Principles of Operating Systems

Lecture 1 - Introduction and overview, operating system structure

Ardalan Amiri Sani (ardalan@uci.edu)

[*lecture slides contains some content adapted from : Silberschatz textbook authors, Anderson textbook authors, John Kubiatowicz (Berkeley), John Ousterhout(Stanford), previous slides by Prof. Nalini Venkatasubramanian,* http://www-inst.eecs.berkeley.edu/~cs162/ *and others*]

Staff

- Instructor
  - Ardalan Amiri Sani (ardalan@uci.edu)

Staff

Teaching Assistants:

- Saehanseul Yi <saehansy@uci.edu>
- Farzad Habibi <habibif@uci.edu>
- Ashwin Gerard Colaco <acolaco@uci.edu>

Course logistics and details

- Course web page -
  - https://www.ics.uci.edu/~ardalan/courses/os/index.html

- Discussions
  - Wednesdays 1:00-1:50pm (SE2 1304)
  - Wednesdays 2:00-2:50pm (SE2 1304)
  - Wednesdays 5:00-5:50pm (MSTB 118)
  - Wednesdays 6:00-6:50pm (MSTB 118)

Course logistics and details

- Textbook:

  Operating System Concepts -- Ninth Edition
  A. Silberschatz, P.B. Galvin, and G. Gagne
  (Tenth, Eighth, Seventh, Sixth, and Fifth editions are fine as well).

- Other suggested Books
  - Operating Systems: Principles and Practice, by T. Anderson and M. Dahlin (second edition)
  - Modern Operating Systems, by Tanenbaum (Third edition)
  - Principles of Operating Systems, by L.F. Bic and A.C. Shaw, 2003.
  - Operating Systems: Three Easy Pieces, by Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau

Course logistics and details

- Homeworks and Assignments
  - 4 written homeworks in the quarter
  - 1 **optional** programming assignment (knowledge of C).
    - Multistep assignment – don't start in last week of classes!!!
  - Late homework policy.
    - Lose 10% of grade for every late hour.
  - All submissions will be made using Gradescope
- Tests
  - Midterm – Thursday, Week 6 (during class time)
  - Final Exam – per UCI course catalog (Thu, 3/23, 1:30pm-3:30pm)

Grading Policy

- Will pick the best of the following two:
- Grade 1:
  - Written Homeworks - 20%
    - 4 written homeworks each worth 5% of the final grade.
  - Project - 20% of the final grade (4% for lab0, 16% for lab1)
  - Midterm - 25% of the final grade
  - Final exam - 35% of the final grade

- Grade 2:
  - Written Homeworks - 20%
  - Midterm - 35% of the final grade
  - Final exam - 45% of the final grade

- Curve will be used if needed.

Lecture Schedule

- Week 1
  - Introduction to Operating Systems, Computer System Structures, Operating System Structures
- Week 2
  - Processes and Threads
- Week 3
  - Processes and Threads, and CPU Scheduling
- Week 4
  - Scheduling
- Week 5
  - Process Synchronization

Lecture Schedule

- Week 6
    - Deadlocks, Midterm exam
- Week 7
    - Memory Management
- Week 8
    - Memory Management, Virtual Memory
- Week 9
    - File Systems Interface and Implementation
- Week 10
    - I/O Subsystems

Classes I will most likely miss

- 2/23/2023: Need to attend HotMobile’23 as the general chair

  (Will announce later if other lectures are to be missed)
- Will announce later the plan for missed lectures
  - Sometimes, the TA will replace me on those dates

Office hours

- Instructor
  - Thursdays 9:30am-10:30am (Zoom link on canvas)
- TA
  - Tuesdays 4:00pm-5:00pm (Zoom link on Canvas)

Office hours will start on the second week of classes

Piazza

- https://piazza.com/uci/winter2023/compsci143a

Slides

- Will upload first draft of the slides for all of the week on Tuesday
- Might (and most likely will) update slides for each class before the class
  - Will mention on the website which pages have been updated

Overview

- What is an operating system?
- Operating systems history
- Computer system and operating system structure

What is an Operating System?

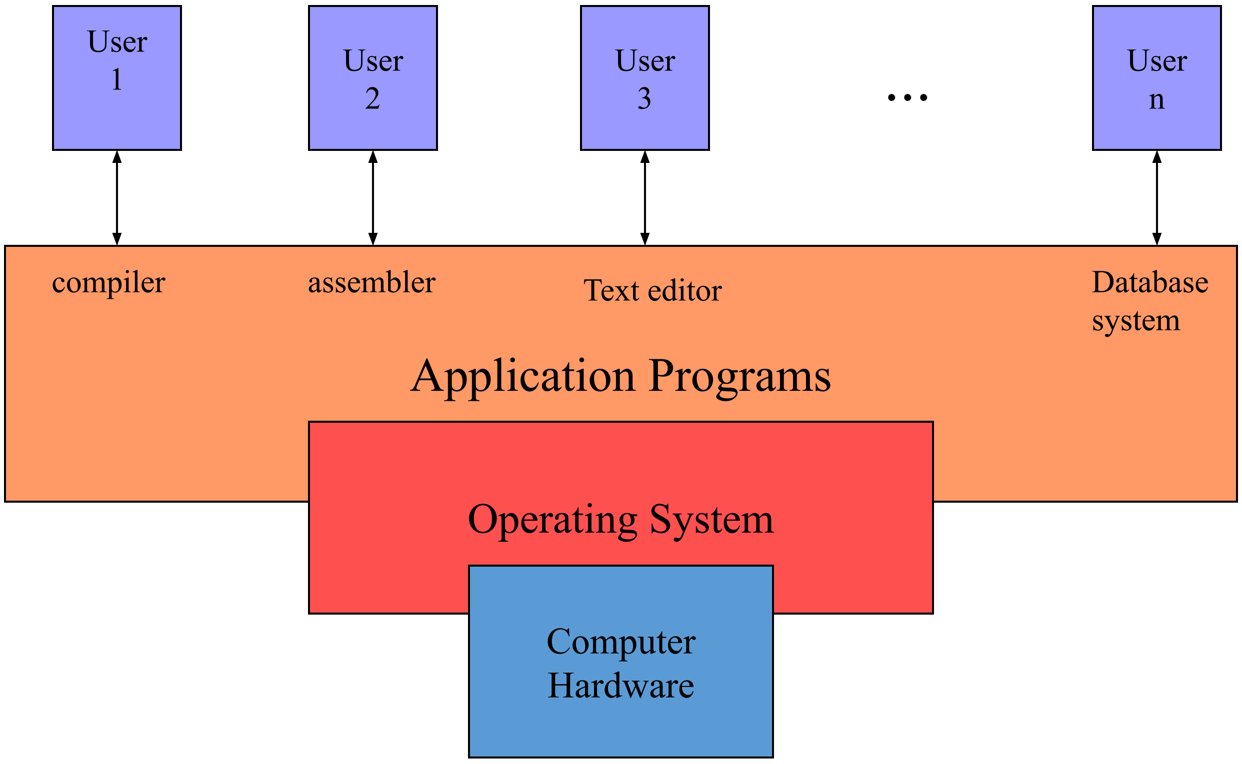
What is an Operating System?

- OS is the software that acts an intermediary between the applications and computer hardware.

Computer System Components

- Hardware
  - Provides basic computing resources (CPU, memory, I/O devices).
- Operating System
  - Controls and coordinates the use of hardware among application programs.
- Application Programs
  - Solve computing problems of users (compilers, database systems, video games, business programs).
- Users
  - People, other computers

Abstract View of System



Operating system roles

- Referee
  - Resource allocation among users, applications
  - Isolation of different users, applications from each other
  - Communication between users, applications

Operating system roles

- Illusionist
  - Each application appears to have the entire machine to itself
  - Infinite number of processors, (near) infinite amount of memory, reliable storage, reliable network transport

Operating system roles

- Glue
  - Libraries, user interface widgets, …
  - Reduces cost of developing software

Example: file systems

- Referee
  - Allocates storage space for files for each user
  - Prevent users from accessing each other's files without permission
- Illusionist
  - Files can grow (nearly) arbitrarily large
  - Files persist even when the machine crashes in the middle of a save
- Glue
  - Named directories, printf, …

OS challenges

OS challenges

- Reliability
  - Does the system do what it was designed to do?

OS challenges

- Availability
  - What portion of the time is the system working?
  - Mean Time To Failure (MTTF), Mean Time to Repair

OS challenges

- Security
  - Can the system be compromised by an attacker?

OS challenges

- Privacy
  - Data is accessible only to authorized users

OS challenges

- Performance
  - Latency/response time
    - How long does an operation take to complete?
  - Throughput
    - How many operations can be done per unit of time?
  - Overhead
    - How much extra work is done by the OS?
  - Fairness
    - How equal is the performance received by different users?
  - Predictability
    - How consistent is the performance over time?

OS challenges

- Portability
  - For programs:
    - Application programming interface (API)
  - For the kernel
    - Hardware abstraction layer

OS needs to keep pace with hardware improvements

	1981	1997	2014	Factor (2014/1981)
Uniprocessor speed (MIPS)	1	200	2500	2.5K
CPUs per computer	1	1	10+	10+
$/Processor MIPS	$100K	$25	$0.20	500K
DRAM Capacity (MiB)/$	0.002	2	1K	500K
Disk Capacity (GiB)/$	0.003	7	25K	10M
Home Internet	300 bps	256 Kbps	20 Mbps	100K
Machine room network	10 Mbps (shared)	100 Mbps (switched)	10 Gbps (switched)	1000
Ratio of users to computers	100:1	1:1	1:several	100+

Why should I study Operating Systems?

Why should I study Operating Systems?

- Need to understand interaction between the hardware and software
- Need to understand basic principles in the design of computer systems
    - efficient resource management, security, etc.

Why should I study Operating Systems?

- Because it enables you to do things that are difficult/impossible otherwise.

Example: Rio: I/O sharing implemented in the operating system kernel

(Slides on Rio are not part of the course material)

Observation: I/O devices important for personal computers

A personal computer today

- Super AMOLED display
- Capacitive touchscreen (multitouch)
- Audio (speaker, microphone)
- Vibration
- S pen
- 13 MP front camera
- 2 MP back camera
- Accelerometer
- Gyroscope
- Proximity sensor
- Compass
- Barometer
- Temperature sensor
- Humidity sensor
- Gesture sensor
- GPS
- 4G LTE
- NFC
- WiFi
- Bluetooth
- Infrared
- 64 GB internal storage (extended by microSD)
- Adreno 330 GPU
- Hexagon DSP
- Multimedia processor

