Shortest Job First Scheduling Algorithm (SJF)

Introduction:

Modern operating systems switch from single task environment to multitask environment. The demand of today’s computing is to maximize the resource utilization of our CPU resources, and in order to maximize utilization of resources available, we use scheduling algorithms. It is important to choose the scheduling algorithm you will be using wisely as it is a major factor to increase the overall system running performance. It is vital to effectively and efficiently schedule operating system resources because the CPU is one of the most important resources of a computer system. A scheduling algorithm, in turns, decides which process executes first and how many resources are assigned to it depending on various factors that are chosen by the specific algorithm.

We will be talking about Shortest Job First (SJF) scheduling algorithm in this project. SJF improves the efficiency of the CPU for real time- and time-sharing environments. The algorithm starts by firstly selecting the process with the shortest burst time, and as the process finishes execution and the CPU is available, the process with the next shortest burst time is allowed to execute. In the case that the two or more processes have the same burst time, another algorithm is used to determine a tie. The First Come First Served (FCFS) algorithm is used to determine which process runs first in that case, based on the arrival time of each process.

Shortest Job First algorithm is optimal because it moves shorter processes to the front of the waiting queue, thus shorter processes execute first and longer processes wait until the shorter ones are finished and the CPU is available again. This is good because the waiting time of shorter processes will be decreased more than the increase of waiting time of longer processes. Consequently, the average waiting time is reduced to the maximum extent.

There are two modes of CPU scheduling that are preemptive and non-preemptive, the scheduling that takes place once a process switches from the running state to the ready state or from the waiting state to the ready state is named preemptive scheduling, On the other hand, the scheduling that takes place once a process terminates or switches from the running to the waiting state this mode of CPU scheduling is named Non-Preemptive scheduling.

In this project, the focus is on the Non-preemptive Shortest Job First Scheduling Algorithm.

Algorithm description:

This function enqueue its work is to enter new process in queue with its all information Then we come to parameters:

1. ID.
2. Arrival Time.
3. Burst Time.

And this function on entering the parameters, do some processes to calculate some info Depends on parameters:

1. Completion Time.
2. Turn Around Time.
3. Waiting Time.

By using equation to calculate every one of them:

1. Completion_T = Arrival_T + Burst_T **[1]**
2. TurnAround_T = Completion_T – Arrival_T
3. Waiting_T = TurnAround_T – Burst_T

- **Note: [1]** When we enter Number of nodes size is bigger than one so it will change to be:

  Completion_T = previous_ Completion_T + Burst_T

The first Node will be entered its order won't change whatever the number of nodes entered, The rest of nodes there order will be change from the smaller to the bigger according to Burst_T.

Now we will recognize about steps we followed to enter the nodes in queue and different cases:

1. We will create New Node.
2. We have some special cases to insert this node in queue.

- If front is equal to NULL, it means the queue is empty so this node will be the first one which have two cases too:

  1- If user enter the arrival Time to be zero then,
  Completion_T = Burst_T

  2- If user enter the arrival Time to be any number then,
  Completion_T = Arrival_T + Burst_T

- But, If front is not equal to NULL, it will be three cases.

  1- If size is equal to One That means that there is only one node in the queue. When entering the new node, this node does not go through the arrangement currently in this case, as we have previously explained in the arrangement.

1- Completion_T = previous_ Completion_T + Burst_T
2- TurnAround_T = Completion_T – Arrival_T
3- Waiting_T = TurnAround_T – Burst_T

Note: these Equations denoted by **[2]**

2- If size is less than one Consequently, we will arrange the node from the smallest to the largest in the queue except for the first node that entered, so we will create a new node, but its function is that it passes on each node separately, and therefore there are two cases:

1- If the node we inserted was less than the node on which we stand, we will place the inserted node before we stand on it and then complete the rest of the operations. **[2]**
2- If the node that we entered was greater than the nodes that we stood on, we will put the entered node the last node in the queue, and then we complete the rest of the operations. **[2]**

Formulas:

In this project we have to calculate three formulas which is also acting as parameters.

