Chapter 5/2
Processor Scheduling

Processes

A **process** is an executing program, including the current values of the program counter, registers, and variables.The subtle difference between a process and a program is that the program is a group of instructions whereas the process is the activity.

In multiprogramming systems, processes are performed in a pseudoparallelism as if each process has its own processor. In fact, there is only one processor but it switches back and forth from process to process.

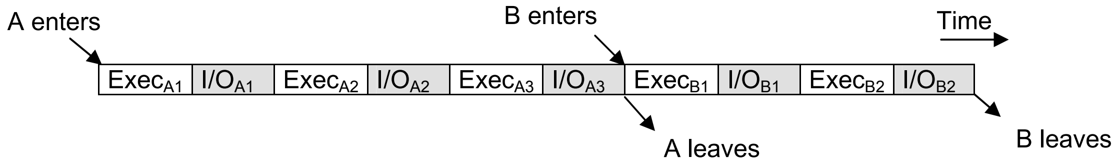
Henceforth, by saying *execution* of a process, we mean the processor's operations on the process like changing its variables, etc. and *I/O work* means the interaction of the process with the I/O operations like reading something or writing to somewhere. They may also be named as *"processor (CPU) burst"* and *"I/O burst"* respectively. According to these definitions, we classify programs as

- **Processor-bound program**: A program having long processor bursts (execution instants).
- **I/O-bound program**: A program having short processor bursts.

Assume we have two processes A and B. Both execute for 1 second and do some I/O work for 1 second. This pattern is repeated 3 times for process A and 2 times for process B.

.

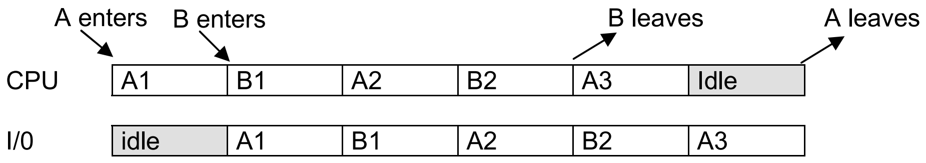
If we have **no multiprogramming**, the processes are executed sequentially as below.



So, the processor executes these two processes in a total time of 10 seconds. However, it is idle at I/O instants of processes. So, it is idle for 5 seconds and utilized for 5 seconds.

Then the processor utilization is $\frac{5}{10} \times 100 = 50\%$

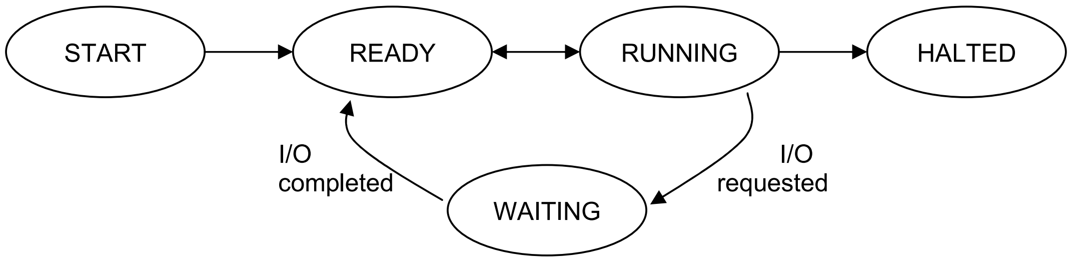
Now let's consider **multiprogramming** case:



In this case, when process A passes to some I/O work (i.e. does not use the processor), processor utilizes its time to execute process B instead of being idle.

Here the processor utilization is $\frac{5}{6} \times 100 \cong 83\%$

Process States



Start	: The process has just arrived.
Ready	: The process is waiting to grab the processor.
Running	: The process has been allocated by the processor.
Waiting	: The process is doing I/O work or blocked.
Halted	: The process has finished and is about to leave the system.

