**EXP.NO:**

**DATE:**

**FIRST COME FIRST SERVE (FCFS) PROCESS SCHEDULING**

**AIM:**

The program to perform scheduling for the processes using First Come First Serve(FCFS) algorithm.

**ALGORITHM:**

1. Create the number of process.
2. Get the ID and Service time for each process.
3. Initially, waiting time of first process is zero and Total time for the first process is the starting time of that process.
4. Calculate the Total time and Processing time for the remaining processes.
5. Waiting time of one process is the Total time of the previous process.
6. Total time of process is calculated by adding Waiting time and Service time.
7. Total waiting time is calculated by adding the waiting time for lack process.
8. Total turnaround time is calculated by adding all total time of each process.
9. Calculate Average waiting time by dividing the total waiting time by total number of process.
10. Calculate Average turnaround time by dividing the total time by the number of process.
11. Display the result.

**PROGRAM:**

#include<stdio.h>  
int main()  
{

int bt[10]={0},wt[10]={0},ct[10]={0};  
float at[10]={0},tat[10]={0};  
int n,sum=0;  
float totalTAT=0,totalWT=0;  
printf("\_\_\_FCFS CPU SCHEDULING\_\_\_");  
printf("\n\nEnter number of process");  
scanf("%d",&n);  
printf("Enter arrival time and burst  time for each process\n\n");  
for(int i=0;i<n;i++)  
{  
printf("Arrival time of process[%d]",i+1);  
scanf("%f",&at[i]);  
printf("Burst time of process[%d] ",i+1);  
scanf("%d",&bt[i]);  
printf("\n");  
}  
for(int j=0;j<n;j++)  
{  
sum+=bt[j];  
ct[j]+=sum;  
}  
for( int k=0;k<n;k++)  
{  
tat[k]=ct[k]-at[k];  
totalTAT+=tat[k];  
}  
for(int k=0;k<n;k++)  
{  
wt[k]=tat[k]-bt[k];  
totalWT+=wt[k];  
}  
printf("Solution: \n\n");  
printf("P\t AT\t BT\t CT\t TAT\t WT\t\n\n");  
for(int i=0;i<n;i++)  
{  
printf("P%d\t %.2f\t %d\t %d\t %.2f\t %d\n",i+1,at[i],bt[i],ct[i],tat[i],wt[i]);  
}  
printf("\n\nAverage Turnaround time = %f\n",totalTAT/n);  
printf("Average WT = %f\n\n",totalWT/n);  
return 0;  
}

**COMMANDS:**

skcet@sk-mca-507:~$ gedit FCFS.c  
skcet@sk-mca-507:~$ cc FCFS.c  
skcet@sk-mca-507:~$ ./a.out

**INPUT:**

\_\_\_FCFS CPU SCHEDULING\_\_\_  
  
Enter number of process 5  
Enter arrival time and burst  time for each process  
  
Arrival time of process[1]0.0  
Burst time of process[1] 10  
  
Arrival time of process[2]1.1  
Burst time of process[2] 5  
  
Arrival time of process[3]3.1  
Burst time of process[3] 2  
  
Arrival time of process[4]5.1  
Burst time of process[4] 7  
Arrival time of process[5]7.1  
Burst time of process[5] 5  
  
**OUTPUT:**  
P AT BT CT TAT WT  
P1 0.00 10 10 10.00 0  
P2 1.10 5 15 13.90 8  
P3 3.10 2 17 13.90 11  
P4 5.10 7 24 18.90 11  
P5 7.10 5 29 21.90 16  
  
Average Turnaround time = 15.719999  
Average WT = 9.200000

**RESULT:**

Thus the C program to implement First Come First Serve (FCFS) scheduling algorithm was done and the turnaround time and waiting time was displayed.

**EXP.NO:**

**DATE:**

**SHORTEST JOB FIRST (SJF) PROCESS SCHEDULING**

**AIM:**

The program to perform scheduling for the processes using Shortest Job First(SJF) algorithm.

**ALGORITHM:**

1. Define an array of structure process with members pid, btime, wtime & ttime.
2. Get length of the ready queue, i.e., number of process (say n).
3. Obtain btime for each process.
4. Sort the processes according to their btime in ascending order.
   1. If two process have same btime, then FCFS is used to resolve the tie.
5. The wtime for first process is 0.
6. Compute wtime and ttime for each process as:
   1. wtime i+1= wtime i+ btime i b. ttime i = wtime i + btime i
7. Compute average waiting time awat and average turnaround time atur.
8. Display btime,ttime and wtime for each process.
9. Display GANTT chart for the above scheduling.
10. Display awat and atur.
11. Stop.

