Operating System Primer

What is an Operating System?

For users, an operating system allows them to interact with the hardware of a computer. However, for developers an operating system is:

- `Low-level APIs` - Developers can the low-level APIs to interact with the various components of a computer. For instance, a developer can use the `read` and `write` system calls to read and write to a file. You can use the `os` module in Python to interact with the operating system.
- `Resource Management` - An operating system manages the resources of a computer. For instance, it manages the memory of a computer. Multiple processes can run on a computer at the same time. However, each process has its own memory space. The operating system manages the memory of a computer and allocates memory to each process. It also manages the CPU of a computer. It schedules the processes to run on the CPU.

Uniprogramming vs Multiprogramming

An operating system can be uniprogramming or multiprogramming. A uniprogramming operating system can only run one process at a time. A multiprogramming operating system can run multiple processes at a same time.

A uniprogramming operating system is simpler than a multiprogramming operating system. However, a multiprogramming operating system is more efficient than a uniprogramming operating system. A multiprogramming operating system can run multiple processes at a same time. This allows the CPU to be utilized more efficiently. A uniprogramming operating system can only run one process at a time. This means that the CPU is not utilized efficiently.

Examples of uniprogramming operating systems are:

- MS-DOS
- ATM operating systems
- Smart devices (e.g. smart watches)

Types of Multiprogramming Operating Systems

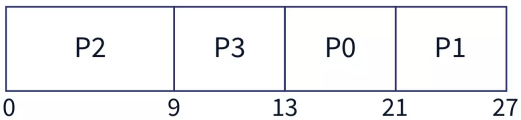
Multiprogramming operating systems can be classified on the basis of the following:

- `Number of users` - A single-user operating system can only run one user at a time. A multi-user operating system can run multiple users at a same time. Personal computers run single-user operating systems. However, servers run multi-user operating systems.
- `Scheduling of processes` - A preemptive operating system can interrupt a process to run another process. A non-preemptive operating system cannot interrupt a process to run another process. A preemptive operating system can run multiple processes at a same time. However, a non-preemptive operating system can only run one process at a time. A non-preemptive operating system is simpler than a preemptive operating system. However, a preemptive operating system is more efficient than a non-preemptive operating system.

Preemptive vs Non-Preemptive

> Non-preemptive Scheduling is used when a process terminates, or a process switches from running to the waiting state. In this scheduling, once the resources (CPU cycles) are allocated to a process, the process holds the CPU till it gets terminated or reaches a waiting state. In the case of non-preemptive scheduling does not interrupt a process running CPU in the middle of the execution. Instead, it waits till the process completes its CPU burst time, and then it can allocate the CPU to another process.

Process	Arrival Time	CPU Burst Time (in millisecond)
P0	2	8
P1	3	6
P2	0	9
P3	1	4



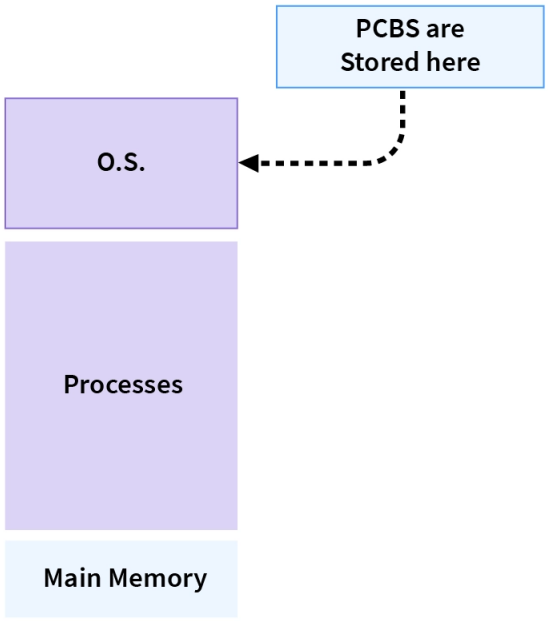
> Preemptive scheduling is used when a process switches from running state to ready state or from the waiting state to ready state. The resources (mainly CPU cycles) are allocated to the process for a limited amount of time and then taken away, and the process is again placed back in the ready queue if that process still has CPU burst time remaining. That process stays in the ready queue till it gets its next chance to execute.

