**LOVELY PROFESSIONAL UNIVERSITY**  
**Academic Task-3 (Operating System)**

**Student Name:** Sunil Kumar Reddy.P

**Student ID:**  11705278

**Email Address:** sunilkumarpaluru143@gmail.com

**GitHub Link:**

# QUESTION NO 10:

Design a scheduler with multilevel queue having two queues which will schedule the processes on the basis of pre-emptive shortest remaining processing time first algorithm (SROT) followed by a scheduling in which each process will get 2 units of time to execute. Also note that queue 1 has higher priority than queue 2. Consider the following set of processes (for reference) with their arrival times and the CPU burst times in milliseconds.

Process Arrival-Time Burst-Time

-----------------------------------------------

|  |  |  |
| --- | --- | --- |
| P1 | 0 | 5 |
| P2 | 1 | 3 |
| P3 | 2 | 3 |
| P4 | 4 | 1 |

Calculate the average turnaround time and average waiting time for each process. The input for number of processes and their arrival time, burst time should be given by the user.

**DESCRIPTION:**

In this, I have used Multilevel queue schedule having two queues where processes have scheduled on preemptive shortest remaining time first with 2 units (Uses Round Robin Algorithm) time taken by each process to execute and it will wait if the burst time of the process is more than 2 units then the process will execute again and again comparing the arrival and burst time till the total time finishes by the process and if the burst time less than 2 then the total time will be executed and terminates the process by the CPU.

Also, we take two queues where the queue1 is having high priority as compare to queue 2. Using this some processes will move on to queue1 and some will move to queue 2.

**ALGORITHM:**

1. Take two queue’s ready and ready 1.
2. In shortest job first scheduling algorithm the process with small Amount of time running until completion is selected to execute.
3. Enqueue processes p1 and p2 inside the ready queue according to the burst time of processes.
4. After completion of p1 and p2 processes we will enqueue our p3 and p4 processes in ready queue1.
5. In Round Robin Scheduling algorithm each process will assign a fixed time slot in CPU.
6. If the process completed its time quantum the then again sort the queue based on the priority of the processes allocated.
7. If the process completed execution before its time quantum then dequeue that process from the ready queue.
8. This process will continue until all the processes completed their execution.

**Entire Code:**

#include<stdio.h>

int n;

int process[10][7];

int readyQueue[20],readyQueue1[20],burst[10],arrivalTime[10];

int front=0,rear=-1,max=-1,cpuTime=0,flag=0,from;

/\*

1 arrival

2 burst

3 completion

\*/

void Enqueue(int a)

{

rear=rear+1;

readyQueue[rear]=a;

}

int Dequeue(){

int a=readyQueue[front];

process[a-1][3]=cpuTime+1;

front=front+1;

flag=1;

from=a;

return a;

}

int IsAllExecuted()

{

int i;

for(i=0;i<2;i++){

if(process[i][2]>0)

break;

}

if(i<2)

return 1;

return 0;

}

int sortReadyQueue()

{

if(readyQueue[front]<=0)

return -1;

else if(readyQueue[front+1]<=0)

return readyQueue[front]-1;

else

{

int a=readyQueue[front]-1;

int b=readyQueue[front+1]-1;

if(process[a][2]<process[b][2])

return a;

readyQueue[front]=b+1;

readyQueue[rear]=a+1;

return readyQueue[front]-1;

}

}

void roundRobin()

{

int count=0,check=0;

int cpuIdle=1;

int i,id;

int r=-1,f=0;

for(i=2;i<n;i++)

{

r=r+1;

readyQueue1[r]=i+1;

}

if(cpuIdle==1)

id=readyQueue1[f]-1;

while(check==0)

{

if(cpuTime<=4)

{

//

}

if(count==2)

{

int a=readyQueue1[f];

f=f+1;

r=r+1;

readyQueue1[r]=a;

id=readyQueue1[f]-1;

if(id!=-1)

{

cpuIdle=0;

}

else

check=1;

count=0;

}

cpuTime++;

process[id][2]=process[id][2]-1;

count++;

if(process[id][2]<=0)

{

f=f+1;

process[id][3]=cpuTime+1;

id=readyQueue1[f]-1;

count=0;

if(id==-1)

check=1;

