Analysis of CPU Scheduling Variants

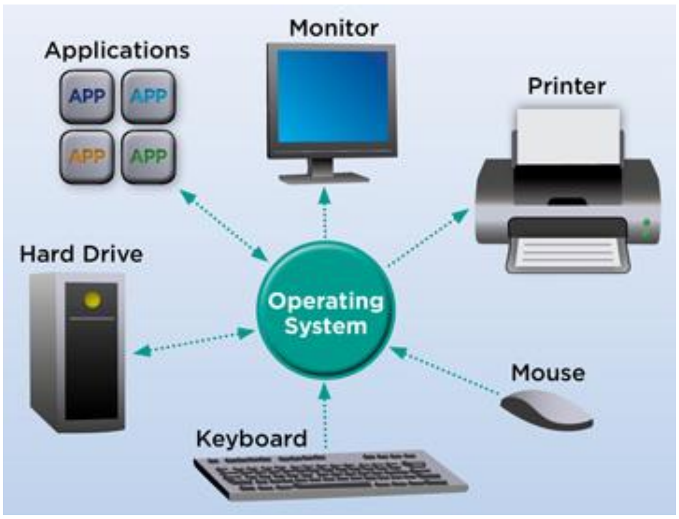
Hima Mohan
Asst.Professor on Contract,
Carmel College Mala,

***Abstract*— An operating system is a type of system software that manages and controls the resources and computing capability of a computer, and provides users a logical interface for accessing the computer to execute applications. Scheduling is a fundamental operating-system function. Almost all programs have some alternating cycle of CPU number crunching and waiting for I/O of some kind. A scheduling system allows one process to use the CPU while another is waiting for I/O, thereby making full use of CPU. The challenge is to make the overall system as efficient and fair as possible. A typical process involves both I/O time and CPU time. In a uniprogramming system like MS-DOS, time spent waiting for I/O is wasted and CPU is free during this time. In multi programming systems, one process can use CPU while another is waiting for I/O. This is possible only with process scheduling. There are several CPU scheduling algorithms but all have certain merits and limitations. CPU scheduling objectives i.e; average waiting time, average turnaround time, average CPU utilization and average throughput are discussed. These will form the base parameters in making a decision for the suitability of the given algorithm for a given objective.**

***Index Terms*— Operating system, scheduling criteria, scheduling algorithm**

I. INTRODUCTION

An operating system (OS) is software which acts as an interface between the end user and computer hardware. Every computer must have at least one OS to run other programs. The OS helps you to communicate with the computer without knowing how to speak the computer's language. Computer operating systems perform basic tasks, such as recognizing input from the keyboard, sending output to the display screen, keeping track of files and directories on the storage drives, and controlling peripheral devices, such as printers. It is not possible for the user to use any computer or mobile device without having an operating system.

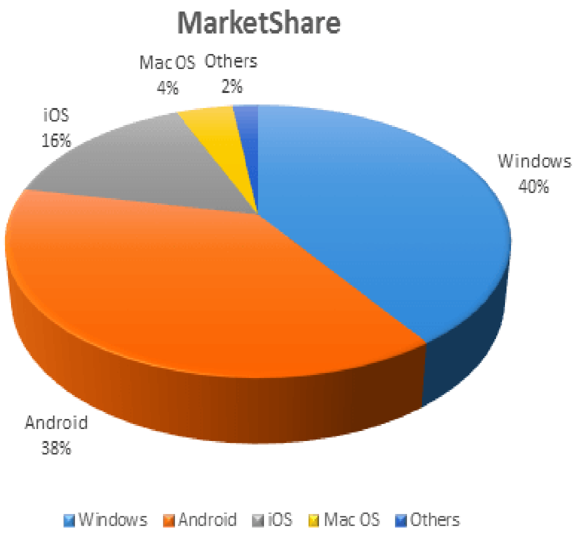


Objectives of Operating System

- To make the computer system convenient to use in an efficient manner.
- To hide the details of the hardware resources from the users.
- To provide users a convenient interface to use the computer system.
- To act as an intermediary between the hardware and its users, making it easier for the users to access and use other resources.
- To manage the resources of a computer system.
- To keep track of who is using which resource, granting resource requests, and mediating conflicting requests from different programs and users.
- To provide efficient and fair sharing of resources among users and programs.

