CSE 306 Operating Systems

Dongyoon Lee

About me

- Dongyoon Lee
- Assistant Professor in CS @ SBU
- Ph.D in CSE @ U of Michigan, Ann Arbor (2013)
- Email: dongyoon@cs.stonybrook.edu
- Office: 339 New Computer Science (NCS)
- Homepage: https://www3.cs.stonybrook.edu/~dongyoon/

Research interests: OS + Compiler + HW for

- Reliability
  - Persistency bugs [OSDI'22][SOSP'21]
  - Concurrency bugs [CGO'18][EuroSys'17][ASPLOS'17][ASPLOS'16]
  - Transient fault (soft error) tolerance [MICRO'16][SC'16][LCTES'15]
- Security
  - Linux kernel permissions [USENIX Security'19]
  - Regular expression denial of service [S&P'21][FSE'19][ASE'19][ASE'19][SEC'18][FSE'18]
  - Memory safety [ASPLOS'19][MICRO'18]

Research interests: OS + Compiler + HW for

- Performance
    - Edge stream processing [ATC’19]
    - Distributed key value stores [SC’18]

About this course

- CSE 306: Operating Systems
- This semester: OS Basics + Linux Kernel Programming
- Goals
  - Understand core subsystems of modern operating systems in depth
  - Design, implement, and modify Linux kernel code and modules for these subsystems
  - Test, debug, and evaluate the performance of systems software in kernel or user space, using debugging, monitoring and tracing tools

What is the *Linux Kernel*?

- One of operating system kernel
    - e.g., Windows, FreeBSD, OSX, etc.
- What does an OS do for you?
    - **Abstract** the hardware for convenience and portability
    - **Multiplex** the hardware among multiple applications
    - **Isolate** applications to contain bugs
    - Allow **sharing** among applications

View: layered organization

- **User**: applications (e.g., vi and gcc)
- **Kernel**: file system, process, etc.
- **Hardware**: CPU, mem, disk, etc.

→ Interface between layers

View: core services

- Processes
- Memory
- File contents
- Directories and file names
- Security
- Many others: users, IPC, network, time, terminals, etc.

→ Abstraction for applications

Example: system calls

- Interface: applications talk to an OS via system calls
- Abstraction: process and file descriptor

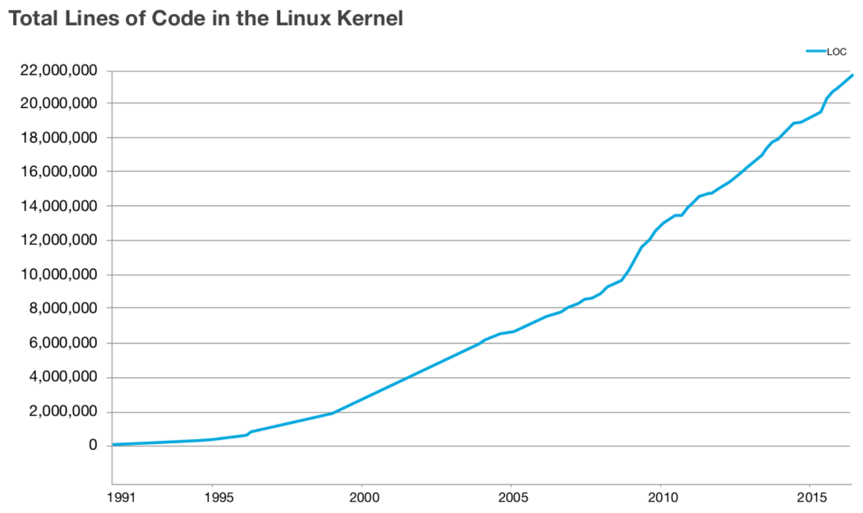
```
fd = open("out", 1);
write(fd, "hello\n", 6);
pid = fork();
```

Why is Linux kernel interesting?

- OS design deals with conflicting goals and trade-offs
    - Efficient yet portable
    - Powerful yet simple
    - Isolated yet interactable
    - General yet performant
- Open problems: multi-core and security
- How does a state-of-the-art OS deal with above issues?
    - **Hack the Linux kernel!**

Why is Linux kernel interesting?

- Extremely large software project
    - more than 25 million lines of code
    - 7,500 lines of code are added every day!



Why is Linux kernel interesting?

- Very fast development cycles
  - release about every 70 days
  - 13,000 patches / release
  - 273 ~~250~~ companies / release (or 1,600 developers / release)
- One of the most well-written/designed/maintained C code
- Ref: Linux Foundation Kernel Report 2017

Linux is eating the World

- 85.1% of smartphones and tables run Linux (Android)
    - iOS: 14.9%
- 98% of top 1 million web servers run Linux
- 99% of super computers run Linux
- SpaceX: From Earth to orbit with Linux and SpaceX
- Ref: Usage share of OS

It is good for your job search

- Contributions from unpaid developers had been in slow decline
  - 14.6% (2012) → 13.6% (2013) → 11.8% (2014) → 7.7% (2015)
- Why?
  - "There are many possible reasons for this decline, but, arguably, the most plausible of those is quite simple: **Kernel developers are in short supply, so anybody who demonstrates an ability to get code into the mainline tends not to have trouble finding job offers.**"
- Ref: Linux Foundation Kernel Report 2017

How to become a Linux kernel developer

- Interview of Sarah Sharp

Who should take this course?

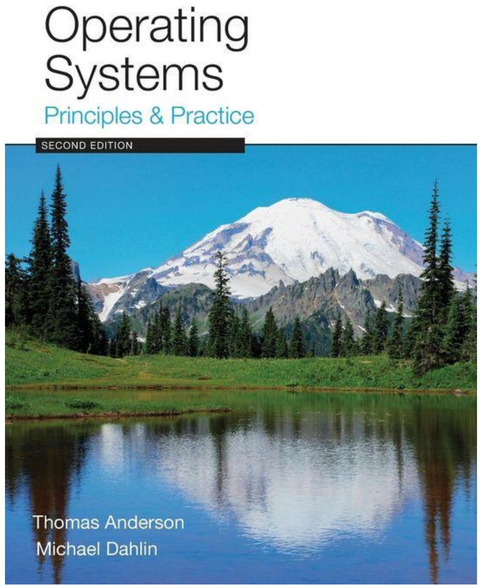
- Anyone wants to work on the above problems
- Anyone cares about what's going on under the hood
- Anyone has to build high-performance systems
- Anyone needs to diagnose bugs or security problems

Prerequisite

- You must have achieved a grade of C or higher in CSE 219 (or CSE 260) and CSE 320 (or ESE 380) to take this course
- C programming
- Linux command line

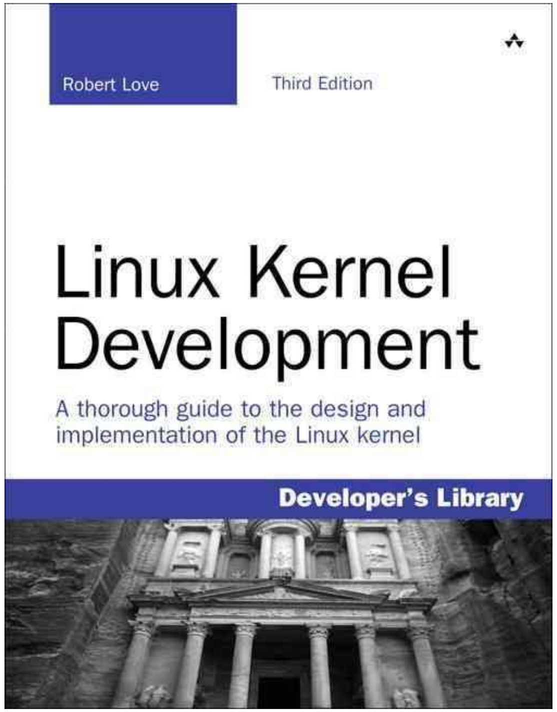
Recommended text book (OS Basics)

- Thomas Anderson and Michael Dahlini, Operating Systems: Principles and Practice, Recursive Books; 2nd edition (August 21, 2014)



Recommended text book (Linux Prog.)