**PROGRAM:**

#include<stdio.h>  
int main()  
{  
int A[100][4];  
int i,j,n,total=0,index,temp;  
float avg\_wt,avg\_tat;  
printf("Enter number of process: ");  
scanf("%d",&n);  
printf("Enter Burst Time:\n");  
for(i=0;i<n;i++){  
printf("P%d: ",i+1);  
scanf("%d",&A[i][1]);  
A[i][0]=i+1;  
}  
for(i=0;i<n;i++){  
index=i;  
for(j=i+1;j<n;j++)  
if(A[j][1]<A[index][1]);  
index=j;  
temp=A[i][1];  
A[i][1]=A[index][1];  
A[index][1]=temp;  
temp=A[i][0];  
A[i][0]=A[index][0];  
A[index][0]=temp;  
}  
A[0][2]=0;  
for(i=1;i<n;i++){  
A[i][2]=0;  
for(j=0;j<i;j++)  
A[i][2]+=A[j][1];  
total+=A[i][2];  
}  
avg\_wt=(float)total/n;  
total=0;  
printf("P  BT  WT  TAT\n");  
for(i=0;i<n;i++){  
A[i][1],A[i][1]+A[i][3];  
total+=A[i][3];  
printf("P%d  %d  %d  %d\n",A[i][0],A[i][1],A[i][2],A[i][3]);  
}  
avg\_tat=(float)total/n;  
printf("Average Waiting Time=%f",avg\_wt);  
printf("\nAverage Turnaround Time=%f",avg\_tat);  
}

**COMMANDS:**

skcet@sk-mca-507:~$ gedit SJF.c  
skcet@sk-mca-507:~$ cc SJF.c  
skcet@sk-mca-507:~$ ./a.out

**INPUT:**  
Enter number of process: 4  
Enter Burst Time:  
P1: 8  
P2: 5  
P3: 10  
P4: 11

**OUTPUT:**

P  BT  WT  TAT  
P0  0  0  0  
P1  8  0  0  
P2  5  8  0  
P3  10  13  6  
Average Waiting Time=5.250000  
Average Turnaround Time=1.500000

**RESULT:**

Thus the C program to implement Shortest Job First (SJF) scheduling algorithm was done and the turnaround time and waiting time was displayed.

**EXP.NO:**

**DATE:**

**ROUND ROBIN (RR) PROCESS SCHEDULING**

**AIM:**

The program to perform scheduling for the processes using Round Robin (RR) algorithm.

**ALGORITHM:**

1. Initialize all the structure elements
2. Receive inputs from the user to fill process id,burst time and arrival time.
3. Calculate the waiting time for all the process id.
   1. The waiting time for first instance of a process is calculated as: a[i].waittime=count + a[i].arrivt
   2. The waiting time for the rest of the instances of the process is calculated as:
      1. If the time quantum is greater than the remaining burst time then waiting time is calculated as:

a[i].waittime=count + tq

* + - 1. Else if the time quantum is greater than the remaining burst time then waiting

time is calculated as:

a[i].waittime=count - remaining burst time

1. Calculate the average waiting time and average turnaround time
2. Print the results of the step 4.

**PROGRAM:**

#include<stdio.h>  
void main()  
{  
int i,NDP,sum=0,count=0,y,quant,wt=0,tat=0,at[10],bt[10],temp[10];  
float avg\_wt,avg\_tat;  
printf("Total number of process int the system: ");  
scanf("%d",&NDP);  
y=NDP;  
for(i=0;i<NDP;i++)  
{  
printf("\nEnter the Arrival and Burst time of the Process[%d]\n",i+1);  
printf("Arrival time is: \t");  
scanf("%d",&at[i]);  
printf("\nBurst time is :  \t");  
scanf("%d",&bt[i]);  
temp[i]=bt[i];      
}  
printf("Enter the Time Quantum for the process:\t");  
scanf("%d",&quant);  
printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting time ");  
for(sum=0,i=0;y!=0;){  
if(temp[i]<=quant&&temp[i]>0)  
{  
sum=sum+temp[i];  
temp[i]=0;  
count=1;  
}  
else if(temp[i]>0){  
temp[i]=temp[i]-quant;  
sum=sum+quant;  
}  
if(temp[i]==0&&count==1)  
{  
y--;  
printf("\nProcess NO[%d] \t\t %d\t\t\t\t %d\t\t\t %d",i+1,bt[i],sum-at[i],sum-at[i]-bt[i]);  
wt=wt+sum-at[i]-bt[i];  
tat=tat+sum-at[i];  
count=0;  
}  
if(i==NDP-1)  
i=0;  
else if(at[i+1]<=sum)  
i++;  
else  
i=0;  
}  
avg\_wt=wt\*1.0/NDP;  
avg\_tat=tat\*1.0/NDP;  
printf("\n Average Waiting Time: \t%f",avg\_wt);  
printf("\n Average Waiting Time: \t%f",avg\_tat);  
}

