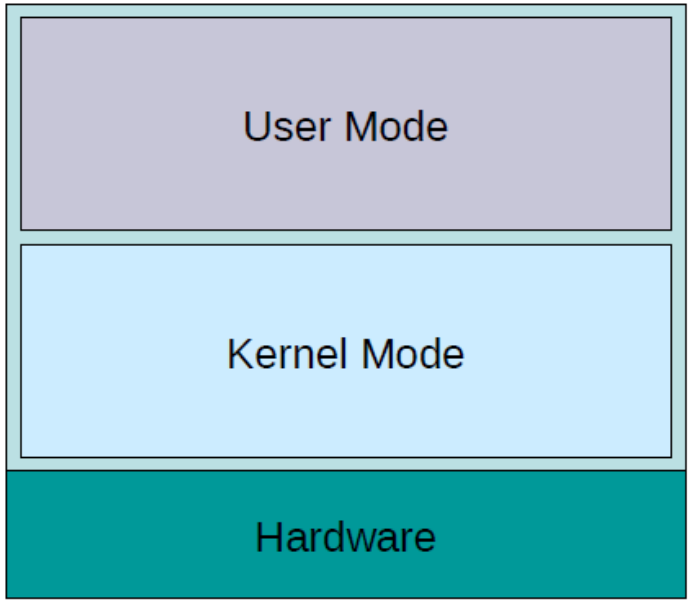
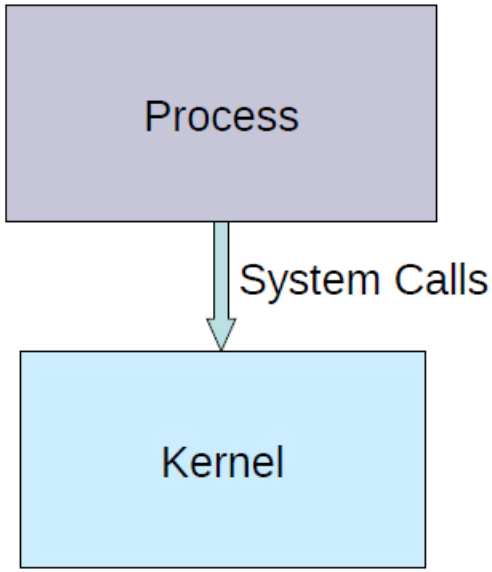
Operating Modes

- User Mode
  - Where processes run
  - Restricted access to resources
  - Restricted capabilities
- Kernel mode a.k.a. Privileged mode
  - Where the OS runs
  - Privileged (can do anything)



Communicating with the OS (System Calls)

- System call invokes a function in the kernel using a Trap
- This causes
  - Processor to shift from user mode to privileged mode
- On completion of the system call, the execution gets transferred back to the user mode process



OS Evolution

- **Evolution driven by Hardware improvements**
- **+ User needs**
  - – eg. low power requirements, Increased / reduced security, lower latency
  - – Evolution by
    - New/better abstractions
    - New/better resource management
    - New/better low level implementations

