

OS PROJECT

PRIORITY SCHEDULING (LARGER THE PRIORITY HIGHER THE NUMBER OF PRIORITY)

SUBMITTED BY: SANJANA

REGISTRATION NUMBER: 11812668

SECTION:K18PA

ROLL NO.:19

GITHUB LINK: <https://github.com/sanjana17082001/OS-Assignment>

SUBMITTED TO: MRS. SURUCHI TALWANI

Question: Design a scheduler that uses a preemptive priority scheduling algorithm based on dynamically changing priority. Larger number for priority indicates higher priority.

Assume that following processes with arrival time and service time wants to execute(for reference):

PID	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 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.

CPU Scheduling is a process which allows one process to use the CPU while the execution of another process is on hold (or in waiting state) due to unavailability of any resource like I/O etc. The main aim of CPU scheduling is to maximise the utilization of the CPU, which makes the system efficient, fast and fair.

There are six types of scheduling:

- 1.FCFS(First Come First Serve)
- 2.SJF(Shortest Job First)
- 3.SRT(Shortest Remaining Time)
- 4.Priority Scheduling
- 5.Round Robin Algorithm

PRIORITY ALGORITHM :

Priority scheduling is associated with each process.

At any instance of time out of all available process, CPU is allocated to the process which process the highest priority.

It supports both preemptive and non-preemptive version.

The priority of process is selected on the basis of memory requirement , user preference or the requirement of the time. Processes are executed on the basis of priority. So high priority does not need to wait for long which saves time. It is easy to use.

ALGORITHM AND FORMULAS USED TO SOLVE THE PROBLEM:

FORMULAS:

$TAT(\text{Turn Around Time}) = \text{Completion Time(CT)} - \text{Arrival Time(AT)}$

$\text{Wait Time(WT)} = \text{Turn Around Time(TAT)} - \text{Burst Time(BT)}$

- 1.Priority is assigned to each process.
- 2.Process with highest priority is executed first and so on.
- 3.Process with same priority is executed in FCFS manner.
- 4.Priority can be decided based on memory requirements, time requirements or any other resource requirement.

ANSWER:

Enter number of process :4

Arrival time of P[1]: 0

Service Time of P[1]: 4

Enter priority: 1

Arrival time of P[2]: 1

Enter priority : 2

Arrival Time of P[3] : 2

Service time of P[3]:2

Enter priority : 3

Arrival time of P[4]: 3

Service time of P[4]: 1

Enter priority : 4

Process	Turn Around Time
P1	2
P2	3
P3	1
P4	4

The average turn around time is 2.500000

Output for Given Program:-

```

Enter Arrival Time: 0
Enter Burst Time: 4
Enter Priority: 1

Enter Details For Process[B]:
Enter Arrival Time: 1
Enter Burst Time: 1
Enter Priority: 2

Enter Details For Process[C]:
Enter Arrival Time: 2
Enter Burst Time: 2
Enter Priority: 3

Enter Details For Process[D]:
Enter Arrival Time: 3
Enter Burst Time: 1
Enter Priority: 4

Process Name    Arrival Time    Burst Time    Priority    Waiting Time
A               0               4             1           0
D               3               1             4           1
C               2               2             3           3
B               1               1             2           6

Average waiting time: 2.500000
Average Turnaround Time: 4.500000

```

TEST CASES:

CASE 1:

PROCESS	ARRIVAL TIME	BURST TIME	WAITING TIME
P1	0	6	0
P2	1	8	5
P3	9	5	5

AVG TURN AROUND TIME: 9.6666

AVG WAITING TIME:3.333

CASE 2:

PROCESS TIME	ARRIVAL TIME	BURST TIME	WAITING
P1	0	4	0
P2	2	5	3
P3	3	9	7
P4	5	2	7
P5	4	1	11
P6			
	1	7	21

AVG TURN AROUND TIME:12.8333

AVG WAITING TIME : 8.16

CASE 3:

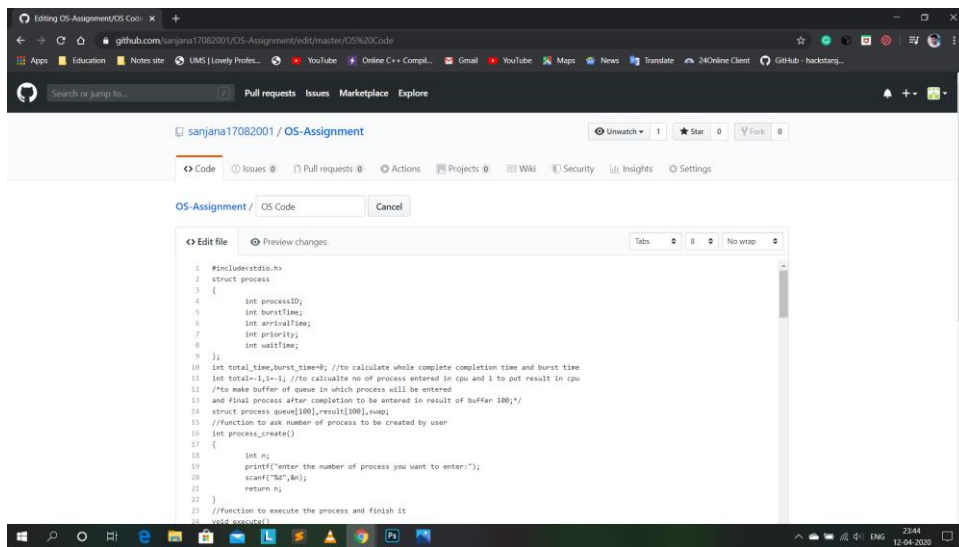
PROCESS TIME	ARRIVAL TIME	BURST TIME	WAITING
P1	0	5	0
P2	1	3	0
P3	2	2	9
P4			
	5	6	12

AVG TURN AROUND TIME:9.25000

AVERAGE WAITING TIME: 5.25000

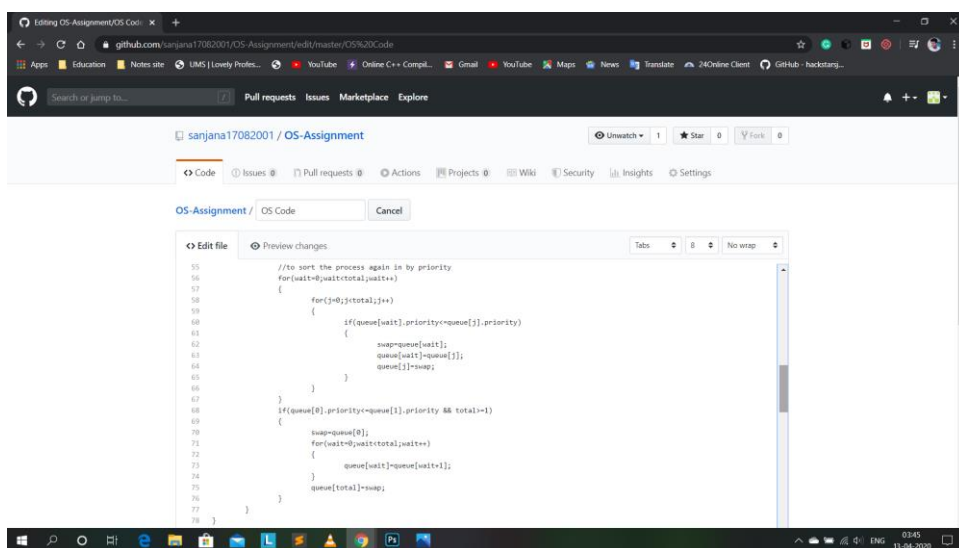
BOUNDARY CONDITION:

The main boundary condition is to execute the process maximum of 2 units time and to then it holds the process after the 2 units. In this period of time the other process will execute which has less arrival time as compare to previous holding processes and it will go on with this boundary condition. If the time limit exceeds for a single process among all processes then the program or the output will become incorrect.



```
1 #include<stdio.h>
2 struct process
3 {
4     int processID;
5     int burstTime;
6     int arrivalTime;
7     int priority;
8     int waitTime;
9 };
10 int total_time,burst_time=0; //to calculate whole complete completion time and burst time
11 int total=1,i=1; //to calculate no of process entered in cpe and 1 to put result in cpe
12 //to make buffer of queue in which process will be entered
13 and final process after completion to be entered in result of buffer 100*/
14 struct process queue[100],result[100],swap;
15 //function to ask number of process to be created by user
16 int process_create()
17 {
18     int n;
19     printf("enter the number of process you want to enter:");
20     scanf("%d",&n);
21     return n;
22 }
23 //function to execute the process and finish it
24 void execute()
```

Date : 12-04-2020



```
55 //to sort the process again in by priority.
56 for(wait=0;wait<total;wait++)
57 {
58     for(j=0;j<total;j++)
59     {
60         if(queue[wait].priority<queue[j].priority)
61         {
62             swap=queue[wait];
63             queue[wait]=queue[j];
64             queue[j]=swap;
65         }
66     }
67     if(queue[0].priority<queue[1].priority && total!=1)
68     {
69         swap=queue[0];
70         queue[0]=queue[total-1];
71         queue[total-1]=swap;
72         for(wait=0;wait<total;wait++)
73         {
74             queue[wait]=queue[wait+1];
75         }
76         queue[total]=swap;
77     }
78 }
```

Date : 13-04-2020

The screenshot shows a GitHub repository page for 'sanjana17082001 / OS-Assignment'. The repository has 1 Unwatch, 1 Star, and 0 Forks. The 'Code' tab is selected, and the 'OS-Assignment / OS Code' file is open in the editor. The code is a C++ implementation of a priority queue-based scheduling algorithm. It includes a function to sort the process again by priority and a function to execute the process. The code is as follows:

```
55 //To sort the process again by priority
56 for(wait=0;wait<total;wait++)
57 {
58     for(j=0;j<total;j++)
59     {
60         if(queue[wait].priority<queue[j].priority)
61         {
62             swap=queue[wait];
63             queue[wait]=queue[j];
64             queue[j]=swap;
65         }
66     }
67 }
68 if(queue[0].priority<queue[1].priority && total!=1)
69 {
70     swap=queue[0];
71     for(wait=0;wait<total;wait++)
72     {
73         queue[wait]=queue[wait+1];
74     }
75     queue[total]=swap;
76 }
77 }
78 }
```

Date : 15-04-2020

The screenshot shows the same GitHub repository page for 'sanjana17082001 / OS-Assignment'. The 'Code' tab is selected, and the 'OS-Assignment / OS Code' file is open in the editor. The code is a C++ implementation of a process execution table and average waiting time calculation. It includes a function to execute the process and a function to calculate the average waiting time. The code is as follows:

```
146 while(burst_time!=0 && count!=n)
147 {
148     execute();
149     total_time++;
150 }
151 if(count==n)
152     break;
153 }
154 printf("PROCESS IN ORDER OF THEIR COMPLETION:\n");
155 printf("..... FINAL PROCESS EXECUTION TABLE :.....\n");
156 printf(".....\n");
157 printf("..... PROCESS ID ..... AVAIL TIME ..... SERVICE TIME ..... WAITING TIME.....\n");
158 printf(".....\n");
159 for(i=0;i<n;i++)
160 {
161     for(j=0;j<n;j++)
162     {
163         if(result[i].processID==pcreate[j].processID)
164         {
165             printf("..... %d ..... %d ..... %d ..... %d\n",result[i].processID,pcreate[j].processID,
166                 result[i].avail_time,result[i].service_time,result[i].waiting_time);
167             break;
168         }
169     }
170     averageWaitingTime+=result[i].waitingTime;
171 }
172 printf("AVERAGE WAITING TIME : %f\n",averageWaitingTime/n);
173 }
```

Date : 18-04-2020