Contents

Scheduling Techniques

The process scheduler makes critical decisions many times every second. An efficient scheduling system will select a good **process mix** of **CPU-bound** processes and **I/O bound** processes. When more than one process is ready to run, the OS must decide which one first. The part of the operating system concerned with this decision is called the ***scheduler***, and algorithm it uses is called the ***scheduling algorithm***.

The Scheduling algorithms can be divided into two categories with respect to how they deal with clock interrupts and can be either:

1. preemptive or
2. non-preemptive.

1. **Preemptive scheduling** - enables the low level scheduler to remove a process from the RUNNING state in order to allow another process to run.
   The strategy of allowing processes that are logically runable to be temporarily suspended is called Preemptive Scheduling and it is contrast to the "run to completion" method.
2. **Nonpreemptive Scheduling** - A scheduling discipline is nonpreemptive if, once a process has been given the CPU, the CPU cannot be taken away from that process ("run to completion").

   In nonpreemptive, the CPU scheduling decisions may take place when a process:

   - Switches from running to waiting state
   - Switches from running to ready state
   - Switches from waiting to ready state
   - Terminates

   They are policies that allow the process to run until it terminates or incurs an I/O wait.

   - short jobs are made to wait by longer jobs but the overall treatment of all processes is fair.
   - response times are more predictable because incoming high priority jobs can not displace waiting jobs.

A scheduler executes jobs in the following two situations:

(1) When a process switches from running state to the waiting state.

(2) When a process terminates.

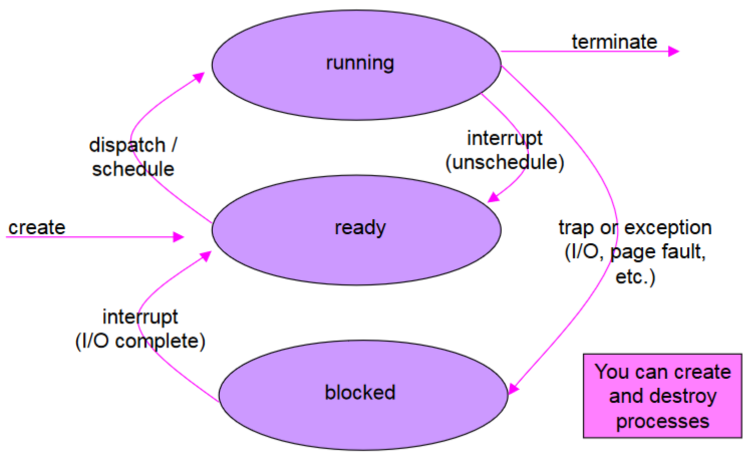


Figure 1 - Process States and State Transitions

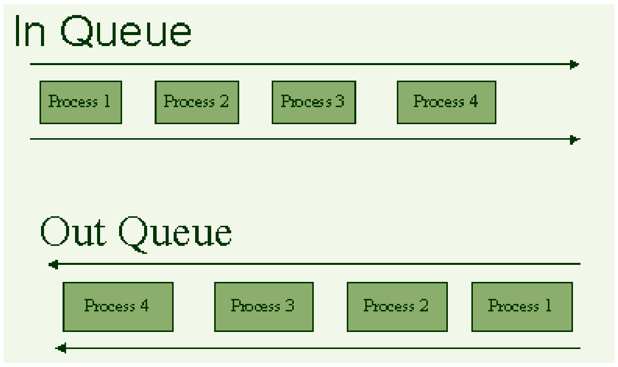
Single CPU Scheduling Algorithms

A selection of low level scheduling policies are as follows:

- FCFS Scheduling.
- SJN Scheduling.
- Round Robin Scheduling.
- Priority Scheduling.

1) First-Come-First-Served (FCFS)

Also known as First In First Out (FIFO), Run-to-Completion, or Run-Until-Done, this policy simply assigns the processor to the process that is first in the READY queue – i.e. the one that has been waiting for the longest time.



