**OPERATING SYSTEM ASSIGNMENT**

**NAME: PRAHLAD PRAJAPATI**

**ID: 11804542 ROLL NO.: 56**

**Email Address:prajapatipp1100@gmail.com**

**GitHub Link:** <https://github.com/Prahlad264/Os-assignment>

**CODE:-**

#include<stdio.h> int main() { int i, limit, total = 0, x, counter = 0, time\_quantum,j; int wait\_time = 0, turnaround\_time = 0,pos,z,p[10],prio[10], a\_time[10], b\_time[10], temp[10],b; float average\_wait\_time, average\_turnaround\_time; printf("\nEnter Total Number of Processes:"); scanf("%d", &limit); x = limit; for(i = 0; i < limit; i++) { p[i]=i+1; prio[i]=0; printf("\nEnter total Details of Process[%d]\n", i + 1); printf("Arrival Time:\t"); scanf("%d", &a\_time[i]); printf("Burst Time:\t"); scanf("%d", &b\_time[i])**;** temp[i] = b\_time[i];

} printf("\nEnter the Time Quantum:"); scanf("%d", &time\_quantum); printf("\nProcess ID\t\tBurst Time\t Turnaround Time\t Waiting Time\t Priority\n"); for(total = 0, i = 0; x != 0;)

{ for(z=0;z<limit;z++)

{

int temp1; pos=z;

for(j=z+1;j<limit;j++)

{

if(prio[j]<prio[pos]) pos=j;

} temp1=prio[z]; prio[z]=prio[pos]; prio[pos]=temp1; temp1=b\_time[z]; b\_time[z]=b\_time[pos]; b\_time[pos]=temp1; temp1=a\_time[z]; a\_time[z]=a\_time[pos]; a\_time[pos]=temp1; temp1=p[z]; p[z]=p[pos]; p[pos]=temp1;

temp1=temp[z];

temp[z]=temp[pos];

temp[pos]=temp1;

}

{

}

if(temp[i] <= time\_quantum && temp[i] > 0)

{

total = total + temp[i];

temp[i] = 0;

counter = 1;

}

else if(temp[i] > 0)

{

temp[i] = temp[i] - time\_quantum;

total = total + time\_quantum;

}

for(b=0;b<limit;b++)

{

if(b==i)

prio[b]+=1;

else

prio[b]+=2;

}

if(temp[i] == 0 && counter == 1)

{

x--;

printf("\nProcess[%d]\t\t%d\t\t %d\t\t %d\t\t%d", p[i], b\_time[i], total - a\_time[i], total - a\_time[i] - b\_time[i],prio[i]);

wait\_time = wait\_time + total - a\_time[i] - b\_time[i];

turnaround\_time = turnaround\_time + total - a\_time[i];

counter = 0;

}

if(i == limit - 1)

{

i = 0;

}

else if(a\_time[i + 1] <= total)

{

i++;

}

else

{

i = 0;

}

}

return 0;

}

**DESCRIPTON:-**

This question base on SRTF ALGORITHM.This Algorithm is the **preemptive version** of **SJF scheduling**. In SRTF, the execution of the process can be stopped after certain amount of time. At the arrival of every process, the short term scheduler schedules the process with the least remaining burst time among the list of available processes and the running process.

Once all the processes are available in the **ready queue**, No preemption will be done and the algorithm will work as **SJF scheduling**. The context of the process is saved in the **Process Control Block** when the process is removed from the execution and the next process is scheduled. This PCB is accessed on the **next execution** of this process.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Process id | Arrival time | Burst time | TAT | Waiting time |
| P1 | 0 | 5 | 12 | 7 |
| P2 | 1 | 3 | 9 | 6 |
| P3 | 2 | 3 | 9 | 6 |
| P4 | 4 | 1 | 3 | 2 |

Average waiting time=5.25 Average Turnaround time=8.25

**Algorithm**:

* Shortest remaining time (SRT) is the preemptive version of the SJN algorithm.
* The processor is allocated to the job closest to completion but it can be preempted by a newer ready job with shorter time to completion.
* Impossible to implement in interactive systems where required CPU time is not known.
* It is often used in batch environments where short jobs need to give preferenc

**Step:-**

1. Since, at time 0, the only available process is P1 with CPU burst time 8. This is the only available process in the list therefore it is scheduled
2. The next process arrives at time unit 1. Since the algorithm we are using is SRTF which is a preemptive one, the current execution is stopped and the scheduler checks for the process with the least burst time
3. If the burst time is same then we will check arrival time of every process which is in ready queue and select the process with minimum process.
4. And repeat all above step simultaneously.

**Complexity:**

**Loop/conditional statement complexity**

for(i = 0; i < limit; i++) n

for(total = 0, i = 0; x != 0;) n

{ for(z=0;z<limit;z++) n\*n { int temp1; pos=z; for(j=z+1;j<limit;j++) n\*n\*n { if(prio[j]<prio[pos]) 1 pos=j; }}}

for(b=0;b<limit;b++) n

complexity of rest of line of code 1

**Total complexity of program O(n^3)**

**Constraint:**

1. The context switch is done a lot more times in SRTF than in SJN and consumes cpu’s valuable time for processing . (ii) Practically it is not possible to predict the burst time.  
   (iii) Processes which have long burst time will have to wait for long time for execution.

**Additional algorithm:**We need to Round Robin algorithm because time quantum is given in the question

Round Robin is a [CPU scheduling algorithm](http://quiz.geeksforgeeks.org/gate-notes-operating-system-process-scheduling/) where each process is assigned a fixed time slot in a cyclic way.

* It is simple, easy to implement, and starvation-free as all processes get fair share of CPU.
* One of the most commonly used technique in CPU scheduling as a core.
* It is preemptive as processes are assigned CPU only for a fixed slice of time at most.
* The disadvantage of it is more overhead of context switching

**Test Cases: Test case1:** Enter the correct no of processes **Test case2:**Enter all time in positive intege**r** otherwise give the error **Test Case 3:**Time quantum should be small otherwise rise starvation condition **Test case4:**Enterthe no of process in limit

Github link: <https://github.com/Prahlad264/Os-assignment>