# Operating Systems 1: Processes

## Introduction

* We are using an OS daily. Even currently while watching the class.
* [POLL] Which OS are you using right now?
* Why learn OS?
  + To understand how the applications we develop or use really work. How are they able to get things done?

### What is an OS?

#### Resource Manager



* Tell how every process basically needs different resources to do their work and to avoid conflicts, there needs to be someone to manage them. Like a Program Manager in a company.

#### Functionalities for Developers

* Interacting with Network
* Working with File System
* Imagine having to write the code to rotate the hard disk to fetch a file at a particular location.

## Uni v/s Multiprogramming Computers

* Multiple programs being executed at a time (not necessary concurrently) instead of a single.
* Run the other one when first is waiting for some resources.
* Make full use of CPU
* [Show multiple programs running in parallel on Mac]
* Multitasking is the same as multiprogramming. Just used in windows.

### Types of Multiprogramming

* Single v/s Multi User
* Preemptive v/s Non Preemptive -> OS can force process to be removed even before completion

## fork()

* System Call
* Makes CPU come in Kernel Mode
* What is Kernel Mode
  + Non Preemptive
  + Key OS functionality
  + Atomic Execution
* Creates a new process with copy of the current variables

| int main() {  int a = 2;  int b = 3;  id = fork();  print("hi"); } |
| --- |

* Id = zero in the child process.
* Non zero in the parent process.
* Less than zero means some error.
* [Show demo of how the program will work with 1 and 2 fork]
* [Show that n forks mean 2^n processes]
* How OS works
  + Even OS is a process that is started at the start of a program by BIOS that internally would start other processes. [Show systemd in Mac]

## Process

* Program in execution
* Has resources allocated to it
* Unit of execution

### What a process consists of



### Process in memory

[Give hints regarding what all details wrt a process might be needed to store by an OS. Assume process class is there.]

* ID
* CPU related info (Priority etc.)
* Memory info (Limits etc.)
* Opened Files
* I/O related details
* Protection (Kernel/ User Mode)

All of these go in PCB (Process Control Block).

## CPU Scheduling

### CPU Bound v/s I/O Bound Processes

* I/O bound like Interactive programs. Most of the time waiting for I/O than using CPU.
* CPU Bound like scientific applications doing calculations

### How I/O Happens

#### Interrupts

* Program raises a CPU Interrupt telling CPU that it is waiting for some I/O to happen
* Interrupts are signals emitted to tell the processor that some interruption is required.
* CPU sits idle till the I/O (network call, command line input etc.) which is very slow wrt CPU completes.
* This is a waste of CPU time and CPU can instead spend this time doing something useful.

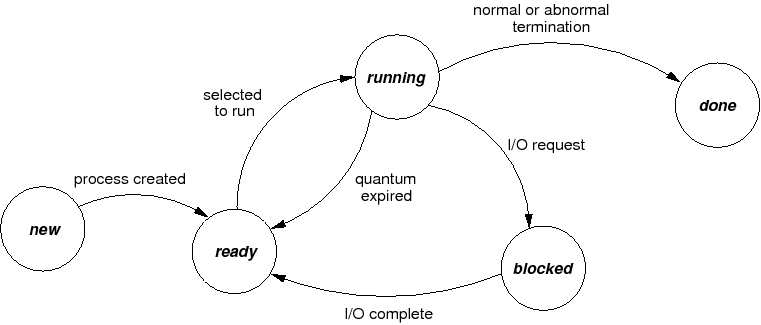
### Why CPU Scheduling

* Utilize the CPU to fullest

Metrics wrt Processes

* Throughput: Number of process per unit time
* Arrival Time: When a process comes to CPU
* Wait Time: Time till completion when process isn’t executing but is waiting for I/O or waiting to be assigned to CPU.
* Burst Time: Time needed to make the process execute completely.
* Completion Time: Time when a process completes its execution,
* I/O Burst Wait Time: Time for I/O to happen
* Turnaround Time = Completion Time - Arrival Time
* Response Time = Time when process first starts running (gets assigned to CPU)
* Schedule Time = Time to completely finish all processes
* Deadline: Expectation by which process should ideally finish
  + Underrun: When process finishes before deadline
  + Overrun: When process finishes after deadline

