Operating Systems Concepts

Operating Systems Structures

CS 4375, Fall 2025

Instructor: MD Armanuzzaman (***Arman***)

marmanuzzaman@utep.edu

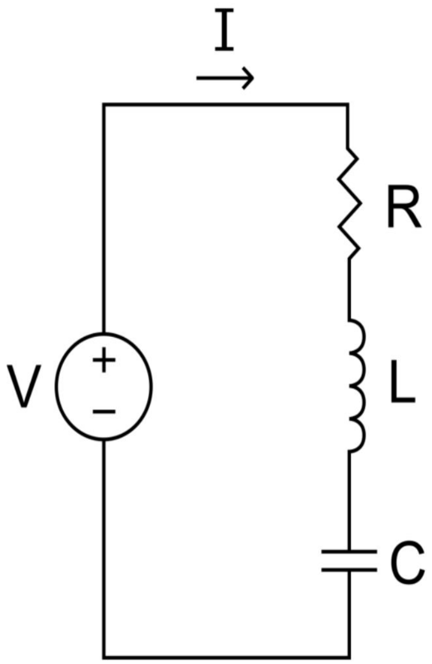
August 27, 2025

Agenda

- A very brief history of computer architecture
- What is an OS?
- What does an OS do?
- The goals for an OS
- Major OS components
- OS design approaches

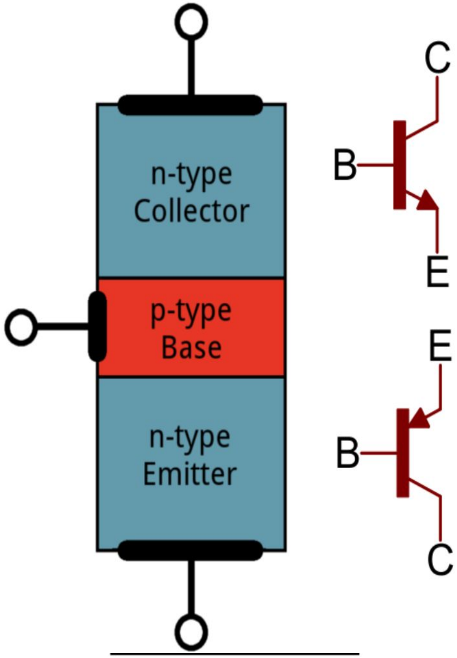
Computer Architecture

- Electronics



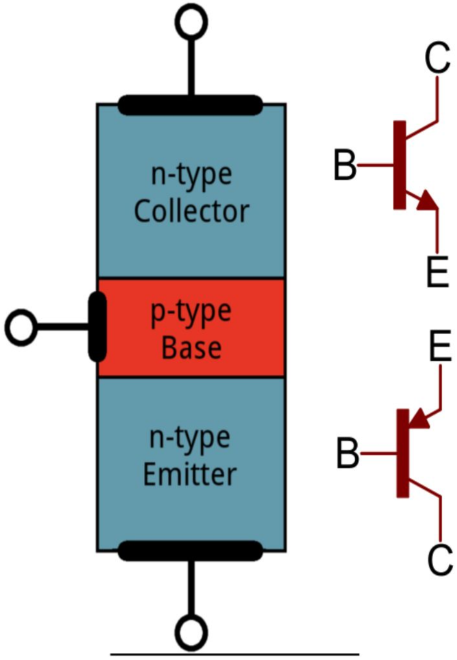
Computer Architecture

- Electronics
- Transistors



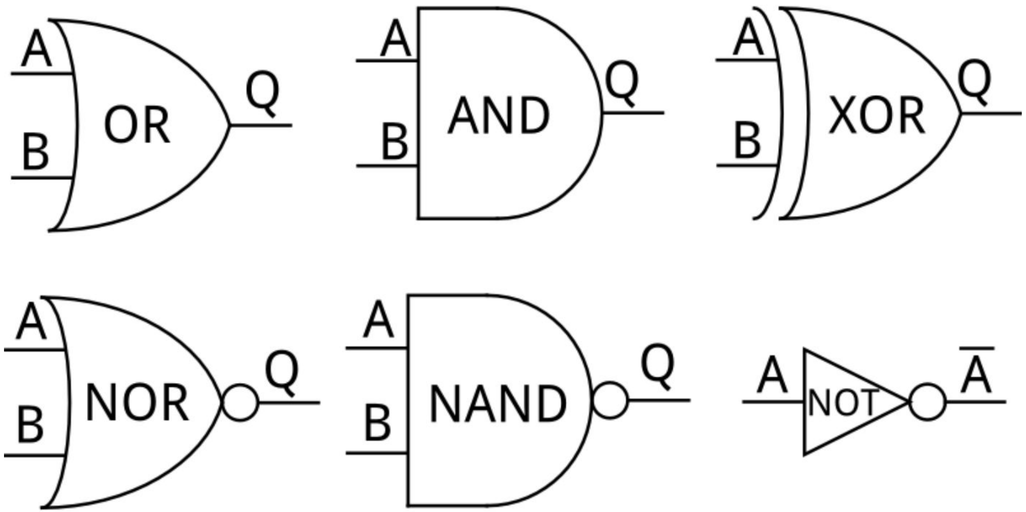
Computer Architecture

- Electronics
- Transistors



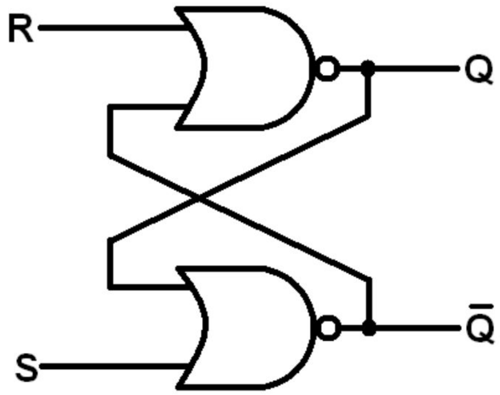
Computer Architecture

- Electronics
- Transistors
- Logic gates



Computer Architecture

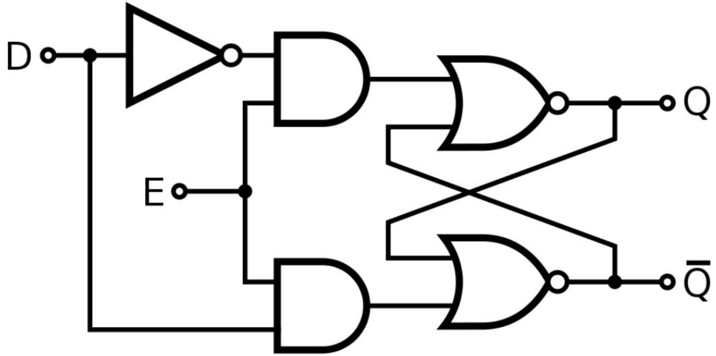
- Electronics
- Transistors
- Logic gates
- SR Latches (Set/Reset)



R	S	Q	Function
0	0	No Change	Latch remained in present state.
0	1	1	Latch SET
1	0	0	Latch RESET
1	1	0	Invalid Condition

Computer Architecture

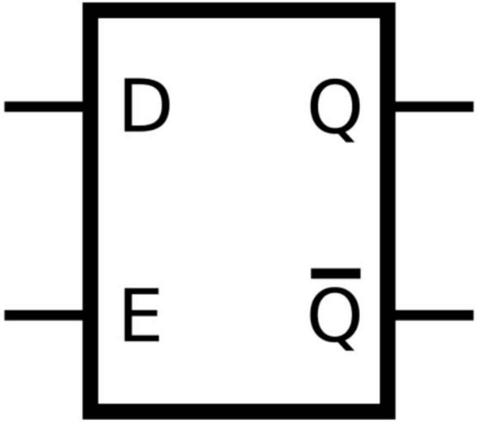
- Electronics
- Transistors
- Logic gates
- SR Latches (Set/Reset)’
- Gated D Latches (Data)



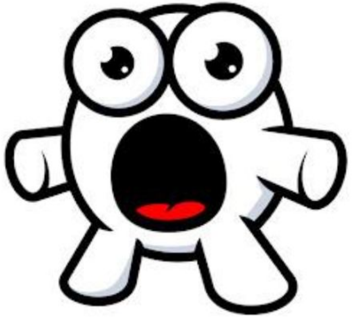
E	D	Q	Function
0	0	No Change	Latch remained in present state.
0	1	No Change	Latch remained in present state.
1	0	0	Latch RESET
1	1	1	Latch SET

Computer Architecture

- Electronics
- Transistors
- Logic gates
- SR Latches (Set/Reset)’
- Gated D Latches (Data)

