**PROCESS SCHEDULING**

The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization. The objective of time sharing is to switch the CPU among processes so frequently that users can interact with each program while it is running. To meet these objectives, the process scheduler selects an available process (possibly from a set of several available processes) for program execution on the CPU. For a single-processor system, there will never be more than one running process. If there are more processes, the rest will have to wait until the CPU is free and can be rescheduled.

The act of determining which process is in the ready state, and should be moved to the running state is known as Process Scheduling. Process scheduling is an important part of multiprogramming operating systems.

The prime aim of the process scheduling system is to keep the CPU busy all the time and to deliver minimum response time for all programs. For achieving this, the scheduler must apply appropriate rules for swapping processes **IN**and **OUT**of CPU.

**1)Scheduling Queues:**

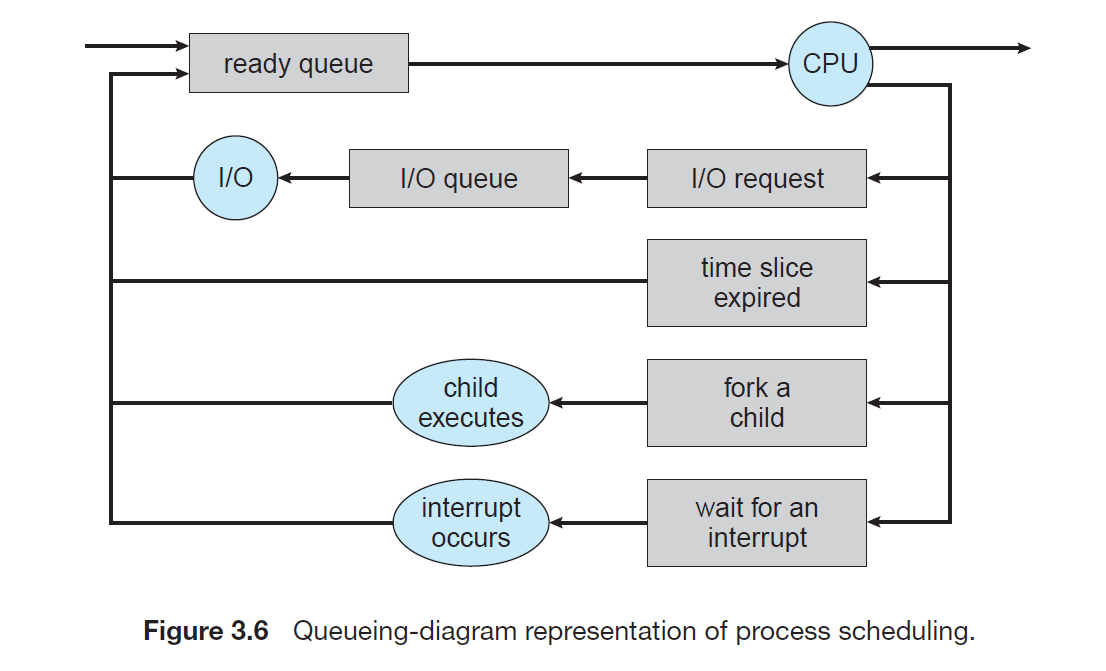
There are multiple states a process has to go through during execution. The OS maintains a separate queue for each state along with the process control blocks (PCB) of all processes. The PCB moves to a new state queue, after being unlinked from its current queue, when the state of a process changes.

The following 3 major process scheduling queues are maintained by the Operating System:

**1)Job Queue:** All processes, upon entering into the system, are stored in the Job Queue.

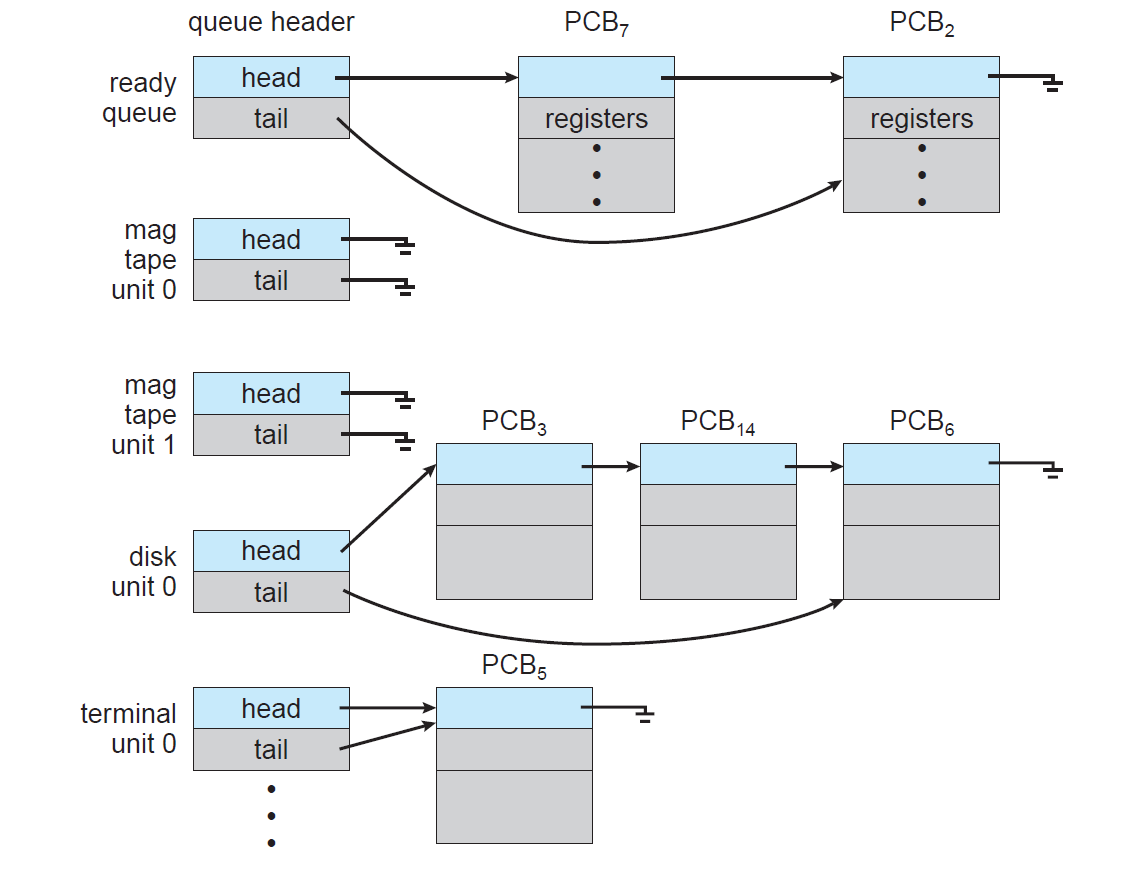
**2)Ready Queue:** Processes in the Ready state are placed in the Ready Queue.

**3)Device Queue:** The operating system maintains a separate device queue for each I/O device. This device queue holds the processes that are waiting to perform I/O on the device.

****

Each rectangular box represents a queue. Two types of queues are present: the ready queue and a set of device queues. The circles represent the resources that serve the queues, and the arrows indicate the flow of processes in the system.  
 A new process is initially put in the ready queue. It waits there until it is selected for execution, or is dispatched. Once the process is allocated the CPU and is executing, one of several events could occur:  
• The process could issue an I/O request and then be placed in an I/O queue.  
• The process could create a new subprocess and wait for the subprocess's termination.  
• The process could be removed forcibly from the CPU, as a result of an interrupt, and be put back in the ready queue.

In the first two cases, the process eventually switches from the waiting state to the ready state and is then put back in the ready queue. A process continues this cycle until it terminates, at which time it is removed from all queues and has its PCB and resources deallocated.



**Fig: The Ready queue and Various I/O device queues**

The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the ready queue. This queue is generally stored as a linked list. A ready-queue header contains pointers to the first and final PCBs in the list. Each PCB includes a pointer field that points to the next PCB in the ready queue. The system also includes other queues. When a process is allocated the CPU, it executes for a while and eventually quits, is interrupted, or waits for the occurrence of a particular event, such as the completion of an I/O request.