Characteristics of Operating System

- Memory Management − Keeps track of the primary memory, i.e. what part of it is in use by whom, what part is not in use, etc. and allocates the memory when a process or program requests it.
- Processor Management − Allocates the processor (CPU) to a process and deallocates the processor when it is no longer required.
- Device Management − Keeps track of all the devices. This is also called I/O controller that decides which process gets the device, when, and for how much time.
- File Management − Allocates and de-allocates the resources and decides who gets the resources.
- Security − Prevents unauthorized access to programs and data by means of passwords and other similar techniques.
- Job Accounting − Keeps track of time and resources used by various jobs and/or users.
- Control Over System Performance − Records delays between the request for a service and from the system.
- Interaction with the Operators − Interaction may take place via the console of the computer in the form of instructions. The Operating System acknowledges the same, does the corresponding action, and informs the operation by a display screen.
- Error-detecting Aids − Production of dumps, traces, error messages, and other debugging and error-detecting methods.
- Coordination Between Other Software and Users − Coordination and assignment of compilers, interpreters, assemblers, and other software to the various users of the computer systems.



II. CPU SCHEDULING CRITERIA

There are many different criteria to check when considering the **"best"** scheduling algorithm, they are:

a) CPU Utilization

To make out the best use of CPU and not to waste any CPU cycle, CPU would be working most of the time(Ideally 100% of the time). Considering a real system, CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)

b) Throughput

It is the total number of processes completed per unit time or rather say total amount of work done in a unit of time. This may range from 10/second to 1/hour depending on the specific processes.

c) Turnaround Time

It is the amount of time taken to execute a particular process, i.e. The interval from time of submission of the process to the time of completion of the process(Wall clock time).

d) Waiting Time

The sum of the periods spent waiting in the ready queue amount of time a process has been waiting in the ready queue to acquire get control on the CPU.

e) Load Average

It is the average number of processes residing in the ready queue waiting for their turn to get into the CPU.

f) Response Time

Amount of time it takes from when a request was submitted until the first response is produced. Remember, it is the time till the first response and not the completion of process execution(final response).

In general CPU utilization and Throughput are maximized and other factors are reduced for proper optimization.

III. CPU SCHEDULING ALGORITHMS

1. FIRST COME FIRST SERVE SCHEDULING

In the "First come first serve" scheduling algorithm, as the name suggests, the process which arrives first, gets executed first, or we can say that the process which requests the CPU first, gets the CPU allocated first.

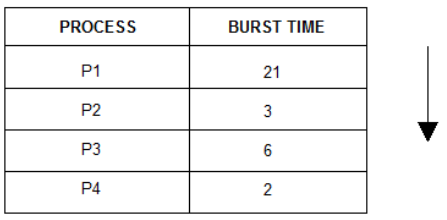
- First Come First Serve, is just like FIFO(First in First out) Queue data structure, where the data element which is added to the queue first, is the one who leaves the queue first.
- This is used in Batch Systems.
- It's easy to understand and implement programmatically using a Queue data structure, where a new process enters through the tail of the queue, and the scheduler selects process from the head of the queue.
- A perfect real life example of FCFS scheduling is buying tickets at ticket counter.

Calculating Average Waiting Time

For every scheduling algorithm, Average waiting time is a crucial parameter to judge it's performance.

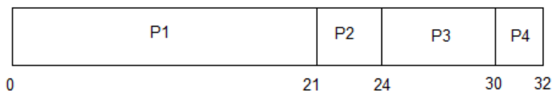
AWT or Average waiting time is the average of the waiting times of the processes in the queue, waiting for the scheduler to pick them for execution. Lower the Average Waiting Time, better the scheduling algorithm.

Consider the processes P1, P2, P3, P4 given in the below table, arrives for execution in the same order, with Arrival Time 0, and given Burst Time, let's find the average waiting time using the FCFS scheduling algorithm.



PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

The average waiting time will be = (0 + 21 + 24 + 30)/4 = 18.75 ms



This is the GANTT chart for the above processes

The average waiting time will be 18.75 ms

For the above given proccesses, first **P1** will be provided with the CPU resources,

- Hence, waiting time for **P1** will be 0
- **P1** requires 21 ms for completion, hence waiting time for **P2** will be 21 ms
- Similarly, waiting time for process **P3** will be execution time of **P1** + execution time for **P2**, which will be (21 + 3) ms = 24 ms.
- For process **P4** it will be the sum of execution times of **P1**, **P2** and **P3**.

Troubles with FCFS Scheduling

Below we have a few shortcomings or problems with the FCFS scheduling algorithm:

1. It is Non Pre-emptive algorithm, which means the process priority doesn't matter.
   If a process with very least priority is being executed, more like daily routine backup process, which takes more time, and all of a sudden some other high priority process arrives, like interrupt to avoid system crash, the high priority process will have to wait, and hence in this case, the system will crash, just because of improper process scheduling.
2. Not optimal Average Waiting Time.
3. Resources utilization in parallel is not possible, which leads to Convoy Effect, and hence poor resource(CPU, I/O etc) utilization.

Convoy Effect: Convoy Effect is a situation where many processes, which need to use a resource for short time, are blocked by one process holding that resource for a long time. This essentially leads to poor utilization of resources and hence poor performance

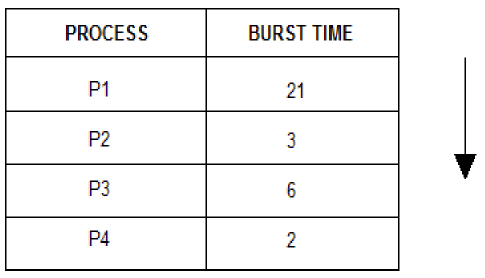
2. SHORTEST JOB FIRST(SJF) SCHEDULING

Shortest Job First scheduling works on the process with the shortest burst time or duration first.

- This is the best approach to minimize waiting time.
- This is used in Batch Systems.
- It is of two types:
  1. Non Pre-emptive
  2. Pre-emptive
- To successfully implement it, the burst time/duration time of the processes should be known to the processor in advance, which is practically not feasible all the time.
- This scheduling algorithm is optimal if all the jobs/processes are available at the same time. (either Arrival time is 0 for all, or Arrival time is same for all)

a. Non Pre-emptive Shortest Job First

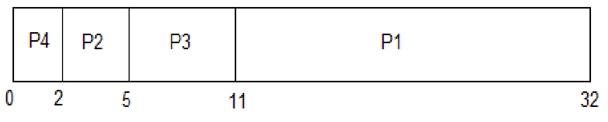
Consider the below processes available in the ready queue for execution, with arrival time as 0 for all and given burst times.



PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

In Shortest Job First Scheduling, the shortest Process is executed first.

Hence the GANTT chart will be following :



Now, the average waiting time will be = (0 + 2 + 5 + 11)/4 = 4.5 ms

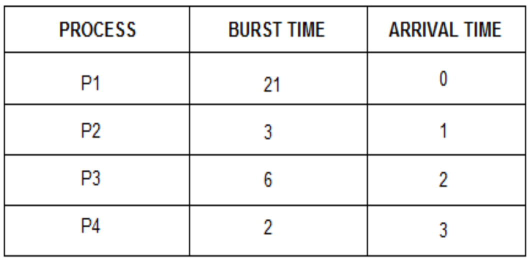
As you can see in the GANTT chart above, the process P4 will be picked up first as it has the shortest burst time, then P2, followed by P3 and at last P1. We scheduled the same set of processes using the First come first serve algorithm in the previous tutorial, and got average waiting time to be 18.75 ms, whereas with SJF, the average waiting time comes out 4.5 ms.

Problem with Non Pre-emptive SJF

If the arrival time for processes are different, which means all the processes are not available in the ready queue at time 0, and some jobs arrive after some time, in such situation, sometimes process with short burst time have to wait for the current process's execution to finish, because in Non Pre-emptive SJF, on arrival of a process with short duration, the existing job/process's execution is not halted/stopped to execute the short job first.
This leads to the problem of Starvation, where a shorter process has to wait for a long time until the current longer process gets executed. This happens if shorter jobs keep coming, but this can be solved using the concept of aging.

