Week 1

Introduction to OS

Max Kopinsky
January 7, 2025

Welcome to OS!

See office hours schedule in Discord. TBD

- Instructor: Max Kopinsky
- TAs:
  - Yiwei Zhang — yiwei.zhang3@mail.mcgill.ca
  - Mohaddeseh Yagoubpour — fatemeh.yaghoubpour@mail.mcgill.ca
  - Mansi Dhanania — mansi.dhanania@mail.mcgill.ca
  - Yuyan Lin — yuyan.lin@mail.mcgill.ca
  - Shruti Bibra — shruti.bibra@mail.mcgill.ca
  - Mohamad Danesh — mo.danesh@mail.mcgill.ca
  - Alejandro Salinas — alejandro.salinas@mail.mcgill.ca
  - Hedi Jaza — mohamed.jaza@mail.mcgill.ca
  - +2 more TBD

About Me

- FL at McGill since Fall 2024 (recent!)
  - Teaching Area: Systems
  - Interests: Systems, hardware, prog. lang.s, CSE
    - Compilers, type systems, category theory
    - Complexity of games
    - More…
- MSc in Computer Science
  - At McGill
- BSc in Mathematics & Computer Science
  - At UIUC (in the US)

TA Introductions

About You ☺

- Cross listed Course: **380 students**

- ~105 Electrical & Computer Engineering students
- ~275 Computer Science students

About the course

Operating Systems

Why should you care?

Why should you care?

What do these have in common?

What do these devices have in common?

They have an operating system (OS)

Every program you will ever write will run on an OS

Its performance and execution will depend on the OS

Develop systems-thinking

- OS is one of the oldest disciplines in CS
- With a lot of influence on other systems disciplines.

OS concepts are the foundation for:

- Distributed systems (e.g., blockchain), Cloud infrastructure, Internet of things, Database infrastructure

Overall goal of COMP-310/ECSE-427

- Learn **principles** of Operating Systems

Method

- Lectures
- Exercises
- Programming assignments
- Labs

Everything is recorded and posted on MyCourses ☺

Grading

- Project, consisting of 3 C programming assignments
  - OS Shell 10%
  - Scheduling 10%
  - Memory management 15%
  - Teamwork Survey 5%
- Non-Project assignment – 10%
- Take-home graded exercises – 10%
- Written final in exam session – 40%
  - If your final exam grade is higher than your take-home exercises grade, then the exercises will not count and the final counts for 50%.

Topic	Monday	Tuesday	Wednesday	Thursday	Friday
Week 1 Introduction	Jan 6	Jan 7 Course logistics + Intro to OS	Jan 8	Jan 9 Intro to OS	Jan 10 Workflow: Mimi, GitLab, Git
Week 2 Process Management	Jan 13	Jan 14 - add/drop deadline Intro to Process Management (1/2) Optional reading: OSTEP Ch. 3-7	Jan 15	Jan 16 Intro to Process Management (2/2)	Jan 17 C Review: C Basics
Week 3 Process Management	Jan 20	Jan 21 Synchronization Primitives (2/2) Optional reading: OSTEP Ch. 25-32	Jan 22 OS Shell Assignment Released	Jan 23 Synchronization Primitives (2/2)	Jan 24 C Tools: GDB Basics
Week 4 Process Management	Jan 27	Jan 28 Multi-process Structuring (1/2) Team registration deadline	Jan 29	Jan 30 Multi-process Structuring (2/2)	Jan 31 C Review: Advanced Debugging
Week 5 Process Management	Feb 3	Feb 4 Multithreading (1/2)	Feb 5	Feb 6 Multithreading (2/2) Exercises Sheet: Proc. Management	Feb 7 C Review: Ptrs & Memory Allocation I
Week 6 Memory Management	Feb 10	Feb 11 Virtual Memory (1/2) Optional reading: OSTEP Ch. 12-18	Feb 12 Scheduling Assignment Released	Feb 13 Virtual Memory (2/2)	Feb 14 C Review: C Files OS Shell Assignment Due
Week 7 Memory Management	Feb 17	Feb 18 Demand Paging (1/4) Optional reading: OSTEP Ch. 19-22	Feb 19	Feb 20 Demand Paging (2/4)	Feb 21 C Review: Working with pthreads I Graded Exercises Released
Week 8 Memory Management	Feb 24	Feb 25 Demand Paging (3/4)	Feb 26 Pthreads Programming Assignment Released	Feb 27 Demand Paging (4/4)	Feb 28 Work on Scheduling Assignment Scheduling Assignment Due
Week 9 Reading Week	Mar 3	Mar 4 No Class	Mar 5	Mar 6 No Class	Mar 7 No Lab
Week 10 File Systems	Mar 10 Graded Exercises Due	Mar 11 Intro to File Systems (1/2) Optional reading: OSTEP Ch. 36,37,39	Mar 12	Mar 13 Intro to File Systems (2/2)	Mar 14 C Review: Working with pthreads II Pthreads Programming Due
Week 11 File Systems	Mar 17	Mar 18 Basic FS Implementation (1/2) Optional reading: OSTEP Ch. 40,41,45	Mar 19 Memory Management Assignment Released	Mar 20 Basic FS Implementation (2/2)	Mar 21 C Review: Complex Structs
Week 12 File Systems	Mar 24	Mar 25 Advanced FS Implementation (1/2)	Mar 26	Mar 27 Advanced FS Implementation (2/2)	Mar 28 C Review: Ptrs & Memory Alloc II
Week 13 File Systems	Mar 31	Apr 1 Fault Tolerance in FS (1/2) Optional Reading: OSTEP Ch. 38,43	Apr 2	Apr 3 Fault Tolerance in FS (2/2) Exercises Sheet: File Systems	Apr 4 More C: Error-Handling Patterns Memory Management Assgn. Due
Week 14 Advanced Topics	Apr 7	Apr 8 Fault Tolerant Data Storage	Apr 9	Apr 10 Extra Topic: TBD	Apr 11 - Last day of class More C: Function Pointers

Tentative class schedule

link

Lectures

- Slides + Recordings
  - Best effort recordings
- Slides available at latest before Tuesday lecture on MyCourses
- Recordings available when McGill uploads them, usually 1-3hr after

Exercises

- Sprinkled through the lectures.
- 2 ungraded exercise sheets for practice.
- 1 graded take-home exercise sheet (details later).

