Name:Prashant Kumar

Reg. No.:11701223

Email id:prashantkr228@gmail.com

Github link: https://github.com/ps3vedi/O_S/tree/master

ASSIGNMENT

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.

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;
}
```

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.

Shortest remaining processing time first algorithm is preemptive mode of SJF algorithm in which jobs are schedule according to shortest remaining time.

Multilevel queue may happen when processes is in the ready queue can be divided into different classes where each class has its own scheduling needs. For example, a common division is a **foreground (interactive)** process and **background (batch)** processes.These two classes have different scheduling needs. For this kind of situation Multilevel Queue Scheduling is used.

Each queue has absolute priority over lower priority queue. Let us consider following priority order **queue 1 > queue 2**.According to this algorithm no process in the batch queue can run unless queue 1 and 2 are system (queue 1) or Interactive process(queue 2) entered the ready queue the batch process is preempted.

<u>ALGORITHM</u>

1. System Processes
2. Interactive Processes
3. Interactive Editing Processes
4. Batch Processes
5. Student Processes

Each queue has absolute priority over lower-priority queues. No process in the batch queue, for example, could run unless the queues for system processes, interactive processes, and interactive editing processes were all empty. If an interactive editing process entered the ready queue while a batch process was running, the batch process will be preempted.