

Assignment on Operating Systems

Course code ------CSE316

Submitted by

BATHINA AAKASH REDDY

Reg No – 11707370

Section --- EE028

Roll No ---A20

Email – akashbabblu9@gmail.com

GitHub Account -

Submitted to

Manjit Kaur

Faculty of Operating Systems

Lovely Professional University

Question:

1. 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;

}

Test case:

