**CPU SCHEDULING**

1. **FCFS**

//FCFS cpu scheduling

#include<stdio.h>

struct process

{

int at, bt, tt, wt;

};

struct process a[10];

void main()

{

int n, burst=0, comp\_t;

float avg\_tt, avg\_wt,total;

printf("\nEnter the number of processes: ");

scanf("%d", &n);

for(int i=0;i<n;i++)

{

printf("\nEnter the arrival time and burst time of process %d: ", i+1);

scanf("%d %d", &a[i].at, &a[i].bt);

}

//calculate turn around time

comp\_t=0;

total=0;

for(int i=0;i<n;i++)

{

comp\_t+=a[i].bt;

a[i].tt=comp\_t-a[i].at;

total+=a[i].tt;

}

avg\_tt=total/n;

//calculate waiting time

total=0;

for(int i=0;i<n;i++)

{

a[i].wt=a[i].tt-a[i].bt;

total+=a[i].wt;

}

avg\_wt=total/n;

//printing putput

printf("PROCESS \t\t ARRIVAL \t\t BURST \t\t TURN AROUND \t WAITING \n");

for(int i=0;i<n;i++)

{

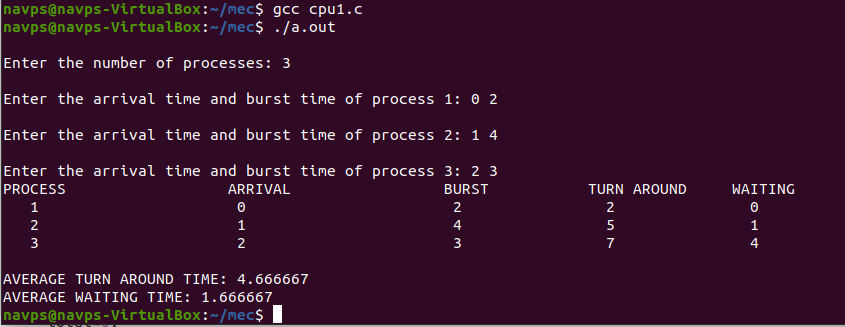
printf(" %d \t\t %d \t\t %d \t\t %d \t %d \n" ,i+1, a[i].at, a[i].bt, a[i].tt, a[i].wt );

}

printf("\nAVERAGE TURN AROUND TIME: %f", avg\_tt);

printf("\nAVERAGE WAITING TIME: %f \n", avg\_wt);

}



1. **SJF non preemptive**

//sjf non preemptive

#include<stdio.h>

struct process

{

int id, at, bt, wt, tt;

}a[10];

void swap(int \*a, int \*b)

{

int temp;

temp=\*a;

\*a=\*b;

\*b=temp;

}

void main()

{

int n, check\_ar=0,comp\_t=0, min\_bt;

float total\_wt=0, total\_tt=0, avg\_wt, avg\_tt;

printf("\nEnter the number of processes");

scanf("%d", &n);

for(int i=0;i<n;i++)

{

printf("\nEnter the arrival time and burst time of process %d: ", i+1);

scanf("%d %d", &a[i].at, &a[i].bt);

a[i].id=i+1;

if(i==0)

check\_ar=a[i].at;

if(check\_ar!=a[i].at)

check\_ar=1;

}

//bubble sort based on arival time if arrival times are different

if(check\_ar!=0)

{

for(int i=0;i<n;i++)

{

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

{

if(a[j].at>a[j+1].at)

{ swap(&a[j].id,&a[j+1].id);

swap(&a[j].at,&a[j+1].at);

swap(&a[j].bt,&a[j+1].bt);

}

}

}

}

//sort based on burst time for different arrival times

if(check\_ar!=0)

{

comp\_t=a[0].at+a[0].bt;

a[0].tt= comp\_t-a[0].at;

a[0].wt=a[0].tt-a[0].bt;

total\_wt=a[0].wt;

total\_tt=a[0].tt;

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

{

min\_bt=a[i].bt;

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

{

if(a[j].bt<min\_bt && a[j].at<=comp\_t)

{

min\_bt=a[j].bt;

swap(&a[i].id,&a[j].id);

swap(&a[i].at,&a[j].at);

swap(&a[i].bt,&a[j].bt);

}

}

// completion time of the process

comp\_t+=a[i].bt;

// Turn Around Time of the process

// compl-Arrival

a[i].tt=comp\_t-a[i].at;

a[i].wt=a[i].tt-a[i].bt;

total\_wt+=a[i].wt;

total\_tt+=a[i].tt;

}

}

//sort based on burst time for same arrival times

else

{

for(int i=0;i<n;i++)

{

min\_bt=a[i].bt;

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

{

if(a[j].bt<min\_bt && a[j].at<=comp\_t)

{

min\_bt=a[j].bt;

swap(&a[i].id,&a[j].id);

swap(&a[i].at,&a[j].at);

swap(&a[i].bt,&a[j].bt);

}

}

// completion time of the process

comp\_t+=a[i].bt;

// Turn Around Time of the process

// compl-Arrival

a[i].tt=comp\_t-a[i].at;

a[i].wt=a[i].tt-a[i].bt;

total\_wt+=a[i].wt;

total\_tt+=a[i].tt;

}

}

avg\_wt=total\_wt/n;

avg\_tt=total\_tt/n;

printf("PROCESS \t\t ARRIVAL \t\t BURST \t\t TURN AROUND \t WAITING \n");

for(int i=0;i<n;i++)

{

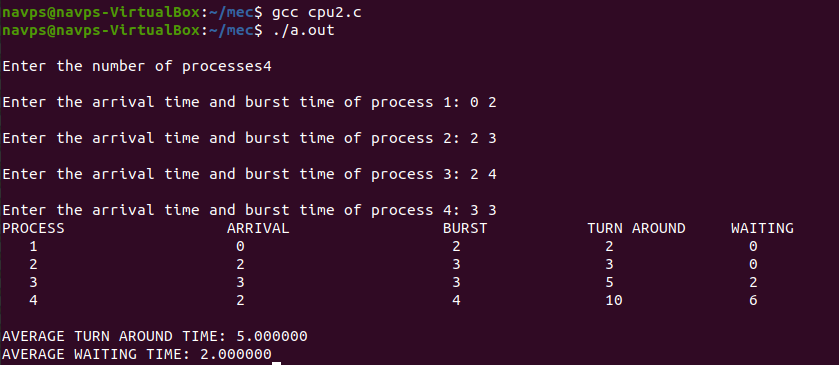
printf(" %d \t\t %d \t\t %d \t\t %d \t %d \n" ,i+1, a[i].at, a[i].bt, a[i].tt, a[i].wt );

}

printf("\nAVERAGE TURN AROUND TIME: %f", avg\_tt);

printf("\nAVERAGE WAITING TIME: %f \n", avg\_wt);

}



1. **SJF preemptive**

//sjf preemptive

#include<stdio.h>

struct process

{

int at, bt, wt, tt;

}a[10];

void main()

{

int n, count=0,temp[10],t,short\_p;

float total\_tt=0, total\_wt=0, avg\_tt=0, avg\_wt=0;

printf("\nEnter the number of proecesses: ");

scanf("%d",&n);

for(int i=0;i<n;i++)

{

printf("\nEnter the arrival time and burst time of process %d: ", i+1);

scanf("%d %d", &a[i].at, &a[i].bt);

temp[i]=a[i].bt;

}

a[9].bt=10000;

for(t=0;count!=n;t++)

{

short\_p=9;

for(int i=0;i<n;i++)

{

//lesser burst time, lesser arrival time, positive burst time

if(a[i].bt<a[short\_p].bt && (a[i].at<=t && a[i].bt>0))

short\_p=i;

}

a[short\_p].bt--;

if(a[short\_p].bt==0)

{

count++;

a[short\_p].tt=t+1-a[short\_p].at;

a[short\_p].wt=a[short\_p].tt-temp[short\_p];

total\_tt+=a[short\_p].tt;

total\_wt+=a[short\_p].wt;

}

}

avg\_tt=total\_tt/n;

avg\_wt=total\_wt/n;

printf("PROCESS \t\t ARRIVAL \t\t BURST \t\t TURN AROUND \t WAITING \n");

for(int i=0;i<n;i++)

{

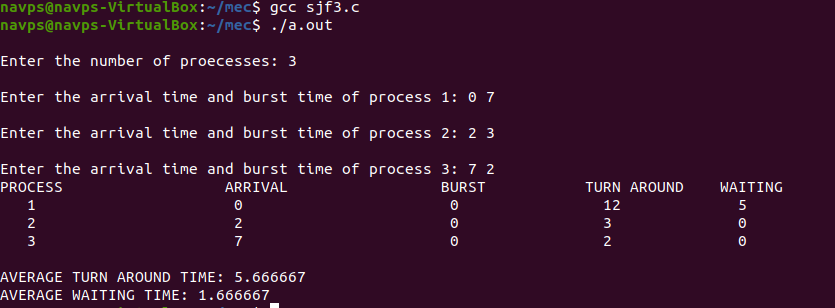
printf(" %d \t\t %d \t\t %d \t\t %d \t %d \n" ,i+1, a[i].at, a[i].bt, a[i].tt, a[i].wt );

}

printf("\nAVERAGE TURN AROUND TIME: %f", avg\_tt);

printf("\nAVERAGE WAITING TIME: %f \n", avg\_wt);

}



1. **PRIORITY non preemptive**

#include<stdio.h>

struct process

{

int id,WT,AT,BT,TAT,PR;

};

struct process a[10];

// function for swapping

void swap(int \*b,int \*c)

{

int tem;

tem=\*c;

\*c=\*b;

\*b=tem;

}

//Driver function

int main()

{

int n,check\_ar=0;

int Cmp\_time=0;

float Total\_WT=0,Total\_TAT=0,Avg\_WT,Avg\_TAT;

printf("Enter the number of process \n");

scanf("%d",&n);

printf("Enter the Arrival time , Burst time and priority of the process\n");

printf("AT BT PR\n");

for(int i=0;i<n;i++)

{

scanf("%d%d%d",&a[i].AT,&a[i].BT,&a[i].PR);

a[i].id=i+1;

