Chapter Two: Process Management

Process Description

Modern computers work in a multitasking environment in which they execute multiple programs simultaneously. These programs cooperate with each other and share the same resource, such as memory and cpu. An operating system manages all processes to facilitate proper utilization of these resources.

Important concept: decomposition. Given hard problem, chop it up into several simpler problems that can be solved separately. A big size program should be broken down into **processes** to be executed.

What is a process?

- A running piece of code or a program in execution.
- "An execution program in the context of a particular process state."
- A process includes code and particular data values
- Process state is everything that can affect, or be affected by,
- Only one thing (one state) happens at a time within a process.
- The term process, used somewhat interchangeably with 'task' or 'job'.

Process States:

A process goes through a series of discrete process states.

- **New State:** The process being created.
- **Ready State:** The new process that is waiting to be assigned to the processor.
- **Running State:** A process is said to be running if it using the CPU at that particular instant.
- **Blocked (or waiting) State:** A process is said to be blocked if it is waiting for some event to happen such that as an I/O request before it can proceed.

- **Terminated state:** The process has finished execution.

How processes are created?

Creating a process from scratch (e.g., the Windows/NT uses **CreateProcess()**):

- Load code and data into memory.
- Create (empty) call stack.
- Create and initialize process control block.
- Make process known to dispatcher/ scheduler.

Forking: want to make a copy of existing process (e.g., UNIX uses **fork ( )** function).

- Make sure process to be copied is not running and has all state saved.
- Make a copy of code, data, and stack.
- Copy PCB of source into new process.
- Make process known to dispatcher/scheduler.

A process in operating system is represented by a data structure known as **a Process Control Block (PCB)** or process descriptor. Process information has to keep track in PCB. For each process, PCB holds information related to that process.

The PCB contains important information about the specific process including:

- The current state of the process i.e., whether it is ready, running, waiting, or whatever.
- Unique identification of the process in order to track "which is which" information.
- A pointer to parent process.
- Similarly, a pointer to child process (if it exists).
- The priority of process (a part of CPU scheduling information).
- Pointers to locate memory of processes.
- A register save area.
- The process it is running on.

The PCB is a certain **store** that allows the operating systems to locate key information about a

process. Thus, the PCB is the data structure that defines a process to the operating systems.

Process table: collection of all process control blocks for all processes.

How can several processes share one CPU? OS must make sure that processes do not interfere with each other. This means

- Making sure each gets a chance to run inside the cpu (fair scheduling).
- Making sure they do not modify each other's state (protection).

Context Switching

Most modern computer systems allow more than one process to be executed simultaneously. This is called ***Multitasking* systems**.

Context switching means switching the cpu to different processes. It requires saving of the state of the current process into the PCB and load the saved PCB of the previous or new process.

How does cpu switch between different processes?

Dispatcher (also called Short Term **Scheduler**): inner-most portion of the OS that runs processes without interference: Scheduler supports and facilitates the following activities

- Run process for a while
- Save its state
- Load state of another process
- Run the new process and after some time it reload the suspended /previous process.

OS needs some agent to facilitate the state saving and restoring code.

- All machines provide some special hardware support for saving and restoring state:

Process Scheduling:

When more than one process is running, the operating system must decide which one should first run. The part of the operating system concerned with this decision is called the ***scheduler***, and algorithm it uses is called the ***scheduling algorithm***.

In multitasking and uniprocessor system scheduling is needed because more than one process is in ready and waiting state. A certain scheduling algorithm is used to get all the processes to run

correctly. The processes should be in such a manner that no processes must be made to wait for a long time.

Goals of scheduling (objectives):

What the scheduler try to achieve?

General Goals

Fairness:

Fairness is important under all circumstances. A scheduler makes sure that each **process gets its fair share of the CPU** and no process can suffer indefinite postponement/delay. Not giving equivalent or equal time is not fair.

Efficiency:

Scheduler should **keep the system (or in particular CPU) busy 100%** of the time when possible. If the CPU and all the Input/Output devices can be kept running all the time, more work gets done per second than if some components are idle.

Response Time:

A scheduler should minimize the response time for interactive user.

Turnaround:

A scheduler should **minimize** the time batch users must wait for an output of executing process. It is **Cpu time + Wait time**

Throughput:

A scheduler should **maximize** the number of jobs processed per unit time.

Preemptive Vs Non preemptive Scheduling:

The Scheduling algorithms can be divided into two categories with respect to how they deal with interrupts.

Non preemptive Scheduling:

A scheduling discipline is non preemptive if once a process has been given the CPU, the CPU cannot be taken away from that process until the assigned process completes its execution.

