

OPERATING SYSTEMS



Module2_Part2

Textbook : Operating Systems Concepts by Silberschatz



Scheduling

Scheduling is a fundamental function of OS.

When a computer is multiprogrammed, it has multiple processes competing for the CPU at the same time.

If only one CPU is available, then a choice has to be made regarding which process to execute next.

This decision making process is known as scheduling and the part of the OS that makes this choice is called a scheduler. The algorithm it uses in making this choice is called scheduling algorithm.

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.

It determines which process is moved from ready state to running state

Scheduling Queues

▮ **Scheduling queues:** As processes enter the system, they are put into a job queue. This queue consists of all process in the system.

The process that are residing in main memory and are ready & waiting to execute is kept on a list called ready queue.

This queue is generally stored as a linked list.

A ready queue header contains pointers to the first & final PCB in the list. The PCB includes a pointer field that points to the next PCB in the ready queue.

The lists of processes waiting for a particular I/O device are kept on a list called device queue. Each device has its own device queue.

A new process is initially put in the ready queue. It waits in the ready queue until it is selected for execution & is given the CPU.

Queuing diagram

A common representation of process scheduling is a **queuing diagram**.

Each rectangular box represents a queue. Two types of queues are present: the ready queue and a set of device queues.

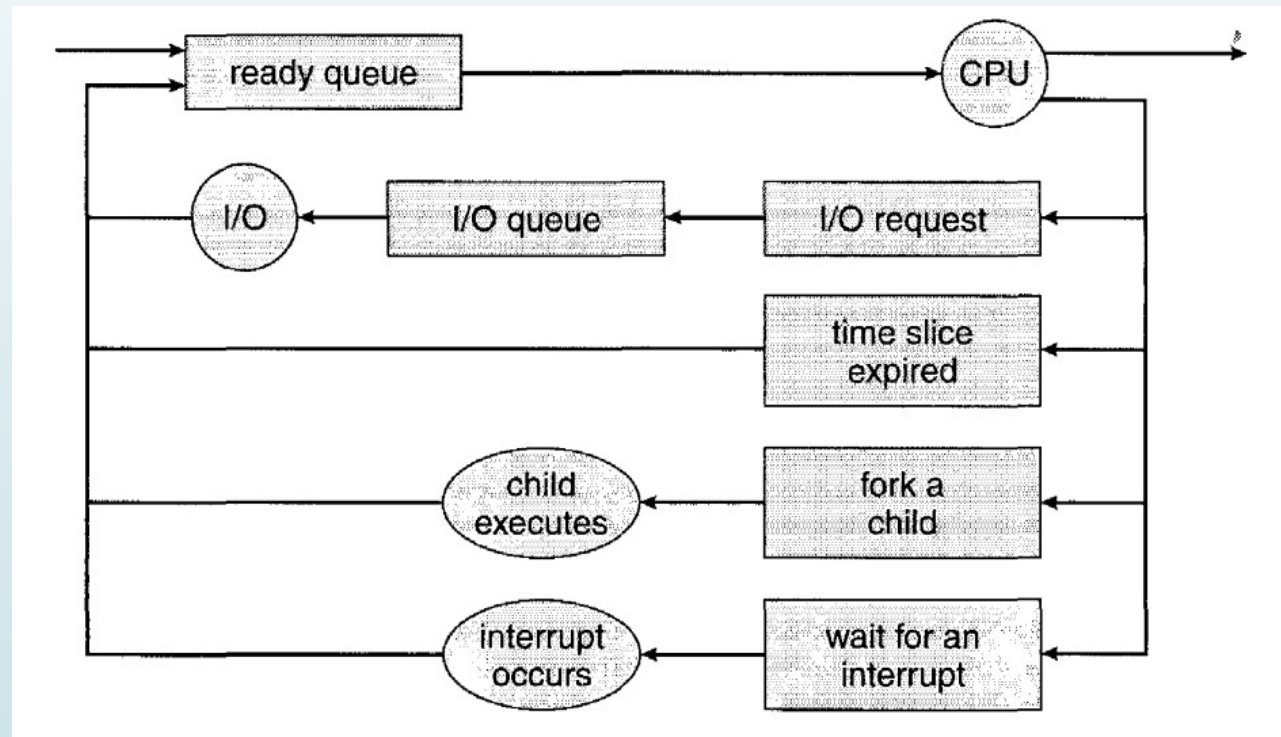
The circles represent the resources that serve the queues, and the arrows indicate the flow of processes in the system.

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

Queuing diagram



Types of schedulers

Types of schedulers:

There are 3 types of schedulers mainly used:

1. Long term scheduler:

Long term scheduler selects process from the disk & loads them into memory for execution.

It controls the degree of multi-programming i.e. no. of processes in memory.

It executes less frequently than other schedulers. So, the long term scheduler is needed to be invoked only when a process leaves the system.

Most processes in the CPU are either I/O bound or CPU bound.

An I/O bound process is one that spends most of its time in I/O operation than it spends in doing computing operation.

A CPU bound process is one that spends more of its time in doing computations than I/O operations.

