**SIMULATION BASED ASSIGNMENT ASSESSMENT**

**BACHELOR OF TECHNOLOGY**

**in**

**COMPUTER SCIENCE AND ENGINEERING**

**NAME OF THE STUDENT: SAURABH TRIPATHI**

**Email ID: saurabhtripathi1102000@gmail.com**

**Registration number: 11807532**

**Roll No: 02**

**Section: K18AP**



**School of Computer Science and Engineering**

Lovely Professional University

Phagwara, Punjab (India)

**SUBMITTED TO:**

**Dr. BALJIT SINGH SAINI**

**1.) DESCRIPTION:**

Q-Consider a scheduling approach which is non-pre-emptive similar to shortest job next in nature. The priority of each job is dependent on its estimated run time, and also the amount of time it has spent waiting. Jobs gain higher priority the longer they wait, which prevents indefinite postponement. The jobs that have spent a long time waiting compete against those estimated to have short run times. The priority can be computed as:

Priority = 1+ Waiting time / Estimated run time

---As this approach is based on non-pre-emptive similar to shortest job next we will first prioritise all the processes and then arrange them in the order of priority. We will use C programming language to implement the above challenge.

**2.) ALGORITHM:**

The scheduling approach is non – pre-emptive which suggests that once a process is started i.e. assigned CPU for execution, so it cannot be overtaken by any other process at that point of time.

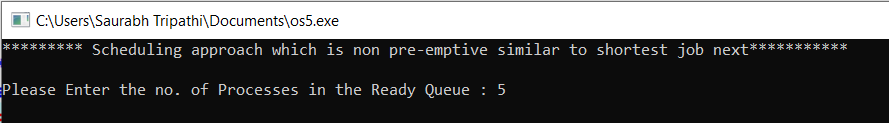
Shortest Job First means CPU is assigned to the process that have the shortest burst time, but the above scheduling approach is slightly different because it considers the factors like estimated run time, and also the amount of time it has spent waiting. All these factors can be accommodated

by using the formula of priority. The priority can be computed as:

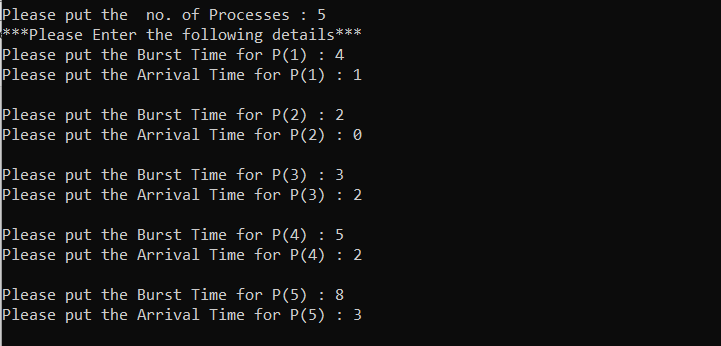
Priority = 1+ Waiting time / Estimated run time

STEPS:

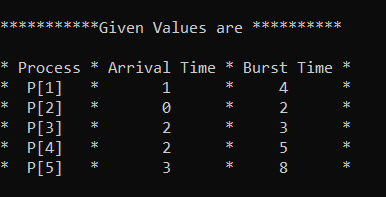
1. First, take the value of number of processes in the ready queue.



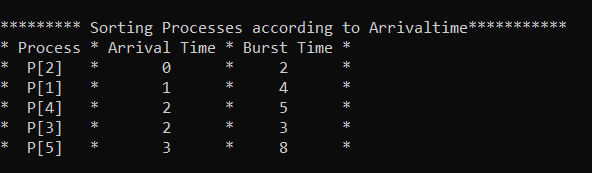
1. Now, for each process take the value of Burst time(required for using CPU cycles) and Arrival time(the time process enters the ready queue).



1. Arrange for processes in table.

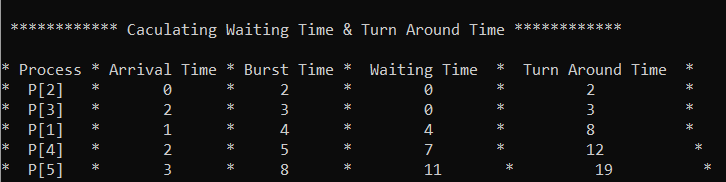


1. Reorder the list according the increasing order of Arrival time i.e. process with less arrival time comes first followed by the process whose arrival time is more.