I. Completion Time (CT).

It has two conditions:

a- If it is the first process that have been enqueued then its formula is:

$$CT = ArrivalTime + Currentprocess\ BurstTime$$

b- If it is not the first one:

$$CT = previous\ CT + Currentprocess\ BurstTime$$

II. Turnaround Time (TAT).

$$TAT = CT - ArrivalTime.$$

III. Waiting Time (WT).

In this project we are implementing SJF algorithm in the Non-Preemptive - as it has been discussed earlier in the introduction section - so the Response Time (RT) is equal to Waiting Time.

$$WT = TAT - BT.$$

Any average could be calculated by following formal assuming that I have N_n numbers:

$$Average\ of\ the\ Numbers = (N_1 + N_2 + \ \dots + N_n)/n$$

Algorithm clarification and an example:

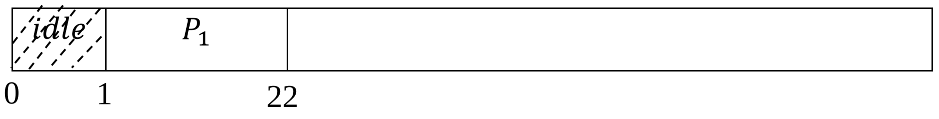
The following example have the Arrival Time and Burst Time as given below in "Table 1.0".

Process ID	*Arrival Time*	*Burst Time*
P_1	1	21
P_2	2	3
P_3	3	6
P_4	4	2

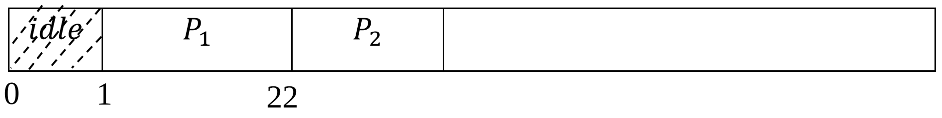
"$Table$ 1.0".

The following steps that the program run into to arrange the processes:

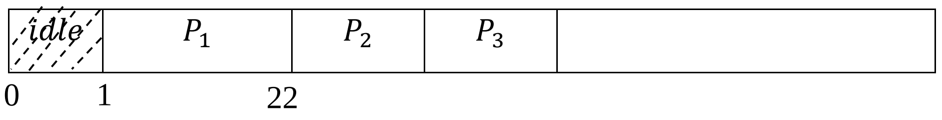
1) At arrival time=1, P_1arrives and starts execution, that is meaning that the CPU was idle at that time.



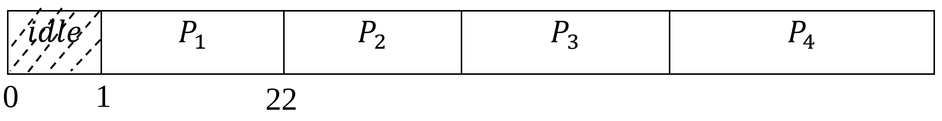
2) At arrival time= 2, Process P2 arrives But, P1 still needs more execution units to complete, so It will continue execution.



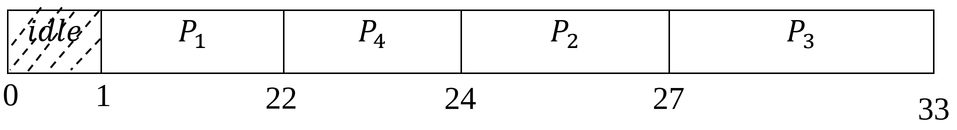
3) At arrival time= 3, Process P3 arrives But, P1 still needs more execution units to complete, so It will continue execution.



4) At arrival time= 4, Process P4 arrives But, P1 still needs more execution units to complete, so It will continue execution.



