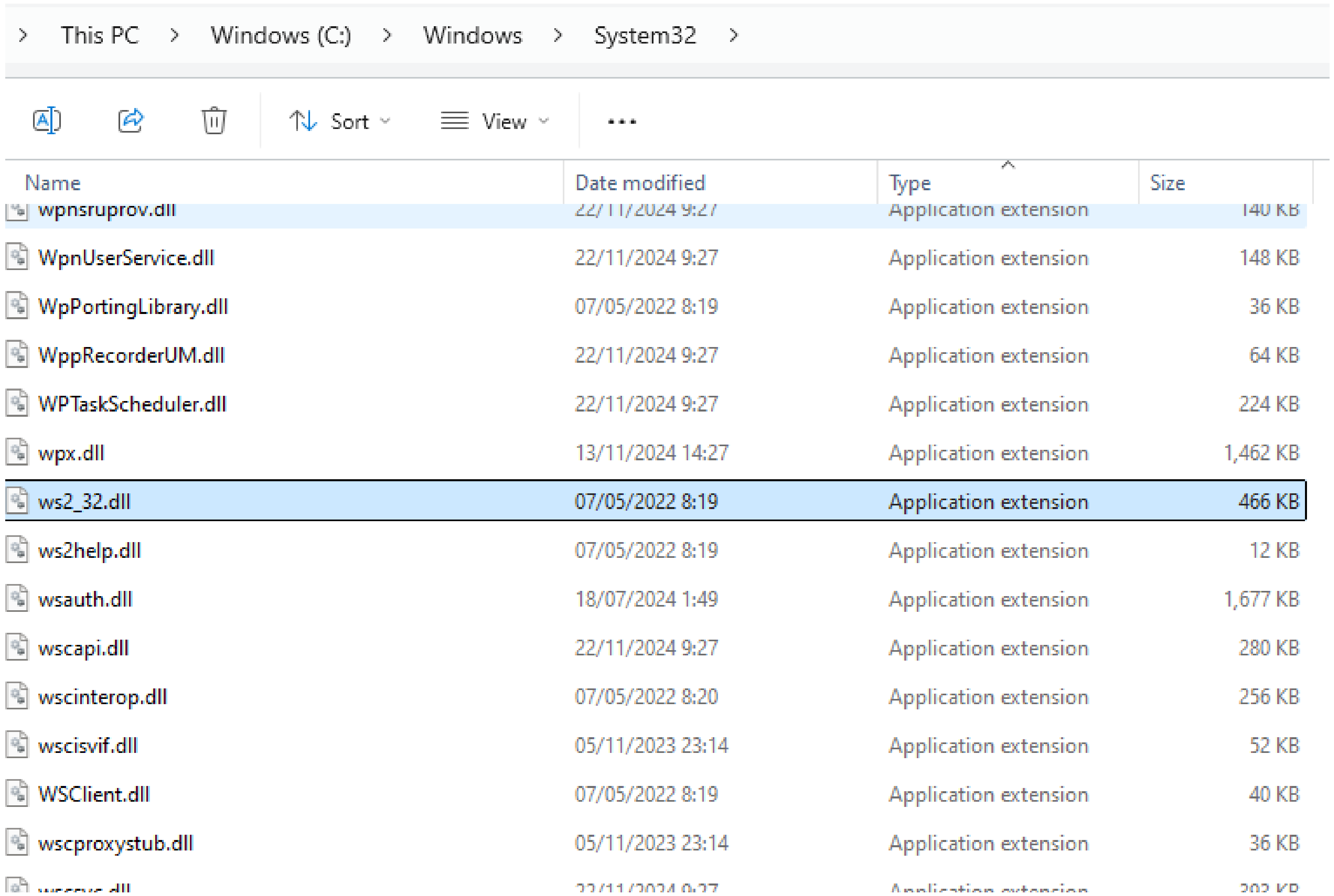


In WSL (Ubuntu)



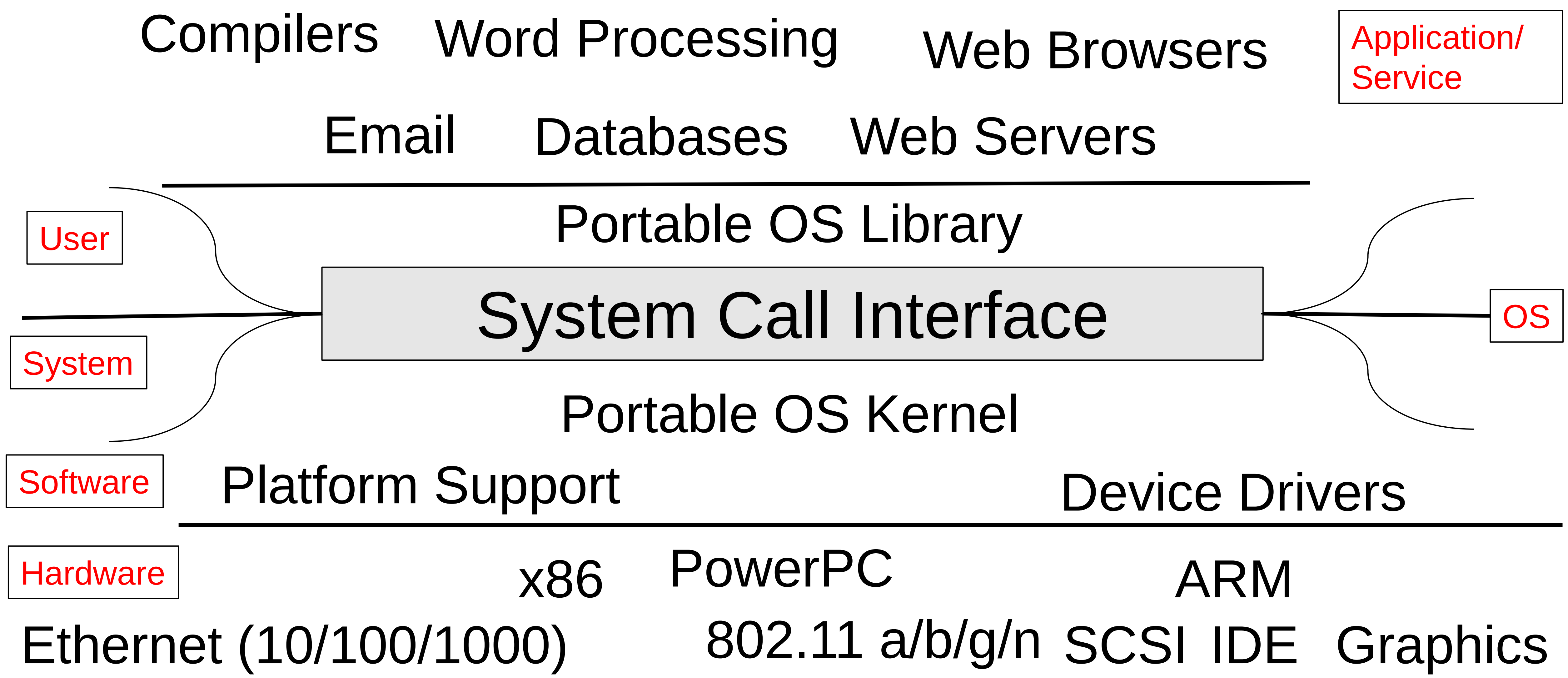
```
mjmay@RAMOT:/lib/x86_64-linux-gnu$ ls
Mcrt1.o                                      libcurl-gnutls.so.4.7.0                        libjack.so.0.1.0                          libpython3.10.so.1.0
Scrt1.o                                      libcurl.so.4                                   libjacknet.so.0                           libquadmath.so.0
audit                                        libcurl.so.4.7.0                               libjacknet.so.0.1.0                       libquadmath.so.0.0.0
avahi                                        libcurses.a                                    libjackserver.so.0                        libraw1394.so.11
awk                                          libcurses.so                                   libjackserver.so.0.1.0                    libraw1394.so.11.1.0
bfd-plugins                                  libdatrie.so.1                                 libjansson.so.4                           libreadline.so.8
caca                                         libdatrie.so.1.4.0                             libjansson.so.4.13.0                      libreadline.so.8.1
crt1.o                                       libdb-5.3.so                                   libjavascriptcoregtk-4.0.so.18            libresolv.a
crti.o                                       libdbus-1.so.3                                 libjavascriptcoregtk-4.0.so.18.25.12      libresolv.so
crtn.o                                       libdbus-1.so.3.19.13                           libjbig.so.0                              libresolv.so.2
dri                                          libdconf.so.1                                  libjpeg.so.8                              librom1394.so.0
e2fsprogs                                    libdconf.so.1.0.0                              libjpeg.so.8.2.2                          librom1394.so.0.3.0
enchant-2                                    libdebconfclient.so.0                          libjson-c.so.5                            librsvg-2.so.2
engines-3                                    libdebconfclient.so.0.0.0                      libjson-c.so.5.1.0                        librsvg-2.so.2.48.0
gawk                                         libdeflate.so.0                                libjson-glib-1.0.so.0                     librt.a
gconv                                        libdevmapper.so.1.02.1                         libjson-glib-1.0.so.0.600.6               librt.so.1
gcrt1.o                                      libdl.a                                        libk5crypto.so.3                          librtmp.so.1
gdk-pixbuf-2.0                               libdl.so.2                                     libk5crypto.so.3.1                        libsamplerate.so.0
gedit                                        libdns-9.18.28-0ubuntu0.22.04.1-Ubuntu.so      libkeyutils.so.1                          libsamplerate.so.0.2.2
gio                                          libdns-export.so.1110                          libkeyutils.so.1.9                        libsasl2.so.2
girepository-1.0                             libdns-export.so.1110.0.2                      libkmod.so.2                              libsasl2.so.2.0.25
glib-2.0                                     libdrm.so.2                                    libkmod.so.2.3.7                          libseccomp.so.2
graphviz                                     libdrm.so.2.4.0                                libkrb5.so.3                              libseccomp.so.2.5.3
```