- Robert Love, Linux Kernel Development, Addison-Wesley


Robert Love
Third Edition
Linux Kernel Development
A thorough guide to the design and implementation of the Linux kernel
Developer's Library

Other useful sources

- Operating Systems, Internals and Design Principles
- Understanding the Linux Kernel, O'Reilly Media
- Professional Linux Kernel Architecture, Wrox
- Linux Device Drivers, O'Reilly Media
- Understanding Linux Network Internals, O'Reilly Media
- Operating Systems: Three Easy Pieces
- Intel 64 and IA-32 Architectures Software Developer Manuals

Meetings

- When: Tues and Thurs 8:00-9:20 AM (EST)
- Where: (Old) Computer Science 2120

Communication

- Course website
    - Syllabus, schedule, etc.
- Brightspace
    - Lecture slides, assignments, grades, etc.
    - Submit your programming assignments here.
    - Lecture recording and office hour Zoom links.
- Piazza (Please sign up)
    - Announcements
    - Ask and answer questions (annonymous options available)

Office hours

- Dongyoon Lee
  - Tuesdays 9:30 AM - 10:30 AM (after class)
  - Thursdays 11:00 AM - 12:00 PM
  - Office: 339 New Computer Science (NCS)
- GTA: TBD
  - by appointment

Grading policy (subject to change)

- Projects (65%)
  - P1. system call (10%)
  - P2. kernel data structure (10%)
  - P3. shared memory (15%)
  - P4. cpu profiler (20%)
  - P5. file system (10%)
- Final exam (35%)
  - TBD

About projects

- All programming projects are individual assignments. You **may discuss** the assignment details, designs, debugging techniques, or anything else with anyone you like in general terms, but you **may not provide, receive, or take code to or from anyone**. The code you submit must be your own work and only your own work. Any evidence that source code has been copied, shared, or transmitted in any way will be regarded as evidence of academic dishonesty.
- Each student should prepare a Linux virtual machine (Details will follow).

Submission policies (subject to change)

- Late submissions: No late project work will be assigned a grade.
- Wrong submissions: (Trivial) submission errors (e.g., a missing file, a wrong patch) are subject to at least 25% penalty. Students should provide an evidence (e.g., last modified time stamp) that the original source codes have not been modified after the due date.

Academic Integrity

Each student must pursue his or her academic goals honestly and be personally accountable for all submitted work. Representing another person’s work as your own is always wrong. Faculty are required to report any suspected instances of academic dishonesty to the Academic Judiciary. For more comprehensive information on academic integrity, including categories of academic dishonesty, please refer to the Academic Integrity website.

Student Accessibility Support Center

If you have a physical, psychological, medical or learning disability that may impact your course work, please contact Student Accessibility Support Center, ECC (Educational Communications Center) Building, room 128, (631) 632-6748. They will determine with you what accommodations, if any, are necessary and appropriate. All information and documentation is confidential.

Critical Incident Management

Stony Brook University expects students to respect the rights, privileges, and property of other people. Faculty are required to report to the Office of Judicial Affairs any disruptive behavior that interrupts their ability to teach, compromises the safety of the learning environment, or inhibits students' ability to learn.

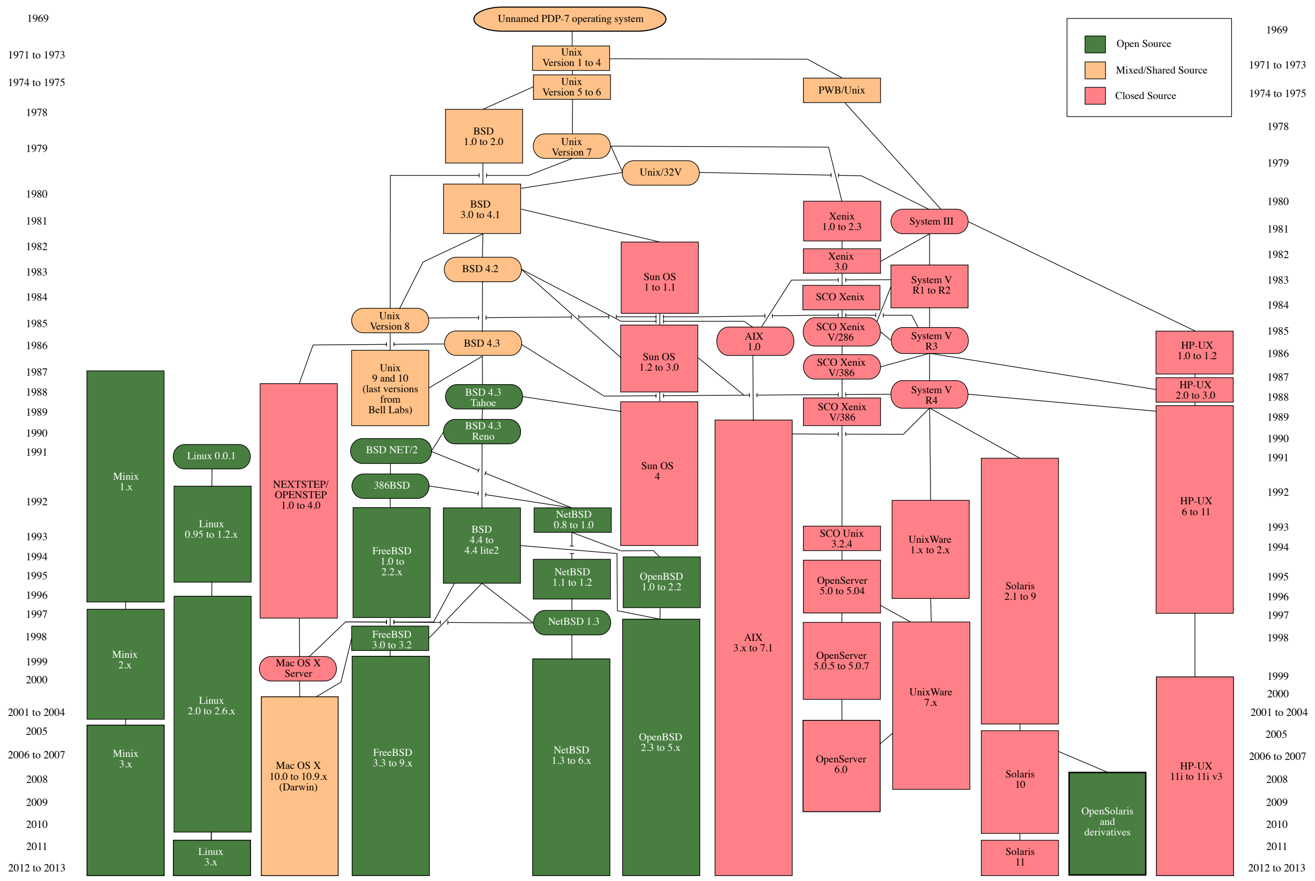
Acknowledgement

- This course reuses some of the material from:
  - VT's ECE 4984/5984 by Dr. Min (main source)
  - SBU's CSE 306 by Dr. Stark
  - SBU's CSE 506 by Drs. Zadok and Ferdman
  - GT's CS 3210
  - UW's CSE 451 and OSPP
  - MIT's 6.828