This is a *non-preemptive* scheme, since it occurs only when the current process relinquishes control, hence the name "Run-Until-Done".

Advantages:

The code for FCFS scheduling is simple to write and understand - FCFS algorithm doesn't include any complex logic, it just puts the process requests in a queue and executes it one by one.

FCFS is more predictable than most of other schemes since it offers time.

Every process will get a chance to run, so **process starvation** – where a process is "starved" of CPU access doesn't occur.

Disadvantages:

- This method can be unfair to short jobs (long jobs make short jobs wait and unimportant jobs make important jobs wait)
- If a CPU-bound process gets the processor, it could run for long periods of time while the

READY queue grows. There is no option for pre-emption of a process. If a process is started, then CPU executes the process until it ends

- FCFS scheme is not useful in scheduling interactive users because it cannot guarantee good response time

FCFS is rarely used as a master scheme in modern operating systems but it is often embedded within other schemes.

Please NOTE: go to lab section for FCFS examples

Variant of this is Last In First Out

The job that arrives last is serviced first but other characteristics remain as the FCFS algorithm.

2) Shortest Job Next (SJN)

Also called ***Shortest Job First*** (SJF) is *non-preemptive* policy selects from the READY queue the process that has the shortest estimated run time. When CPU is available, it is assigned to the process that has smallest next CPU burst. Short jobs will effectively jump past longer jobs in the queue.

SJN is especially appropriate for batch jobs for which the run times are known in advance - In the production environment where the same jobs run regularly, it may be possible to provide reasonable estimate of run time, based on the past performance of the process. But in the development environment users rarely know how their program will execute.

In SJN scheme, once a job begin executing, it run to completion.

In SJN scheme, a running process may be preempted by a new arrival process with shortest estimated run-time.

The throughput is increased because more processes can be executed in less amount of time.

SJN is non preemptive therefore, it is not useful in timesharing environment in which reasonable response time must be guaranteed.

Disadvantage:

A long job may be delayed indefinitely by a succession of short jobs overtaking it. Therefore longer processes will have more waiting time, eventually they'll suffer **process starvation**.

3) Shortest Remaining Time (SRT)

SRT is a *preemptive* counterpart of SJN and is useful in time-sharing environment

The process with the smallest estimated run-time to completion is run next, including new arrivals.

SRT has higher overhead than its counterpart SJN.

SRT must keep track of the elapsed time of the running process and must handle occasional preemptions.

Arrival of small processes will run almost immediately. However, one big ***disadvantage*** is that longer jobs remain for longer in the READY queue and have even longer mean waiting time.

Say process A is in its RUNNING state and meanwhile process B arrives in the READY queue. If process B’s running time is shorter than process A’s remaining run-time, then process A will be put back to the READY state and process B will be put into RUNNING state.

4) Priority Scheduling

The SJN algorithm is a special case of the general priority scheduling algorithm

A priority number (integer) is associated with each process and this Priority can be decided based on memory requirements, time requirements or any other resource requirement

The CPU is allocated to the process with the

- highest priority (smallest integer = highest priority) are executed first
- Same priority are executed in FCFS manner

Priority scheduling can be either preemptive or non-preemptive

- A preemptive approach will preempt the CPU if the priority of the newly-arrived process is higher than the priority of the currently running process
- A non-preemptive approach will simply put the new process (with the highest priority) at the head of the ready queue

SJN is a priority scheduling algorithm where priority is the predicted next CPU burst time

The main problem with priority scheduling is starvation, that is, low priority processes may never execute

A solution is ***aging***; as time progresses, the priority of a process in the ready queue is increased

5) Round Robin (RR)

One of the oldest, simplest, fairest and most widely used algorithm is round robin (RR) in which ***context switches*** are performed (to save states of preempted processes.)

In the round robin scheduling, processes are dispatched in a FCFS manner but are given a limited amount of CPU time called a time-slice or a quantum, therefore **process starvation** cannot occur.

