OS Simulation Based Assignment- CA3

Name	Nishant Pandey
Registration No	11804362
Section	K18NS
Roll.No	65
Course Code	CSE-316
Submitted to	Ms. Suruchi Talwani
Github Link	
E-mail	Nishant201pandey@gmail.com

Question Assigned:

Write a program to implement priority scheduling algorithm with context switching time. Prompt to user to enter the number of processes and then enter their priority, burst time and arrival time also. Now whenever operating system preempts a process and shifts CPU's control to some another process of higher priority assume that it takes 2 seconds for context switching (dispatcher latency). Form a scenario, where we can give the processes are assigned with priority where the lower integer number is higher priority and then context switch. as the process waits the priority of the process increase at rate of one per 2-time units of wait. Calculate waiting time and turnaround time for each process.

DESCRIPTION:

Priority scheduling algorithm

Priority Scheduling is a method of scheduling processes that is based on priority. In this algorithm, the scheduler selects the tasks to work as per the priority.

The processes with higher priority should be carried out first, whereas jobs with equal priorities are carried out on a round-robin or FCFS basis. Priority depends upon memory requirements, time requirements, etc.

Priorities can be defined either internally or externally. Internally defined priorities use some measurable quantity or quantities to compute the priority of a process. For example, time limits, memory requirements, the number of open files, and the ratio of average I/O burst to average CPU burst

have been used in computing priorities. External priories are set by criteria outside the operating system, such as the important of the process, the type and amount of funds being paid for computer use, the department sponsoring the work, and other, often political, factors.

Priority scheduling can be either preemptive or non-preemptive. When a process arrives at the ready queue, its priority is compared with the current running process. A preemptive scheduling algorithm will preempt the CPU if the priority of the newly arrived process is higher than the priority of the currently running process. A non-preemptive scheduling algorithm will simply put the new process at the head of the ready queue.

ALGORITHM:

Step 1: Assign the process to ready queue.

Step 2: Assign the process to the CPU according to the priority, higher priority process will get the CPU first than lower priority process.

Step 3: If two processes have similar priority then SJF is used to break the tie.

Step 4: Repeat the step 1 to 3 until ready queue is empty.

Step 5: Calculate Waiting time and Turnaround time of individual Process.

Step 6: Calculate Average waiting time and Average Turnaround time.

COMPLEXITY:

At any given instance, the ready queue will be having at most n elements. For a priority scheduling algo, we need to find the element of highest priority. This problem is reduced to a search problem of an unsorted queue. The worst-case complexity of such a problem is O(n)But we can also see that if we implement

the problem using priority queue (max-heap), then the possible operations could be:

1. Max- happify: O(log(n))

2. Find-max: O(log(n))

This implementation reduces the overall complexity to O(log(n))

So, answer (b) O(log(n))

Code Snippet for assigned question:

```
#include<stdio.h>
#include<conio.h>
int main()
{
   int B_T[10],Process_no[10],W_T[10],T_A_T[10];
   int
A_T[10],Prior[10],i,j,Number_of_Process,tot=0,flag,temp,Avg_W_T,
Avg_T_A_T;
   printf ("Enter Number of Process:");
   scanf("%d",&Number_of_Process);
   printf ("\nEnter Burst Time and Prior\n");
   for(i=0;i<Number_of_Process; i++)
   {
      printf("\nP[%d]\n",(i+1));
      printf("Enter Burst Time:");
      scanf ("%d”, &B_T[i]);
      printf("Enter Arrival Time:");
      scanf("%d”, &A_T[i]);
      printf("Enter Prior:");
      scanf("%d”, &Prior[i]);
      Process_no[i]=i+1;
   }
   for (i=0;i<Number_of_Process; i++)
```