Programming Assignments

- 4 assignments in C
- 3 of the “project” assignments build upon each other
  - OS Shell
  - Scheduling
  - Memory Management
- Project goal: create a simple OS simulation (run in user-mode)
- Separate assignment to practice multithreaded programming

Assignments Logistics

- Must run on Mimi server
  - Details on how to connect to and test code on the server will follow.

- Must be solved in C
  - If you need a C refresher, attend the C labs.

- For remote development, use the SOCS mimi servers

```
ssh <SOCSusername>@mimi.cs.mcgill.ca
```

- I do not recommend working locally

CS Accounts

If you don't have a CS account, make one today

- Most likely, you do not have an account if ECSE student
- Make an account here:

https://www.cs.mcgill.ca/docs/

Assignment Grading

- Based on Unit tests and code quality

Unit Tests

- We will provide you with all the unit tests and expected output for each assignment.

- Unit tests will be automatically executed, daily, on the Mimi server.

- You will get points for each correct expected output.
    - Formatting (spaces, capitalization, new lines, etc.) of your actual output **will not** be taken into account when comparing expected outputs.

Code quality

- TAs will check for **hardcoded results.**
  - And will remove all the points for hardcoded test outputs
- TAs can remove points for **coding style.**
  - New this term: experimenting with automatic style checking
  - Your code **must** follow a reasonable and consistent programming style
  - You should use a *source code formatter* such as **GNU Indent** or **AStyle**

Topic	Monday	Tuesday	Wednesday	Thursday	Friday
Week 1 Introduction	Jan 6	Jan 7 Course logistics + Intro to OS	Jan 8	Jan 9 Intro to OS	Jan 10 Workflow: Mimi, GitLab, Git
Week 2 Process Management	Jan 13	Jan 14 - add/drop deadline Intro to Process Management (1/2) Optional reading: OSTEP Ch. 3-7	Jan 15	Jan 16 Intro to Process Management (2/2)	Jan 17 C Review: C Basics
Week 3 Process Management	Jan 20	Jan 21 Synchronization Primitives (2/2) Optional reading: OSTEP Ch. 25-32	Jan 22 OS Shell Assignment Released	Jan 23 Synchronization Primitives (2/2)	Jan 24 C Tools: GDB Basics
Week 4 Process Management	Jan 27	Jan 28 Multi-process Structuring (1/2) Team registration deadline	Jan 29	Jan 30 Multi-process Structuring (2/2)	Jan 31 C Review: Advanced Debugging
Week 5 Process Management	Feb 3	Feb 4 Multithreading (1/2)	Feb 5	Feb 6 Multithreading (2/2) Exercises Sheet: Proc. Management	Feb 7 C Review: Ptrs & Memory Allocation I
Week 6 Memory Management	Feb 10	Feb 11 Virtual Memory (1/2) Optional reading: OSTEP Ch. 12-18	Feb 12 Scheduling Assignment Released	Feb 13 Virtual Memory (2/2)	Feb 14 C Review: C Files OS Shell Assignment Due
Week 7 Memory Management	Feb 17	Feb 18 Demand Paging (1/4) Optional reading: OSTEP Ch. 19-22	Feb 19	Feb 20 Demand Paging (2/4)	Feb 21 C Review: Working with pthreads I Graded Exercises Released
Week 8 Memory Management	Feb 24	Feb 25 Demand Paging (3/4)	Feb 26 Pthreads Programming Assignment Released	Feb 27 Demand Paging (4/4)	Feb 28 Work on Scheduling Assignment Scheduling Assignment Due
Week 9 Reading Week	Mar 3	Mar 4 No Class	Mar 5	Mar 6 No Class	Mar 7 No Lab
Week 10 File Systems	Mar 10 Graded Exercises Due	Mar 11 Intro to File Systems (1/2) Optional reading: OSTEP Ch. 36,37,39	Mar 12	Mar 13 Intro to File Systems (2/2)	Mar 14 C Review: Working with pthreads II Pthreads Programming Due
Week 11 File Systems	Mar 17	Mar 18 Basic FS Implementation (1/2) Optional reading: OSTEP Ch. 40,41,45	Mar 19 Memory Management Assignment Released	Mar 20 Basic FS Implementation (2/2)	Mar 21 C Review: Complex Structs
Week 12 File Systems	Mar 24	Mar 25 Advanced FS Implementation (1/2)	Mar 26	Mar 27 Advanced FS Implementation (2/2)	Mar 28 C Review: Ptrs & Memory Alloc II
Week 13 File Systems	Mar 31	Apr 1 Fault Tolerance in FS (1/2) Optional Reading: OSTEP Ch. 38,43	Apr 2	Apr 3 Fault Tolerance in FS (2/2) Exercises Sheet: File Systems	Apr 4 More C: Error-Handling Patterns Memory Management Assgn. Due
Week 14 Advanced Topics	Apr 7	Apr 8 Fault Tolerant Data Storage	Apr 9	Apr 10 Extra Topic: TBD	Apr 11 - Last day of class More C: Function Pointers

Use this time to
Get familiar with git, linux environment,
and grading infrastructure.

2+ weeks to solve each assignment; 4 late days to use however you wish
➔ No other extensions

Starter Code

- We provide starter code for assignments
- Starter code isn't perfect
  - But it provides basic functionality we are looking for in class

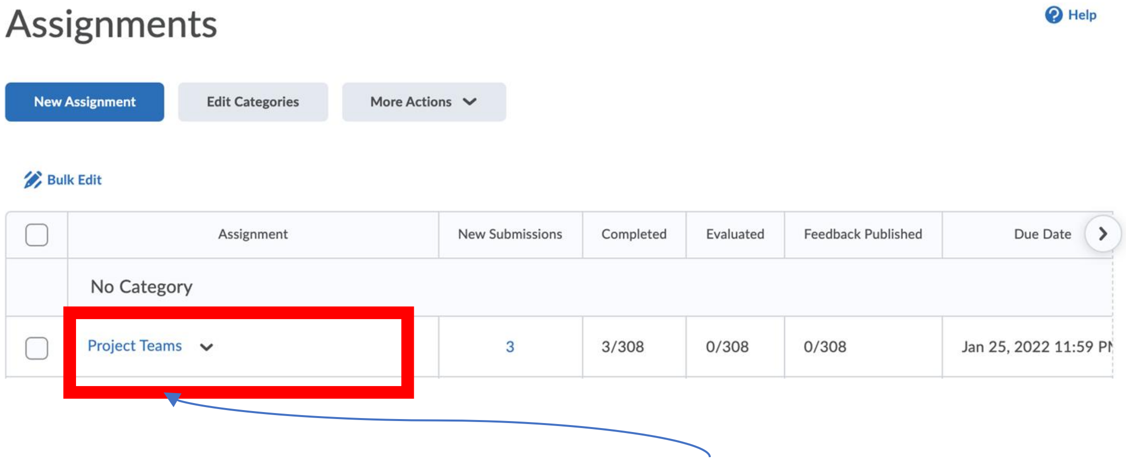
- Feel free to use our starter code
- or implement your project from scratch ☺

Teams for Assignments

- Assignments can be done individually
  - Workload and timeframe are designed to solve on your own.