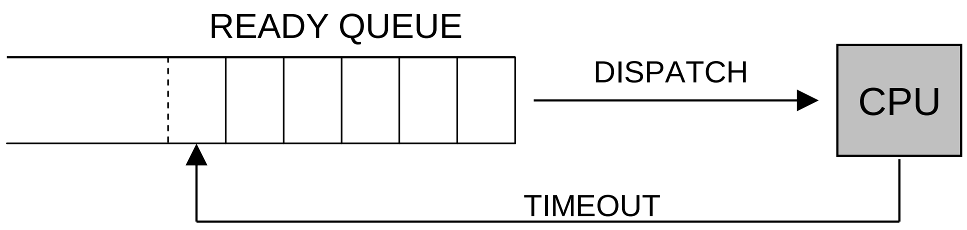
In the Round Robin scheme, a process is selected for RUNNING from the READY queue in FIFO (FCFS) sequence.

However, if the process runs beyond a certain fixed length of time, called a time quantum and its CPU time expires, the CPU is preempted and given to the next process waiting in a queue. The preempted process is then placed at the back of the ready list

In other words, the processor works on the process for a fixed amount of time (its *time slice*) before it terminates or returns back to the end of the READY queue.

The RR scheme is essentially the preemptive (at the end of the time slice) version of FCFS, but processes are only interrupted at the expiry of the time quantum, which is the key parameter of this scheduling algorithm

Therefore it is effective in time-sharing environments in which the system needs to guarantee reasonable response times for interactive users



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.

Variants of Round Robbin

➢ Cyclic RR

Just like normal round robin except that the quantum is:

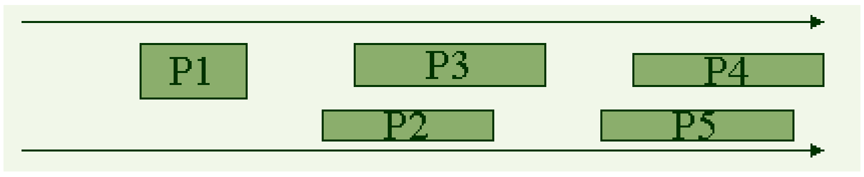
q=Q/n

where n is total number processes waiting and

Q is a time chunk

Basic Characteristics of cyclic RR are that there is graceful degradation, low-switching overhead.

It performs very similarly to Round Robin



For the above example, the number of processes in the Queue are 5, so each CPU burst will be Q/5.

➢ State dependent RR

Same as RR but q is varied dynamically depending on the state of the system.

It favours processes holding important resources, for example, non-swappable memory.

Perhaps this should be considered medium term scheduling since q is not recalculated each time

- Selfish-RR

With this there are two classes of jobs (NEW, ACCEPTED).

When priority of a processes changes from NEW to ACCEPTED, it is moved to ACCEPTED and these ACCEPTED processes are allocated CPU time.

Its basic characteristics and performance is similar to RR.

Multilevel CPU Scheduling Algorithms

- Multilevel Queue Scheduling.
- Multilevel Feedback Queue Scheduling.

6) Multilevel Queue Scheduling

This algorithm partitions the ready queue in several separate queues and processes are permanently assigned to one queues. It is a common practice to associate some priority depending upon where the process may have or originated. For instance, systems programs may have a higher priority over the user programs.

Within the users there may be level of importance:

- In an on-line system the priority may be determined by the criticality of the source or destination. In such a case, an OS may maintain many process queues, one for each level of priority.
- In a real-time system we may even follow an earliest deadline first schedule.