Gen 1: Vacuum Tubes

- Hardware

– Vacuum tubes and IO with punchcards

– Expensive and slow

- User Apps

– Generally straightforward numeric computations done in machine language

Gen2: Mainframes

- Hardware

– transistors

- • User Programs

– Assembly or Fortran entered using punch cards

- OS : Batch systems

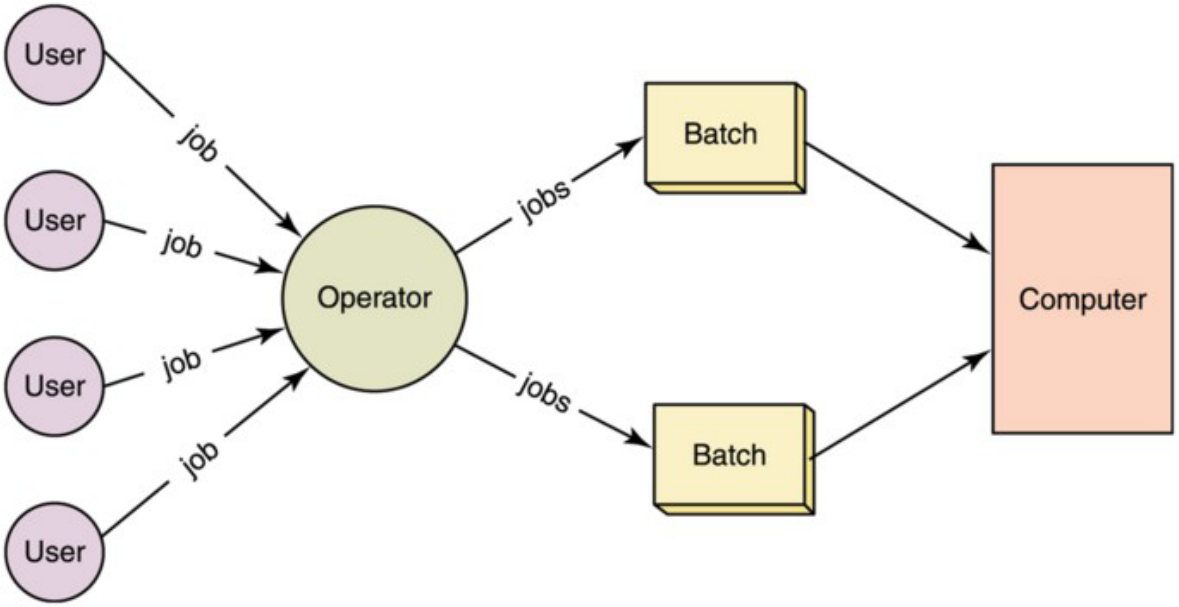
– Possibly greatest invention in OS

• Computers may be able to schedule their own workload by means of software

Batch Systems

- In a batch operating system, tasks are submitted to a pool, and only after the completion of one job, the next job is executed
- In this system, the CPU idle time (free time) is high and throughput of the system is low. Throughput is the number of jobs executed per unit time.
- Advantages
  - Reduce setup time by batching similar jobs
  - Automatic job sequencing – automatically transfers control from one job to another.
  - The operator would run the job in a batch.
- Disadvantages
  - CPU used to be idle – because the speed of I/O devices are comparatively slower.
  - Mismatch of speed of I/O and CPU – fast card reader can read 20 cards/sec.

Batch System



Gen 3 : Mini computers

- Hardware

– SSI/MSI/LSI ICs

– Random access memories

– Interrupts (used to simulate concurrent execution)

- User Programs

– High level languages (Fotran, COBOL, C, …)