Processes

We install a lot of different applications on are system. An application is a program that we can run on our system. For instance, we can run a web browser, a text editor, etc. Each of these applications is a program. However, each of these applications is a different program. When we install an application, it is installed as a file on our system. When we run the application, the operating system loads the application into memory and runs it. When we run an application, the operating system creates a process for the application.

A process is an instance of a program. A process is a program that is running on our system. When we run an application, the operating system creates a process for the application. When we run multiple applications, the operating system creates multiple processes for the applications. Each process has its own memory space. A process can access the memory of other processes. However, a process cannot access the memory of another process.

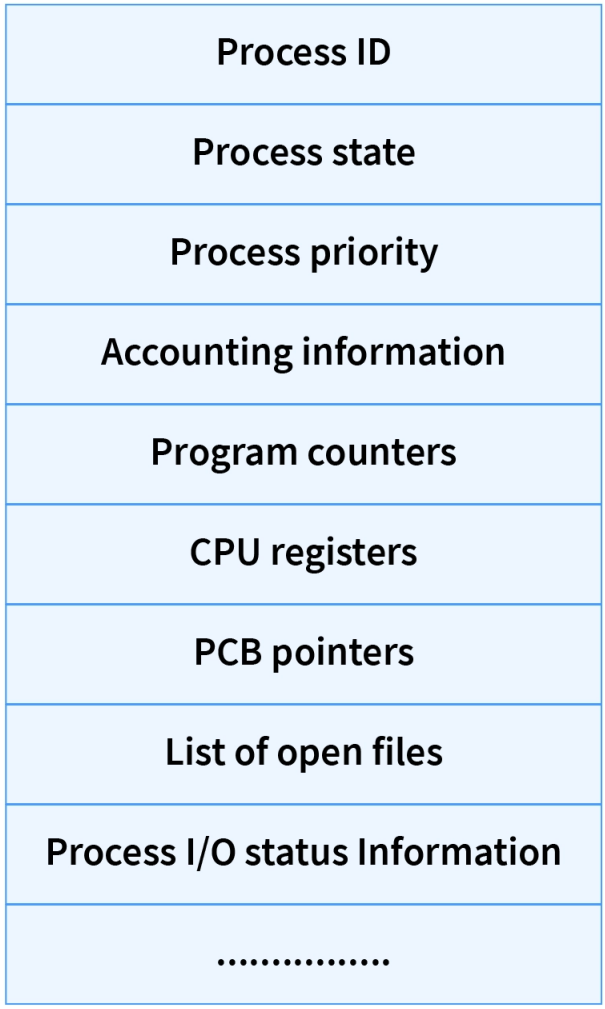
When the process is created by the operating system it creates a data structure to store the information of that process. This is known as Process Control Block (PCB). Process Control block (PCB) is a data structure that stores information of a process.



It's the job of the operating system to assign a CPU to a process as the process doesn't need a CPU all the time. Let's take an example of the input/output process, they are only used by the CPU when triggered.

The role of the process control block arises as an identification card for each process. The Operating System doesn't know which process is which, until Operating System refers through the PCB of every process.

The process control block contains many attributes such as process ID, process state, process priority, accounting information, program counter, CPU registers`, etc for each process.



SCALER Topics

Types of Processes

A process can be classified into the following types:

- `I/O bound process` - An I/O bound process spends most of its time waiting for I/O operations to complete. For instance, a process that reads data from a file spends most of its time waiting for the

data to be read from the file. An I/O bound process spends most of its time waiting for I/O operations to complete. For instance, a process that reads data from a file spends most of its time waiting for the data to be read from the file.

