CASE STUDY

CSE316: Operating Systems

Bharath Goud

11810473

Email: chinnabharat139@gmail.com

GitHub: https://github.com/bharatgoud21

Code : 7

Question:

Design a scheduling program to implements a Queue with two levels:
Level 1 : Fixed priority pre-emptive Scheduling
Level 2 : Round Robin Scheduling
For a Fixed priority pre-emptive Scheduling (Queue 1), the Priority 0 is highest priority. If one process P1 is scheduled and running , another process P2 with higher priority comes. The New process (high priority) process P2 preempts currently running process P1 and process P1 will go to second level queue. Time for which process will strictly execute must be considered in the multiples of 2..
All the processes in second level queue will complete their execution according to round robin scheduling.
Consider: 1. Queue 2 will be processed after Queue 1 becomes empty.
2. Priority of Queue 2 has lower priority than in Queue 1.

Description:

Scheduling of processes/work is done to finish the work on time. A typical process involves both I/O time and CPU time. In a uni programming system like MS-DOS, time spent waiting for I/O is wasted and CPU is free during this time. In multi programming systems, one process can use CPU while another is waiting for I/O. This is possible only with process scheduling.

Arrival Time: Time at which the process arrives in the ready queue.
Completion Time: Time at which process completes its execution.
Burst Time: Time required by a process for CPU execution.
Turn Around Time: Time Difference between completion time and arrival time.

$$\textit{Turn Around Time} = \textit{Completion Time} - \textit{Arrival Time}$$

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

$$\textit{Waiting Time} = \textit{Turn Around Time} - \textit{Burst Time}$$

Objectives of Process Scheduling Algorithm
Max CPU utilization [Keep CPU as busy as possible]
Fair allocation of CPU.
Max throughput [Number of processes that complete their execution per time unit]
Min turnaround time [Time taken by a process to finish execution]
Min waiting time [Time a process waits in ready queue]
Min response time [Time when a process produces first response]

Algorithm:

- In this problem algorithm for round robin scheduling and multilevel queue scheduling is used.

Algorithm for round robin scheduling:

1- Create an array rem_bt[] to keep track of remaining burst time of processes. This array is initially a copy of bt[] (burst times array)

2- Create another array wt[] to store waiting times of processes. Initialize this array as 0.

3- Initialize time : t = 0

4- Keep traversing the all processes while all processes are not done. Do following for i'th process if it is not done yet.

a- If rem_bt[i] > quantum

(i) t = t + quantum

(ii) bt_rem[i] -= quantum;

c- Else // Last cycle for this process

(i) t = t + bt_rem[i];

(ii) wt[i] = t - bt[i]

(ii) bt_rem[i] = 0; // This process is over

Algorithm For Multilevel Queue:

1. When a process starts executing then it first enters queue 1.
2. In queue 1 process executes for 4 unit and if it completes in this 4 unit or it gives CPU for I/O operation in this 4 unit than the priority of this process does not change and if it again comes in the ready queue than it again starts its execution in Queue 1.
3. If a process in queue 1 does not complete in 4 unit then its priority gets reduced and it shifted to queue 2.
4. Above points 2 and 3 are also true for queue 2 processes but the time quantum is 8 unit.In a general case if a process does not complete in a time quantum than it is shifted to the lower priority queue.
5. In the last queue, processes are scheduled in FCFS manner.
6. A process in lower priority queue can only execute only when higher priority queues are empty.
7. A process running in the lower priority queue is interrupted by a process arriving in the higher priority queue.

Code:

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
int main()
{
   char p[10][5],temp[5];
   int i,j,pt[10],wt[10],totwt=0,pr[10],temp1,n;
   float avgwt;
   printf("enter no of processes:");
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
      printf("enter process%d name:",i+1);
      scanf("%s",&p[i]);
      printf("enter process time:");
      scanf("%d",&pt[i]);
      printf("enter priority:");
      scanf("%d",&pr[i]);
   }
   for(i=0;i<n-1;i++)
   {
      for(j=i+1;j<n;j++)
      {
         if(pr[i]>pr[j])
         {
            temp1=pr[i];
            pr[i]=pr[j];
            pr[j]=temp1;
            temp1=pt[i];
            pt[i]=pt[j];
            pt[j]=temp1;
            strcpy(temp,p[i]);
            strcpy(p[i],p[j]);
            strcpy(p[j],temp);
         }
      }
   }
   wt[0]=0;
   for(i=1;i<n;i++)
   {
      wt[i]=wt[i-1]+wt[i-1];
      totwt=totwt+wt[i];
```