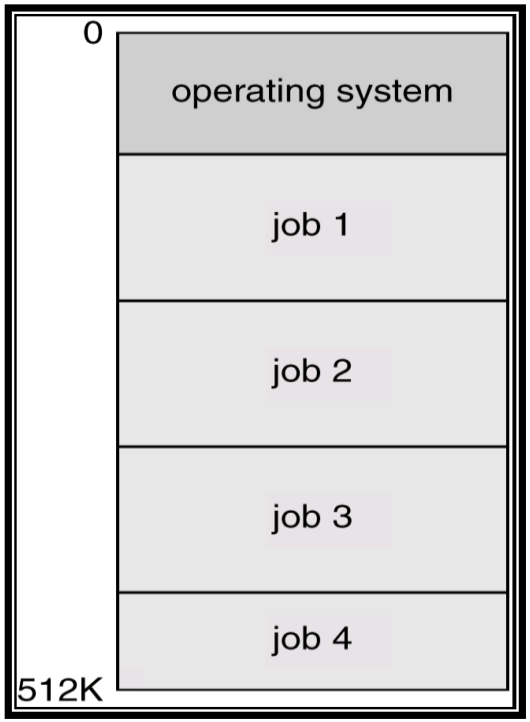
- Operating Systems

– Multiprogramming

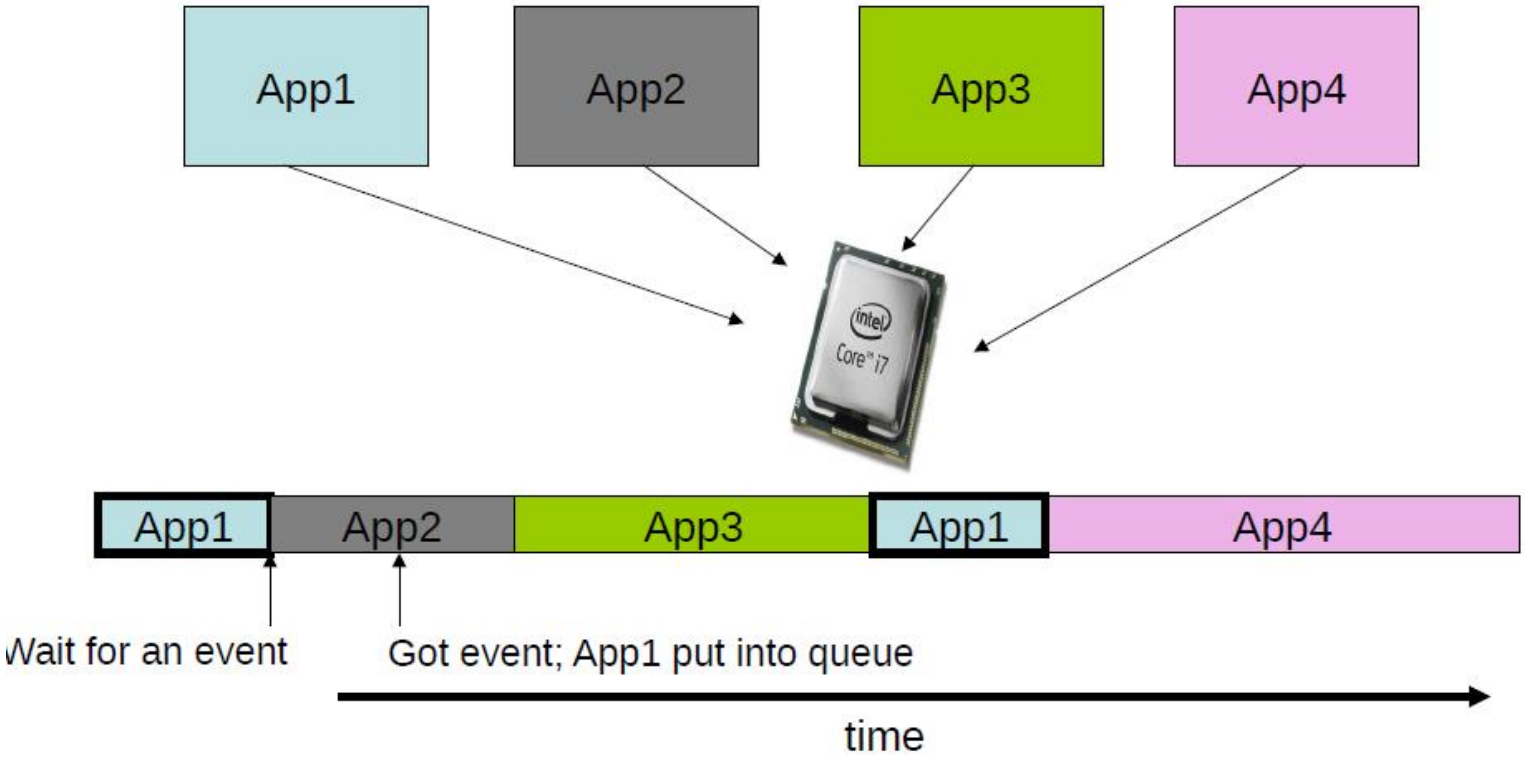
– Spooling

Multiprogramming Operating Systems

- Multiple jobs are kept in main memory at the same time
- CPU is multiplexed among the jobs
- When one waits for I/O the next job executes
- OS control Scheduling of jobs and protection between jobs

