**Student Name:** Manideep Alla

**Student ID:** 11808651

**Email Id: manideepalla0208@gmail.com**

**GitHub Link:** https://github.com/??????/Os-project.git

Problem:

**Ques 9**. Design a scheduler that uses a preemptive priority scheduling algorithm based on

dynamically changing priority. Larger number for priority indicates higher priority.

Assume that the following processes with arrival time and service time wants to execute (for

reference):

ProcessID Arrival Time Service Time

P1 0 4

P2 1 1

P3 2 2

P4 3 1

When the process starts execution (i.e. CPU assigned), priority for that process changes at the

rate of m=1.When the process waits for CPU in the ready queue (but not yet started execution),

its priority changes at a rate n=2. All the processes are initially assigned priority value of 0 when

they enter ready queue for the first time . The time slice for each process is q = 1. When two

processes want to join ready queue simultaneously, the process which has not executed recently

is given priority. Calculate the average waiting time for each process. The program must be

generic i.e. number of processes, their burst time and arrival time must be entered by user.

**Ans:** We can solve this question by using two algorithms of Operating System:

1. Round Robin Scheduling Algorithm : for fixed time slice i.e for X unit and Y unit of time

Algorithm:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue along with arrival time and burst time.

Step 3: For each process in the ready Q, assign the process with a time slice of X unit in first iteration.

Step 4: For each process again assign the process with a time slice of Y unit in second iteration.

Step 5: Assign Processes with least execution time i.e CPU is assigned to process with less CPU burst time

Step 6: calculate:

Waiting time for process(n) = waiting time of process(n-1)+ burst time of process(n-1 )+ the time difference in getting the CPU from process(n-1)

1. Turn around time for process(n) = waiting time of process(n) + burst time of process(n)+ the time difference in getting CPU from process(n).

Step 7: Calculate

1. Average waiting time = Total waiting Time / Number of process
2. Average Turnaround time = Total Turnaround Time / Number of process Step 8: Stop the process

Description:

This problem is done in Round robin algorithm and here are some of its characteristics

* Round robin is a pre-emptive algorithm
* The CPU is shifted to the next process after fixed interval time, which is called time quantum/time slice.
* The process that is preempted is added to the end of the queue.
* Round robin is a hybrid model which is clock-driven
* Time slice should be minimum, which is assigned for a specific task that needs to be processed. However, it may differ OS to OS.
* It is a real time algorithm which responds to the event within a specific time limit.
* Round robin is one of the oldest, fairest, and easiest algorithm.
* Widely used scheduling method in traditional OS.

Code:

|  |
| --- |
| #include<stdio.h>  #include<conio.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; |
|  | } |

Output:

