Operating Systems

CS 3281

Spring 2023

https://github.com/cs3281/lectures

ISIS
Tel (615) 343-7472 Fax (615) 343-7440
1025 16th Avenue South|Nashville, TN 37212
www.isis.vanderbilt.edu

VANDERBILT UNIVERSITY

Team

- Instructors
  - Dr. Daniel Balasubramanian, Dr. Shervin Hajiamini
- Graduate TAs
  - Oluwatito Ebiwonjumi, Yi Zhang
- Graders
  - Ritvik Singh, Lucas Smulders, Rohit Khurana, Wesley Minton, Xiaoliang Zhu

Important Links

Textbook	http://pages.cs.wisc.edu/~remzi/OSTEP/
Discussion Forum/ Question and Answers/ Announcements	https://piazza.com/class/l6lmo7515wi3wb
Lectures, schedule, assignment submissions	https://github.com/cs3281/
Reading Assignments	https://vanderbilt.edu/brightspace/

GitHub organization

- Please fill out this form to be added to the GitHub organization: https://forms.gle/dHjPJcjtGyhvNR2T8
- Once you fill this out, you'll get an email inviting you to join the GitHub organization
  - Please accept this invitation

Office Hours (Dr. Hajiamini)

- Listed on syllabus (https://github.com/cs3281/lectures/syllabus.md)

- Mondays, Wednesdays, and Fridays

  FGH 384, 10:00 am – 11:00 am

Office Hours (Dr. Balasubramanian)

- Listed on syllabus (https://github.com/cs3281/lectures/syllabus.md)

MAILING/PHYSICAL ADDRESS:
VUSE-ISIS building
1025 16th Ave S, Suite 102
Nashville, TN 37212

CAMPUS MAIL ADDRESS:
PMB351829
Nashville, TN 37235

PHONE NUMBERS:
ISIS tel: +1 (615) 343-7472
ISIS fax: +1 (615) 343-7440

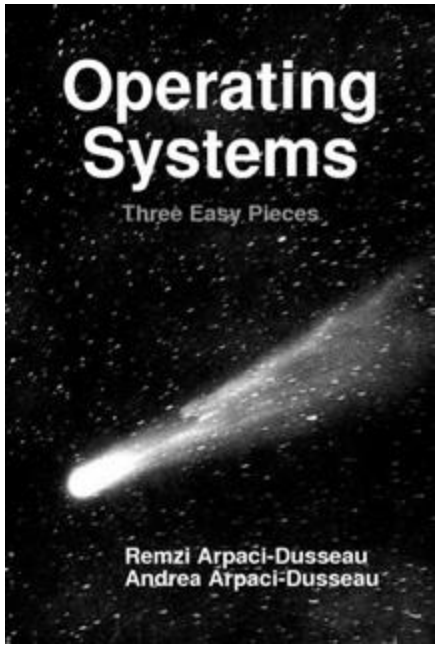
SYSTEM/WEB ADMINISTRATOR: sysadmin (at) isis (dot) vanderbilt (dot) edu

ISIS Location - 1025 16th Ave S, Nashville

View Larger Map

Textbook

- We will draw material from a variety of sources
- Primary textbook: Operating Systems: Three Easy Pieces
  - Chapters available for free at http://pages.cs.wisc.edu/~remzi/OSTEP/
- Other texts:
  - The Linux Programming Interface
  - Computer Systems: A Programmer's Perspective
  - Linux Kernel Development



Course Assessment

- Programming assignments: 60%
  - Learn by doing
- Mid-term exam: 10%
- End-term exam: 10%
  - Covering the second half of the course material
- Final project: 20%

Late days

- You have a total of 7 late days that you can use across programming assignments as you wish
  - A maximum of two late days can be used on a given programing assignment.
  - Example: assignment is due by 11:59pm Monday; you can use two late days to submit that assignment by 11:59pm Wednesday with no penalty
- To use late days: push a file named late_days.md to the top-level directory of your assignment repo with a line stating whether you're using one or two late days
- Assignments submitted more than two days late will not be accepted
- **No collaborations unless explicitly permitted**.