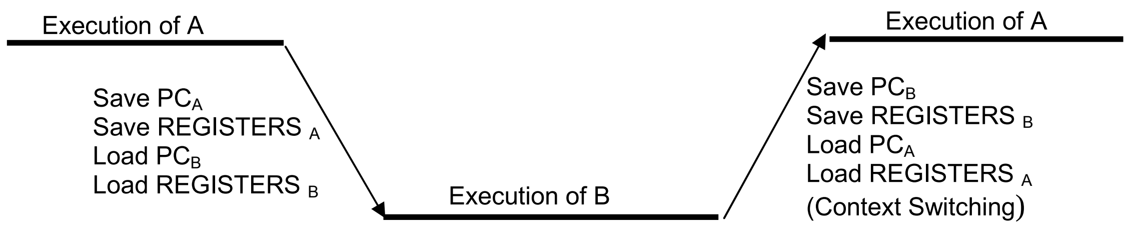
In the OS, each process is represented by its PCB (Process Control Block). The PCB, generally contains the following information:

- Process State,
- Process ID,
- Program Counter (PC) value,
- Register values
- Memory Management Information (page tables, base/bound registers etc.)
- Processor Scheduling Information (priority, last processor burst time etc.)
- I/O Status Info (outstanding I/O requests, I/O devices held, etc.)
- List of Open Files
- Accounting Info.

If we have a single processor in our system, there is only one running process at a time. Other ready processes wait for the processor. The system keeps these ready processes (their PCBs) on a list called *Ready Queue* which is generally implemented as a linked-list structure.

When a process is allocated by the processor, it executes and after some time it either finishes or passes to waiting state (for I/O). The list of processes waiting for a particular I/O device is called a *Device Queue*. Each device has its own device queue.

In multiprogramming systems, the processor can be switched from one process to another. Note that when an interrupt occurs, PC and register contents for the running process (which is being interrupted) must be saved so that the process can be continued correctly afterwards.Switching between processes occurs as depicted below.



Scheduler

If we consider batch systems, there will often be more processes submitted than the number of processes that can be executed immediately. So incoming processes are spooled (to a disk). The long-term scheduler selects processes from this process pool and loads selected processes into memory for execution.

The short-term scheduler selects the process to get the processor from among the processes which are already in memory.

The short-time scheduler will be executing frequently (mostly at least once every 10 milliseconds). So it has to be very fast in order to achieve a better processor utilization. The short-time scheduler, like all other OS programs, has to execute on the processor. If it takes 1 millisecond to choose a process that means (1 / (10 + 1)) = 9% of the processor time is being used for short time scheduling and only 91% may be used by processes for execution.

The long-term scheduler on the other hand executes much less frequently. It controls the degree of multiprogramming (no. of processes in memory at a time). If the degree of multiprogramming is to be kept stable (say 10 processes at a time), then the long-term scheduler may only need to be invoked when a process finishes execution.

The long-term scheduler must select a good process mix of I/O-bound and processor bound processes. If most of the processes selected are I/O-bound, then the ready queue will almost be empty while the device queue(s) will be very crowded. If most of the processes are processor-bound, then the device queue(s) will almost be empty while the ready queue is very crowded and that will cause the short-term scheduler to be invoked very frequently.

Time-sharing systems (mostly) have no long-term scheduler. The stability of these systems either depends upon a physical limitation (no. of available terminals) or the self-adjusting nature of users (if you can't get response, you quit).

It can sometimes be good to reduce the degree of multiprogramming by removing processes from memory and storing them on disk. These processes can then be reintroduced into memory by the medium-term scheduler. This operation is also known as swapping. Swapping may be necessary to improve the process mix or to free memory.

Performance Criteria

In order to achieve an efficient processor management, OS tries to select the most appropriate process from the ready queue. For selection, the relative importance of the followings may be considered as performance criteria.

Processor Utilization: The ratio of busy time of the processor to the total time passes for processes to finish. We would like to keep the processor as busy as possible.

Processor Utilization = (Processor buy time) / (Processor busy time + Processor idle time)

Throughput: The measure of work done in a unit time interval.

Throughput = (Number of processes completed) / (Time Unit)

Turnaround Time (tat): The sum of time spent waiting to get into the ready queue, execution time and I/O time.

tat = t(process completed) – t(process submitted)

Waiting Time (wt): Time spent in ready queue. Processor scheduling algorithms only affect the time spent waiting in the ready queue. So, considering only waiting time instead of turnaround time is generally sufficient.

Response Time (rt): The amount of time it takes to start responding to a request. This criterion is important for interactive systems.

rt = t(first response) – t(submission of request)

We, normally, want to maximize the processor utilization and throughput, and minimize tat, wt, and rt. However, sometimes other combinations may be required depending on to processes.