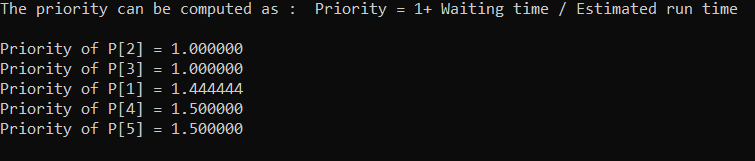
1. Calculate the Waiting time and Turnaround time of each process.

Turnaround time =Waiting time + Arrival Time

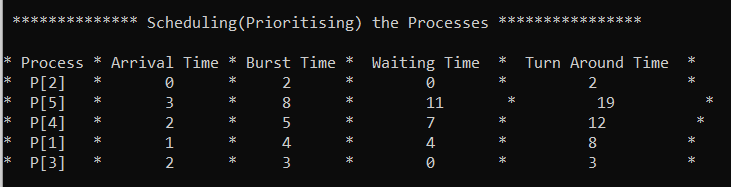


1. Then , calculate the priority of all the processes by using the formula

Priority = 1+ Waiting time / Estimated run time



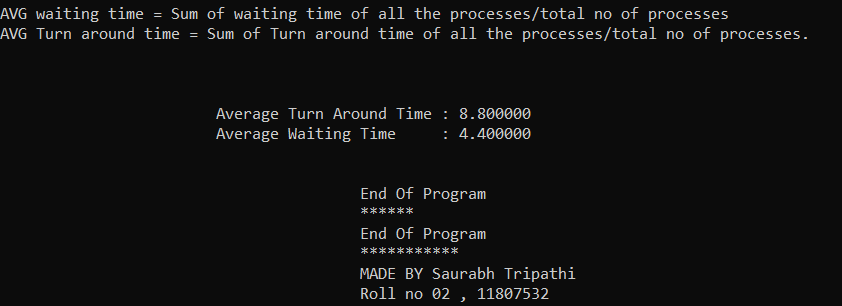
1. Arrange the processes according to the calculated priorities.



1. Calculate Average waiting and Turn around time (Optional)

AVG waiting time = Sum of waiting time of all the processes/total no of processes

AVG Turn around time = Sum of Turn around time of all the processes/total no of processes.



1. Exit

**3.) PURPOSE OF USE:**

The purpose of using this algorithm is to schedule the processes according to the priority. The factors that are considered while scheduling or while calculating the priority of processes are

estimated run time, and also the amount of time it has spent waiting. Jobs gain higher priority the longer they wait, which prevents indefinite postponement.