// here we are checking that arrival time

// of the process are same or different

if(i==0)

check\_ar=a[i].AT;

if(check\_ar!=a[i].AT )

check\_ar=1;

}

// if process are arrived at the different time

// then sort the process on the basis of AT

if(check\_ar!=0)

{

for(int i=0;i<n;i++)

{

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

{

if(a[j].AT>a[j+1].AT)

{

swap(&a[j].id,&a[j+1].id);

swap(&a[j].AT,&a[j+1].AT);

swap(&a[j].BT,&a[j+1].BT);

swap(&a[j].PR,&a[j+1].PR);

}

}

}

}

// logic of Priority scheduling ( non preemptive) algo

// if all the process are arrived at different time

if(check\_ar!=0)

{

comp\_t=a[0].at+a[0].bt;

a[0].tt= comp\_t-a[0].at;

a[0].wt=a[0].tt-a[0].bt;

total\_wt=a[0].wt;

total\_tt=a[0].tt;

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

{

int min=a[i].PR;

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

{

if(min>a[j].PR && a[j].AT<=Cmp\_time)

{

min=a[j].PR;

swap(&a[i].id,&a[j].id);

swap(&a[i].AT,&a[j].AT);

swap(&a[i].BT,&a[j].BT);

swap(&a[i].PR,&a[j].PR);

}

}

// completion time of the process

comp\_t+=a[i].bt;

// Turn Around Time of the process

// compl-Arrival

a[i].tt=comp\_t-a[i].at;

a[i].wt=a[i].tt-a[i].bt;

total\_wt+=a[i].wt;

total\_tt+=a[i].tt;

}

}

// if all the process are arrived at same time

else

{

for(int i=0;i<n;i++)

{

int min=a[i].PR;

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

{

if(min>a[j].PR && a[j].AT<=Cmp\_time)

{

min=a[j].PR;

swap(&a[i].id,&a[j].id);

swap(&a[i].AT,&a[j].AT);

swap(&a[i].BT,&a[j].BT);

swap(&a[i].PR,&a[j].PR);

}

}

// completion time of the process

comp\_t+=a[i].bt;

// Turn Around Time of the process

// compl-Arrival

a[i].tt=comp\_t-a[i].at;

a[i].wt=a[i].tt-a[i].bt;

total\_wt+=a[i].wt;

total\_tt+=a[i].tt;

}

}

Avg\_WT=Total\_WT/n;

Avg\_TAT=Total\_TAT/n;

// Printing of the results

printf("The process are\n");

printf("ID WT TAT\n");

for(int i=0;i<n;i++)

{

printf("%d\t%d\t%d\n",a[i].id,a[i].WT,a[i].TAT);

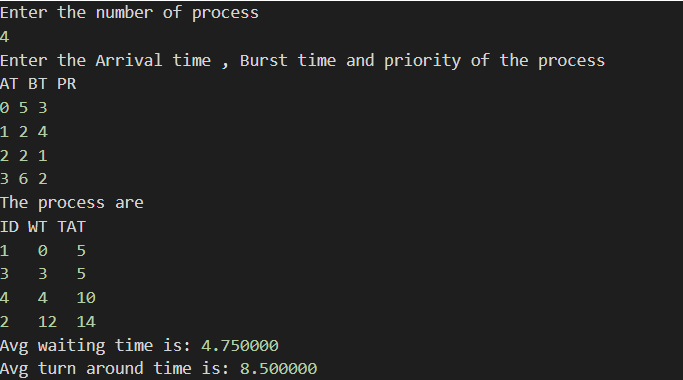
}

printf("Avg waiting time is: %f\n",Avg\_WT);

printf("Avg turn around time is: %f",Avg\_TAT);

return 0;

}



1. **PRIORITY preemptive**

#include<stdio.h>

struct process

{

int WT,AT,BT,TAT,PT;

};

struct process a[10];

int main()

{

int n,temp[10],t,count=0,short\_p;

float total\_WT=0,total\_TAT=0,Avg\_WT,Avg\_TAT;

printf("Enter the number of the process\n");

scanf("%d",&n);

printf("Enter the arrival time , burst time and priority of the process\n");

printf("AT BT PT\n");

for(int i=0;i<n;i++)

{

scanf("%d%d%d",&a[i].AT,&a[i].BT,&a[i].PT);

// copying the burst time in

// a temp array fot futher use

temp[i]=a[i].BT;

}

// we initialize the burst time

// of a process with maximum

a[9].PT=10000;

for(t=0;count!=n;t++)

{

short\_p=9;

for(int i=0;i<n;i++)

{

if(a[short\_p].PT>a[i].PT && a[i].AT<=t && a[i].BT>0)

{

short\_p=i;

}

}

a[short\_p].BT=a[short\_p].BT-1;

// if any process is completed

if(a[short\_p].BT==0)

{

// one process is completed

// so count increases by 1

count++;

a[short\_p].WT=t+1-a[short\_p].AT-temp[short\_p];

a[short\_p].TAT=t+1-a[short\_p].AT;

// total calculation

total\_WT=total\_WT+a[short\_p].WT;

total\_TAT=total\_TAT+a[short\_p].TAT;

