FCFS ------------------->

#include<iostream>

#define max 20

using namespace std;

int main()

{

    int j,n,Burst\_time[max],Waiting\_time[max],Turn\_around\_time[max],Arrival\_time[max],Completion\_time[max];

    int Process\_ID[max];

    float Avg\_turn\_around\_time=0,Avg\_waiting\_time=0;

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

    cin>>n;

    for (int i = 0; i<n; i++)

    {

        Process\_ID[i] = i + 1;

        cout<<"Enter the Burst time for process "<<i+1<<" :";

        cin>>Burst\_time[i];

        cout<<"Enter the Arrival time for process "<<i+1<<" :";

        cin>>Arrival\_time[i];

    }

    for (int i = 0; i < n; i++) {

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

            if (Arrival\_time[i] > Arrival\_time[j]) {

                swap(Arrival\_time[i], Arrival\_time[j]);

                swap(Burst\_time[i], Burst\_time[j]);

                swap(Process\_ID[i], Process\_ID[j]);

            }

        }

    }

    Completion\_time[0]=Arrival\_time[0];

    for(int i=0;i<n;i++)

    {

        Turn\_around\_time[i]=0;

        Waiting\_time[i]=0;

        Completion\_time[i+1]=Completion\_time[i]+Burst\_time[i];

        Turn\_around\_time[i]=Completion\_time[i+1]-Arrival\_time[i];

        Waiting\_time[i]=Turn\_around\_time[i]-Burst\_time[i];

        Avg\_waiting\_time=Avg\_waiting\_time+Waiting\_time[i];

        Avg\_turn\_around\_time=Avg\_turn\_around\_time+Turn\_around\_time[i];

    }

    cout<<"GANTT-CHART"<<endl;

    cout<<endl;

    cout<<"|";

    for(int i=0;i<n;i++)

    {

        cout<<"   "<<Process\_ID[i]<<"   |";

    }

    cout<<endl;

    for(int i=0;i<=n;i++)

    {

        cout<<Completion\_time[i]<<"\t";

    }

    cout<<endl;

    cout<<endl<<"TURN-AROUND TIME "<<endl;

    for (int i = 0; i<n; i++)

    {

        cout<<"Turn-around time of process "<<i+1<<"= "<<Turn\_around\_time[i]<<" msec"<<endl;

    }

    Avg\_turn\_around\_time=Avg\_turn\_around\_time/n;

    cout<<"Average Turn-around time of = "<<Avg\_turn\_around\_time<<" msec"<<endl;

    cout<<endl;

    for (int i = 0; i<n; i++)

    {

        cout<<"Waiting time of "<<i+1<<"= "<<Waiting\_time[i]<<" msec"<<endl;

    }

    Avg\_waiting\_time=Avg\_waiting\_time/n;

    cout<<"Average Waiting time of = "<<Avg\_waiting\_time<<" msec"<<endl;

    return 0;

}

Output --------------------->

Enter the number of process: 4

Enter the Burst time for process 1 :5

Enter the Arrival time for process 1 :0

Enter the Burst time for process 2 :3

Enter the Arrival time for process 2 :1

Enter the Burst time for process 3 :8

Enter the Arrival time for process 3 :2

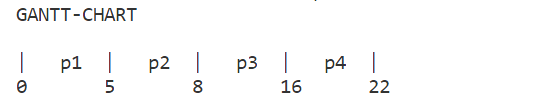
Enter the Burst time for process 4 :6

Enter the Arrival time for process 4 :3

GANTT-CHART

| p1 | p2 | p3 | p4 |

0 5 8 16 22



TURN-AROUND TIME

Turn-around time of process 1= 5 msec

Turn-around time of process 2= 7 msec

Turn-around time of process 3= 14 msec

Turn-around time of process 4= 19 msec

Average Turn-around time of = 11.25 msec

Waiting time of 1= 0 msec

Waiting time of 2= 4 msec

Waiting time of 3= 6 msec

Waiting time of 4= 13 msec

Average Waiting time of = 5.75 msec

Priority ------------------>

#include<iostream>

#define max 20

using namespace std;

int main()