- Or in **teams of 2**
  - If you decide to team-up,
  - you commit to your teammate for **all 4 programming assignments**
  - **both teammates get the same grade**
  - Do not team up “just because you can.” Last term, students who partnered with someone that they did not know performed noticeably worse than solo students.

How to form teams on MyCourses



- Upload a .txt file with the required info
- If you want to work alone, **you will need to register a team of 1**
- More info + assignment release on MyCourses coming soon.

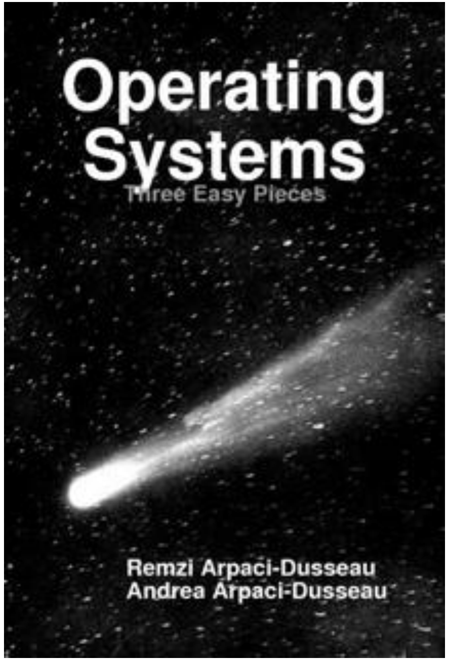
Friday
Jan 10
Workflow: Mimi, GitLab, Git
Jan 17
C Review: C Basics
Jan 24
C Tools: GDB Basics
Jan 31
C Review: Advanced Debugging
Feb 7
C Review: Ptrs & Memory Allocation I
Feb 14
C Review: C Files
OS Shell Assignment Due
Feb 21
C Review: Working with pthreads I
Graded Exercises Released
Feb 28
Work on Scheduling Assignment
Scheduling Assignment Due
Mar 7 No Lab
Mar 14
C Review: Working with pthreads II
Pthreads Programming Due
Mar 21
C Review: Complex Structs
Mar 28
C Review: Ptrs & Memory Alloc II
Apr 4
More C: Error-Handling Patterns
Memory Management Assgn. Due
Apr 11 - Last day of class
More C: Function Pointers

Labs

- In-person sessions on Fridays.
- Recordings + slides posted on MyCourses by following week.
- Goal: Refresh your C knowledge
  - Meant as a support for students who are a bit rusty in C.

- Labs are fully led by TAs.

Recommended Book



A *free* online book: http://pages.cs.wisc.edu/~remzi/OSTEP/

Prerequisites

- **ECSE-324** or
- **COMP-273**

Late policy

- You get 4 late days to use however you want
  - You will need to tell us that you intend to do so *before* the deadline
  - More info later
- No other extensions
- See syllabus for exceptional situations (e.g., medical emergencies)

Discord – Main place to communicate

- Announcements + Updates mainly through Discord
    - Important announcements mirrored on MyCourses

- Do not send course content questions by email

- Discord join link: https://discord.gg/ZEU9g4cHrh
    - Join ASAP

Email Policy

- Do not email course-related questions
  - Use Discord
- For issues with **grading**, email **grading TA**
  - not the instructor.
- For **personal and medical** issues, feel free to **send email to instructor**.

Questions?

Introducing the OS

- What does the OS do?
- Where does the OS live?
- OS interfaces
- OS control flow
- OS structure

What does an OS do?

A Bit of History

- Early days
  - Users program raw machine
- First “abstraction”
  - Libraries for scientific functions (sin, cos, …)
  - Libraries for doing I/O
- I/O libraries are the first pieces of an OS

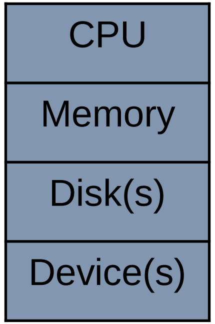
What does the OS do?

- Abstraction: makes hardware easier to use

What does the OS do?

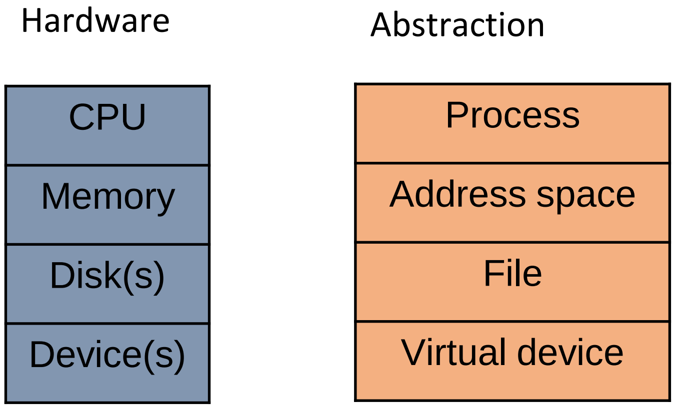
- Abstraction: makes hardware easier to use

Hardware



What does the OS do?

- Abstraction: makes hardware easier to use



What does the OS do?

- Abstraction: makes hardware easier to use

Hardware	Abstraction	OS component
CPU	Process	Process management
Memory	Address space	Memory management
Disk(s)	File	File systems
Device(s)	Virtual device	Device management

What does the OS do?