**2)Schedulers:**

A process migrates among the various scheduling queues throughout its lifetime. The operating system must select, processes from these queues in some fashion. The selection process is carried out by the appropriate scheduler. A scheduler is a type of system software that allows you to handle process scheduling.

**Types of process schedulers:**

There are three types of process schedulers in operating systems based on their features:

* Long Term or job scheduler
* Short-term or CPU scheduler
* Medium-term scheduler

### 1.Long-Term Scheduler or Job Scheduler:

Long term scheduler is also known as a job scheduler. The long-term scheduler, or job scheduler, selects processes from Secondary storage and loads them into main memory for execution. That means it brings the processes from New State to the ‘Ready State’.

The long-term scheduler controls the degree of multiprogramming (the number of processes in main memory). In general, most processes can be described as either I/O bound or CPU bound. An I/O-bound process is one that spends more of its time doing I/O than it spends doing computations. A CPU-bound process, in contrast, generates I/O requests infrequently, using more of its time doing computations. The main goal of the Long-Term scheduler is to select the best or balanced mix of I/O bound and CPU bound processes from the pool of jobs (Secondary Storage). If the job scheduler chooses more I/O bound processes, then all of the jobs may reside in the blocked state all the time and the CPU will remain idle most of the time. This will reduce the degree of Multiprogramming.  This scheduler increases the efficiency of CPU utilization since it maintains a balance between I/O and CPU-bound processes. Therefore, the Job of long-term scheduler is very crucial and may affect the system for a very long time.

The long-term scheduler executes much less frequently. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system. Thus, the long-term scheduler may need to be invoked only when a process leaves the system.

### 

### 2)Short-Term Scheduler or CPU Scheduler:

### Short term scheduler is also known as CPU scheduler. It selects one of the processes from the ready queue here all scheduling algorithms are used to select which job is going to be dispatched for the execution and after short-term scheduler decides the process, the dispatcher is responsible for loading these selected processes on the CPU.

### The Short-Term scheduler’s task can be essential in the sense that if it chooses a process with a long CPU burst time then all subsequent jobs will have to wait in a ready queue for a long period. This problem is called starvation which may arise if the short-term scheduler makes some mistakes while selecting the process. The short-term scheduler must select a new process for the CPU frequently.

### 

### 3) Medium-Term Scheduler:

Some operating systems, such as time-sharing systems, may introduce an additional, intermediate level of scheduling. This medium-term scheduler is diagrammed in below Figure. A running process may become suspended if it makes an I/O request. Suspended processes cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to secondary storage. This process is called **swapping**. The process is swapped out, and is later swapped in, by the medium-term scheduler.

Swapping may be necessary to improve the process mix or because a change in memory requirements has overcommitted available memory, requiring memory to be freed up. That means, it is helpful in maintaining a perfect balance between the I/O bound and the CPU bound. It reduces the degree of [multiprogramming](https://www.geeksforgeeks.org/multiprogramming-in-operating-system/).

### 

**3)Context Switch:**

Context switching is a technique or method used by the operating system that involves switching of the CPU from one process to another process. That means, the execution of the process that is present in the running state is suspended by the kernel and another process that is present in the ready state is executed by the CPU.

You can't directly switch a process from the running state to the ready state. Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process. This task is known as a context switch. When a context switch occurs, the kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run.

Context-switch time is pure overhead, because the system does no useful work while switching. Switching speed varies from machine to machine, depending on the memory speed, the number of registers that must be copied, and the existence of special instructions. A typical speed is a few milliseconds. Context-switch times are highly dependent on hardware support.

Context switching can happen due to the following reasons:

* When a process of high priority comes in the ready state. In this case, the execution of the running process should be stopped and the higher priority process should be given the CPU for execution.
* When an interruption occurs then the process in the running state should be stopped and the CPU should handle the interrupt before doing something else.
* When a transition between the user mode and kernel mode is required then you have to perform the context switching.