In Windows 11



Name	Date modified	Type	Size
wpnsruprov.dll	22/11/2024 9:27	Application extension	140 KB
WpnUserService.dll	22/11/2024 9:27	Application extension	148 KB
WpPortingLibrary.dll	07/05/2022 8:19	Application extension	36 KB
WppRecorderUM.dll	22/11/2024 9:27	Application extension	64 KB
WPTaskScheduler.dll	22/11/2024 9:27	Application extension	224 KB
wpx.dll	13/11/2024 14:27	Application extension	1,462 KB
ws2_32.dll	07/05/2022 8:19	Application extension	466 KB
ws2help.dll	07/05/2022 8:19	Application extension	12 KB
wsauth.dll	18/07/2024 1:49	Application extension	1,677 KB
wscapi.dll	22/11/2024 9:27	Application extension	280 KB
wscinterop.dll	07/05/2022 8:20	Application extension	256 KB
wscisvif.dll	05/11/2023 23:14	Application extension	52 KB
WSClient.dll	07/05/2022 8:19	Application extension	40 KB
wscproxystub.dll	05/11/2023 23:14	Application extension	36 KB

A Narrow Waist



© Reuters

https://www.independent.co.uk/news/world/asia/hundreds-of-cars-caught-up-in-beijing-traffic-during-golden-week-a6685961.html

So Far

- Signals
- Syscalls
- Basic Support for I/O (drivers, etc.)
  - Files and Streams
  - Low Level
- I/O and Drivers
- Sockets and networks

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

https://www.pinterest.com/pin/35254809557826141/

Key Unix I/O Design Concepts

Kernel buffered reads

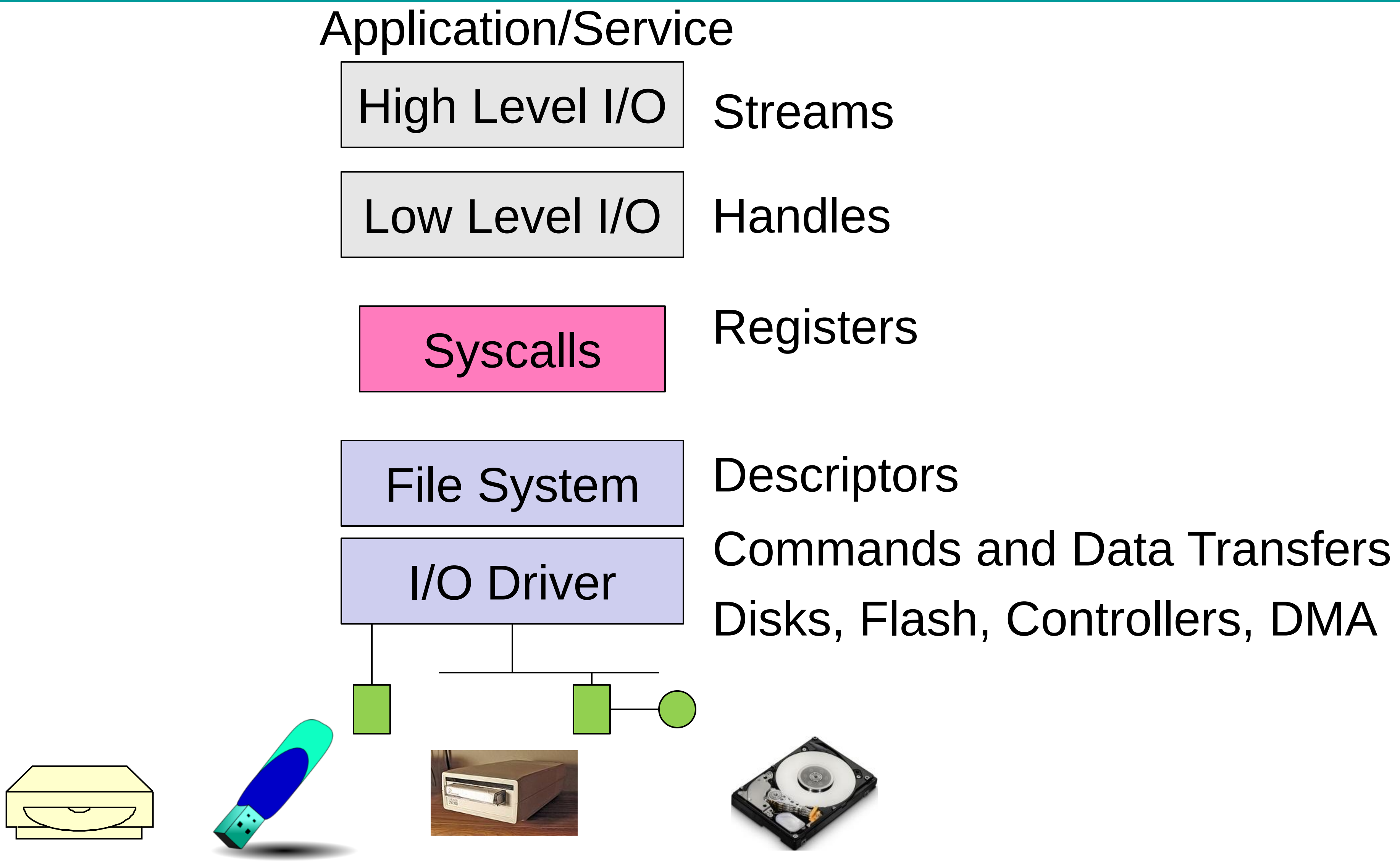
- Streaming and block devices looks the same
- read blocks process, yielding processor to other task

Kernel buffered writes

- Completion of out-going transfer decoupled from the application, allowing it to continue

Explicit close

I/O & Storage Layers



The file system abstraction

- File

Named collection of data in a file system

File data
- Text, binary, linearized objects

File Metadata:
- Size, Modification Time, Owner, Security info
- Basis for access control

- Directory

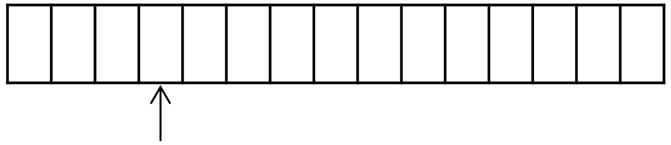
“Folder” containing files & Directories

Hierarchical (graphical) naming
- Path through the directory graph
- Uniquely identifies a file or directory
  - `/home/mjmay/se317/public_html/fa24/index.html`

Links and Volumes (later)

C high level File API – streams

- 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	Description
r	rb	Open existing file for reading
w	wb	Open for writing; create if doesn’t exist
a	ab	Open for appending; create if doesn’t exist
r+	rb+	Open existing file for reading and writing
w+	wb+	Open for reading and writing; truncated to zero if exists, create otherwise
a+	ab+	Open for reading and writing; create if doesn’t exist. Read from beginning, write as append

Connecting Processes, Filesystem, and Users

Every process has a "current working directory"

Absolute Paths

- `/home/mjmay/se317`

Relative paths

- `index.html, ./index.html` - current WD
- `../index.html` - parent of current WD

Path aliases

- `~, ~se317` - home directory

You are here!

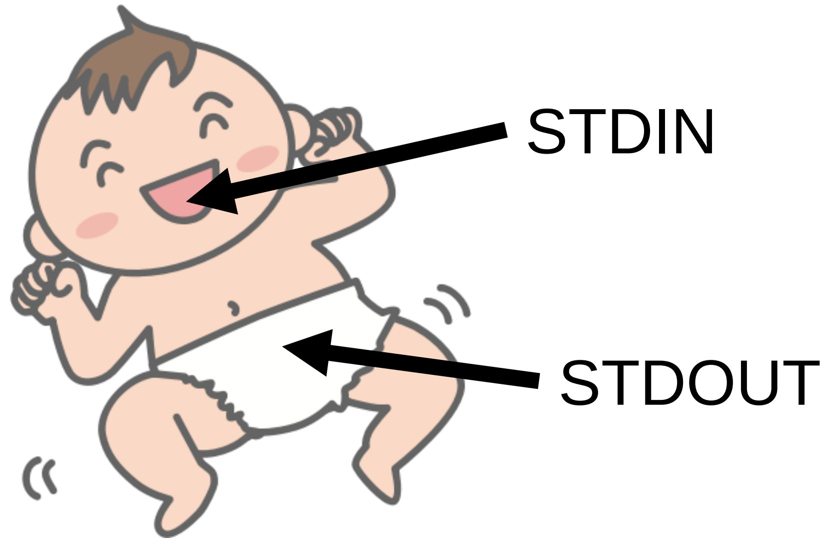
C API Standard Streams

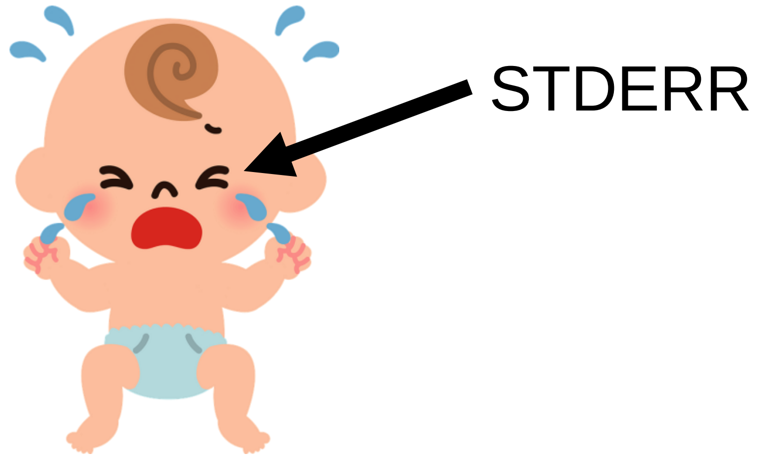
- Three predefined streams are opened implicitly when the program is executed.

