**OPERATING SYSTEM LAB FILE**

**University School of Information Communication and Technology**

**  
Guru Gobind Singh Indraprastha University  
Dwarka, Sector 16C, New Delhi**

**Submitted By : Kshitij   
Enrollment No. : 04616401523  
Course : B.Tech IT  
Semester/Batch : First/B  
Year : 2023**

**INDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S NO.** | **NAME OF EXPERIMENT** | **DATE** | **PAGE NO.** | **REMARKS** |
| **1.** | Simulate the following CPU scheduling algorithms:  a) FCFS  b) SJF (non-preemptive)  c) Round Robin  d) Priority |  |  |  |
| **2.** | Create a semaphore user-defined data type using wait () and signal () operations. |  |  |  |
| **3.** | Simulate Resource Allocation Graph method for deadlock avoidance. |  |  |  |
| **4.** | Simulate Banker's algorithm for deadlock avoidance. |  |  |  |
| **5.** | Simulate deadlock detection algorithm for multiple resource instances. |  |  |  |
| **6.** | Simulate contiguous memory allocation using fixed size partitions. |  |  |  |
| **7.** | Simulate contiguous memory allocation using variable size partitions. |  |  |  |
| **8.** | Simulate: a) First Fit  b) Best Fit &  c) Worst Fit algorithms of contiguous allocation |  |  |  |
| **9.** | Simulate following page replacement algorithms:  a) FIFO  b) LRU  c) OPTIMAL |  |  |  |
| **10.** | Simulate: a) FCFS  b) SSTF &  c) SCAN disk scheduling algorithms. |  |  |  |

**EXPERIMENT NO. 1**

**AIM:** Simulate the following CPU scheduling algorithms:

a) FCFS b) SJF (non-preemptive) c) Round Robin d) Priority

**Code: a) FCFS**

#include <iostream>

using namespace std;

struct Process

{

    int serial;

    int arrival;

    int burst;

    int waiting;

};

void sort\_arrival(Process processes\_1[], int n)

{

    for (int i=0 ; i<n-1 ; i++) {

        for (int j=0 ; j<n-i-1 ; j++) {

            if (processes\_1[j].arrival > processes\_1[j+1].arrival)

            {

                Process temp = processes\_1[j];

                processes\_1[j] = processes\_1[j+1];

                processes\_1[j+1] = temp;

            }

        }

    }

}

int main()

{

    int size, sumBurst = 0;

    cout << "Enter no. of Processes: ";

    cin >> size;

    Process processes[size];

    for (int i=0 ; i<size ; i++)

    {

        cout << "Enter Arrival time & Burst time for P" << i+1 << ": ";

        cin >> processes[i].arrival >> processes[i].burst;

        processes[i].serial = i+1;

    }

    sort\_arrival(processes, size);

    for (int i=0 ; i<size ; i++)

    {

        if (i>0)

        {

            sumBurst += processes[i-1].burst;

        }

        int extra = 0;

        if (processes[i].arrival > sumBurst)

        {

            extra = (processes[i].arrival - sumBurst);

            sumBurst += extra;

        }

        processes[i].waiting = sumBurst - processes[i].arrival;

    }

    cout << "\n After Sorting:-" << endl;

    cout << "\nProcess" << "   " << "Arrival" << "  " << "Burst" << "  " << "Waiting" << endl;

    for (int i=0 ; i<size ; i++)

    {

        cout << "   P" << processes[i].serial << "        " << processes[i].arrival \

        << "       " << processes[i].burst << "      " << processes[i].waiting;