Following are some characteristics of non preemptive scheduling

1. In non preemptive system, short jobs are made to wait by longer jobs..
2. In non preemptive system, response times are more predictable, maximum, because incoming high priority jobs cannot displace waiting jobs.
3. In non preemptive scheduling, a scheduler executes jobs in the following two situations.
    a. When a process switches from running state to the waiting state.
    b. When a process terminates.

Preemptive Scheduling:

The strategy of allowing processes that are logically running to be temporarily suspended is called Preemptive Scheduling and it is contrast to the "run to completion" method.

Scheduling Algorithms

CPU Scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated the CPU.

The following are some scheduling algorithms:

- First Come First Served (FCFS) Scheduling.
- Shortest Job First (SJF) Scheduling.
- Round Robin Scheduling.
- Priority Scheduling.

First-Come-First-Served (FCFS) Scheduling

Other names of this algorithm are:

- First-In-First-Out (FIFO)
- Run-to-Completion
- Run-Until-Done

First-Come-First-Served algorithm is the simplest scheduling algorithm. Processes are dispatched according to their arrival time on the ready queue. Being a non preemptive discipline, once a process gets a CPU, it runs to completion. The FCFS scheduling is fair in the formal sense

or human sense of fairness but it is unfair in the sense that long jobs make short jobs wait a long period.

FCFS scheme is not useful in scheduling interactive users because it cannot guarantee good response time. The code for FCFS scheduling is simple to write and understand. One of the major drawbacks of this scheme is that the average time is often quite long.

Example:

Shortest-Job-First (SJF) Scheduling

Other name of this algorithm is Shortest-Process-Next (SPN).

Shortest-Job-First (SJF) is a non-preemptive discipline in which waiting job (or process) with the smallest estimated run-time-to-completion is run first. In other words, when CPU is available, it is assigned to the process that has smallest next CPU burst.

The SJF algorithm favors for short jobs (or processes) at the expense of longer ones. Since the SJF scheduling algorithm gives the minimum average time for a given set of processes, it is probably optimal.

Example:

Round Robin Scheduling

One of the simplest, fairest and most widely used algorithms is round robin (RR).

In the round robin scheduling, processes are dispatched in a FIFO manner but are given a limited amount of CPU time called a time-slice or a quantum.

If a process does not complete before its CPU-time expires, the CPU is preempted/interrupted and given to the next process waiting in a queue. The preempted process is then placed at the back of the ready list.

Round Robin Scheduling is preemptive (at the end of time-slice) therefore it is effective in

time-sharing environments in which the system needs to guarantee reasonable response times for interactive users.

The only interesting issue with round robin scheme is the length of the quantum. Setting the quantum too short causes too many context switches and lower the CPU efficiency. On the other hand, setting the quantum too long may cause poor response time and approximates FCFS. So the slice time should not be too short and too long.

Example:

Priority Scheduling

The basic idea is straightforward: each process is assigned a priority, and priority is allowed to run. Equal-Priority processes are scheduled in FCFS order or in RR. The shortest-Job-First (SJF) algorithm is a special case of general priority scheduling algorithm.

SJF algorithm is also a priority algorithm where the priority is the inverse of the (predicted) next CPU burst. That is, the longer the CPU burst, the lower the priority and vice versa

Example:

Inter process Communication (IPC)

In the case of uniprocessor systems, processes interact on the bases of the degree to which they are aware of each other's existence.

1. Processes unaware of each other: The processes are not working together; the OS needs to be concerned about competition for resources (cpu or memory).
2. Processes directly aware each other: Processes that are designed to work jointly on the same program or application. These processes exhibit cooperation but competition for shared variables (global variables).

One of the benefits of multitasking is that several processes can be made to cooperate in order to achieve their ends. To do this, they must do one of the following.

Communicate: Inter-process communication (IPC) involves sending information from one process to another, since the output of one process may be an input for the other process.

Share data: A segment of memory must be available to both processes.

Waiting: Some processes wait for other processes to give a signal before continuing. All these are issues of synchronization/communication.

Since processes frequently need to communicate with other processes therefore, there is a need

for a well-structured communication. And the interaction or communication between two or more processes is termed as **Interprocess communication (IPC).**Information sharing between processes or IPC achieved through shared variable ,by passing through shared file and by using OS services provided for this purpose like OS signals(system calls like interrupts).

When processes co-operate each other's there is a problem of how to synchronize cooperating processes. For example, suppose two processes modify the same file. If both processes tried to write simultaneously the result would be a senseless mixture. We must have a way of synchronizing processes, so that even concurrent processes must stand in line to access shared data *serially* without any interference or competition or racing.