Today's agenda

- The history of Linux
- Linux open source model and community
- High level overview of the Linux kernel

History of UNIX (Wikipedia)



Beginning of Linux

```
From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups: comp.os.minix
Subject: What would you like to see most in minix?
Summary: small poll for my new operating system
Message-ID: <1991Aug25.205708.9541@klaava.Helsinki.FI>
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 Id like to know what features most
people would want. Any suggestions are welcome, but I won't promise I'll implement them 🙂

Linus (torvalds@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 :-(.
```

Linux History

- 1991: First apparition, author: Linus Torvalds
- 1992: GPL License, first Linux distributions
- 1994: v1.0 - Single CPU for i386, then ported to Alpha, Sparc, MIPS
- 1996: v2.0 - Symmetric multiprocessing (SMP) support
- 1999: v2.2 - Big Kernel Lock removed
- 2001: v2.4 - USB, RAID, Bluetooth, etc.
- 2003: v2.6 - Physical Address Expansion (PAE), new architectures, etc.
- 2011: v3.0 - Incremental release of v2.6
- 2015: v4.0 - Livepatch → today's latest version: http://www.kernel.org

Linux open source model

- Linux is licensed under **GPLv2**

> *You may copy, distribute and modify the software as long as you track changes/dates in source files. Any modifications to or software including (via compiler) GPL-licensed code must also be made available under the GPL along with build & install instructions.*

- Source code is freely available at https://www.kernel.org/
- Ref: td;lrLegal, GPLv2

Benefit of open source model

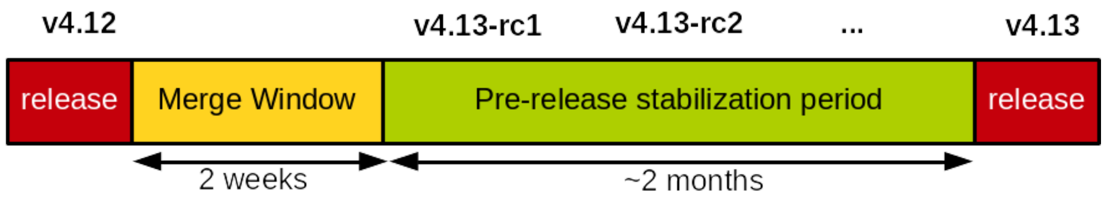
> *Given enough eyeballs, all bugs are shallow*

> *Given a large enough beta-test and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone.*

- **Linus's Law**
  - The Cathedral & the Bazaar by Eric S. Raymond
- Security, stability, quality, speed of innovation, education, research, etc

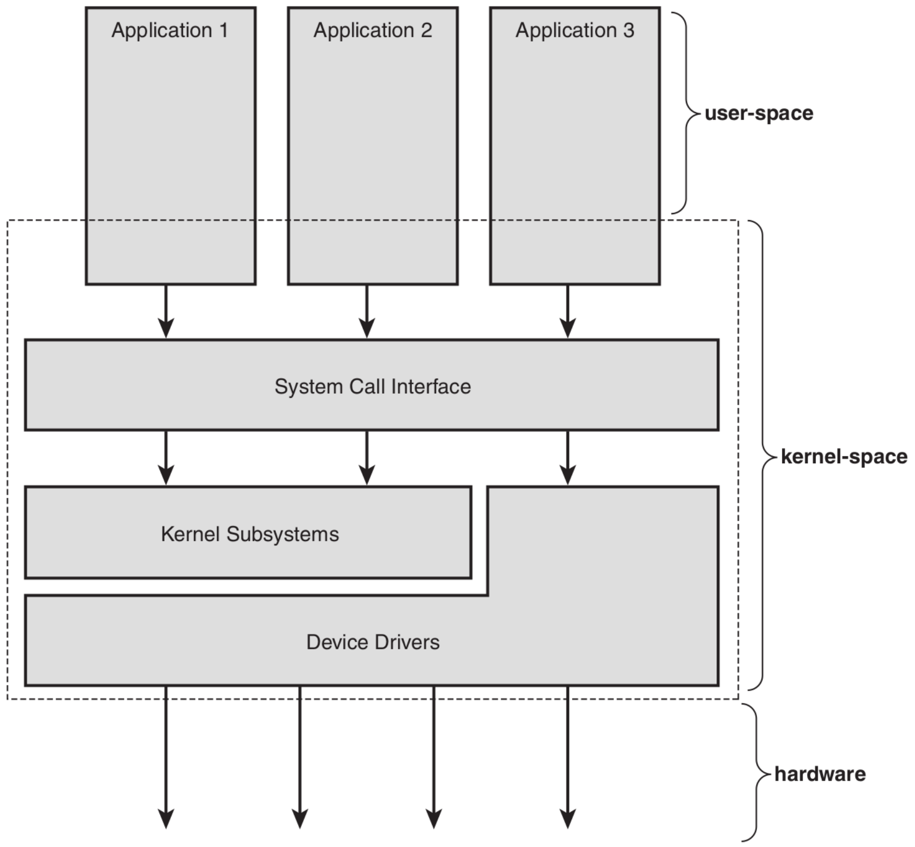
Kernel release cycle

- (major).(minor).(stable) → E.g., 5.8.3



- Prepatch or "RC" kernel release → for testing before the mainline release
- Mainline release → maintained by Linus with all new features
- Stable release → additional bug fixes after the mainline kernel release
- Long term support (LTS) for a subset of releases → e.g., 4.19, 4.14, 4.9