}

}

}

void scheduler()

{

int id,i,count=0;

while(IsAllExecuted())

{

if(cpuTime<2)

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

if(cpuTime==arrivalTime[i])

Enqueue(i+1);

id=sortReadyQueue();

if(id!=-1)

{

process[id][2]=process[id][2]-1;

count++;

if(process[id][2]<=0)

{

Dequeue();

}

}

else

{

break;

}

cpuTime++;

}

roundRobin();

}

void calculateWaitingTime()

{

int i,wait\_avg=0;

printf("\n==================================================================\n\n");

printf("Process\t\tWaitingTime\n");

printf("------------------------------------\n");

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

{

process[i][5]=process[i][4]-burst[i];

wait\_avg+=process[i][5];

printf(" P[%d]\t\t%d\n",i+1,process[i][5]);

}

printf("\nThe Average Waiting Time is %d\n",wait\_avg/n);

}

void calculateTurnAroundTime()

{

int i,turn\_avg=0;

printf("\n==================================================================\n\n");

printf("Process\t\tTurnAroundTime\n");

printf("------------------------------------\n");

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

{

process[i][4]=process[i][3]-process[i][1];

turn\_avg+=process[i][4];

printf(" P[%d]\t\t%d\n",i+1,process[i][4]);

}

printf("\nThe Average TurnAround Time is %d\n",turn\_avg/n);

}

int main()

{

int i;

printf("\n===============Welcome To The Scheduler Design Software===============\n");

printf("\n Please Read the following Details and Enter The Details Carefully \n");

printf("\n Enter no of Processes n : ");

scanf("%d",&n);

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

{

printf("Enter ArrivalTime of P[%d] :: ",i+1);

scanf("%d",&arrivalTime[i]);

printf("Enter BurstTime of P[%d] :: ",i+1);

scanf("%d",&burst[i]);

process[i][1]=arrivalTime[i];

process[i][2]=burst[i];

}

printf("\n\n");

scheduler();

printf("\n");

calculateTurnAroundTime();

calculateWaitingTime();

return 0;

}

**Complete Solution: -**

1. **Completion Time (C.T):** Time at which process completes its execution.

2. **Turn Around Time (T.A.T):** Time Difference between completion time and arrival time.

* Turn Around Time = Completion Time – Arrival Time

3. **Waiting Time (W.T):** Time Difference between turn around time and burst time.

* Waiting Time = Turn Around Time – Burst Time

**ANSWER:**

**Process** **Arrival-Time** **Burst-Time**

-----------------------------------------------

|  |  |  |
| --- | --- | --- |
| P1 | 0 | 5 |
| P2 | 1 | 3 |
| P3 | 2 | 3 |
| P4 | 4 | 1 |

----------------------------------------------

Enter no of Processes n : 4

Arrival Time of P[1] :: 0

Burst Time of P[1] :: 5

Arrival Time of P[2] :: 1

Burst Time of P[2] :: 3

Arrival Time of P[3] :: 2

Burst Time of P[3] :: 3

Arrival Time of P[4] :: 4

Burst Time of P[4] :: 1

**Process** **Turn Around Time**

------------------------------------

|  |  |
| --- | --- |
| P[1] | 8 |
| P[2] | 3 |
| P[3] | 11 |
| P[4] | 8 |

The Average Turn Around Time is 7.