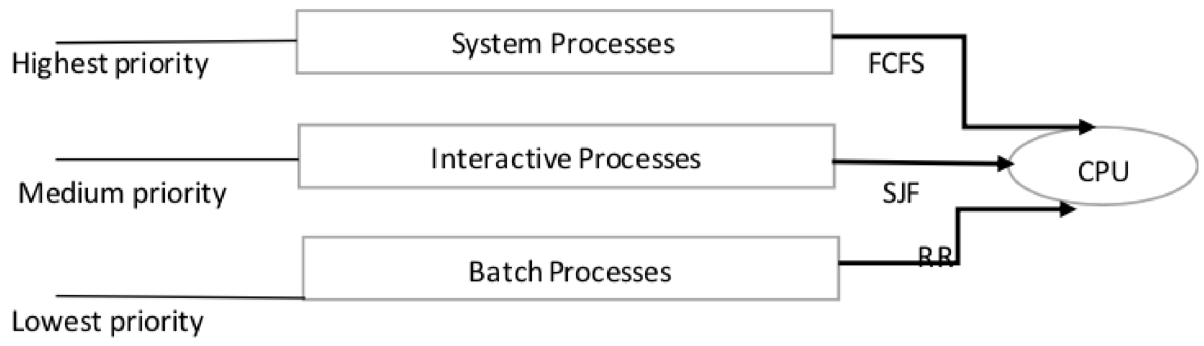
This policy introduces a notion ***priority*** on the basis of the deadline.

The processes are permanently assigned to one another, based on some property of the process, such as

- Memory size
- Process priority
- Process type

Algorithm choose the process from the occupied queue that has the highest priority, and run that process either ***preemptive*** or ***non-preemptively***

Each queue has its own scheduling algorithm or policy. For example:



- **System process** has the highest priority. If an interrupt is generated in the system, then the Operating system stops all the processes and handle the interrupt. According to different response time requirements, process needs different scheduling algorithm.
- **Interactive process** has medium priority. If we are using VLC player, we directly interact with the application. So all these processes will come in an interactive process. These queues will use scheduling algorithm according to requirement.
- **Batch processes** has the lowest priority. The processes which run automatically in the background comes under the batch processes. It is also known as background processes.

Therefore, according to process priority and type, the processes are scheduled with different scheduling algorithm.

<u>Problem in Multilevel Queue Scheduling</u>

These processes can't move across the queues.

- Running process can be preempted from a low priority queue when process arrives at high priority queue. Scheduling can lead to starvation.

- If dynamically process is entering in system queue, then other processes are lead to ***starvation***. Since we have scheduled the process queue as high priority and low priority, neither process can jump across the queue nor process can switch over.

These disadvantages are overcome by multilevel feedback queue scheduling

7) Multilevel Feedback Queue Scheduling

Process can move between the queues. It uses many READY queues and associates a different priority with each queue, ***ageing*** can be implemented this way.

The multilevel feedback algorithm selects always the first job of the lowest queue (i.e., the queue with the highest priority) that is not empty