- Abstraction: makes hardware easier to use

Hardware	Abstraction	OS component
CPU	Process	Process management
Memory	Address space	Memory management
Disk(s)	File	File systems
Device(s)	Virtual device	Device management

In this class

Simple Example

Simple Example

- Write a Photoshop application

- Easier to deal with files containing photos
- Than to deal with data locations on disk

- OS provides **file abstraction**
- Finds data locations on disk given file name

Another Simple Example

- Write a web server

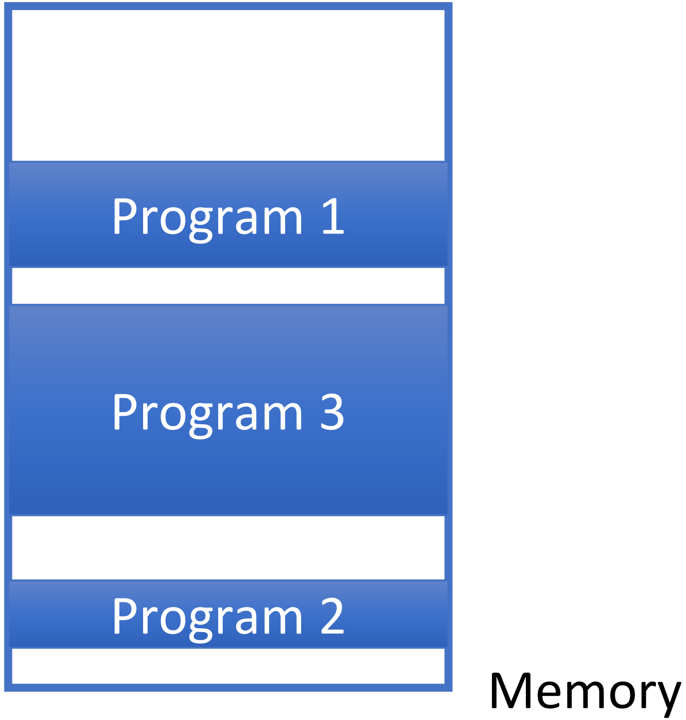
- Easier to deal with sending/receiving packets
- Than with NIC device registers

- OS provides **packet abstraction**
- Does the NIC device register manipulation

A Bit More History

- At some point, multiprogramming
- More than one program runs at the same time

Multiprogramming



Multiprogramming

- Need to protect programs from each other
- Need to protect OS from programs

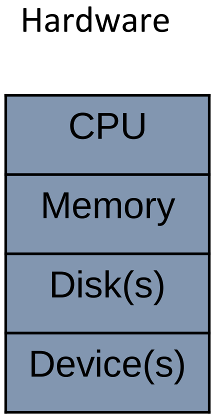
- Need to allocate/free memory

What does the OS do?

- Resource management: allocates hardware resources between programs

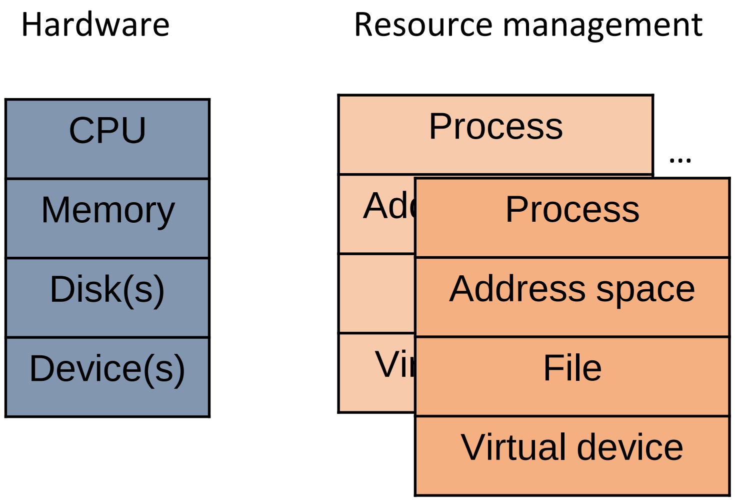
What does the OS do?

- Resource management: allocates hardware resources between programs



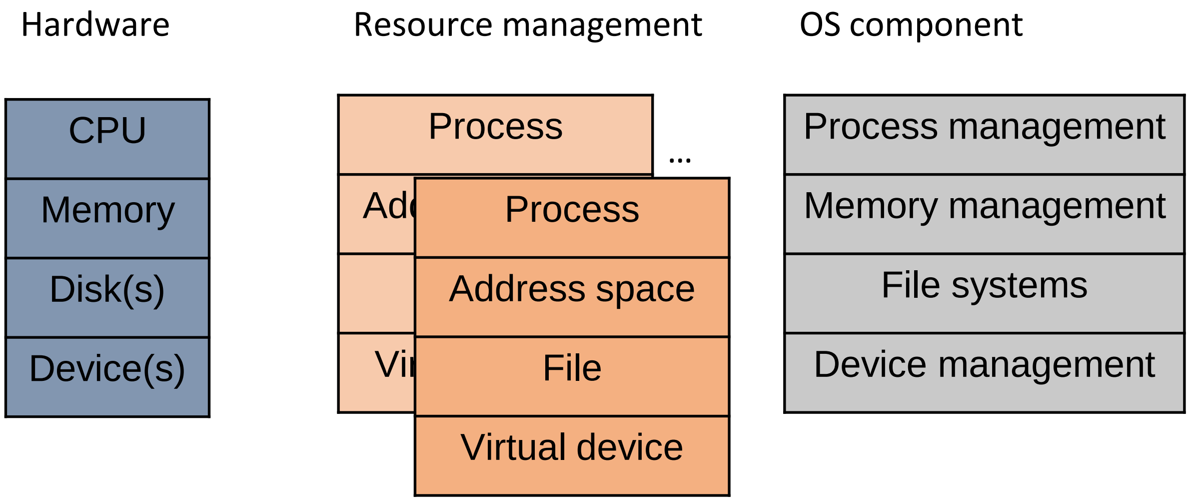
What does the OS do?

- Resource management: allocates hardware resources between programs



What does the OS do?

- Resource management: allocates hardware resources between programs



A Simple Example

- Many users want to compute
- OS allocates CPU to different users

Another Simple Example

- Many users want to use memory

- OS allocates memory between users

A Final Example

- Many files need to be stored on disk

- OS allocates disk space to files

What does the OS do?

- Abstraction: makes hardware easier to use
- Resource management: allocates hardware resources between programs

- **OS does *both* at the same time**

What Is and What Is Not in the OS

- Web browser?
- Graphics library?
- Device driver?
- Printer server?