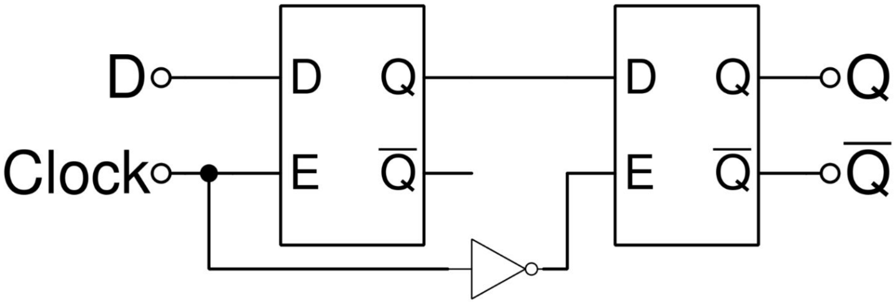


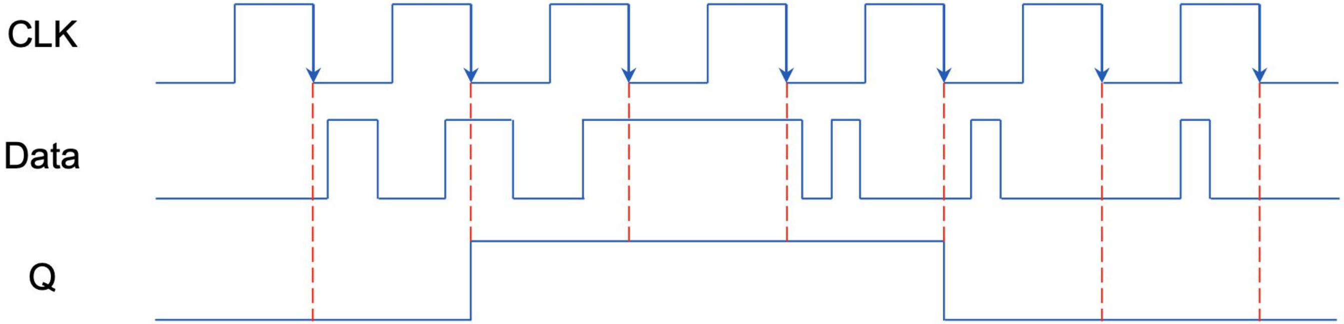
- We have a memory cell!!



Computer Architecture

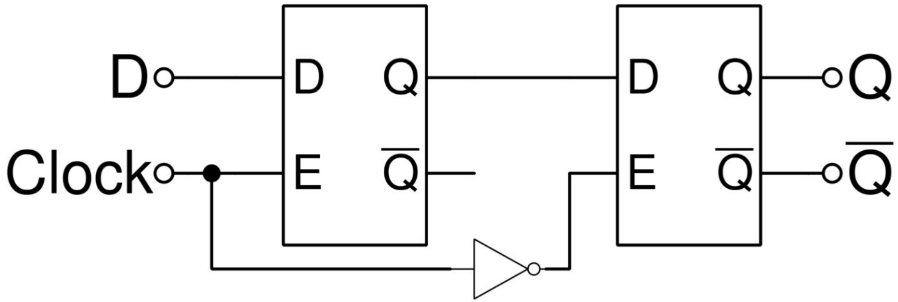
- Electronics
- Transistors
- Logic gates
- SR Latches (Set/Reset)’
- Gated D Latches (Data)
- D Flip-Flop (Data/Delay)



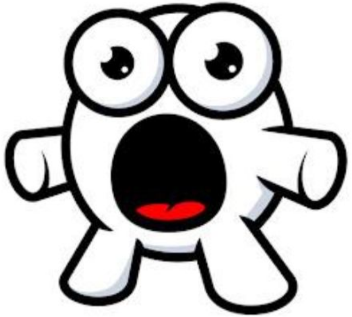


Computer Architecture

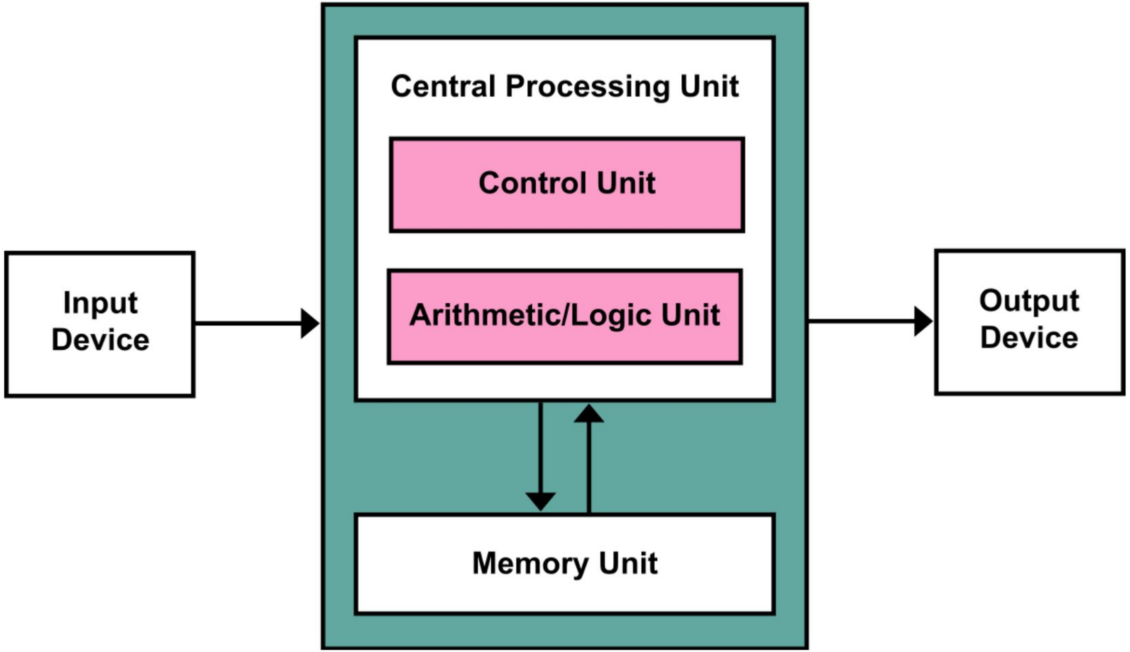
- Electronics
- Transistors
- Logic gates
- SR Latches (Set/Reset)'
- Gated D Latches (Data)
- D Flip-Flop (Data/Delay)



- We have a controlled flow of data!!!

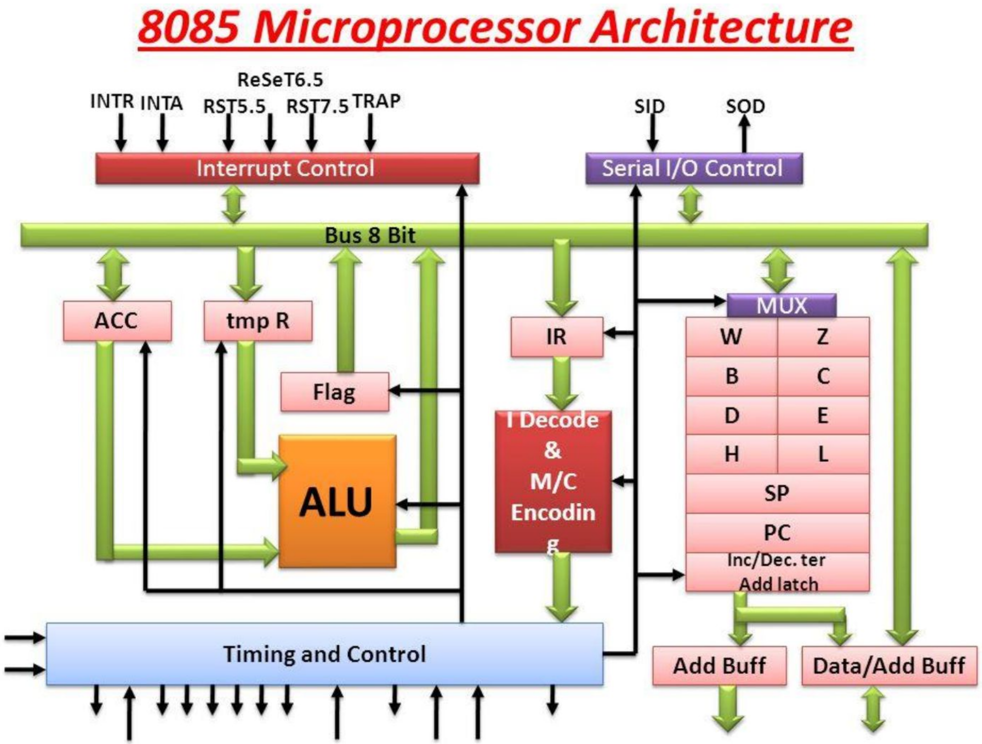


Von Neumann Architecture



- From: https://en.wikipedia.org/wiki/Von_Neumann_architecture

Intel 8085



- From: https://slideplayer.com/slide/2817140/

Some of 8085 Important Components

- Program Counter
- Instruction Register and Decoder
- ALU
- Flag Register
- Registers
- Stack Pointer
- Interrupt Controller
    - Timer (Hint!)
    - Communication
    - TRAP
    - Software Interrupt