Course expectations

- You are expected to read the material for a lecture beforehand and participate in class discussions.
  - We may occasionally assign videos to watch; please watch them ahead of time so you can participate in class discussions

Course expectations: office hours

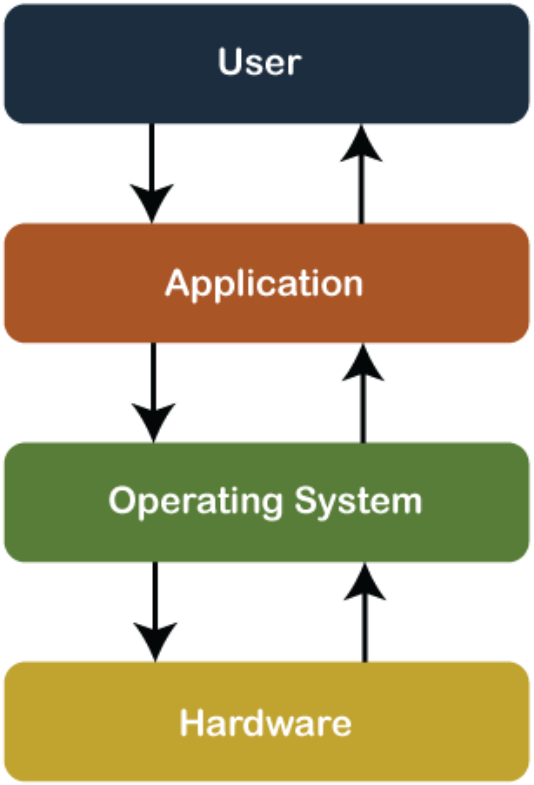
- We will use an office hour policy similar to the one listed here: https://www2.seas.gwu.edu/~gparmer/resources/2021-09-20-Office-Hours-HOWTO.html
  - Please read this policy carefully and adhere to its guidelines. It will help you and us.

Development Environment

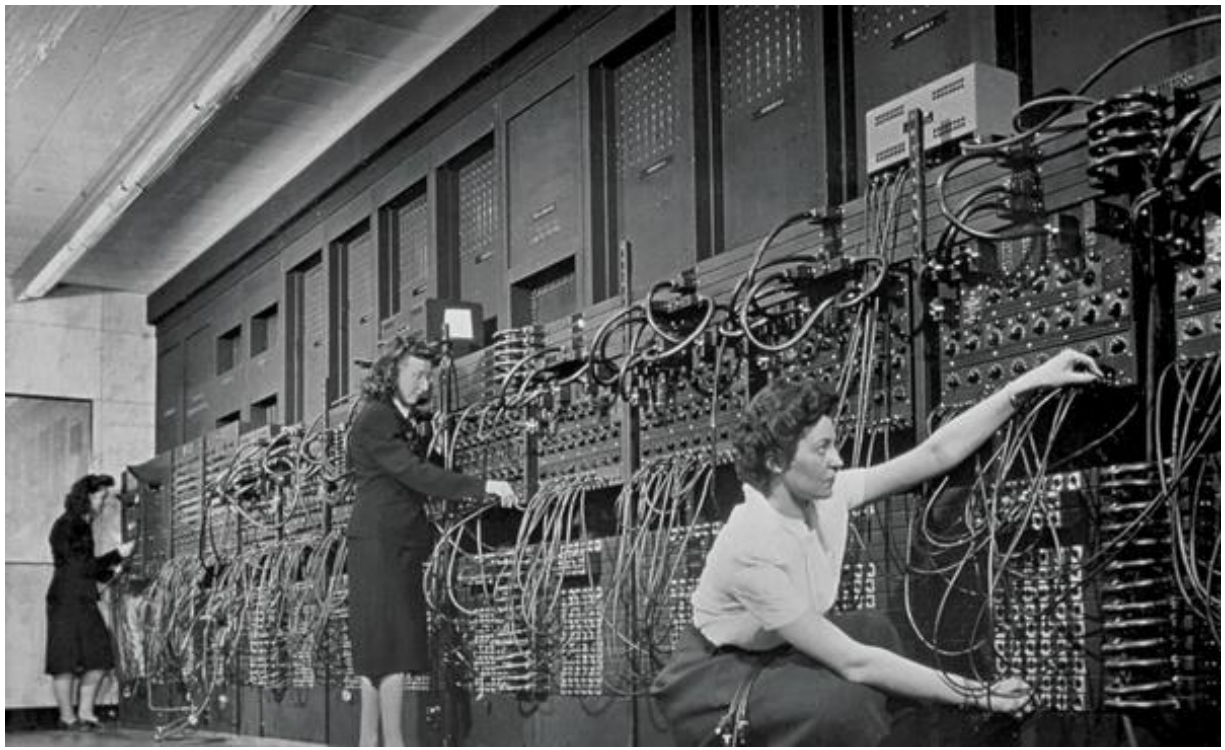
- Most of this course will use C
- We will use Ubuntu 20.04 LTS as the development environment
  - You're free to choose your particular Linux distribution as long as you can compile and run the assignments
  - If we have issues compiling/running your programs, we'll request to switch to an Ubuntu-based distribution
- We will use GitHub and git for content and assignment management