**COMMANDS:**

skcet@sk-mca-507:~$ gedit roundRobin.c  
skcet@sk-mca-507:~$ cc roundRobin.c  
skcet@sk-mca-507:~$ ./a.out

**INPUT:**

Total number of process int the system: 4  
  
Enter the Arrival and Burst time of the Process[1]  
Arrival time is: 0  
  
Burst time is :   8  
  
Enter the Arrival and Burst time of the Process[2]  
Arrival time is: 1  
  
Burst time is :   5  
  
Enter the Arrival and Burst time of the Process[3]  
Arrival time is: 2  
  
Burst time is :   10  
  
Enter the Arrival and Burst time of the Process[4]  
Arrival time is: 3  
  
Burst time is :   11  
Enter the Time Quantum for the process: 6

**OUTPUT:**

Process No Burst Time TAT Waiting time  
Process NO[2] 5 10 5  
Process NO[1] 8 25 17  
Process NO[3] 10 27 17  
Process NO[4] 11 31 20

Average Waiting Time: 14.750000  
Average Waiting Time: 23.250000

**RESULT:**

Thus the C program to implement Round Robin(RR) scheduling algorithm was done and the turnaround time and waiting time was displayed.

**EXP.NO:**

**DATE:**

**DINING PHILOSOPER ALGORITHM**

**AIM:**

To write a [C Program](http://sourcecodesonline.blogspot.com/search/label/C%20Programs) for the Implementation of Dining Philosopher Algorithm.

**ALGORITHM:**

1. Initialize the semaphores for each fork to 1 (indicating that they are available).

2. Initialize a binary semaphore (mutex) to 1 to ensure that only one philosopher can attempt to pick up a fork at a time.

3. For each philosopher process, create a separate thread that executes the following code:

* While true:
  + Think for a random amount of time.
  + Acquire the mutex semaphore to ensure that only one philosopher can attempt to pick up a fork at a time.
  + Attempt to acquire the semaphore for the fork to the left.
* If successful, attempt to acquire the semaphore for the fork to the right.
* If both forks are acquired successfully, eat for a random amount of time and then release both semaphores.
* If not successful in acquiring both forks, release the semaphore for the fork to the left (if acquired) and then release the mutex semaphore and go back to thinking.

4. Run the philosopher threads concurrently.

**PROGRAM:**

#include<stdio.h>

#define n 4

int compltedPhilo = 0,i;

struct fork{

int taken;

}ForkAvil[n];

struct philosp{

int left;

int right;

}Philostatus[n];

void goForDinner(int philID){

if(Philostatus[philID].left==10 && Philostatus[philID].right==10)

printf("Philosopher %d completed his dinner\n",philID+1);

else if(Philostatus[philID].left==1 && Philostatus[philID].right==1){

printf("Philosopher %d completed his dinner\n",philID+1);

Philostatus[philID].left = Philostatus[philID].right = 10;

int otherFork = philID-1;

if(otherFork== -1)

otherFork=(n-1);

ForkAvil[philID].taken = ForkAvil[otherFork].taken = 0;

printf("Philosopher %d released fork %d and fork %d\n",philID+1,philID+1,otherFork+1);

compltedPhilo++;

}

else if(Philostatus[philID].left==1 && Philostatus[philID].right==0){

if(philID==(n-1)){

if(ForkAvil[philID].taken==0){

ForkAvil[philID].taken = Philostatus[philID].right = 1;

printf("Fork %d taken by philosopher %d\n",philID+1,philID+1);

