

**Report on Pre-emptive Scheduler**

**Course:** Operating Systems (CSE316)

**Student Name:** Santhosh M

**Student ID :** 11808892

**Roll No:**  09

**Section:** K18FG

**Email Address:** santhoshsandy918@gmail.com

**GitHub Link:** <https://github.com/Santhosh2396/OS--Preemptive-Scheduling-with-dynamically-changing-priorities>

**Submitted To:** Navneet Kaur

**Problem Description:**

* The Main Problem statement is to create a Scheduler that uses a Pre-emptive Priority Scheduling algorithm based on dynamically changing priority.
* When the process starts execution, priority for that process changes at the rate of m=1.When the process waits for CPU in the ready queue, 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.This means each process gets one second to run before it is exempted by another process.
* 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 User must enter number of processes, their burst time and arrival time.

**Algorithm:**

1. **Create a struct process and declare:** int process\_id,arrival\_time, Service\_time, ct, waiting\_time, turnaround\_time, priority;
2. **Create an array** process\_queue[10];
3. **Create a function which sorts the array “Process”**

void Arrival\_Time\_Sorting()

{ struct process temp;

int i, j;

for(i = 0; i < limit - 1; i++)

{for(j = i + 1; j < limit; j++)

{if(process\_queue[i].arrival\_time > process\_queue[j].arrival\_time)

{temp = process\_queue[i];

process\_queue[i] = process\_queue[j];

process\_queue[j] = temp; }

}}}

1. **Initialize** float wait\_time = 0, turnaround\_time = 0, average\_waiting\_time, average\_turnaround\_time;
2. **Get Inputs :**

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

scanf("%d", &limit);

for(i = 0, c =1; i < limit; i++, c++)

{process\_queue[i].process\_id = c;

printf("\nEnter Details For Process[%d]:\n", process\_queue[i].process\_id);

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

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

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

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

process\_queue[i].status = 0;

Service\_time+=process\_queue[i].Service\_time;}

1. **Processes are executed:**

printf("\nProcess Id\tArrival Time\tService Time\tPriority\tWaiting Time");

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

{largest = 9;

for(i = 0; i < limit; 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].Service\_time; process\_queue[largest].ct = time;

**7. Calculate Waiting Time:** process\_queue[largest].waiting\_time = process\_queue[largest].ct - process\_queue[largest].arrival\_time - process\_queue[largest].Service\_time;

**8. Calculates Turn Around Time:** process\_queue[largest].turnaround\_time = process\_queue[largest].ct - process\_queue[largest].arrival\_time;

process\_queue[largest].status = 1;

wait\_time = wait\_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\_id, process\_queue[largest].arrival\_time, process\_queue[largest].Service\_time, process\_queue[largest].priority, process\_queue[largest].waiting\_time);}

**9.** **Calculates Average Waiting Time :**

average\_waiting\_time = wait\_time / limit;

**10.Calculates Average Turnaround Time:**

average\_turnaround\_time = turnaround\_time / limit**;**

**11.Prints Average Waiting Time:**

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

**Time Complexity:**

1. Constant time complexity = O(1).
2. Constant time complexity = O(1).
3. Complexity = O(n 2).
4. Constant time complexity = O(1).
5. Complexity = O(n).
6. Complexity = O(n).
7. Constant time complexity = O(1).
8. Constant time complexity = O(1).
9. Constant time complexity = O(1).
10. Constant time complexity = O(1).
11. Constant time complexity = O(1).

Total Complexity = O(n2)

**Test Cases:**