}

}

Avg\_WT=total\_WT/n;

Avg\_TAT=total\_TAT/n;

// printing of the answer

printf("ID WT TAT\n");

for(int i=0;i<n;i++)

{

printf("%d %d\t%d\n",i+1,a[i].WT,a[i].TAT);

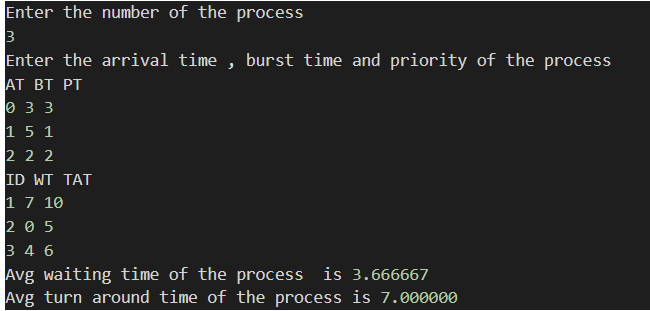
}

printf("Avg waiting time of the process is %f\n",Avg\_WT);

printf("Avg turn around time of the process is %f\n",Avg\_TAT);

return 0;

}



1. **ROUND ROBIN - preemptive only**

#include<stdio.h>

struct process

{

int id,AT,BT,WT,TAT;

};

struct process a[10];

// declaration of the ready queue

int queue[100];

int front=-1;

int rear=-1;

// function for insert the element

// into queue

void insert(int n)

{

if(front==-1)

front=0;

rear=rear+1;

queue[rear]=n;

}

// function for delete the

// element from queue

int delete()

{

int n;

n=queue[front];

front=front+1;

return n;

}

int main()

{

int n,TQ,p,TIME=0;

int temp[10],exist[10]={0};

float total\_wt=0,total\_tat=0,Avg\_WT,Avg\_TAT;

printf("Enter the number of the process\n");

scanf("%d",&n);

printf("Enter the arrival time and burst time of the process\n");

printf("AT BT\n");

for(int i=0;i<n;i++)

{

scanf("%d%d",&a[i].AT,&a[i].BT);

a[i].id=i;

temp[i]=a[i].BT;

}

printf("Enter the time quantum\n");

scanf("%d",&TQ);

// logic for round robin scheduling

// insert first process

// into ready queue

insert(0);

exist[0]=1;

// until ready queue is empty

while(front<=rear)

{

p=delete();

if(a[p].BT>=TQ)

{

a[p].BT=a[p].BT-TQ;

TIME=TIME+TQ;

}

else

{

TIME=TIME+a[p].BT;

a[p].BT=0;

}

//if process is not exist

// in the ready queue even a single

// time then insert it if it arrive

// at time 'TIME'

for(int i=0;i<n;i++)

{

if(exist[i]==0 && a[i].AT<=TIME)

{

insert(i);

exist[i]=1;

}

}

// if process is completed

if(a[p].BT==0)

{

a[p].TAT=TIME-a[p].AT;

a[p].WT=a[p].TAT-temp[p];

total\_tat=total\_tat+a[p].TAT;

total\_wt=total\_wt+a[p].WT;

}

else

{

insert(p);

}

}

Avg\_TAT=total\_tat/n;

Avg\_WT=total\_wt/n;

// printing of the answer

printf("ID WT TAT\n");

for(int i=0;i<n;i++)

{

printf("%d %d %d\n",a[i].id,a[i].WT,a[i].TAT);

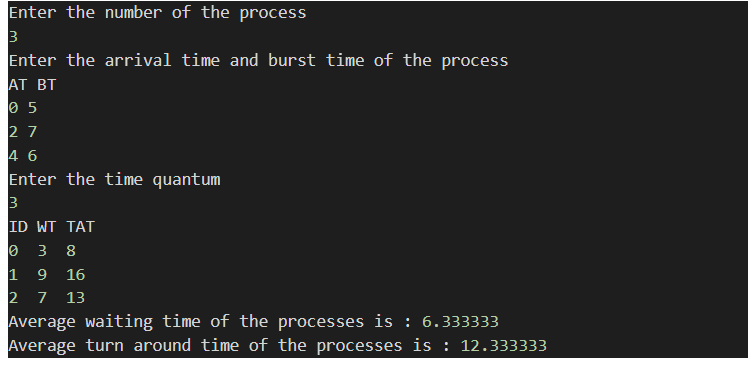
}

printf("Average waiting time of the processes is : %f\n",Avg\_WT);

printf("Average turn around time of the processes is : %f\n",Avg\_TAT);

return 0;

}



1. **FIRST FIT**

// C implementation of First - Fit algorithm

#include<stdio.h>

// Function to allocate memory to

// blocks as per First fit algorithm

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

{

int i, j;

// Stores block id of the

// block allocated to a process

int allocation[n];

// Initially no block is assigned to any process

for(i = 0; i < n; i++)

{

allocation[i] = -1;

}

// pick each process and find suitable blocks

// according to its size ad assign to it

for (i = 0; i < n; i++) //here, n -> number of processes

{

for (j = 0; j < m; j++) //here, m -> number of blocks

{

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

{

// allocating block j to the ith process

allocation[i] = j;