What Is and What Is Not in the OS

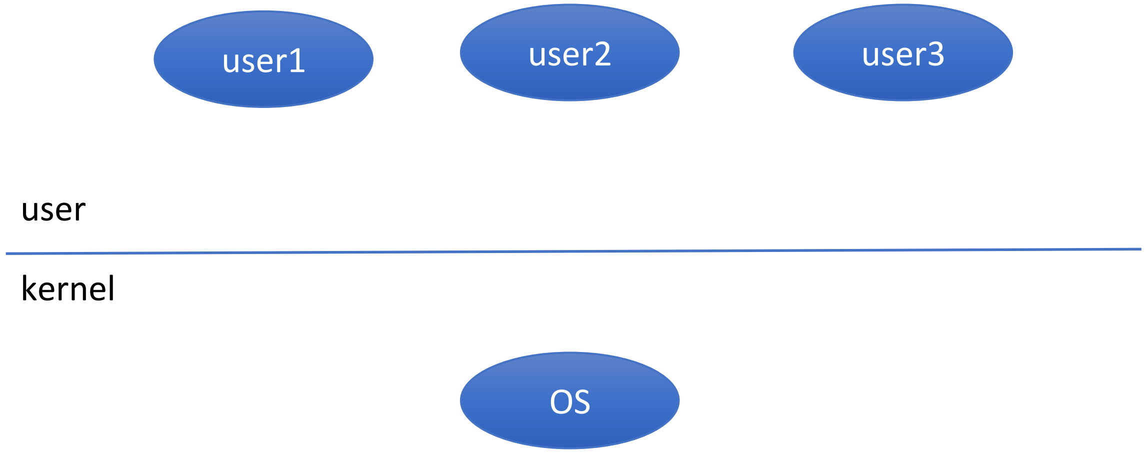
- Web browser: only abstraction
  - Not considered part of the OS
- Graphics library: only abstraction
  - Not considered part of the OS
- Device driver: both
  - Part of the OS
- Printer server: both
  - Part of the OS

Where does the OS live?

A Bit of Computer Architecture: CPU: Dual-Mode Operation

- Kernel mode vs. user mode
- Mode bit provided by hardware

User/OS Separation



Kernel Mode

- Privileged instructions:
  - Set mode bit
  - …
- Direct access to all of memory
- Direct access to devices

User Mode

- No privileged instructions:
  - Set mode bit
  - …
- No direct access to all of memory
- No direct access to devices

In General

- OS runs in kernel mode
- Applications run in user mode

- This allows OS
  - To protect itself
  - To manage applications/devices

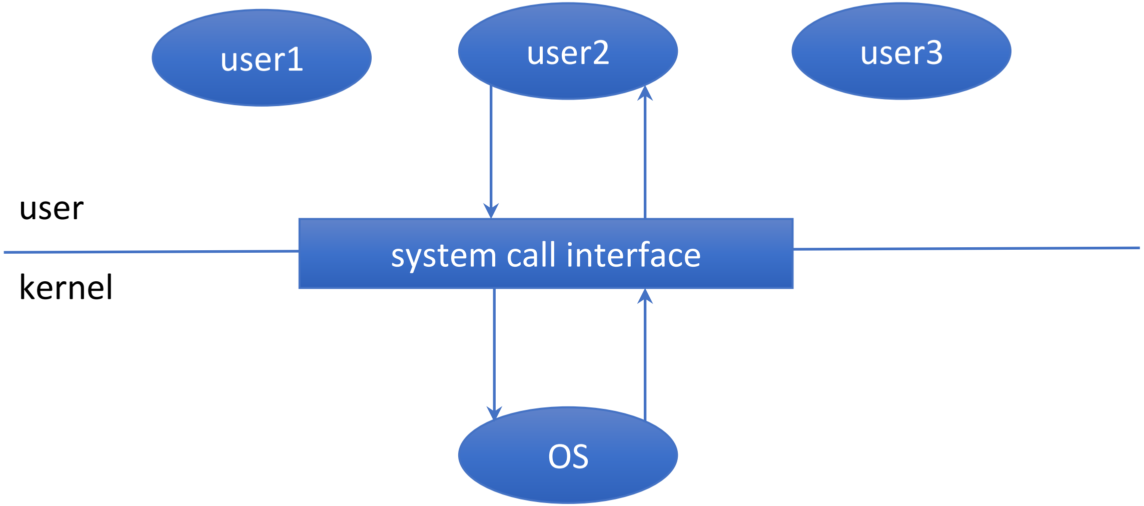
From Kernel to User Mode

- By the OS setting the mode bit to user
- Usually as a by-product of an instruction

From User to Kernel Mode

- By a device generating an interrupt
- By a program executing a trap or system call

System Calls:
Across User/Kernel Boundary



System Calls

- Are the *only* interface from program to OS
- Narrow interface essential for integrity of OS

System Calls in Linux?

System call number	System call name
0	restart_syscall
1	exit
2	fork
3	read
4	write
5	open
6	close
7	waitpid
8	creat
9	link
10	unlink
…	

System Calls in Linux?

System call number	System call name
…	
350	name_to_handle_at
351	open_by_handle_at
352	clock_adjtime
353	syncfs
354	sendmmsg
355	setns
356	process_vm_readv
357	process_vm_writev

System Call Implementation

- Architecture-specific

System Call Identification

- Unique system call number

System call number	System call name
0	restart_syscall
1	exit
2	fork
3	read
4	write
5	open
6	close
…	

To Perform a Given System Call

- Architecture-specific, example for x86
- Put system call number in register %eax
- Execute system call instruction

System Call Parameter Passing

- Again, architecture-specific

- Put in designated registers
- Put on the stack
- Put in table and have register point to it

In Linux/x86

- System call number in %eax register
- Parameters in registers
- If more parameters, register used as pointer

Question

- Ever called the OS?

Question

- Ever called the OS?
  - Yes, of course, e.g., any file system operation.
- Ever written a system call instruction?

Question

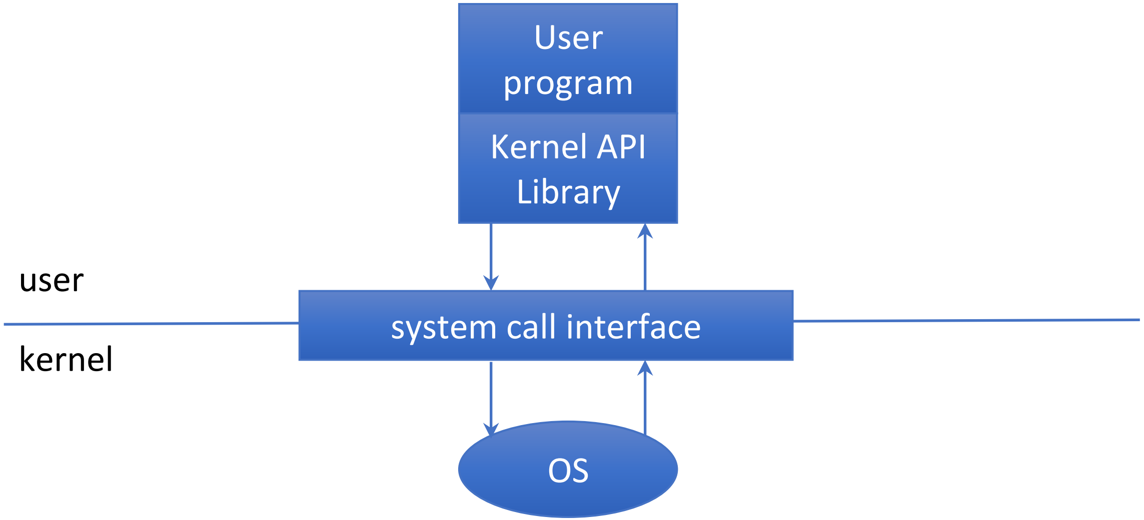
- Ever called the OS?
  - Yes, of course, e.g., any file system operation.
- Ever written a system call instruction?
  - I doubt it
- How so?