1. `FILE *stdin` – normal source of input, can be redirected
2. `FILE *stdout` – normal source of output, can be redirected too
3. `FILE *stderr` – diagnostics and errors

- `STDIN`/`STDOUT` enable composition in Unix
  - Recall: Use of pipe symbols connects `STDOUT` and `STDIN`
    - `find | grep | wc …`

Picturing it





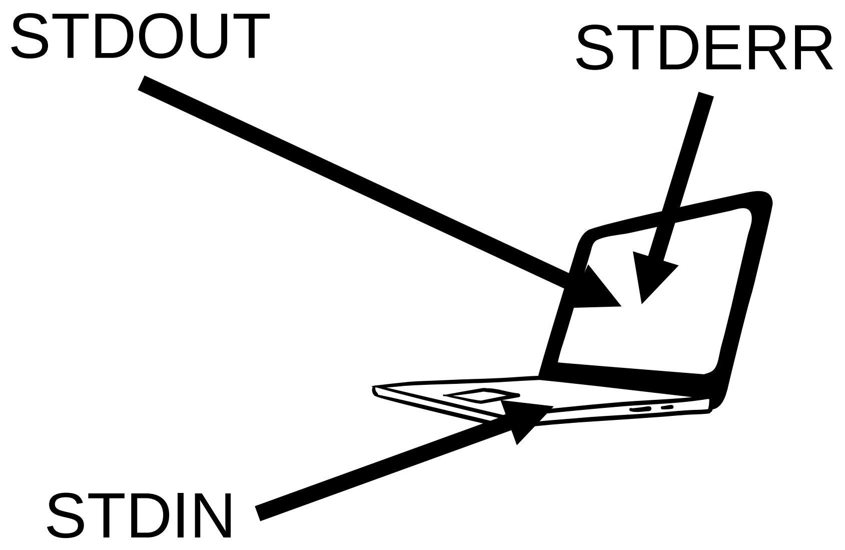


Image sources: https://openclipart.org/detail/300835/baby-2
https://creazilla.com/media/clipart/34014/baby-is-crying

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);
```

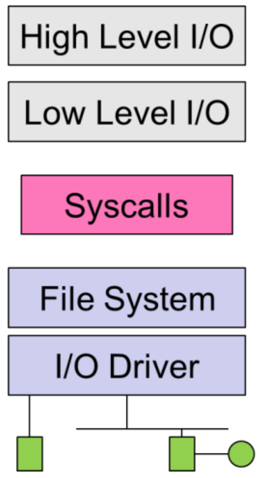
```
// formatted
int fprintf(FILE *restrict stream, const char *restrict format, ...);
int fscanf(FILE *restrict stream, const char *restrict format, ... );
```

Example Stream Code

```
#include <stdio.h>
#include <string.h>

#define BUFLEN 256
FILE *outfile;
char mybuf[BUFLEN];

int storetofile(){
          char *instring;

          outfile = fopen("/home/mjmay/Lecture5-Images/tokens", "w+");
          if (!outfile) {
                    return (-1); // Error!
          }
          while (1) {
                    instring = fgets(mybuf, BUFLEN, 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
}
```

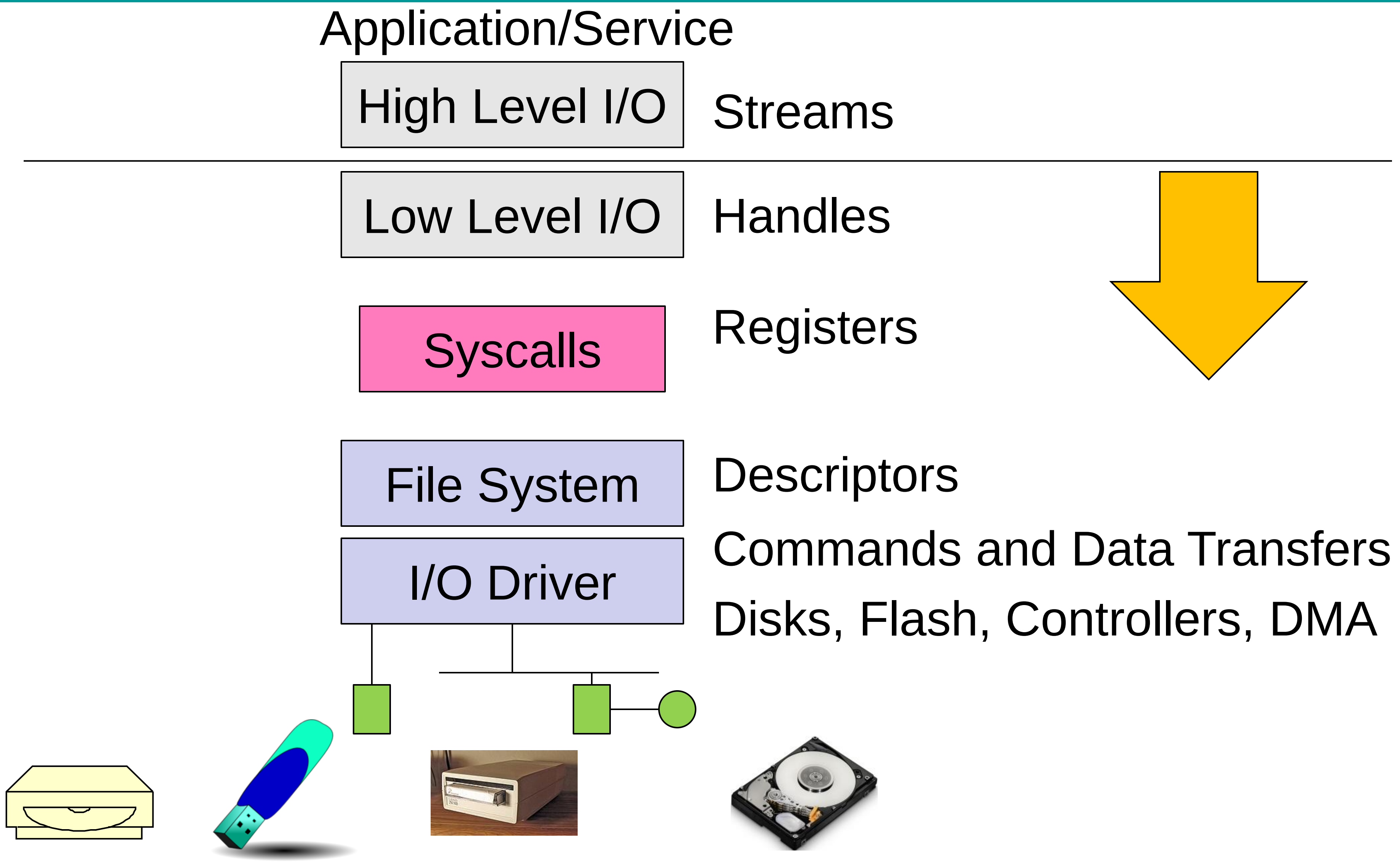
C Stream API positioning

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



- Preserves high level abstraction of a uniform stream of objects
- Adds buffering for performance (don’t forget to flush `fflush`)

What’s below the surface?



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

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

```
#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)
```

Flags: Bit Vector of:
- Access Mode (Rd, Wr, …)
- Open flags (Create,…)
- Operating modes (Appends,…)

mode: Bit Vector of Permission Bits
- User | Group | Other $\times$ R | W | X

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!

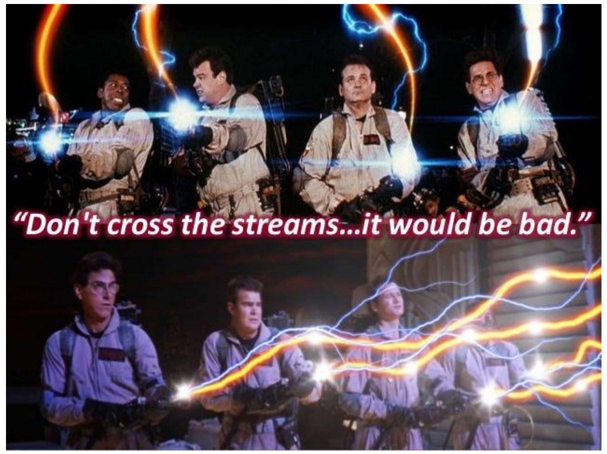


Image source: https://www.truthaboutthreats.com/p/never-cross-the-streams-it-would

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 fildes) /*– 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!

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)

freopen(const char
*filename, const char
*mode, FILE *stream)
```