Execution Steps Simplified

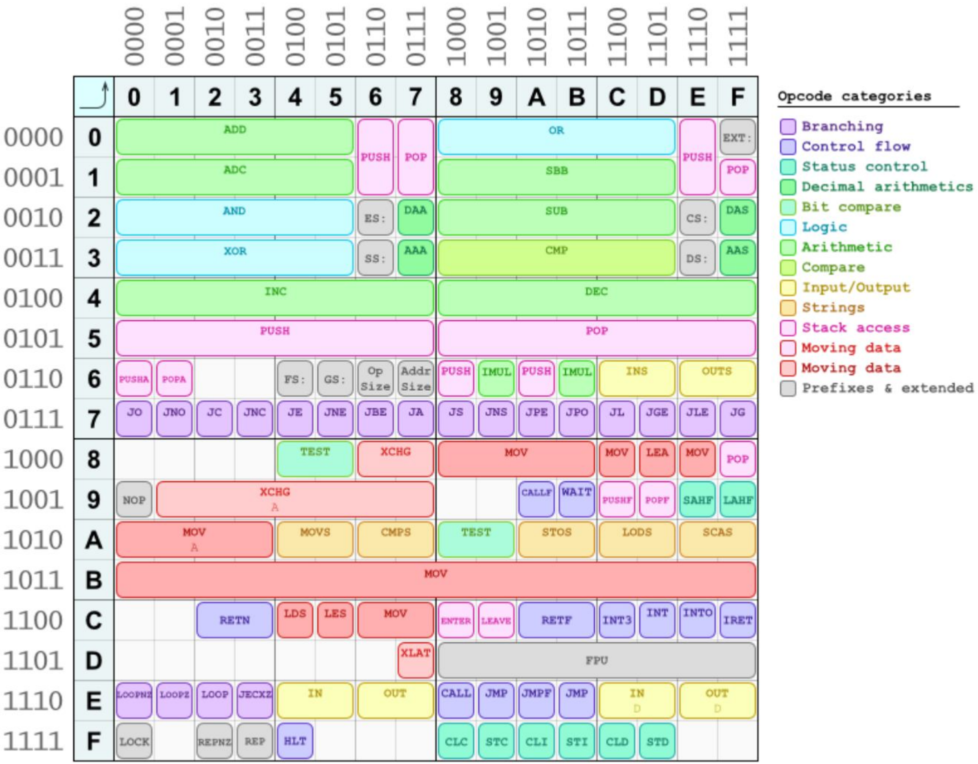
- Fetch instruction at PC
- Decode the instruction
- Execute (possibly using registers and modifying them)
- Write possible result to registers/memory
- Increase PC to point to the next instruction

REPEAT!

Execution Steps

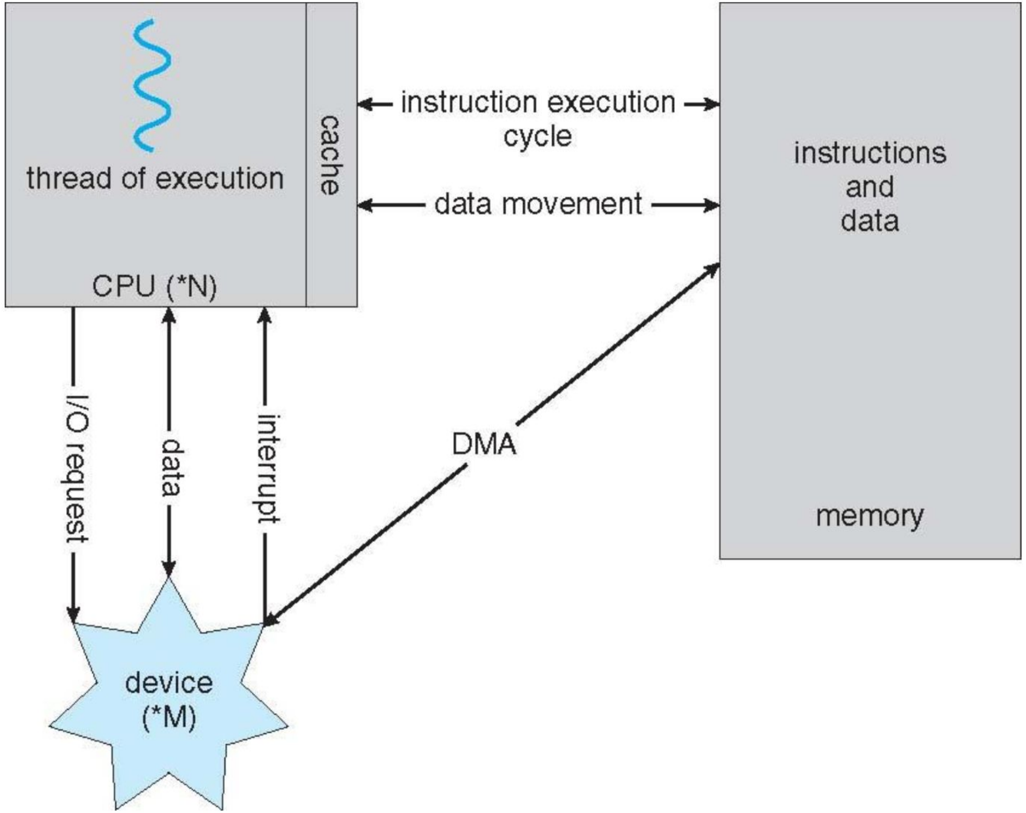
- From: https://www.vectorstock.com/royalty-free-vector/manufacture-interior-with-vector-23241154

Instruction Set



- From: https://stackoverflow.com/questions/6924912/finding-number-of-operands-in-an-instruction-from-opcodes

Modern Computer Architecture



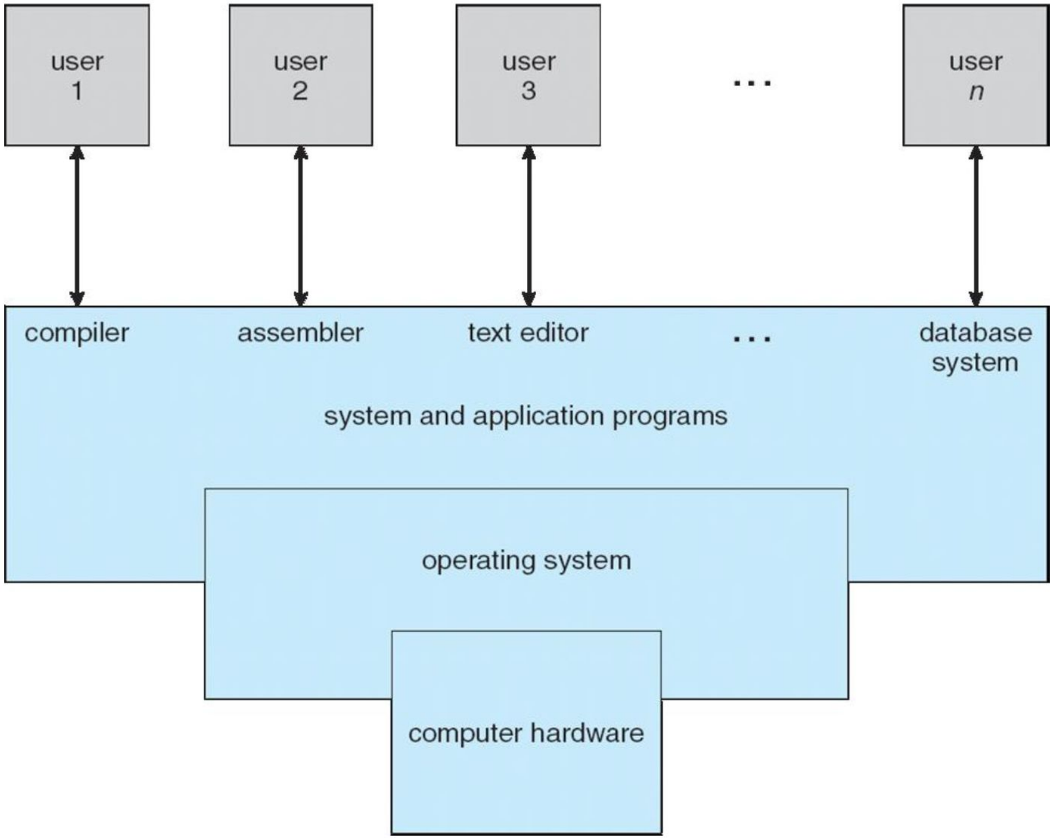
Limitations

- Only a single user at a time
  - Who manages?
- Each user will need to be independent, and have full stack software
  - Drivers, all of them!
  - Compilers
  - I/O
  - etc.
- Each user have full access!
  - Protection and Security

What is an Operating System?