Answer: Kernel API

- A set of function calls that wrap system calls
- Easier to use
- More portable

- Example: Linux Kernel API

Kernel API


User program
Kernel API Library
user
system call interface
kernel
OS

Linux Kernel API

- Process management
  - fork(), exec(), wait(), …
- Memory management
  - mmap(), munmap(), sbrk(), …
- File system
  - open(), close(), read(), write(), …
- Device management
  - ioctl(), read(), write(), …
- Other examples
  - getpid(), alarm(), sleep(), chmod(), …

What Do Wrapper Functions Do?

- At the time of the call
  - Put arguments in registers
  - Put system call number in register %eax
  - Execute system call instruction
- At the time of the return
  - Take return value out of register
  - Return

Kernel API

```
main() {
        …
        write(…)
        …
}

write(…) {
        …
        execute system call instruction
        …
}
```

Question

- Ever called the OS?
  - Yes, of course, e.g., any file system operation.
- Ever written a system call instruction?
  - I doubt it
- Have you ever had to invoke the kernel API?

Question

- Ever called the OS?
  - Yes, of course, e.g., any file system operation.
- Ever written a system call instruction?
  - I doubt it
- Have you ever had to invoke the kernel API?
  - Maybe, maybe not

Answer: The Language Library

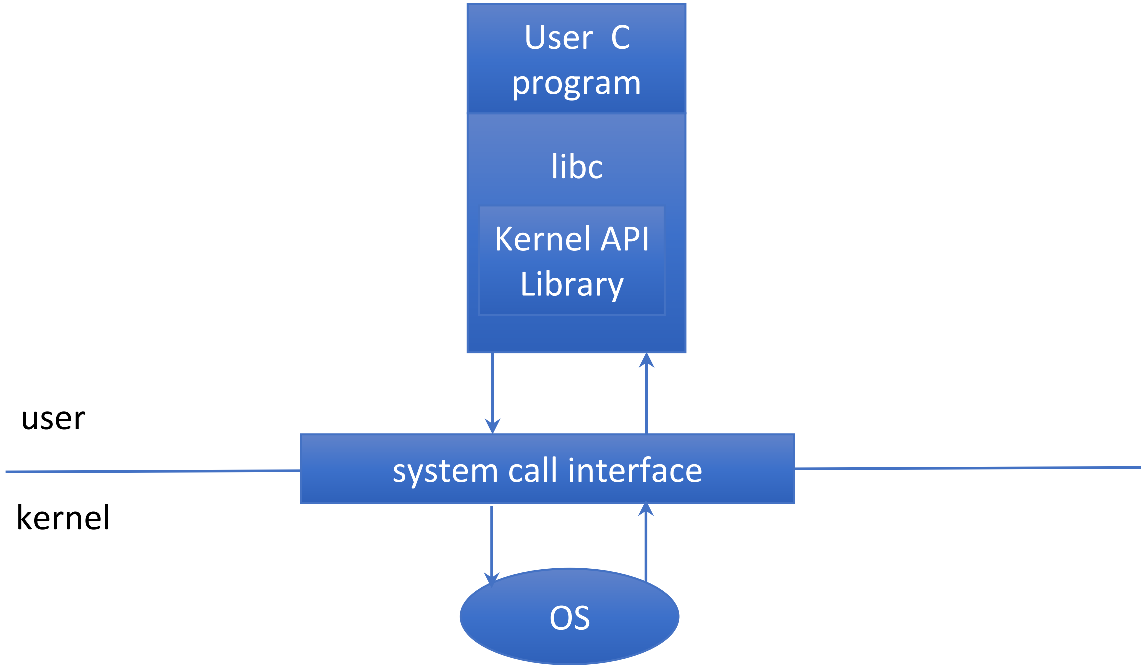
- A language-specific library
- Wraps the kernel API

- Classic example: the standard C library libc

libc

- printf, sprintf, fprintf, …
- getchar, putchar, …

libc


User C program
libc
Kernel API Library
user
system call interface
kernel
OS

libc

```
#include <stdio.h>
main() {
        …
        printf(…)
        …
}

printf(…) {
        …
        write(…)
        …
}

write(…) {
        …
        execute system call instruction
        …
}
```

Please Note!

- libc wraps system call to look like function call
- Inside the libc function, the system call is *not* a function call
- It is a user – kernel transition
  - From one program (user) to another (kernel)
  - Much more expensive

Traps

- Trap (aka Exception) is generated by CPU as a result of error
  - Divide by zero
  - Execute privileged instruction in user mode
  - Illegal access to memory
  - …
- Works like an “involuntary” system call
  - Sets mode to kernel mode
  - Transfers control to kernel

Interrupts

- Generated by a device that needs attention
  - Packet arrived from the network
  - Disk I/O completed
  - …