A personal computer today

- Super AMOLED display
- Capacitive touchscreen (multitouch)
- Audio (speaker, microphone)
- Vibration
- S pen
- 13 MP front camera
- 2 MP back camera
- Accelerometer
- Gyroscope
- Proximity sensor
- Compass
- Barometer
- Temperature sensor
- Humidity sensor
- Gesture sensor
- GPS
- 4G LTE
- NFC
- WiFi
- Bluetooth
- Infrared
- 64 GB internal storage (extended by microSD)
- Adreno 330 GPU
- Hexagon DSP
- Multimedia processor

interaction

A personal computer today

- Super AMOLED display
- Capacitive touchscreen (multitouch)
- Audio (speaker, microphone)
- Vibration
- S pen
- 13 MP front camera
- 2 MP back camera
- Accelerometer
- Gyroscope
- Proximity sensor
- Compass
- Barometer
- Temperature sensor
- Humidity sensor
- Gesture sensor
- GPS
- 4G LTE
- NFC
- WiFi
- Bluetooth
- Infrared
- 64 GB internal storage (extended by microSD)
- Adreno 330 GPU
- Hexagon DSP
- Multimedia processor

sensing

A personal computer today

- Super AMOLED display
- Capacitive touchscreen (multitouch)
- Audio (speaker, microphone)
- Vibration
- S pen
- 13 MP front camera
- 2 MP back camera
- Accelerometer
- Gyroscope
- Proximity sensor
- Compass
- Barometer
- Temperature sensor
- Humidity sensor
- Gesture sensor
- GPS
- 4G LTE
- NFC
- WiFi
- Bluetooth
- Infrared
- 64 GB internal storage (extended by microSD)
- Adreno 330 GPU
- Hexagon DSP
- Multimedia processor

connectivity, storage

A personal computer today

- Super AMOLED display
- Capacitive touchscreen (multitouch)
- Audio (speaker, microphone)
- Vibration
- S pen
- 13 MP front camera
- 2 MP back camera
- Accelerometer
- Gyroscope
- Proximity sensor
- Compass
- Barometer
- Temperature sensor
- Humidity sensor
- Gesture sensor
- GPS
- 4G LTE
- NFC
- WiFi
- Bluetooth
- Infrared
- 64 GB internal storage (extended by microSD)
- Adreno 330 GPU
- Hexagon DSP
- Multimedia processor

acceleration

Multiple computers for unique I/O

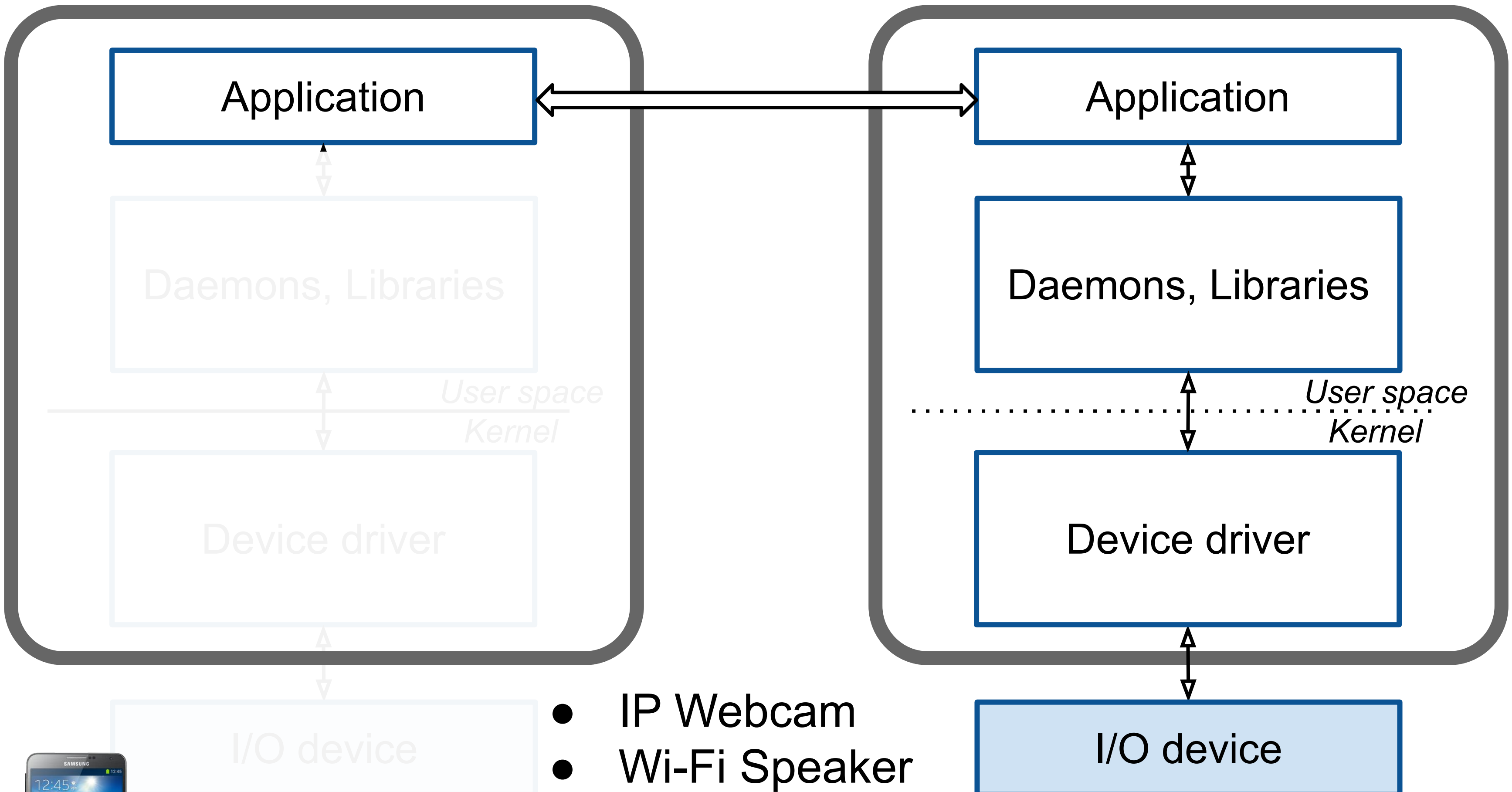
Multiple computers for unique I/O

Multiple computers for unique I/O

I/O sharing

How to build this?

Application layer



- IP Webcam
- Wi-Fi Speaker
- MightyText

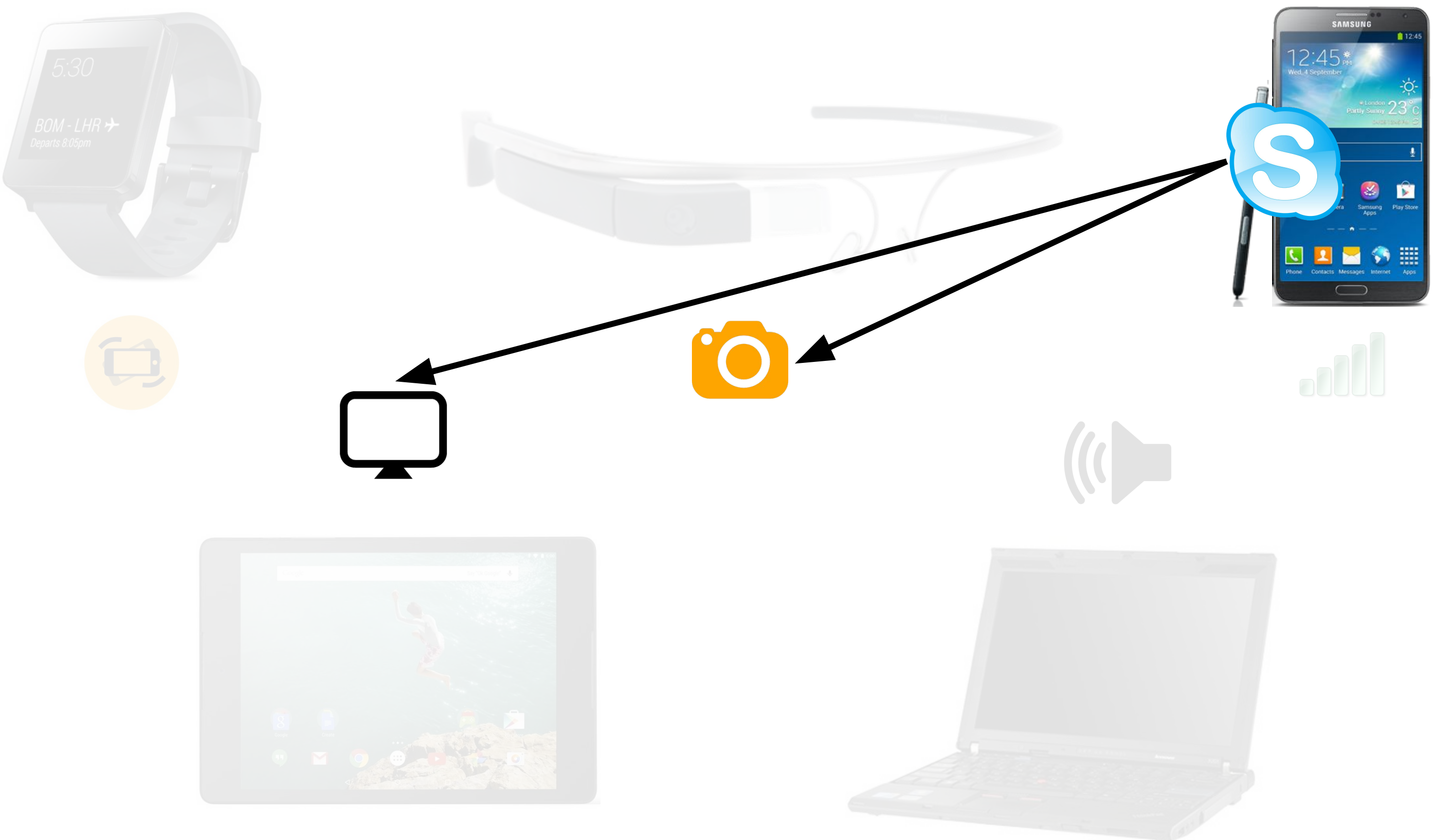
Client

Server

Do not meet our criteria

- High engineering effort
- No support for legacy applications
- No support for all I/O device features

Rio: I/O servers for sharing I/O between mobile systems

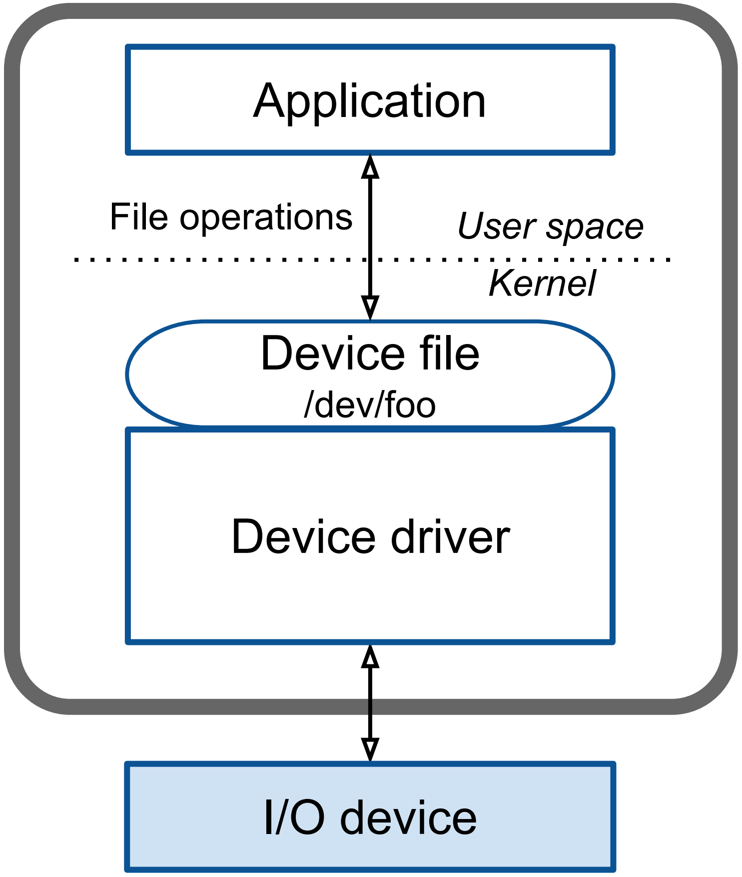


Ardalan Amiri Sani, Kevin Boos, Min Hong Yun, and Lin Zhong, "Rio: A System Solution for Sharing I/O between Mobile Systems," in *Proc.* ***ACM MobiSys***, June 2014. **(Best Paper Award)**