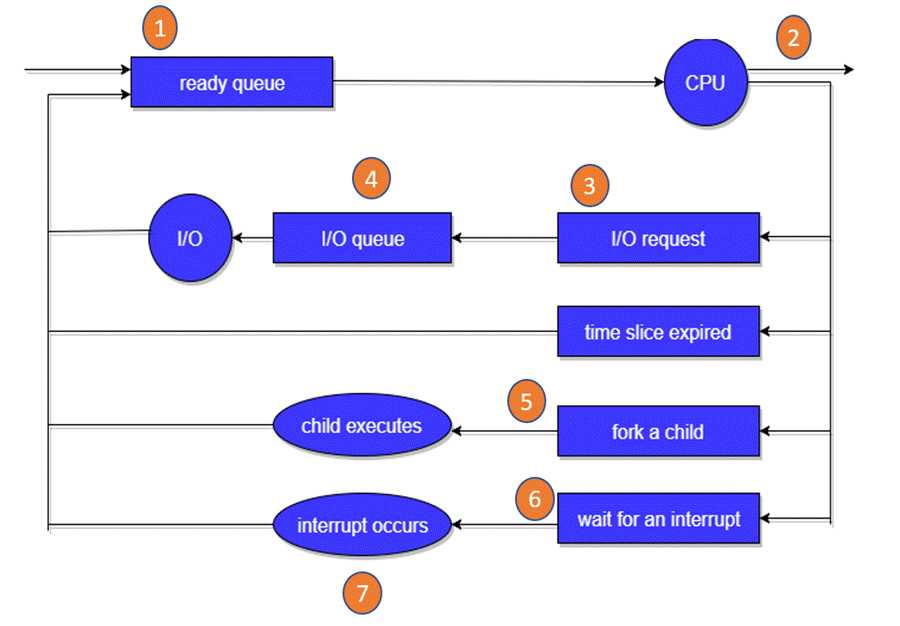
### Process Lifecycle



### 

### Process Scheduling Queues

When a process is not running, it is waiting in one of the scheduling queues.



#### Dispatcher

* Picks the process from Ready Queue and assigns it to the CPU for scheduling.
* Moves the PCB out of CPU into the registers for execution.

#### Context Switch

* We as humans when we do multiple tasks in parallel, it takes some time to recall what had happened earlier.
* Moving a process out of register, storing it in memory in ready queue, replacing it with another process is known as context switch
* Context switch takes time and thus an algorithm that involves a lot of context switch isn’t good.

### Scheduling Algorithms

* Assume no I/O
* No context switch time

#### FCFS

* Non Preemptive.
* When we have to select a new process from the ready queue, select the process that has arrived the earliest.

| PId | Arrival Time | Burst Time | Response Time | Completion Time | TAT | Wait Time |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 4 |  |  |  |  |
| 2 | 1 | 5 |  |  |  |  |
| 3 | 2 | 6 |  |  |  |  |
| 4 | 3 | 8 |  |  |  |  |
| 5 | 4 | 2 |  |  |  |  |
| 6 | 5 | 4 |  |  |  |  |

[Show how scheduling will work for above by filling the above table, as well as via a Gantt chart]

#### SRTF

* Shortest Response Time First
* Preemptive

| PId | Arrival Time | Burst Time |
| --- | --- | --- |
| 1 | 0 | 8 |
| 2 | 1 | 6 |
| 3 | 2 | 4 |
| 4 | 3 | 2 |
| 5 | 4 | 6 |
| 6 | 5 | 1 |

[Show the scheduling via Gantt chart and show how preemptive behaviour happens.]

#### Starvation

* Some processes have to keep on waiting for very long and don’t get CPU time because of other new processes coming ahead.

#### Round Robin

* Preemptive
* Time Quantum (q)
* After every q units, check the process most ahead in the queue and schedule that.
* New process gets added behind the queue.
* Current process gets added at the end when preempted.

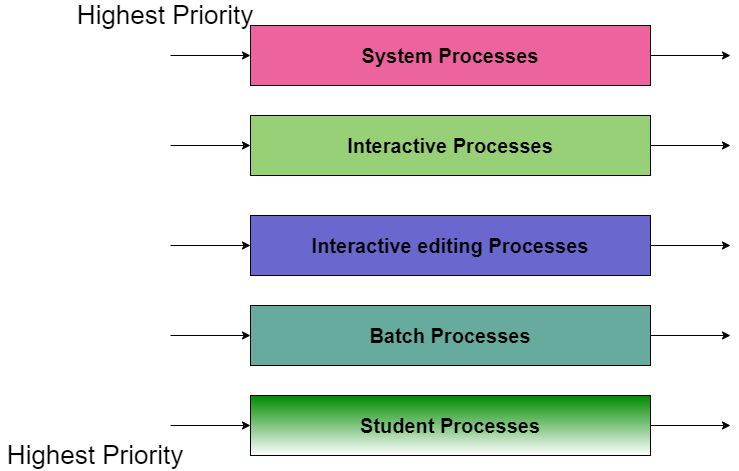
| PId | Arrival Time | Burst Time |
| --- | --- | --- |
| 1 | 0 | 4 |
| 2 | 1 | 5 |
| 3 | 2 | 6 |
| 4 | 3 | 3 |
| 5 | 4 | 4 |
| 6 | 5 | 1 |

Homework

* Learn about SJF and Priority scheduling.
* [Give a brief idea about them. The only difference is in one, the process with highest priority is selected, in the other the process with the shortest time.]

#### Multilevel Queues

* In real systems, a single scheduling algorithm might not work well.
* For different type of processes, we might need different scheduling algorithms.
* Example:
  + For I/O bound processes, RR with short time
  + For CPU intensive, maybe something else.



* If there is a process in the highest queue, pick that. Else go to lower queues. Each queue internally may have its own scheduling algorithm.