- It is a program!
    - A big one - Linux source code is 25+ Million lines of C code!
- OS manages the computer hardware resources
    - Manages all resources
    - Decides between conflicting requests for efficient and fair resource use
- OS is a control program
    - Controls execution of programs to prevent errors and improper use of the computer
- Acts as an intermediary between a user of a computer and the computer hardware

What is an Operating System?



Operating System Goals

- From the user perspective:
  - Executes user programs and make solving user problems easier
  - Makes the computer system convenient to use
  - Hides the messy details which must be performed
  - Presents user with a virtual machine easier to use
- From the system/HW perspective:
  - Manages the resources
  - Uses the computer hardware in an efficient manner
  - Time sharing: each program gets some time to use a resource
  - Resource sharing: each program gets a portion of a resource

What is kernel?

- No universally accepted definition
- Everything a vendor ships when you order an operating system is a good approximation (But varies wildly!)
- “The one program running at all times on the computer” is the kernel. Everything else is either a system program (ships with the operating system), or an application program.

Early Operating Systems: Serial Operations

- One application at a time
    - Had complete control of hardware
    - OS was runtime library
    - Users would stand in line to use the computer
- Batch systems
    - Keep CPU busy by having queue of jobs
    - OS would load next job while current one runs
    - Users would submit jobs, and wait, and wait...

Time-Sharing Operating Systems

- Multiple users on computer at same time
- Interactive performance: try to complete everyone’s tasks quickly
- As computers became cheaper, it became more important to optimize for user time, not computer time.

Computer Performance Over

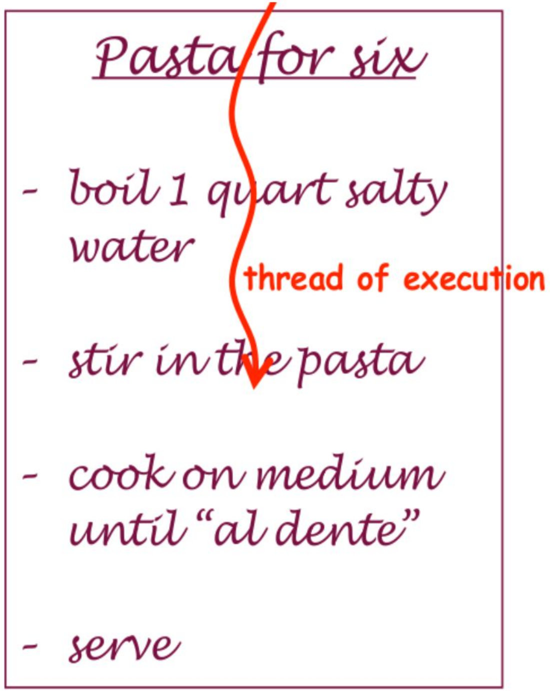
	1981	1997	2014	Factor (2014/1981)
Uniprocessor speed (MIPS)	1	200	2500	2.5K
CPUs per computer	1	1	10+	10+
Processor MIPS/$	$100K	$25	$0.20	500K
DRAM Capacity (MiB)/$	0.002	2	1K	500K
Disk Capacity (GiB)/$	0.003	7	25K	10M
Home Internet	300 bps	256 Kbps	20 Mbps	100K
Machine room network	10 Mbps (shared)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

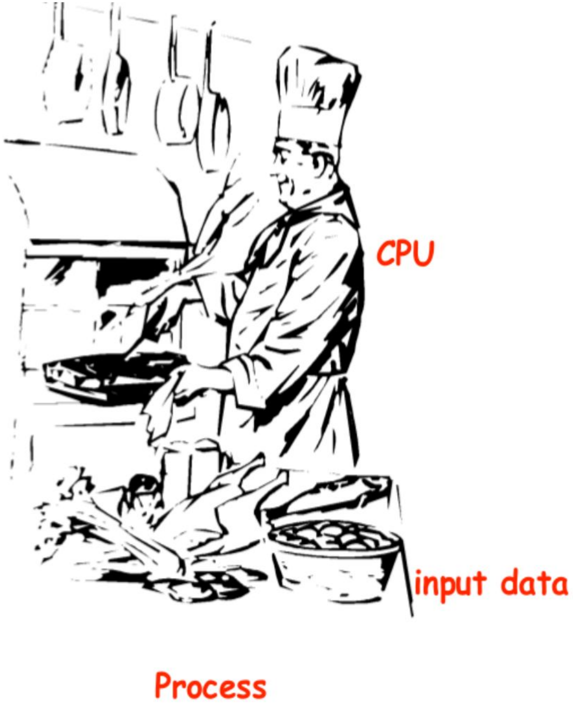
Major OS Components

- Processes and Threads
- CPU Scheduling
- I/O Management
- Memory Management

Processes and Threads

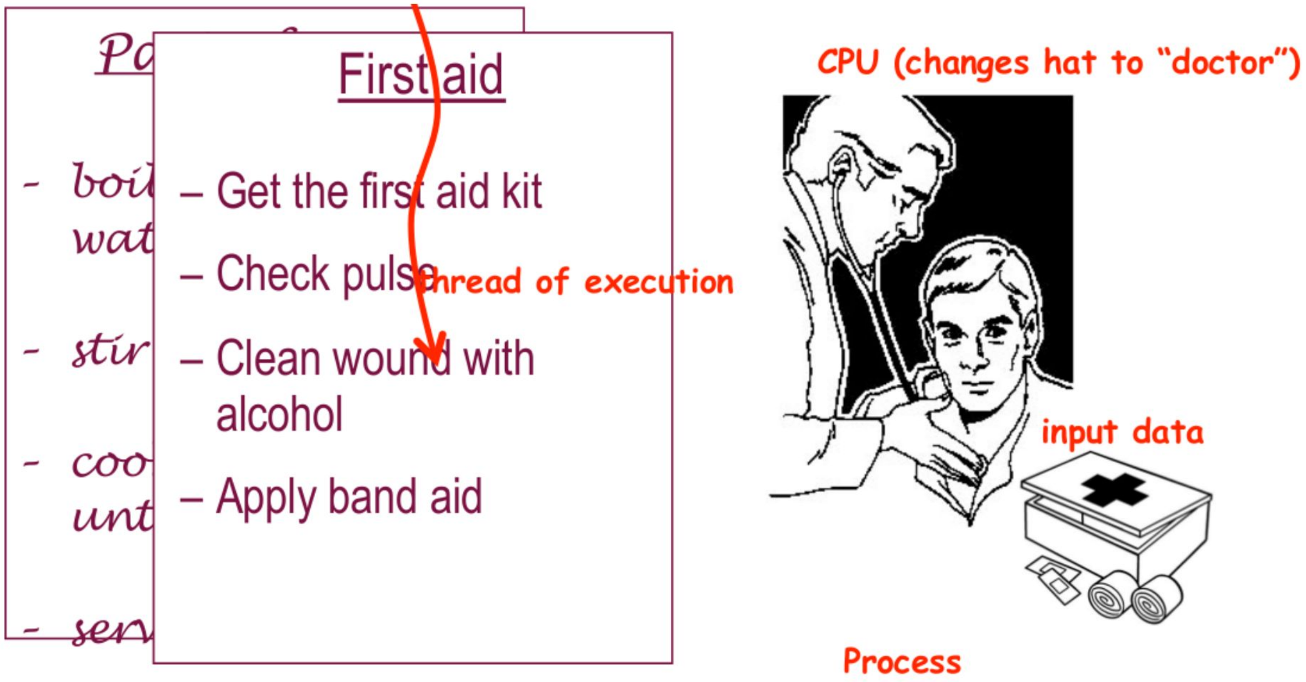
- A process is a program in execution.





Processes and Threads

- It can be interrupted to let the CPU execute a higher-priority process.



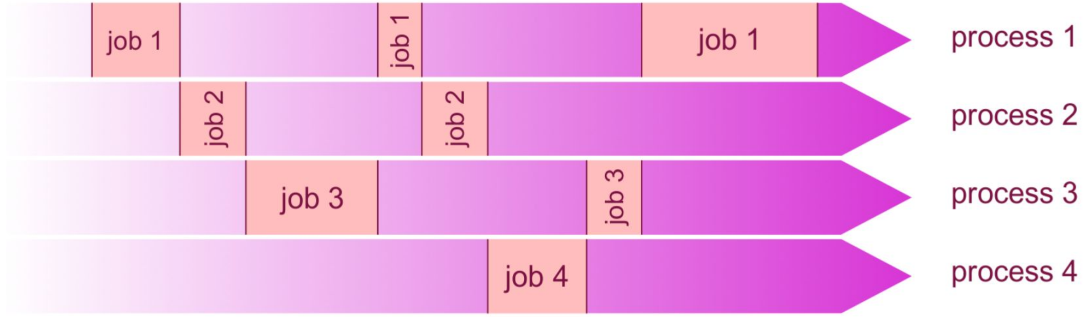
Processes and Threads

- … and then resume exactly where the CPU left off.