{

    struct process

    {

        int id,Burst\_time,Waiting\_time,Turn\_around\_time,Arrival\_time,ST,CT,priority;

    }p[20];

    int n,i=0,current\_time=0,completion\_time=0,completed=0,is\_comp[max],min\_index=-1;

    float Avg\_turn\_around\_time=0,Avg\_waiting\_time=0;

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

    cin>>n;

    for (int i = 0; i<n; i++)

    {

        p[i].id=i+1;

        cout<<"Enter the Burst time for process "<<i+1<<" :";

        cin>>p[i].Burst\_time;

        cout<<"Enter the Arrival time for process "<<i+1<<" :";

        cin>>p[i].Arrival\_time;

        cout<<"Enter the Priority of process "<<i+1<<" :";

        cin>>p[i].priority;

        is\_comp[i]=0;

    }

    cout<<endl<<"GANTT-CHART"<<endl;

    while(completed!=n)

    {

        int min=-1;

        for (int i = 0; i <n; i++)

        {

            if (p[i].Arrival\_time<=current\_time && is\_comp[i]==0)

            {

                if (p[i].priority>min)

                {

                    min=p[i].priority;

                    min\_index=i;

                }

                if (p[i].priority==min)

                {

                    if (p[i].Arrival\_time<p[min\_index].Arrival\_time)

                    {

                        min=p[i].priority;

                        min\_index=i;

                    }

                }

            }

        }

        if(min\_index==-1)

        {

            current\_time++;

        }

        else

        {

            p[min\_index].ST=current\_time;

            p[i].CT=p[min\_index].ST+p[min\_index].Burst\_time;

            p[min\_index].Turn\_around\_time=p[i].CT-p[min\_index].Arrival\_time;

            p[min\_index].Waiting\_time=p[min\_index].Turn\_around\_time-p[min\_index].Burst\_time;

            is\_comp[min\_index]=1;

            Avg\_turn\_around\_time+=p[min\_index].Turn\_around\_time;

            Avg\_waiting\_time+=p[min\_index].Waiting\_time;

            current\_time=p[i].CT;

            i++;

            completed++;

            cout<<"|   P"<<p[min\_index].id<<"\t";

        }

    }

    cout<<"|";

    cout<<endl<<"0\t";

    for (int i = 0; i <n; i++)

    {

        cout<<p[i].CT<<"\t";

    }

    cout<<endl<<"\nTURN-AROUND TIME "<<endl;

    for (int i = 0; i<n; i++)

    {

        cout<<"Turn-around time of process "<<i+1<<"= "<<p[i].Turn\_around\_time<<" msec"<<endl;

    }

    Avg\_turn\_around\_time=Avg\_turn\_around\_time/n;

    cout<<"Average Turn-around time of = "<<Avg\_turn\_around\_time<<" msec"<<endl;

    cout<<endl;

    cout<<endl<<"WAITING TIME "<<endl;

    for (int i = 0; i<n; i++)

    {

        cout<<"Waiting time of "<<i+1<<"= "<<p[i].Waiting\_time<<" msec"<<endl;

    }

    Avg\_waiting\_time=Avg\_waiting\_time/n;

    cout<<"Average Waiting time of = "<<Avg\_waiting\_time<<" msec"<<endl;

    return 0;

