CS 134
Operating Systems

January 23, 2019

Overview
Brief Introduction to JOS and xv6

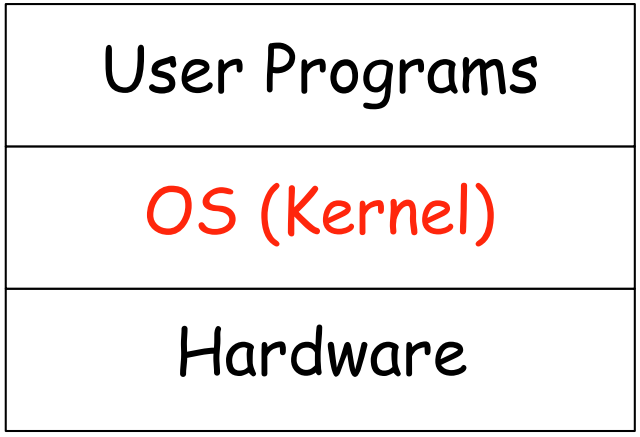
Instructor: Neil Rhodes

Introduction

- What will you be learning?
  - What an operating system is
    - Various capabilities provided by an OS
  - Distinction between an OS/Kernel and the rest of the system
  - How an operating system is written
  - The spectrum of services provided by an OS
    - Lightweight/small/fast
    - More capable/larger/slower
- Why?
  - All programmers interact with an OS
    - Care about what's happening internally
    - Remove black-box
    - Need to write high-performance programs
    - Need to diagnose bugs or security problems
  - Some programmers must write an OS (or modify)
- What is the structure?
  - Combination of theory and practice

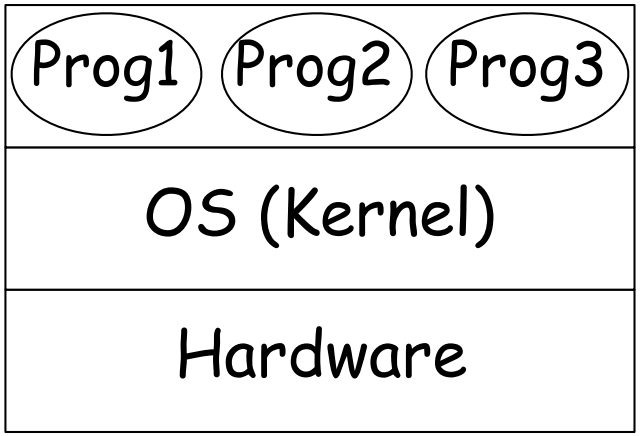
Operating System

- A level of software between programs and the raw hardware



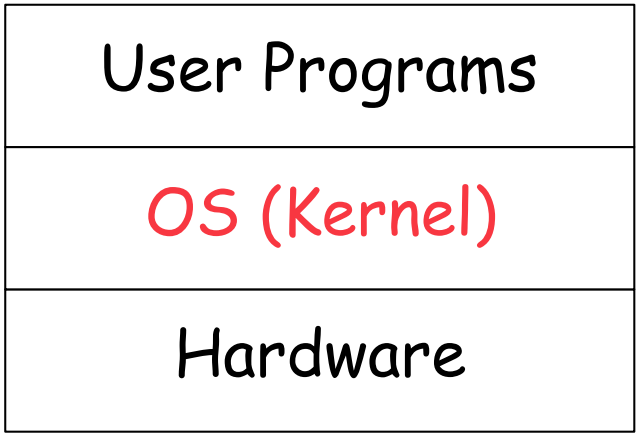
What an Operating System Does (view number 1)

- Manages resources needed by various programs
  - _____
  - _____
  - _____
  - _____
- Time multiplexing
  - _____
  - _____
  - _____
- Space multiplexing
  - _____
  - _____