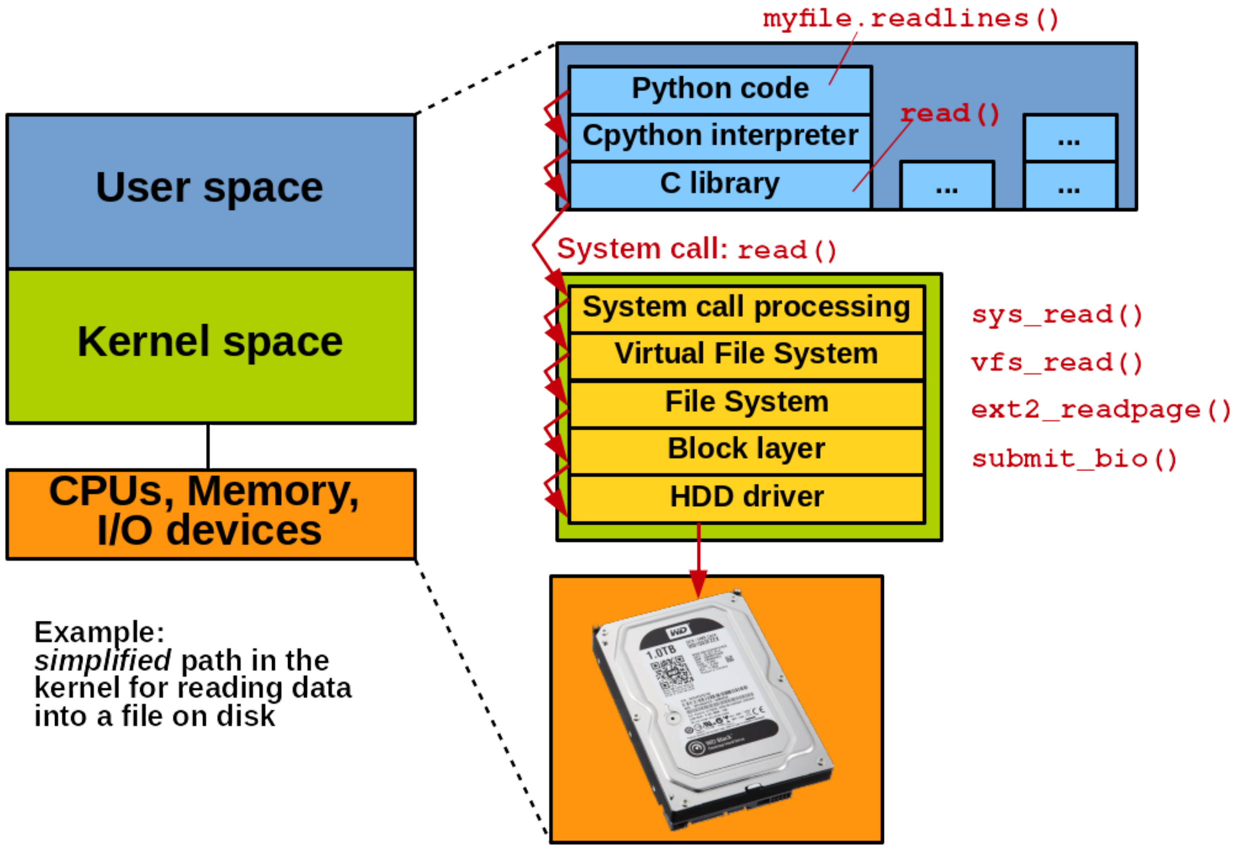
Overview of operating systems



User space vs. kernel space

- A CPU is executing in either of **user space** or in **kernel space**
- Only the kernel is allowed to perform **privileged operations** such as controlling CPU and IO devices
  - E.g., protection ring in x86 architecture
  - ring 3: user-space application
  - ring 0: operating system kernel
- An user-space application talks to the kernel space through **system call** interface
  - E.g., `open()`, `read()`, `write()`, `close()`

User space vs. kernel space



Example:
simplified path in the kernel for reading data into a file on disk

Linux is a *monolithic kernel*

- A traditional design: all of the OS runs in kernel, privileged mode
    - share the same address space
- Kernel interface ~= system call interface
- Good: easy for subsystems to cooperate
    - one cache shared by file system and virtual memory
- Bad: interactions are complex
    - leads to bugs, no isolation within kernel

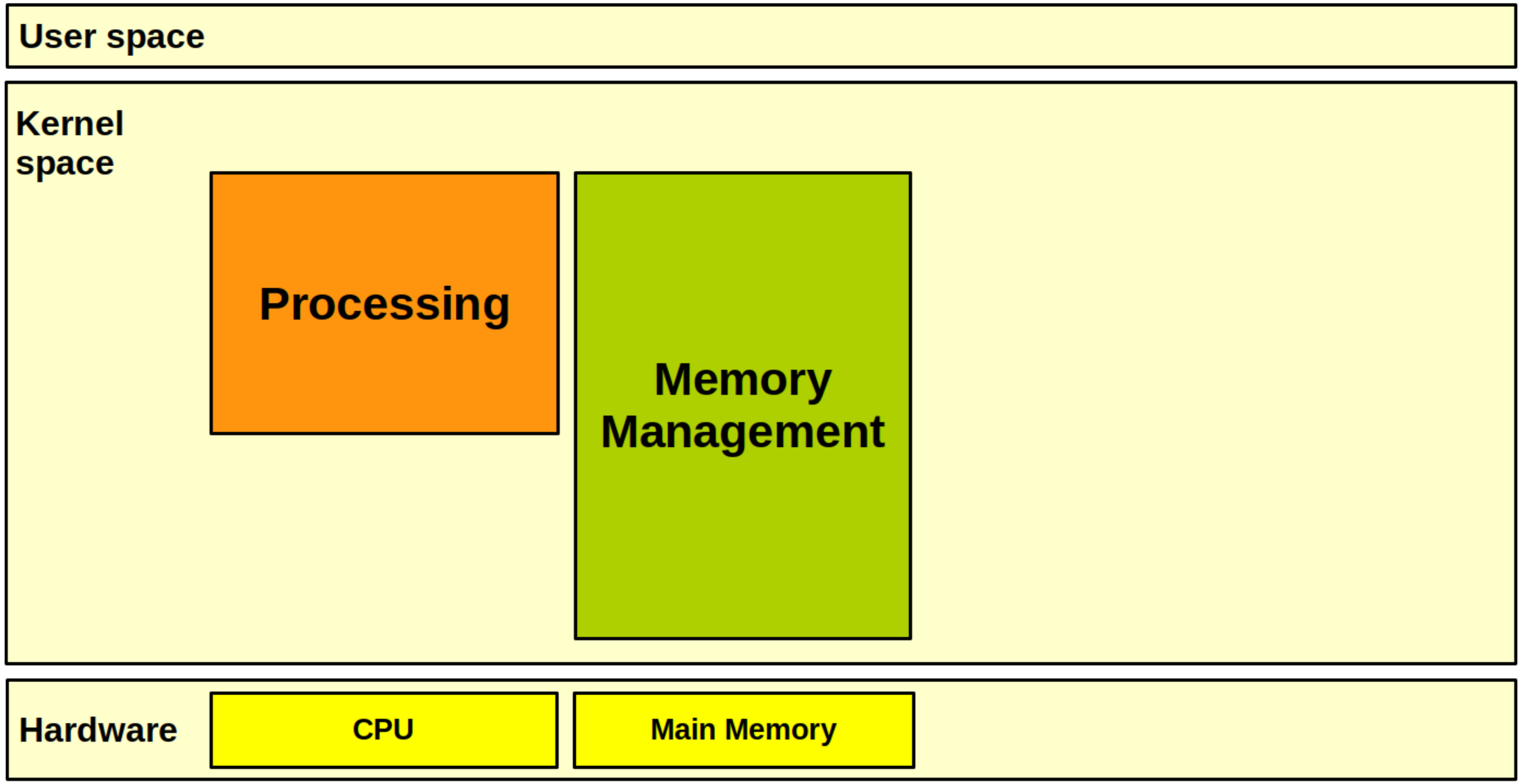
Alternative: *microkernel design*

- Many OS services run as ordinary user programs
    - e.g., file system in a file server
- Kernel implements minimal mechanism to run services in user space
    - IPC, virtual memory, threads
- Kernel interface != system call interface
    - applications talk to servers via IPCs
- Good: more isolation
- Bad: IPCs may be slow