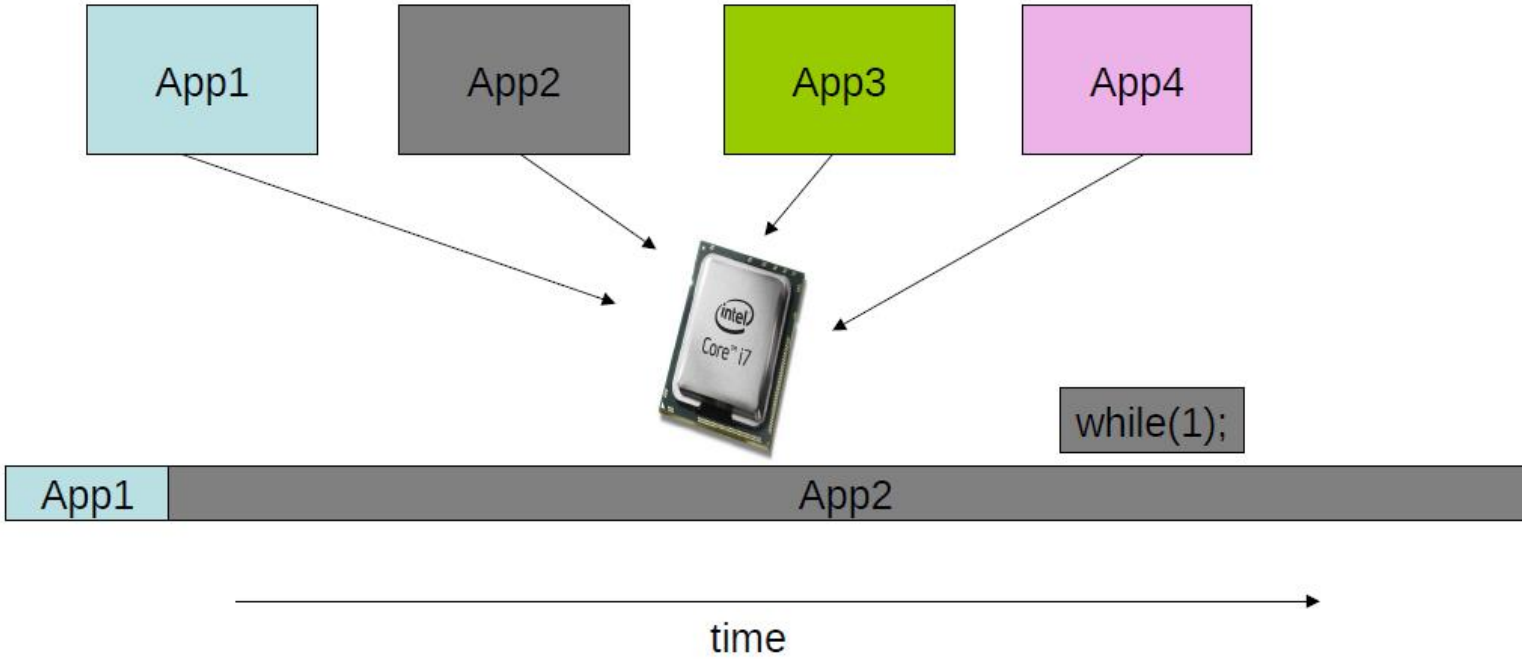


When OS supports Multiprogramming



When CPU idle, switch to another app

Multiprogramming could cause starvation

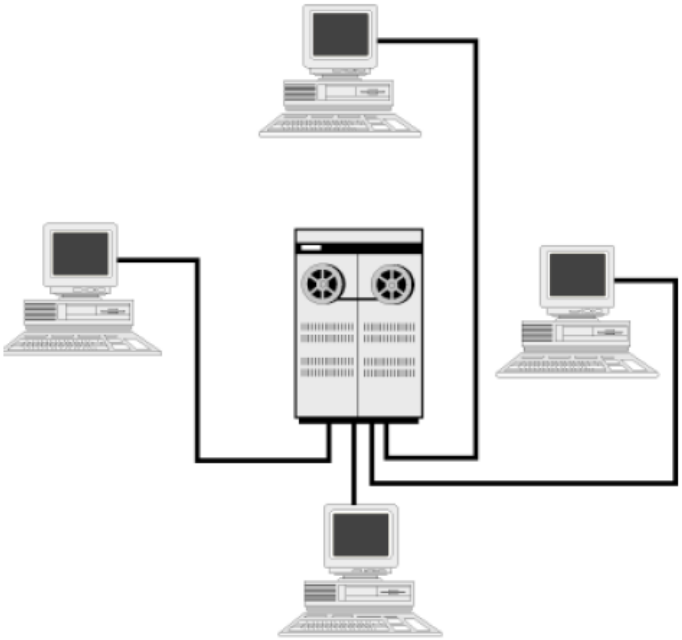


One app can hang the entire system

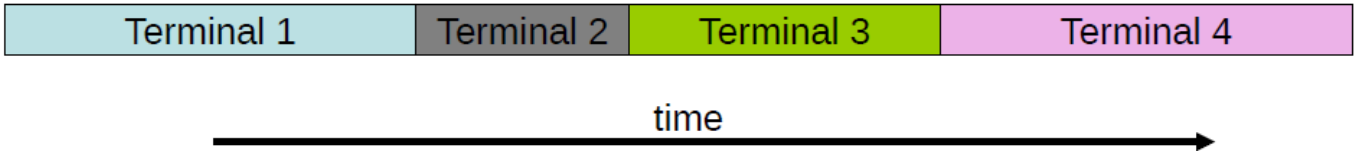
Multiprogramming Operating Systems

Jobs to be executed are maintained in the memory simultaneously and the OS switches among these jobs for their execution. When one job waits for some input (WAIT state), the CPU switches to another job. This process is followed for all jobs in memory. When the wait for the first job is over, the CPU is back to serve it. The CPU will be busy till all the jobs have been executed. Thus, increasing the CPU utilization and throughput.

Timesharing

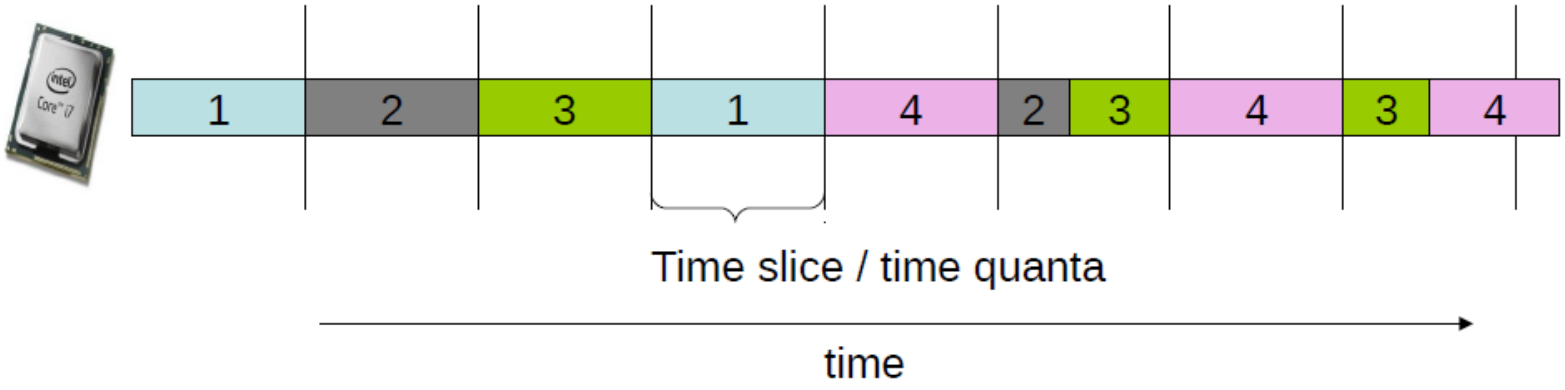


Who uses the CPU?