Multitasking/Time-sharing

- Multitasking gives the illusion of parallel processing on one CPU.
- Time-sharing is a logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, creating interactive computing.

(a) Multitasking from the CPU's viewpoint



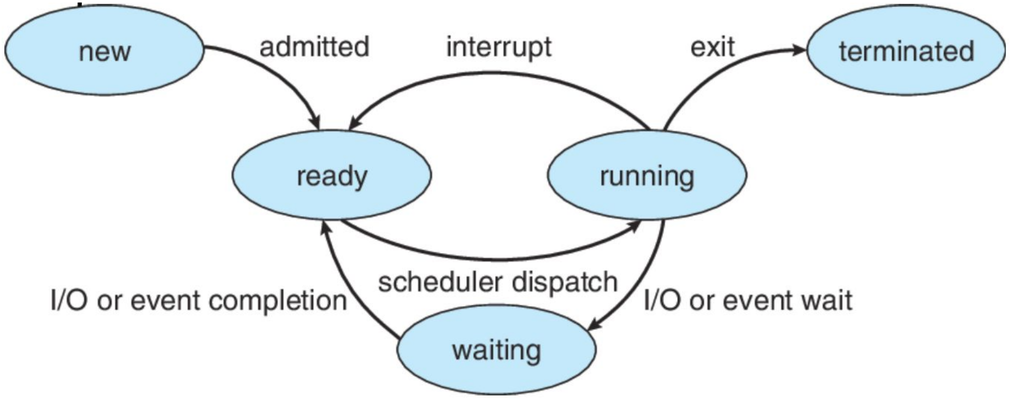
(b) Multitasking from the processes' viewpoint = 4 virtual program counters

Processes and Threads: OS Responsibilities

- The OS creates and deletes processes and threads.
- The OS suspends and resumes processes and threads.
- The OS schedules processes and threads.
- The OS provides mechanisms for process synchronization.
- The OS provides mechanisms for interprocess communication.
- The OS provides mechanisms for deadlock handling.

CPU Scheduling

- A process changes its state during execution:
    - **New**: The process is being created
    - **Ready**: The process is waiting to be assigned to a processor
    - **Running**: Instructions are being executed
    - **Waiting**: The process is waiting for some event to occur
    - **Terminated**: The process has finished execution

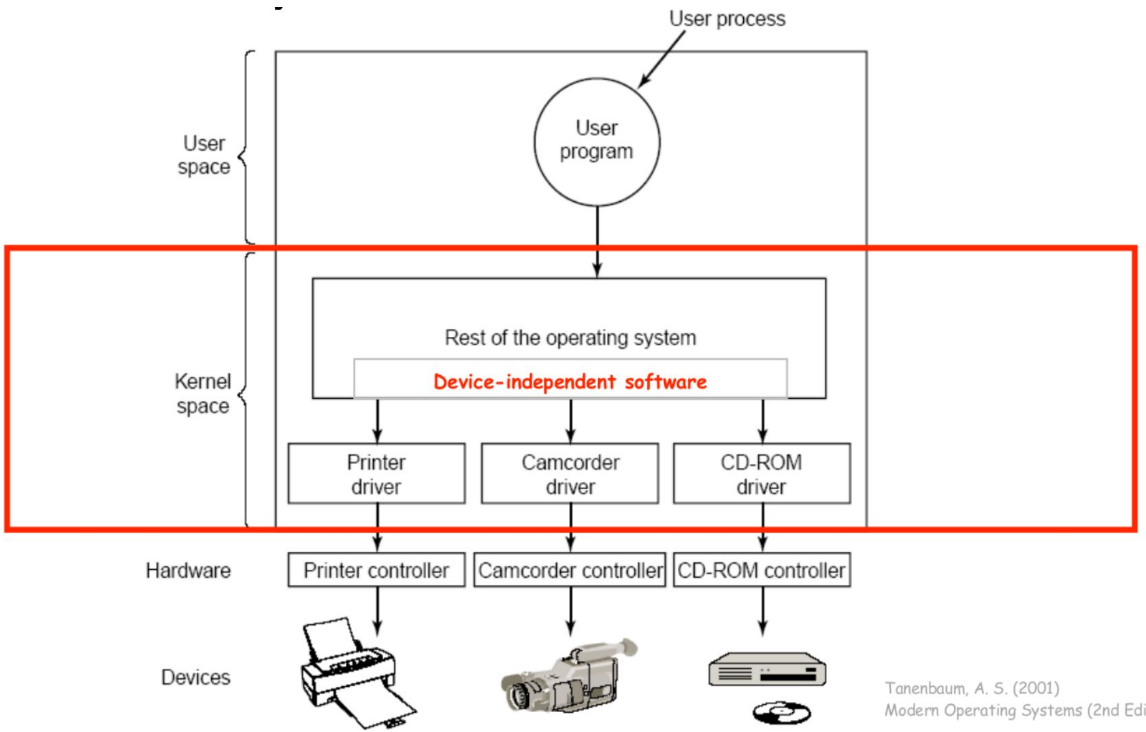


CPU Scheduling: OS Responsibilities

- The OS decides which available processes in memory are to be executed by the processor.
- The OS decides what process is executed when and for how long.
- The OS handles external events such as I/O interrupts, possibly changing the schedule flow.
- OS relies on a scheduling algorithm that attempts to optimize CPU utilization, throughput, latency, and/or response time, depending on the system requirements.