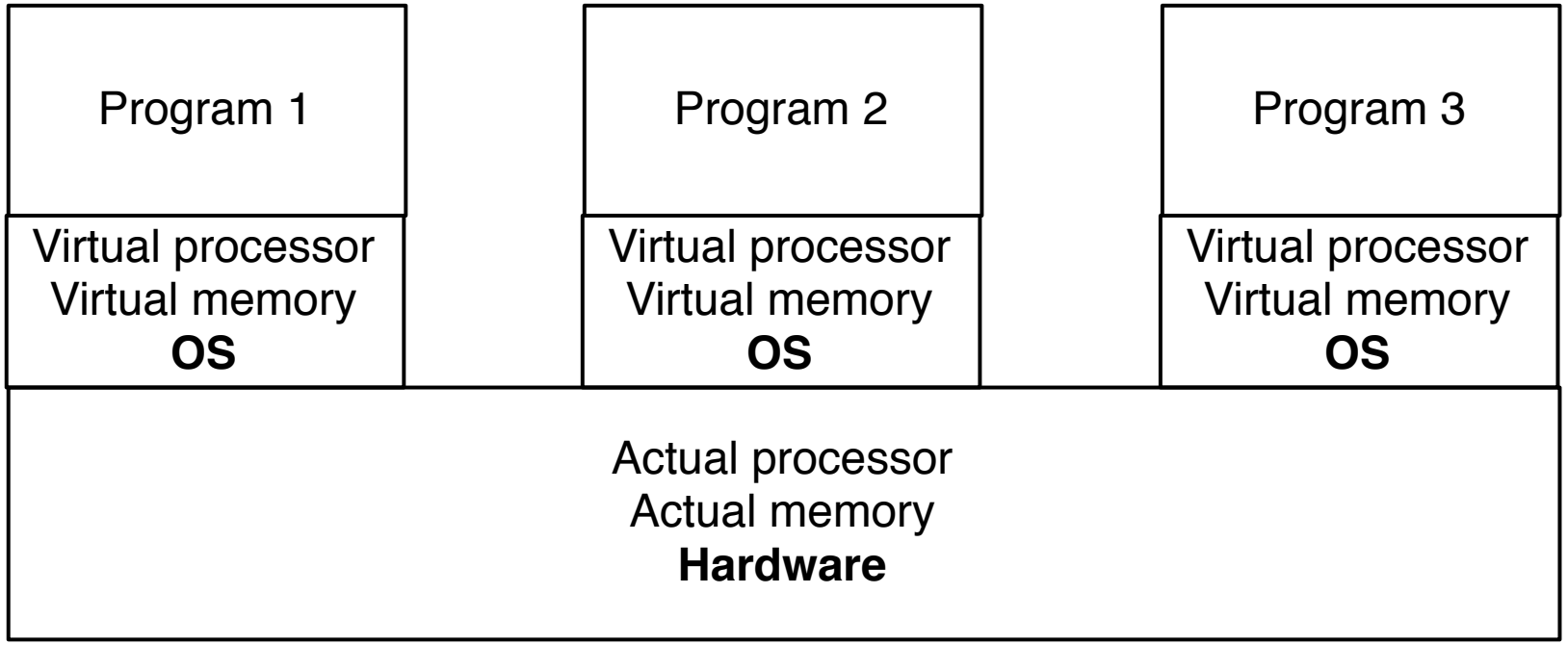
What an Operating System Does (view number 2)

- Operating system as extended machine
  - The operating system provides an interface higher than that of the raw machine
    - Level 1: write to the floppy disk controller directly
      - Start motor spinning
      - Move the disk arm
      - …
      - Stop motor spinning
    - Level 2: write to floppy disk sector-by-sector
      - Impose your own structure on the floppy.
    - Level 3: Open/close files.
      - use file system



  - Provide abstractions
    - Same low-level functions to write to IDE drive as SATA drive as USB drive
    - Same high-level functions to write to a file on disk as to a network connection

What an Operating System Does (view number 2)



The Kernel

- Uses supervisor mode of the CPU
  - Certain operations only available if processor is running in supervisor mode
    - Enable/Disable ***interrupts***
    - Change memory map
    - Change supervisor mode
- Normal applications run in user mode
  - Hardware disallows some operations
  - How to call to kernel?
  - Make System call
    - Put system call number and parameters in special location (registers?)
    - Issue special TRAP instruction
    - Causes execution to switch to kernel's trap handler (now in supervisor mode)
    - Makes call to appropriate system call handler
    - Returns to user mode with special instruction

The Kernel

- Handling input
  - I/O device generates an *interrupt*
  - Causes execution to switch to kernel's interrupt handler (now in supervisor mode)
  - Deals with that particular interrupt
  - Returns to where it was interrupted from with special instruction

The Kernel

- Services
  - Processes
  - Memory allocation
  - File system
    - Directories/filenames
    - File contents
    - Attributes about files
  - Security
  - Networking
  - …