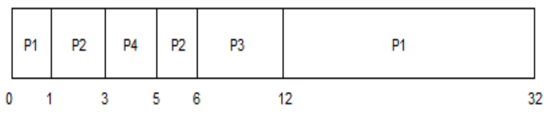
b. Pre-emptive Shortest Job First

In Preemptive Shortest Job First Scheduling, jobs are put into ready queue as they arrive, but as a process with short burst time arrives, the existing process is preempted or removed from execution, and the shorter job is executed first.



PROCESS	BURST TIME	ARRIVAL TIME
P1	21	0
P2	3	1
P3	6	2
P4	2	3

The GANTT chart for Preemptive Shortest Job First Scheduling will be,



The average waiting time will be, ((5-3) + (6-2) + (12-1))/4 = 4.25 ms

The average waiting time for preemptive shortest job first scheduling is less than both, non-preemptive SJF scheduling and FCFS scheduling.

As you can see in the GANTT chart above, as P1 arrives first, hence it's execution starts immediately, but just after 1 ms, process P2 arrives with a burst time of 3 ms which is less than the burst time of P1, hence the process P1(1 ms done, 20 ms left) is preempted and process P2 is executed.
As P2 is getting executed, after 1 ms, P3 arrives, but it has a burst time greater than that of P2, hence execution of P2 continues. But after another millisecond, P4 arrives with a burst time of 2 ms, as a result P2(2 ms done, 1 ms left) is preempted and P4 is executed.
After the completion of P4, process P2 is picked up and finishes, then P2 will get executed and at last P1.
The Pre-emptive SJF is also known as Shortest Remaining Time First, because at any given point of time, the job with the shortest remaining time is executed first.

3. PRIORITY CPU SCHEDULING

In the Shortest Job First scheduling algorithm, the priority of a process is generally the inverse of the CPU burst time, i.e. the larger the burst time the lower is the priority of that process.
In case of priority scheduling the priority is not always set as the inverse of the CPU burst time, rather it can be internally or externally set, but yes the scheduling is done on the basis of priority of the process where the process which is most urgent is processed first, followed by the ones with lesser priority in order.
Processes with same priority are executed in FCFS manner.
The priority of process, when internally defined, can be decided based on memory requirements, time limits, number of open files, ratio of I/O burst to CPU burst etc.
Whereas, external priorities are set based on criteria outside the operating system, like the importance of the process, funds paid for the computer resource use, market factor etc.
Priority scheduling can be of two types:

a. Preemptive Priority Scheduling

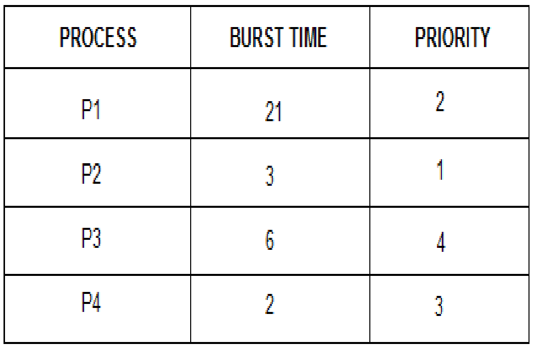
If the new process arrived at the ready queue has a higher priority than the currently running process, the CPU is preempted, which means the processing of the current process is stopped and the incoming new process with higher priority gets the CPU for its execution.

b. Non-Preemptive Priority Scheduling

In case of non-preemptive priority scheduling algorithm if a new process arrives with a higher priority than the current running process, the incoming process is put at the head of the ready queue, which means after the execution of the current process it will be processed.

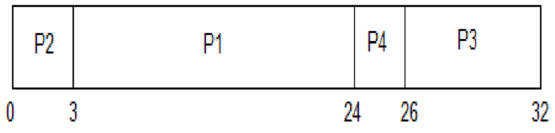
Example of Priority Scheduling Algorithm

Consider the below table fo processes with their respective CPU burst times and the priorities.