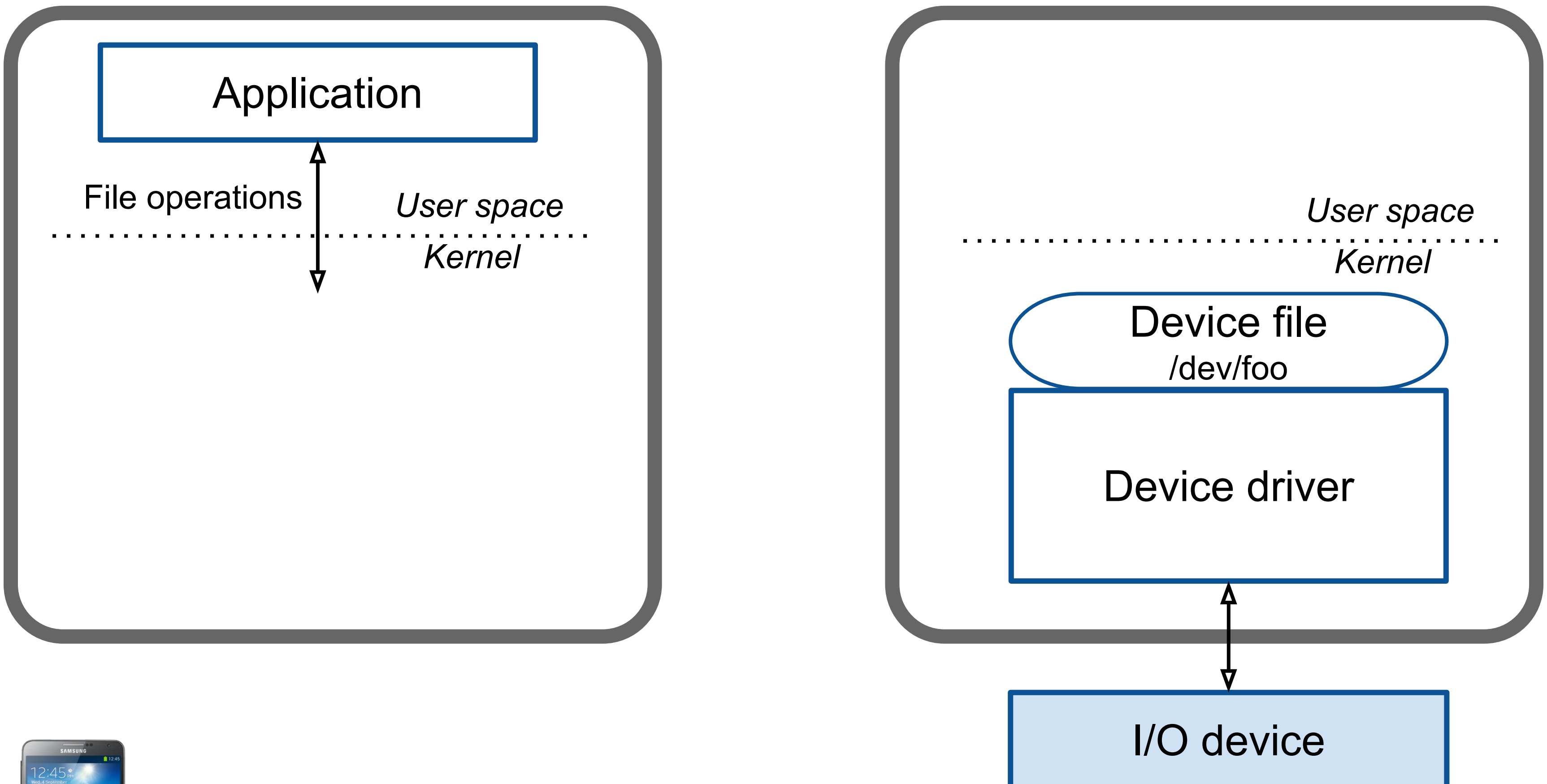
Key idea: device files as the boundary

I/O devices abstracted as
(device) files in Unix-like OSes
e.g., /dev/foo

Key idea: device files as the boundary



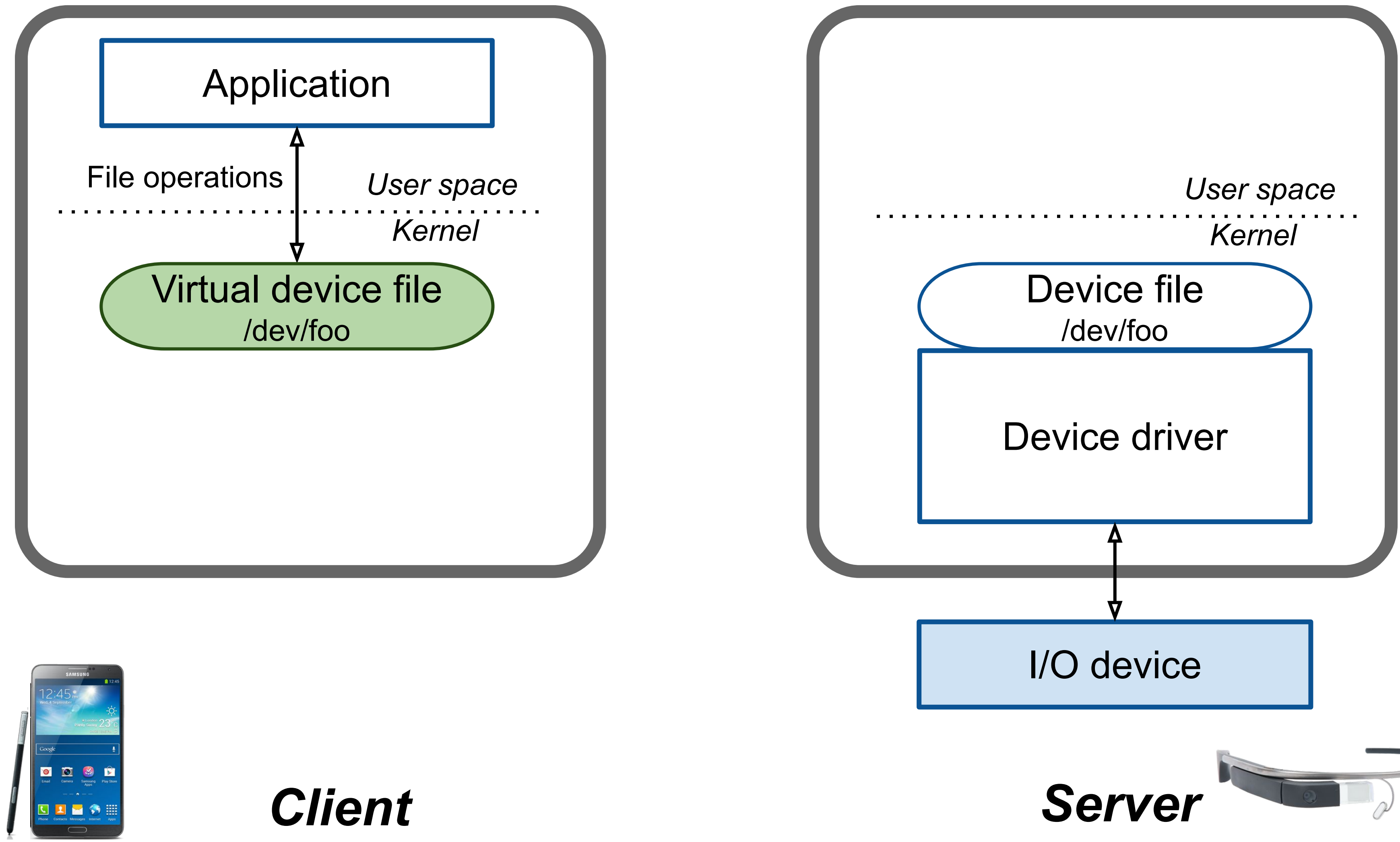
Key idea: device files as the boundary



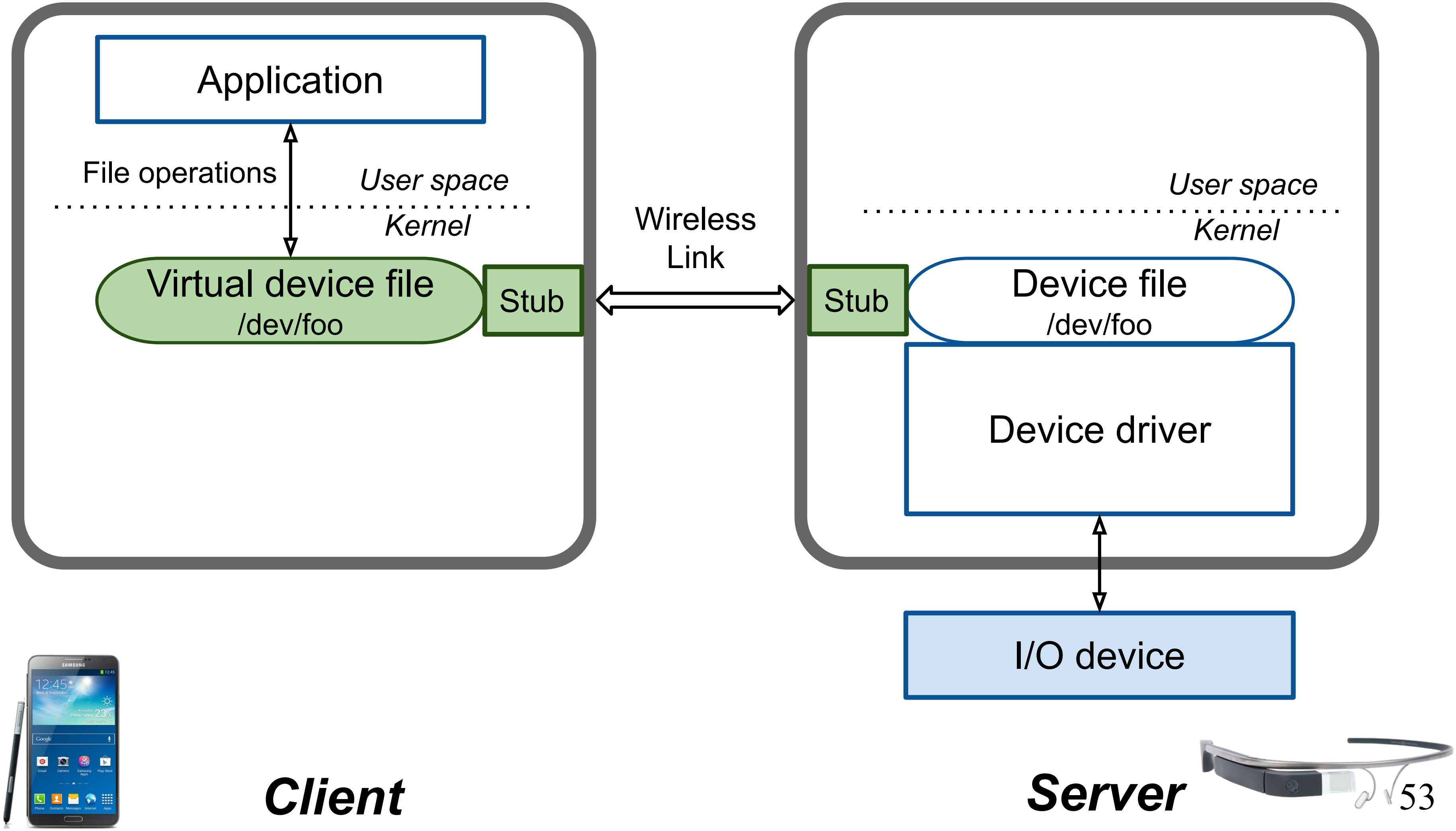
Client

Server

Key idea: device files as the boundary



Key idea: device files as the boundary



Video demo of Rio

https://www.yecl.org/rio.html

(end of slides on Rio)

Operating systems are everywhere

Operating systems are everywhere

Operating systems are everywhere






Operating systems are everywhere

Overview

- What is an operating system?
- Operating systems history
- Computer system and operating system structure

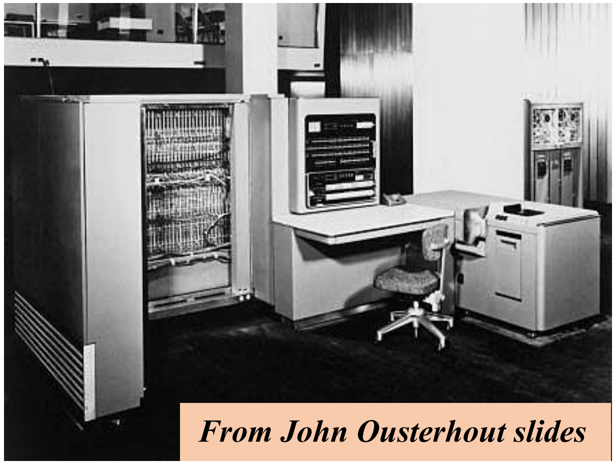
Operating systems history (start)

- Early Systems
- Simple Batch Systems
- Multiprogrammed Batch Systems
- Time-sharing Systems
- Personal and Mobile Computer Systems

(The slides on the OS history are for your own study and won't be used in the exams.)