Processor Scheduling algorithms

Now, let's discuss some processor scheduling algorithms again stating that the goal is to select the most appropriate process in the ready queue. For the sake of simplicity, we will assume that we have a single I/O server and a single device queue, and we will assume our device queue always implemented with FIFO method. We will also neglect the switching time between processors (*context switching*).

First-Come-First-Served (FCFS)

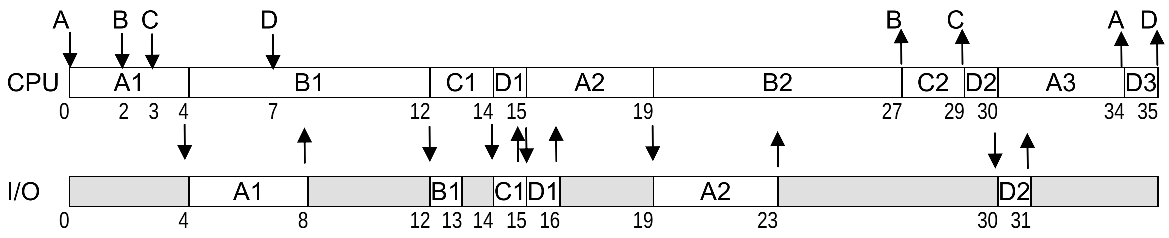
In this algorithm, the process to be selected is *the process which requests the processor first*. This is the process whose PCB is at the head of the ready queue. Contrary to its simplicity, its performance may often be poor compared to other algorithms.

FCFS may cause processes with short processor bursts to wait for a long time. If one process with a long processor burst gets the processor, all the others will wait for it to release it and the ready queue will be filled very much. This is called the *convoy effect*.

Example

Consider the following information and draw the timing (Gannt) chart for the processor and the I/O server using FCFS algorithm for processor scheduling.

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1



Processor utilization = (35 / 35) * 100 = 100 %

Throughput = 4 / 35=0.11

tat_A = 34 – 0 = 34
tat_B = 27 – 2 = 25
tat_C = 29 – 3 = 26
tat_D = 35 – 7 = 28

tat_{AVG} = (34 + 25 + 26 + 28) / 4 = 28.25

wt_A = (0 – 0) + (15 – 8) + (30 – 23) = 14
wt_B = (4 – 2) + (19 – 13) = 12
wt_C = (12 – 3) + (27 – 15) = 21
wt_D = (14 – 7) + (29 – 16) + (34 – 31) = 23

wt_{AVG} = (14 + 12 + 21 + 23) / 4 = 17.3

rt_A = 0 – 0 = 0
rt_B = 4 – 2 = 2
rt_C = 12 – 3 = 9
rt_D = 14 – 7 = 7

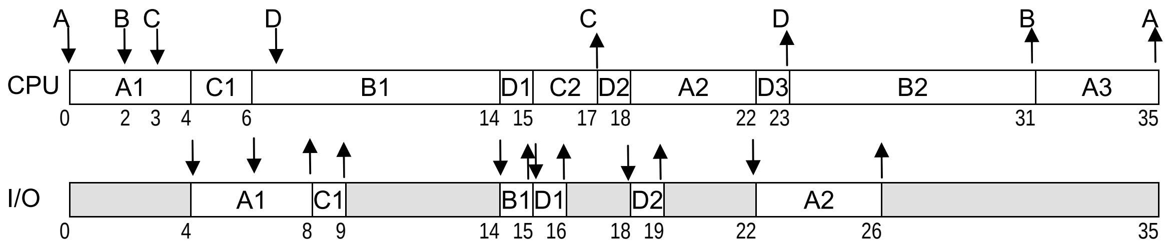
rt_{AVG} = (0 + 2 + 9 + 7) / 4 = 4.5

Shortest-Process-First (SPF)

In this method, the processor is assigned to *the process with the smallest execution (processor burst) time.* This requires the knowledge of execution time. In our examples, it is given as a table but actually these burst times are not known by the OS. So it makes prediction. One approach for this prediction is using the previous processor burst times for the processes in the ready queue and then the algorithm selects the shortest predicted next processor burst time.