5) At arrival time= 22, process P_1 will finish its execution the burst time of P_2, P_3, and P_4 is compared, process P_4 , P_2, P_3 is executed because its burst time is the lowest, then at arrival time= 24, P_4 will finish its execution, P_2 is executed, then at arrival time= 27, P_2 will finish its execution,P_3 is executed.

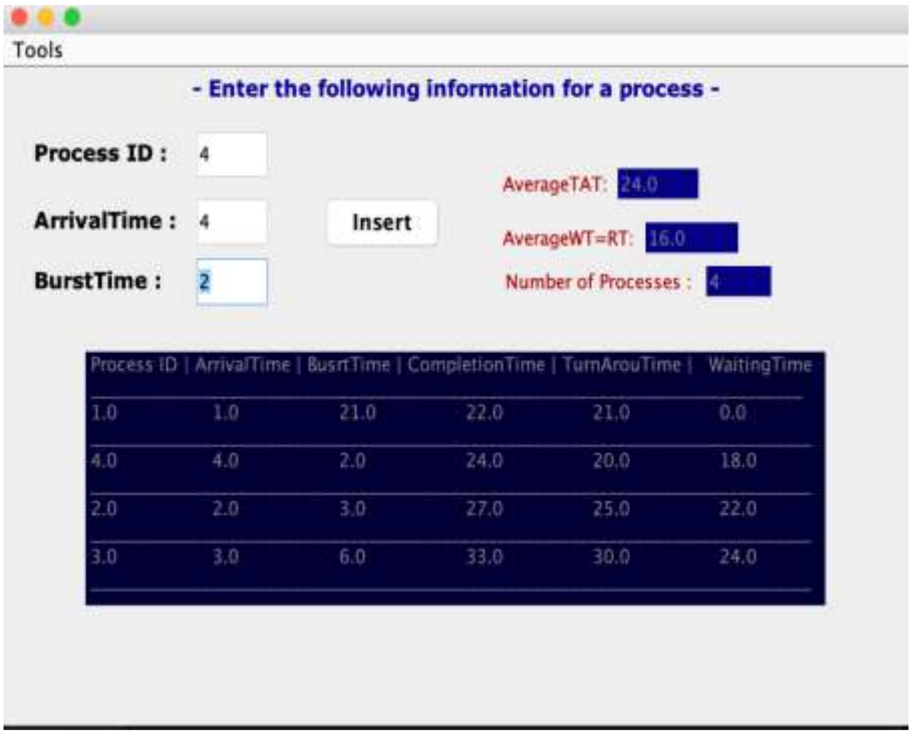


By applying the above equations for each process to calculate the Turnaround Time and Waiting Time shown in the formula section we get the results as shown in "Table 2.0".

Process ID	*ArrivalTime*	*BurstTime*	*CompletionTime*	*TurnAroundTime*	*WaitingTime*
P_1	1	21	22	21	0
P_2	2	3	27	25	22
P_3	3	6	33	30	24
P_4	4	2	24	20	18
AVG				24.0	16.0

"Table 2.0*"*

User interface of the discussed example:



- Enter the following information for a process -

Process ID : 4

ArrivalTime : 4

BurstTime : 2

AverageTAT: 24.0

AverageWT=RT: 16.0

Number of Processes : 4

Process ID	ArrivalTime	BusrtTime	CompletionTime	TurnArouTime	WaitingTime
1.0	1.0	21.0	22.0	21.0	0.0
4.0	4.0	2.0	24.0	20.0	18.0
2.0	2.0	3.0	27.0	25.0	22.0
3.0	3.0	6.0	33.0	30.0	24.0

References:

1- Silberschatz, A., Galvin, P., & Gagne, G. Operating system concepts (9th ed., pp. 267-270).

2- Mitrani, I. (1987). Modelling of computer and communication systems (24th ed., p. 108). Cambridge University Press.