I/O Management

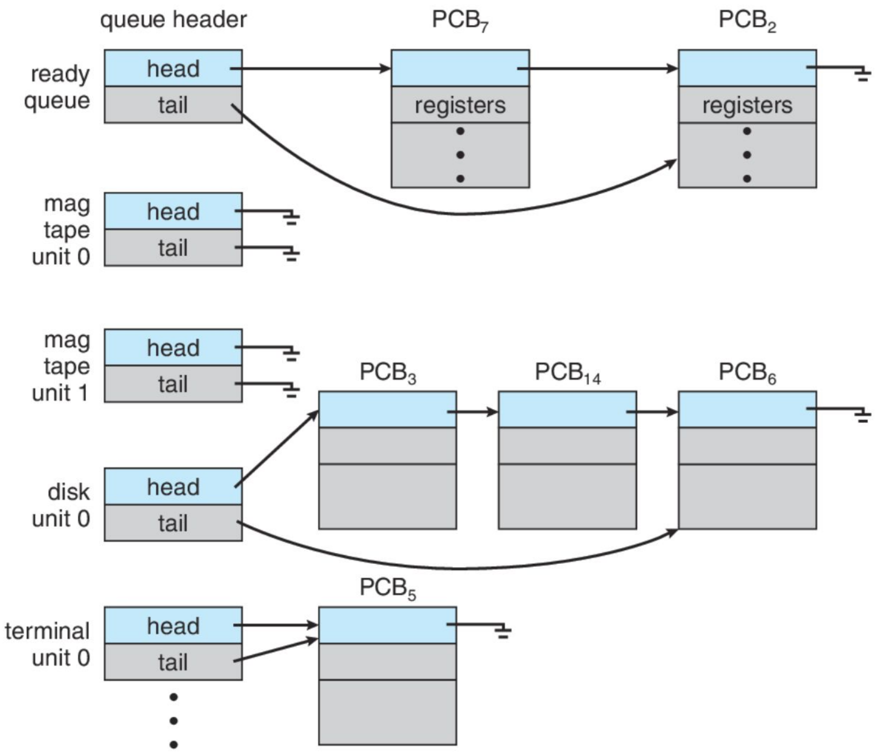
- Layers of the I/O subsystem



Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

I/O Management

- I/O Device Queues



I/O Management: Two Methods

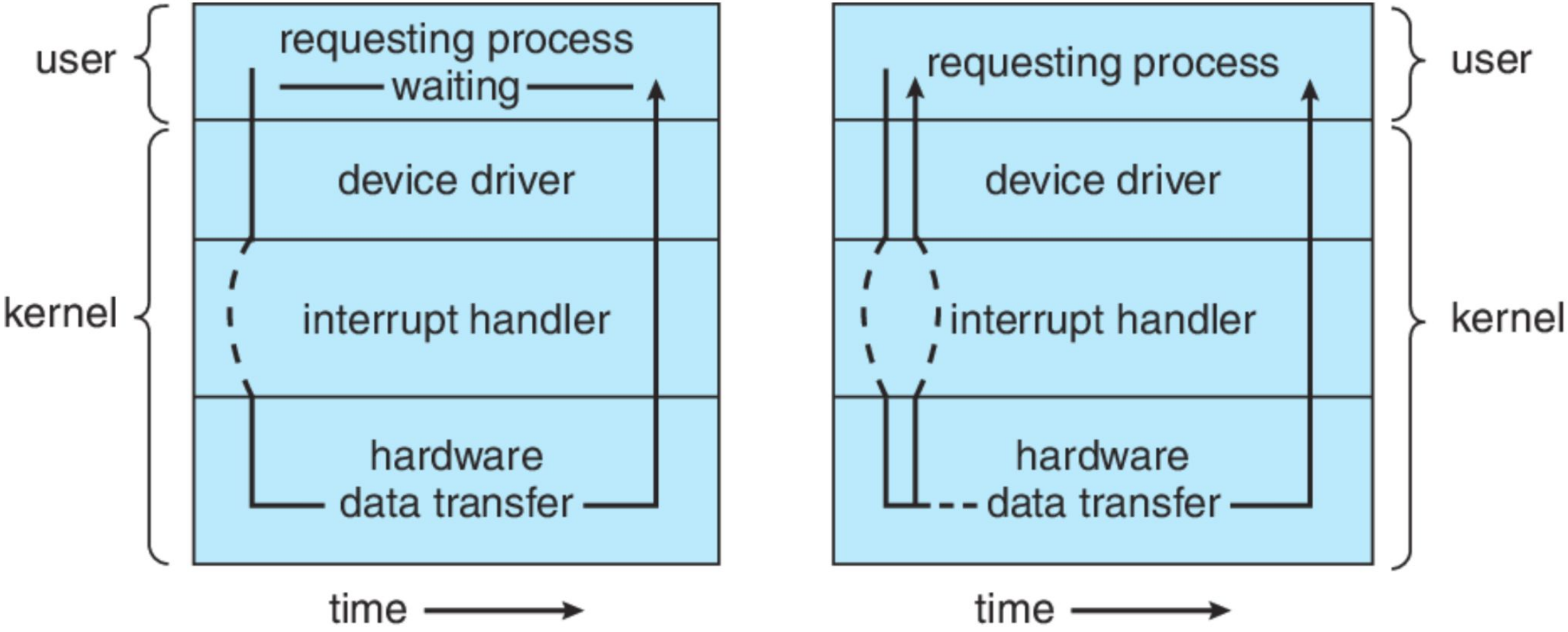
Synchronous

- Control returns to the user program only upon I/O completion
- Wait loop until next interrupt
- At most one I/O request is outstanding at a time, no simultaneous I/O processing for the process

Asynchronous

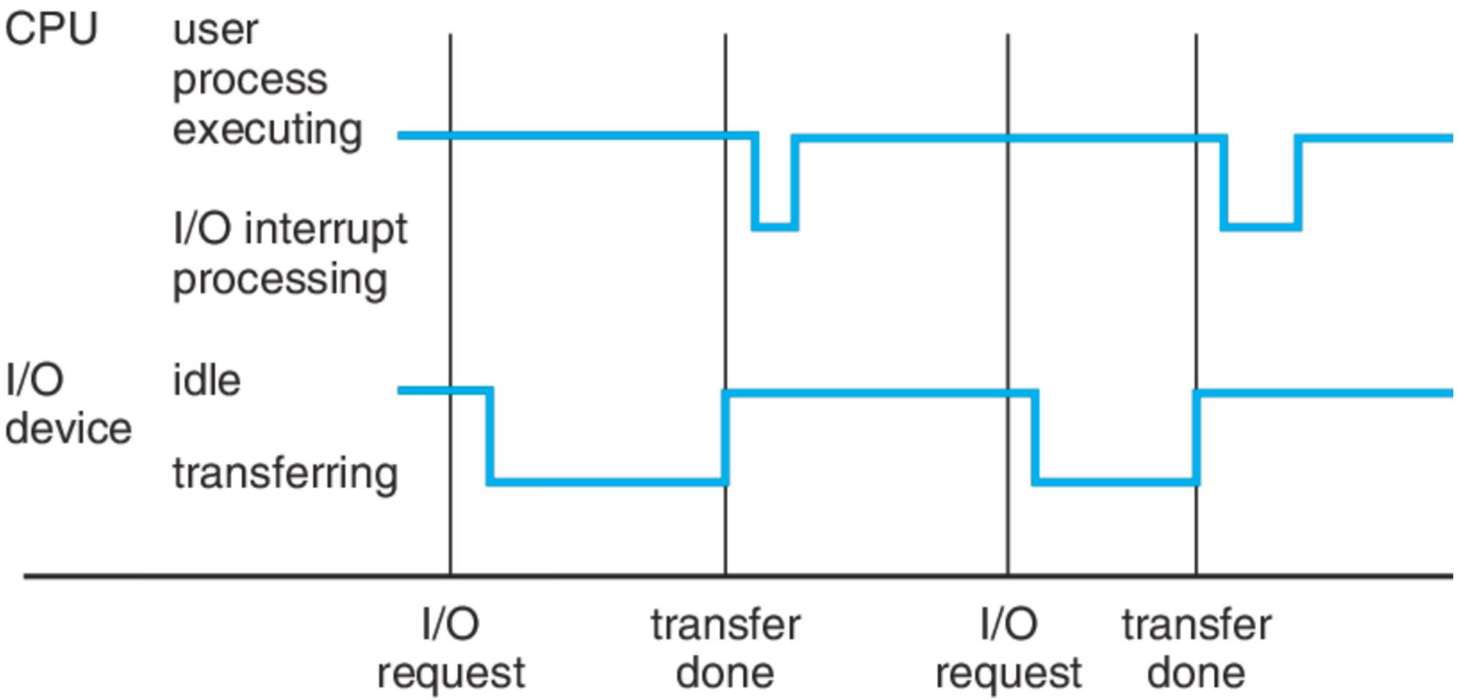
- Control returns to user program without waiting for I/O completion
- Operating system communicates the completion of request through a signal, call-back, etc.
- Process can run while the I/O gets finished

I/O Management: Two Methods

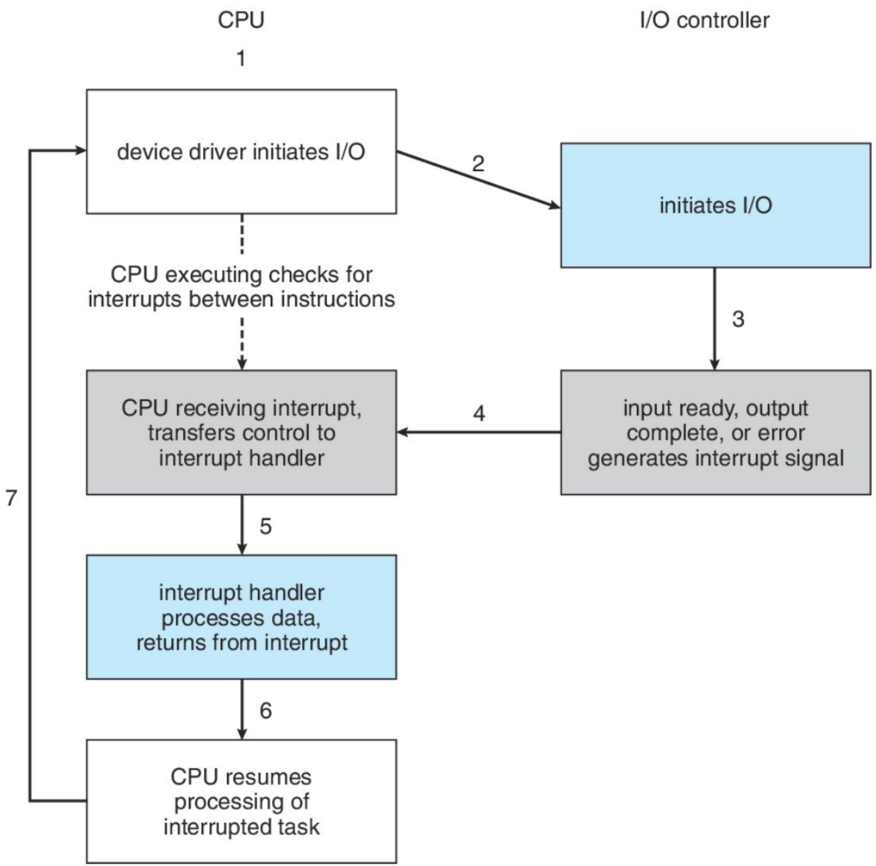


I/O Management: Interrupt Handling

- Interrupts, interrupt the execution flow!