Early Systems - Bare Machine (1950s)

Hardware – ***expensive*** ; **Human –** ***cheap***



From John Ousterhout slides

- Structure
  - Large machines run from console
  - Single user system
    - Programmer/User as operator
  - Punched cards, paper tape, and magnetic tape
- Early software
  - Assemblers, compilers, linkers, loaders, device drivers, libraries of common subroutines.
- Secure execution
- Inefficient use of expensive resources
  - Low CPU utilization, high setup time.

Batch Systems (1960's)

- Reduce setup time by batching jobs with similar requirements.
- Hire an operator
  - User is NOT the operator
- Automatic job sequencing
  - Forms a rudimentary OS.
  - Resident Monitor
    - Holds initial control, control transfers to job and then back to monitor.
  - Problem
    - Need to distinguish job from job and data from program.
    - Special cards indicate what to do.

From John Ousterhout slides

Batch Systems (1960's)

- Problem: I/O is slow!
- Solutions to speed up I/O:
- Offline Processing: Reading from cards to tapes and writing from tapes to line printers were done offline.
  - User submits card deck
  - cards put on tape
  - tape processed by operator
  - output written to tape
  - tape printed on printer
  - Separate user from computer
- Problems
  - Long turnaround time - up to 2 DAYS!!!
  - Low CPU utilization
    - I/O and CPU could not overlap;   slow mechanical devices.

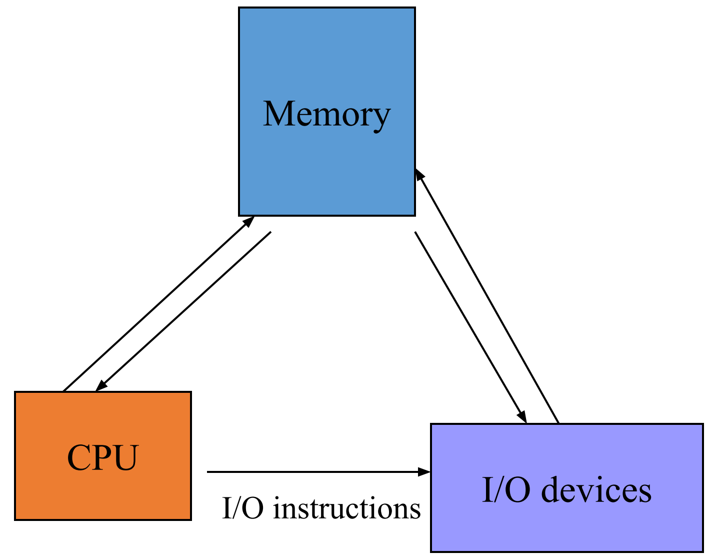
From John Ousterhout slides

Batch Systems (1960's)

- Solution to speed up I/O: Spooling (Simultaneous Peripheral Operation On-Line)
  - Use disk (random-access device) as large storage for reading as many input files as possible and storing output files until output devices are ready to accept them.
  - Allows overlap - I/O for multiple jobs as well as computation of another.
  - Introduces notion of a job pool that allows OS to choose next job to run so as to increase CPU utilization.

Speeding up I/O: Direct Memory Access (DMA)

- Data moved directly between I/O devices and memory
- CPU can work on other tasks



Batch Systems - I/O completion

- How do we know that I/O is complete?
  - Polling:
    - Device sets a flag when it is busy.
    - Program tests the flag in a loop waiting for completion of I/O.
  - Interrupts:
    - On completion of I/O, device forces CPU to jump to a specific instruction address that contains the interrupt service routine.
    - After the interrupt has been processed, CPU returns to code it was executing prior to servicing the interrupt.

Multiprogramming

- Use interrupts to run multiple programs simultaneously
  - When a program performs I/O, instead of polling, execute another program till interrupt is received.
- Requires secure memory, I/O for each program.
- Requires intervention if program infinite loops.
- Requires CPU scheduling to choose the next job to run.

Timesharing

Hardware – *getting cheaper*; **Human** – *getting expensive*

- Programs queued for execution in FIFO order.
- Like multiprogramming, but timer device interrupts after a quantum (timeslice).
    - Interrupted program is returned to end of FIFO
    - Next program is taken from head of FIFO
- Control card interpreter replaced by command language interpreter.

Timesharing (cont.)

- Interactive (action/response)
  - when OS finishes execution of one command, it seeks the next control statement from user.
- File systems
  - online filesystem is required for users to access data and code.
- Virtual memory
  - Job is swapped in and out of memory to disk.

Personal Computing Systems - desktops

Hardware – ***cheap*** ; **Human** – ***expensive***

- Single user systems, portable.
- I/O devices - keyboards, mice, display screens, small printers.
- Single user systems may not need advanced CPU utilization or protection features.
- Advantages:
  - user convenience, responsiveness, ubiquitous

Personal Computing Systems - Mobile and wearable Systems

Hardware – ***very cheap*** ; **Human – *very expensive***

- Single user, multiple computers
- Laptops
- Smartphones
- Tablets
- Smart glasses
- Smart watches

(End of slides on history)

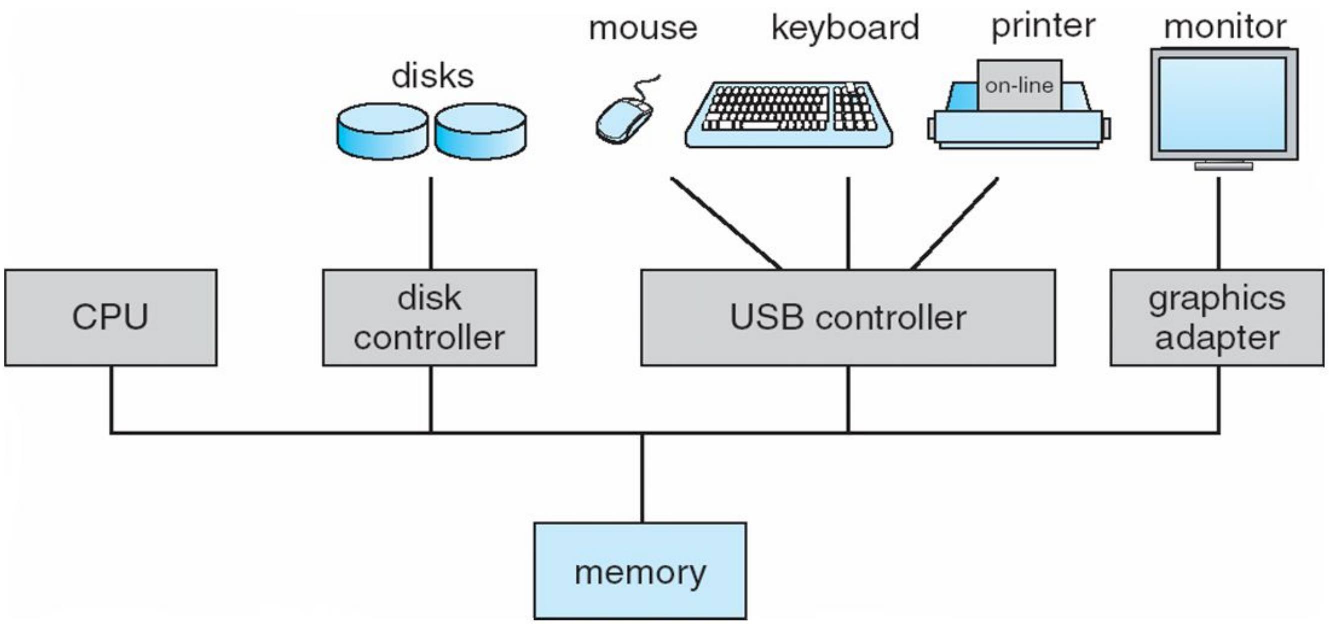
Overview

- What is an operating system?
- Operating systems history
- Computer system and operating system structure

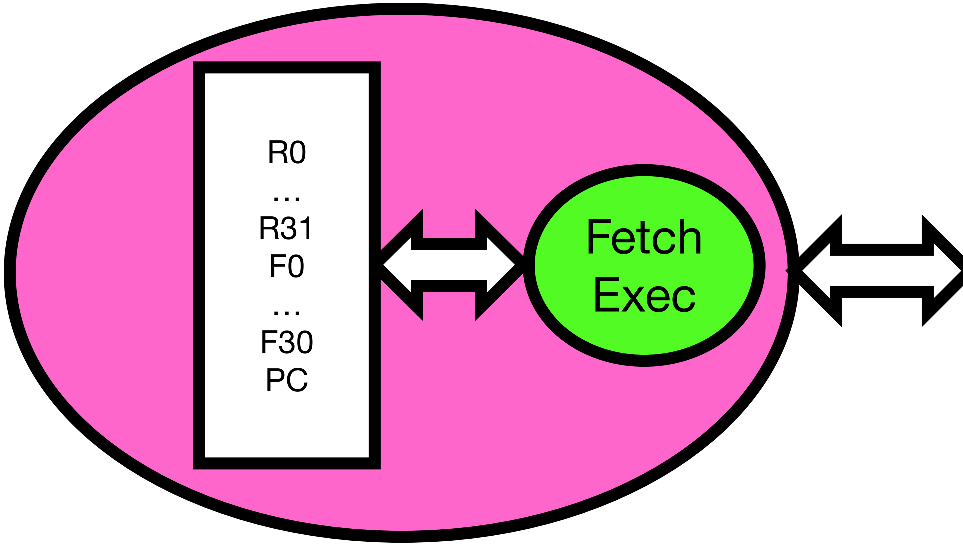
Computer System & OS Structures

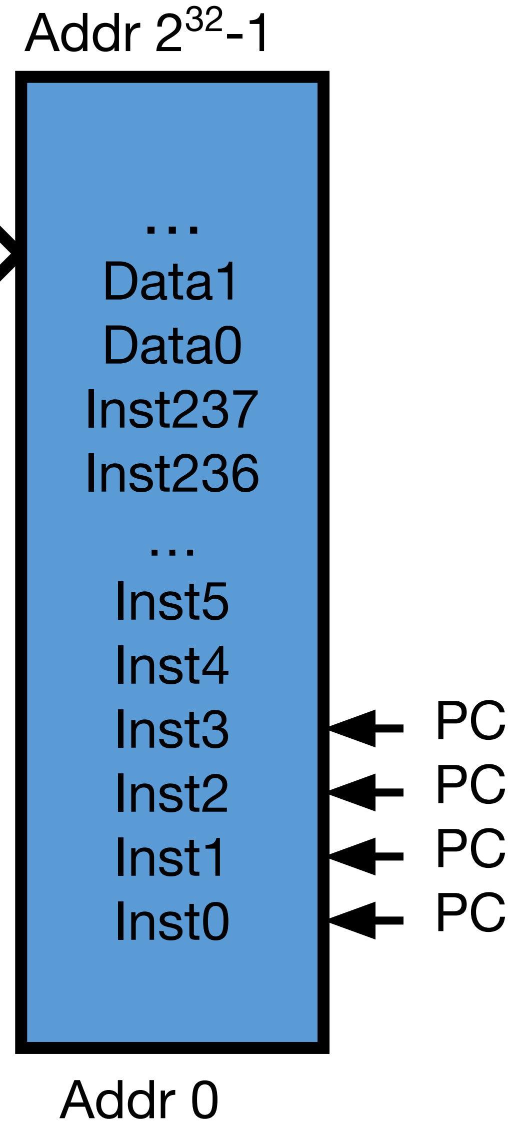
- Computer System Organization
- Process abstraction and hardware protection
- System call and OS services
- Storage architecture
- OS organization
- OS tasks