Course Goals

- Understand operating systems by learning their architecture and services
- Experience with writing applications that use operating-system services



Historical Perspective

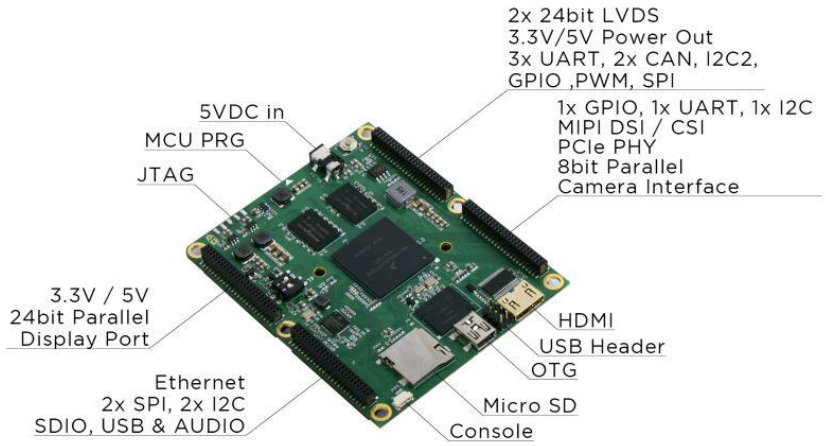


Early computers did not have an operating system. People manually performed functions that are now controlled in software systems that operate the machine

Why is this important?

- Operating system is responsible for
  - Abstracting the hardware details for convenience and portability
  - Multiplex the hardware among multiple applications
  - Isolate applications to contain bugs
  - Allow sharing of resources among applications

Why is this important?

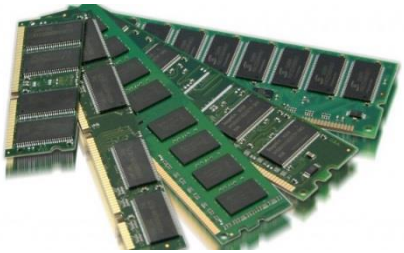


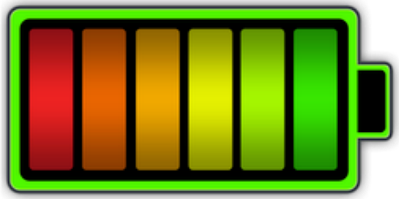
- Operating system is responsible for
  - Abstracting the hardware details for convenience and portability
  - Multiplex the hardware among multiple applications
  - Isolate applications to contain bugs
  - Allow sharing of resources among applications

Why is this important?

- Operating system is responsible for
  - Abstracting the hardware details for convenience and portability
  - Multiplex the hardware among multiple applications
  - Isolate applications to contain bugs
  - Allow sharing of resources among applications

Why is this important?





And many more

- Operating system is responsible for
  - Abstracting the hardware details for convenience and portability
  - Multiplex the hardware among multiple applications
  - Isolate applications to contain bugs
  - Allow sharing of resources among applications