OS Control Flow

OS Control Flow: Event-Driven Program

- Nothing to do

Do nothing

OS Control Flow:
Event-Driven Program

- Nothing to do

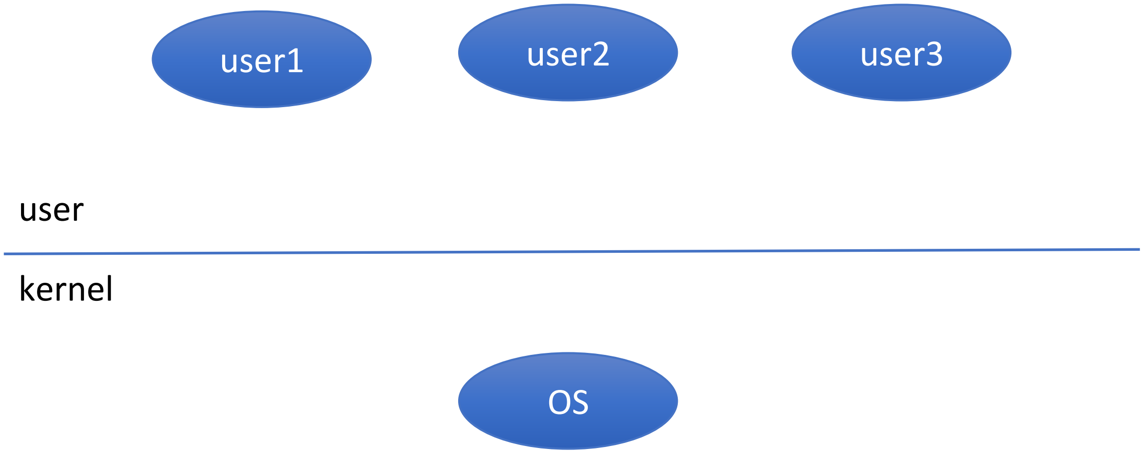
Do nothing

- Interrupt (from device)
- Trap (from process)
- System call (from process}

Start running

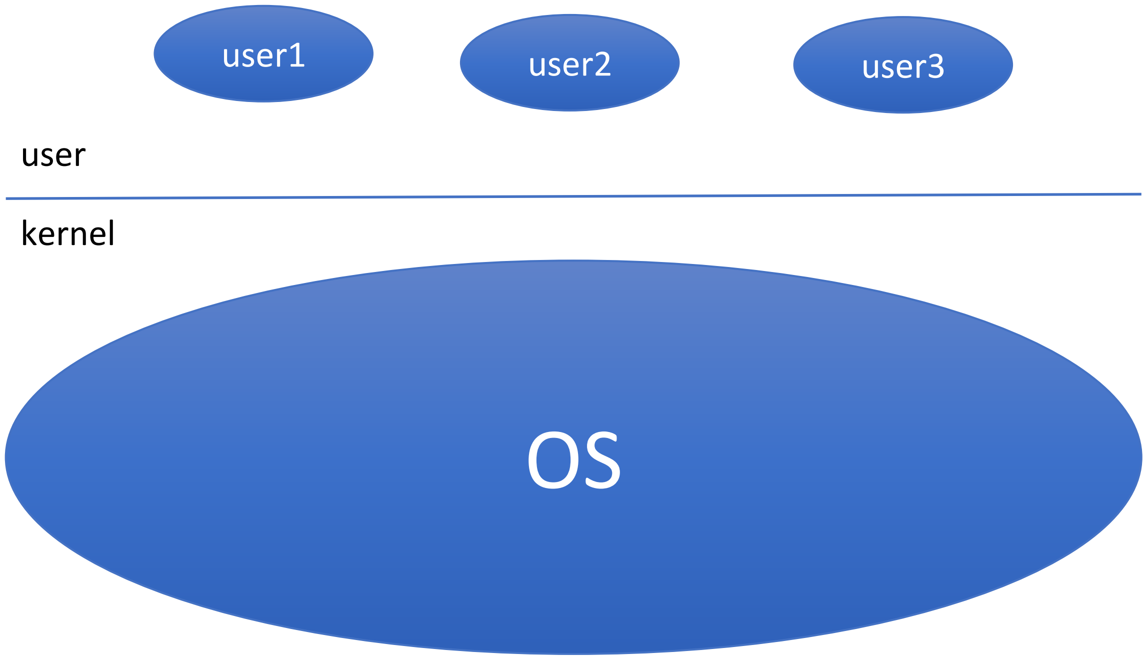
OS Structure

User/OS Separation



This approach is called the “monolithic OS”

It looks more like this



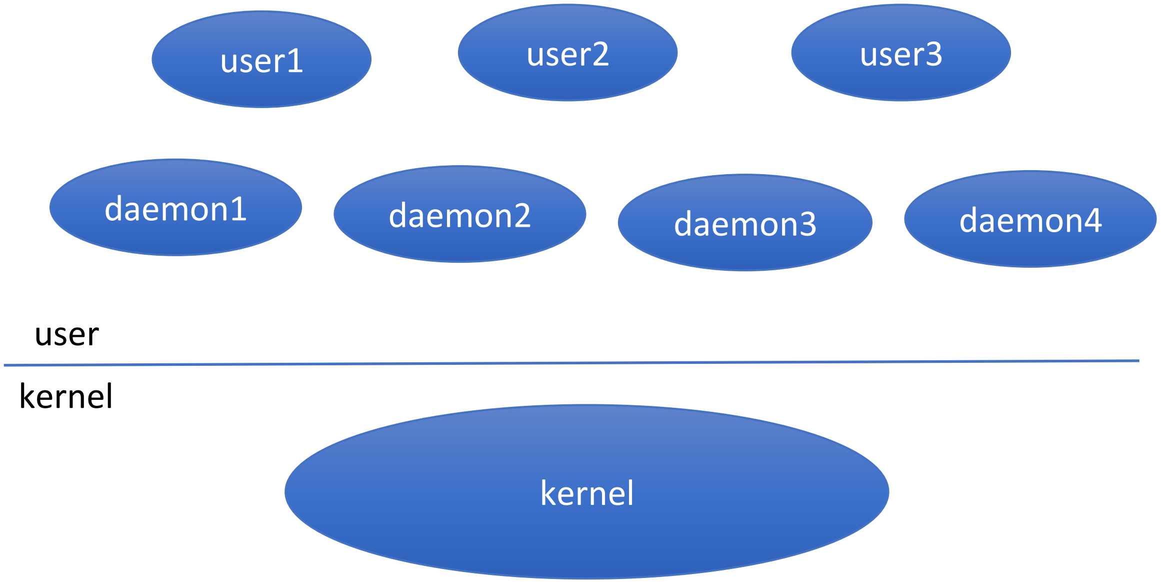
Downside of Monolithic OS

- The OS is a huge piece of software
  - Millions of lines of code and growing
- Something goes wrong in kernel mode
  - Most likely, machine will halt or crash
- Incentive to move stuff out of kernel mode

No need for entire OS in kernel mode

- Some pieces can be in user mode
  - No need for privileged access
  - No need for speed
- Example: daemons
  - System log
  - Printer daemon
  - Etc.