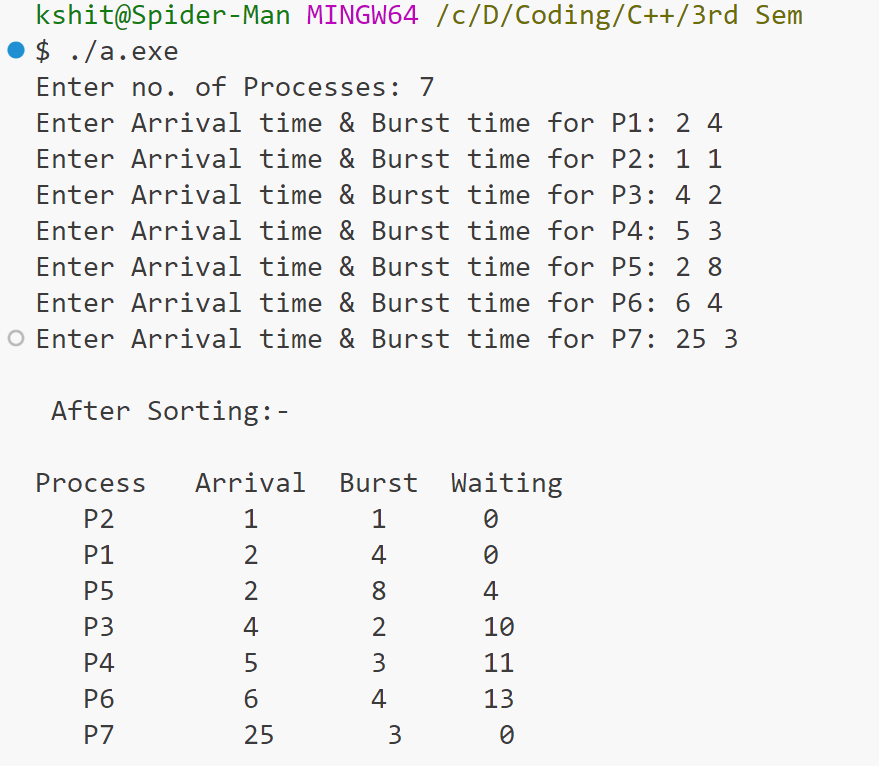
        cout << endl;

    }

    return 0;

}

**Output:**



**Code: b) SJF (non-preemptive)**

# include <iostream>

using namespace std;

struct Process{

    int Serial,Arrival,CpuBurst,WaitingTime;

    void Print\_info(){

        cout<<"P"<<Serial<<". CPU Burst Time: "<<CpuBurst<<"\t";

        cout<<"    Arrival Time: "<<Arrival<<endl;

    }

};

    void swap(Process&p1, Process&p2) {

        Process temp = p1;

        p1 = p2;

        p2 = temp;

    }

    void Sorting(Process processes[], int n) {

    for(int i=0;i<n-1;i++){

        for(int j=0;j<n-1;j++){

            if(processes[j].Arrival > processes[j+1].Arrival){

                swap(processes[i], processes[j]);

                }

            }

        for(int j=1;j<n;j++){

            if(processes[j].CpuBurst > processes[j+1].CpuBurst){

                swap(processes[j], processes[j+1]);

                }

            }

        }

    }

int main(){

    int n;

    cout<<"Enter the number of Process: ";

    cin>>n;

    Process processes[n];

    for(int i=0;i<n;i++){

        processes[i].Serial = i+1;

        cout<<"Enter the CPU Burst for P"<<processes[i].Serial<<": ";

        cin>>processes[i].CpuBurst;

        cout<<"Enter the Arrival Time for P"<<processes[i].Serial<<": ";

        cin>>processes[i].Arrival;

    }

    cout<<endl<<"Total Number of Processes: "<<n<<endl<<endl;

    cout<<"Before Sorting:"<<endl;

    for(int i=0;i<n;i++){

        processes[i].Print\_info();

    }

    Sorting(processes,n);

    cout<<endl<<"After Sorting:"<<endl<<"1. Waiting Time of Process P"<<processes[0].Serial<<": 0"<<endl;

    int Prev\_Finish\_Time = processes[0].Arrival;

    for(int i=1;i<n;i++){

        Prev\_Finish\_Time += processes[i-1].CpuBurst;

        if(Prev\_Finish\_Time <= processes[i].Arrival){

            for(int j=n-1;j>=i;j--){

                if(processes[j].Arrival < processes[j-1].Arrival){

                    swap(processes[j],processes[j-1]);

