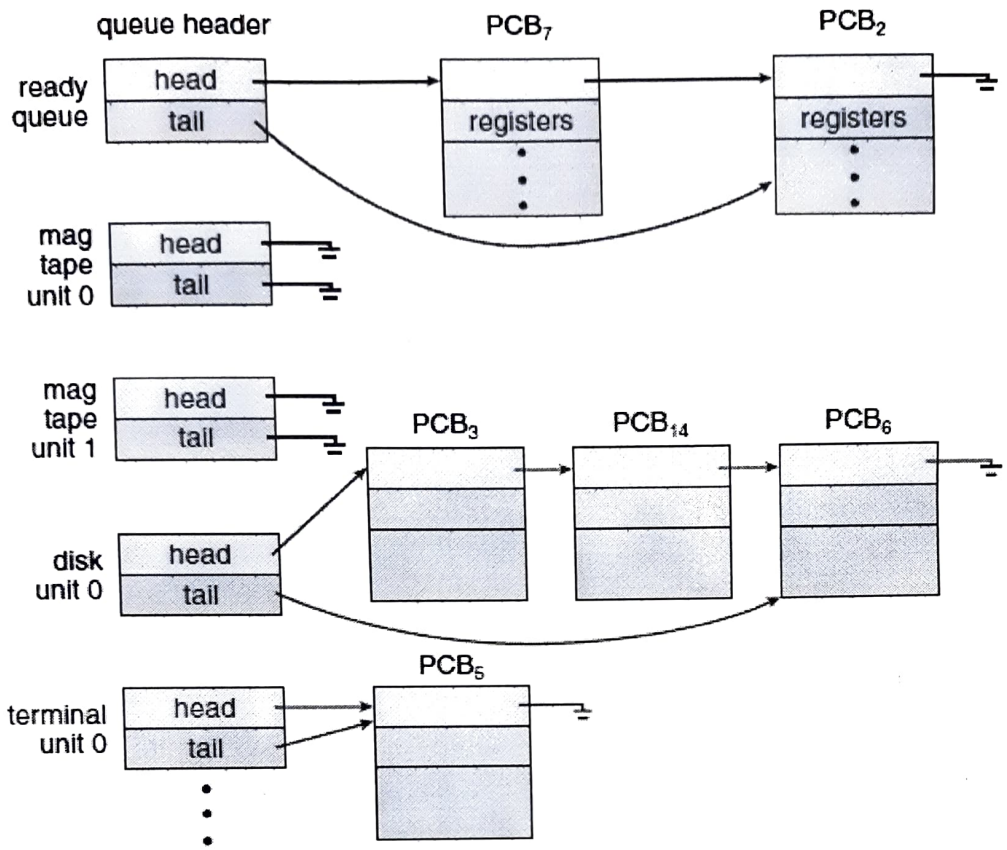
Process Scheduling



ready & IO devices queue.

In a multiprogramming environment, the process scheduler selects an available process for program execution on the CPU.

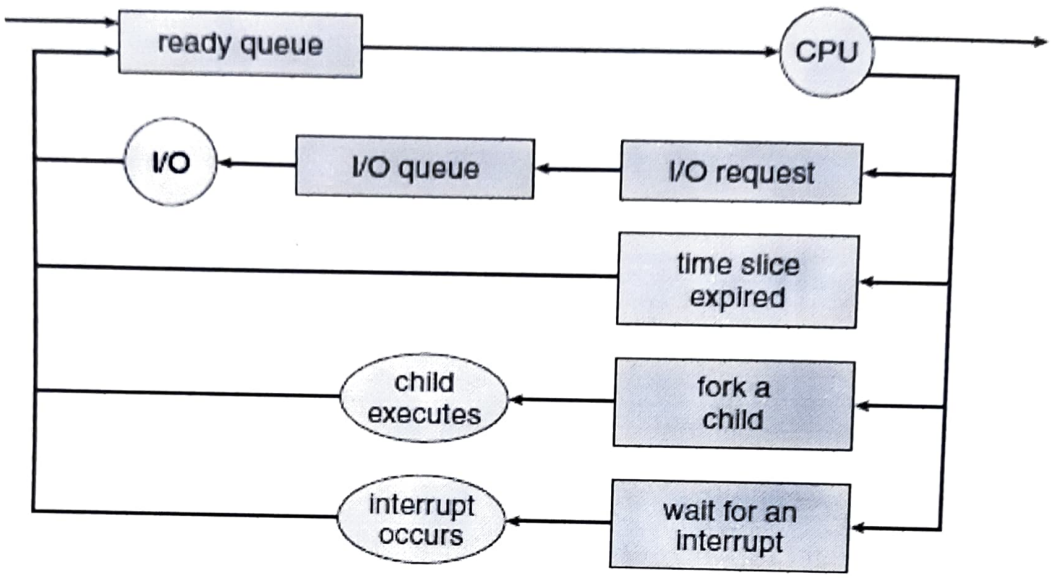
As process enters the system, they are put into a job queue.

The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the ready queue.

Linked list with pointers to first & final PCBs.

The queue with a list of processes waiting for a particular I/O device is called a device queue.

Queuing Diagram



A new process is initially put in the ready queue. It waits there until it is selected for execution, or dispatched.

Once the process is allocated CPU and is executing, one of several events could occur.

- The process could issue an I/O request & then be placed in an I/O queue
- The process could create a new child

process and wait for the child'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.

Activity:

For events 1 & 2, identify the process states that correspond to the events.

Answer:
It goes to "waiting" state & then back to "ready" after completion.

↳ ready queue.

Schedulers.

Long-term / Job Scheduler

When multiple processes are submitted, the processes are spooled to a mass storage device. Job scheduler selects processes from this pool & loads them into memory for execution.