}

Output -------------------------->

Enter the number of process: 4

Enter the Burst time for process 1 :5

Enter the Arrival time for process 1 :0

Enter the Priority of process 1 :1

Enter the Burst time for process 2 :3

Enter the Arrival time for process 2 :1

Enter the Priority of process 2 :2

Enter the Burst time for process 3 :8

Enter the Arrival time for process 3 :2

Enter the Priority of process 3 :1

Enter the Burst time for process 4 :6

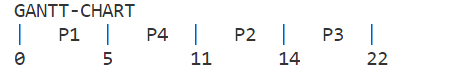
Enter the Arrival time for process 4 :3

Enter the Priority of process 4 :3

GANTT-CHART

| P1 | P4 | P2 | P3 |

0 5 11 14 22



TURN-AROUND TIME

Turn-around time of process 1= 5 msec

Turn-around time of process 2= 13 msec

Turn-around time of process 3= 20 msec

Turn-around time of process 4= 8 msec

Average Turn-around time of = 11.5 msec

WAITING TIME

Waiting time of 1= 0 msec

Waiting time of 2= 10 msec

Waiting time of 3= 12 msec

Waiting time of 4= 2 msec

Average Waiting time of = 6 msec

SJF ---------------->

#include <iostream>

#include <string>

#include <algorithm>

using namespace std;

void SJF\_preemptive(int n, int burst\_time[], string p[], int arrival\_time[]) {

    int finish\_time[10] = {0};

    int turnaround\_time[10] = {0};

    int waiting\_time[10] = {0};

    int remaining\_time[10];

    copy(burst\_time, burst\_time + n, remaining\_time);

    float avg\_turnaround\_time = 0, avg\_waiting\_time = 0;

    int current\_time = 0;

    int completed = 0;

    int exe\_ord[10],curr\_time[10],exe\_ind=0;

    while (completed < n) {

        int shortest\_job = -1;

        int shortest\_time = INT\_MAX;

        for (int i = 0; i < n; i++) {

            if (arrival\_time[i] <= current\_time && remaining\_time[i] < shortest\_time && remaining\_time[i] > 0) {

                shortest\_job = i;

                shortest\_time = remaining\_time[i];

            }

        }

        if(shortest\_job != exe\_ord[exe\_ind-1]){

            exe\_ord[exe\_ind]=shortest\_job;

            curr\_time[exe\_ind] = current\_time;

            exe\_ind++;

        }

        if (shortest\_job == -1) {

            current\_time++;

        } else {

            remaining\_time[shortest\_job]--;

            current\_time++;

            if (remaining\_time[shortest\_job] == 0) {

                completed++;

                finish\_time[shortest\_job] = current\_time;

                turnaround\_time[shortest\_job] = finish\_time[shortest\_job] - arrival\_time[shortest\_job];

                waiting\_time[shortest\_job] = turnaround\_time[shortest\_job] - burst\_time[shortest\_job];

            }

        }

    }

    cout<<"GANTT-CHART"<<endl;

    for (int i = 0; i <exe\_ind; i++)

    {

        cout<<"|   P"<<exe\_ord[i]<<"\t";

    }

    cout<<"|"<<endl;

    for (int i = 0; i < exe\_ind; i++)

    {

        cout<<curr\_time[i]<<"\t";

    }

    cout<<current\_time<<endl;

    for (int i = 0; i < n; i++) {

        avg\_turnaround\_time += turnaround\_time[i];

        avg\_waiting\_time += waiting\_time[i];

    }

    avg\_turnaround\_time /= n;

    avg\_waiting\_time /= n;

    cout << "\nGANTT CHART" << endl;

    for (int i = 0; i < n; i++) {

        cout << "Turnaround time of " << p[i] << " = " << turnaround\_time[i] << " msec" << endl;

    }

    cout << "\nAVERAGE TURNAROUND TIME = " << avg\_turnaround\_time << " msec" << endl;

    cout << "\nWAITING TIME" << endl;

    for (int i = 0; i < n; i++) {

        cout << "Waiting time of P" <<p[i] << " = " << waiting\_time[i] << " msec" << endl;

    }

    cout << "\nAVERAGE WAITING TIME = " << avg\_waiting\_time << " msec" << endl;

}

int main() {

    int n;

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

    cin >> n;

    int burst\_time[10];

    int arrival\_time[10];

    string p[10];

    for (int i = 0; i < n; i++) {

        p[i]=i+1;

        cout << "Enter the Burst time: ";

        cin >> burst\_time[i];

        cout << "Enter the Arrival time: ";

        cin >> arrival\_time[i];

