**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.

**Definition About Problem in terms of OS:**

**Dynamic priority scheduling** is a type of scheduling algorithm in which the priorities are calculated during the execution of the system. The goal of dynamic priority scheduling is to adapt to dynamically changing progress and form an optimal configuration in self-sustained manner. It can be very hard to produce well-defined policies to achieve the goal depending on the difficulty of a given problem.

Earliest deadline first scheduling and Least slack time scheduling are examples of Dynamic priority scheduling algorithms.

**Algorithm:**

Real-time kernels generally support 256 priority levels, in which 0 is the highest and 255 the lowest. Some kernels appoint the priorities in reverse order, where 255 is the highest and 0 the lowest. Regardless, the concepts are basically the same. With a pre-emptive priority-based scheduler, each task has a priority, and the highest-priority task runs first. If a task with a priority higher than the current task becomes ready to run, the kernel immediately saves the current task’s context in its TCB and switches to the higher-priority task. task 1 is pre-empted by higher-priority task 2, which is then pre-empted by task 3. When task 3 completes, task 2 resumes; likewise, when task 2 completes, task 1 resumes.

Although tasks are assigned a priority when they are created, a task’s priority can be changed dynamically using kernel-provided calls. The ability to change task priorities dynamically allows an embedded application the flexibility to adjust to external events as they occur, creating a true real-time, responsive system. Note, however, that misuse of this capability can lead to priority inversions, deadlock, and eventual system failure.

**Purpose of use:**

In the scheduling of programs to the cpu. pre-emptive scheduling helps the cpu to switch between processes even without their full/complete completion. This pre-emptive method uses priority of the processes and the priority dynamically changes making the cpu much harder to keep track and shift between processes.

**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; |
|  | } |
|  |  |

