**Experiment 3**

**Title:** Preemptive Scheduling

Simulate working of any 2 preemptive scheduling algorithms: SRTN, Round Robin and Priority based scheduling.

**Estimated time to complete this experiment:** 2 hours

**Objective:** Learning about the different preemptive scheduling algorithm.Implementing programs to demonstrate working of SRTN, RR and Priority Scheduling Algorithms.

**Expected Outcome of Experiment:** To determine average wait time and average turn-around time of various processes and compare algorithms based on it.

**Books/ Journals/ Websites referred:**

1. William Stallings, Operating System: Internals and Design Principles, Prentice Hall, 8thEdition, 2014, ISBN-10: 0133805913 • ISBN-13: 9780133805918.
2. Abraham Silberschatz, Peter Baer Galvin and Greg Gagne, Operating System Concepts, John Wiley &Sons, Inc., 9thEdition, 2016, ISBN 978-81-265-5427-0

**Pre Lab/ Prior Concepts:** Programming platform, Process scheduling algorithms and parameters for evaluating their performance.

**Brief description:**

In [computing](https://en.wikipedia.org/wiki/Computing), scheduling is the method by which work is assigned to resources that complete the work. The work may be virtual computation elements such as [threads](https://en.wikipedia.org/wiki/Thread_(computer_science)) or [processes](https://en.wikipedia.org/wiki/Process_(computing)) which are in turn scheduled onto hardware resources such as [processors](https://en.wikipedia.org/wiki/Central_processing_unit).

A scheduler is what carries out the scheduling activity. Schedulers are often implemented so they keep all computer resources busy (as in [load balancing](https://en.wikipedia.org/wiki/Load_balancing_(computing))), allow multiple users to share system resources effectively, or to achieve a target [quality of service](https://en.wikipedia.org/wiki/Quality_of_service). Scheduling is fundamental to computation itself, and an intrinsic part of the [execution model](https://en.wikipedia.org/wiki/Execution_model) of a computer system; the concept of scheduling makes it possible to have [computer multitasking](https://en.wikipedia.org/wiki/Computer_multitasking) with a single [central processing unit](https://en.wikipedia.org/wiki/Central_processing_unit) (CPU).

**New Concepts to be learned:**  Preemptive scheduling policies and evaluating parameters.

**Requirements:** PC with any programming platform.

**Flow Chart: -**

**Theory:**

Preemptive Scheduling: Preemptive scheduling is used when a process switches from running state to ready state or from waiting state to ready state. The resources (mainly CPU cycles) are allocated to the process for the limited amount of time and then is taken away, and the process is again placed back in the ready queue if that process still has CPU burst time remaining. That process stays in ready queue till it gets next chance to execute.

Shortest Remaining Time Next (SRTN)/Shortest remaining time ﬁrst (SRTF): It is a scheduling method that is a preemptive version of shortest job next scheduling. In this scheduling algorithm, the process with the smallest amount of time remaining until completion is selected to execute.

Round Robin (RR): Round Robin is the preemptive process scheduling algorithm. Each process is provided a fix time to execute, called a quantum. Once a process is executed for a given time period, it is preempted and other process executes for a given time period.

**Program:**

**SRTN:**

#include<iostream>

#include<list>

#include<vector>

#include<queue>

#include<algorithm>

using namespace std;

struct ProcessControlBlock

{

    int pid;

    int BT;

    int AT;

    int CT;

    int waitingtime;

    int TAT;

    int remainingtime;

    int FBT;

};

vector<ProcessControlBlock>PCB;

vector<ProcessControlBlock>Arrived;

queue<ProcessControlBlock>running;

vector<ProcessControlBlock>ans;

int sortByarrival(ProcessControlBlock a , ProcessControlBlock b)

{

    if(a.AT < b.AT)

        return 1;

    else

        return 0;

}

int sortByburst(ProcessControlBlock a , ProcessControlBlock b)

{

    if(a.BT < b.BT)

        return 1;

    else

        return 0;

}

int sortByRemaining(ProcessControlBlock a , ProcessControlBlock b)

{

    if(a.FBT < b.FBT)

        return 1;

    else

        return 0;

}

int main()

{

    int clock=0,pcbindex=0;

    int noOfProcess;

    cout<<"ENTER THE NO. PROCESS =>"<<"\t";

    cin>>noOfProcess;

    cout<<endl;

    int n=noOfProcess;

    for(int i=0;i<n;i++)