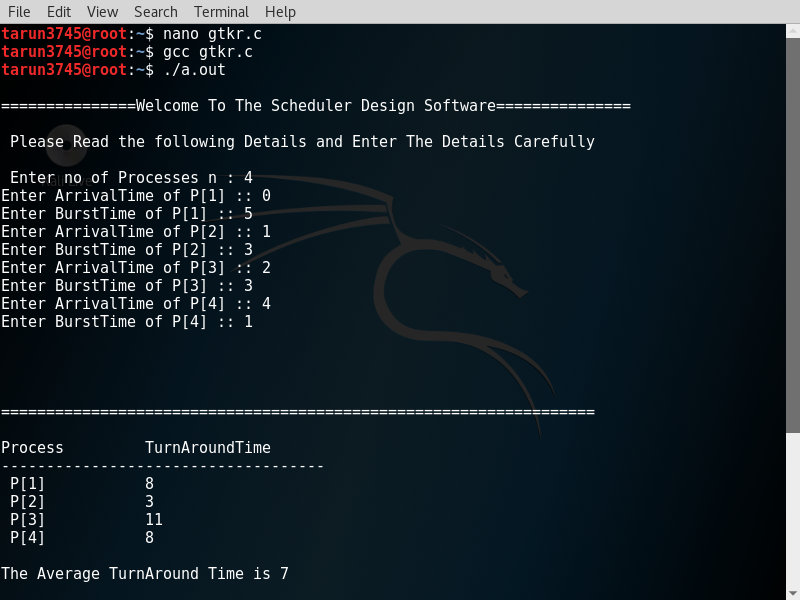
**Process** **Waiting Time**

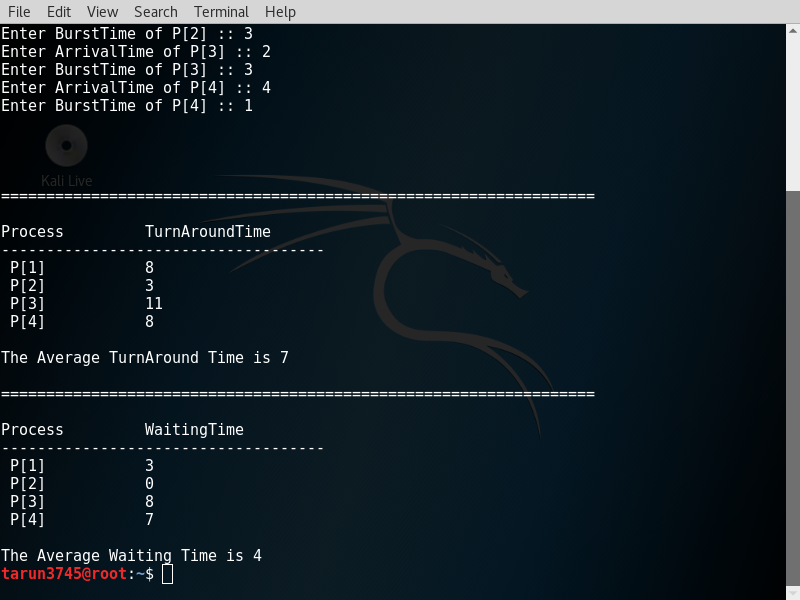
--------------------------------

|  |  |
| --- | --- |
| P[1] | 3 |
| P[2] | 0 |
| P[3] | 8 |
| P[4] | 7 |

The Average Waiting Time is 4.

**Output for Given Program: -**





**Test Cases: -**

Test Case1: -

**Process ArrivalTime BurstTime Turn. A. T Waiting. T**

P1 0 66 0

P2 1 8 13 5

P3 9 5 11 6

Average Turn Around Time :- 10

Average Waiting Time :- 3

Reference: -

A screenshot of a cell phone

Description automatically generated

Test Case2: -

**Process ArrivalTime BurstTime Turn. A. T Waiting. T**

P1 0 44 0

P2 2 5 -2 -7

P3 3 9 22 13

P4 5 2 11 9

P5 4 1 13 12

P6 1 7 17 10

Average Turn Around Time :- 10

Average Waiting Time :- 6

A screenshot of a computer

Description automatically generatedReference: -

A screenshot of a cell phone

Description automatically generated

* In test case 2, I have got the values of both Turn Around Time and Waiting Time for Process P[2] are in Negative as -2 and -7.

