**PRIORITY SCHEDULING ALGORITHM WITH CONTEXT SWITCHING**

*Mini Project submitted in partial fulfilment of the requirements for the Degree of*

**BACHELORS OF TECHNOLOGY**

**in**

**COMPUTER SCIENCE AND ENGINEERING**

By

**Abhishek Pawar**

**11705474**

**K17HU | 36**

Under the Guidance of

**Ms. Priya Virdi Mam**

**ASSISSTANT PROFESSOR**



**School of Computer Science and Engineering**

Lovely Professional University

Phagwara, Punjab (India)

**Email:**

[**ap9407073180@gmail.com**](mailto:ap9407073180@gmail.com)

**GitHub Link:**

**https://github.com/ap9407073180**

**1. Explain the problem in terms of Operating System Concepts.**

Strick priority scheduling will give CPU time to the highest priority operation that is ready to run. Starvation happens when the machine has been given too much work and no time is left for some of the lower priority operations. The most common (in my experience) cause of this is for some operation to waste processor time waiting for some real work to do, with priority scheduling it is important that every process ‘Block’ (wait for something, giving time to other processes) when it doesn’t have real work to do.Time taken by the scheduler to suspend the running task, switch the context, and dispatch the new incoming task.Indefinite blocking or starvation.A priority scheduling can leave some low priority waiting processes indefinitely for CPU.If the system eventually crashes then all unfinished low priority processes gets lost. If starvation isn’t caused by that, but by actually having too much work to get done, then the only solutions are to either reduce the amount of work that needs to get done or increase the power of the processor. One patch that can be done is to artificially increase the priority of operations that haven’t gotten time for awhile. Starvation doesn’t occur in ‘normal’ operating systems because rather than always giving time to the ‘highest priority’ operation, they share time with all operations that are available, maybe giving more time to those marked important and less to those marked unimportant. On such a system, when it is ‘overloaded’ things just run slow.

**2. Write the algorithm for the proposed solution of the assigned problem.**

* **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 FCFS 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.

**3. Calculate complexity of implemented algorithm.**

O (log(n))

**4. Explain all the constraints given in the problem. Attach**

**the code snippet of the implemented constraint.**

#include<stdio.h>

#include<conio.h>

struct processinfo

{

int status,waiting\_time, turnaround\_time, priority,arrival\_time, burst\_time, compltime;

char process\_name;

}process\_queue[10];

int max;

void ArrivalTime\_ordering()

{

struct processinfo temp;

int m, n;

for(m = 0; m < max - 1; m++)

{

for(n = m + 1; n < max; n++)

{

if(process\_queue[m].arrival\_time > process\_queue[n].arrival\_time)

{

temp = process\_queue[m];

process\_queue[m] = process\_queue[n];

process\_queue[n] = temp;

}

}

}

}

void main()

{

int i, time = 0, burst\_time = 0, largest;

char c;

float waiting\_time = 0, turnaround\_time = 0, avg\_waiting\_time,

avg\_turnaround\_time;

printf("\nEnter Total Number of Processes:\t");

scanf("%d", &max);

for(i = 0, c = 'A'; i < max; i++, c++)

{

process\_queue[i].process\_name = c;

printf("\nEnter Process Details[%C]:\n", process\_queue[i].process\_name);

printf("Enter Arrival Time:\t");

scanf("%d", &process\_queue[i].arrival\_time );

printf("Enter Burst Time:\t");

scanf("%d", &process\_queue[i].burst\_time);

printf("Enter Priority:\t");

scanf("%d", &process\_queue[i].priority);

process\_queue[i].status = 0;

burst\_time = burst\_time + process\_queue[i].burst\_time;

}

ArrivalTime\_ordering();

process\_queue[9].priority = -9999;

printf("\nProcess Name\tArrival Time\tBurst Time\tPriority\tWaiting Time");

for(time = process\_queue[0].arrival\_time; time < burst\_time;)

{

largest = 9;

for(i = 0; i < max; i++)

{

if(process\_queue[i].arrival\_time <= time && process\_queue[i].status != 1 && process\_queue[i].priority > process\_queue[largest].priority)

{

largest = i;

}

}

time = time + process\_queue[largest].burst\_time;

process\_queue[largest].compltime = time;

process\_queue[largest].waiting\_time = process\_queue[largest].compltime –

process\_queue[largest].arrival\_time - process\_queue[largest].burst\_time;

process\_queue[largest].turnaround\_time =

process\_queue[largest].compltime - process\_queue[largest].arrival\_time;

process\_queue[largest].status = 1;

waiting\_time = waiting\_time + process\_queue[largest].waiting\_time;

turnaround\_time = turnaround\_time + process\_queue[largest].turnaround\_time;

printf("\n%c\t\t%d\t\t%d\t\t%d\t\t%d", process\_queue[largest].process\_name,

process\_queue[largest].arrival\_time,

process\_queue[largest].burst\_time,

process\_queue[largest].priority,

process\_queue[largest].waiting\_time);

}

avg\_waiting\_time = waiting\_time / total;

avg\_turnaround\_time = turnaround\_time / total;

printf("\n\nAverage waiting time:\t%f\n", avg\_waiting\_time);

printf("Average Turnaround Time:\t%f\n", avg\_turnaround\_time);

}

**5. If you have implemented any additional algorithm to support the solution, explain the need and usage of the same.**

Yes,I have implemented the use of fcfs algorithm because whenever two processes are there with same priority then to preempt the one we use fcfs.

**6. Explain the boundary conditions of the implemented code.**

1. The boundary conditions for the Priority Scheduling algorithm is whether the processes are assigned to the cpu according to priority.
2. If two processes are having same priority then it works like first come and first serve.
3. To calculate the average turnaround time, average waiting time .
4. In priority scheduling algorithm with context switching,it is very much needed to switch the process as soon as the process with higher priority comes while the current process is running.
5. To give the priority from the smallest to largest integer value.

**7.Explain all the test cases applied on the solution of assigned problem.**

**. Sorting of arrival time**

**.** **avg\_waiting\_time = waiting\_time / total;**

**.avg\_turnaround\_time = turnaround\_time / total;**

**.for (m = 0; m < total - 1; m++)**

**. for (n = m + 1; n < total; n++)**

1. **Have you made minimum 5 revisions of solution on GitHub?**

Yes I have made minimum 5 revisions of solution on Github.

**GitHub Link: https://github.com/ap9407073180**