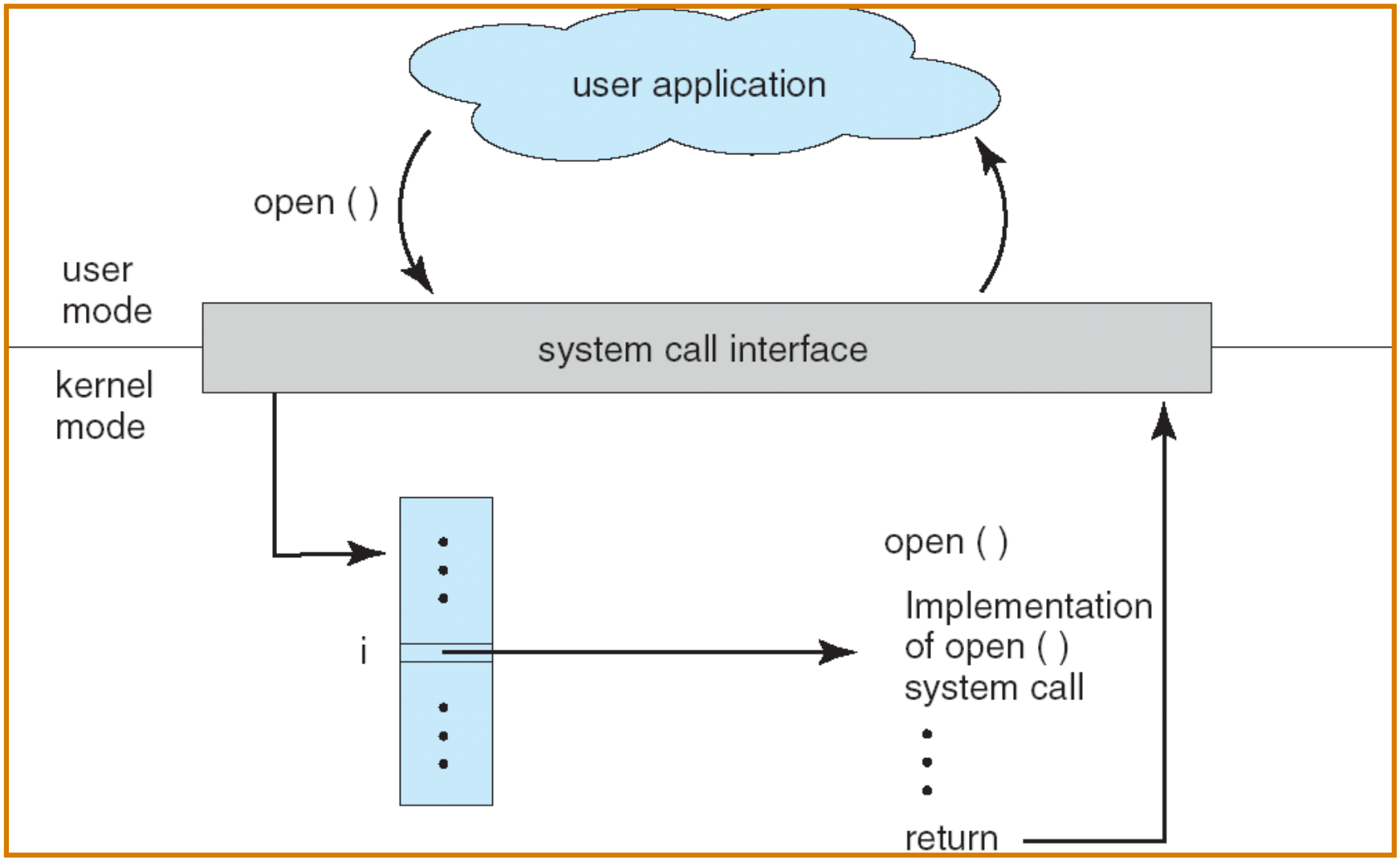
Operating System Abstractions

- System calls
  - A function that exists in the kernel, called by the application
  - For example:

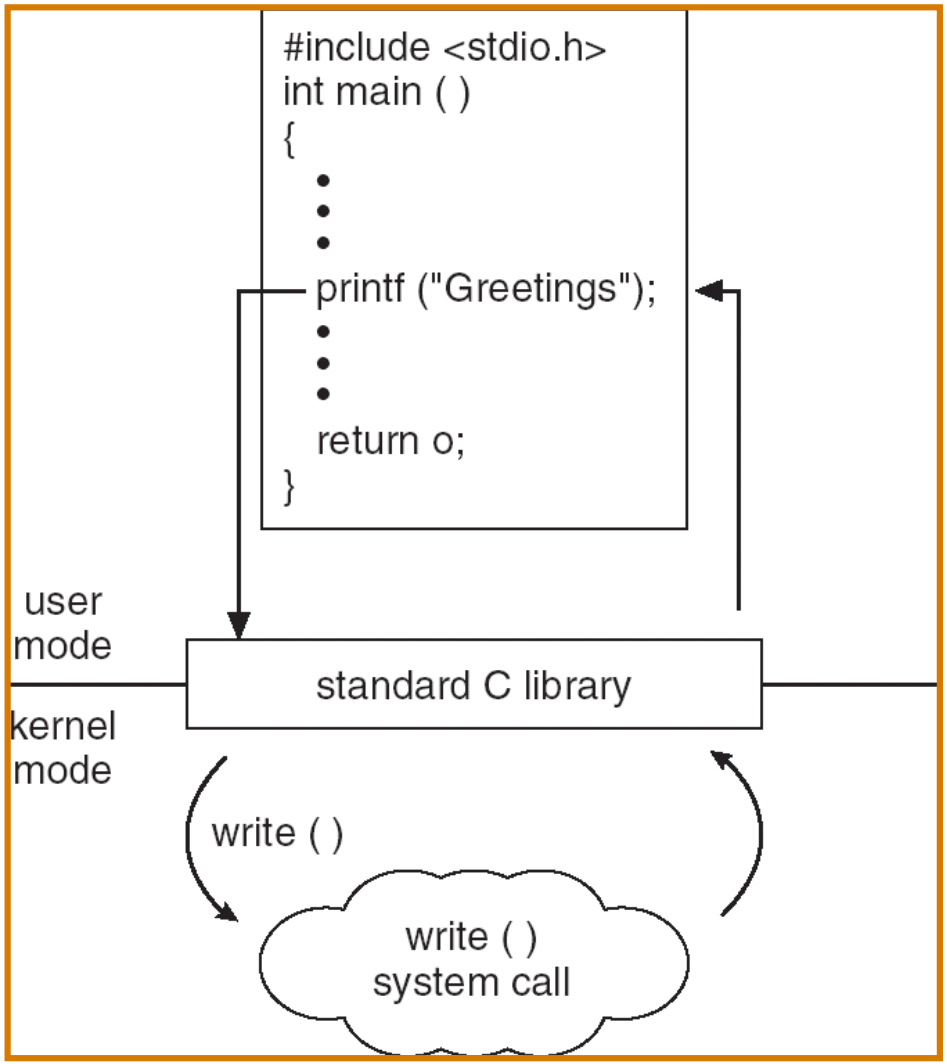
```
int fd = open('myfile.txt', 1);
write(fd, "hello, world!\n", 14);

int pid = fork();
```

API – System Call – OS Relationship



Standard C Library Example



Why is Operating System design hard?

- Difficult environment
- Want efficiency, but also portability
- Want powerful, but also simple
- Interactions between different calls
  - Example:

Lecture format

- Operating System ideas
- Detailed inspection of xv6, a traditional OS
- xv6 programming homework will motivate lectures

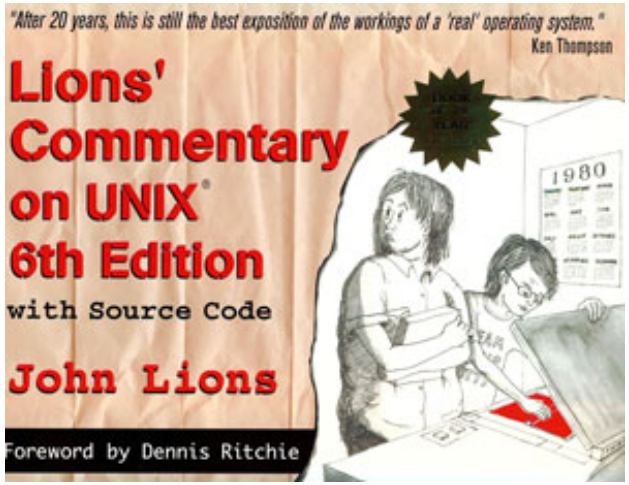
Labs

Lab 1 is out now!

- You'll be building up JOS, a small OS for x86 (exokernel style)
- Kernel interface: expose, but protect
- Unprivileged user-level library:
  - fork
  - exec
  - pipe
  - …
- Dev environment
  - GCC
  - qemu