                }

            }

            Prev\_Finish\_Time = processes[i].Arrival;

            cout<<i+1<<". Waiting Time of Process P"<<processes[i].Serial<<": 0"<<endl;

        }

        else{

            processes[i].WaitingTime = max(0, Prev\_Finish\_Time - processes[i].Arrival);

            cout<<i+1<<". Waiting Time of Process P"<<processes[i].Serial<<": "<<processes[i].WaitingTime<<endl;

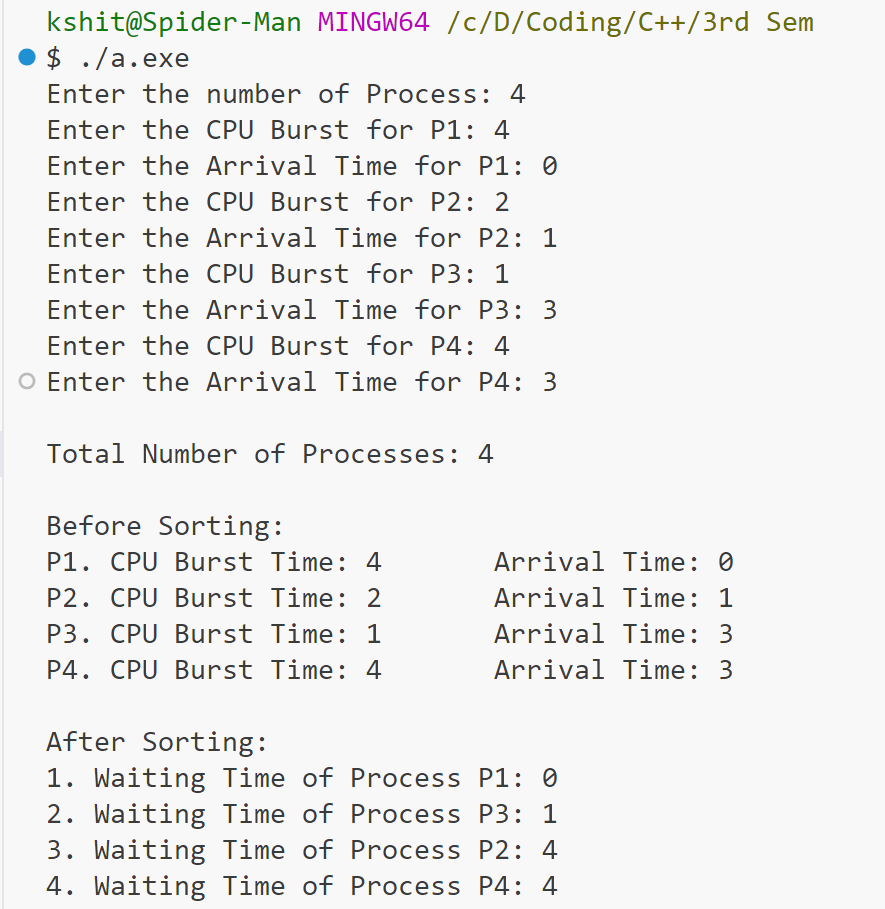
        }

    }

    return 0;

}

**Output:**



**Code: c) Round Robin**

# include <iostream>

using namespace std;

struct Process{

    int Serial,Arrival,CpuBurst,WaitingTime,Priority;

    void Print\_info(){

        cout<<"P"<<Serial<<". CPU Burst Time: "<<CpuBurst<<"\t";

        cout<<"    Arrival Time: "<<Arrival<<"\t";

        cout<<"    Priority: "<<Priority<<endl;

    }

};

    void Sorting(Process processes[], int n) {

    for(int i=0;i<n-1;i++){

        for(int j=0;j<n-1;j++){

            if(processes[j].Priority > processes[j+1].Priority){

                Process temp = processes[j];

                processes[j] = processes[j+1];

                processes[j+1] = temp;

            }

        }

    }

}

int main(){

    int n;

    cout<<"Enter the number of Process: ";

    cin>>n;