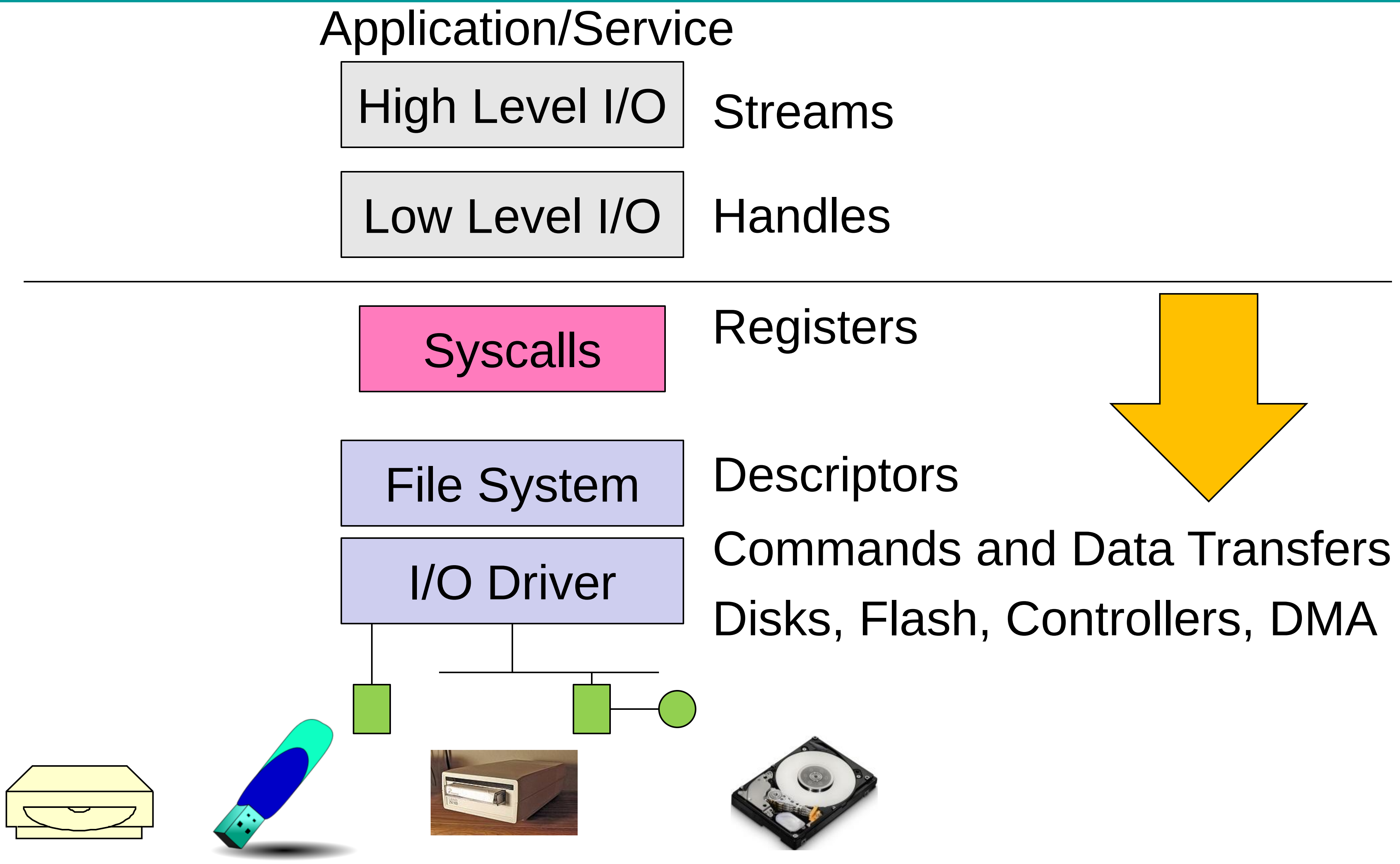
Example – lowio

```
#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);
}
```

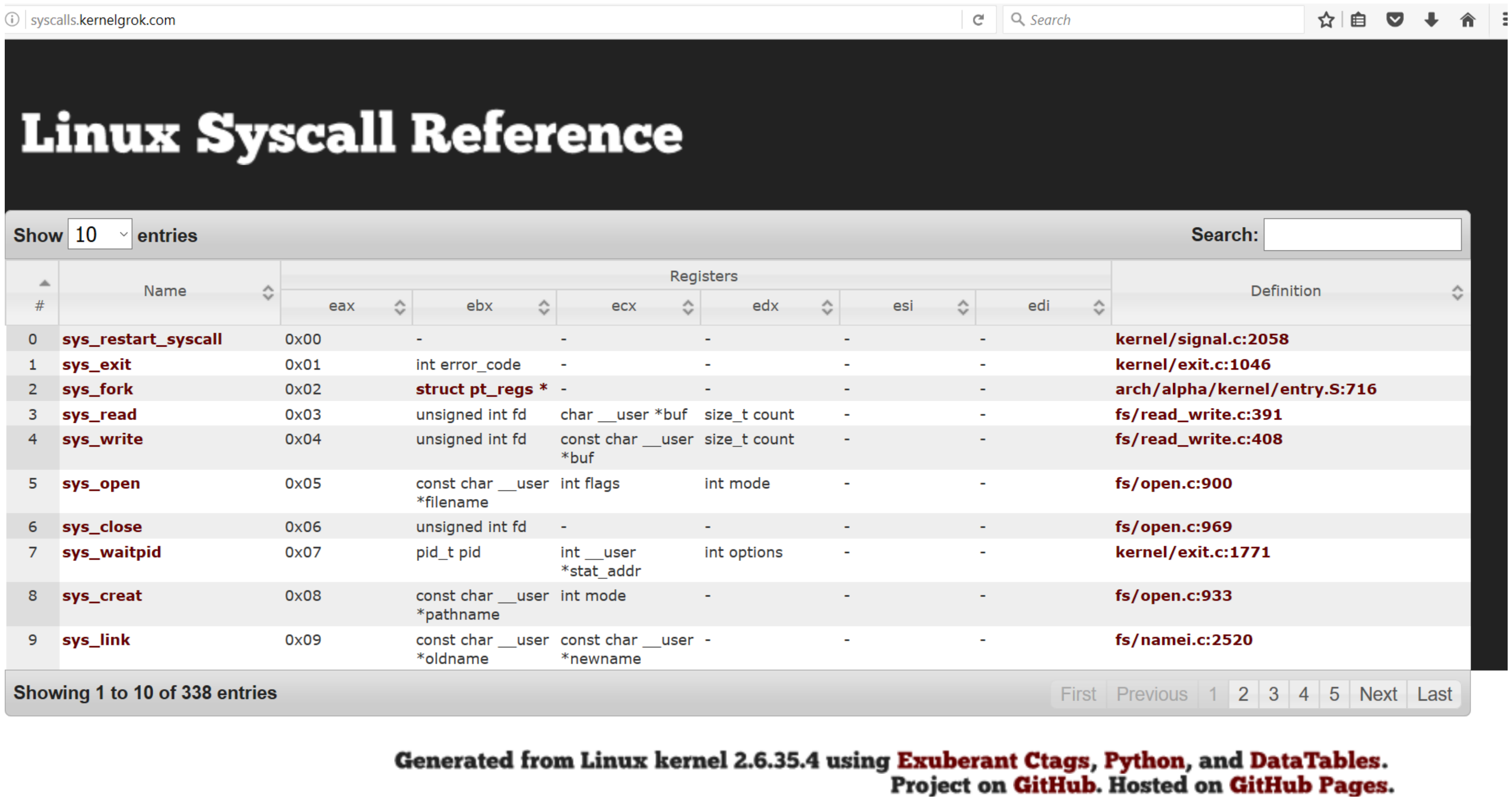
So Far

- Signals
- Syscalls
- Basic Support for I/O (drivers, etc.)
  - Files and Streams
  - Low Level
- I/O and Drivers
- Sockets and networks

What’s below the surface?



Linux Syscalls



syscalls.kernelgrok.com

Linux Syscall Reference

Show 10 entries Search:

#	Name	Registers						Definition
		eax	ebx	ecx	edx	esi	edi	
0	**sys_restart_syscall**	0x00	-	-	-	-	-	**kernel/signal.c:2058**
1	**sys_exit**	0x01	int error_code	-	-	-	-	**kernel/exit.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:391**
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/exit.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:2520**

Showing 1 to 10 of 338 entries

First Previous 1 2 3 4 5 Next Last

Generated from Linux kernel 2.6.35.4 using Exuberant Ctags, Python, and DataTables.
Project on GitHub. Hosted on GitHub Pages.