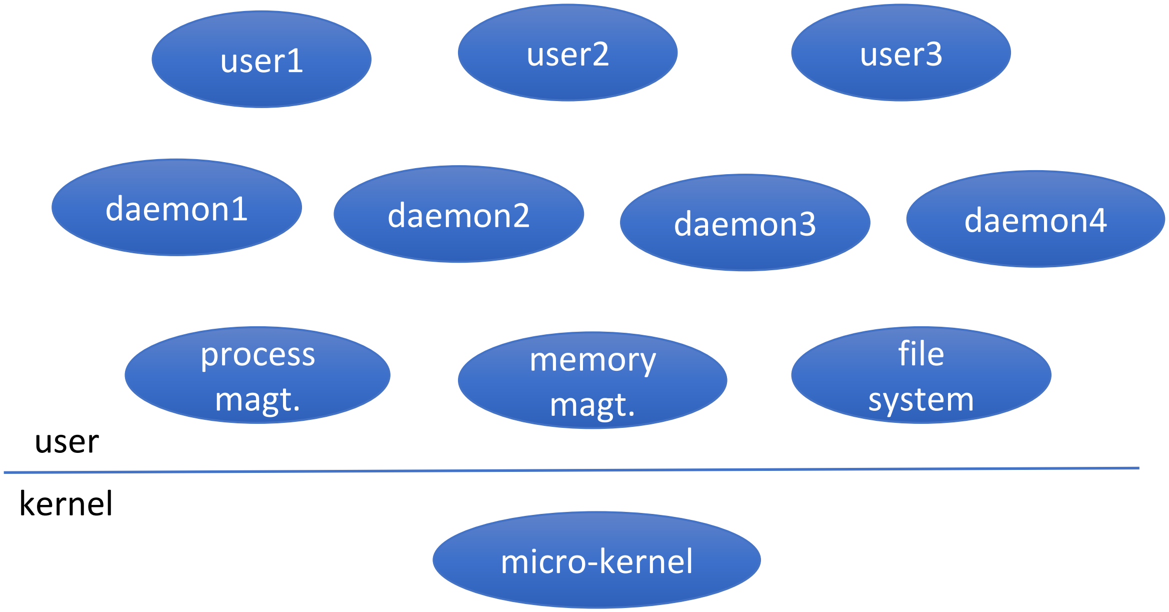
User/OS Separation: Systems Programs



The Ultimate Minimum: Microkernel

- Absolute minimum in kernel mode
  - Interprocess communication primitives
- All the rest in user mode

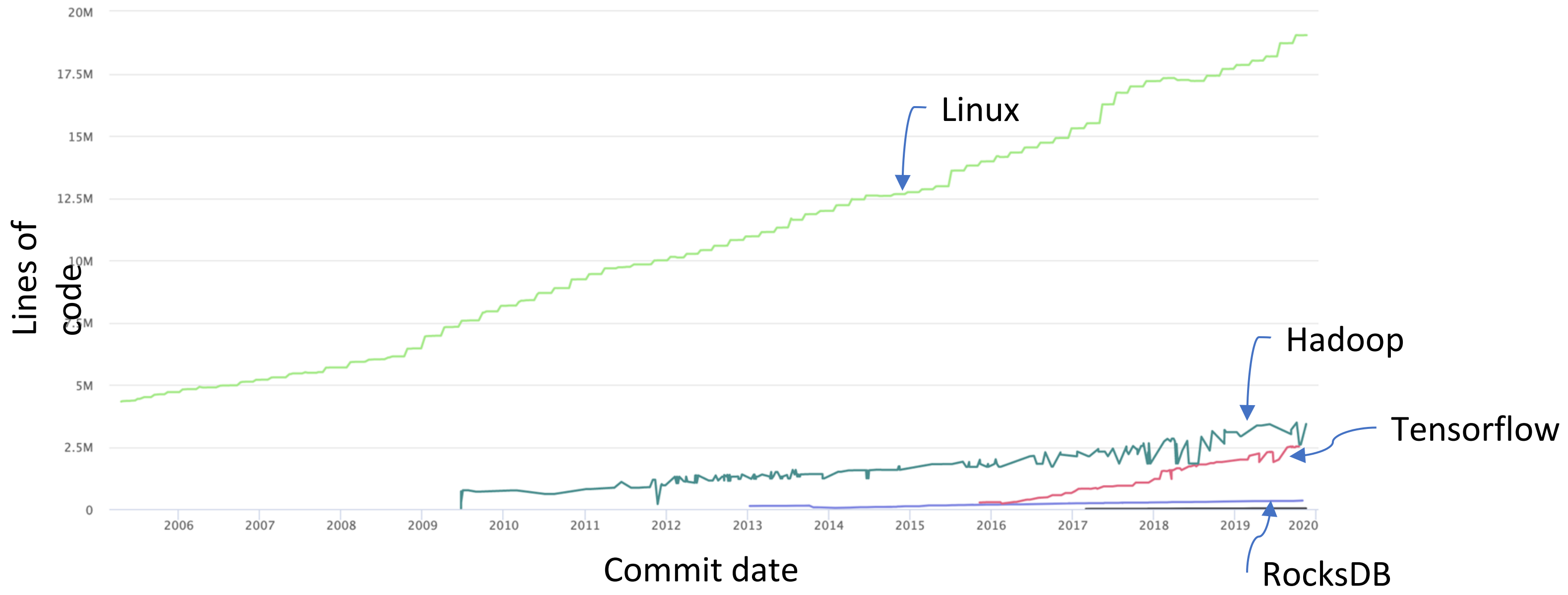
Microkernel



In Practice

- Microkernels have failed commercially
  - Except for niches like embedded computing
- The “systems programs” model has won out

The Price: Lines of Code in Linux Kernel



Source: https://panthema.net/2019/1122-Lines-of-Code-Plotted-over-Time/

Summary – Key Concepts

- What does the OS do?
  - Abstraction, resource management
- Where does the OS live?
  - User mode / Kernel mode
- OS interfaces
  - System call interface, Kernel API, Language library
- OS structure
  - Monolithic, systems OS, microkernel

Further Reading

Operating Systems: Three Easy Pieces by R. & A. Arpaci-Dusseau

Chapter 2 https://pages.cs.wisc.edu/~remzi/OSTEP/

Credits:

Some slides adapted from the OS courses of Profs. Remzi and Andrea Arpaci-Dusseau (University of Wisconsin-Madison), Prof. Willy Zwaenepoel (University of Sydney), Prof. Youjip Won (Hanyang University), and Prof. Natacha Crooks (UC Berkeley)