```
}
avgwt=(float)totwt/n;
printf("p_name\t p_time\t priority\t w_time\n");
for(i=0;i<n;i++)
{
  printf(" %s\t %d\t %d\t %d\n" ,p[i],pt[i],pr[i],wt[i]);
}
printf("total waiting time=%d\n avg waiting time=%f",totwt,avgwt);

int ts,pid[10],need[10],wt1[10],tat[10],i1,j1,n2,n1;
int bt[10],flag[10],ttat=0,twt=0;
float awt,atat;
printf("\nEnter the number of Processors \n");
scanf("%d",&n);
n1=n;
printf("\n Enter the Timeslice \n");
scanf("%d",&ts);
for(i=1;i<=n;i++)
{
   printf("\n Enter the process ID %d",i);
   scanf("%d",&pid[i]);
   printf("\n Enter the Burst Time for the process");
   scanf("%d",&bt[i]);
   need[i]=bt[i];
}
for(i=1;i<=n;i++)
{
   flag[i]=1;
   wt[i]=0;
}
while(n!=0)
{
   for(i=1;i<=n;i++)
   {
      if(need[i]>=ts)
      {
         for(j=1;j<=n;j++)
         {
            if((i!=j)&&(flag[i]==1)&&(need[j]!=0))
            wt[j]+=ts;
         }
         need[i]-=ts;
         if(need[i]==0)
```