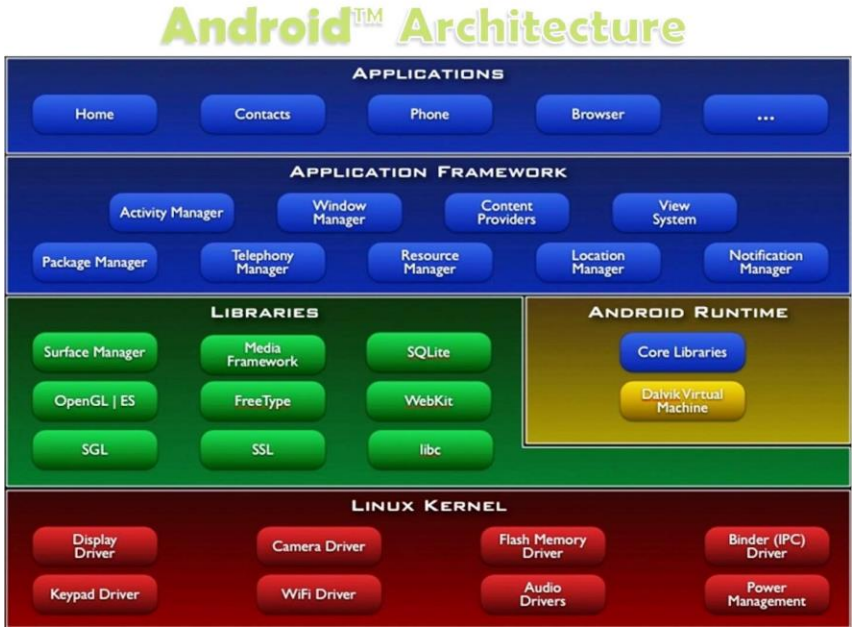
Example: USB device insertion

- Consider what happens when you plug-in a USB device to your laptop
  - USB controller informs its driver
    - The driver is part of the kernel
  - The driver asks the device to identify itself
    - Device sends back an id
  - Driver uses id to match a driver to the new device

A typical USB connector, called an "A" connection

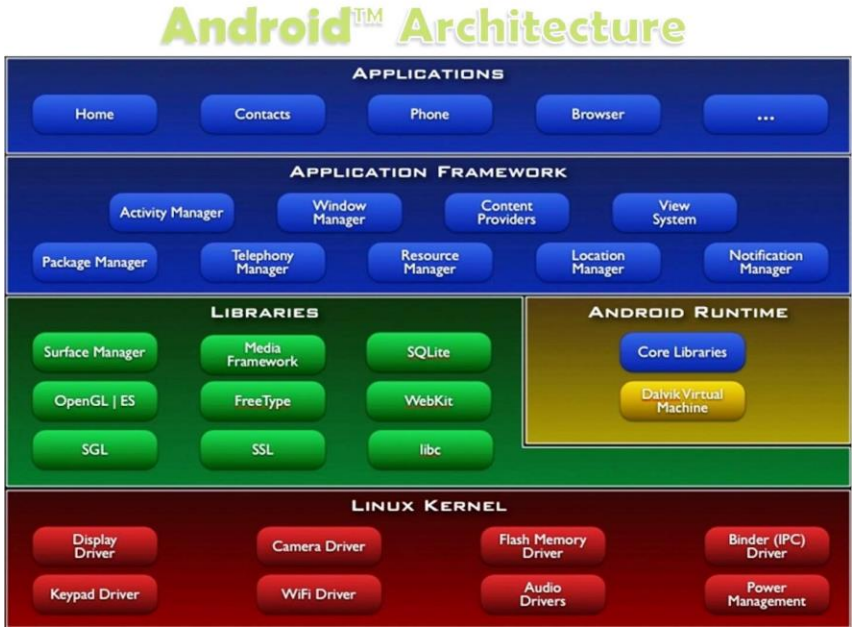
Inside a USB cable: There are two wires for power -- +5 volts (red) and ground (brown) -- and a twisted pair (yellow and blue) of wires to carry the data. The cable is also shielded.

Layers of a modern computing system



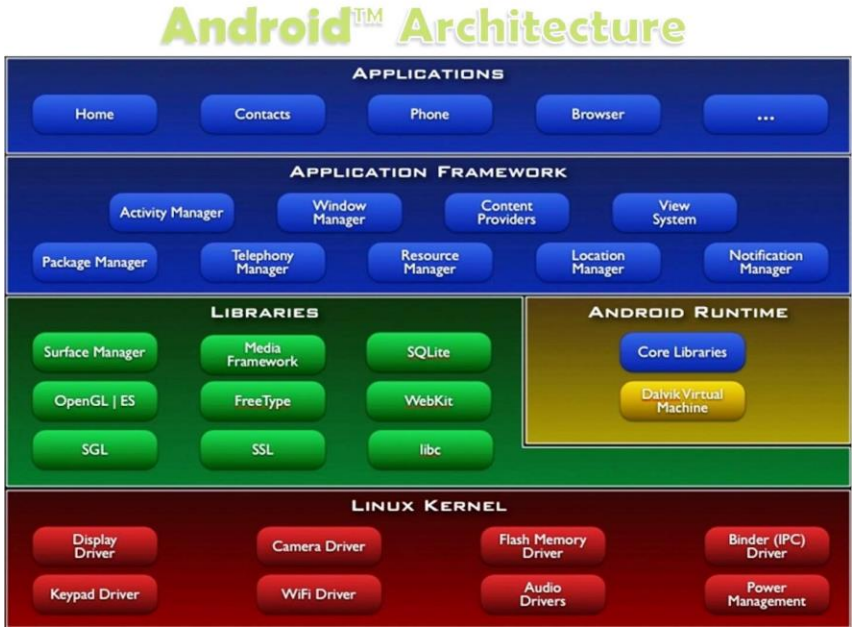
- Application – the user program
- Application framework
  - Helpful libraries for providing modularity and reuse
- System Libraries – the core services of OS are encapsulated by these libraries, e.g. libc
- The operating system kernel