I/O Management: Interrupt Handling



I/O Management: OS Responsibilities

- The OS hides the peculiarities of specific hardware devices from the user.
- The OS issues the low-level commands to the devices, catches interrupts and handles errors.
- The OS relies on software modules called “device drivers”.
- The OS provides a device-independent API to the user programs, which includes buffering.

Memory Management

- Memory needs to be subdivided to accommodate multiple processes
- Memory management is an optimization task under constraints
- Movement between levels of storage hierarchy:

Level	1	2	3	4	5
Name	registers	cache	main memory	solid state disk	magnetic disk
Typical size	< 1 KB	< 16MB	< 64GB	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS SRAM	flash memory	magnetic disk
Access time (ns)	0.25 - 0.5	0.5 - 25	80 - 250	25,000 - 50,000	5,000,000
Bandwidth (MB/sec)	20,000 - 100,000	5,000 - 10,000	1,000 - 5,000	500	20 - 150
Managed by	compiler	hardware	operating system	operating system	operating system
Backed by	cache	main memory	disk	disk	disk or tape

Memory Management: OS Responsibilities

- The OS keeps track of which parts of memory are currently being used and by whom.
- The OS allocates and deallocates memory space as needed.
- The OS decides which processes to load or swap out.
- The OS ensures process isolation.
- The OS regulates how different processes and users can sometimes share the same portions of memory.
- The OS transfers data between main memory and disk and ensures long-term storage.

OS Design Approaches

- Start defining goals and specifications
- Affected by choice of hardware, type of system
  - Batch, time-shared, single user, multi user, distributed
- Goal perspective:
  - User goals: operating system should be convenient to use, easy to learn, reliable, safe, and fast.
  - System goals: operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient.

- No unique solution! A variety of operating systems!

OS Design Approaches: Policy vs Mechanism

- It is an important principle to separate policies and mechanisms:
  - Policy: What should be done?
  - Mechanism: How will it get done?

Example: to ensure CPU protection:

- Policy: each process gets a limited execution time.
- Mechanism: Use Timer construct

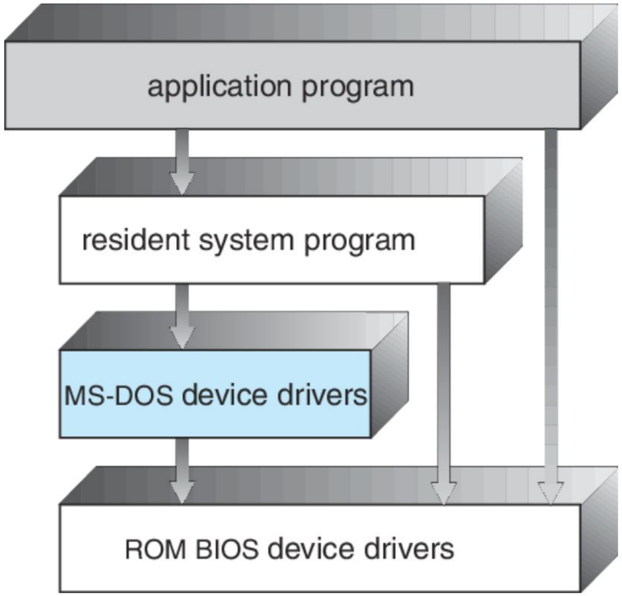
This separation allows flexibility if policy decisions are to be changed later.

OS Design Approaches

- Simple Structure
- Layered
- Microkernels
- Modular
- Hybrid Systems

OS Design Approaches: Simple Structure

- No well defined structure
- Start as small, simple, limited systems, and then grow
- No well defined layers, not divided into modules
- Example: MS-DOS
    - Initially written to provide the most functionality in the least space
    - Grew beyond its original scope
    - Levels not well separated: Programs access I/O directly
    - Excuse? The hardware of that time was limited (no user/kernel mode)



OS Design Approaches: Layered

Monolithic Operating Systems:

- No one had experience in building truly large software systems
- The problems caused by mutual dependence and interaction were grossly underestimated
- Such lack of structure became unsustainable as OS grew
- Early UNIX, Linux, Windows systems were monolithic, partially layered

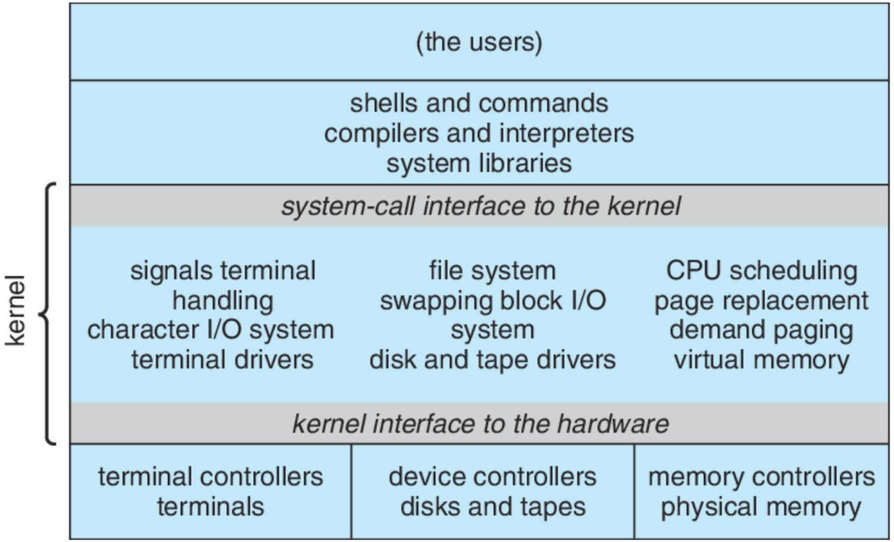
Enter hierarchical layers and information abstraction

- Each layer is implemented exclusively using operations provided by the lower layers
- It does not need to know how they are implemented
- Lower layers abstract certain data structures and operations

Simple Layered Approach

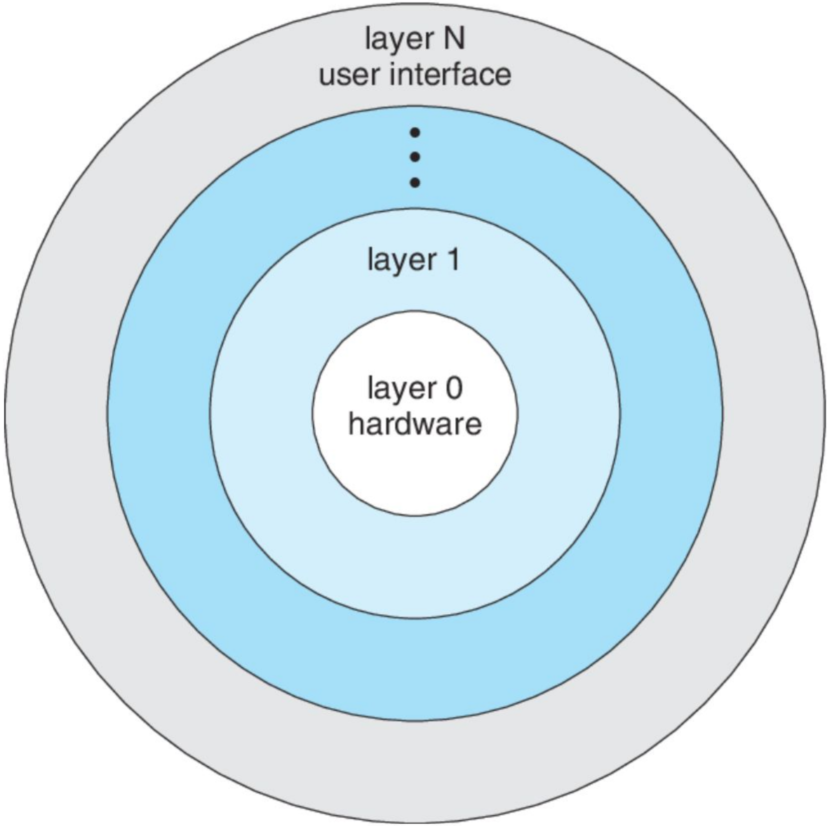
- Enormous amount of functionality crammed into the kernel
- "The Big Mess": a collection of procedures that can call any of the other procedures whenever they need to
- No encapsulation, total visibility across the system
- Very minimal layering