Example

Consider the same process table in *Example 2.1* and draw the timing charts of the processor and I/O assuming SPF is used for processor scheduling. (Assume FCFS for I/0)



Processor utilization = (35 / 35) * 100 = 100 %

Throughput = 4 / 35 = 0.11

tat_A = 35 – 0 = 35
tat_B = 31 – 2 = 29
tat_C = 17 – 3 =14
tat_D = 23 – 7 = 16

tat_{AVG} = (35 + 29 + 15 + 16) / 4 = 23.5

wt_A = (0 – 0) + (18 – 8) + (31 – 26) = 15
wt_B = (6 – 2) + (23 – 15) = 12
wt_C = (4 – 3) + (15 – 9) = 7
wt_D = (14 – 7) + (17 – 16) + (22 – 19) = 11

wt_{AVG} = (15 + 12 + 7 + 11) / 4 = 11.25

rt_A = 0 – 0 = 0
rt_B = 6 – 2 = 4
rt_C = 4 – 3 = 1
rt_D = 14 – 7 = 7

rt_{AVG} = (0 + 4 + 1 + 7) / 4 = 3

Shortest-Remaining-Time-First (SRTF)

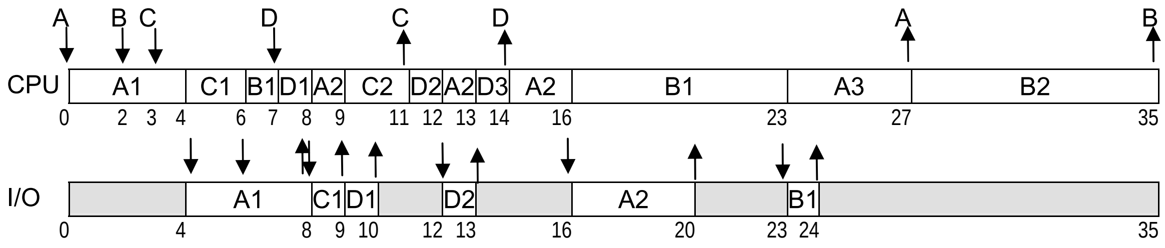
The scheduling algorithms we discussed so far are all non-preemptive algorithms. That is, once a process grabs the processor, it keeps the processor until it terminates or it requests I/O.

To deal with this problem (if so), preemptive algorithms are developed. In this type of algorithms, at some time instant, the process being executed may be preempted to execute a new selected process. The preemption conditions are up to the algorithm design.

SPF algorithm can be modified to be preemptive. Assume while one process is executing on the processor, another process arrives. The new process may have a predicted next processor burst time shorter than what is left of the currently executing process. If the SPF algorithm is preemptive, the currently executing process will be preempted from the processor and the new process will start executing. The modified SPF algorithm is named as Shortest-Remaining-Time-First (SRTF) algorithm.

Example

Consider the same process table in *Example 2.1* and draw the timing charts of the processor and I/O assuming SRTF is used for processor scheduling.



Processor utilization = (35 / 35) * 100 = 100 %

Throughput = 4 / 35 = 0.11

$tat_A = 27 – 0 = 27$
$tat_B = 35 – 2 = 33$
$tat_C = 11 – 3 = 8$
$tat_D = 14 – 7 = 7$

$tat_{AVG} = (27 + 33 + 8 + 7) / 4 = 18.75$

$wt_A = (0 – 0) + (8 – 8) + (12 - 9) + (14 – 13) + (23 - 20) = 7$
$wt_B = (6 – 2) + (16 – 7) + (27-24) = 16$
$wt_C = (4 – 3) + (9 – 9) = 1$
$wt_D = (7 – 7) + (11 – 10) + (13 – 13) = 1$

$wt_{AVG} = (7 + 16 + 1 + 1) / 4 = 6.25$

$rt_A = 0 – 0 = 0$
$rt_B = 6 – 2 = 4$
$rt_C = 4 – 3 = 1$
$rt_D = 7 – 7 = 0$

$rt_{AVG} = (0 + 4 + 1 + 0) / 4 = 1.25$

Round-Robin Scheduling (RRS)

In RRS algorithm the ready queue is treated as a FIFO circular queue. The RRS traces the ready queue allocating the processor to each process for a time interval which is smaller than or equal to a predefined time called *time quantum (slice)*.