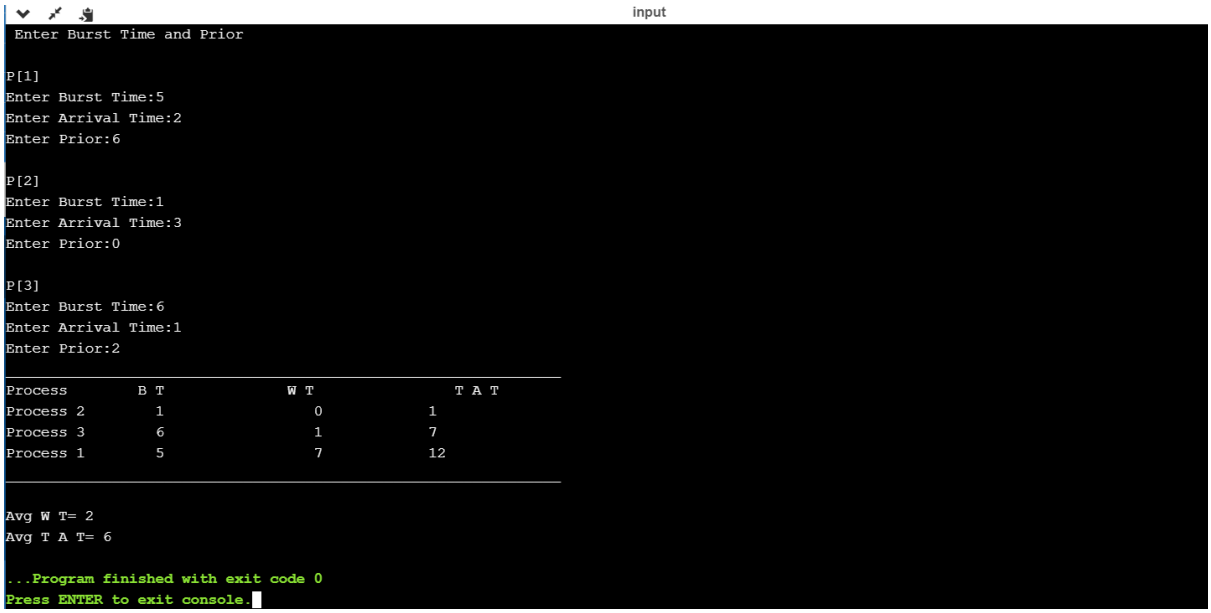
```
    {
        flag=i;
        for (j=i+1; j<Number_of_Process;j++)
        {
            if(Prior[j]<Prior[flag])
            flag=j;
        }
        temp=Prior[i];
        Prior[i]=Prior[flag];
        Prior[flag]=temp;
        temp=B_T[i];
        B_T[i]=B_T[flag];
        B_T[flag]=temp;
        temp=Process_no[i];
        Process_no[i]=Process_no[flag];
        Process_no[flag]=temp;
    }
    W_T[0]=0;
    for(i=1;i<Number_of_Process;i++)
    {
        W_T[i]=0;
        for(j=0;j<i;j++)
            W_T[i]+=B_T[j];
        tot+=W_T[i];
    }
    Avg_W_T=tot/Number_of_Process;
    tot=0;

printf("_____
_____");
    printf("\nProcess\t       B T              W T              T A T");
    for(i=0;i<Number_of_Process;i++)
    {
        T_A_T[i]=B_T[i]+W_T[i];
        tot+=T_A_T[i];
        printf ("\nProcess %d \t %d\t \t
%d\t\t%d",Process_no[i],B_T[i],W_T[i],T_A_T[i]);
```

```
    }

printf("\n_____
_____");
    Avg_T_A_T=tot/Number_of_Process;
    printf("\n\nAvg W T= %d",Avg_W_T);
    printf("\nAvg T A T= %d",Avg_T_A_T);
    return 0;
}
```

Sample result:



BOUNDARY CONDITIONS:

priority scheduling algorithm, a major problem to be considered is the starvation of a process i.e. a process which is ready to be executed but is waiting for the CPU because of its low priority. This can lead to the indefinite waiting of the

low-priority processes. A process that is ready to run but waiting for the CPU can be considered blocked. A priority scheduling algorithm can leave some low priority processes waiting indefinitely. In a heavily loaded computer system, a steady stream of higher priority processes can prevent a low priority processes from never getting the CPU. A solution to the problem of indefinite blockage is aging. Aging involves gradually increasing the priority of processes that wait in the system for a long time

Test cases:

Test case 1:

processes	priority	Arrival time	Burst time	Waiting time	Turnaround time
Process 1	8	2	2	11	13
Process 2	2	1	5	0	5
Process 3	6	0	6	5	11

Average turnaround time:9 expected output: p2, p3, p1

Average waiting time:5 original output: p2, p3, p1

Test case passed.

Test case 2:

processes	priority	Arrival time	Burst time	Waiting time	Turnaround time
Process 1	2	0	2	5	7
Process 2	0	5	5	0	5
Process 3	6	2	3	7	10

Average turnaround time:7 expected output: p2, p1, p3

Average waiting time:4 original output: p2, p1, p3

Test case passed

Test case 3:

processes	priority	Arrival time	Burst time	Waiting time	Turnaround time
Process 1	5	3	3	10	13
Process 2	2	1	6	0	6
Process 3	3	0	4	6	10

Average turnaround time:9 average waiting time:5

Expected output: p2, p3, p1 original output: p2, p3, p1

Test case passed.