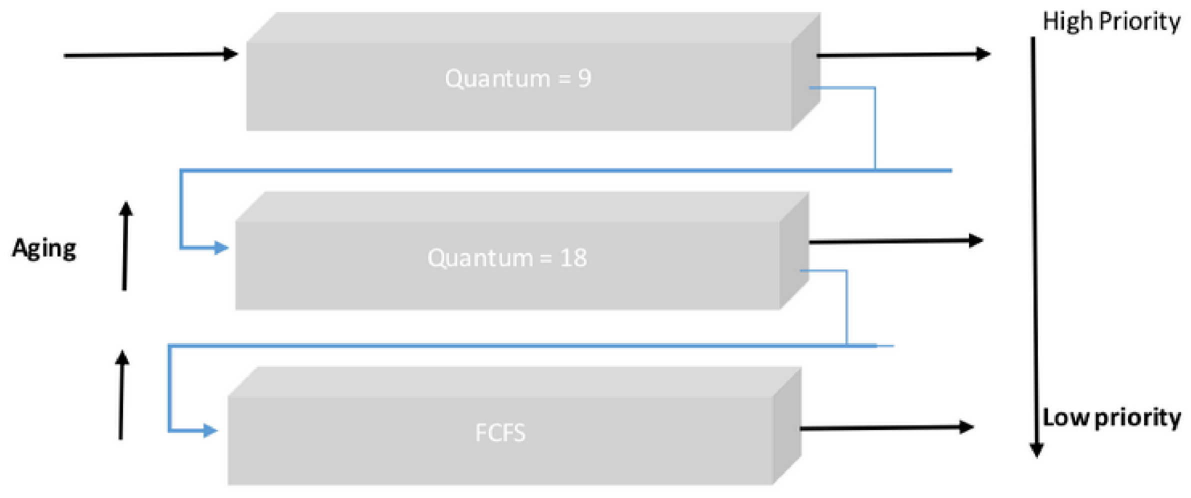


Figure 2 - Eg of Multilevel Feedback Queue

A technique called ***Aging*** promotes lower priority process to the next higher priority queue after a suitable interval of time.

Advantages

A process that waits too long in a lower priority queue may be moved to a higher priority queue.

Disadvantages

Moving the process around queue produces more CPU overhead.

It may happen that processes in the ready queue can be divided into different classes where each class has its own scheduling needs. For example, a common division is a foreground (interactive) process and background (batch) processes. These two classes have different scheduling needs. For this kind of situation Multilevel Queue Scheduling is used.

There is no universal “best” scheduling algorithm, and many operating systems use extended or combinations of the scheduling algorithms above. For example, Windows NT/XP/Vista uses a **multilevel** feedback queue, a combination of fixed-priority preemptive scheduling, round-robin, and first in, first out algorithms.

Windows Scheduling

Windows scheduled threads using a priority-based, preemptive scheduling algorithm.

The scheduler ensures that the highest priority thread will always run.

The portion of the Windows kernel that handles scheduling is called the dispatcher to switch the processor from one processor to another.

The dispatcher uses a 32-level priority scheme to determine the order of thread execution.

Priorities are divided into two classes.

1. The ***variable class*** contains threads having priorities 1 to 15.
2. The ***real-time class*** contains threads with priorities ranging from 16 to 31.

There is also a thread running at priority 0 that is used for memory management.

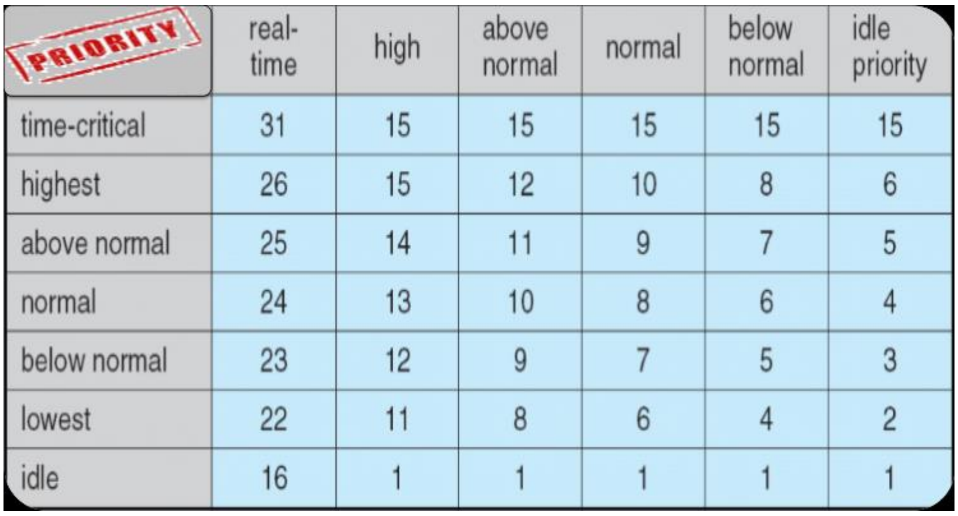
The dispatcher uses a queue for each scheduling priority and traverses the set of queues from highest to lowest until it finds a thread that is ready to run.