**4.) CODE SNIPPET:**

**#include<stdio.h>**

**#include<conio.h>**

**void proc\_inp() //function**

**{**

**prog\_main: //main program block**

**{**

**printf("\*\*\*\*\*\*\*\*\* Scheduling the processes\*\*\*\*\*\*\*\*\*\*\*\n\n");**

**printf("Please put the no. of Processes : ");**

**int n,y=0,z=0;**

**double avg\_waiting,avg\_turnaround,burstTime[n],arrivalTime[n];**

**scanf("%d",&n );**

**double min,total=0,total2=0,wait\_final, turnaround\_final;**

**if (n<=0)**

**{**

**printf("\nPlease enter a postive value in no of processes.\n");**

**goto prog\_main;**

**}**

**double wait\_avg, turnaround\_avg,process[n], var3,completionTime[n];**

**printf("\*\*\*Please Enter the following details\*\*\*");**

**double waitingTime[n],turnaroundTime[n],priority[n];**

**for(y=0;y<n;y++)**

**{**

**printf("\nPlease put the Burst Time for P(%d) : ", y+1 );**

**scanf("%lf", &burstTime[y]);**

**printf("Please put the Arrival Time for P(%d) : ", y+1 );**

**scanf("%lf", &arrivalTime[y] );**

**process[y]=y+1;**

**}**

**printf("\n\n\*\*\*\*\*\*\*\*\*\*\*Given Values are \*\*\*\*\*\*\*\*\*\*\n\n");**

**printf("\* Process \* Arrival Time \* Burst Time \*\n");**

**for(y=0;y<n;y++)**

**{**

**printf("\* P[%0.0lf] \* %0.0lf \* %0.0lf \*\n",process[y],arrivalTime[y],burstTime[y]);**

**}**

**printf("\n\n\*\*\*\*\*\*\*\*\* Sorting Processes according to ArrivalTime\*\*\*\*\*\*\*\*\*\*\*\n");**

**for(y=0;y<n;y++)**

**{**

**for(z=0;z<n;z++)**

**{**

**if(arrivalTime[y]<arrivalTime[z])**

**{**

**var3 = burstTime[z];**

**burstTime[z] = burstTime[y];**

**burstTime [y] = var3;**

**var3 = process[z];**

**process[z] = process[y];**

**process[y] = var3;**

**var3 = arrivalTime[z];**

**arrivalTime[z] = arrivalTime[y];**

**arrivalTime[y] = var3;**

**}**

**}**

**}**

**printf("\* Process \* Arrival Time \* Burst Time \*\n");**

**for(y=0;y<n;y++)**

**{**

**printf("\* P[%0.0lf] \* %0.0lf \* %0.0lf \*\n",process[y],arrivalTime[y],burstTime[y]);**

**}**

**int k = 1;**

**double b\_time = 0;**

**for(z=0;z<n;z++)**

**{**

**b\_time = b\_time + burstTime[z];**

**min = burstTime[k];**

**for(y=k;y<n;y++)**

**{**

**if((b\_time >= arrivalTime[y])&&(burstTime[y]<min))**

**{**

**var3 = burstTime[k];**

**burstTime[k] = burstTime[y];**

**burstTime[y] = var3;**

**var3 = arrivalTime[k];**

**arrivalTime[k] = arrivalTime[y];**

**arrivalTime[y] = var3;**

**var3 = process[k];**

**process[k] = process[y];**

**process[y] = var3;**

**}**

**}**

**k++;**

**}**

**waitingTime[0] = 0;**

**for(y=1;y<n;y++)**

**{**

**total += burstTime[y-1];**

**waitingTime[y] = total - arrivalTime[y];**

**wait\_final += waitingTime[y];**

**}**

**wait\_avg = wait\_final/n;**

**for(y=0;y<n;y++)**

**{**

**total2 += burstTime[y];**

**turnaroundTime[y] = total2 - arrivalTime[y];**

**turnaround\_final += turnaroundTime[y];**

**}**

**turnaround\_avg=turnaround\_final/n;**

**printf("\n\n \*\*\*\*\*\*\*\*\*\*\*\* Caculating Waiting Time & Turn Around Time \*\*\*\*\*\*\*\*\*\*\*\*\n\n");**

**printf("\* Process \* Arrival Time \* Burst Time \* Waiting Time \* Turn Around Time \*\n");**

**for(y=0;y<n;y++)**

**{**

**printf("\* P[%0.0lf] \* %0.0lf \* %0.0lf \* %0.0lf \* %0.0lf \*\n",**

**process[y],arrivalTime[y],burstTime[y],waitingTime[y],turnaroundTime[y]);**

**}**

**completionTime[0] = burstTime[0];**

**for(y=1;y<n;y++)**

**{**

**completionTime[y] = completionTime[y-1] + burstTime[y];**

**}**

**printf("The priority can be computed as : Priority = 1+ Waiting time / Estimated run time\n\n");**

**for(y=0;y<n;y++)**

**{**

**priority[y] = 1+waitingTime[y]/completionTime[y];**

**printf("Priority of P[%0.0lf] = %lf \n",process[y],priority[y]);**

**}**

**printf("\n\n \*\*\*\*\*\*\*\*\*\*\*\*\*\* Scheduling(Prioritising) the Processes \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");**