// Reduce available memory in this block.

blockSize[j] -= processSize[i];

break; //go to the next process in the queue

}

}

}

printf("\nProcess No.\tProcess Size\tBlock no.\n");

for (int i = 0; i < n; i++)

{

printf(" %i\t\t\t", i+1);

printf("%i\t\t\t\t", processSize[i]);

if (allocation[i] != -1)

printf("%i", allocation[i] + 1);

else

printf("Not Allocated");

printf("\n");

}

}

// Driver code

int main()

{

int m; //number of blocks in the memory

int n; //number of processes in the input queue

int blockSize[] = {100, 500, 200, 300, 600};

int processSize[] = {212, 417, 112, 426};

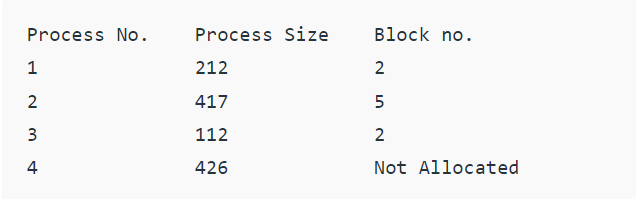
m = sizeof(blockSize) / sizeof(blockSize[0]);

n = sizeof(processSize) / sizeof(processSize[0]);

firstFit(blockSize, m, processSize, n);

return 0 ;

}



1. **BEST FIT**

#include<bits/stdc++.h>

using namespace std;

// Function to allocate memory to blocks as per Best fit

// algorithm

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

{

// Stores block id of the block allocated to a

// process

int allocation[n];

// Initially no block is assigned to any process

memset(allocation, -1, sizeof(allocation));

// pick each process and find suitable blocks

// according to its size ad assign to it

for (int i=0; i<n; i++)

{

// Find the best fit block for current process

int bestIdx = -1;

for (int j=0; j<m; j++)

{

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

{

if (bestIdx == -1)

bestIdx = j;

else if (blockSize[bestIdx] > blockSize[j])

bestIdx = j;

}

}

// If we could find a block for current process

if (bestIdx != -1)

{

// allocate block j to p[i] process

allocation[i] = bestIdx;

// Reduce available memory in this block.

blockSize[bestIdx] -= processSize[i];

}

}

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

for (int i = 0; i < n; i++)

{

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

if (allocation[i] != -1)

cout << allocation[i] + 1;

else

cout << "Not Allocated";

cout << endl;

}

}

// Driver code

int main()

{

int blockSize[] = {100, 500, 200, 300, 600};

int processSize[] = {212, 417, 112, 426};

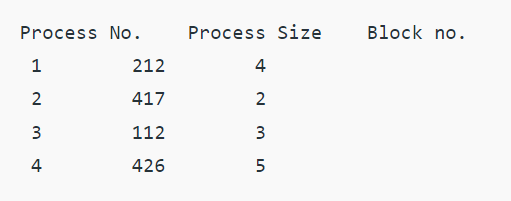
int m = sizeof(blockSize)/sizeof(blockSize[0]);

int n = sizeof(processSize)/sizeof(processSize[0]);

bestFit(blockSize, m, processSize, n);

return 0 ;

}



1. **WORST FIT**

// C++ implementation of worst - Fit algorithm

#include<bits/stdc++.h>

**using** **namespace** std;

// Function to allocate memory to blocks as per worst fit

// algorithm

**void** worstFit(**int** blockSize[], **int** m, **int** processSize[],

**int** n)

{

    // Stores block id of the block allocated to a

    // process

**int** allocation[n];

    // Initially no block is assigned to any process

**memset**(allocation, -1, **sizeof**(allocation));

    // pick each process and find suitable blocks

    // according to its size ad assign to it

**for** (**int** i=0; i<n; i++)

    {

        // Find the best fit block for current process

**int** wstIdx = -1;

**for** (**int** j=0; j<m; j++)

        {

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

            {

**if** (wstIdx == -1)

                    wstIdx = j;

**else** **if** (blockSize[wstIdx] < blockSize[j])

                    wstIdx = j;

            }

        }

        // If we could find a block for current process

**if** (wstIdx != -1)

        {

            // allocate block j to p[i] process

            allocation[i] = wstIdx;

            // Reduce available memory in this block.

            blockSize[wstIdx] -= processSize[i];

        }

    }

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

**for** (**int** i = 0; i < n; i++)

    {

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

**if** (allocation[i] != -1)

            cout << allocation[i] + 1;

**else**

            cout << "Not Allocated";

        cout << endl;

    }

}

// Driver code

**int** main()

{

**int** blockSize[] = {100, 500, 200, 300, 600};

**int** processSize[] = {212, 417, 112, 426};

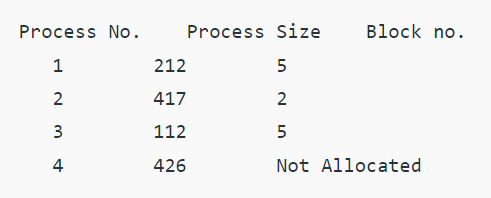
**int** m = **sizeof**(blockSize)/**sizeof**(blockSize[0]);