Layers of a modern computing system



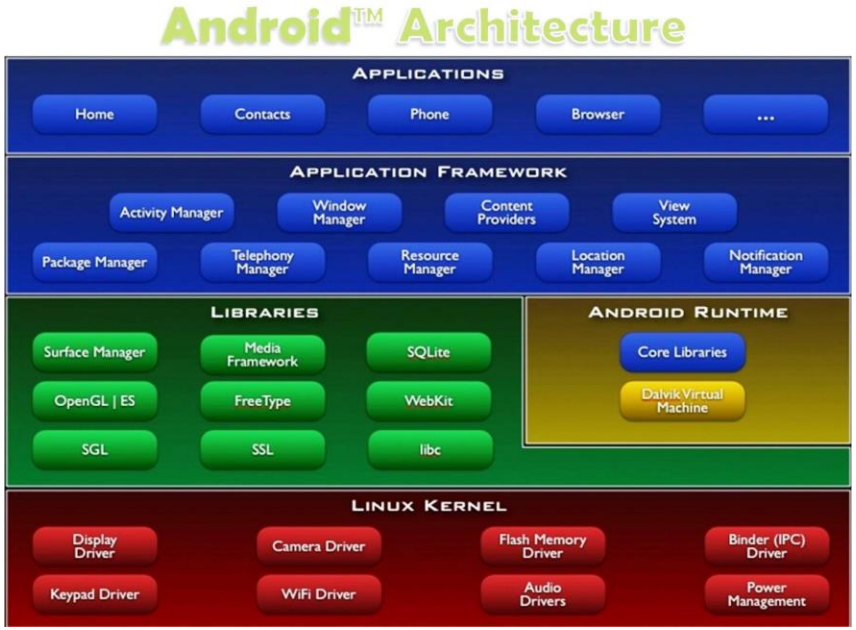
- Application – the user program
- Application framework
  - Helpful libraries for providing modularity and reuse
- System Libraries – the core services of OS are encapsulated by these libraries, e.g. libc
- The operating system kernel

Layers of a modern computing system



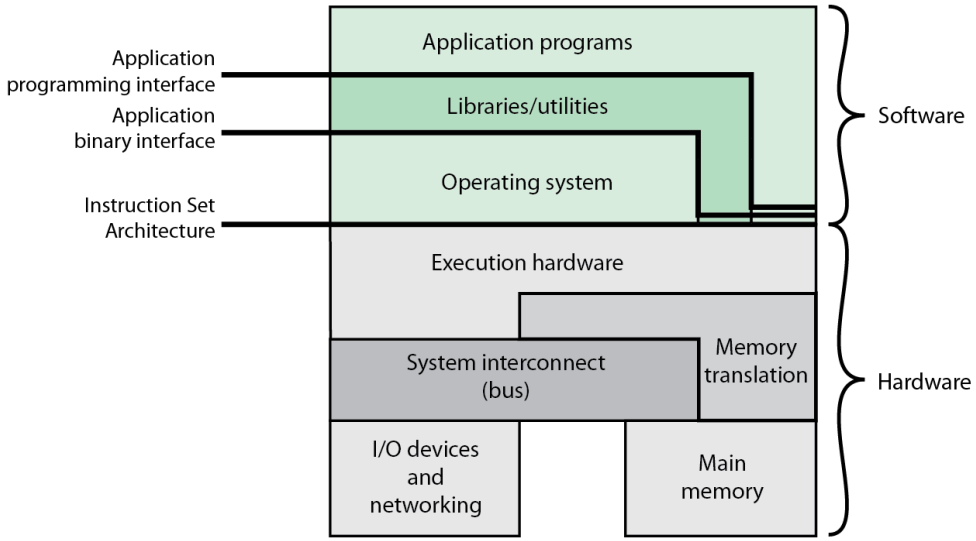
- Application – the user program
- Application framework
  - Helpful libraries for providing modularity and reuse
- System Libraries – the core services of OS are encapsulated by these libraries, e.g. libc
- The operating system kernel