PROCESS	BURST TIME	PRIORITY
P1	21	2
P2	3	1
P3	6	4
P4	2	3

The GANTT chart for following processes based on Priority scheduling will be,



The average waiting time will be, (0 + 3 + 24 + 26)/4 = 13.25 ms

As you can see in the GANTT chart that the processes are given CPU time just on the basis of the priorities.

Problem with Priority Scheduling Algorithm

In priority scheduling algorithm, the chances of indefinite blocking or starvation is high. A process is considered blocked when it is ready to run but has to wait for the CPU as some other process is running currently.
But in case of priority scheduling if new higher priority processes keeps coming in the ready queue then the processes waiting in the ready queue with lower priority may have to wait for long durations before getting the CPU for execution.

Using Aging Technique with Priority Scheduling

To prevent starvation of any process, we can use the concept of aging where we keep on increasing the priority of low-priority process based on the its waiting time.

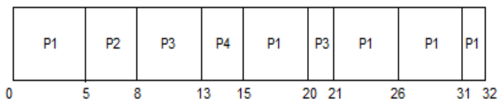
For example, if we decide the aging factor to be 0.5 for each day of waiting, then if a process with priority 20(which is comparatively low priority) comes in the ready queue. After one day of waiting, its priority is increased to 19.5 and so on. Doing so, we can ensure that no process will have to wait for indefinite time for getting CPU time for processing.

4. ROUND ROBIN SCHEDULING

- A fixed time is allotted to each process, called quantum, for execution.
- Once a process is executed for given time period that process is preempted and other process executes for given time period.
- Context switching is used to save states of preempted processes.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

The GANTT chart for round robin scheduling will be,



The average waiting time will be, 11 ms.

Characteristics of Round-Robin Scheduling

- Round robin is a pre-emptive algorithm
- The CPU is shifted to the next process after fixed interval time, which is called time quantum/time slice.
- The process that is preempted is added to the end of the queue.
- Round robin is a hybrid model which is clock-driven
- Time slice should be minimum, which is assigned for a specific task that needs to be processed. However, it may differ OS to OS.
- It is a real time algorithm which responds to the event within a specific time limit.
- Round robin is one of the oldest, fairest, and easiest algorithm.
- Widely used scheduling method in traditional OS.

Advantage of Round-robin Scheduling

- It doesn't face the issues of starvation or convoy effect.
- All the jobs get a fair allocation of CPU.
- It deals with all process without any priority
- If you know the total number of processes on the run queue, then you can also assume the worst-case response time for the same process.
- This scheduling method does not depend upon burst time. That's why it is easily implementable on the system.
- Once a process is executed for a specific set of the period, the process is preempted, and another process executes for that given time period.
- Allows OS to use the Context switching method to save states of preempted processes.
- It gives the best performance in terms of average response time.

Disadvantages of Round-robin Scheduling

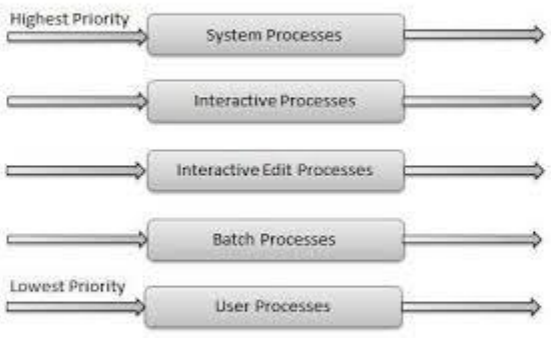
- If slicing time of OS is low, the processor output will be reduced.
- This method spends more time on context switching
- Its performance heavily depends on time quantum.
- Priorities cannot be set for the processes.
- Round-robin scheduling doesn't give special priority to more important tasks.
- Decreases comprehension
- Lower time quantum results in higher the context switching overhead in the system.
- Finding a correct time quantum is a quite difficult task in this system.

