PROCESS MANAGEMENT

Process concept

It is important to note that before a processor executes a program, a process must be created. In many instances, several processes can be running in one single program. Therefore, the Operating System takes the initiative of;

- Creation and deletion of user and system processes
- Suspension and resumption of a process
- Process synchronization
- Process communication
- Handling of deadlocks among processes

Reasons which lead to creation of a process

- When user logs in a system, process is created
- When a user starts a program
- Where the operating system must create a process to provide services such as printing services
- A process starts another process

What is a process?

- A program in execution. NB: a program is a static entity made up of program statements.
- An asynchronous activity
- An entity to which the processor is assigned
- A dispatch-able unit

What does a process include?

It includes;

- The current value of the program counter (PC)
- Contents of processor's registers
- Value of variables
- Process stacks which contain temporary data such as subroutine parameters, return addresses and temporary variables
- A data section which contain global variables

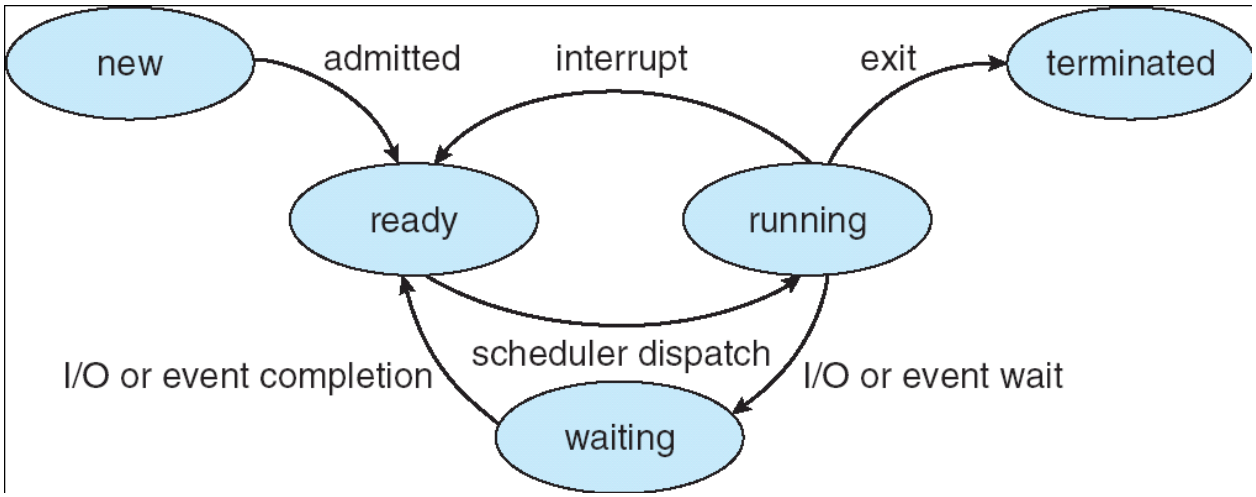
Process lifecycle/states/transition

Once a process is created it might consist of several states depending on what resources it requires and/or CPU scheduling algorithm used. These states include;

1. BLOCKED: A blocked process is the one waiting for some event to occur before it continues execution. This event could be an I/O operation. A process in this state cannot proceed until the blocking event is completed.
2. READY: A process that is not now allocated the CPU is said to be ready to run.
   RUNNING: This is the process that is executing in the CPU.
3. SWAPPED BLOCKED: A process that has been swapped to secondary storage but waiting for an event to occur.

4. SWAPPED READY: A process that is swapped to secondary storage but is ready to run and has not been allocated the main memory
5. TERMINATED: This is a process whose execution has been halted but a record of its execution is still maintained by the Operating System. It is also referred to as zombie process.

All the above states are called process transition and can been illustrated as show below



Process transition

Terminated process

There are several reasons why operating system may want to retain a record of a terminated process.

- The process may have a pending incomplete I/O operation.
- It may have a child process that is still executing which may need to retain information of the process that created it.
- It may have a parent or related process that in future may request information about terminated process.

NB: a process may be described as **I/O bound process** meaning that the process spends more time doing I/O Operations or it may be **CPU bound process** to mean a process that spends more time doing computations in the processor.

A process may be terminated for the following reasons

- Because the process have done its job
- If it discovers a fatal error such as when it tries to compile a program that does not exist
- If it is killed by another process
- If a process executes the last statement
- If resources are re-allocated by the OS
- If a process terminates execution of a child process

- A child has exceeded its allocated resources
- A task assigned to a child is no longer required
- When the OS does not allow a child to continue because its parent has terminated

CPU Scheduling

In a multiprogramming environment, scheduling is used to determine which process is going to be given control of CPU. Scheduling is divided into three phases:

- Long term scheduling
- Medium term scheduling
- Short term scheduling

During long term scheduling, the long term scheduler determines which process (es) must be brought or admitted into the ready queue. In short term scheduling, the short term scheduler select which process should be executed next and allocate CPU. Medium term scheduler swaps processes in and out of memory.