When OS supports Time Sharing (Multitasking)

- Time sliced
- Each app executes within a slice
- Gives impression that apps run concurrently
- No starvation. Performance improved

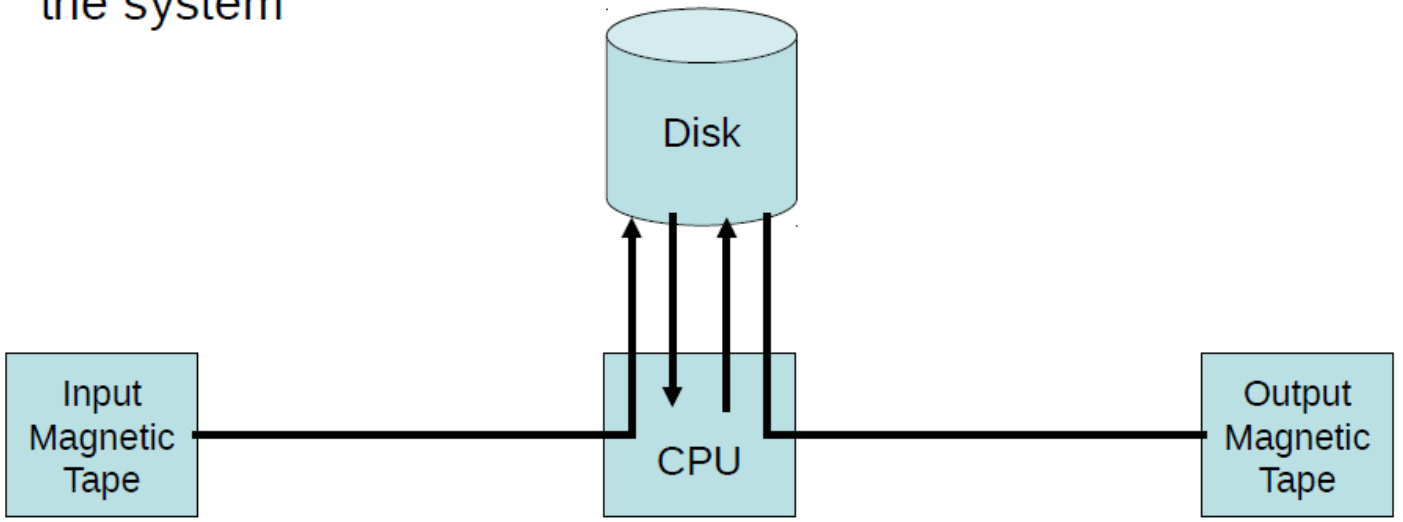


Time-Sharing Systems–Interactive Computing

- The CPU is multiplexed among several jobs that are kept in memory and on disk (the CPU is allocated to a job only if the job is in memory).
- A process swapped in and out of memory to the disk.
- On-line communication between the user and the system is provided; when the operating system finishes the execution of one command, it seeks the next “control statement” from the user’s keyboard.

Spooling

- Uses buffers to continuously stream inputs and outputs to the system



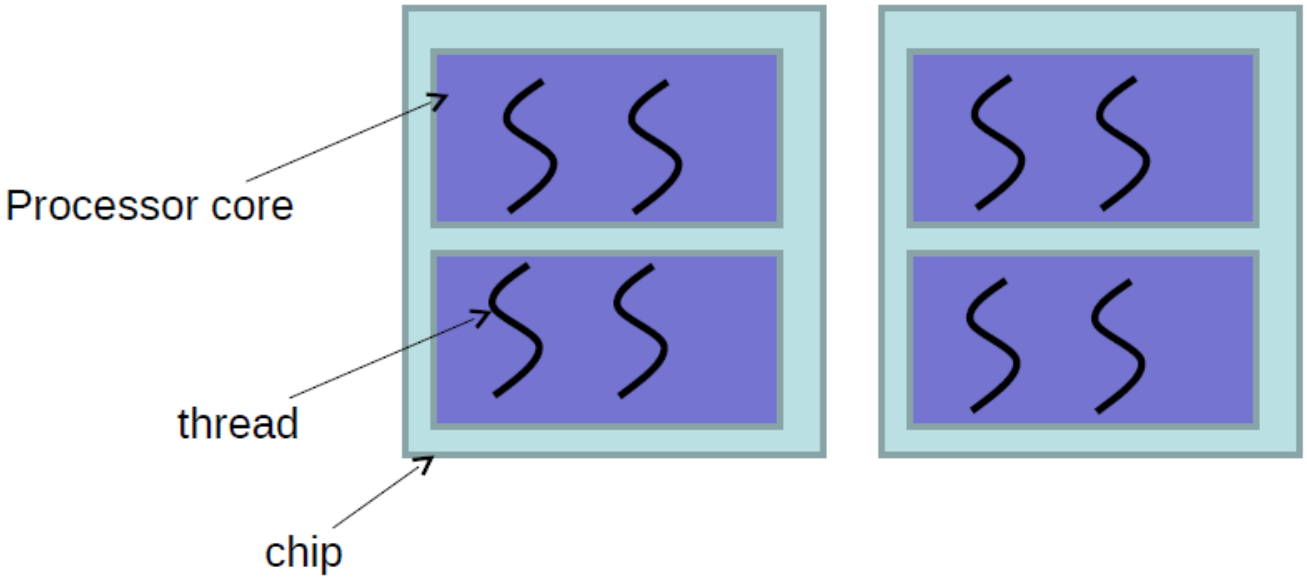
- Pros : better utilization / throughput
- Cons : still not interactive