https://filippo.io/linux-syscall-table/

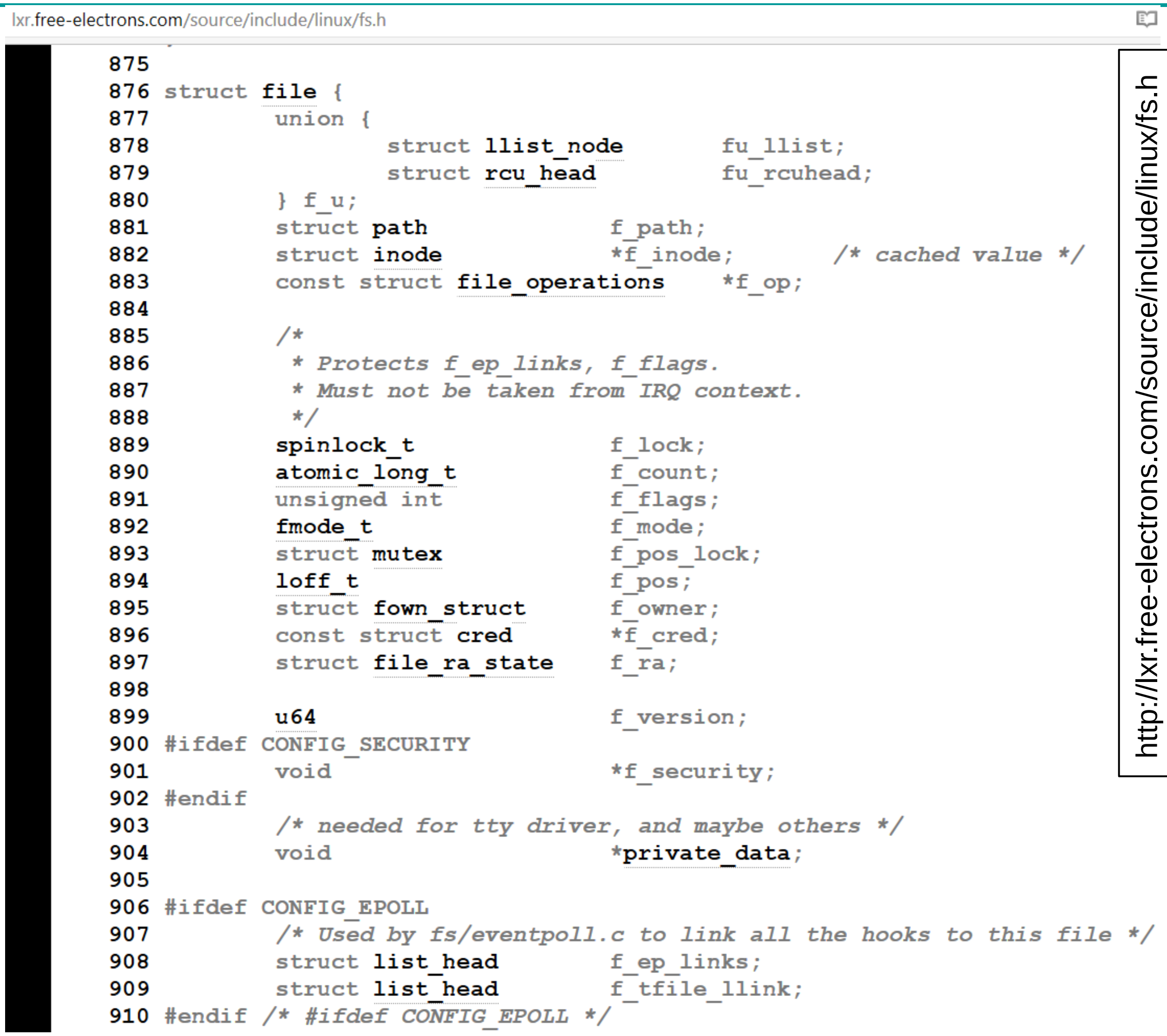
Linux Syscalls

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 the kernel

Internal OS File Descriptor

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

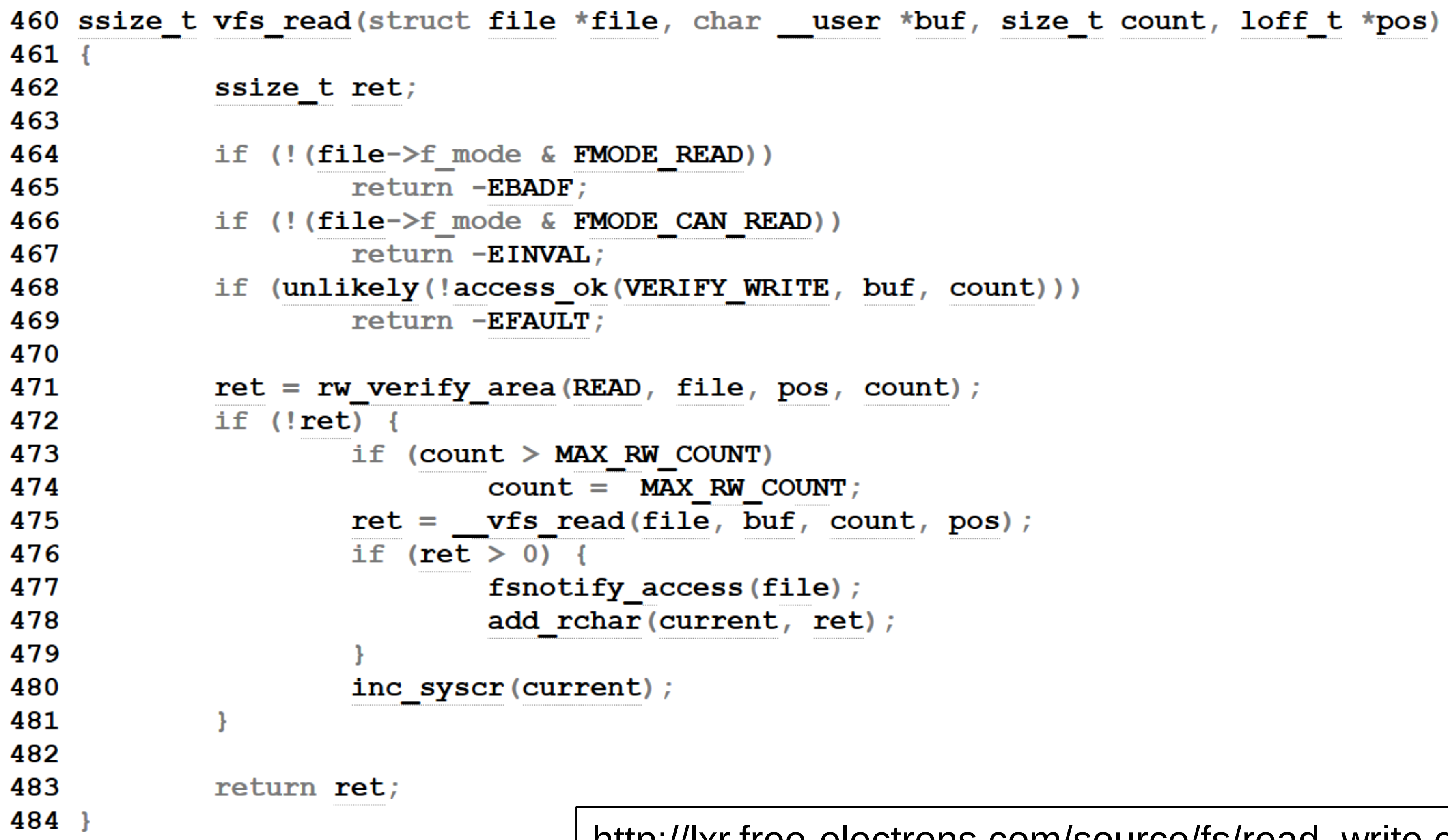


lxr.free-electrons.com/source/include/linux/fs.h

```

struct file {
        union {
                struct llist_node       fu_llist;
                struct rcu_head         fu_rcuhead;
        } f_u;
        struct path             f_path;
        struct inode            *f_inode;       /* cached value */
        const struct file_operations    *f_op;

        /*
         * Protects f_ep_links, f_flags.
         * Must not be taken from IRQ context.
         */
        spinlock_t              f_lock;
        atomic_long_t           f_count;
        unsigned int            f_flags;
        fmode_t                 f_mode;
        struct mutex            f_pos_lock;
        loff_t                  f_pos;
        struct fown_struct      f_owner;
        const struct cred       *f_cred;
        struct file_ra_state    f_ra;

        u64                     f_version;
#ifdef CONFIG_SECURITY
        void                    *f_security;
#endif
        /* needed for tty driver, and maybe others */
        void                    *private_data;

#ifdef CONFIG_EPOLL
        /* Used by fs/eventpoll.c to link all the hooks to this file */
        struct list_head        f_ep_links;
        struct list_head        f_tfile_llink;
#endif /* #ifdef CONFIG_EPOLL */
```

http://lxr.free-electrons.com/source/include/linux/fs.h

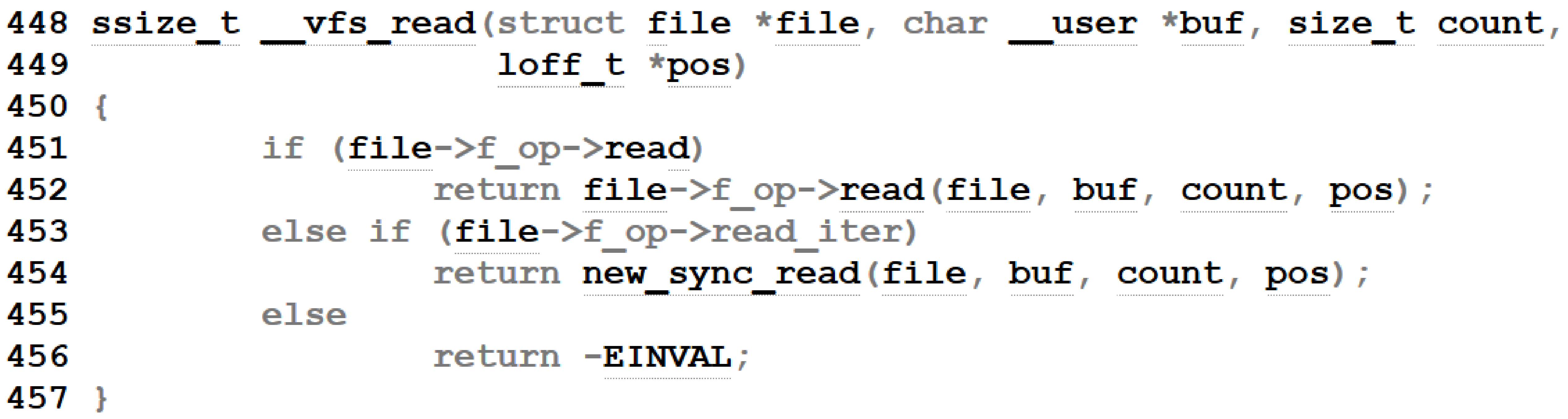
File System: from syscall to driver



```
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_mode & FMODE_CAN_READ))
                return -EINVAL;
        if (unlikely(!access_ok(VERIFY_WRITE, buf, count)))
                return -EFAULT;

        ret = rw_verify_area(READ, file, pos, count);
        if (!ret) {
                if (count > MAX_RW_COUNT)
                        count =  MAX_RW_COUNT;
                ret = __vfs_read(file, buf, count, pos);
                if (ret > 0) {
                        fsnotify_access(file);
                        add_rchar(current, ret);
                }
                inc_syscr(current);
        }

        return ret;
}
```