    Process processes[n];

    for(int i=0;i<n;i++){

        processes[i].Serial = i+1;

        cout<<"Enter the CPU Burst for P"<<processes[i].Serial<<": ";

        cin>>processes[i].CpuBurst;

        cout<<"Enter the Arrival Time for P"<<processes[i].Serial<<": ";

        cin>>processes[i].Arrival;

        cout<<"Enter the Priority for P"<<processes[i].Serial<<": ";

        cin>>processes[i].Priority;

    }

    cout<<endl<<"Total Number of Processes: "<<n<<endl<<endl;

    cout<<"Before Sorting:"<<endl;

    for(int i=0;i<n;i++){

        processes[i].Print\_info();

    }

    Sorting(processes,n);

    cout<<endl<<"After Sorting:"<<endl;

    for(int i=0;i<n;i++){

        processes[i].Print\_info();

    }

    cout<<endl<<"1. Waiting Time of Process P"<<processes[0].Serial<<": 0"<<endl;

    int Prev\_Finish\_Time = processes[0].Arrival;

    for(int i=1;i<n;i++){

        Prev\_Finish\_Time += processes[i-1].CpuBurst;

        processes[i].WaitingTime = max(0, Prev\_Finish\_Time - processes[i].Arrival);

        if(Prev\_Finish\_Time < processes[i].Arrival){

            Prev\_Finish\_Time = processes[i].Arrival;

            cout<<i+1<<". Waiting Time of Process P"<<processes[i].Serial<<": 0"<<endl;

        }

        else{

        cout<<i+1<<". Waiting Time of Process P"<<processes[i].Serial<<": "<<processes[i].WaitingTime<<endl;

