**Experiment 2**

**Title:** Non-Preemptive Scheduling

Simulate working of any 2 non-preemptive scheduling algorithms: FCFS, SJF and Priority based scheduling.

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

**Objective:** Learning about the different non-preemptive scheduling algorithm.Implementing programs to demonstrate working of FCFS and SJF 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:**  Non-preemptive scheduling policies and evaluating parameters.

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

**Flow Chart: -**

**Theory:** Non Preemptive Scheduling:

When any process is in execution,and any other high priority process enters the system, the control is not passed to the new process irrespective of the priority. This type of scheduling algorithm is known as Non-Preemptive Scheduling Algorithm.

First Come First Serve (FCFS): In this algorithm, the processes are executed in the order that they arrive. There is no change in the order of the processes. Control is passed to the next process only after the execution of the previous process is completed.

Shortest Job First(SJF): is a scheduling policy that selects the waiting process with the smallest execution time to execute next. SJF can be used in specialized environments where accurate estimates of running time are available.

**Program:**

**1)FCFS🡺**

#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;

};

vector<ProcessControlBlock> PCB;

vector<ProcessControlBlock> Active;

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 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;

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

        cin >> dummy.AT;

        cout << endl;

        PCB.push\_back(dummy);

    }

*//--------------------------------------------Logic for FCFS----------------------------------------//*

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

    int envActiveFor1;

    int time1 = 0;

    int currentpcb1 = 0;

    int qfront1 = 0;

    float wating1 = 0;

    float TurnAround = 0;

    cout << "For FCFS for how much time the environment must be active "

         << "\t";

    cin >> envActiveFor1;

    while (envActiveFor1--)

    {

        while (time1 >= PCB[currentpcb1].AT)

        {

            Active.push\_back(PCB[currentpcb1]);

            currentpcb1 = currentpcb1 + 1;

        }

        if (Active[0].BT == 0)

        {

            PCB[qfront1].waitingtime = time1 - PCB[qfront1].BT - PCB[qfront1].AT;

            PCB[qfront1].CT = time1;

            PCB[qfront1].TAT = time1 - PCB[qfront1].AT;

            qfront1 = qfront1 + 1;

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

        }

        time1++;

        Active[0].BT--;

    }

    cout << "\t\tFCFS" << endl;

    cout << "PID"

         << "\t"

         << "BT"

         << "\t"

         << "AT"

         << "\t"

         << "CT"

         << "\t"

         << "TAT"

         << "\t"

         << "WT" << endl;

    for (int i = 0; i < n; i++)

    {

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

    }

    for (int i = 0; i < n; i++)

    {

        wating1 = wating1 + PCB[i].waitingtime;

        TurnAround = TurnAround + PCB[i].TAT;

    }

    cout << "Avg Waiting for FCFS:" << (float)(wating1 / n) << endl;

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

    return 0;

}

**SJF🡺**

#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 SJF----------------------------------------------------------//*

    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(),sortByburst);

        }

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

        {

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

            running.push(Arrived[0]);

           cout<< Arrived[0].pid<<endl;;

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

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

            time--;

        }

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

        time++;

        running.front().FBT--;

    }

     cout<<"\t\tSJF"<<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;

    }

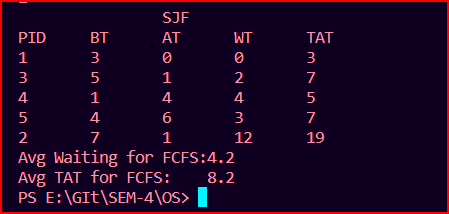
    cout << "Avg Waiting for FCFS:" << (avgWT/ n) << endl;

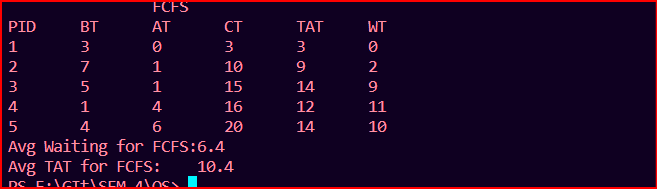
    cout << "Avg TAT for FCFS:    " << (avgTAT/ n) << endl;

    return 0;

}

**Output:**





**Conclusion** Hence we have understood various non-preemptive scheduling policies such as FCFS, SJF 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.