**int** n = **sizeof**(processSize)/**sizeof**(processSize[0]);

    worstFit(blockSize, m, processSize, n);

**return** 0 ;

}



1. **FIFO page replacement**

#include<stdio.h>

int main()

{

int i,j,n,a[50],frame[10],no,k,avail,count=0;

            printf("\n ENTER THE NUMBER OF PAGES:\n");

scanf("%d",&n);

            printf("\n ENTER THE PAGE NUMBER :\n");

            for(i=1;i<=n;i++)

            scanf("%d",&a[i]);

            printf("\n ENTER THE NUMBER OF FRAMES :");

            scanf("%d",&no);

for(i=0;i<no;i++)

            frame[i]= -1;

                        j=0;

                        printf("\tref string\t page frames\n");

for(i=1;i<=n;i++)

                        {

                                    printf("%d\t\t",a[i]);

                                    avail=0;

                                    for(k=0;k<no;k++)

if(frame[k]==a[i])

                                                avail=1;

                                    if (avail==0)

                                    {

                                                frame[j]=a[i];

                                                j=(j+1)%no;

                                                count++;

                                                for(k=0;k<no;k++)

                                                printf("%d\t",frame[k]);

}

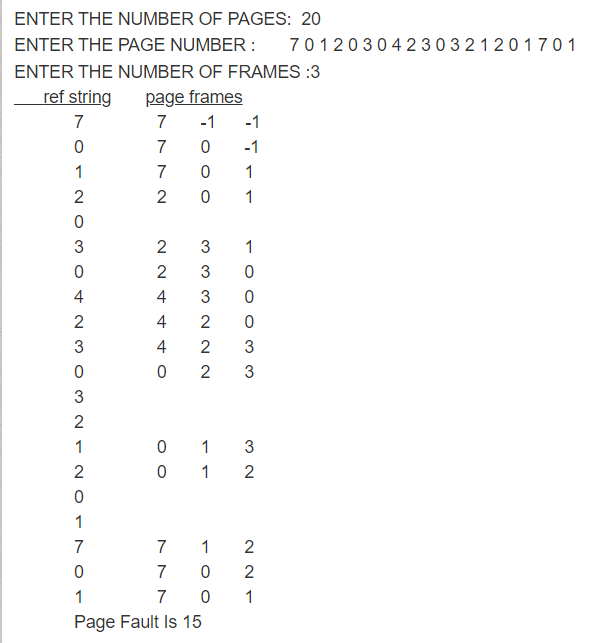
                                    printf("\n");

}

                        printf("Page Fault Is %d",count);

                        return 0;

}



1. **LRU page replacement**

#include<stdio.h>

**int** findLRU(**int** time[], **int** n){

**int** i, minimum = time[0], pos = 0;

**for**(i = 1; i < n; ++i){

**if**(time[i] < minimum){

minimum = time[i];

pos = i;

}

}

**return** pos;

}

**int** main()

{

**int** no\_of\_frames, no\_of\_pages, frames[10], pages[30], counter = 0, time[10], flag1, flag2, i, j, pos, faults = 0;

**printf**("Enter number of frames: ");

scanf("%d", &no\_of\_frames);

**printf**("Enter number of pages: ");

scanf("%d", &no\_of\_pages);

**printf**("Enter reference string: ");

**for**(i = 0; i < no\_of\_pages; ++i){

     scanf("%d", &pages[i]);

    }

**for**(i = 0; i < no\_of\_frames; ++i){

     frames[i] = -1;

    }

**for**(i = 0; i < no\_of\_pages; ++i){

     flag1 = flag2 = 0;

**for**(j = 0; j < no\_of\_frames; ++j){

**if**(frames[j] == pages[i]){

     counter++;

     time[j] = counter;

   flag1 = flag2 = 1;

**break**;

   }

     }

**if**(flag1 == 0){

**for**(j = 0; j < no\_of\_frames; ++j){

**if**(frames[j] == -1){

     counter++;

     faults++;

     frames[j] = pages[i];

     time[j] = counter;

     flag2 = 1;

**break**;

     }

     }

     }

**if**(flag2 == 0){

     pos = findLRU(time, no\_of\_frames);

     counter++;

     faults++;

     frames[pos] = pages[i];

     time[pos] = counter;

     }

**printf**("\n");

**for**(j = 0; j < no\_of\_frames; ++j){

**printf**("%d\t", frames[j]);

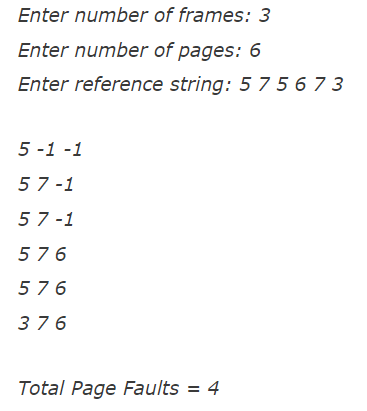
     }

}

**printf**("\n\nTotal Page Faults = %d", faults);

**return** 0;

}



1. **LFU page replacement**

#include<stdio.h>

void print(int frameno,int frame[])