- `CPU bound process` - A CPU bound process spends most of its time executing instructions. For instance, a process that performs a lot of computations spends most of its time executing instructions. A CPU bound process spends most of its time executing instructions. For instance, a process that performs a lot of computations spends most of its time executing instructions.

Scheduling Algorithms

Why do we need Scheduling Algorithms?

A process to complete its execution needs both CPU time and I/O time. In a multiprogramming system, there can be one process using CPU while another is waiting for I/O whereas, in a uni programming system, time spent waiting for I/O is completely wasted as CPU is idle at this time. The multiprogramming can be achieved by the use of process scheduling.

The objectives of a scheduling algorithm are as follows:

- Maximize the CPU utilization, meaning that keep the CPU as busy as possible.
- Fair allocation of CPU time to every process
- Maximize the Throughput
- Minimize the turnaround time
- Minimize the waiting time
- Minimize the response time

To achieve these objectives, we need to have a scheduling algorithm. A scheduling algorithm is an algorithm that decides which process to run next. To demonstrate the various scheduling algorithms, we will use the following processes:

Process	Arrival Time	Burst Time
P1	2	6
P2	1	8
P3	0	3
P4	4	4

First Come First Serve (FCFS)

First Come First Serve (FCFS) is a non-preemptive scheduling algorithm. In this algorithm, the process that comes first is executed first. If two processes arrive at the same time, then the process that comes first in the ready queue is executed first. This can be decided on the basis of the process ID or the burst time.

Let us take the example of the above processes. The process P3 arrives first, so it is executed first. The process P2 arrives next, so it is executed next. The process P1 arrives next, so it is executed next. The process P4 arrives last, so it is executed last.

At time t=0

At time t=0, the process P3 arrives. So, it is executed first. The process P3 is executed for 3 time units. The process P3 is completed at time t=3.

At time t=1

At time t=1, the process P2 arrives, but it cannot be executed as the process P3 is still executing. So, the process P2 is added to the ready queue.

At time t=2

At time t=2, the process P1 arrives, but it cannot be executed as the process P3 is still executing. So, the process P1 is added to the ready queue. Now, there are two processes in the ready queue, P2 and P1.

At time t=3

Process P3 is completed at time t=3. So, the process P2 is executed next. The process P2 is executed for 8 time units. The process P2 is completed at time t=11.

At time t=4

At time t=4, the process P4 arrives, but it cannot be executed as the process P2 is still executing. So, the process P4 is added to the ready queue. Now, there are two processes in the ready queue, P1, and P4.

At time t=11

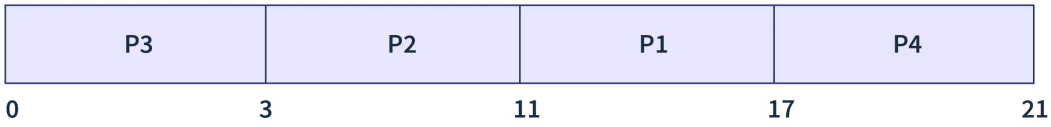
Process P2 is completed at time t=11. So, the process P1 is executed next. The process P1 is executed for 6 time units. The process P1 is completed at time t=17.

At time t=17

Process P1 is completed at time t=17. So, the process P4 is executed next. The process P4 is executed for 4 time units. The process P4 is completed at time t=21.

At time t=21

Process P4 is completed at time t=21. So, there are no more processes to execute.



Advantages

- Involves no complex logic and just picks processes from the ready queue one by one.
- Easy to implement and understand.
- Every process will eventually get a chance to run so no starvation occurs.

Disadvantages

- Waiting time for processes with less execution time is often very long.
- It favors CPU-bound processes then I/O processes.
- Leads to convoy effect.
- Causes lower device and CPU utilization.
- Poor in performance as the average wait time is high.