Computer System Organization



CPU execution

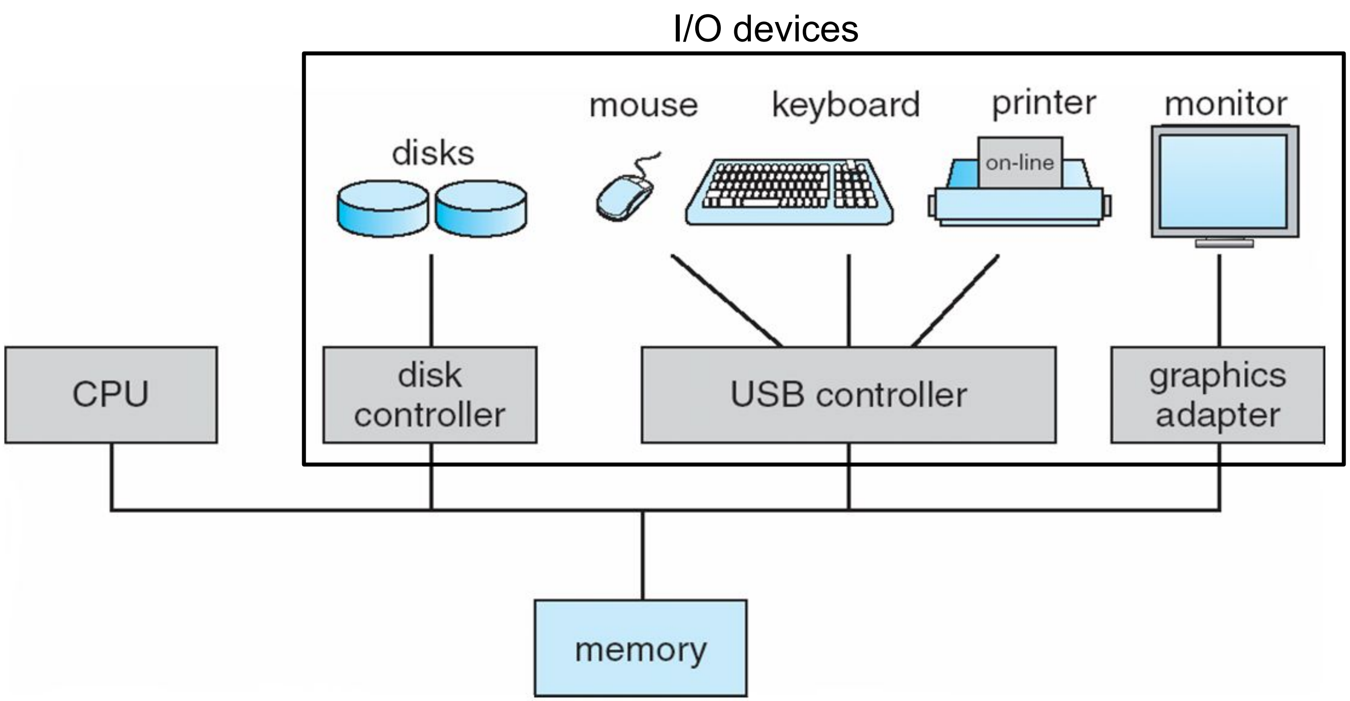




- Execution sequence:
  - Fetch Instruction at PC
  - Decode
  - Execute (possibly using registers)
  - Write results to registers/mem
  - PC = Next Instruction(PC)
  - Repeat

From Berkeley OS course

Computer System Organization

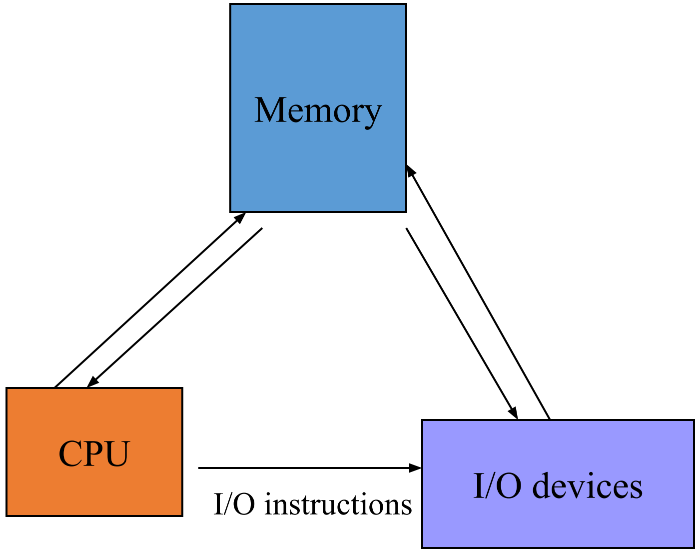


I/O devices

- I/O devices and the CPU execute concurrently.
- Each device controller is in charge of a particular device type
  - Each device controller has a local buffer. I/O is from the device to local buffer of controller
- CPU moves data from/to main memory to/from the local buffers

Direct Memory Access (DMA)

- Typically used for I/O devices with a lot of data to transfer (in order to reduce load on CPU).
- Device controller transfers blocks of data to/from local buffer directly to main memory without CPU intervention.



I/O completion

- How do we know that I/O is complete (e.g., data is ready in local buffer or DMA is complete)?

I/O completion

- How do we know that I/O is complete (e.g., data is ready in local buffer or DMA is complete)?
  - Polling:
    - Device controller sets a flag when it is busy.
    - Program tests the flag in a loop waiting for completion of I/O.
  - Interrupts:
    - On completion of I/O, device controller interrupts CPU.

Interrupts

- Interrupt transfers control to the interrupt service routine
  - Interrupt Service Routine: Segments of code that determine action to be taken for interrupt.
- Determining the type of interrupt
  - Polling: same interrupt handler called for all interrupts, which then polls all devices to figure out the reason for the interrupt
  - Interrupt Vector Table: different interrupt handlers will be executed for different interrupts

Interrupt Number	Address
0	0003h
1	000Bh
2	0013h
3	001Bh
4	0023h
5	002Bh
6	0033h
7	003Bh
8	0043h
9	004Bh
10	0053h
11	005Bh
12	0063h
13	006Bh
14	0073h
15	007Bh

Interrupt Number	Address
16	0083h
17	008Bh
18	0093h
19	009Bh
20	00A3h
21	00ABh
22	00B3h
23	00BBh
24	00C3h
25	00CBh
26	00D3h
27	00DBh
28	00E3h
29	00EBh
30	00F3h
31	00FBh

Interrupt handling

- OS preserves the state of the CPU

Interrupt handling

- OS preserves the state of the CPU
  - stores registers and the program counter (address of interrupted instruction).
- What happens to a new interrupt when the CPU is handling one interrupt?

Interrupt handling

- OS preserves the state of the CPU
  - stores registers and the program counter (address of interrupted instruction).
- What happens to a new interrupt when the CPU is handling one interrupt?
  - Incoming interrupts can be disabled (masked) while another interrupt is being processed. In this case, incoming interrupts may be lost or may be buffered until they can be delivered.
  - Incoming interrupts are delivered, i.e., nested interrupts.

Process Abstraction

Process Abstraction

- Process: an *instance* of a program, running with limited rights

Process Abstraction

- Process: an *instance* of a program, running with limited rights
  - Thread: a sequence of instructions within a process
    - Potentially many threads per process (for now 1:1)
  - Each process has a set of rights
    - Memory that the process can access (address space)
    - Other permissions the process has (e.g., which system calls it can make, what files it can access)

How to limit process rights?

Hardware Protection

- CPU Protection:
  - Dual Mode Operation
  - Timer interrupts
- Memory Protection
- I/O Protection

Should a process be able to execute any instructions?

Should a process be able to execute any instructions?

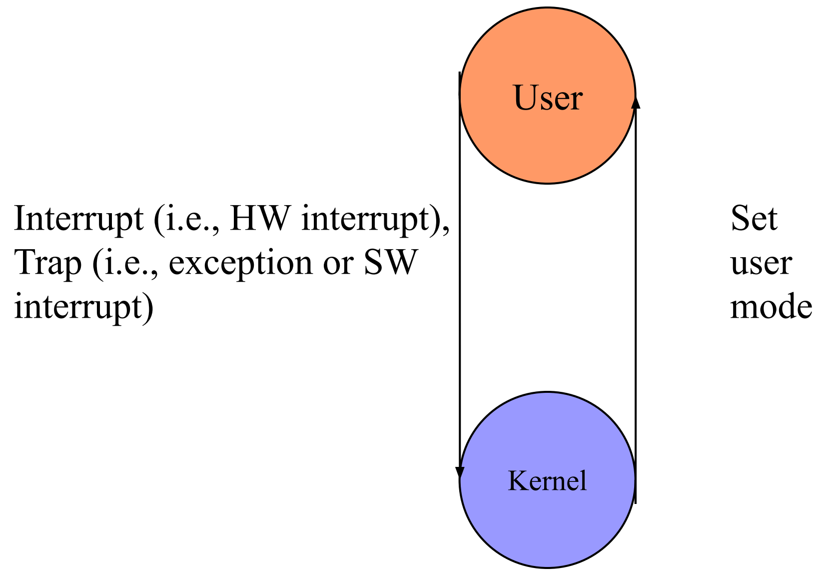
- No
  - Can alter critical system configurations and violate permissions
    - e.g., instructions to alter memory address spaces
    - e.g., instructions to program I/O devices
- How to prevent?

Dual-mode operation

- Provide hardware support to differentiate between at least two modes of operation:
  1. User mode -- execution done on behalf of a user.
  2. Kernel mode (monitor/supervisor/system mode) -- execution done on behalf of operating system.
- "Privileged" instructions are only executable in the kernel mode
- Executing privileged instructions in the user mode "traps" into the kernel mode

Dual-mode operation(cont.)

- Mode bit added to computer hardware to indicate the current mode: kernel(0) or user(1).
- When an interrupt or trap occurs, hardware switches to kernel mode.



CPU Protection

- How to prevent a process from executing indefinitely?

CPU Protection

- Timer - interrupts computer after specified period to ensure that OS maintains control.
  - Timer is decremented every clock tick.
  - When timer reaches a value of 0, an interrupt occurs.
- Timer is commonly used to implement time sharing.
- Timer is also used to compute the current time.
- Should programming the timer require privileged instructions?

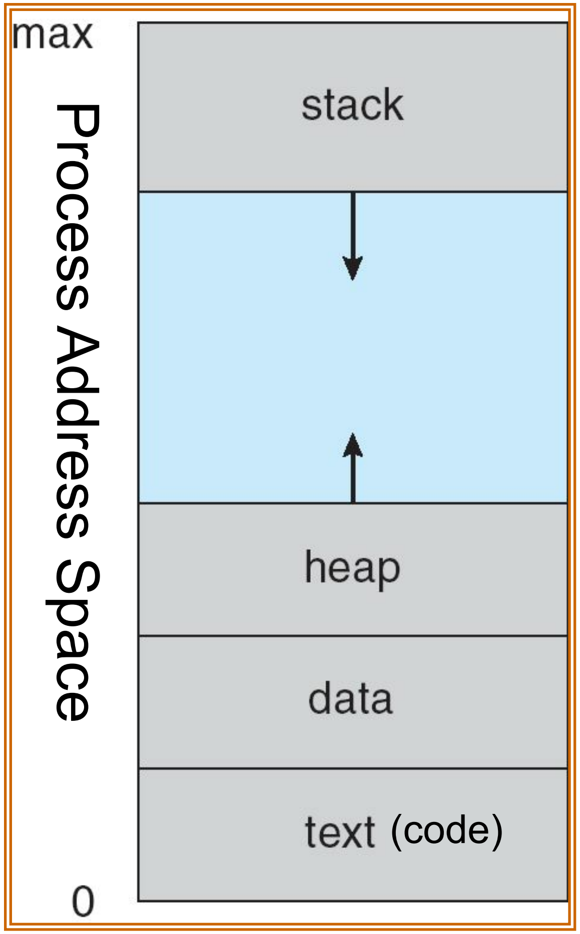
CPU Protection

- Timer - interrupts computer after specified period to ensure that OS maintains control.
  - Timer is decremented every clock tick.
  - When timer reaches a value of 0, an interrupt occurs.
- Timer is commonly used to implement time sharing.
- Timer is also used to compute the current time.
- Should programming the timer require privileged instructions? Yes!