    {

        ProcessControlBlock dummy;

        cout<<"Enter the pid          "<<i+1<<"\t";

        cin>>dummy.pid;

        cout<<"Enter the Burst Time   "<<i+1<<"\t";

        cin>>dummy.BT;

        dummy.FBT=dummy.BT;

        cout<<"Enter the Arrival Time "<<i+1<<"\t";

        cin>>dummy.AT;

        cout<<endl;

        PCB.push\_back(dummy);

    }

*//----------------------------------------Logic for SRTN----------------------------------------------------------//*

    sort(PCB.begin(),PCB.end(),sortByarrival);

    int envruntime=30;

    int inpcb=0;

    int time=0;

    int currBT=0;

    int index=0;

    float avgTAT=0;

    float avgWT=0;

    cout<<"For how much time the env must be active";

   cin>>envruntime;

    while(envruntime--)

    {

         while(time==PCB[inpcb].AT)

        {

            Arrived.push\_back(PCB[inpcb]);

            inpcb++;

            sort(Arrived.begin(),Arrived.end(),sortByRemaining);

        }

        if(running.empty())

        {

*// cout<<Arrived.size()<<endl;*

            running.push(Arrived[0]);

        }

        if(running.front().FBT>Arrived[0].FBT)

        {

            running.pop();

            running.push(Arrived[0]);

        }

        if(running.front().FBT==0)

        {

        ans.push\_back(running.front());

            ans[index].waitingtime=time-ans[index].BT-ans[index].AT;

            ans[index].CT=time;

            ans[index].TAT=time-ans[index].AT;

            cout<<time<<endl;

            index=index+1;

            running.pop();

            Arrived.erase(Arrived.begin()+0);

            cout<<Arrived.size()<<endl;

*// time--;*

        }

*// cout<<time<<endl;*

        time++;

        running.front().FBT--;

        Arrived[0].FBT--;

    }

     cout<<"\t\tSRTN"<<endl;

    cout<<"PID"<<"\t"<<"BT"<<"\t"<<"AT"<<"\t"<<"WT"<<"\t"<<"TAT"<<endl;

    for(int i=0;i<n;i++)

    {

         cout<<ans[i].pid<<" \t"<<ans[i].BT<<"  \t"<<ans[i].AT<<"  \t"<<ans[i].waitingtime<<"\t"<<ans[i].TAT<<endl;

         avgTAT=avgTAT+ans[i].TAT;

         avgWT=avgWT+ans[i].waitingtime;

    }

    cout << "Avg Waiting for FCFS:" << (avgWT/ n) << endl;

    cout << "Avg TAT for FCFS:    " << (avgTAT/ n) << endl;

    return 0;

}

**RR:**

#include <iostream>

#include <list>

#include <vector>

#include <queue>

#include <algorithm>

using namespace std;

struct ProcessControlBlock

{

    int pid;

    int BT;

    int AT;

    int CT;

    int waitingtime;

    int TAT;

    int FBT;

};

vector<ProcessControlBlock> PCB;

vector<ProcessControlBlock> Arrived;

queue<ProcessControlBlock> running;

vector<ProcessControlBlock> ans;

int sortByarrival(ProcessControlBlock a, ProcessControlBlock b)

{

    if (a.AT < b.AT)

        return 1;

    else

        return 0;

}

int sortByRemaining(ProcessControlBlock a, ProcessControlBlock b)

{

    if (a.FBT < b.FBT)

        return 1;

    else

        return 0;

}

int main()

{

    int clock = 0, pcbindex = 0;

    int noOfProcess;

    cout << "ENTER THE NO. PROCESS =>"

         << "\t";

    cin >> noOfProcess;

    cout << endl;

    int n = noOfProcess;

    for (int i = 0; i < n; i++)

    {

        ProcessControlBlock dummy;

        cout << "Enter the pid          " << i + 1 << "\t";

        cin >> dummy.pid;

        cout << "Enter the Burst Time   " << i + 1 << "\t";

        cin >> dummy.BT;

        dummy.FBT = dummy.BT;

        cout << "Enter the Arrival Time " << i + 1 << "\t";

        cin >> dummy.AT;

        cout << endl;