    }

    SJF\_preemptive(n, burst\_time, p, arrival\_time);

    return 0;

}

Output ---------------------->

Enter the number of processes: 5

Enter the Burst time: 3

Enter the Arrival time: 0

Enter the Burst time: 5

Enter the Arrival time: 1

Enter the Burst time: 2

Enter the Arrival time: 3

Enter the Burst time: 5

Enter the Arrival time: 9

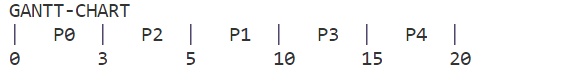
Enter the Burst time: 5

Enter the Arrival time: 12

GANTT-CHART

| P0 | P2 | P1 | P3 | P4 |

0 3 5 10 15 20



GANTT CHART

Turnaround time of P1 = 3 msec

Turnaround time of P2 = 9 msec

Turnaround time of P3 = 2 msec

Turnaround time of P4 = 6 msec

Turnaround time of P5 = 8 msec

AVERAGE TURNAROUND TIME = 5.6 msec

WAITING TIME

Waiting time of P1 = 0 msec

Waiting time of P2 = 4 msec

Waiting time of P3 = 0 msec

Waiting time of P4 = 1 msec

Waiting time of P5 = 3 msec

Round Robin ------------------->

#include<iostream>

#define max 20

using namespace std;

int main()

{

    struct process

    {

        int id,Burst\_time,Waiting\_time,Turn\_around\_time,Arrival\_time,Completion\_time=0;

    }p[20];

    int n,time\_quantum,k=0,min\_ind,completed=0,exe\_ind=0,cur\_time=0,exe\_ord[max],remaining\_time[max],current\_time[max];

    float Avg\_turn\_around\_time=0,Avg\_waiting\_time=0;

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

    cin>>n;

    cout<<"Enter the time quantum: ";

    cin>>time\_quantum;

    for (int i = 0; i<n; i++)

    {

        p[i].id=i;

        cout<<"Enter the Burst time for process "<<i<<" :";

        cin>>p[i].Burst\_time;

        remaining\_time[i]=p[i].Burst\_time;

        cout<<"Enter the Arrival time for process "<<i<<" :";

        cin>>p[i].Arrival\_time;

    }

    while(completed<n)

    {

        current\_time[0]=0;

        for(int i=0;i<n;i++)

        {

            if (remaining\_time[i]>0 && p[i].Arrival\_time<=cur\_time)

            {

                int time=min(time\_quantum,remaining\_time[i]);

                remaining\_time[i]-=time;

                cur\_time+=time;

                exe\_ord[exe\_ind]=i;

                current\_time[exe\_ind+1]=cur\_time;

                exe\_ind++;

                if (remaining\_time[i]==0)

                {

                    completed++;

                    p[i].Completion\_time=cur\_time;

                    p[i].Turn\_around\_time=p[i].Completion\_time-p[i].Arrival\_time;

                    p[i].Waiting\_time=p[i].Turn\_around\_time-p[i].Burst\_time;

                    Avg\_turn\_around\_time+=p[i].Turn\_around\_time;

                    Avg\_waiting\_time+=p[i].Waiting\_time;

                }

            }

        }

    }

    cout<<"GANTT-CHART"<<endl;

    for (int i = 0; i <exe\_ind; i++)

    {

        cout<<"|   P"<<p[exe\_ord[i]].id<<"\t";

    }

    cout<<"|"<<endl;

    for (int i = 0; i <= exe\_ind; i++)

    {

        cout<<current\_time[i]<<"\t";

    }

    cout<<endl;

    cout<<endl<<"\nTURN-AROUND TIME "<<endl;

    for (int i = 0; i<n; i++)

    {

        cout<<"Turn-around time of process "<<i+1<<"= "<<p[i].Turn\_around\_time<<" msec"<<endl;