}else{

printf("Philosopher %d is waiting for fork %d\n",philID+1,philID+1);

}

}else{

int dupphilID = philID;

philID-=1;

if(philID== -1)

philID=(n-1);

if(ForkAvil[philID].taken == 0){

ForkAvil[philID].taken = Philostatus[dupphilID].right = 1;

printf("Fork %d taken by Philosopher %d\n",philID+1,dupphilID+1);

}else{

printf("Philosopher %d is waiting for Fork %d\n",dupphilID+1,philID+1);

}

}

}

else if(Philostatus[philID].left==0){

if(philID==(n-1)){

if(ForkAvil[philID-1].taken==0){

ForkAvil[philID-1].taken = Philostatus[philID].left = 1;

printf("Fork %d taken by philosopher %d\n",philID,philID+1);

}else{

printf("Philosopher %d is waiting for fork %d\n",philID+1,philID);

}

}else{

if(ForkAvil[philID].taken == 0){

ForkAvil[philID].taken = Philostatus[philID].left = 1;

printf("Fork %d taken by Philosopher %d\n",philID+1,philID+1);

}else{

printf("Philosopher %d is waiting for Fork %d\n",philID+1,philID+1);

}

}

}else{}

}

int main(){

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

ForkAvil[i].taken=Philostatus[i].left=Philostatus[i].right=0;

while(compltedPhilo<n){

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

goForDinner(i);

printf("\nTill now num of philosophers completed dinner are %d\n\n",compltedPhilo);

}

return 0;

}

**COMMANDS:**

skcet@sk-mca-507:~$ gedit roundRobin.c  
skcet@sk-mca-507:~$ cc dpp.c

skcet@sk-mca-507:~$ ./a.out

**OUTPUT:**

Fork 1 taken by Philosopher 1

Fork 2 taken by Philosopher 2

Fork 3 taken by Philosopher 3

Philosopher 4 is waiting for fork 3

Till now num of philosophers completed dinner are 0

Fork 4 taken by Philosopher 1

Philosopher 2 is waiting for Fork 1

Philosopher 3 is waiting for Fork 2

Philosopher 4 is waiting for fork 3

Till now num of philosophers completed dinner are 0

Philosopher 1 completed his dinner

Philosopher 1 released fork 1 and fork 4

Fork 1 taken by Philosopher 2

Philosopher 3 is waiting for Fork 2

Philosopher 4 is waiting for fork 3

Till now num of philosophers completed dinner are 1

Philosopher 1 completed his dinner

Philosopher 2 completed his dinner

Philosopher 2 released fork 2 and fork 1

Fork 2 taken by Philosopher 3

Philosopher 4 is waiting for fork 3

Till now num of philosophers completed dinner are 2

Philosopher 1 completed his dinner

Philosopher 2 completed his dinner

Philosopher 3 completed his dinner

Philosopher 3 released fork 3 and fork 2

Fork 3 taken by philosopher 4

Till now num of philosophers completed dinner are 3

Philosopher 1 completed his dinner

Philosopher 2 completed his dinner

Philosopher 3 completed his dinner

Fork 4 taken by philosopher 4

Till now num of philosophers completed dinner are 3

Philosopher 1 completed his dinner

Philosopher 2 completed his dinner

Philosopher 3 completed his dinner

Philosopher 4 completed his dinner

Philosopher 4 released fork 4 and fork 3

Till now num of philosophers completed dinner are 4

**RESULT:**

Thus the C program to implement the dining philosopher’s problem was implemented successfully.

**EXP.NO:**

**DATE:**

**BANKER’S ALGORTIHM**

**AIM:**

To implement of banker’s algorithm for deadlock avoidance using C.

**ALGORITHM:**

1. 1) Assume Work and Finish are two vectors, each with lengths of m and n.
2. Initialize: Work = Available
3. Finish[i] is false when i=1, 2, 3, 4...n
4. 2) Find an I such that both
5. a) Finish[i] = false
6. b) Needi <= Work
7. if such an I does not exist. goto  step (4)
8. 3) Work = Work + Allocation[i]
9. Finish[i] = true
10. go to step (2)
11. 4) If Finish[i] = true for each and every i
12. then system is in a secure state.
13. 1) Proceed to step 2 if Requesti >= Needi; otherwise, report an error condition because the process has made more claims than it can handle.
14. 2) Proceed to step (3) if Requesti <= Accessible; otherwise, Pi will have to wait since the resources are not available.
15. 3) Change the state in such a way that the system appears to have given Pi the required resources:
16. Requested - Available = Available
17. Allocationi = Allocationi + Requesti
18. Needi = Needi - Requesti