Test Case3: -

**Process ArrivalTime BurstTime Turn. A. T Waiting. T**

P1 0 58 3

P2 1 3 3 0

P3 2 2 9 7

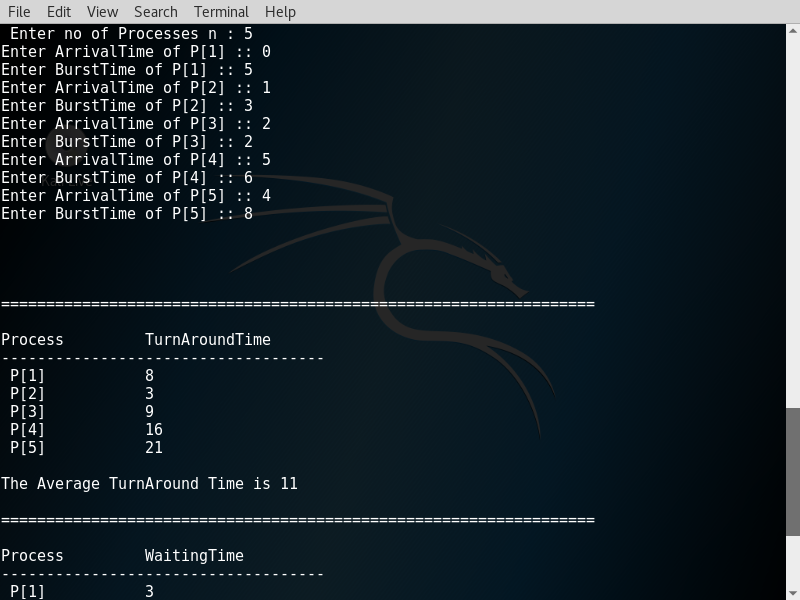
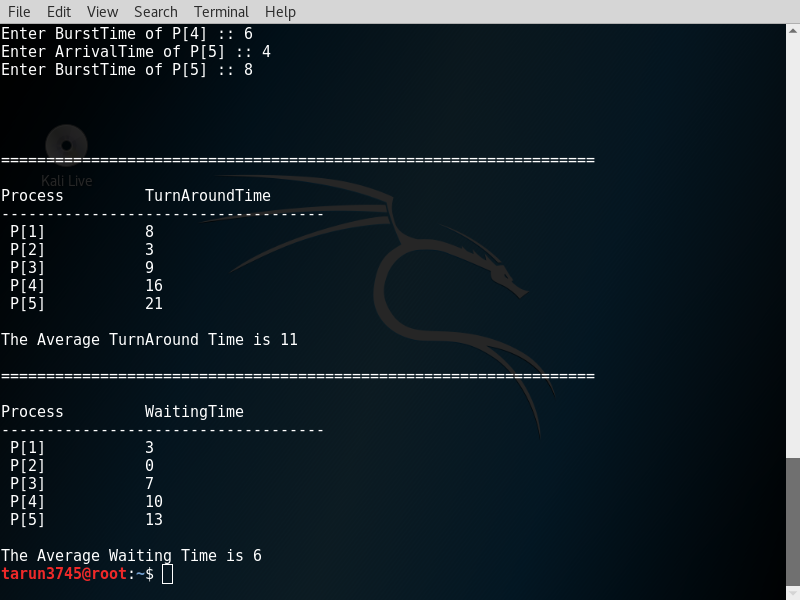
P4 5 6 16 10

P5 4 8 21 13

Average Turn Around Time :- 11

Average Waiting Time :- 6

Reference: -

**Constraints: -**

Some Constraints used in my scheduling program are :

* Condition used for 2 units time to take by the CPU in the process

for(i=2;i<n;i++)

{

r=r+1;

readyQueue1[r]=i+1;

}

* To check whether all the processes are completed their execution or not

int IsAllExecuted()

{

int i;

for(i=0;i<2;i++)

{

if(process[i][2]>0)

break;

}

if(i<2)

return 1;

return 0;

}

* Here is the constraint of the given question to take the value of the no. of processes to execute and details are given by the user.

printf("\n===============Welcome To The Scheduler Design Software===============\n");

printf("\n Please Read the following Details and Enter The Details Carefully \n");

printf("\n Enter no of Processes n : ");

scanf("%d",&n);

* Also the compiler will take the values of arrival and burst time by the user as mentioned in question

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

{

printf("Enter ArrivalTime of P[%d] :: ",i+1);

scanf("%d",&arrivalTime[i]);

printf("Enter BurstTime of P[%d] :: ",i+1);

scanf("%d",&burst[i]);

process[i][1]=arrivalTime[i];

process[i][2]=burst[i];

}

**Boundary Condition: -**

Here the main boundary condition is to execute the process maximum of 2 units time and to then it holds the process after the 2 units. In this period of time the other process will execute which has less arrival time as compare to previous holding process and it will go on with this boundary condition. If the time limit exceeds for a single process among all processes(i.e. 2 units will be as 3 or 4 units ) then the program or the output will become error or incorrect.