    }

    Avg\_turn\_around\_time=Avg\_turn\_around\_time/n;

    cout<<"Average Turn-around time = "<<Avg\_turn\_around\_time<<" msec"<<endl;

    cout<<endl;

    cout<<endl<<"WAITING TIME "<<endl;

    for (int i = 0; i<n; i++)

    {

        cout<<"Waiting time of "<<i+1<<"= "<<p[i].Waiting\_time<<" msec"<<endl;

    }

    Avg\_waiting\_time=Avg\_waiting\_time/n;

    cout<<"Average Waiting time = "<<Avg\_waiting\_time<<" msec"<<endl;

    return 0;

}

output ----------------------------->

Enter the number of process: 6

Enter the time quantum: 5

Enter the Burst time for process 0 :7

Enter the Arrival time for process 0 :0

Enter the Burst time for process 1 :4

Enter the Arrival time for process 1 :1

Enter the Burst time for process 2 :15

Enter the Arrival time for process 2 :2

Enter the Burst time for process 3 :11

Enter the Arrival time for process 3 :3

Enter the Burst time for process 4 :20

Enter the Arrival time for process 4 :4

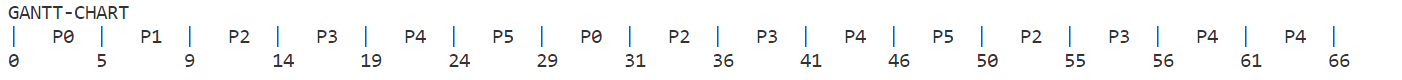
Enter the Burst time for process 5 :9

Enter the Arrival time for process 5 :4

GANTT-CHART

| P0 | P1 | P2 | P3 | P4 | P5 | P0 | P2 | P3 | P4 | P5 | P2 | P3 | P4 | P4 |

0 5 9 14 19 24 29 31 36 41 46 50 55 56 61 66



TURN-AROUND TIME

Turn-around time of process 1= 31 msec

Turn-around time of process 2= 8 msec

Turn-around time of process 3= 53 msec

Turn-around time of process 4= 53 msec

Turn-around time of process 5= 62 msec

Turn-around time of process 6= 46 msec

Average Turn-around time = 42.1667 msec

WAITING TIME

Waiting time of 1= 24 msec

Waiting time of 2= 4 msec

Waiting time of 3= 38 msec

Waiting time of 4= 42 msec

Waiting time of 5= 42 msec

Waiting time of 6= 37 msec

Average Waiting time = 31.1667 msec

B1.java

import java.io.\*;

import java.util.\*;

class B1 {

    static {

         System.loadLibrary("b1");

    }

    private native int add(int a, int b);

    public static void main(String[] args) {

    Scanner sc=new Scanner(System.in);

    int a, b,ch;

    System.out.println("\nEnter value of a : ");

    a = sc.nextInt();

    System.out.println("\nEnter value of b : ");

    b = sc.nextInt();

    do

    {

        System.out.println("\nENTER YOUR CHOICE : ");

        ch = sc.nextInt();

        switch(ch)

        {

            case 1 : new B1().add(a,b);

                 break;

            default : System.out.println("Your choice is wrong.");

        }

    }while(ch<2);

     }

 }

B1.c

#include <jni.h>

#include <stdio.h>

#include "B1.h"

JNIEXPORT int JNICALL Java\_B1\_add(JNIEnv \*env, jobject obj, jint a, jint b)

{

    printf("\n%d + %d = %d\n",a,b,(a+b));

    return 0;

}

Output ---------------------->

Steps to execute the program -------------------->

javac -h . B1.java

(base) admin1@408-21:~$ javac B1.java

(base) admin1@408-21:~$ javah B1

(base) admin1@408-21:~$ gcc -fPIC -I"$JAVA\_HOME/include" -I"$JAVA\_HOME/include/linux" -shared -o libb1.so B1.c

(base) admin1@408-21:~$ java -Djava.library.path=. B1

Enter value of a :

2

Enter value of b :

3

ENTER YOUR CHOICE :

1

2 + 3 = 5

ENTER YOUR CHOICE :