Gen 4: Personal Computers

- Hardware
  - VLSI ICs
- User Programs
  - High level languages
- Operating Systems
  - Multi tasking
  - More complex memory management and scheduling
  - Synchronization
  - Examples : Windows, Linux, etc

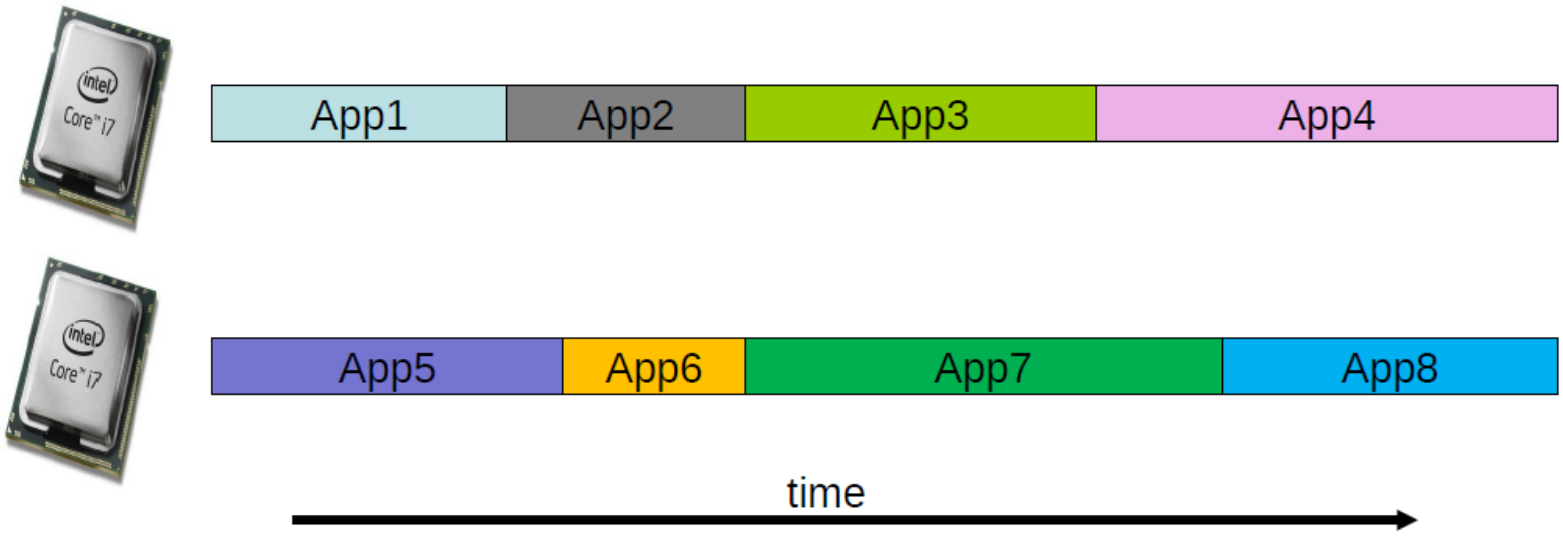
Multiprocessors

- Multiple processors chips in a single system
- Multiple cores in a single chip
- Multiple threads in a single core