Layers of a modern computing system



- Application – the user program
- Application framework
  - Helpful libraries for providing modularity and reuse
- System Libraries – the core services of OS are encapsulated by these libraries, e.g. libc
- The operating system kernel

Layers of a modern computing system



Another view of the layers

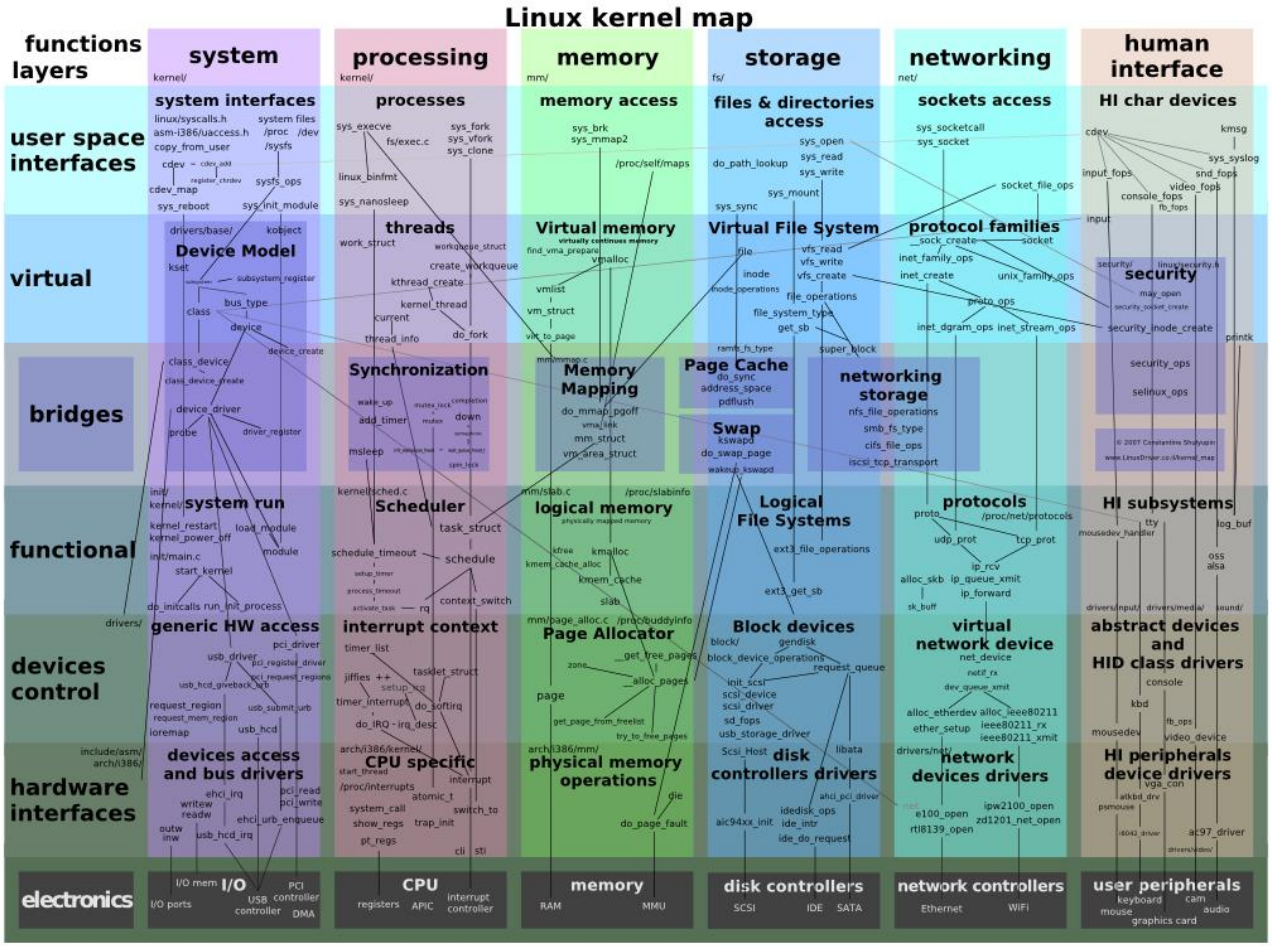
- Application – the user program.
- Application framework
  - Helpful libraries for providing modularity and reuse