**PROGRAM:**

#include<stdio.h>  
int main()  
{  
int n,m,i,j,k;  
n=5;  
m=3;  
int alloc[5][3]={{0,1,0},{2,0,0},{3,0,2},{2,1,2},{0,0,2}};  
int max[5][3]={{7,5,3},{3,2,2},{9,0,2},{2,2,2},{4,3,3}};  
int avail[3]={3,3,2};  
int f[n],ans[n],ind=0;  
for(k=0;k<n;k++){  
f[k]=0;  
}  
int need[n][m];  
for(i=0;i<n;i++){  
for(j=0;j<m;j++){  
need[i][j]=max[i][j]-alloc[i][j];  
 }  
}  
int y=0;  
for(k=0;k<5;k++){  
for(i=0;i<n;i++){  
if(f[i]==0){  
int flag=0;  
for(j=0;j<m;j++){  
if(need[i][j]>avail[j]){  
flag=1;  
break;  
}  
}  
if(flag==0){  
ans[ind++]=i;  
for(y=0;y<m;y++)  
avail[y]+=alloc[i][y];  
f[i]=1;  
}  
}  
}  
}  
int flag=1;  
for(int i=0;i<n;i++){  
if(f[i]==0){  
flag=0;  
printf("The folowing system is not safe");  
break;  
}  
}  
if(flag==1){  
printf("Following is the SAFE Sequence\n");  
for(i=0;i<n;i++)  
printf(" P%d ->",ans[i]);  
printf(" P%d",ans[n-1]);  
}  
return 0;  
}

**COMMANDS:**

skcet@sk-mca-507:~$ gedit BA.c  
skcet@sk-mca-507:~$ cc BA.c  
skcet@sk-mca-507:~$ ./a.out

**OUTPUT:**

Following is the SAFE Sequence  
 P1 -> P3 -> P4 -> P0 -> P2

**RESULT:**

Thus the C program to implement the banker’s algorithm for deadlock avoidance was implemented successfully.

**EXP.NO:**

**DATE:**

**FIRST FIT MEMORY MANAGEMENT**

**AIM:**

The program to implement the First Fit memory management strategy.

**ALGORITHM:**

1. Input memory blocks with size and processes with size.
2. Initialize all memory blocks as free.
3. Start by picking each process and check if it can
4. be assigned to current block.
5. If size-of-process <= size-of-block if yes, then
6. assign and check for next process.
7. If not, then keep checking the further blocks.

**PROGRAM:**

#include<stdio.h>

#define max 25

void main(){

int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;

static int bf[max],ff[max];int flag,flagn[max],fragi = 0,fragx = 0;

printf("\n\_\_\_First Fit\_\_\_\n");

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

scanf("%d",&nb);

printf("Enter the number of Process:");

scanf("%d",&nf);

printf("\nEnter the size of the blocks:-\n");

for(i=1;i<=nb;i++) {

printf("Block %d:",i);

scanf("%d",&b[i]);

ff[i] = i;

}

printf("Enter the size of the Processes :-\n");

for(i=1;i<=nf;i++) {

printf("Process %d:",i);

scanf("%d",&f[i]);

}

int x = 1;

printf("\n\nProcess\_No\tProcess\_Size\tBlock\_No\tBlock\_Size\tFragment\n");

for(i=1;i<=nf;i++){

flag = 1;

for(j=x;j<=nb;j++){

if(f[i] <= b[j]){

flagn[j] = 1;

printf("%-15d\t%-15d\t%-15d\t%-15d\t",i, f[i],ff[j],b[j]);

b[j] = b[j] - f[i];

fragi = fragi + b[j];

printf("%-15d\n",b[j]);

break;

}

else{

flagn[j] = 0;

x = 1;

flag++;

} }

if(flag > nb)

printf("%-15d\t%-15d\t%-15s\t%-15s\t%-15s\n",i,f[i],"Has to wait...","...","...");

}}

**COMMANDS:**

skcet@sk-mca-507:~$ gedit firstfit.c  
skcet@sk-mca-507:~$ cc firstfit.c  
skcet@sk-mca-507:~$ ./a.out

**INPUT:**

\_\_First Fit\_\_  
  
Enter the number of blocks:5  
Enter the number of process:4

Enter the size of the blocks:-  
Block 1:100  
Block 2:500  
Block 3:200  
Block 4:300  
Block 5:600  
Enter the size of the processes :-  
Process 1:212  
Process 2:417  
Process 3:112  
Process 4:426

**OUTPUT:**

Process\_No Process\_Size Block\_No Block\_Size Fragment  
1               212             2               500             288              
2               417             5               600             183              
3               112             2               288             176              
4               426             Has to wait..   ...             ...

