**Lab no 4**

**Shortest Job First Scheduling**

***Objectives:***

* What is shortest job first scheduling?
* Implementation of Non-preemptive SJF
* Implementation of preemptive SJF

**Shortest job first:** is an algorithm in which the process having the smallest execution time is chosen for the next execution. This scheduling method can be preemptive or non-preemptive. It significantly reduces the average waiting time for other processes awaiting execution.

**Characteristics of SJF Scheduling**

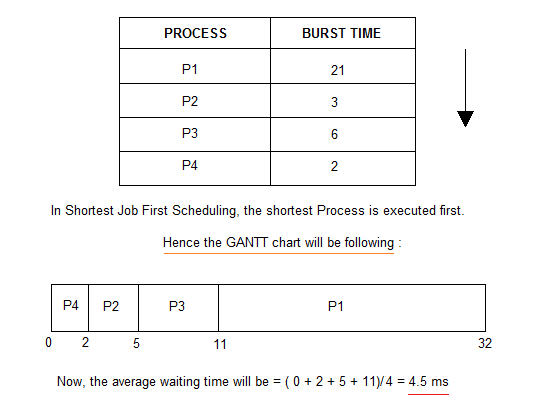
* It is associated with each job as a unit of time to complete.
* This algorithm method is helpful for batch-type processing, where waiting for jobs to complete is not critical.
* It can improve process throughput by making sure that shorter jobs are executed first, hence possibly have a short turnaround time.
* It improves job output by offering shorter jobs, which should be executed first, which mostly have a shorter turnaround time.

**Algorithm:**

* Sort all the process according to the arrival time.
* Then select that process which has minimum arrival time and minimum Burst time.
* After completion of process make a pool of process which after till the completion of previous process and select that process among the pool which is having minimum Burst time.

**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.



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.

print("Shortest Job SCHEDULLING")

n= int(input("Enter number of processes : "))

d = dict()

for i in range(n):

    key = "P"+str(i+1)

    a = float(input("Enter arrival time of process"+str(i+1)+": "))

    b = float(input("Enter burst/Executing time of process"+str(i+1)+": "))

    l = []

    l.append(a)

    l.append(b)

    d[key] = l

d = sorted(d.items(), key=lambda item: item[1][1])

ET = []

for i in range(len(d)):

    # first process

    if(i==0):

        ET.append(d[i][1][1])

    # get prevET + newBT

    else:

        ET.append(ET[i-1] + d[i][1][1])

TAT = []

for i in range(len(d)):

    TAT.append(ET[i] - d[i][1][0])

WT = []

for i in range(len(d)):

    WT.append(TAT[i] - d[i][1][1]/1000)

avg\_WT = 0

for i in WT:

    avg\_WT +=i

avg\_WT = (avg\_WT/n)

avg\_TAT = 0

for i in TAT:

    avg\_TAT +=i

avg\_TAT = (avg\_TAT/n)

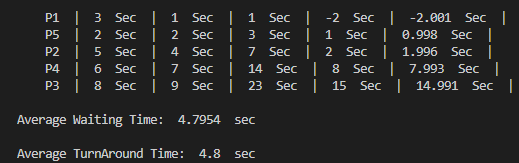
print("Process | Arrival | Execute time | Exit | Turn Around | Wait |")

for i in range(n):

      print("   ",d[i][0]," | ",d[i][1][0]," Sec"," | ",d[i][1][1]," Sec"," | ",ET[i]," Sec"," | ",TAT[i]," Sec"," | ",WT[i]," Sec"," | ")

print("\nAverage Waiting Time: ",avg\_WT," sec")

print("\nAverage TurnAround Time: ",avg\_TAT," sec")

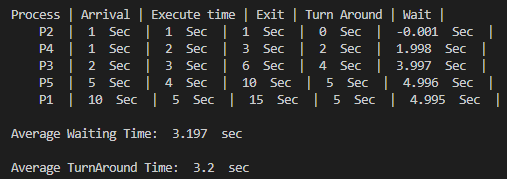
****

**Task:**

* **Consider the following set of processes, with the length of the CPU-burst time given in seconds:**

|  |  |  |
| --- | --- | --- |
| **Process** | **burst Time** | **Priority/who executes first** |
|  |  |  |
| P1 | 10 | 5 |
| P2 | 1 | 1 |
| P3 | 2 | 3 |
| P4 | 1 | 2 |
| P5 | 5 | 4 |

1. Give the turnaround time (total time from first arrival into ready state until CPU-burst is completed) of each process.
2. Give the waiting time (total time spent in the Ready state) of *each* process.
3. Give the average waiting time of all the processes.
4. Run given code for SJF above data.



* **Suppose the following three processes arrive for execution at the arrival times indicated.**

|  |  |  |
| --- | --- | --- |
| **Process** | **Arrival Time** | **Burst Time** |
|  |  |  |
| P1 | 0.0 | 8 |
| P2 | 0.4 | 4 |
| P3 | 1.0 | 1 |

1. Run Given code for SJF scheduling using above given data.

### *Text Description automatically generated with medium confidence*