{

            int j;

            for(j=0;j<frameno;j++)

            printf("%d\t",frame[j]);

            printf("\n");

}

int main()

{

            int i,j,k,n,page[50],frameno,frame[10],move=0,flag,count=0,count1[10]={0},

                        repindex,leastcount;

            float rate;

            printf("Enter the number of pages\n");

            scanf("%d",&n);

            printf("Enter the page reference numbers\n");

            for(i=0;i<n;i++)

            scanf("%d",&page[i]);

            printf("Enter the number of frames\n");

            scanf("%d",&frameno);

            for(i=0;i<frameno;i++)

            frame[i]=-1;

            printf("Page reference string\tFrames\n");

            for(i=0;i<n;i++)

            {

                        printf("%d\t\t\t",page[i]);

                        flag=0;

                        for(j=0;j<frameno;j++)

                        {

                                    if(page[i]==frame[j])

                                    {

                                                flag=1;

                                                count1[j]++;

                                                printf("No replacement\n");

                                                break;

                                    }

                        }

                        if(flag==0&&count<frameno)

                        {

                                    frame[move]=page[i];

                                    count1[move]=1;

                                    move=(move+1)%frameno;

                                    count++;

                                    print(frameno,frame);

                        }

                        else if(flag==0)

                        {

                                    repindex=0;

                                    leastcount=count1[0];

                                    for(j=1;j<frameno;j++)

                                    {

                                                if(count1[j]<leastcount)

                                                {

                                                            repindex=j;

                                                            leastcount=count1[j];

                                                }

                                    }

                                    frame[repindex]=page[i];

                                    count1[repindex]=1;

                                    count++;

                                    print(frameno,frame);

                        }

            }

            rate=(float)count/(float)n;

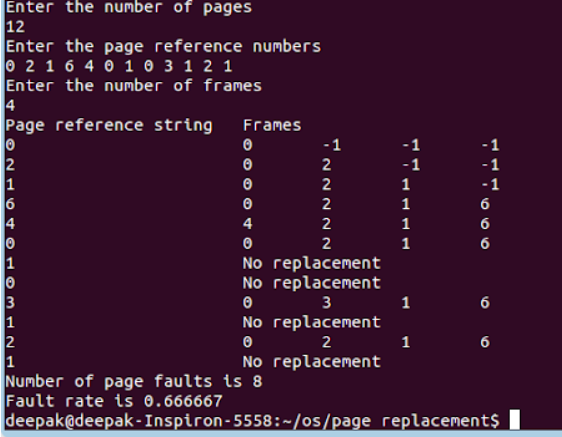
            printf("Number of page faults is %d\n",count);

            printf("Fault rate is %f\n",rate);

            return 0;