**RESULT:**

Thus the C program to implement the first fit memory management scheme is successful.

**EXP.NO:07**

**DATE:**

**BEST FIT MEMORY MANAGEMENT**

**AIM:**

The program to implement the Best Fit memory management strategy.

**ALGORITHM:**

1. Input memory blocks and processes with sizes.
2. Initialize all memory blocks as free.
3. Start by picking each process and find the minimum block size that can be assigned to current process i.e., find min (bock Size [1], block Size [2],.....block Size[n]) > process Size[current], if found then assign it to the current process.
4. If not, then leave that process and keep checking the further processes.

**PROGRAM:**

#include<stdio.h>

#define max 25

void main()

{

int frag[max],b[max],f[max],i,j,nb,nf,temp,lowest=10000;

static int bf[max],ff[max],fragi=0;

printf("\n\_\_\_Best Fit\_\_\_\n");

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

scanf("%d",&nb);

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

scanf("%d",&nf);

printf("\nEnter the size of the blocks:-\n");

for(i=1;i<=nb;i++){printf("Block %d:",i);

scanf("%d",&b[i]);

ff[i]=i;

}

printf("\nEnter the size of the processors :-\n");

for(i=1;i<=nf;i++){

printf("Process %d:",i);

scanf("%d",&f[i]);

}

int y,m,z,temp1,flag;

for(y=1;y<=nb;y++)

{

for(z=y;z<=nb;z++)

{

if(b[y]>b[z])

{

temp=b[y];

b[y]=b[z];

b[z]=temp;

temp1=ff[y];

ff[y]=ff[z];

ff[z]=temp1;

}

}

}

int flagn[max];

int fragx=0;

printf("\n\nProcess\_No\tProcess\_Size\tBlock\_No\tBlock\_Size\tFragment\n");

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

{

flag=1;

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

{

if(f[i]<=b[j]){

flagn[j]=1;

printf("%-15d\t%-15d\t%-15d\t%-15d\t",i,f[i],ff[j],b[j]);

b[j]=b[j]-f[i];

fragi=fragi+b[j];

printf("%-15d\n",b[j]);

break;

}

else

{

flagn[j]=0;

flag++;

}

}

if(flag>nb)

printf("%-15d\t%-15d\t%-15s\t%-15s\t%-15s\n",i,f[i],"Has to wait..","..","..");

}

}

**COMMANDS:**

skcet@sk-mca-507:~$ gedit bestfit.c  
skcet@sk-mca-507:~$ cc bestfit.c  
skcet@sk-mca-507:~$ ./a.out

**INPUT:**

\_\_\_Best Fit\_\_\_  
  
Enter the number of blocks:5  
  
Enter the number of files:4  
  
Enter the size of the blocks:-  
Block 1:100  
Block 2:500  
Block 3:200  
Block 4:300  
Block 5:600  
  
Enter the size of the processors :-  
Process 1:212  
Process 2:417  
Process 3:112  
Process 4:426

**OUTPUT:**

Process\_No Process\_Size Block\_No Block\_Size Fragment  
1               212             4               300             88              
2               417             2               500             83              
3               112             3               200             88              
4               426             5               600             174

**RESULT:**

Thus the C program to implement the best fit memory management scheme is successful.

**EXP.NO:**

**DATE:**

**WORST FIT MEMORY MANAGEMENT**

**AIM:**

The program to implement the Worst First memory management strategy.

**ALGORITHM:**

1. Input memory blocks and processes with sizes.
2. Initialize all memory blocks as free.
3. Start by picking each process and find the maximum block size that can be assigned to current process i.e., find max (bock Size [1], block Size [2],.....block Size[n]) > process Size[current], if found then assign it to the current process.
4. If not then leave that process and keep checking the further processes.