**printf("\* Process \* Arrival Time \* Burst Time \* Waiting Time \* Turn Around Time \*\n");**

**printf("\* P[%0.0lf] \* %0.0lf \* %0.0lf \* %0.0lf \* %0.0lf \*\n",**

**process[0],arrivalTime[0],burstTime[0],waitingTime[0],turnaroundTime[0]);**

**for(y=n-1;y>0;y--)**

**{**

**printf("\* P[%0.0lf] \* %0.0lf \* %0.0lf \* %0.0lf \* %0.0lf \*\n",**

**process[y],arrivalTime[y],burstTime[y],waitingTime[y],turnaroundTime[y]);**

**}**

**printf("\n\nAVG waiting time = Sum of waiting time of all the processes/total no of processes\n");**

**printf("AVG Turn around time = Sum of Turn around time of all the processes/total no of processes.\n");**

**printf("\n\n\n\t\t\tAverage Turn Around Time : %lf",turnaround\_avg);**

**printf("\n\t\t\tAverage Waiting Time : %lf\n\n",wait\_avg);**

**printf("\n\t\t\t\t\tEnd Of Program");**

**printf("\n\t\t\t\t\t\*\*\*\*\*\*");**

**}**

**}**

**int main()**

**{**

**proc\_inp();**

**printf("\n\t\t\t\t\tEnd Of Program");**

**printf("\n\t\t\t\t\t\*\*\*\*\*\*\*\*\*\*\*");**

**printf("\n\t\t\t\t\tMADE BY Saurabh Tripathi");**

**printf("\n\t\t\t\t\tRoll no 02 , 11807532");**

**}**

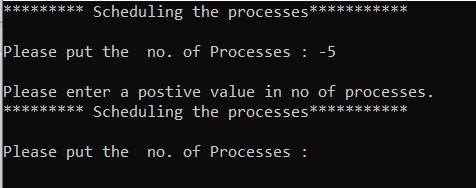
**5.) Boundary condition And Constraints of implemented code:**

* Entered value should be a positive integer in the field of number of values.
* Burst time And Arrival time should be positive or equal to zero.
* Calculated Priority should be positive. More is the calculated priority value of the process higher its order in priority table.

**6.) Test cases:**

**Case 1:**

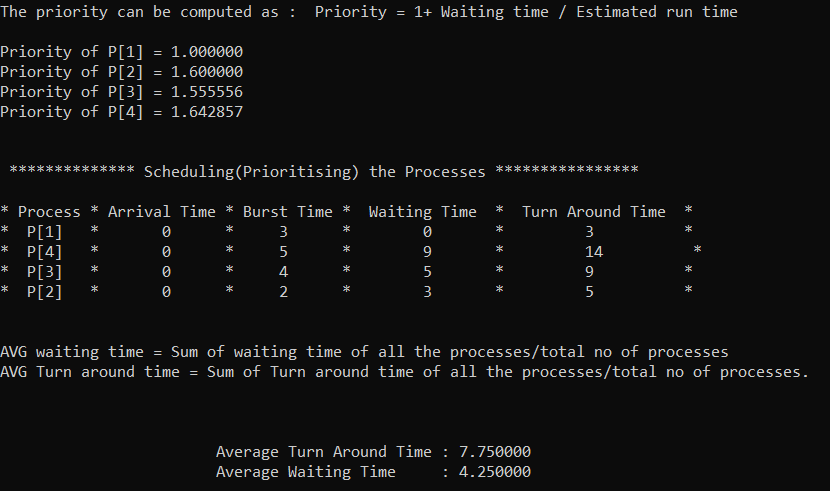
If entered value in number of processes is negative(or zero), program will throw exception and ask again to enter.



**Result-Program handles exception very smoothly.**

**Case 2:**

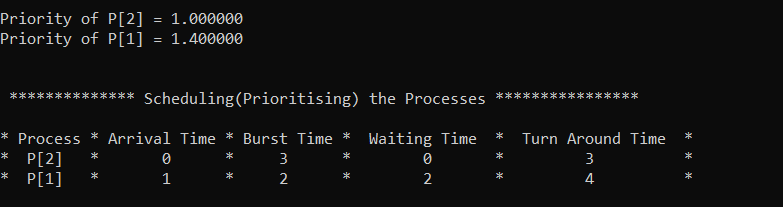
If all the proceses comes at T=0(i.e. at Arrival time 0).



**Result-Program gives right order of scheduling.**

**Case 3:**

If the first process not comes at T=0, then next process in priority queue is assigned the CPU.



**Result-Program working fine.**

**7.) GITHUB LINK:**

<https://github.com/beingsaurabh/os-project>

End of Report