First Fit ------------->

#include <iostream>

using namespace std;

void firstFit(int blockSize[], int m,

            int processSize[], int n)

{

    int allocation[n];

    for (int i = 0; i < n; i++)

        allocation[i] = -1;

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < m; j++)

        {

            if (blockSize[j] >= processSize[i])

            {

                allocation[i] = j;

                blockSize[j] -= processSize[i];

                break;

            }

        }

    }

    cout << "\nProcess No.\tProcess Size\tBlock no.\n";

    for (int i = 0; i < n; i++)

    {

        cout << " P" << i+1 << "\t\t"

            << processSize[i] << "\t\t";

        if (allocation[i] != -1)

            cout << allocation[i] + 1;

        else

            cout << "Not Allocated";

        cout << endl;

    }

}

int main()

{

       int m;

       cout << "Enter the size of the blockSize array: ";

       cin >> m;

       int blockSize[m];

       for (int i = 0; i < m; i++) {

              cout << "Enter the value for blockSize[" << i << "]: ";

              cin >> blockSize[i];

       }

       int n;

       cout << "Enter the size of the processSize array: ";

       cin >> n;

       int processSize[n];

       for (int i = 0; i < n; i++) {

              cout << "Enter the value for processSize[" << i+1 << "]: ";

              cin >> processSize[i];

       }

        firstFit(blockSize, m, processSize, n);

        return 0 ;

}

output ---------------------->

Enter the size of the blockSize array: 5

Enter the value for blockSize[0]: 100

Enter the value for blockSize[1]: 500

Enter the value for blockSize[2]: 200

Enter the value for blockSize[3]: 300

Enter the value for blockSize[4]: 600

Enter the size of the processSize array: 3

Enter the value for processSize[1]: 212

Enter the value for processSize[2]: 417

Enter the value for processSize[3]: 426

Process No. Process Size Block no.

P1 212 2

P2 417 5

P3 426 Not Allocated

Worst fit ------------>

#include <iostream>

using namespace std;

void worstFit(int blockSize[], int m, int processSize[], int n)

{

    int allocation[n];

    for (int i = 0; i < n; i++)

        allocation[i] = -1;

    for (int i=0; i<n; i++)

    {

        int wstIdx = -1;

        for (int j=0; j<m; j++)

        {

            if (blockSize[j] >= processSize[i])

            {

                if (wstIdx == -1 || blockSize[wstIdx] < blockSize[j])

                    wstIdx = j;

            }

        }

        if (wstIdx != -1)

        {

            allocation[i] = wstIdx;

            blockSize[wstIdx] -= processSize[i];

        }

    }

    cout << "\nProcess No.\tProcess Size\tBlock no.\n";

    for (int i = 0; i < n; i++)

    {

        cout << " P" << i+1 << "\t\t" << processSize[i] << "\t\t";

        if (allocation[i] != -1)

            cout << allocation[i] + 1;

        else

            cout << "Not Allocated";

        cout << endl;

    }

}

int main()

{

       int m;

       cout << "Enter the size of the blockSize array: ";

       cin >> m;

       int blockSize[m];

       for (int i = 0; i < m; i++) {

              cout << "Enter the value for blockSize[" << i << "]: ";

              cin >> blockSize[i];

       }

       int n;

       cout << "Enter the size of the processSize array: ";

       cin >> n;

       int processSize[n];

       for (int i = 0; i < n; i++) {

              cout << "Enter the value for processSize[" << i+1 << "]: ";

              cin >> processSize[i];

       }

        worstFit(blockSize, m, processSize, n);

        return 0 ;

}

Output ------------------->

Enter the size of the blockSize array: 5

Enter the value for blockSize[0]: 100

Enter the value for blockSize[1]: 500

Enter the value for blockSize[2]: 200

Enter the value for blockSize[3]: 300

Enter the value for blockSize[4]: 600

Enter the size of the processSize array: 3

Enter the value for processSize[1]: 212

Enter the value for processSize[2]: 417

Enter the value for processSize[3]: 426

Process No. Process Size Block no.