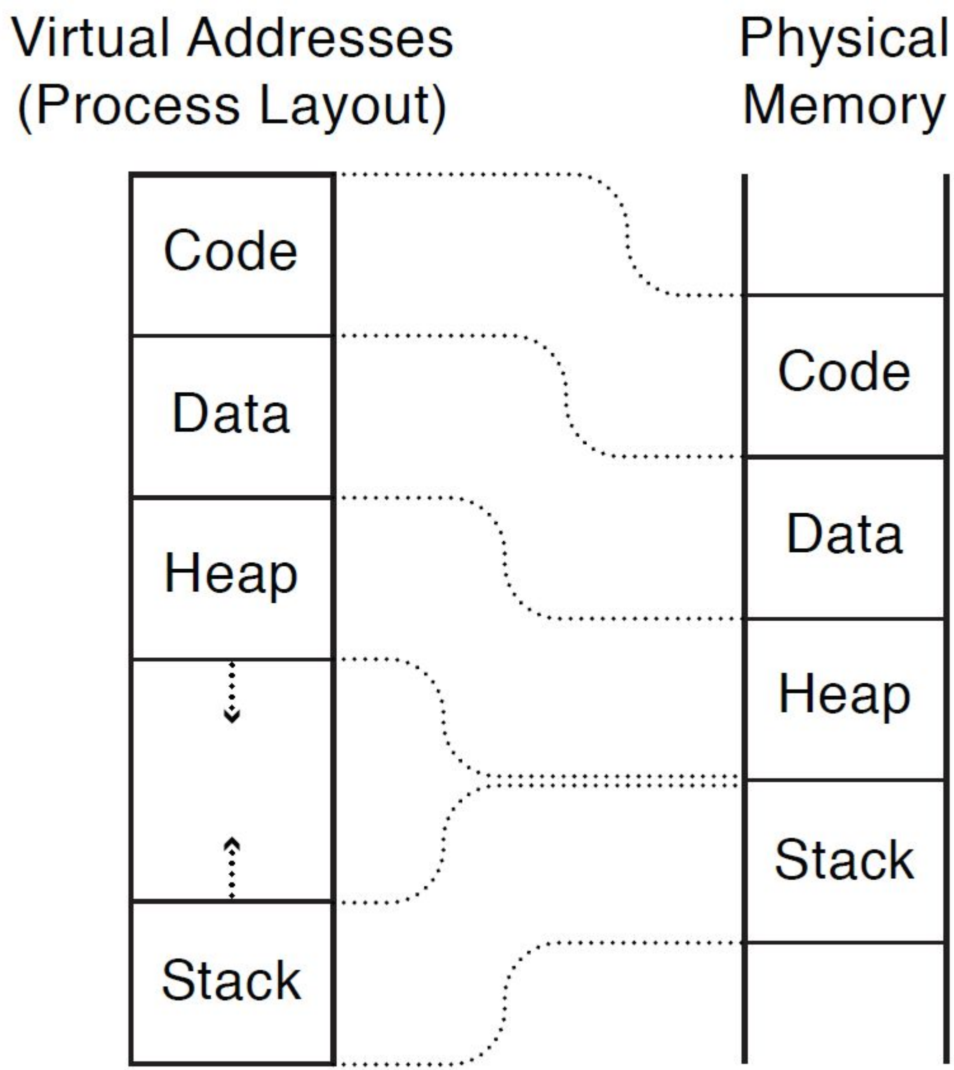
How to isolate memory access?

Process address space

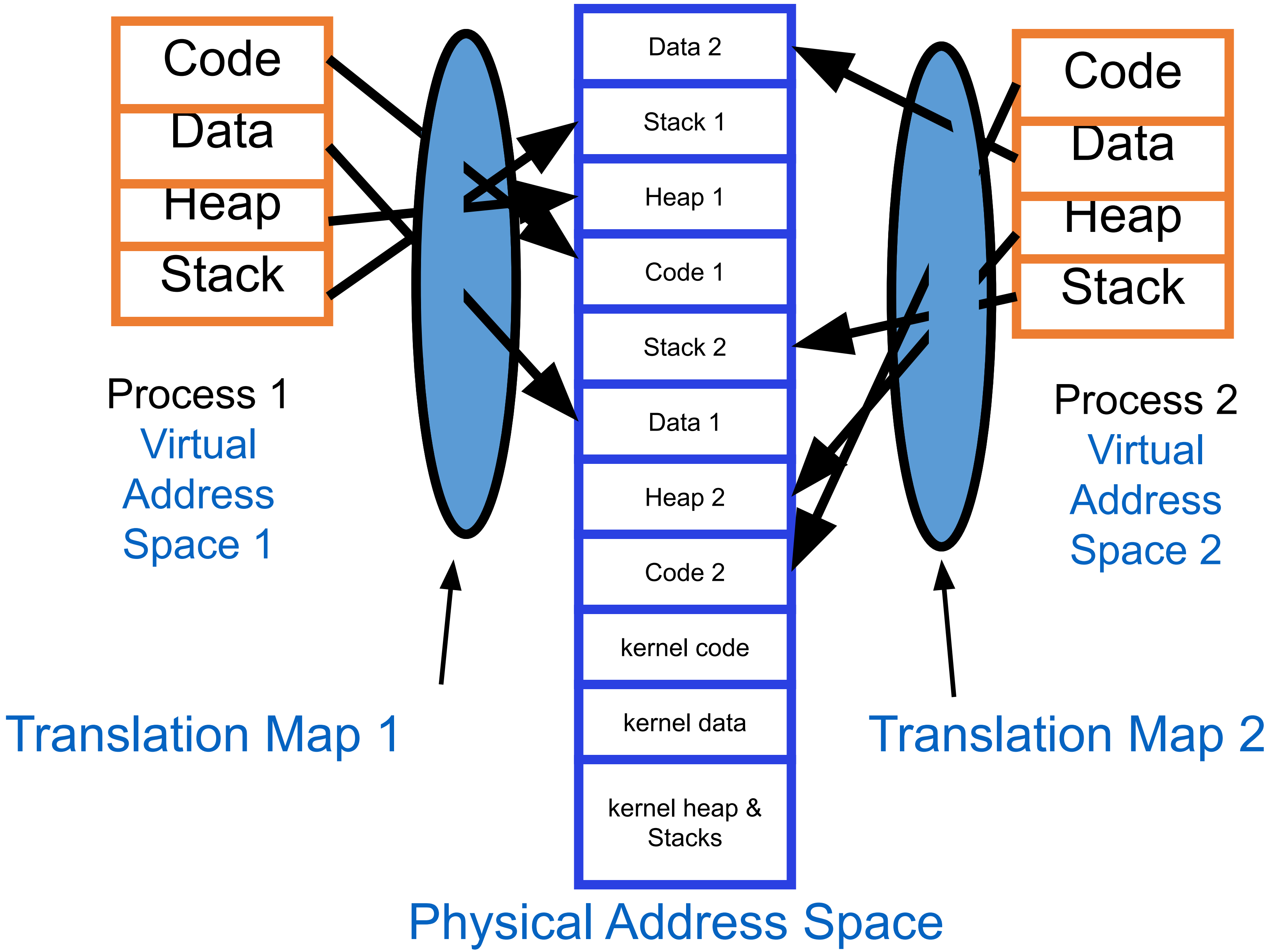
- Address space ⇒ the set of accessible addresses :
  - For a 32-bit processor there are 2^{32} = 4 billion addresses