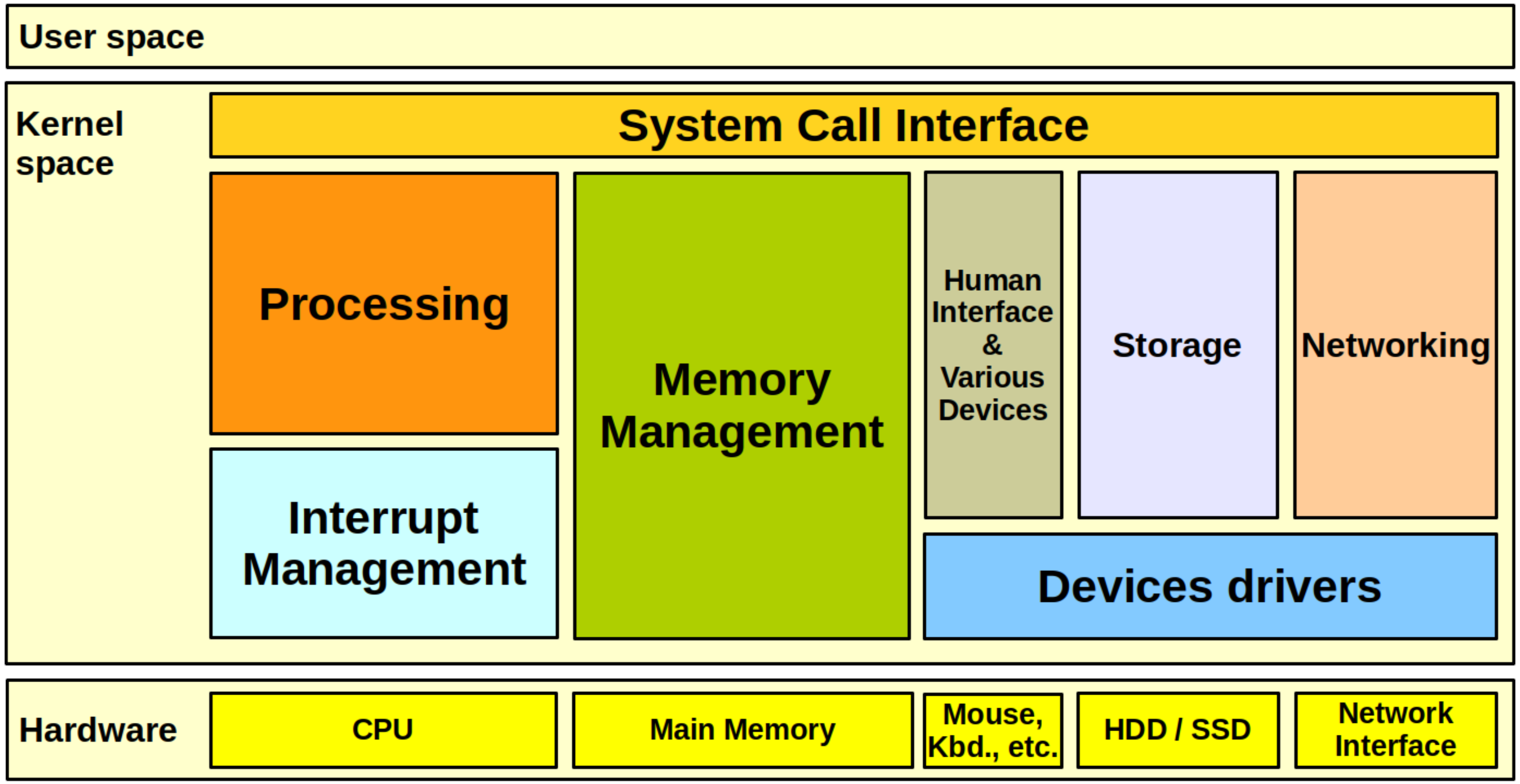
Debate

- Tanenbaum-Torvalds debate
- Most real-world kernels are mixed: Linux, OS X, Windows
    - e.g., X Window System