5. MULTILEVEL QUEUE SCHEDULING

Another class of scheduling algorithms has been created for situations in which processes are easily classified into different groups.
A common division is made between foreground (or interactive) processes and background (or batch) processes. These two types of processes have different response-time requirements, and so might have different scheduling needs. In addition, foreground processes may have priority over background processes.
A multi-level queue scheduling algorithm partitions the ready queue into several separate queues. The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type. Each queue has its own scheduling algorithm.
Separate queues might be used for foreground and background processes. The foreground queue might be scheduled by Round Robin algorithm, while the background queue is scheduled by an FCFS algorithm.
In addition, there must be scheduling among the queues, which is commonly implemented as fixed-priority preemptive scheduling. For example: The foreground queue may have absolute priority over the background queue.
Let us consider an example of a multilevel queue-scheduling algorithm with five queues:

1. System Processes
2. Interactive Processes
3. Interactive Editing Processes
4. Batch Processes
5. Student Processes

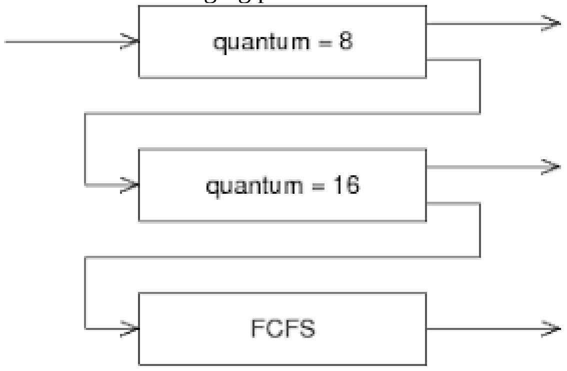
Each queue has absolute priority over lower-priority queues. No process in the batch queue, for example, could run unless the queues for system processes, interactive processes, and interactive editing processes were all empty. If an interactive editing process entered the ready queue while a batch process was running, the batch process will be preempted.



6. MULTILEVEL FEEDBACK QUEUE SCHEDULING

In a multilevel queue-scheduling algorithm, processes are permanently assigned to a queue on entry to the system. Processes do not move between queues. This setup has the advantage of low scheduling overhead, but the disadvantage of being inflexible.

Multilevel feedback queue scheduling, however, allows a process to move between queues. The idea is to separate processes with different CPU-burst characteristics. If a process uses too much CPU time, it will be moved to a lower-priority queue. Similarly, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue. This form of aging prevents starvation.



A multilevel feedback queue scheduler is defined by the following parameters:

- The number of queues.
- The scheduling algorithm for each queue.
- The method used to determine when to upgrade a process to a higher-priority queue.
- The method used to determine when to demote a process to a lower-priority queue.
- The method used to determine which queue a process will enter when that process needs service.

The definition of a multilevel feedback queue scheduler makes it the most general CPU-scheduling algorithm. It can be configured to match a specific system under design. Unfortunately, it also requires some means of selecting values for all the parameters to define the best scheduler. Although a multilevel feedback queue is the most general scheme, it is also the most complex.

IV. CONCLUSION

A lot of attempts were developed to find a solution for the high turnaround time, high waiting time and the overhead of extra context switches in all types of algorithms. It is recommended to use the shortest burst time concept because it will give the operating system the ability to adapt to the user behavior which may lead us to rethink building an intelligent, learnable and adaptable operating system. Scheduling algorithms should not affect the behavior of the system. The algorithms impact the system's efficiency and scheduling criteria.

V. REFERENCES

[1] Silberschatz,A. and P.B. Galvin,1997. Operating System concepts, Fifth Edition, John Wiley & Sons, Inc.,
[2] Stallings,W., 1996. Computer Organization and Architecture; Designing for Performance, Fourth Edition, Prentice Hall
[3] https://www.studytonight.com/operating-system/multilevel-queue-scheduling
[4] https://www.quora.com/What-are-the-goals-of-CPU-scheduling
[5] http://www.uniassignment.com/essay-
[6] https://study.com/academy/lesson/scheduling-policies-for-operating-systems-importance-criteria.html
[7] https://www.researchgate.net/publication/281615591
[8] https://www.researchgate.net/publication/317426546
[9] https://www.guru99.com/round-robin- scheduling-example.html