P1 212 5

P2 417 2

P3 426 Not Allocated

Next\_fit ----------------->

#include <iostream>

using namespace std;

void NextFit(int blockSize[], int m, int processSize[], int n)

{

    int allocation[n], j = 0, t = m - 1;

    for (int i = 0; i < n; i++)

        allocation[i] = -1;

    for(int i = 0; i < n; i++){

        while (j < m){

            if(blockSize[j] >= processSize[i]){

                allocation[i] = j;

                blockSize[j] -= processSize[i];

                t = (j - 1) % m;

                break;

            }

            if (t == j){

                t = (j - 1) % m;

                break;

            }

            j = (j + 1) % m;

        }

    }

    cout << "\nProcess No.\tProcess Size\tBlock no.\n";

    for (int i = 0; i < n; i++) {

        cout << " P" << i + 1 << "\t\t" << processSize[i]

            << "\t\t";

        if (allocation[i] != -1)

            cout << allocation[i] + 1;

        else

            cout << "Not Allocated";

        cout << endl;

    }

}

int main()

{

       int m;

       cout << "Enter the size of the blockSize array: ";

       cin >> m;

       int blockSize[m];

       for (int i = 0; i < m; i++) {

              cout << "Enter the value for blockSize[" << i << "]: ";

              cin >> blockSize[i];

       }

       int n;

       cout << "Enter the size of the processSize array: ";

       cin >> n;

       int processSize[n];

       for (int i = 0; i < n; i++) {

              cout << "Enter the value for processSize[" << i+1 << "]: ";

              cin >> processSize[i];

       }

        NextFit(blockSize, m, processSize, n);

        return 0;

}

output ------------------>

Enter the size of the blockSize array: 5

Enter the value for blockSize[0]: 100

Enter the value for blockSize[1]: 500

Enter the value for blockSize[2]: 200

Enter the value for blockSize[3]: 300

Enter the value for blockSize[4]: 600

Enter the size of the processSize array: 3

Enter the value for processSize[1]: 212

Enter the value for processSize[2]: 417

Enter the value for processSize[3]: 426

Process No. Process Size Block no.

P1 212 2

P2 417 5

P3 426 Not Allocated

Best\_fit --------------------->

#include<iostream>

using namespace std;

void bestFit(int blockSize[], int m, int processSize[], int n)

{

    int allocation[n] ;

    for (int i = 0; i < n; i++)

        allocation[i] = -1;

    for (int i = 0; i < n; i++)

    {

        int bestIdx = -1;

        for (int j = 0; j < m; j++)

        {

            if (blockSize[j] >= processSize[i])

            {

                if (bestIdx == -1 || blockSize[bestIdx] > blockSize[j])

                    bestIdx = j;

            }

        }

        if (bestIdx != -1)

        {

            allocation[i] = bestIdx;

            blockSize[bestIdx] -= processSize[i];

        }

    }

    cout << "\nProcess No.\tProcess Size\tBlock no.\n";

    for (int i = 0; i < n; i++)

    {

        cout << " P" << i+1 << "\t\t" << processSize[i] << "\t\t";

        if (allocation[i] != -1)

            cout << allocation[i] + 1;

        else

            cout << "Not Allocated";

        cout << endl;

    }

}

int main()

{

       int m;

       cout << "Enter the size of the blockSize array: ";

       cin >> m;

       int blockSize[m];

       for (int i = 0; i < m; i++) {

              cout << "Enter the value for blockSize[" << i << "]: ";

              cin >> blockSize[i];

       }

       int n;

       cout << "Enter the size of the processSize array: ";

       cin >> n;

       int processSize[n];

       for (int i = 0; i < n; i++) {

              cout << "Enter the value for processSize[" << i+1 << "]: ";

              cin >> processSize[i];

       }

        bestFit(blockSize, m, processSize, n);

        return 0 ;

}

Output ------------------>

Enter the size of the blockSize array: 5

Enter the value for blockSize[0]: 100

Enter the value for blockSize[1]: 500

Enter the value for blockSize[2]: 200

Enter the value for blockSize[3]: 300

Enter the value for blockSize[4]: 600

Enter the size of the processSize array: 3

Enter the value for processSize[1]: 212

Enter the value for processSize[2]: 417

Enter the value for processSize[3]: 426

Process No. Process Size Block no.