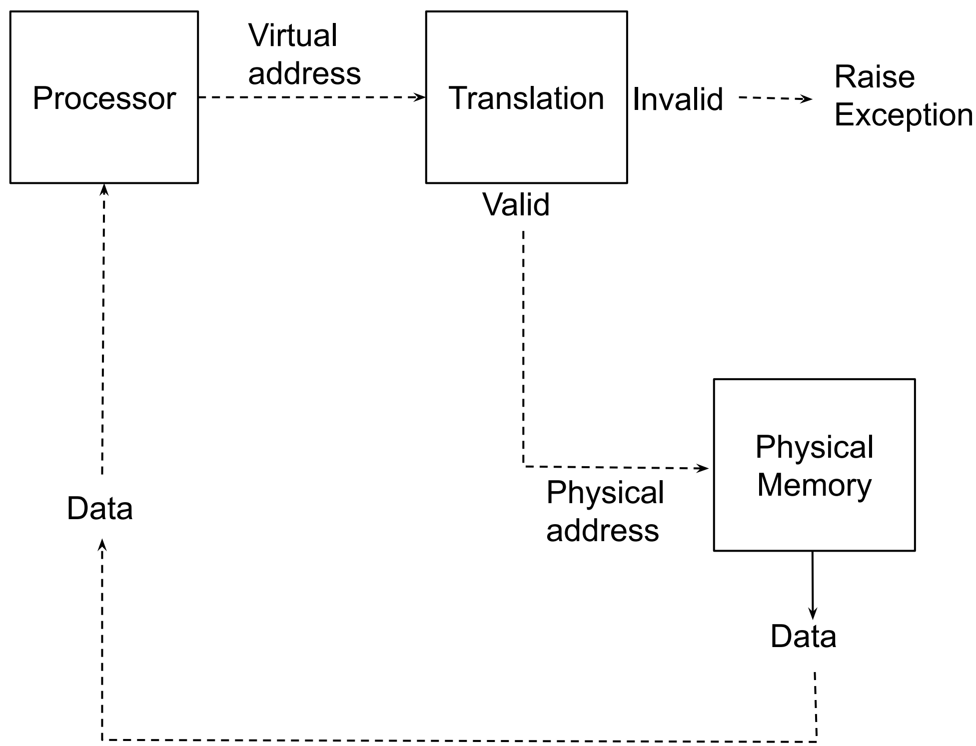
Virtual Address



Providing the Illusion of Separate Address Spaces



Address translation and memory protection

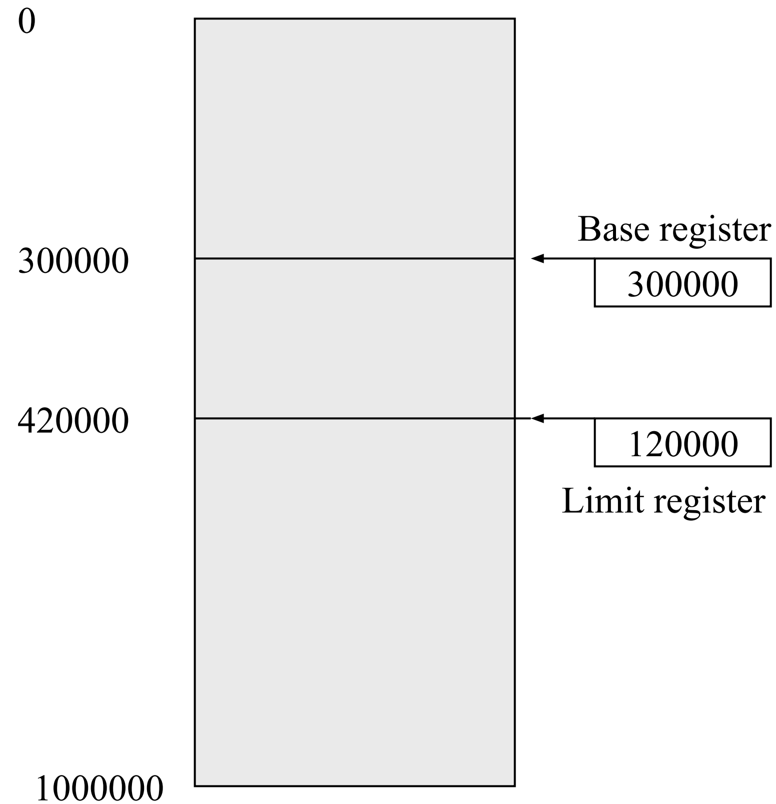


Memory Protection

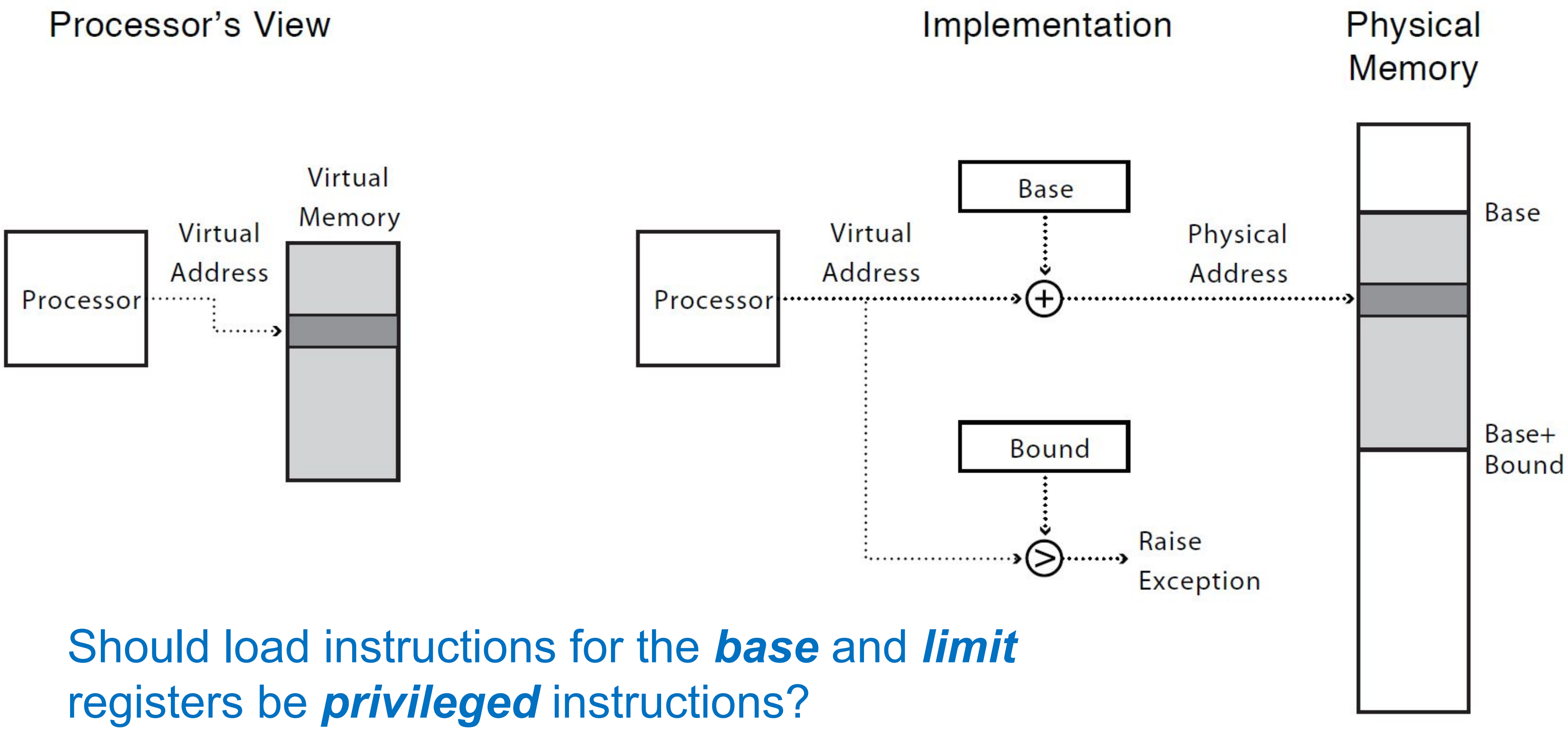
- When a process is running, only memory in that process address space must be accessible.
- When executing in kernel mode, the kernel has unrestricted access to all memory.

Memory Protection: base and bounds

- To provide memory protection, add two registers that determine the range of legal addresses a program may address.
    - Base Register - holds smallest legal physical memory address.
    - Limit register - contains the size of the range.
- Memory outside the defined range is protected.

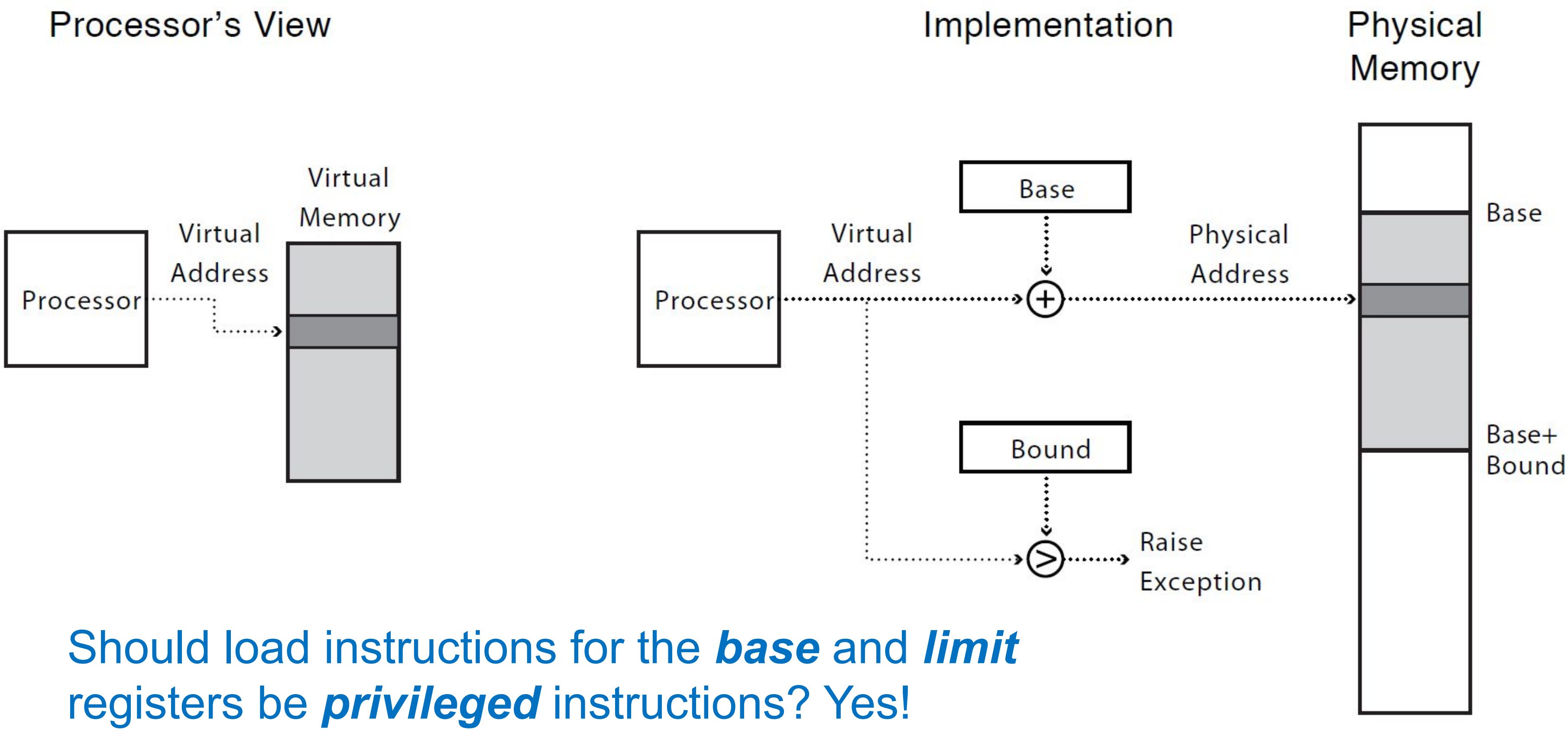


Virtual Address translation using the Base and Bounds method



Should load instructions for the ***base*** and ***limit*** registers be ***privileged*** instructions?

Virtual Address translation using the Base and Bounds method



Should load instructions for the ***base*** and ***limit*** registers be ***privileged*** instructions? Yes!

I/O Protection

- All I/O instructions are privileged instructions.

Question

- Given the I/O instructions are privileged, how do users perform I/O?

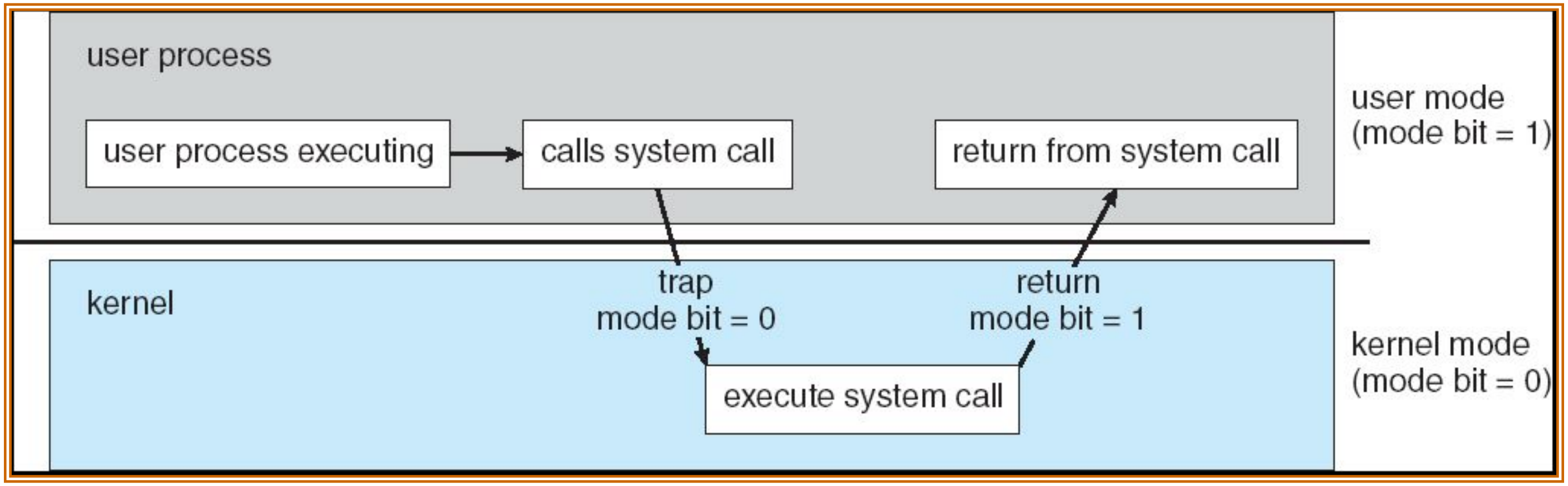
Question

- Given the I/O instructions are privileged, how do users perform I/O?

- Via system calls - the method used by a process to request action by the operating system.