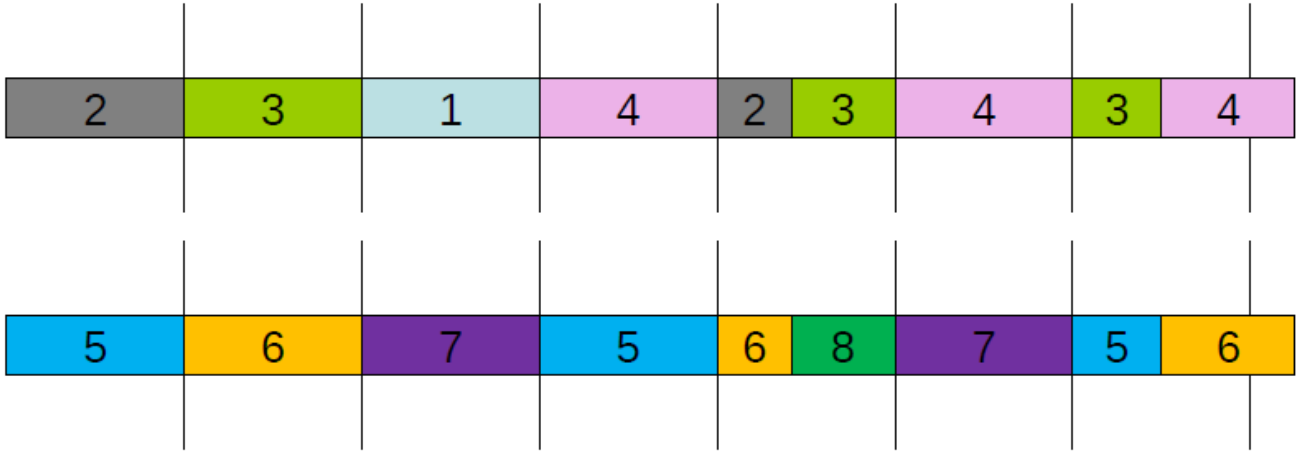


Multiprocessors

- Each processor can execute an app independent of the others

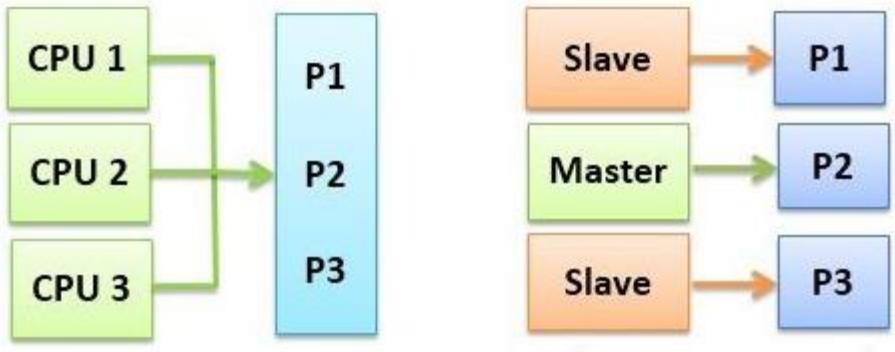


Multiprocessors and Multithreading



Multiprocessor Systems

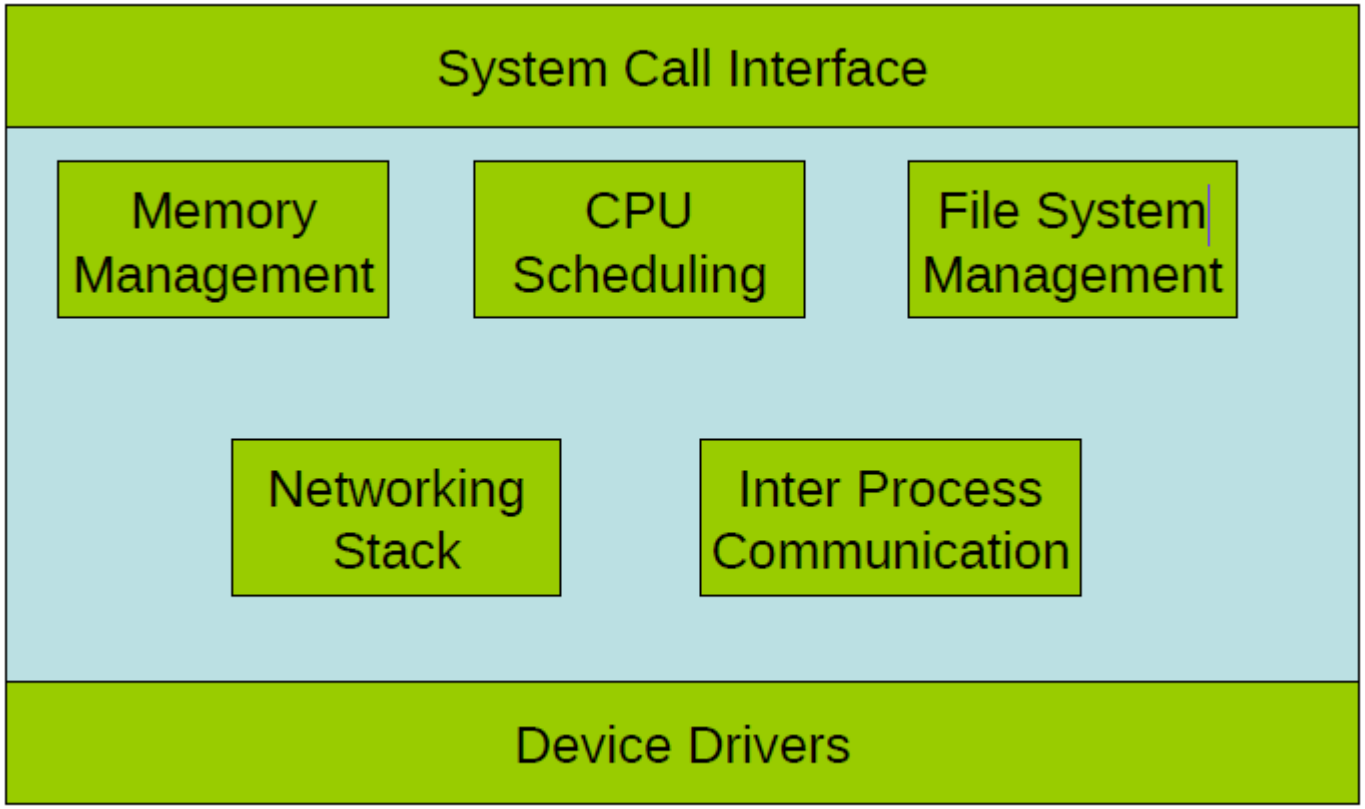
- ***Symmetric multiprocessing (SMP)***
  - Each processor runs and identical copy of the operating system.
  - Many processes can run at once without performance deterioration.
  - Most modern operating systems support SMP
- ***Asymmetric multiprocessing***
  - Each processor is assigned a specific task; master processor schedules and allocated work to slave processors.
  - More common in extremely large systems