http://lxr.free-electrons.com/source/fs/read_write.c

File System: from syscall to driver

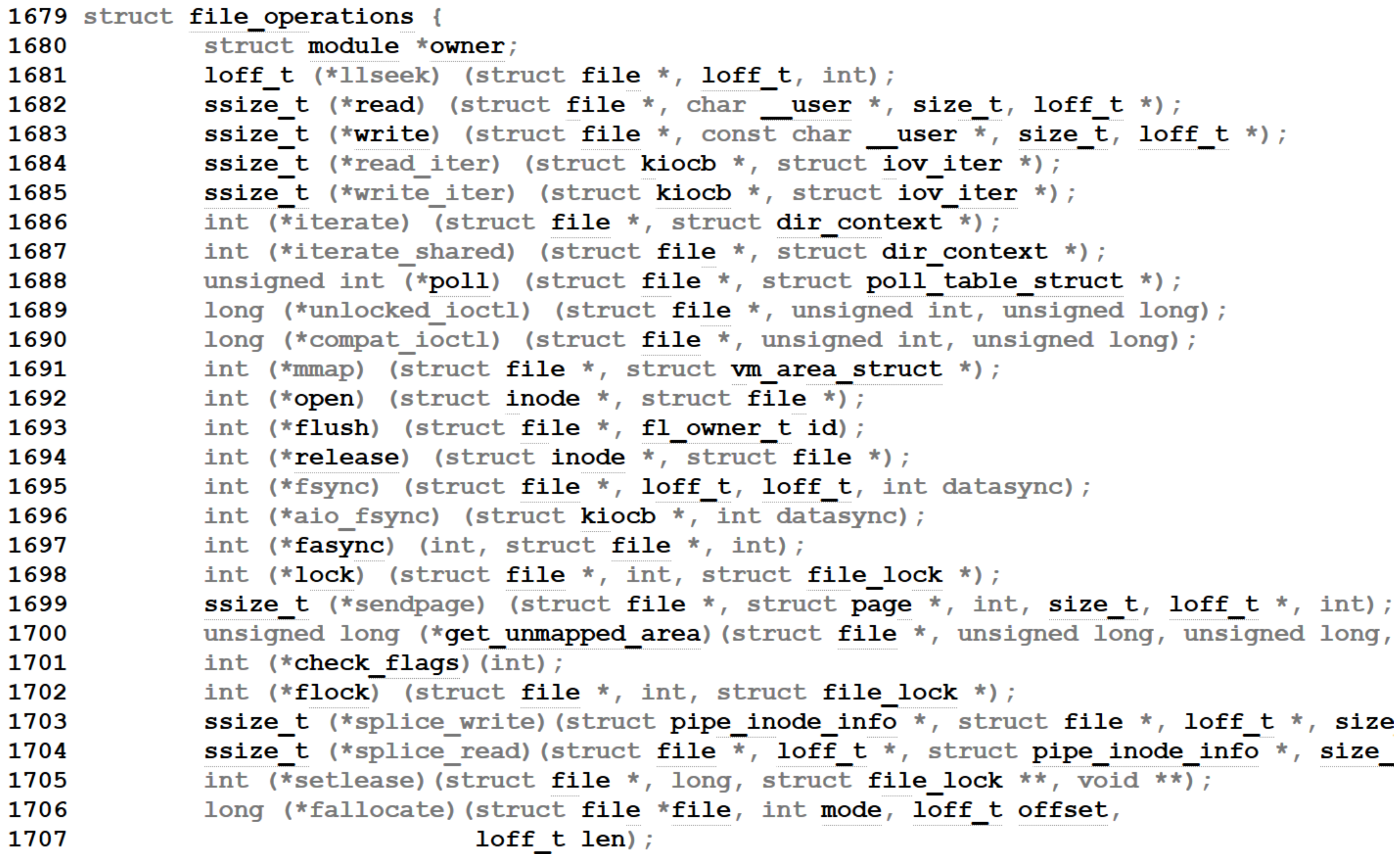


```
ssize_t __vfs_read(struct file *file, char __user *buf, size_t count,
                   loff_t *pos)
{
        if (file->f_op->read)
                return file->f_op->read(file, buf, count, pos);
        else if (file->f_op->read_iter)
                return new_sync_read(file, buf, count, pos);
        else
                return -EINVAL;
}
```

http://lxr.free-electrons.com/source/fs/read_write.c

Low Level Driver

http://lxr.free-electrons.com/source/include/linux/fs.h#L1679



```
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 (*read_iter) (struct kiocb *, struct iov_iter *);
        ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);
        int (*iterate) (struct file *, struct dir_context *);
        int (*iterate_shared) (struct file *, struct dir_context *);
        unsigned int (*poll) (struct file *, struct poll_table_struct *);
        long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
        long (*compat_ioctl) (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 *, loff_t, loff_t, int datasync);
        int (*aio_fsync) (struct kiocb *, int datasync);
        int (*fasync) (int, struct file *, int);
        int (*lock) (struct file *, int, struct file_lock *);
        ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
        unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long,
        int (*check_flags)(int);
        int (*flock) (struct file *, int, struct file_lock *);
        ssize_t (*splice_write)(struct pipe_inode_info *, struct file *, loff_t *, size
        ssize_t (*splice_read)(struct file *, loff_t *, struct pipe_inode_info *, size_
        int (*setlease)(struct file *, long, struct file_lock **, void **);
        long (*fallocate)(struct file *file, int mode, loff_t offset,
                          loff_t len);
```