        PCB.push\_back(dummy);

        cout << PCB[i].AT << endl;

    }

*//----------------------------------------Logic for Round Robin----------------------------------------------------------//*

    sort(PCB.begin(), PCB.end(), sortByarrival);

    int envruntime = 30;

    int inpcb = 0;

    int time = 0;

    int currBT = 0;

    int index = 0;

    int timequantum = 0;

    float avgTAT = 0;

    float avgWT = 0;

    int temp;

    cout << "Enter the Time quantum: ";

    cin >> timequantum;

    cout << "For how much time the env must be active: ";

    cin >> envruntime;

    while (envruntime)

    {

        while (time == PCB[inpcb].AT && inpcb < n)

        {

            Arrived.push\_back(PCB[inpcb]);

            inpcb++;

        }

        if (running.empty() && !Arrived.empty())

        {

            running.push(Arrived[0]);

            Arrived.erase(Arrived.begin() + 0);

        }

        if (!running.empty())

        {

            running.front().FBT--;

            time++;

            if (running.front().FBT == 0)

            {

                ans.push\_back(running.front());

                ans[index].CT = time;

                ans[index].TAT = time - ans[index].AT;

                ans[index].waitingtime = ans[index].TAT - ans[index].BT;

                index++;

*// cout << time << "c" << endl;*

*// for (int i = 0; i < Arrived.size(); i++)*

*// {*

*//     cout << Arrived[i].pid << endl;*

*// }*

                running.pop();

                if (!Arrived.empty())

                {

                    running.push(Arrived[0]);

                    Arrived.erase(Arrived.begin() + 0);

*// cout<<running.front().FBT<<endl;*

                }

*//  time++;*

            }

            else if (time % timequantum == 0)

            {

                while (time == PCB[inpcb].AT && inpcb < n)

                {

                    Arrived.push\_back(PCB[inpcb]);

                    inpcb++;

                }

                ProcessControlBlock temp = running.front();

                running.pop();

                Arrived.push\_back(temp);

*// cout << time << "p" << endl;*

*// time--;*

            }

        }

        envruntime--;

    }

*// sort(ans.begin(),ans.end(),sortByarrival);*

    cout << "\t\tRound Robin" << endl;

    cout << "PID"

         << "\t"

         << "BT"

         << "\t"

         << "AT"

         << "\t"

         << "WT"

         << "\t"

         << "TAT"

         << "\t"

         << "CT" << endl;

    for (int i = 0; i < n; i++)

    {

        cout << ans[i].pid << " \t" << ans[i].BT << "  \t" << ans[i].AT << "  \t" << ans[i].waitingtime << "\t" << ans[i].TAT << "\t" << ans[i].CT << endl;

        avgTAT = avgTAT + ans[i].TAT;

        avgWT = avgWT + ans[i].waitingtime;

    }

    cout << "Avg Waiting for FCFS:" << (avgWT / n) << endl;

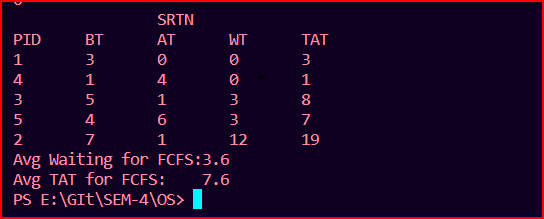
    cout << "Avg TAT for FCFS:    " << (avgTAT / n) << endl;

    return 0;

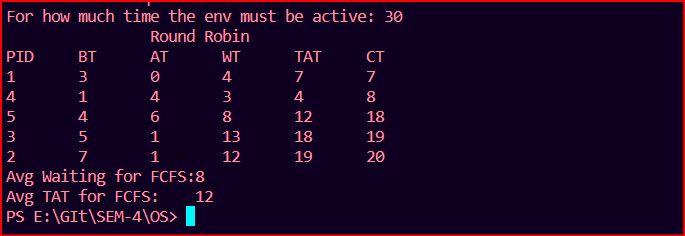
}

**Output:**

SRTN



RR



**Conclusion** Hence we have understood various preemptive scheduling policies such as SRTN, RR and Priority based scheduling. Simulated the scheduling policies and compared them on basis of average waiting time and average turn-around time.

**Real Life Application:**

1. Developing task scheduling policies in multiprocessing over a single CPU.