Homeworks

- Almost all using xv6
- Dev environment
  - GCC
  - qemu
- Lions' commentary
  - Unix v6 allowed classroom use of source code
  - Unix v7+ did not
  - Lions book provided commentary + source code
  - Very widely copied (samizdat style)
- xv6: v6 for x86



xv6 vs. JOS

- Two different small x86 OSes
  - xv6: traditional, similar in spirit to Unix v6
  - JOS: exokernel style (much work done in user mode)
  - Neither one as complicated as modern Linux

```
From: torv...@klaava.Helsinki.FI (Linus Benedict Torvalds)
Date: 25 Aug 91 20:57:08 GMT
Organization: University of Helsinki

Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and
professional like gnu) for 386(486) AT clones.  This has been brewing
since april, and is starting to get ready.  I'd like any feedback on
things people like/dislike in minix, as my OS resembles it somewhat
(same physical layout of the file-system (due to practical reasons)
among other things).

I've currently ported bash(1.08) and gcc(1.40), and things seem to work.
This implies that I'll get something practical within a few months, and
I'd like to know what features most people would want.  Any suggestions
are welcome, but I won't promise I'll implement them :-)

                    Linus (torv...@kruuna.helsinki.fi)

PS.  Yes - it's free of any minix code, and it has a multi-threaded fs.
It is NOT protable (uses 386 task switching etc), and it probably never
will support anything other than AT-harddisks, as that's all I have :-(.
```

Labs

- Six programming labs
  - First five are solo
  - Last one can be done solo or in groups of up to 3
  - Modifying an existing operating system: Jos
    - Jos is written in C. You'll modify/extend the OS by writing Java code
    - Jos runs on x86. Rather than running on bare metal, we'll run on a Virtual Machine: QEMU

Resources

- Webpage (including announcements, schedules, etc.)
  - http://cs.hmc.edu/~rhodes/cs134
- Discussion board
  - https://piazza.com/hmc/spring2019/cs134

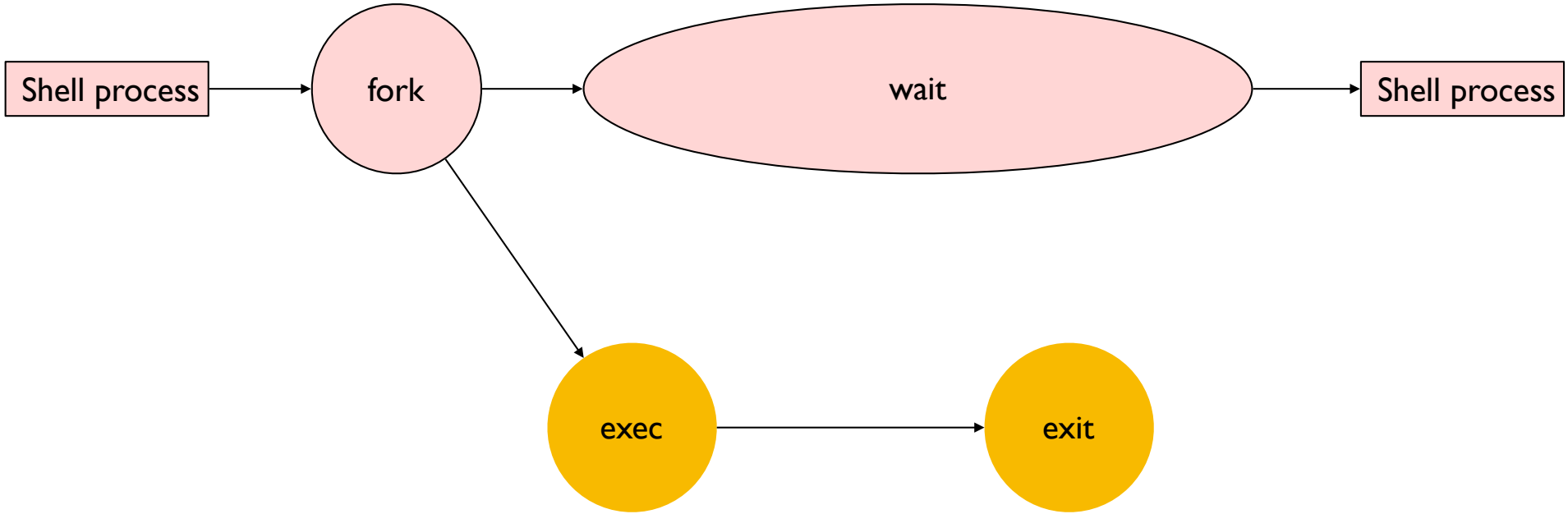
Our focus

- System calls
- Unix-like systems:
  - Linux/Posix/Mac OS

Shell

- User-level program that is the command-line interface
- Supports:
  - redirection
    - `ls > output`
    - `mail "neil@pobox.com" < my message`
    - `ls | wc -l`
- How does shell get control again after command executes?
  - Uses `fork` to run the command in a child process
    - Copies user memory
    - Copies kernel data structures (e.g., open file descriptors)
    - Child is now a clone of parent (with different process ID)
    - Both are now running the same program!
      - How make them act different?

Shell: executing a command



Shell: implementing Pipeline

```
ls | wc -l
```