Shortest Remaining Time First (SRTF)

Shortest Remaining Time First (SRTF) is a preemptive scheduling algorithm. In this algorithm, the process that has the least remaining time is executed next. If two processes have the same remaining time, then the process that comes first in the ready queue is executed next. This can be decided on the basis of the process ID or the burst time.

The algorithm runs whenever a process is completed or a new process arrives. The algorithm is as follows:

- If the ready queue is empty, then the CPU remains idle.
- A new process arrives and the ready queue is empty. The process is added to the ready queue.
- The algorithm checks if the remaining time of the process in the CPU is less than the remaining time of the process in the ready queue. If yes, then the process in the CPU continues to execute. If no, then the process in the CPU is preempted and the process in the ready queue is executed.

Let us take the example of the above processes.

At time t=0

At time t=0, the process P3 arrives. So, it is executed first. The process P3 is executed for either 3 time units or until a new process arrives.

At time t=1

Process	Arrival Time	Burst Time	Remaining Time
P3	0	3	2
P2	1	8	8

At time t=1, the process P2 arrives. The process P2 is added to the ready queue. The algorithm checks if the remaining time of the process in the CPU is less than the remaining time of the process in the ready queue. The remaining time of the process in the CPU is 2 and the remaining time of the process in the ready queue is 8. So, the process in the CPU continues to execute.

At time t=2

Process	Arrival Time	Burst Time	Remaining Time
P3	0	3	1
P2	1	8	8
P1	2	6	6

At time t=2, the process P1 arrives. The process P1 is added to the ready queue. The algorithm checks if the remaining time of the process in the CPU is less than the remaining time of the process in the ready queue. The remaining time of the process in the CPU is 1 and the remaining time of the processes in the ready queue is 8 and 6.

At time t=3

Process	Arrival Time	Burst Time	Remaining Time
P3	0	3	0
P2	1	8	8
P1	2	6	6

At time t=3, the process P3 is completed, and the algorithm compares the remaining time of the processes in the ready queue. The remaining time of the processes in the ready queue is 8 and 6. So, the process P1 is executed next. The process P1 will execute for 8 time units unless a new process arrives.

At time t=4

Process	Arrival Time	Burst Time	Remaining Time
P2	1	8	8

Process	Arrival Time	Burst Time	Remaining Time
P1	2	6	5
P4	3	4	4

At time t=4, the process P4 arrives. The process P4 is added to the ready queue. The algorithm checks if the remaining time of the process in the CPU is less than the remaining time of the process in the ready queue. The remaining time of the process in the CPU is 5 and the remaining time of the processes in the ready queue is 8 and 4. So, the process in the CPU is preempted and the process in the ready queue is executed. The process P4 is executed for 4 time units unless a new process arrives and process P1 is added to the ready queue.

At time t=8

Process	Arrival Time	Burst Time	Remaining Time
P2	1	8	8
P1	2	6	5
P4	3	4	0

At time t=8, the process P4 is completed. The algorithm finds the process with the least remaining time. The remaining time of the processes in the ready queue is 8 and 5. So, the process P1 is executed next. The process P1 is executed for 5 time units unless a new process arrives.

At time t=13

Process	Arrival Time	Burst Time	Remaining Time
P2	1	8	8
P1	2	6	0

At time t=13, the process P1 is completed. So, the process P2 is executed next.

At time t=21

Process	Arrival Time	Burst Time	Remaining Time
P2	1	8	0

At time t=21, the process P2 is completed. So, there are no more processes to execute.

Advantages

- Processes are executed faster than SJF, being the preemptive version of it.

Disadvantages

- Context switching is done a lot more times and adds to the more overhead time.
- It may still lead to starvation and requires the knowledge of process time beforehand.

- Impossible to implement in interactive systems where the required CPU time is unknown.

References

- What is an OS?
- Scheduling Algorithms
- First Come First Serve (FCFS) Scheduling Algorithm