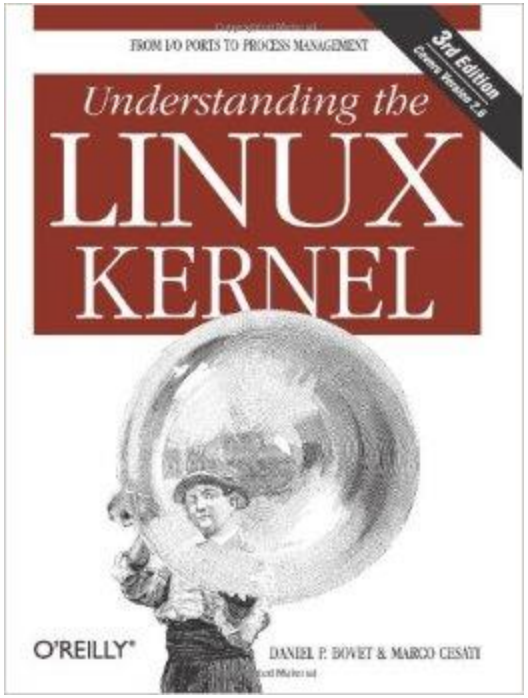
System Libraries – the core services of OS are encapsulated by these libraries, e.g. libc

- **The operating system kernel**

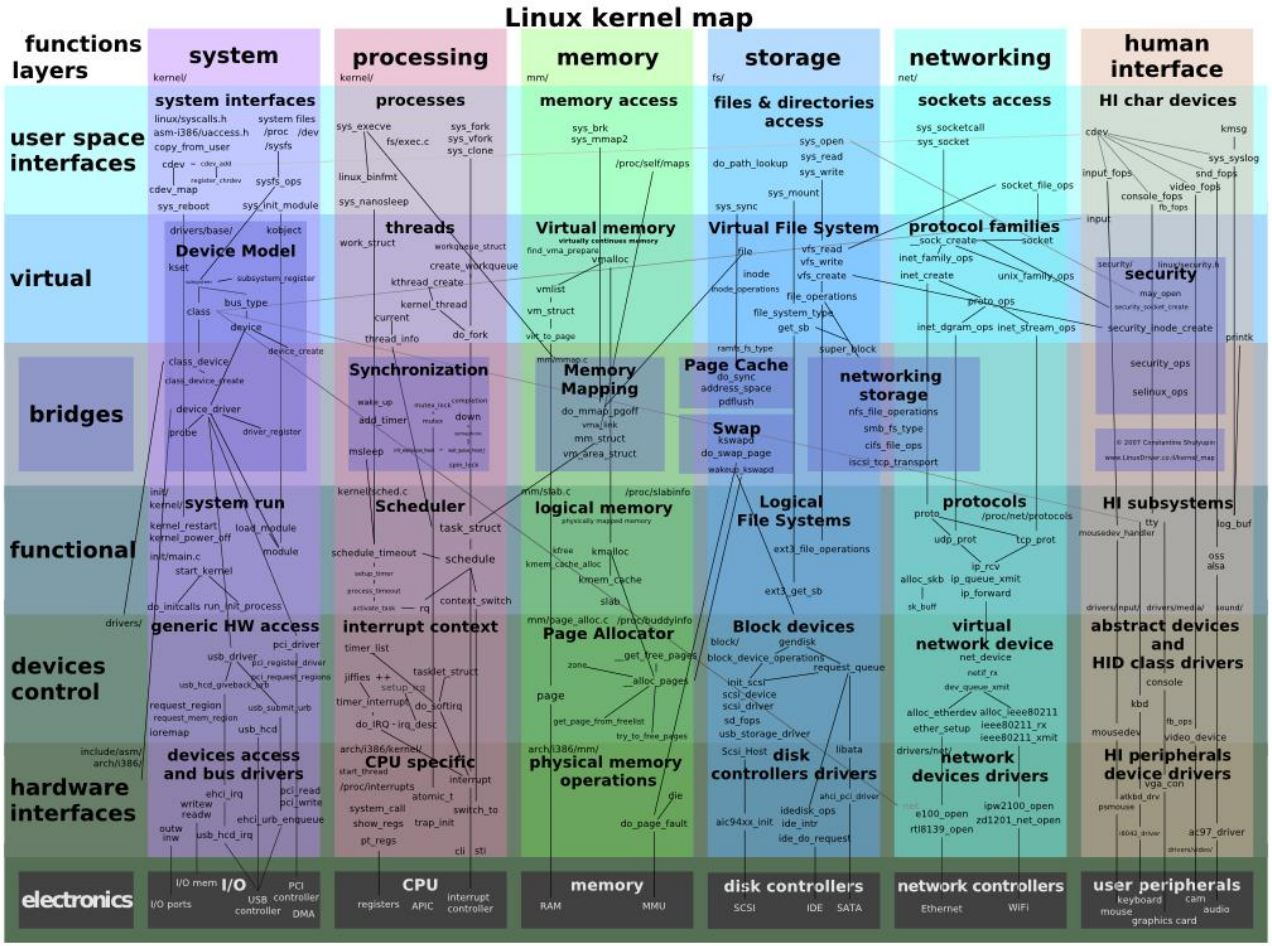
In this course we will primarily focus on The kernel and the system libraries.