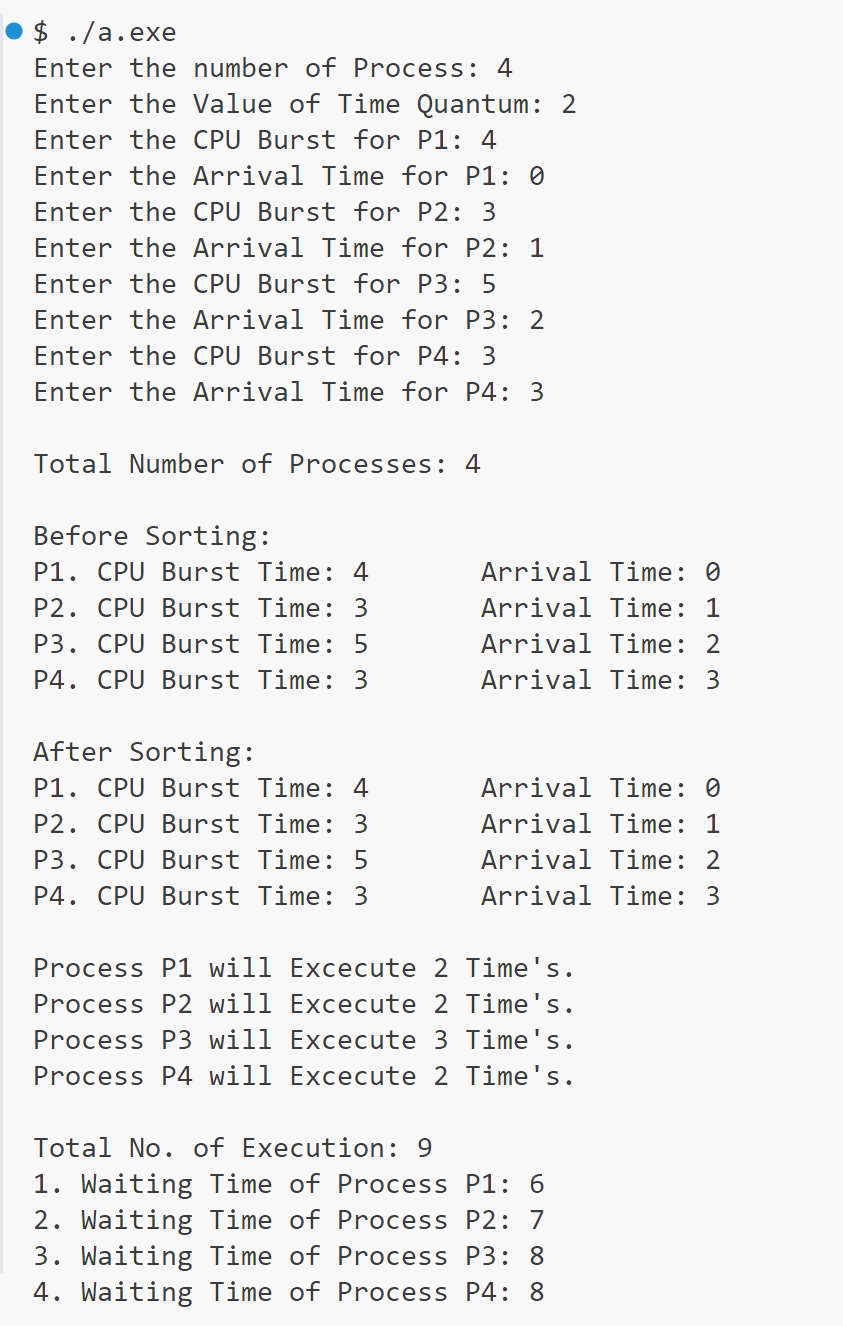
        }

    }

    return 0;

}

**Output:**



**Code: d) Priority**

# include <iostream>

using namespace std;

struct Process{

    int Serial,Arrival,CpuBurst,WaitingTime,Priority;

    void Print\_info(){

        cout<<"P"<<Serial<<". CPU Burst Time: "<<CpuBurst<<"\t";

        cout<<"    Arrival Time: "<<Arrival<<"\t";

        cout<<"    Priority: "<<Priority<<endl;

    }

};

    void Sorting(Process processes[], int n) {

    for(int i=0;i<n-1;i++){

        for(int j=0;j<n-1;j++){

            if(processes[j].Priority > processes[j+1].Priority){

                Process temp = processes[j];

                processes[j] = processes[j+1];

                processes[j+1] = temp;

            }

        }

    }

}

int main(){

    int n;

    cout<<"Enter the number of Process: ";

    cin>>n;

    Process processes[n];

    for(int i=0;i<n;i++){

        processes[i].Serial = i+1;

        cout<<"Enter the CPU Burst for P"<<processes[i].Serial<<": ";

        cin>>processes[i].CpuBurst;

        cout<<"Enter the Arrival Time for P"<<processes[i].Serial<<": ";

        cin>>processes[i].Arrival;

        cout<<"Enter the Priority for P"<<processes[i].Serial<<": ";

        cin>>processes[i].Priority;

    }

    cout<<endl<<"Total Number of Processes: "<<n<<endl<<endl;

    cout<<"Before Sorting:"<<endl;

    for(int i=0;i<n;i++){

        processes[i].Print\_info();

    }

    Sorting(processes,n);

    cout<<endl<<"After Sorting:"<<endl;

    for(int i=0;i<n;i++){

        processes[i].Print\_info();

    }

    cout<<endl<<"1. Waiting Time of Process P"<<processes[0].Serial<<": 0"<<endl;

    int Prev\_Finish\_Time = processes[0].Arrival;

    for(int i=1;i<n;i++){

        Prev\_Finish\_Time += processes[i-1].CpuBurst;

        processes[i].WaitingTime = max(0, Prev\_Finish\_Time - processes[i].Arrival);

        if(Prev\_Finish\_Time < processes[i].Arrival){

            Prev\_Finish\_Time = processes[i].Arrival;

            cout<<i+1<<". Waiting Time of Process P"<<processes[i].Serial<<": 0"<<endl;

        }

        else{

        cout<<i+1<<". Waiting Time of Process P"<<processes[i].Serial<<": "<<processes[i].WaitingTime<<endl;

        }

    }

    return 0;

}

**Output:**