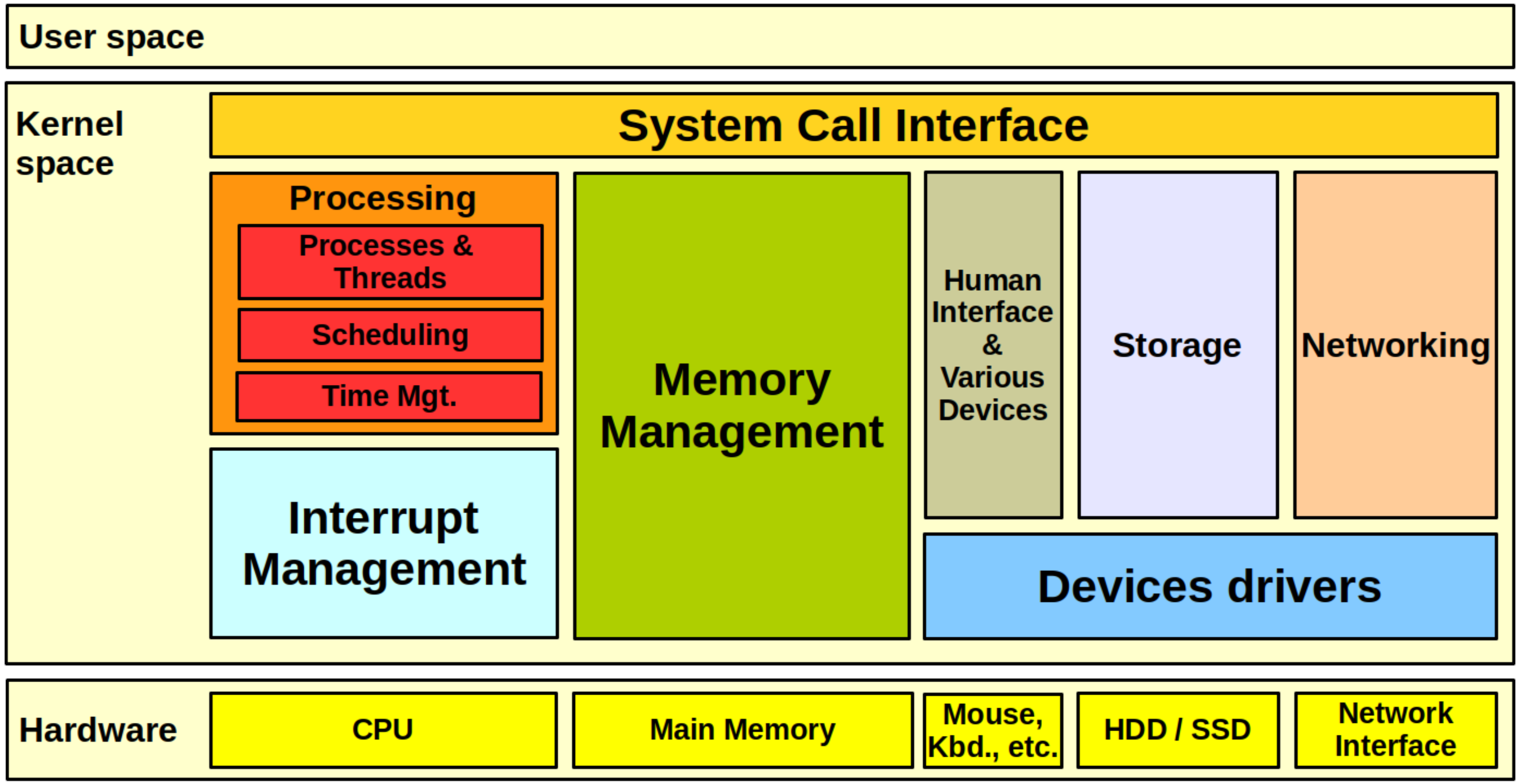
Kernel & course map



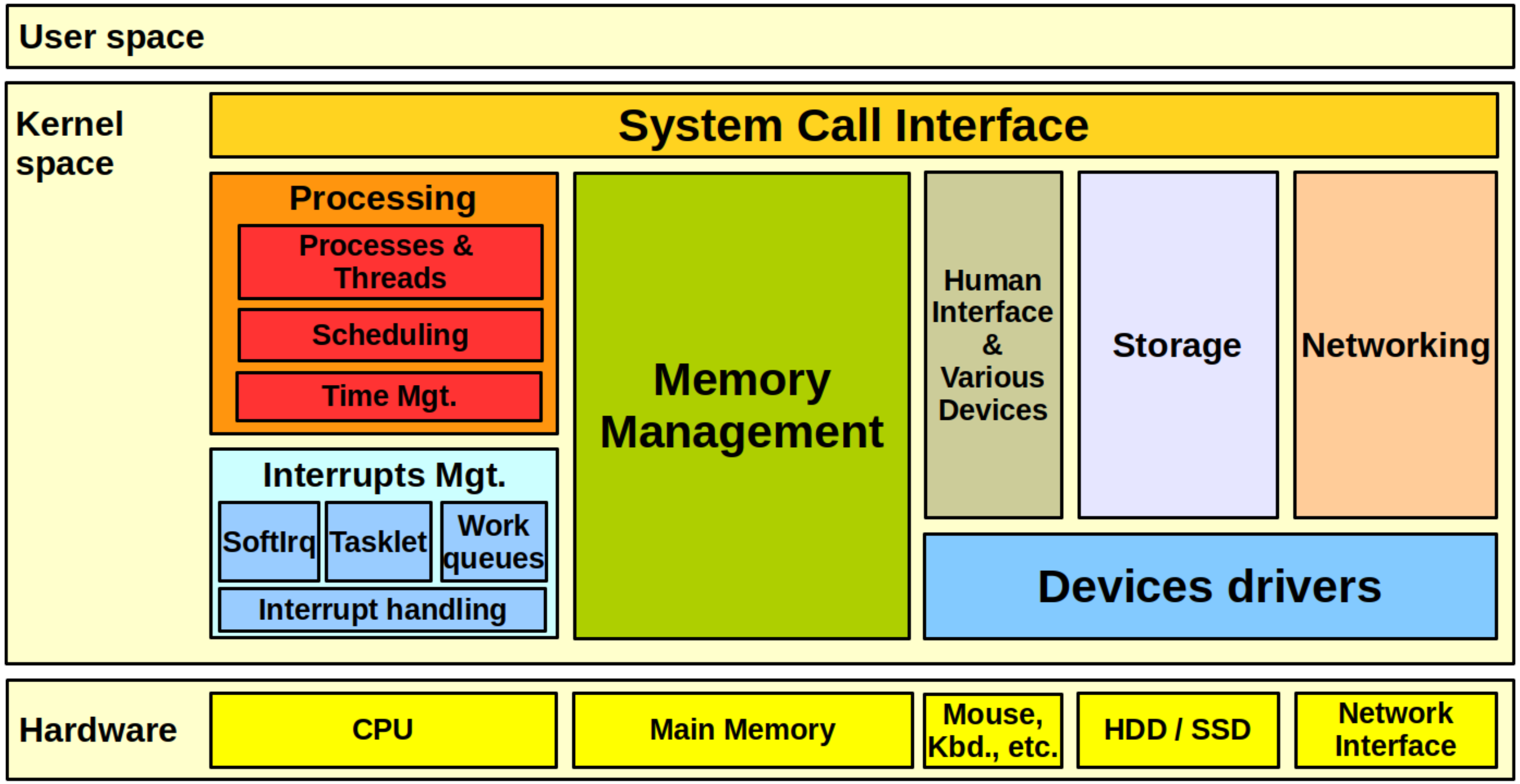
Kernel & course map



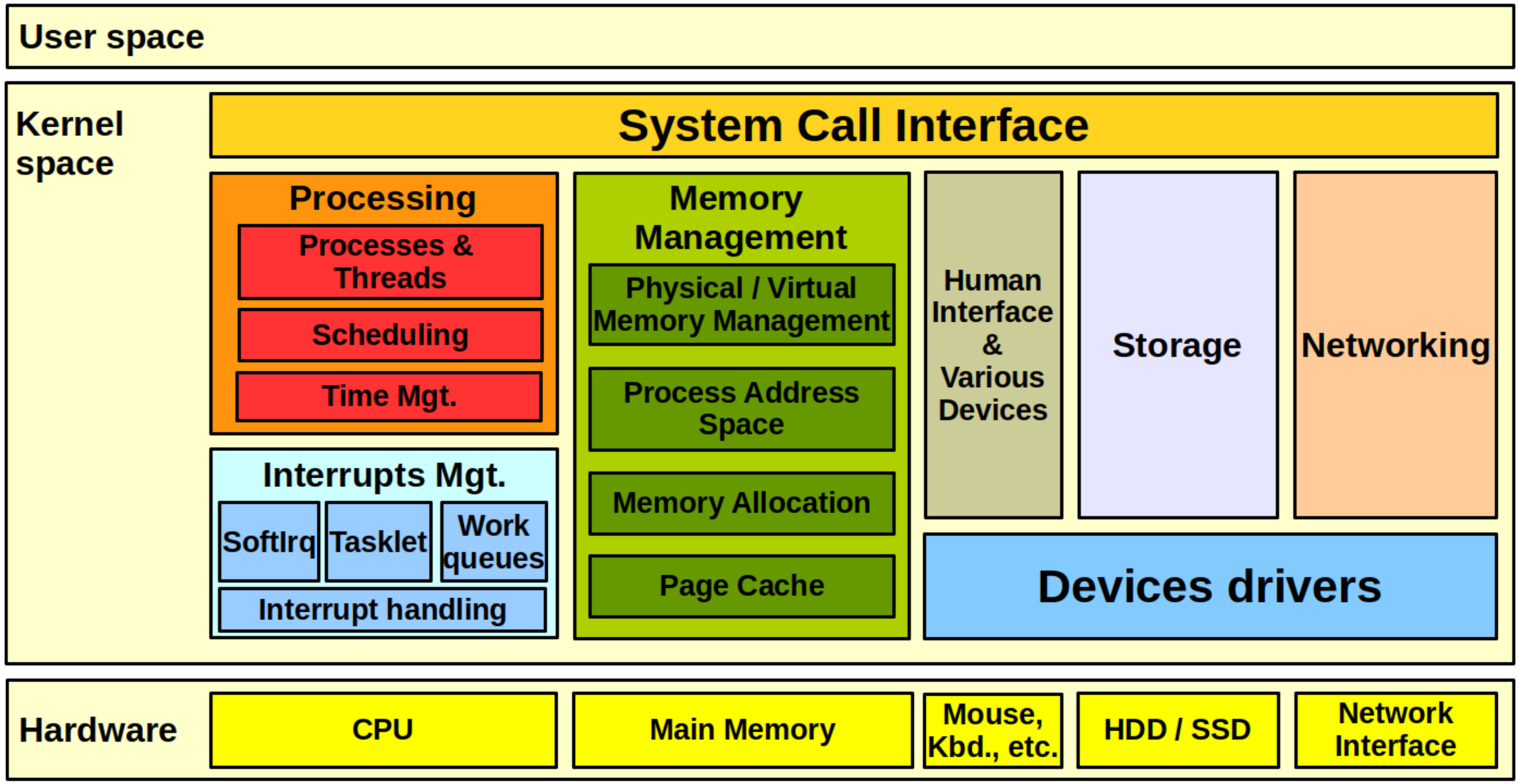
Kernel & course map



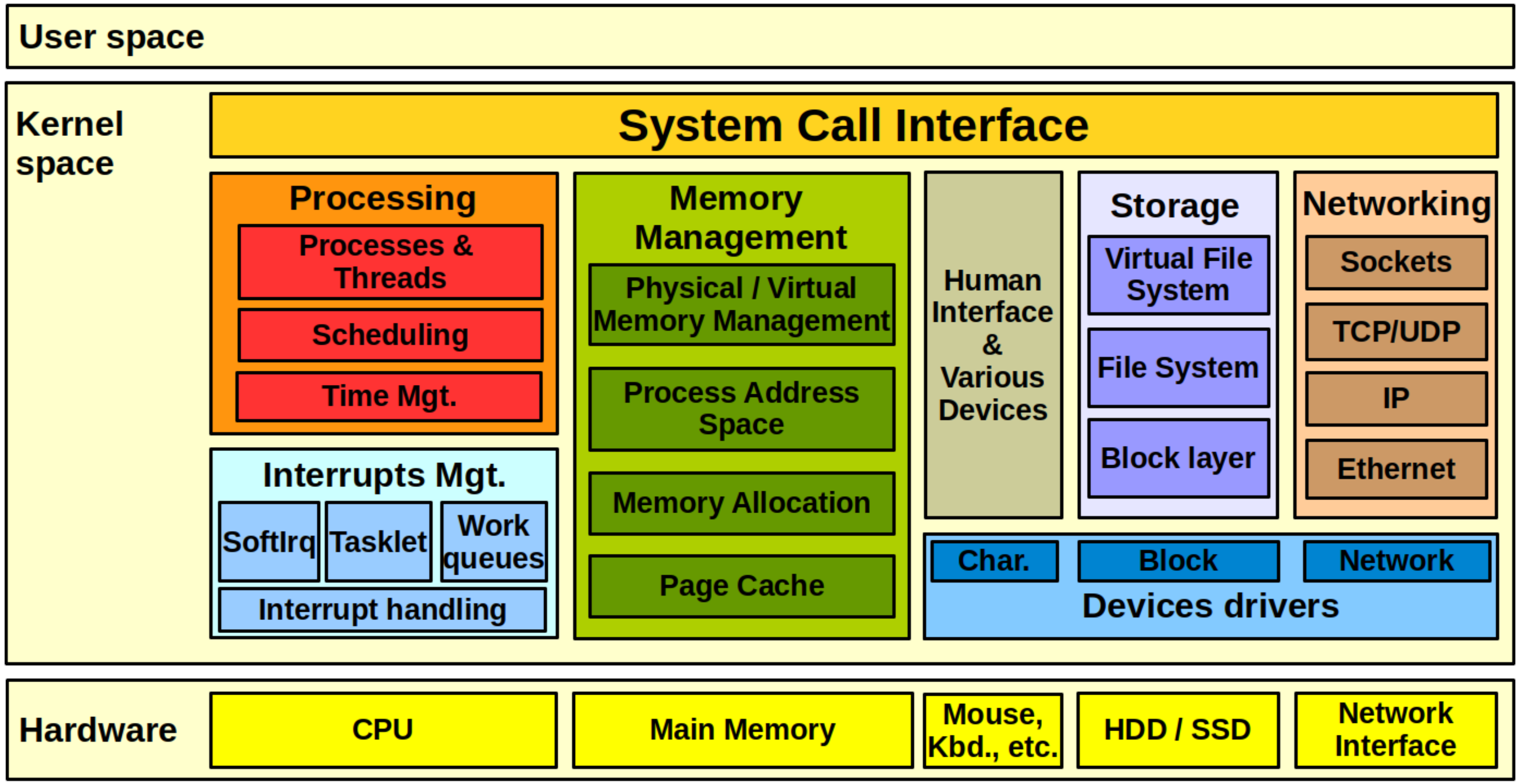
Kernel & course map



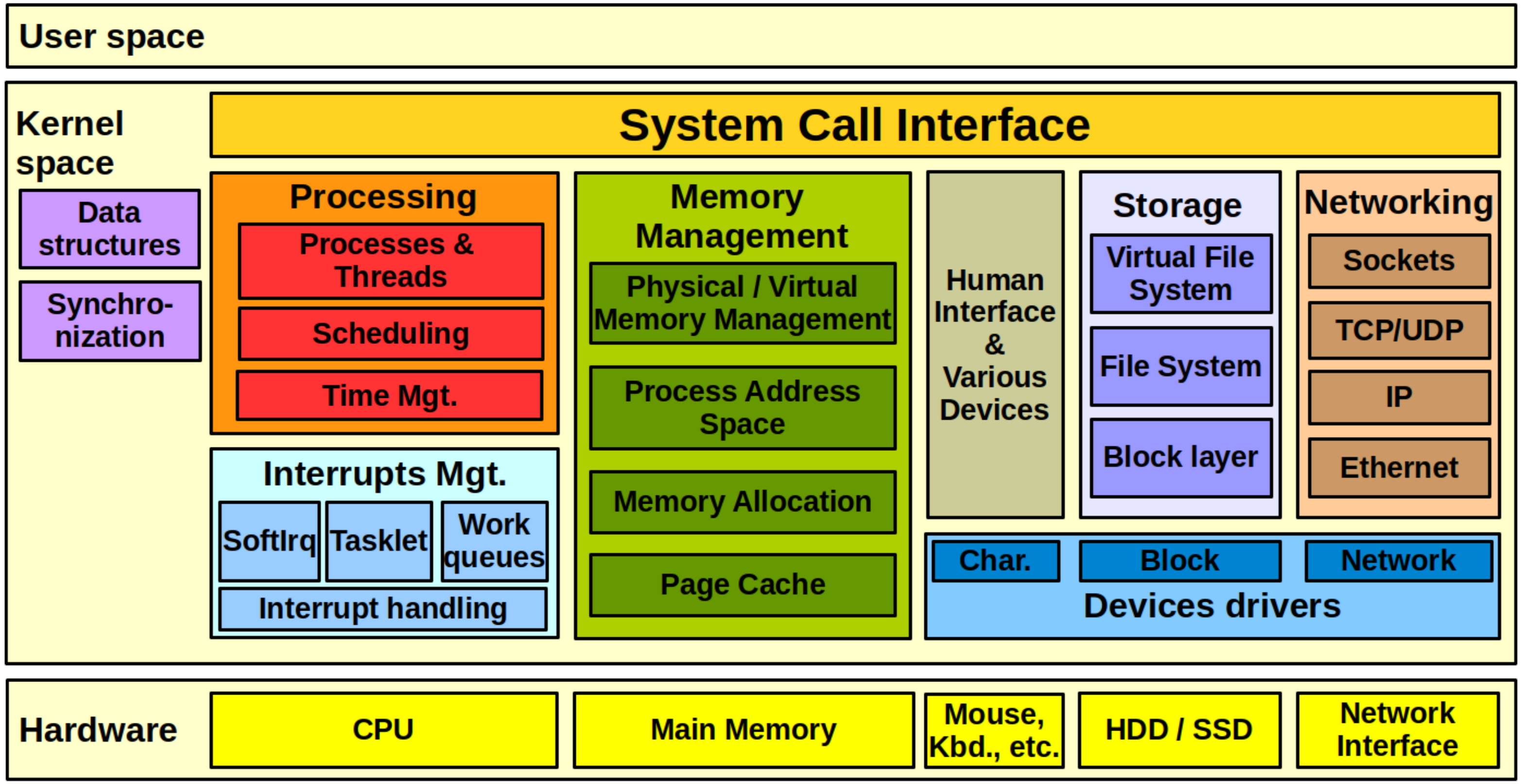
Kernel & course map



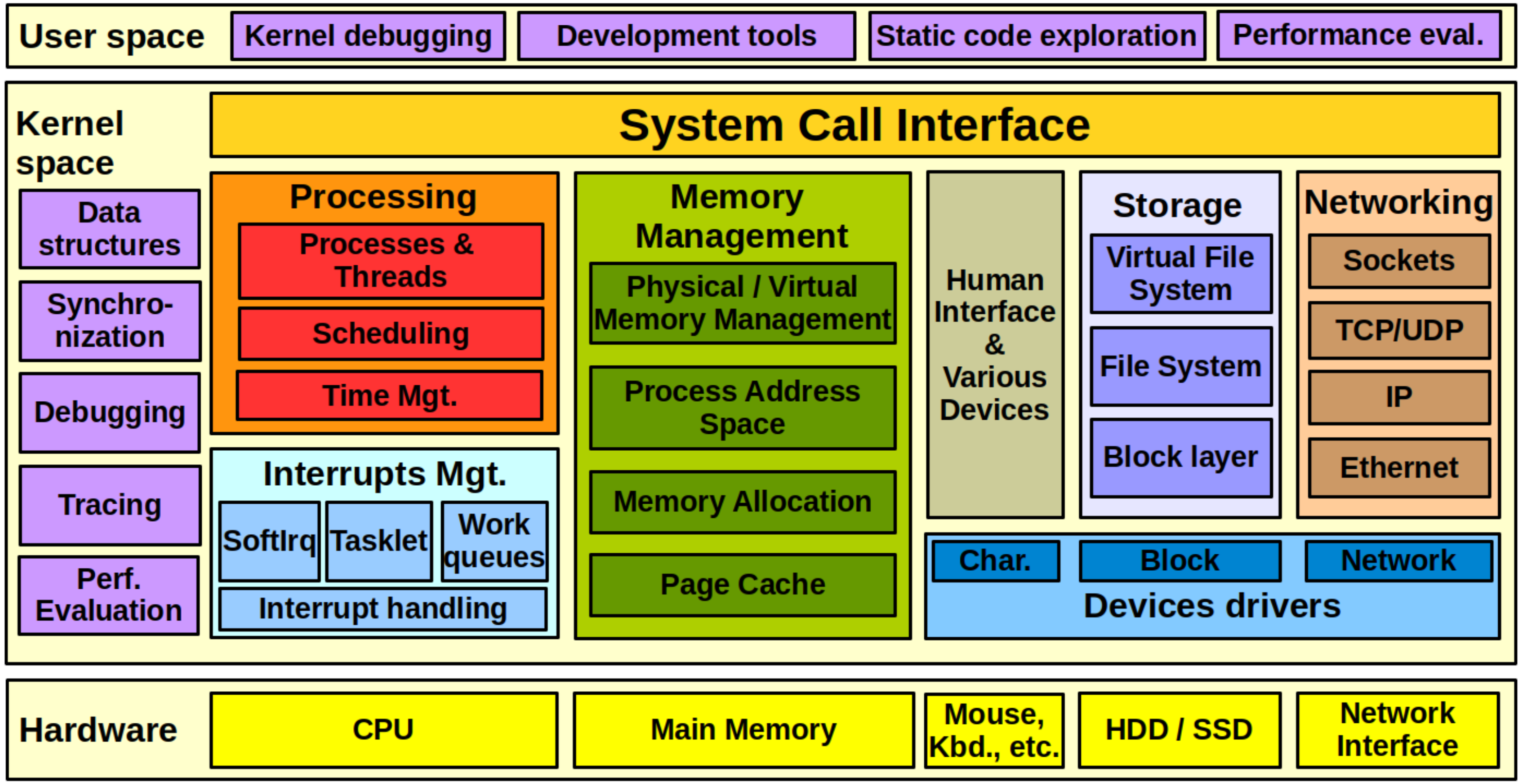
Kernel & course map



Kernel & course map



Kernel & course map



Kernel & course map

- Let’s check course schedule
    - Will be further updated

Set up course environment

- VirtualBox to run Linux VM
  - Recommended setting
    - disk >= 64GB, RAM >= 4GB, # CPU >= 2
  - Add port forwarding rule
    - protocol: TCP, host IP: 127.0.0.1
    - host port: 2222, guest port: 22 (ssh)
  - Use `scp' (over ssh) or `Shared folders` for file sharing between Linux VM and your host

Set up course environment

- Ubuntu 22.04.1 LTS Server for Linux distribution
    - Recommended disk space: 64 GB or more
    - Set up root password and create your user account
    - After login as a root user, add your account to `sudoers`
- SSH client on your laptop
    - Check whether you can `ssh` from host
        - `ssh -p 2222 {username}@localhost`
- Linux kernel: v5.15 released at October 31, 2021

Next actions

- Finish to set up course environment
- If you are not familiar with Linux commands, learn followings:
  - `vim`, `ssh`, `scp`, `tmux`, `git`, and more
  - Check this awesome online lectures: The Missing Semester of Your CS Education
- Download the latest Linux kernel source inside your Linux VM

```
$ git clone https://github.com/torvalds/linux.git
```

Next lecture

- Building and exploring Linux kernel