The OS Kernel





The OS Kernel



- Process management
- Memory management
- File-system management
- Security
- Communication and networking
- Time Synchronization
- Many others: users, IPC, network, time, terminals

How do we interact with the kernel?

- Applications only see them via system calls (system calls are the API of the kernel)
- Examples, from UNIX / Linux:

```
pid_t pid = getpid();
printf("mypid is %d\n", pid);
```

How do we interact with the kernel?

- Applications only see them via system calls (system calls are the API of the kernel)
- Examples, from UNIX / Linux:

```
pid_t pid = getpid();
printf("mypid is %d\n", pid);
```

```
brk(0x18c9000)                  = 0x18c9000
clone(child_stack=0,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SE
TTID|SIGCHLD, child_tidptr=0x7fbf4bda1a10) =
3710
getpid()                        = 3709
fstat(1, {st_mode=S_IFCHR|0620,
st_rdev=makedev(136, 6), ...}) = 0
write(1, "mypid is 3709\n", 14mypid is 3709
)       = 14
exit_group(0)
```

Strace output (system calls in bold)

Why is OS design challenging

- The environment is unforgiving: weird h/w, hard to debug
- It must be efficient (thus low-level?)
  - but abstract/portable (thus high-level?)
- Powerful (thus many features?)
  - but simple (thus a few composable building blocks?)
- Features interact: `fd = open(); ...; fork()`
- Behaviors interact: CPU priority vs memory allocator.
- Open problems: security, multi-core

Why is OS design challenging

- The environment is unforgiving: weird h/w, hard to debug
- It must be efficient (thus low-level?)
  - but abstract/portable (thus high-level?)
- Powerful (thus many features?)
  - but simple (thus a few composable building blocks?)
- Features interact: `fd = open(); ...; fork()`
- Behaviors interact: CPU priority vs memory allocator.
- Open problems: security, multi-core

Why is OS design challenging

- The environment is unforgiving: weird h/w, hard to debug
- It must be efficient (thus low-level?)
  - but abstract/portable (thus high-level?)
- Powerful (thus many features?)
  - but simple (thus a few composable building blocks?)
- Features interact: `fd = open(); ...; fork()`
- Behaviors interact: CPU priority vs memory allocator.
- Open problems: security, multi-core

Why is OS design challenging

- The environment is unforgiving: weird h/w, hard to debug
- It must be efficient (thus low-level?)
  - but abstract/portable (thus high-level?)
- Powerful (thus many features?)
  - but simple (thus a few composable building blocks?)
- Features interact: `fd = open(); ...; fork()`
- Behaviors interact: CPU priority vs memory allocator.
- Open problems: security, multi-core

Why is OS design challenging

- The environment is unforgiving: weird h/w, hard to debug
- It must be efficient (thus low-level?)
  - but abstract/portable (thus high-level?)
- Powerful (thus many features?)
  - but simple (thus a few composable building blocks?)
- Features interact: `fd = open(); ...; fork()`
- Behaviors interact: CPU priority vs memory allocator.
- Open problems: security, multi-core

Why is OS design challenging

- The environment is unforgiving: weird h/w, hard to debug
- It must be efficient (thus low-level?)
  - but abstract/portable (thus high-level?)
- Powerful (thus many features?)
  - but simple (thus a few composable building blocks?)
- Features interact: `fd = open(); ...; fork()`
- Behaviors interact: CPU priority vs memory allocator.
- Open problems: security, multi-core

Before Next Class

- If you are not familiar with how to use the command line, do this tutorial:
  - https://www.codecademy.com/learn/learn-the-command-line
- Go here and see how far you can get:
  - https://overthewire.org/wargames/bandit/
- The next class will be a review of the C programming language, an overview of Git, and an overview of Linux
- Install the VM