If no ready thread is found, the dispatcher will execute a special thread called the ***idle thread***

When a user is running an interactive program, the system needs to provide especially good performance for that process.

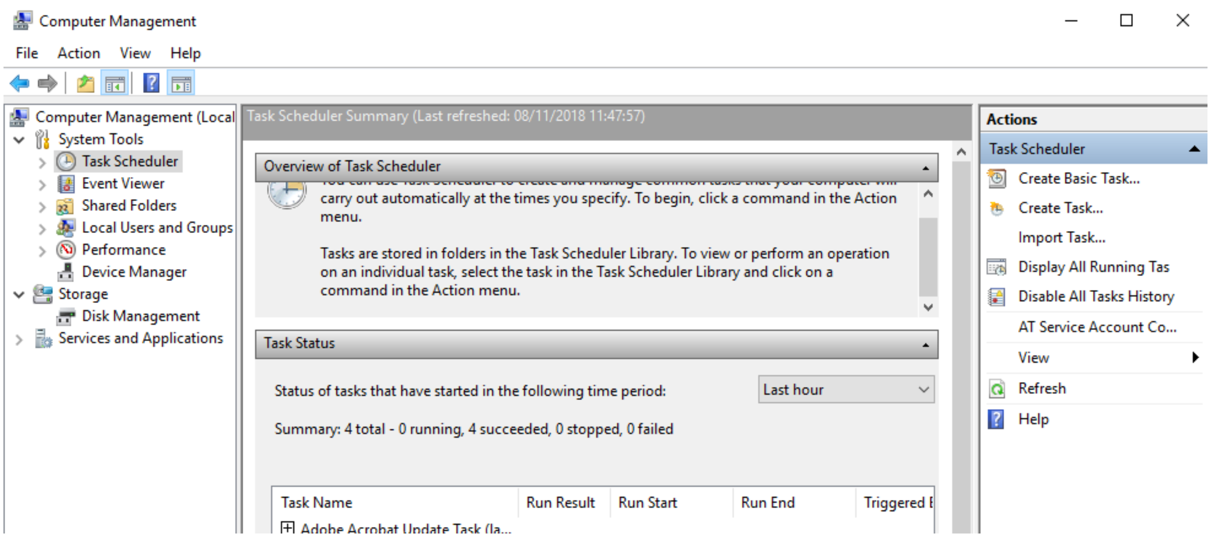
The priority of each thread is based on the priority class it belongs to and its relative priority within that class.

Within each of the priority classes is a relative priority:



PRIORITY	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

Depending on the usage of your Windows 10 computer, you can configure processor scheduling, so that it gives you the best performance while using Programs or for Background Processes You can make this adjustment easily via the Control Panel.



Task Scheduler Summary (Last refreshed: 08/11/2018 11:47:57)

Overview of Task Scheduler

carry out automatically at the times you specify. To begin, click a command in the Action menu.

Tasks are stored in folders in the Task Scheduler Library. To view or perform an operation on an individual task, select the task in the Task Scheduler Library and click on a command in the Action menu.

Task Status

Status of tasks that have started in the following time period: Last hour

Summary: 4 total - 0 running, 4 succeeded, 0 stopped, 0 failed

Task Name	Run Result	Run Start	Run End	Triggered
Adobe Acrobat Update Task (la...				

The Task Scheduler in Windows 10 executes scripts or programs at specific times or after certain events (we refer to these as triggers or conditions.)

It’s useful as a maintenance or automation tool – have a look now:

1. Navigate to the Task Scheduler and clicking on Task Scheduler Library.
2. Toward the center pane, a list of applications will appear. The majority of entries located here relate to updating software. However, a lot of **p**otentially **u**nwanted **p**rograms (PUP) store entries in this area. If you see a lot of programs that you don’t recognise, you might want to consider running a ***malware scan***.