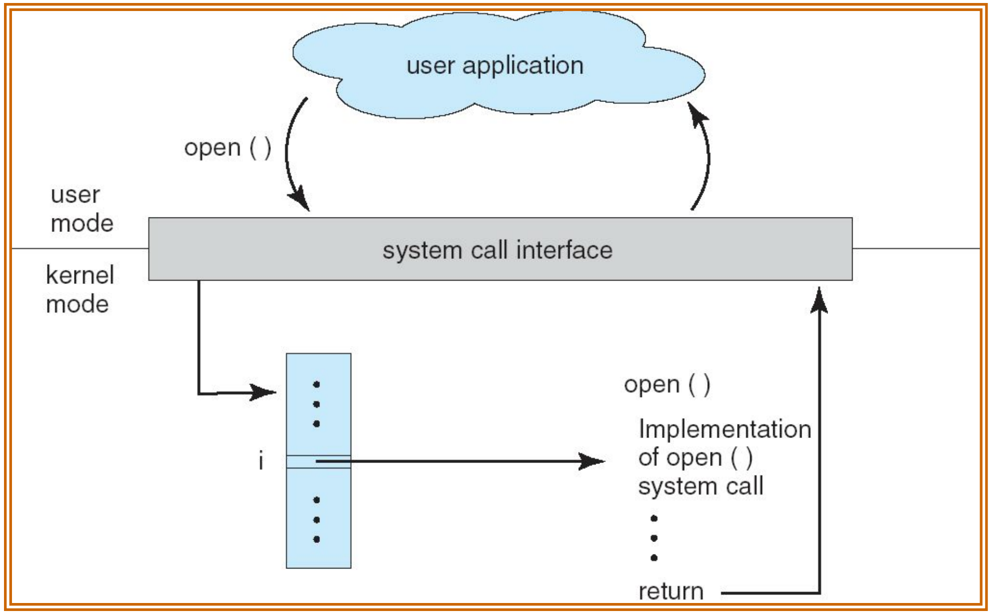
System Calls

- User code can issue a syscall, which causes a trap
- Kernel handles the syscall



System Calls

- Interface between applications and the kernel.
  - Application uses an assembly instruction to trap into the kernel
  - Some higher level languages provide wrappers for system calls (e.g., C)
- System calls pass parameters between an application and OS via registers or memory

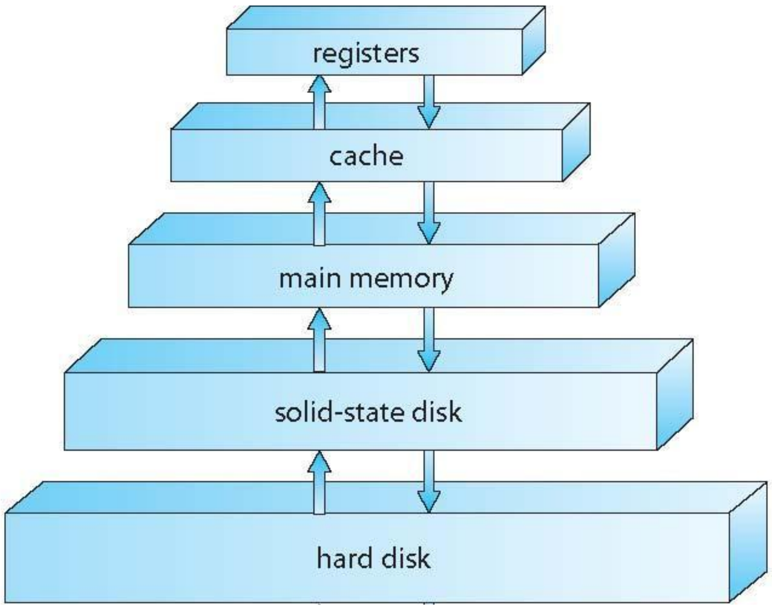


- Linux has about 300 system calls
  - read(), write(), open(), close(), fork(), exec(), ioctl(),…..

System services or system programs

- Components of the OS that provide help for program development and execution.
  - Command Interpreter (i.e., shell) - parses commands and executes other programs
  - Window management
  - System libraries, e.g., libc

Storage Device Hierarchy



Storage Structure

- Main memory - only large storage media that the CPU can access directly.
- Secondary storage - has large nonvolatile storage capacity.
  - Example: Magnetic disks - rigid metal or glass platters covered with magnetic recording material.
    - Disk surface is logically divided into tracks, subdivided into sectors.
    - Disk controller determines logical interaction between device and computer.

Storage Hierarchy

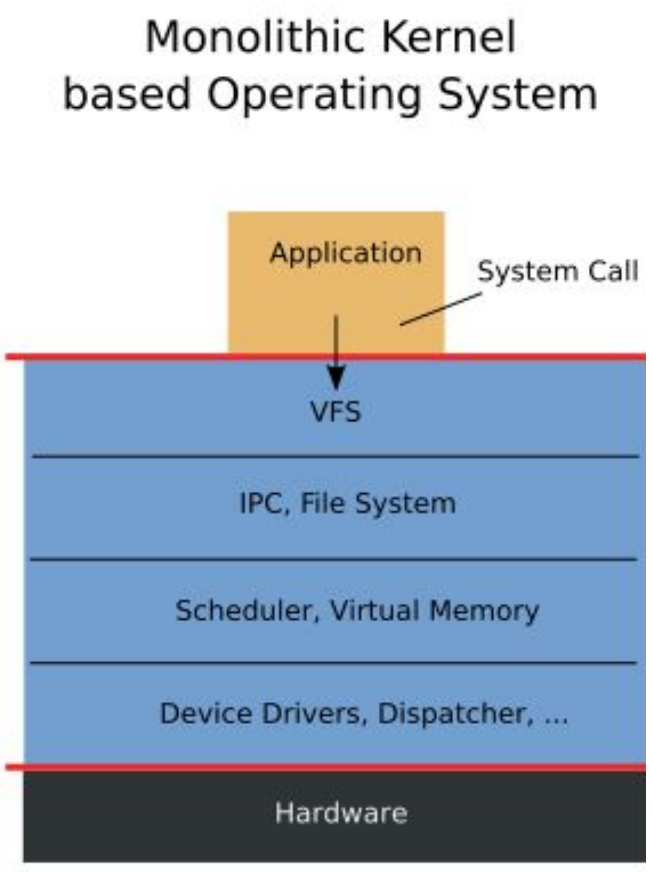
- Storage systems are organized in a hierarchy based on
  - Storage space
  - Access time
  - Cost
  - Volatility
- Caching - process of copying information into faster storage system; main memory can be viewed as fast cache for secondary storage.

Operating Systems: How are they organized?

Monolithic vs. Microkernel OS

- Monolithic OSes have large kernels with a lot of components
  - Linux, Windows, Mac

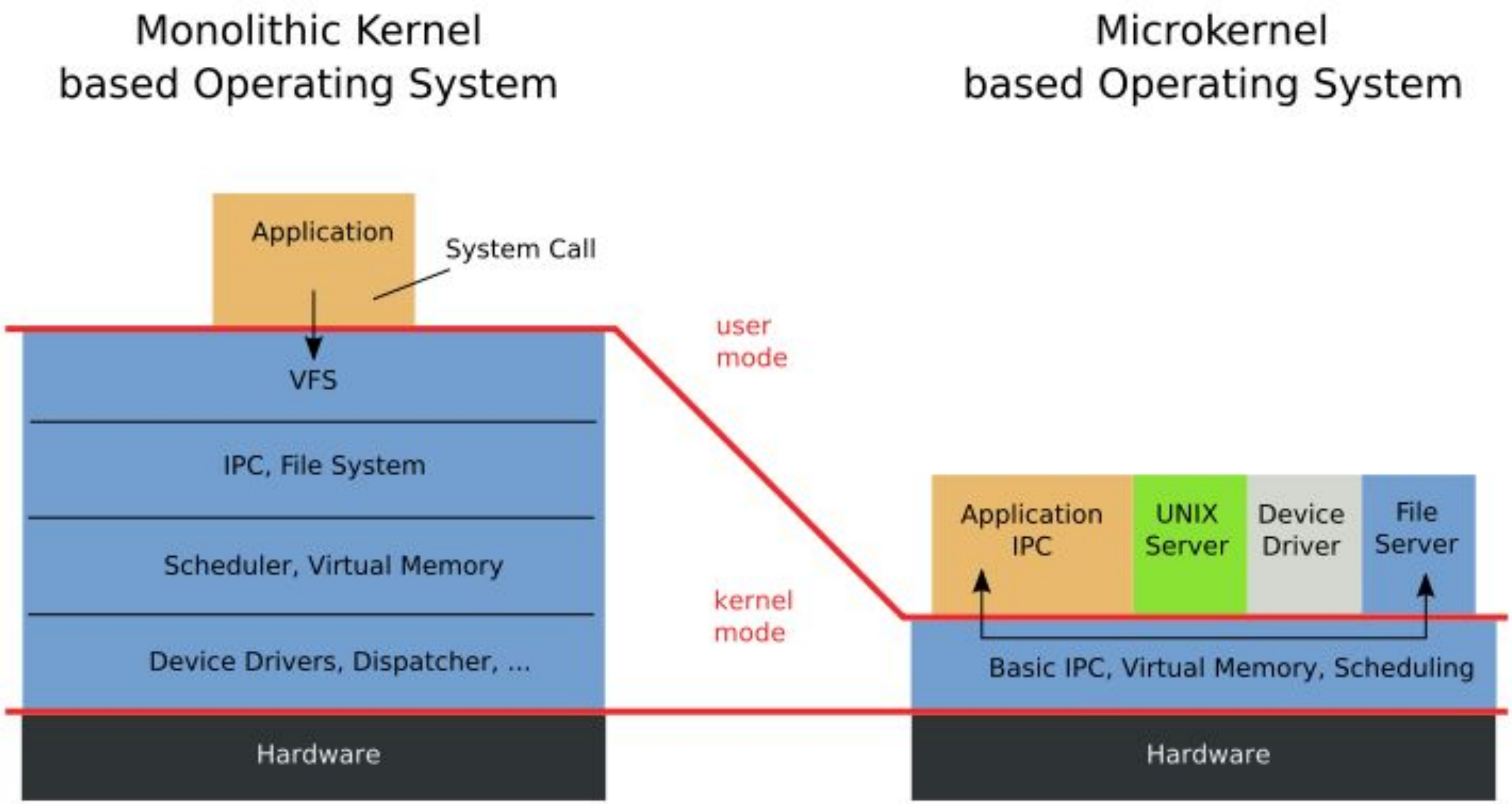
Monolithic vs. Microkernel OS



Monolithic vs. Microkernel OS

- Monolithic OSes have large kernels with a lot of components
  - Linux, Windows, Mac
- Microkernels moves as much from the kernel into "*user*" space
  - Small core OS components running at kernel level
  - OS Services built from many independent user-level processes
  - Communication between modules with message passing
  - Benefits:
    - Easier to extend a microkernel
    - Easier to port OS to new architectures
    - More reliable and more secure (less code is running in kernel mode)
  - Detriments:
    - Performance overhead severe for naïve implementation

Monolithic vs. Microkernel OS



Slide adapted from http://web.cecs.pdx.edu/~walpole/class/cs533/fall2015/home.html

OS Task: Process Management

- Process - fundamental concept in OS
  - Process is an instance of a program in execution.
  - Process needs resources - CPU time, memory, files/data and I/O devices.
- OS is responsible for the following process management activities.
  - Process creation and deletion
  - Process suspension and resumption
  - Process synchronization and interprocess communication
  - Process interactions - deadlock detection, avoidance and correction

OS Task: Memory Management

- Main Memory is an array of addressable words or bytes.
- Main Memory is volatile.
- OS is responsible for:
  - Allocate and deallocate memory to processes.
  - Manage multiple processes within memory - keep track of which parts of memory are used by which processes. Manage the sharing of memory between processes.
  - Determining which processes to load when memory becomes available.

OS Task: Secondary Storage and I/O Management

- Since primary storage (i.e., main memory) is expensive and volatile, secondary storage is required for backup.
- Disk is the primary form of secondary storage.
  - OS performs storage allocation, free-space management, etc.
- I/O subsystem in the OS consists of
  - Device driver interface that abstracts device details
  - Drivers for specific hardware devices

OS Task: File System Management

- File is a collection of related information - represents programs and data.

- OS is responsible for
  - File creation and deletion
  - Directory creation and deletion
  - Supporting primitives for file/directory manipulation.
  - Mapping files to disks (secondary storage).

OS Task: Protection and Security

- Protection mechanisms control access of processes to user and system resources.
- Protection mechanisms must:
  - Distinguish between authorized and unauthorized use.
  - Specify access controls to be imposed on use.
  - Provide mechanisms for enforcement of access control.

Summary of this week's lecture

- What is an operating system?
- Operating systems history
- Computer system and operating system structure