```
            {
               flag[i]=0;
               n--;
            }
         }
         else
         {
            for(j=1;j<=n;j++)
            {
               if((i!=j)&&(flag[i]==1)&&(need[j]!=0))
               wt[j]+=need[i];
            }
            need[i]=0;
            n--;
            flag[i]=0;
         }
      }
   }
   for(i=1;i<=n1;i++)
   {
      tat[i]=wt[i]+bt[i];
      twt=twt+wt[i];
      ttat=ttat+tat[i];
   }
   awt=(float)twt/n1;
   atat=(float)ttat/n1;
   printf("\n\n Process \t Process ID  \t BurstTime \t Waiting Time \t TurnaroundTime \n ");
   for(i=1;i<=n1;i++)
   {
      printf("\n %5d \t %5d \t\t %5d \t\t %5d \t\t %5d \n", i,pid[i],bt[i],wt[i],tat[i]);
   }
   printf("\n The average Waiting Time=4.2f",awt);
   printf("\n The average Turn around Time=4.2f",atat);
}
```

Boundary conditions:

Level 1 : Fixed priority preemptive Scheduling

Level 2 : Round Robin Scheduling

Consider: 1. Queue 2 will be processed after Queue 1 becomes empty.

Consider 2. Priority of Queue 2 has lower priority than in Queue 1.

Complexities:

for loop 1: O(n)

for loop 2: n^2

for loop 3: O(n)

for loop 4: O(n)

for loop 5: O(n)

while loop : $O(n^3)$

Total Complexity: $n+n^2+n+n+n+n^3 = O(n^3)$

Test Cases:

Test case no(1)

Input Values :

Processes:1

Process Name: Chrome

Process time: 3

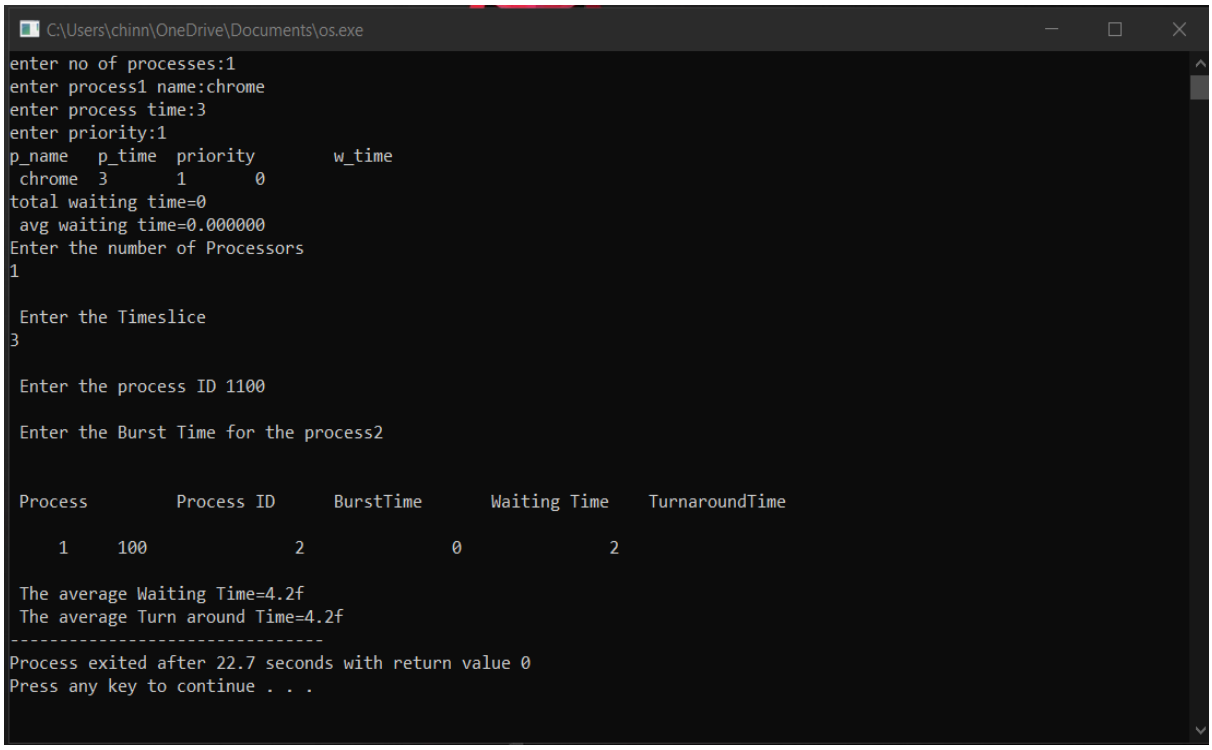
Priority : 1

Number of Processes: 1

Time Slice: 3

Process ID:100

Burst Time: 2



Test Case No(2):

Input Values:

Processes:1

Process Name: Explorer

Process time: 3

Priority : 2

Number of Processes: 2

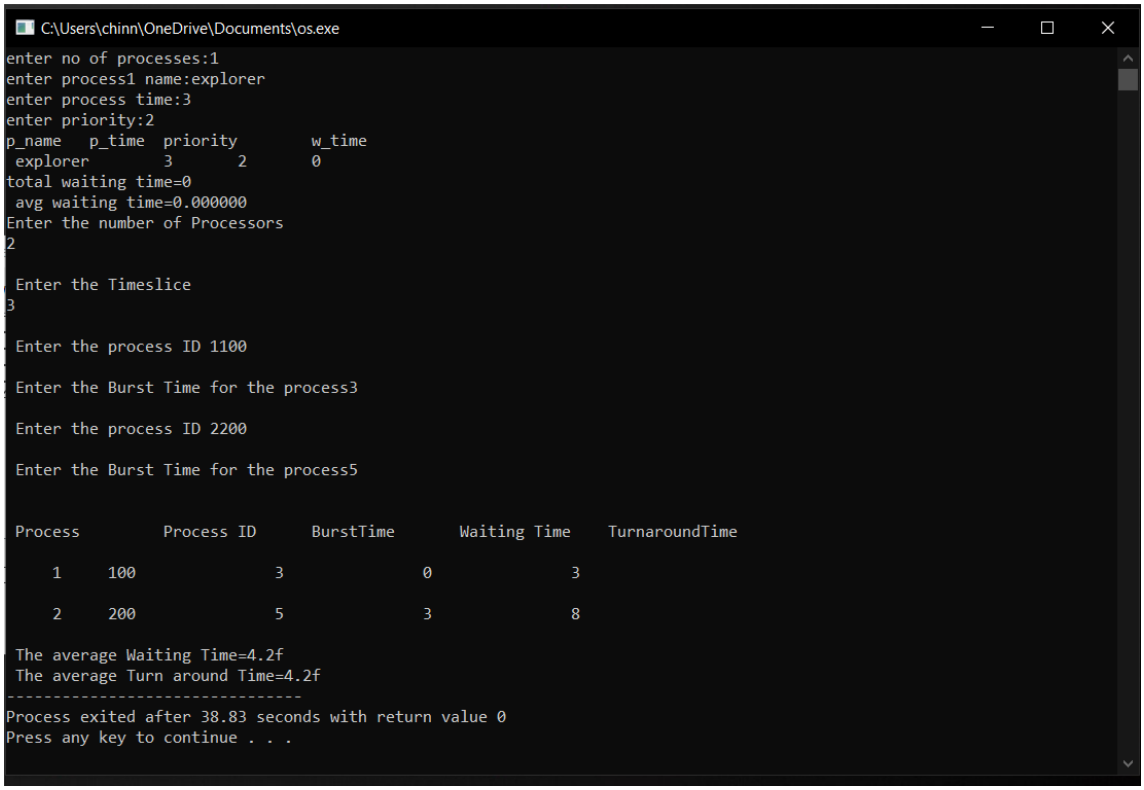
Time Slice: 3

Process ID1:100

Burst Time: 3

Process ID2:200

Burst Time: 5



Git Hub Link is: https://github.com/bharatgoud21/bharathgoud316