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

Low Level Driver

Photo by Lewis J Goetz on Unsplash

Associated with particular hardware device

Registers / Unregisters itself with the kernel

Handler functions for each of the file operations

Life Cycle of An I/O Request

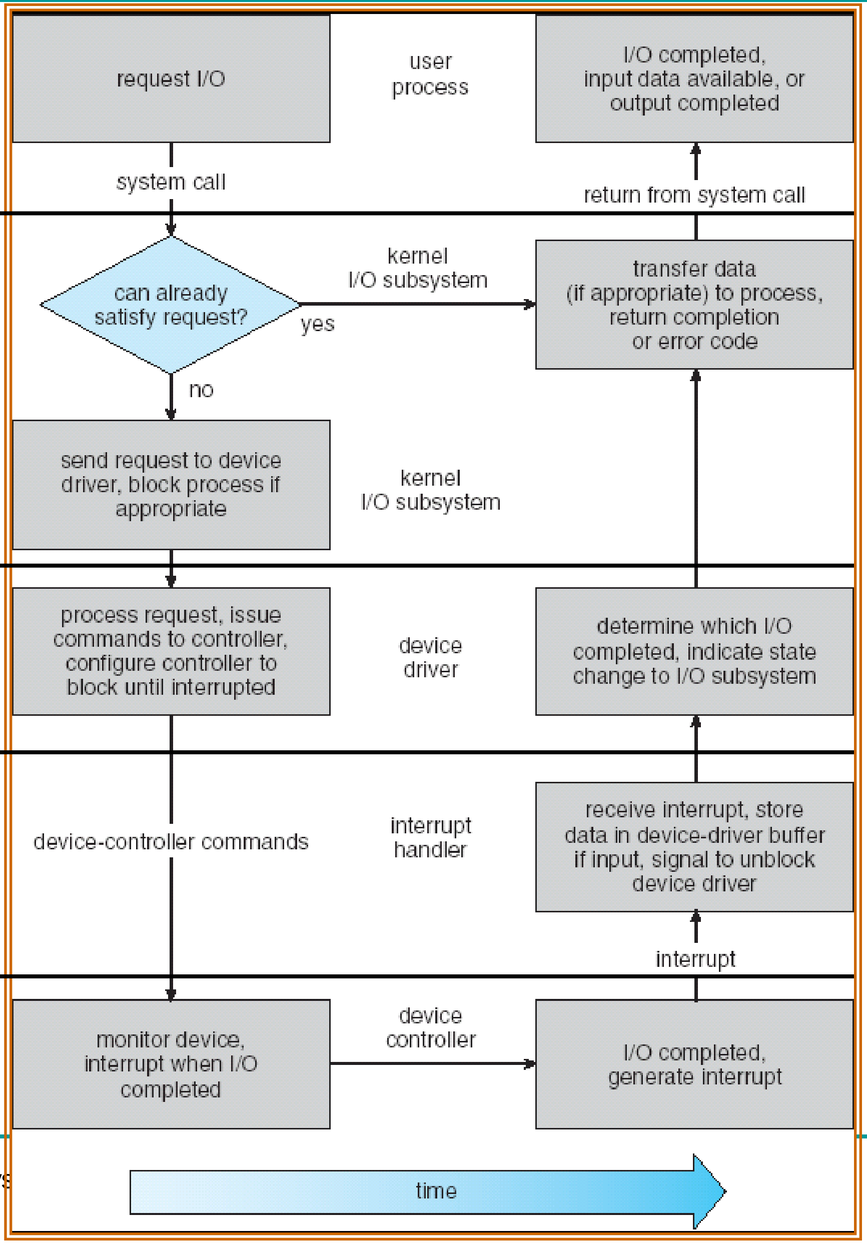
User Program

Kernel I/O Subsystem

Device Driver top half

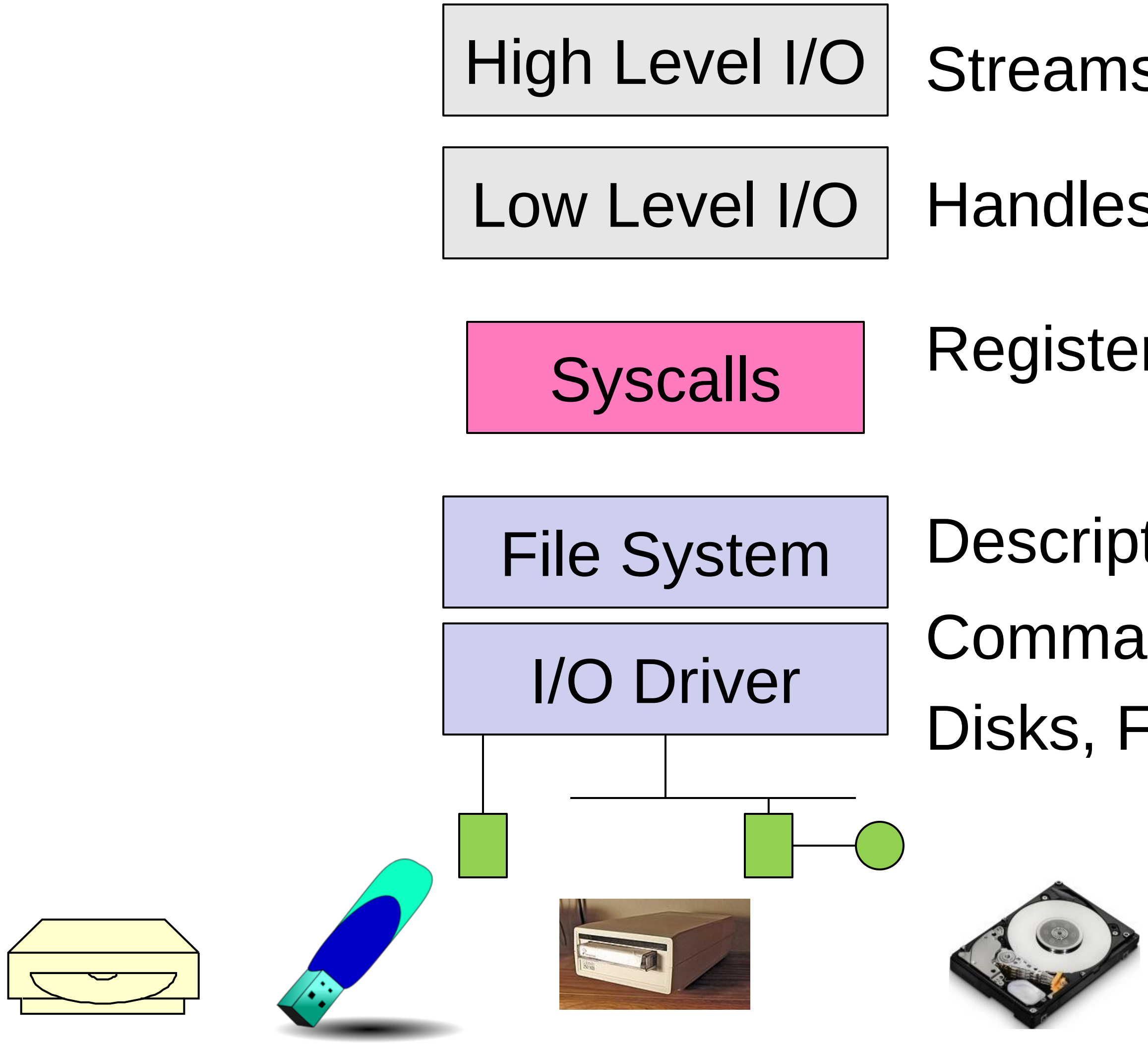
Device Driver bottom half

Device Hardware


request I/O
user process
I/O completed, input data available, or output completed
system call
return from system call
can already satisfy request?
kernel I/O subsystem
yes
transfer data (if appropriate) to process, return completion or error code
no
send request to device driver, block process if appropriate
kernel I/O subsystem
process request, issue commands to controller, configure controller to block until interrupted
device driver
determine which I/O completed, indicate state change to I/O subsystem
device-controller commands
interrupt handler
receive interrupt, store data in device-driver buffer if input, signal to unblock device driver
interrupt
monitor device, interrupt when I/O completed
device controller
I/O completed, generate interrupt
time

So what happens when you `fgetc`?

Application/Service



Streams

Handles

Registers

Descriptors

Commands and Data Transfers

Disks, Flash, Controllers, DMA

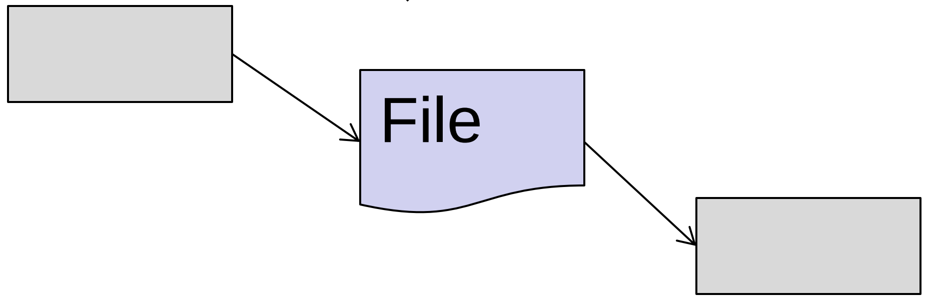
So Far

- Signals
- Syscalls
- Basic Support for I/O (drivers, etc.)
  - Files and Streams
  - Low Level
- I/O and Drivers
- Sockets and networks

Communication Between Processes

- Can we view files as communication channels?

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

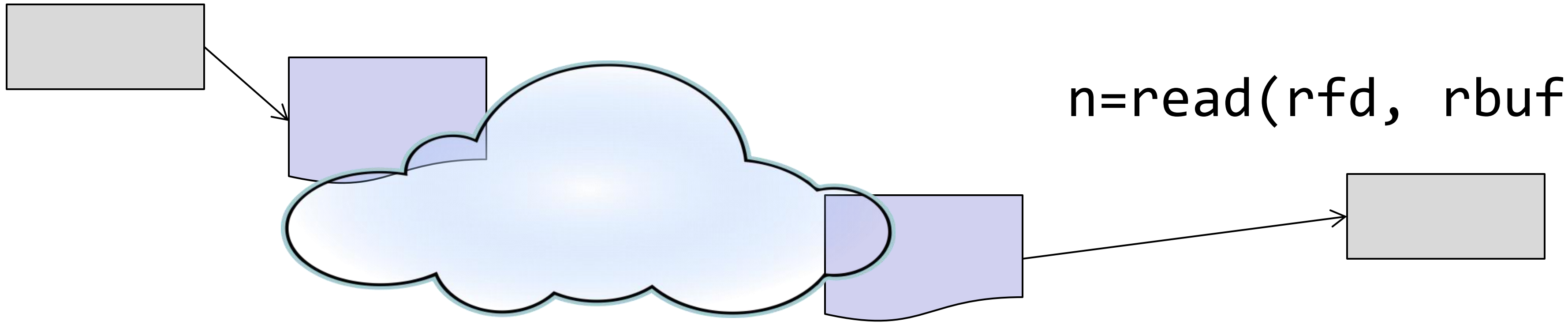


```
n=read(rfd, rbuf, rmax);
```

- Producer and Consumer of a file may be different processes
  - May be separated in time (or not)
- However, what if data written once and consumed once?
  - Would be more like a queue, but still look like File I/O!