Short-term/CPU scheduler

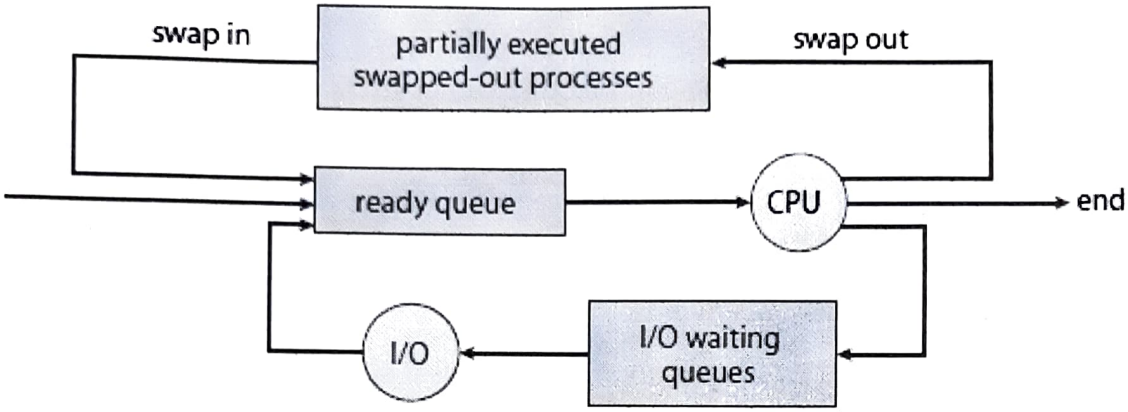
Selects from among the processes that are ready to execute and allocates CPU to one of them.

The long-term scheduler must select a proper mix of I/O-bound and CPU-bound processes. If all processes are I/O bound, the ready queue will almost always be empty and the short-term scheduler will have little to do.
If all the processes are CPU-bound, the I/O waiting queue will always almost always be empty, devices will go unused, and again the system will be unbalanced.

I/O-bound: Spends more time doing I/O.
CPU-bound: uses more of its time doing computations.

Long-term scheduler controls the degree of multiprogramming.
↳ the number of processes in memory.

How to improve process mix?
↳ Swapping.
↳ remove the process from memory & later re-introduce to continue the execution from where it left off.
(swap out, swap in)
↳ who does that?
↳ medium-term scheduler



Context Switch

Switching the CPU to another process by saving the state of current process (PCB) & restoring the state of a different process is called as a Context switch.

Context switch time is a pure over-head as the system does no useful work while switching.

Key terms:
- context
- state save
- state restore

GATE Question:

The maximum number of processes that can be in "ready" state for a computer with n CPUs is:

a) n b) n^2 c) 2^n d) Independent of n

Answer: d

Long-Term	Short-Term	Medium-Term
Long term is also known as a job scheduler	Short term is also known as CPU scheduler	Medium-term is also called swapping scheduler.
It is either absent or minimal in a time-sharing system.	It is insignificant in the time-sharing order.	This scheduler is an element of Time-sharing systems.
Speed is less compared to the short term scheduler.	Speed is the fastest compared to the short-term and medium-term scheduler.	It offers medium speed.
Allow you to select processes from the loads and pool back into the memory	It only selects processes that is in a ready state of the execution.	It helps you to send process back to memory.
Offers full control	Offers less control	Reduce the level of multiprogramming.

Threads

Process is a program that peyorms a single Thread of execution.

Eg:- User cannot simultaneously type in characters & run the spell checker within the same process.

Multiple threads can run in parallel & On a system that supports threads, PCB is expanded to include information for each thread.

MCQ:

* PCB's of all the current processes have a entry in,
  a. Process Register
  b. Process Table
  c. Program Counter
  d. Process Unit

Answer: b

Operations on Processes

prakash
hegade

Most operating systems identify processes according to a unique process identifier (pid) which is typically an integer number.

We can obtain a listing of process using following commands:

* ps

* ps -el

The init process serves as a root parent process for all user processes.

Process id of init is always 1 (one)

Scheduler process (known as swapper) has PID = 0.

Command to see the process creation tree:

pstree

To get process id and parent process id of the current process, we use the following functions from <unistd.h>

- getpid()
- getppid()

} → returns integer

When a process creates a child process,

- A child process may be able to obtain its resources directly from OS
- Constrained to a subset of the resources of the parent process.

The parent may have to partition its resources among its children or it may be able to share some resources among several of its children.

When a process creates a new process,

- The parent continues to execute concurrently with its children
- The parent waits until some or all of its children have terminated.

There are also two address-space possibilities for the new process:

- The child process is a duplicate of the parent process.
  - ↳ It has same program & data as the parent
- The child process has a new program loaded into it.

fork()

From the man page:

fork - creates a child process

```
#include <unistd.h>
pid_t fork(void);
```

fork() creates a new process by duplicating the calling process.
new process - child
calling process - parent

They both run in separate memory space. At the time of fork() both memory spaces have same content.

Child is exact duplicate of parent, except, to name a few,

- child has its own unique pid.
- child's parent process id is same as parent's process id.
- Process resource utilizations and CPU time counters are reset to zero in child.
- The termination signal of the child is always SIGCHLD.
- The child does not inherit its parent's memory locks.

prakash
begade

Return Type:

On success

the PID of child process is returned to the parent & 0 is returned in the child.

On Failure

-1 is returned in the parent, no child is created, and errno is set to indicate the error.

Program

```
#include<stdio.h>
#include<unistd.h>

int main()
{
    fork();
    printf("Ospp\n");
    return 0;
}
```

Output:

```
Ospp
Ospp
```

Both parent and child print Ospp once. Hence outputs Ospp twice.