The OS using RRS, takes the first process from the ready queue, sets a timer to interrupt after one time quantum and gives the processor to that process. If the process has a processor burst time smaller than the time quantum, then it releases the processor voluntarily, either by terminating or by issuing an I/O request. The OS then proceed with the next process in the ready queue.

On the other hand, if the process has a processor burst time greater than the time quantum, then the timer will go off after one time quantum expires, and it interrupts (preempts) the current process and puts its PCB to the end of the ready queue.

The performance of RRS depends heavily on the selected time quantum.

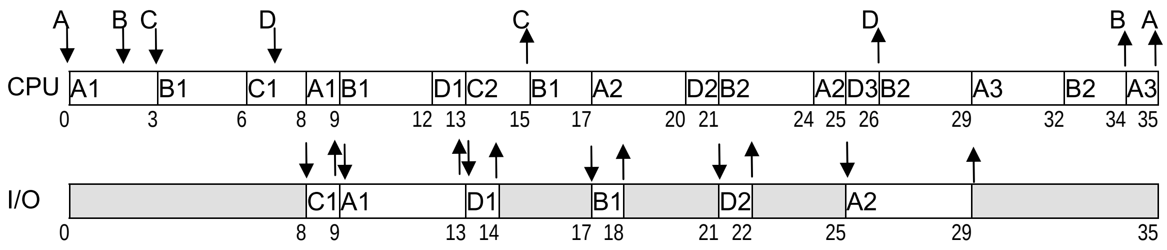
- Time quantum → ∞ $\Rightarrow$ RRS becomes FCFS

- Time quantum → 0 $\Rightarrow$ RRS becomes processor sharing (It acts as if each of the n processes has its own processor running at processor speed divided by n)

For an optimum time quantum, it can be selected to be greater than 80 % of processor bursts and to be greater than the context switching time.

Example

Consider the following information and draw the timing chart for the processor and the I/O server using RRS algorithm with time quantum of 3 for processor scheduling.



Processor utilization = (35 / 35) * 100 = 100 %

Throughput = 4 / 35

tat_A = 35 – 0 = 35
tat_B = 34 – 2 = 32
tat_C = 15 – 3 =12
tat_D = 26 – 7 = 19

tat_{AVG} = (35 + 32 + 12 + 19) / 4 = 24.5

wt_A = (0 – 0) + (8 – 3) + (17 - 13) + (24 – 20) + (29 - 29) + (34 – 32) = 15
wt_B = (3 – 2) + (9 – 6) + (15 -12) + (21 – 18) + (26 – 24) + (32 – 29) = 15
wt_C = (6 – 3) + (13 – 9) = 7
wt_D = (12 – 7) + (20 – 14) + (25 – 22) = 14

wt_{AVG} = (15 + 12 + 7 + 11) / 4 = 11.25

rt_A = 0 – 0 = 0
rt_B = 36 – 2 = 1
rt_C = 6 – 3 = 3
rt_D = 12 – 7 = 5

rt_{AVG} = (0 + 1 + 3 + 5) / 4 = 2.25

	FCFS	SPF	SRT	RR
tat_{avg}	28.25	23.5	18.75	24.5
wt_{avg}	16.5	10.5	6.25	12.25
rt_{avg}	4.5	3	1.25	2.25
	Easy to implement	Not possible to know next CPU burst exactly, it can only be guessed	Not possible to know next CPU burst exactly, it can only be guessed	Implementable, rt_{max} is important for interactive systems

Priority Scheduling

In this type of algorithms a priority is associated with each process and the processor is given to the process with the highest priority. Equal priority processes are scheduled with FCFS method.

To illustrate, SPF is a special case of priority scheduling algorithm where

Priority(i) = 1 / next processor burst time of process i

Priorities can be fixed externally or they may be calculated by the OS from time to time. Externally, if all users have to code time limits and maximum memory for their programs, priorities are known before execution. Internally, a next processor burst time prediction such as that of SPF can be used to determine priorities dynamically.

A priority scheduling algorithm can leave some low-priority processes in the ready queue indefinitely. If the system is heavily loaded, it is a great probability that there is a higher-priority process to grab the processor. This is called the *starvation problem.* One solution for the starvation problem might be to gradually increase the priority of processes that stay in the system for a long time.