P1 212 4

P2 417 2

P3 426 5

Semaphore ---------------->

#include <iostream>

#include <thread>

#include <mutex>

#include <condition\_variable>

#include <queue>

#include <chrono>

using namespace std;

const int BUFFER\_SIZE = 5;

mutex mtx;

condition\_variable buffer\_not\_full, buffer\_not\_empty;

queue<int> buffer;

void producer() {

    int item;

    cout << "Enter the item to produce: ";

    cin >> item;

    unique\_lock<mutex> lock(mtx);

    if (buffer.size() >= BUFFER\_SIZE) {

        cout << "Buffer is full. Producer is waiting." << endl;

        // Introduce a timeout

        if (buffer\_not\_full.wait\_for(lock, chrono::seconds(4)) == cv\_status::timeout) {

            cout << "Timeout: Buffer is still full." << endl;

        }

    }

    if (buffer.size() < BUFFER\_SIZE) {

        buffer.push(item);

        cout << "Produced: " << item << endl;

        lock.unlock();

        buffer\_not\_empty.notify\_one();

    }

}

void consumer() {

    unique\_lock<mutex> lock(mtx);

    if (buffer.empty()) {

        cout << "Buffer is empty. Consumer is waiting." << endl;

        // Introduce a timeout

        if (buffer\_not\_empty.wait\_for(lock, chrono::seconds(4)) == cv\_status::timeout) {

            cout << "Timeout: Buffer is still empty." << endl;

        }

    }

    if (!buffer.empty()) {

        int item = buffer.front();

        buffer.pop();

        cout << "Consumed: " << item << endl;

        lock.unlock();

        buffer\_not\_full.notify\_one();

    }

}

int main() {

    while (true) {

        int choice;

        cout << "Menu:" << endl;

        cout << "1. Producer" << endl;

        cout << "2. Consumer" << endl;

        cout << "3. Exit" << endl;

        cout << "Enter your choice: ";

        cin >> choice;

        switch (choice) {

            case 1:

                producer();

                break;

            case 2:

                consumer();

                break;

            case 3:

                return 0;

            default:

                cout << "Invalid choice. Please try again." << endl;

                break;

        }

    }

}

Output --------------------->

Menu:

1. Producer

2. Consumer

3. Exit

Enter your choice: 2

Buffer is empty. Consumer is waiting.

Timeout: Buffer is still empty.

Menu:

1. Producer

2. Consumer

3. Exit

Enter your choice: 1

Enter the item to produce: 100

Produced: 100

Menu:

1. Producer

2. Consumer

3. Exit

Enter your choice: 1

Enter the item to produce: 200

Produced: 200

Menu:

1. Producer

2. Consumer

3. Exit

Enter your choice: 1

Enter the item to produce: 300

Produced: 300

Menu:

1. Producer

2. Consumer

3. Exit

Enter your choice: 1

Enter the item to produce: 400

Produced: 400

Menu:

1. Producer

2. Consumer

3. Exit

Enter your choice: 1

Enter the item to produce: 500

Produced: 500

Menu:

1. Producer

2. Consumer

3. Exit

Enter your choice: 1

Enter the item to produce: 600

Buffer is full. Producer is waiting.

Timeout: Buffer is still full.

Menu:

1. Producer

2. Consumer

3. Exit

Enter your choice: 2

Consumed: 100

Menu:

1. Producer

2. Consumer

3. Exit

Enter your choice: 1

Enter the item to produce: 600

Produced: 600

Menu:

1. Producer

2. Consumer

3. Exit

Enter your choice:3