It is important that the long term scheduler should select a



Types of schedulers

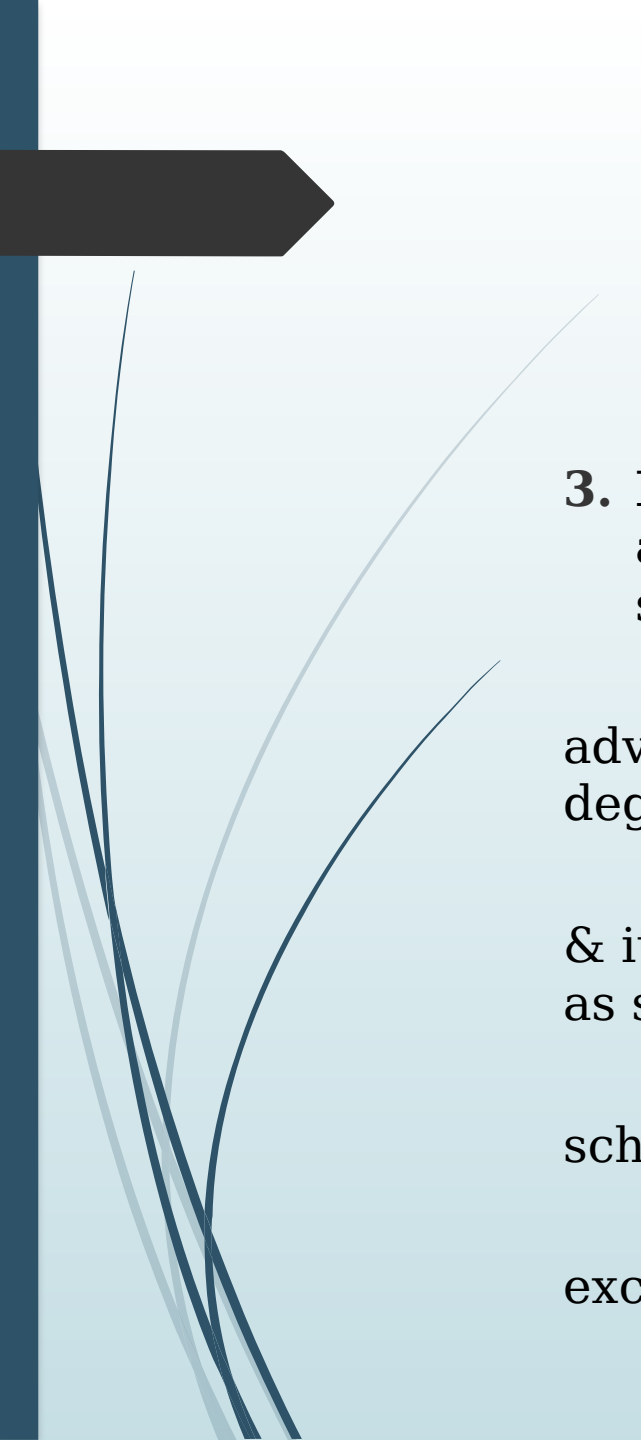
2. Short - term scheduler:

The short term scheduler selects among the process that are ready to execute & allocates the CPU to one of them.

The primary distinction between these two schedulers is the frequency of their execution.

The short-term scheduler must select a new process for the CPU quite frequently. It must execute at least one in 100ms.

Due to the short duration of time between executions, it must be very fast.



3. Medium - term scheduler: some operating systems introduce an additional intermediate level of scheduling known as medium - term scheduler.

The main idea behind this scheduler is that sometimes it is advantageous to remove processes from memory & thus reduce the degree of multiprogramming.

At some later time, the process can be reintroduced into memory & its execution can be continued from where it had left off. This is called as swapping.

The process is swapped out & swapped in later by medium term scheduler.

Swapping is necessary when the available memory limit is exceeded which requires some memory to be freed up.



Context switch

Switching the CPU to another process requires
saving the state of the old process
loading the saved state for the new process.

This task is known as a **Context Switch**.

The **context** of a process is represented in the **Process Control Block(PCB)** of a process;

it includes the value of the CPU registers, the process state and memory-management information.

When a context switch occurs,

- the Kernel saves the context of the old process in its PCB
- loads the saved context of the new process scheduled to run.



Operations on processes

- The processes in most systems can execute concurrently, and they may be created and deleted dynamically.
- Thus, these systems must provide a mechanism for process creation and termination.

Process creation

□ Process Creation

A process may create several new processes, via a create-process system call, during the course of execution. The creating process is called a parent process, and the new processes are called the children of that process, forming a **tree** of processes

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

□ Resource sharing options

- Parent and children share all resources
- Children share subset of parent's resources
- Parent and child share no resources

□ Execution options

- Parent and children execute concurrently
- Parent waits until children terminate



Process termination

Process Termination

A process terminates when it finishes executing its final statement and asks the operating system to delete it by using the **exit()** system call. At that point, the process may return a status value (typically an integer) to its parent process (via the wait() system call).

All the resources of the process—including physical and virtual memory, open files, and

I/O buffers—are deallocated by the operating system.

A parent may terminate the execution of one of its children for a variety of reasons, such as these:

- The child has exceeded its usage of some of the resources that it has been allocated.

- The task assigned to the child is no longer required.

- The parent is exiting, and the operating system does not allow a child to continue if its

- parent terminates.



Cascading termination

Some systems do not allow a child to exist if its parent has terminated. In such systems, if a process terminates (either normally or abnormally), then all its children must also be terminated.

This phenomenon, referred to as **cascading termination**, is normally initiated by the operating system.