The original UNIX



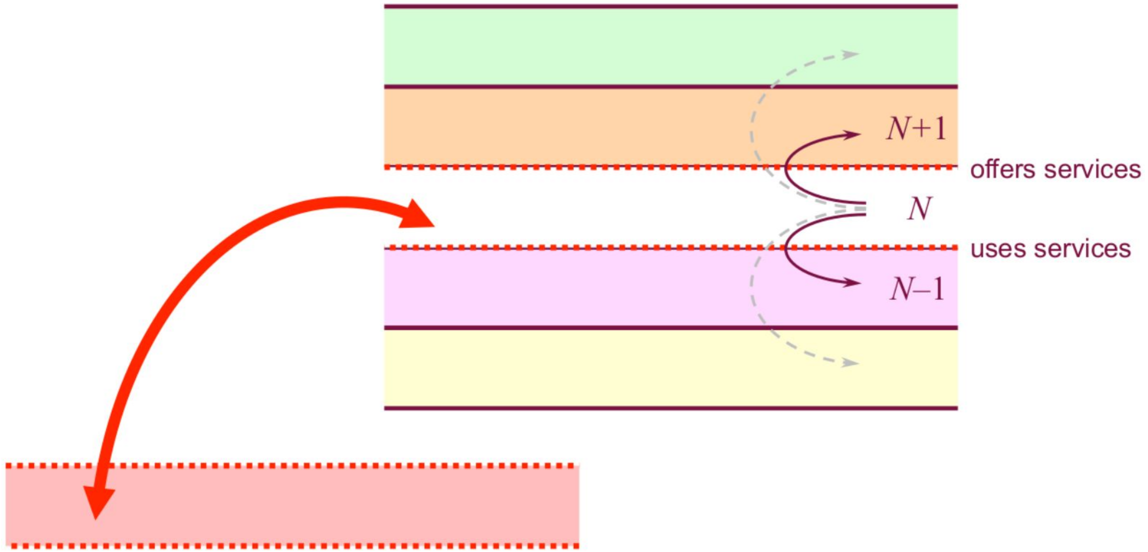
Fully Layered Approach

- The OS is divided into a number of layers, each built on top of lower layers.
- With modularity, layers are selected such that each uses functions and services of only lower layers
- THE system (by Dijkstra), MULTICS,
- GLUnix, VAX/VMS



Layered Approach

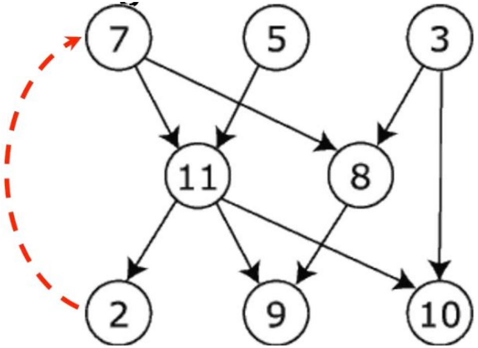
- Layers can be debugged and replaced independently without bothering the other layers above or below.
- Famous example: TCP/IP networking stack



Layered Approach

Major difficulty with layering:

- Appropriately defining the various layers!
- The more layers, the more indirections from function to function and the bigger the overhead in function calls
- Layering is only possible if all function dependencies can be sorted out in a Directed Acyclic Graph (DAG)
- However there might be conflicts in the form of circular dependencies (cycles)

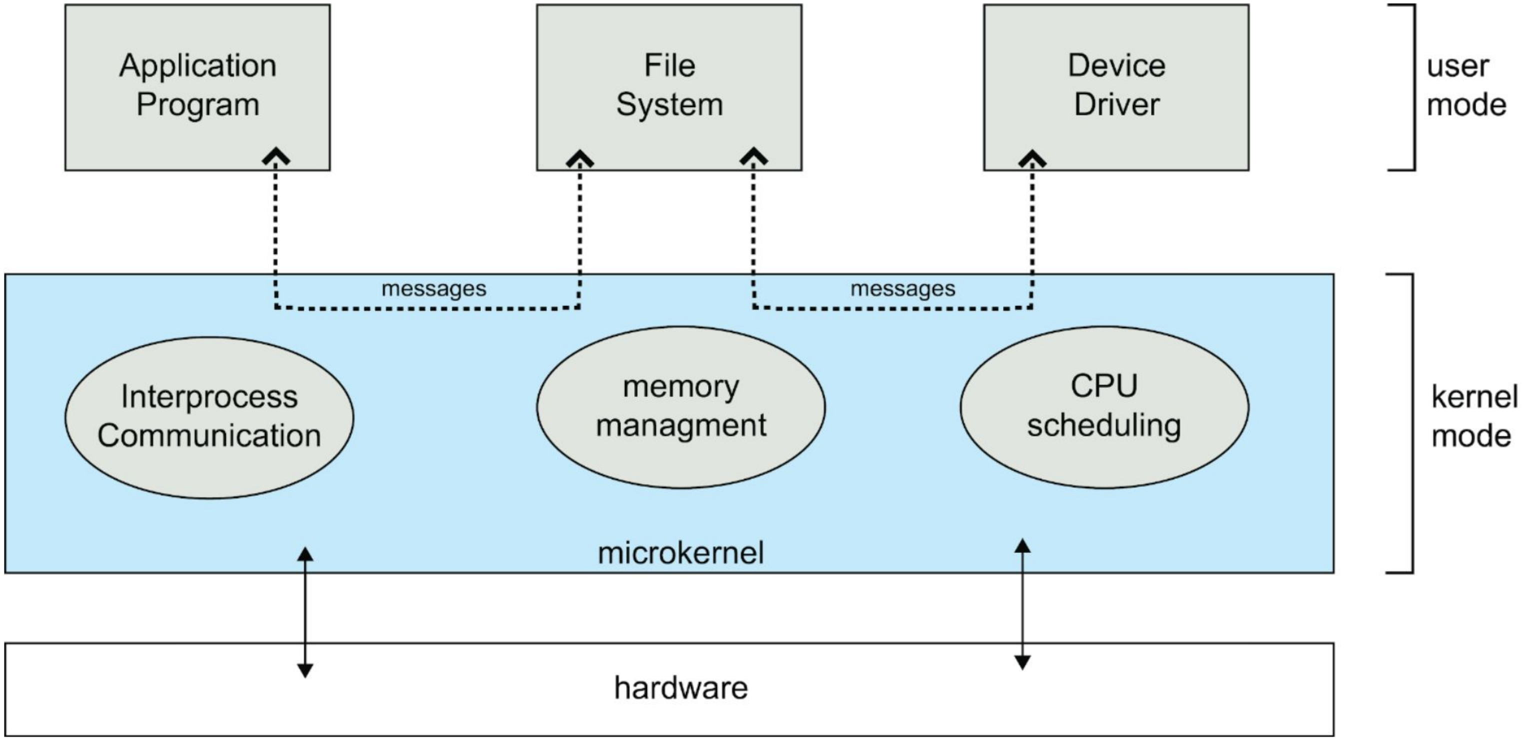


OS Design Approaches: Microkernels

- A microkernel is a reduced operating system core that contains only essential OS functions.
- The idea is to minimize the kernel by moving up as much functionality as possible from the kernel into user space.
- Many services traditionally included in the OS are now external subsystems running as user processes:
  - Device drivers
  - File systems
  - Virtual memory manager
  - Windowing GUI system
  - Security services

Microkernel Approach

Examples: QNX, Tru64 UNIX, Mach (CMU), Windows NT



Microkernel Approach

- Pros:
  - Extensibility: It is easier to extend a microkernel-based OS as new services are added in user space, not in the kernel.
  - Portability: It is easier to port to a new CPU, as changes are needed only in the microkernel, not in the other services
  - Reliability & security: Much less code is running in kernel mode. Failures in user space services don't affect kernel space.
- Cons:
  - Again, performance overhead due to communication from user space to kernel space
  - Not always realistic, some functions must remain in kernel space.

OS Design Approaches: Modular

- Many modern operating systems implement kernel modules.
  - Modern UNIX, Solaris, Linux, Windows, Mac OS X
- This is similar to object-oriented approach:
  - Each module talks to the others over known interfaces.
  - Each module is loadable dynamically as needed within the kernel.
- Overall, modules are similar to layers, but with more flexibility.
  - All are within kernel and any module could call any other module.
- Modules are also similar to the microkernels, except that they are inside the kernel and don't need message passing.

OS Design Approaches: Hybrid Systems

- Many real OSs use a combination of different approaches
  - Linux: Monolithic & modular
  - Windows: Monolithic & microkernel & modular
  - Mac OS X: Microkernel & modular

Review

- A very brief history of computer architecture
- High level topics of the course
- Goals or Responsibilities of OS
- Task of an OS
- Major OS components
- OS design approaches

Acknowledgements

- “Operating Systems Concepts” book and supplementary material by A. Silberschatz, P. Galvin and G. Gagne
- “Operating Systems: Internals and Design Principles” book and supplementary material by W. Stallings
- “Modern Operating Systems” book and supplementary material by A. Tanenbaum
- R. Doursat and M. Yuksel from University of Nevada, Reno
- Farshad Ghanei from Illinois Tech
- T. Kosar and K. Dantu from University at Buffalo