Objectives of scheduling criteria

1. Fairness: the scheduler makes sure that each process must get a fair share of the CPU and therefore, no process is allowed to suffer indefinite postponement.
2. Policy enforcement: the scheduler makes sure that the system policy is enforced.
3. Efficiency: the scheduler must keep the CPU 100% busy all the time if possible. As a result, more work gets done.
4. Turn around: scheduler should minimize the time batch users must wait to receive an output.
5. Throughput: a scheduler must maximize the number of jobs processed per unit time.

CPU Scheduling Algorithms

In early types of operating systems, scheduling was non-preemptive where a process retained control of the CPU until a process blocked or terminated but in the modern OS scheduling is preemptive where a process can be blocked or terminated by the scheduler in order to allocate the CPU to another process. Therefore, there are six scheduling algorithms that have been widely used and they include:

1. First Come First Served (FCFS)
2. Shortest Jobs First (SJF)
3. Shortest Remaining Time (SRT)
4. Round Robin (RR)
5. Priority
6. Multilevel Feedback Queue (MFQ)

1. First Come First Served (FCFS)

This is also known as First in First out (FIFO), Run to completion or Run until done. In this scheduling algorithm, processes are dispatched according to their arrival time on the ready queue. FCFS is a non-preemptive discipline where once a process has been given CPU it runs to completion. FCFS is fair in formal sense of human fairness but it is unfair in the sense that long jobs make short jobs to wait for long time. It is the most predictable of the other scheduling algorithms. However, it is not useful in scheduling interacting users because it cannot guarantee good response time. It is rarely used in modern OS but it's the simplest of all the scheduling algorithms.

2. Shortest Job First (SJF)

This is a non-preemptive discipline in which waiting jobs with the smallest estimated runtime to completion is run next when the CPU is available, i.e. the CPU is assigned to the process that has the smallest next CPU burst. This algorithm is appropriate for batch jobs for which runtimes are known in advance. This algorithm favors short jobs at the expense of long jobs. One of the problems with SJF is that it requires precise knowledge of how long a process will run but this information is not usually available and the best SJF can do is to rely on the user estimate of runtime. Similar to FCFS, SJF it is not useful in time sharing environment.

3. Shortest Remaining Time (SRT)

This is the preemptive counterpart of SJF and is useful in timesharing environment. In this algorithm, the process with the smallest estimated runtime to completion is run next including new arrivals. A running process may be preempted by a new arrival process with the shortest estimated runtime. The CPU must keep track of the elapsed time of the running process and must be ready to handle occasional preemptions. In this scheme, arrival of small jobs will run immediately but longer jobs have longer mean waiting time. Compared to SJF, SRT has the highest overhead.

4. Round Robin (RR)

This is the one of the oldest, simplest, fairest, and most widely used algorithm. In RR processes are dispatched in a FIFO manner but are given a limited amount of CPU time called a time slice or quantum. If a process does not complete before CPU time expires, the CPU is preempted and given to the next process waiting. The preempted process is then placed at the back of ready queue. RR is preemptive (at the end of time slice) and therefore, it is effective in time sharing environment in which system needs to guarantee reasonable response time for interactive users. The only issue with RR scheme is the length of a quantum because setting the quantum too short results into many **context switches** and this lowers the CPU efficiency. On the

other hand, setting the quantum too long may cause poor response time but the average waiting under RR is often quite long.

CONTEXT SWITCH: to give each process a fair share of the CPU, a hardware clock generates interrupts periodically. This allows the OS to schedule all processes in main memory to run on the CPU at equal intervals. Each time a clock interrupt occurs, the interrupt handler checks how much time process has used. If it has used up its entire time slice then the CPU scheduling algorithm picks a different process to run. Each switch of CPU from one process to another is called **a context switch**.

5. Priority

In this scheduling, a process is assigned a priority to run where priority is defined either internally or externally. Equal priority processes are scheduled FCFS. Internally defined priorities use some measureable quantities or qualities to compute priority of a process. Some of these priorities could be time limit, memory requirement, file requirement such as number of open files, CPU, or I/O requirements etc. Externally defined priority are set by criteria that is external to OS such as importance of the process, amount of funds being paid for computer usage, department sponsoring the works, politics, etc.

Priority scheduling can either be pre-emptive or non-preemptive. A preemptive priority scheme will preempt the CPU if the priority of the new arrival process is higher than the priority of the currently running process. A non-preemptive priority scheme will simply put a new process at the head of the ready queue. A major problem that arises is indefinite blocking or starvation. A solution to this blocking of low priority process indefinitely is **aging,** which is a technique of gradually increasing the priority of processes that wait in the system for long period of time.

6. Multilevel Feedback Queue (MFQ)

This scheme partitions the ready queue into several separate queues. These queues are then assigned processes based on the properties of process such as memory size, priority and type. The algorithm chooses the process from the occupied queue that has the highest priority and run the processes preemptively or non-preemptively and each queue has its own scheduling policy. If a process use too much of CPU time is moved to a lower priority queue. Similarly, if a process waits too long in a lower priority queue it may be moved to a higher priority queue. In this form of scheduling, starvation is prevented.