Multiprogramming Vs Multitasking

- Multiprogramming (older OSs) one program as a whole keeps running until it blocks for IO. Increases CPU utilization
- Multitasking (modern OSs) time sharing is best manifested because each running process takes only a fair quantum of the CPU time. Reduces response time

Structure of an OS

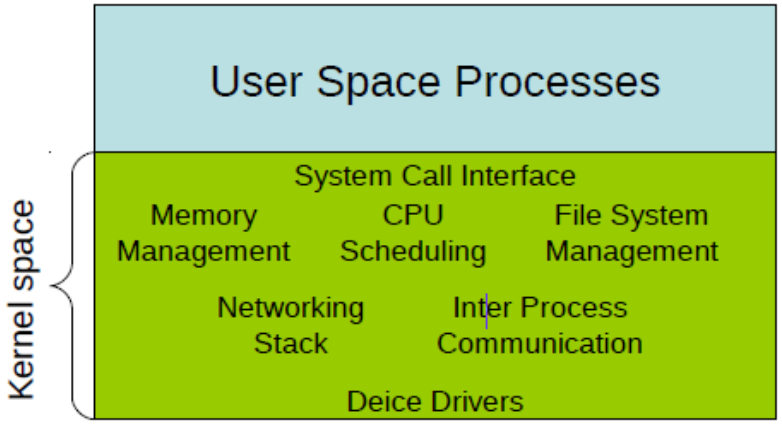


What is kernel?

- **Kernel** is the core part of an operating system which manages system resources. It also acts like a bridge between application and hardware of the computer. It is one of the first programs loaded on start-up (after the Boot loader).

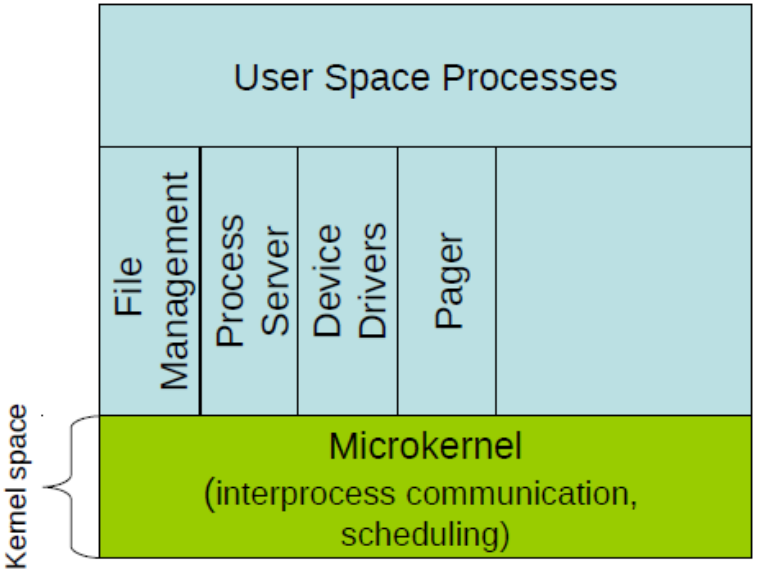
OS Structure

Monolithic Structure



- Linux, MS-DOS, xv6
- All components of OS in kernel space
- Cons : Large size, difficult to maintain, likely to have more bugs, difficult to verify
- Pros : direct communication between modules in the kernel by procedure calls

OS Structure : Microkernel



Eg. QNX and L4

- Highly modular.
  - Every component has its own space.
  - Interactions between components strictly through well defined interfaces (no backdoors)
- Kernel has basic inter process communication and scheduling
  - Everything else in user space.
  - Ideally kernel is so small that it fits the first level cache

Monolithic Vs Microkernel

	Monolithic	**Microkernel**
Inter process communication	Signals, sockets	Message queues
Memory management	Everything in kernel space (allocation strategies, page replacement algorithms,)	Memory management in user space, kernel controls only user rghts
Stability	Kernel more 'crashable' because of large code size	Smaller code size ensures kernel crashes are less likely
I/O Communication (Interrupts)	By device drivers in kernel space. Request from hardware handled by interrupts in kernel	Requests from hardware converted to messages directed to user processes
Extendibility	Adding new features requires rebuilding the entire kernel	The micro kernel can be base of an embedded system or of a server
Speed	Fast (Less communication between modules)	Slow (Everything is a message)