}

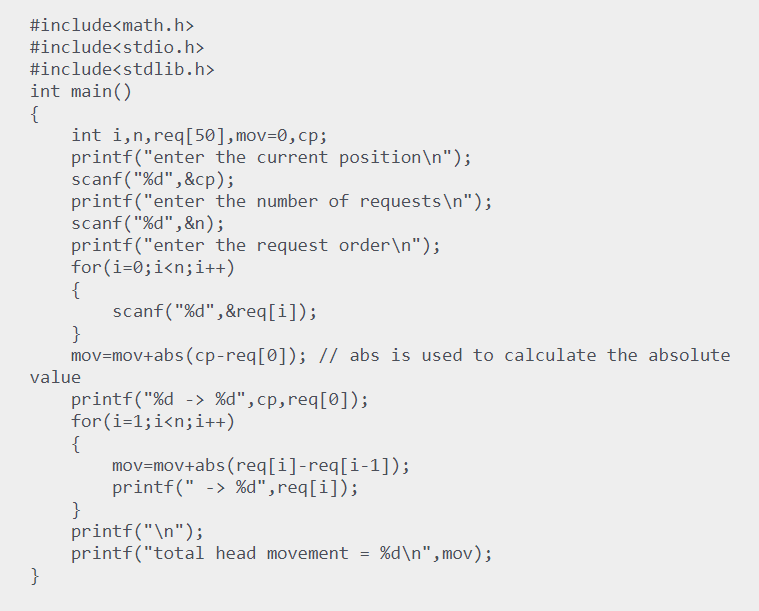
 **OUTPUT :**

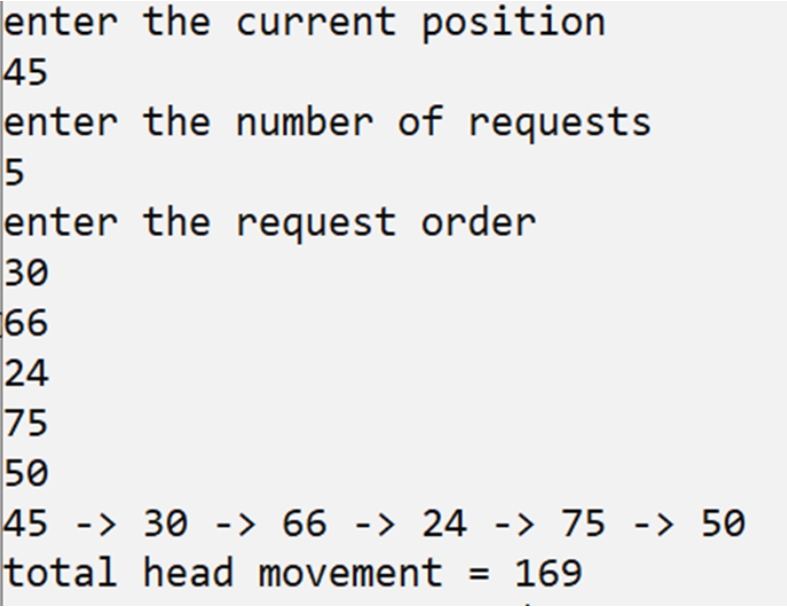


1. **BANKERS ALGORITHM**

**Check pdf**

1. **FCFS DISK SCHEDULING ALGO**





1. **SCAN DISK SCHEDULING**

#include <stdio.h>

#include <math.h>

int main()

{

    int queue[20], n, head, i, j, k, seek = 0, max, diff, temp, queue1[20],

    queue2[20], temp1 = 0, temp2 = 0;

    float avg;

    printf("Enter the max range of disk\n");

    scanf("%d", &max);

    printf("Enter the initial head position\n");

    scanf("%d", &head);

    printf("Enter the size of queue request\n");

    scanf("%d", &n);

    printf("Enter the queue of disk positions to be read\n");

    for (i = 1; i <= n; i++)

    {

        scanf("%d", &temp);

        if (temp >= head)

        {

            queue1[temp1] = temp;

            temp1++;

        }

        else

        {

            queue2[temp2] = temp;

            temp2++;

        }

    }

    for (i = 0; i < temp1 - 1; i++)

    {

        for (j = i + 1; j < temp1; j++)

        {

            if (queue1[i] > queue1[j])

            {

                temp = queue1[i];

                queue1[i] = queue1[j];

                queue1[j] = temp;

            }

        }

    }

    for (i = 0; i < temp2 - 1; i++)

    {

        for (j = i + 1; j < temp2; j++)

        {

            if (queue2[i] < queue2[j])

            {

                temp = queue2[i];

                queue2[i] = queue2[j];

                queue2[j] = temp;

            }

        }

    }

    for (i = 1, j = 0; j < temp1; i++, j++)

        queue[i] = queue1[j];

    queue[i] = max;

    for (i = temp1 + 2, j = 0; j < temp2; i++, j++)

        queue[i] = queue2[j];

    queue[i] = 0;

    queue[0] = head;

    for (j = 0; j <= n + 1; j++)

    {

        diff = abs(queue[j + 1] - queue[j]);

        seek += diff;

        printf("Disk head moves from %d to %d with seek %d\n", queue[j],

        queue[j + 1], diff);

    }

    printf("Total seek time is %d\n", seek);

    avg = seek / (float)n;

    printf("Average seek time is %f\n", avg);

    return 0;

}

1. **C-SCAN DISK SCHEDULING**

#include <stdio.h>

#include <math.h>

int main()

{

    int queue[20], n, head, i, j, k, seek = 0, max, diff, temp, queue1[20],

    queue2[20], temp1 = 0, temp2 = 0;

    float avg;

    printf("Enter the max range of disk\n");

    scanf("%d", &max);

    printf("Enter the initial head position\n");

    scanf("%d", &head);

    printf("Enter the size of queue request\n");

    scanf("%d", &n);

    printf("Enter the queue of disk positions to be read\n");

    for (i = 1; i <= n; i++)

    {

        scanf("%d", &temp);

        if (temp >= head)

        {

            queue1[temp1] = temp;

            temp1++;

        }

        else

        {

            queue2[temp2] = temp;

            temp2++;

        }

    }

    for (i = 0; i < temp1 - 1; i++)

    {

        for (j = i + 1; j < temp1; j++)

        {

            if (queue1[i] > queue1[j])

            {

                temp = queue1[i];

                queue1[i] = queue1[j];

                queue1[j] = temp;

            }

        }

    }

    for (i = 0; i < temp2 - 1; i++)

    {

        for (j = i + 1; j < temp2; j++)

        {

            if (queue2[i] > queue2[j])

            {

                temp = queue2[i];

                queue2[i] = queue2[j];

                queue2[j] = temp;

            }

        }

    }

    for (i = 1, j = 0; j < temp1; i++, j++)

        queue[i] = queue1[j];

    queue[i] = max;

    queue[i + 1] = 0;

    for (i = temp1 + 3, j = 0; j < temp2; i++, j++)

        queue[i] = queue2[j];

    queue[0] = head;

    for (j = 0; j <= n + 1; j++)

    {

        diff = abs(queue[j + 1] - queue[j]);

        seek += diff;

        printf("Disk head moves from %d to %d with seek %d\n", queue[j],

        queue[j + 1], diff);

    }

    printf("Total seek time is %d\n", seek);

    avg = seek / (float)n;

    printf("Average seek time is %f\n", avg);

    return 0;

}