Communication across the world

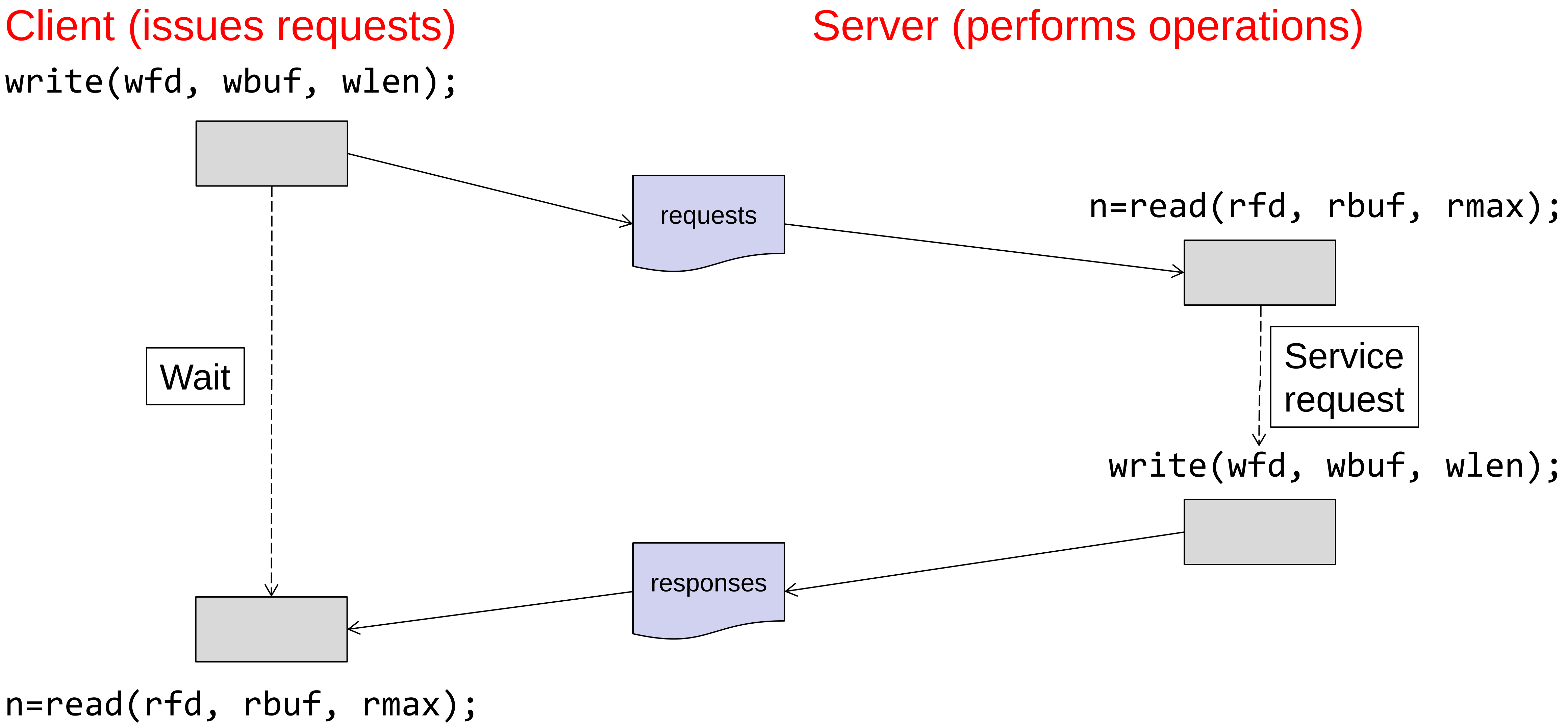
```
write(wfd, wbuf, wlen);
```



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

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

Request Response Protocol

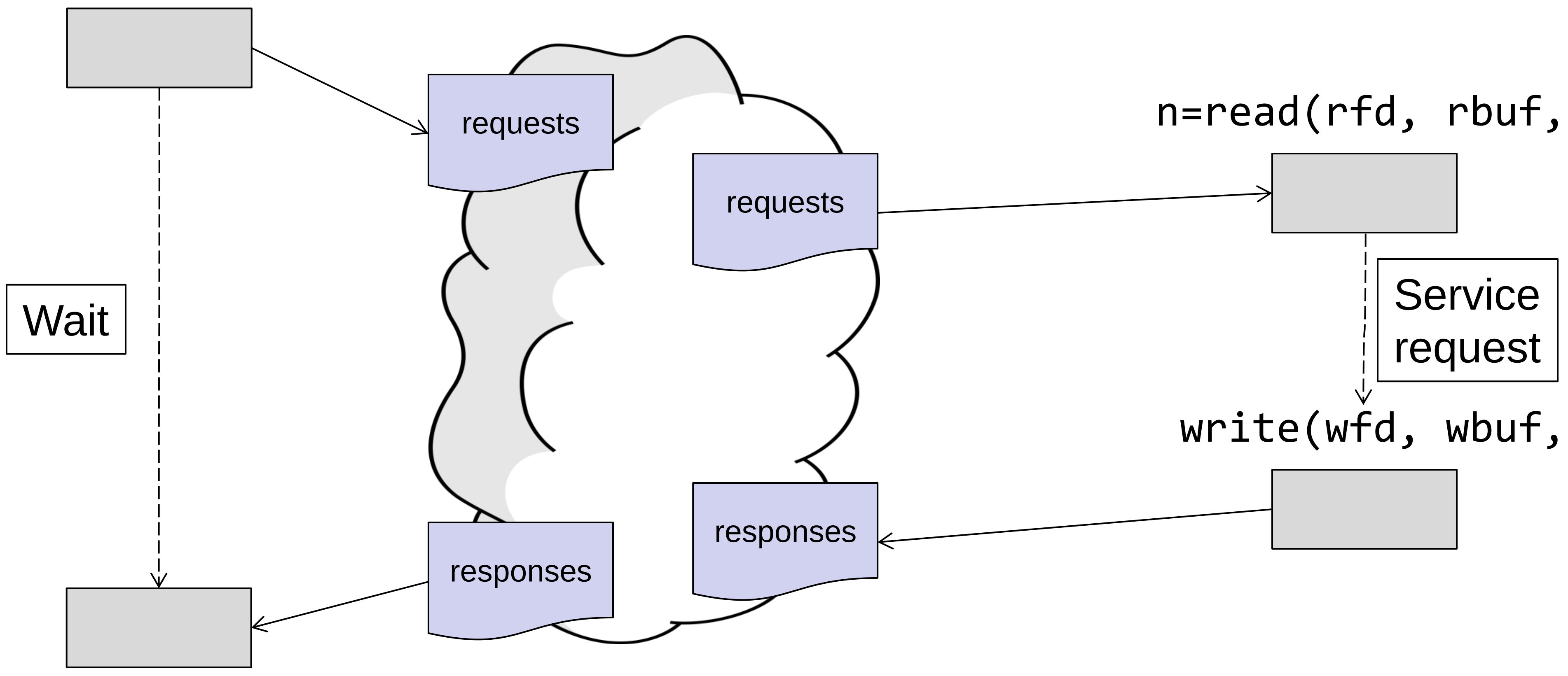


Request Response Protocol

Client (issues requests)

Server (performs operations)

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

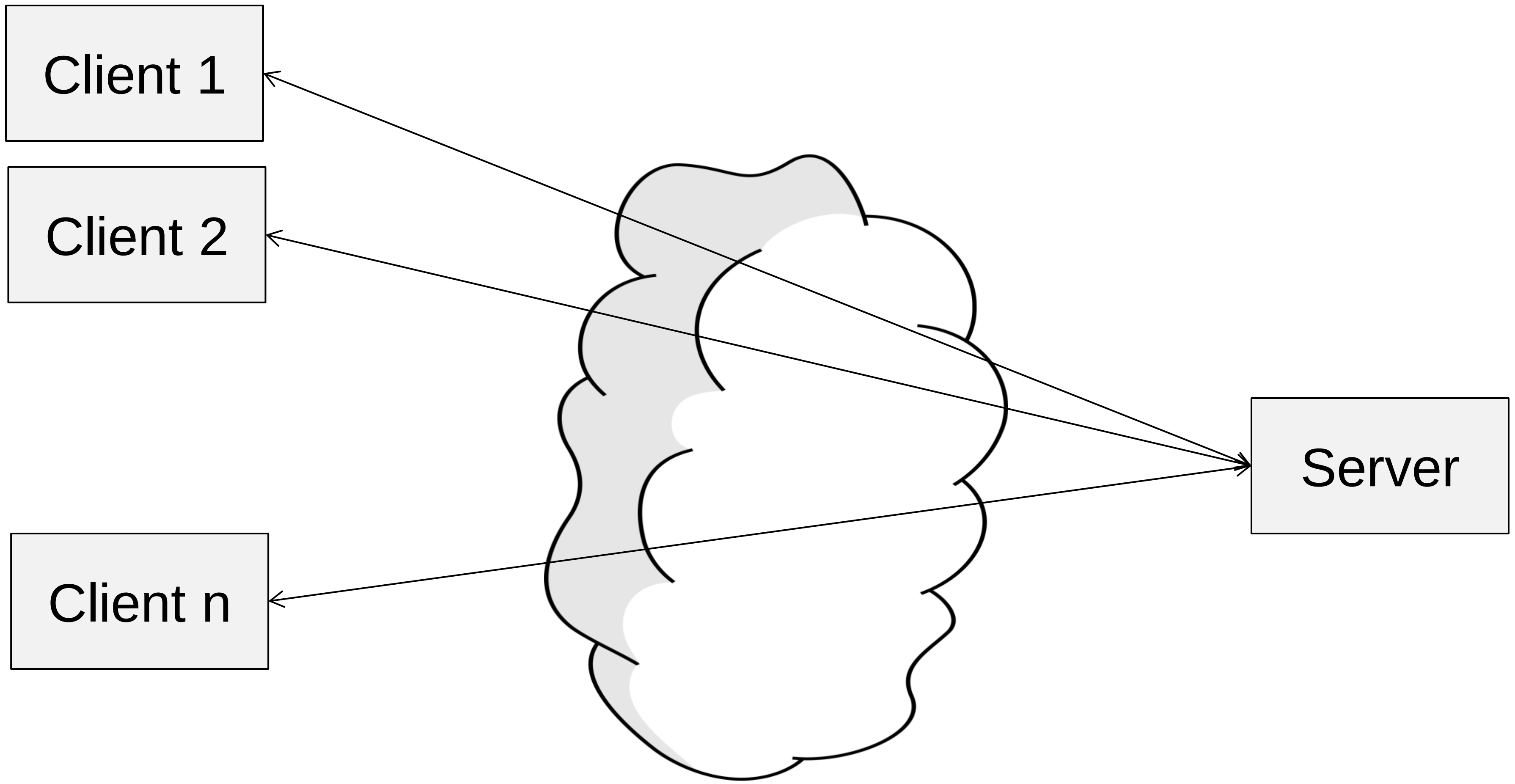


```
n=read(rfd, rbuf, rmax);
```

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

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

Client-Server Models



- File servers, Web servers, FTP servers, Databases
- Many clients access a common server

Sockets

Socket: an abstraction of a network I/O queue

- Mechanism for inter-process communication
- Embodies one side of a communication channel
  - Same interface regardless of location of other end
  - Could be local machine (“UNIX socket”) or remote machine (“network socket”)
- First introduced in 4.2 BSD UNIX: big innovation at time
  - Now most operating systems provide some notion of socket

Data transfer like files

- Read / Write against a descriptor

Over any kind of network

- Local to a machine
- Over the internet (TCP/IP, UDP/IP)
- OSI, Appletalk, SNA, IPX, SIP, NS, …

Socket Creation and Connection

File systems

- Provide a collection of permanent objects in structured name space
- Processes open, read/write/close them
- Files exist independent of the processes

Sockets

- Provide a means for processes to communicate (transfer data) to other processes.
  - Creation and connection is more complex
  - Form 2-way pipes between processes, possibly worlds away

Conclusion

- Signals
- Syscalls
- Basic Support for I/O (drivers, etc.)
  - Files and Streams
  - Low Level
- I/O and Drivers
- Sockets and networks