Example

Following may be used as a priority defining function:
Priority (n) = 10 + t_{now} – ts(n) – tr(n) – cpu(n)
where

ts(n)	: the time process n is submitted to the system
tr(n)	: the time process n entered to the ready queue last time
cpu(n)	: next processor burst length of process n
t_{now}	: current time

exercise

1 Consider the following job arrival and CPU burst times given:

Job	Arrival time	CPU burst
A	0	7
B	2	4
C	3	1
D	6	6

a. Using the shortest job first scheduling method, draw the Gannt chart (showing the order of execution of these jobs), and calculate the average waiting time for this set of jobs.

b. Repeat **a.** using the shortest remaining time first scheduling method.

c. What is the main difference between these two methods ?

5. Explain the following briefly:

a. What is an I/O bound job?

b. What is CPU bound job?

c. Suppose there is one I/O bound job and one CPU bound job in a ready queue. Which one should get the processor in order to achieve a better CPU

d. Repeat **c**. for a better I/O device utilization.

6. A processor scheduling algorithm determines an order of execution of active processes in a computer system.

a. If there are n processes to be scheduled on one processor, how many possible different schedules are there? Give a formula in terms of n.

b. Repeat part **a.** for n processes to be scheduled on m processors.

7. Explain the following terms :

a. Long-term scheduler

b. Short-term scheduler

Which one controls the degree of multiprogramming?

8. a. Find and draw the Gannt Chart for the following processes assuming a preemptive shortest remaining time first processor scheduling algorithm.

process	arrival	next CPU
A	0	12
B	2	9
C	5	2
D	5	7
E	9	3
F	10	1

Clearly indicate every preemption on your Gannt Chart.

b. Calculate the turnaround times for each process, and find the average turnaround time for the above processes.

9. Consider a priority_based processor scheduling algorithm in which priorities are computed as the ratio of processor execution time to real time (total elapsed time).

a. Does this algorithm give higher priority to processor bound processes, or to I/O bound processes? Explain.

b. If priorities are recomputed every second, and if there is no I/O, to which algorithm does this processor scheduling algorithm degenerate to?

10. Show the execution of the following processes on Gannt Charts for the processor and the I/O device if Shortest Process First Algorithm for the Processor is used. Assume that the processes in the I/O queues are processes in First Come First Served manner . Find also the average waiting time in ready queue.

CPU and I/0 Bursts

Arrival	Process	1st CPU	1st I/O	2nd CPU	2nd I/O	3rd CPU
0	A	4	4	4	4	4
2	B	6	2	6	2	-
3	C	2	1	2	1	-
7	D	1	1	1	1	1

11. Show the execution of the following processes on Gannt Charts for the processor and the I/O device if **Shortest Remaining Time First** Algorithm for the Processor is used. Find also the average waiting time in ready queue. If the processess are to enter to the ready queue **at the same time** due to **a.** preemption of the processor, **b.** new submission, or **c.** completion of I/O operation, then the process of case **a.**, will be in front of the process of case **b.**, which will be in front of the process of case **c.**. Assume that I/O queue works in First Come First Served manner.

CPU and I/0 Bursts

Arrival	Process	1st CPU	1st I/O	2nd CPU	2nd I/O	3rd CPU
0	A	4	4	4	4	4
2	B	8	1	8	1	-
3	C	1	1	1	1	-

12. Consider a priority based processor scheduling algorithm such that whenever there is a process in the ready queue having priority higher priority than the one executing in CPU, it will force the executing process to prempt the CPU, and itself will grab the CPU. If there are more then one such processes having highests priority, then among these the one that entered into the ready queue first should be chosen. Assume that all the processes have a single CPU burst and have no I/O operation and consider the following table:

Process id	Submit (i) msec	Burst(i) msec
P1	0	8
P2	2	4
P3	3	2

where Submit(i) denotes when process Pi is submitted into the system (assume they are submitted just before the tic of the clock) and Burst(i) denotes the length of the CPU burst for process Pi . Let let Tnow to denote the present time and Execute(i) to denote the total execution of process Pi in cpu until Tnow. Assuming that the priorities are calculated every msec (just after the tic of the clock), draw the corresponding Gannt charts for the following priority formulas, which are valid when the process has been submitted but not terminated yet,

a. Priority(i)= Tnow-Submit(i)

b. Priority(i)= Burst(i)-Execute(i)

c. Priority(i)=$(\text{Burst(i)-Execute(i)})^{-1}$