**PROGRAM:**

#include<stdio.h>  
#define max 25  
void main()  
{  
int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;  
static int bf[max],ff[max];  
int flag,fragi=0;  
printf("\n\_\_Worst Fit\_\_\n");  
printf("\nEnter the number of memory blocks:");  
scanf("%d",&nb);  
printf("Enter the number of Process:");  
scanf("%d",&nf);  
printf("\nEnter the size of the memory blocks:\n");  
for(int i=1;i<=nb;i++)  
{  
printf("Block %d: ",i);  
scanf("%d",&b[i]);  
ff[i]=i;  
}  
printf("Enter the size of the processes :\n");  
for(i=1;i<=nf;i++)  
{  
printf("Process %d: ",i);  
scanf("%d",&f[i]);  
}  
int y,z,temp1;  
for(y=1;y<=nb;y++)  
{  
for(z=y;z<=nb;z++)  
{  
if(b[y]<b[z])  
{  
temp=b[y];  
b[y]=b[z];  
b[z]=temp;  
temp1=ff[y];  
ff[y]=ff[z];  
ff[z]=temp1;  
}  
}  
}  
int flagn[max];  
int fragx=0;  
printf("\n\nProcess No\tProcess Size\tMemory No\tMemory Size\tRemaining\n");  
for(i=1;i<=nf;i++)  
{  
flag=1;  
for(int j=1;j<=nb;j++)  
{  
if(f[i]<=b[j])  
{flagn[j]=1;  
printf("%-15d\t%-15d\t%-15d\t%-15d\t",i,f[i],ff[j],b[j]);  
b[j]=b[j]-f[i];  
fragi=fragi+b[j];  
printf("%-15d\n",b[j]);  
break;}  
else  
{flagn[j]=0;  
flag++;  
}  
}  
if(flag>nb)  
printf("%-15d\t%-15d\t%-15s\t%-15s\t%-15s\n",i,f[i],"Has to wait..","..","..");}}

**COMMANDS:**

skcet@sk-mca-507:~$ gedit \*worstfit.c  
skcet@sk-mca-507:~$ cc \*worstfit.c  
skcet@sk-mca-507:~$ ./a.out

**INPUT:**

\_\_Worst Fit\_\_  
  
Enter the number of memory blocks:5  
Enter the number of Process:4  
  
Enter the size of the memory blocks:  
Block 1: 100  
Block 2: 500  
Block 3: 200  
Block 4: 300  
Block 5: 600  
Enter the size of the processes :  
Process 1: 212  
Process 2: 417  
Process 3: 112  
Process 4: 426

**OUTPUT:**

Process No Process Size Memory No Memory Size Remaining  
1               212             5               600             388              
2               417             2               500             83              
3               112             5               388             276              
4               426             Has to wait..   ..             ..

**RESULT**:

Thus the C program to implement the worst fit memory management scheme is successful.

**EXP.NO:**

**DATE:**

**FIRST IN FIRST OUT(FIFO) PAGE REPLACEMENT**

**AIM:**

The program to implement first in first out page replacement for replacing the pages in the virtual memory and calculating the number of page faults.

**ALGORITHM:**

1. Start the program
2. Read the block size and compare with the pages in the block
3. When matches found in the block set hit to one, when the page is not present in the block increment page fault count by one
4. Store the page in the block by getting index of the block through mod function
5. Display the number of page fault count, after reading new page the contents of the block will be displayed
6. End the process.

**PROGRAM:**

#include<stdio.h>  
int main()  
{  
    int incomingStream[] = {4, 1, 2, 4, 5};  
    int pageFaults = 0;  
    int frames = 3;  
    int m, n, s, pages;  
  
    pages = sizeof(incomingStream)/sizeof(incomingStream[0]);  
  
    printf("Incoming \t Frame 1 \t Frame 2 \t Frame 3");  
    int temp[frames];  
    for(m = 0; m < frames; m++)  
    {  
        temp[m] = -1;  
    }  
  
    for(m = 0; m < pages; m++)  
    {  
        s = 0;  
  
        for(n = 0; n < frames; n++)  
        {  
            if(incomingStream[m] == temp[n])  
            {  
                s++;  
                pageFaults--;  
            }  
        }  
        pageFaults++;  
         
        if((pageFaults <= frames) && (s == 0))  
        {  
            temp[m] = incomingStream[m];  
        }  
        else if(s == 0)  
        {  
            temp[(pageFaults - 1) % frames] = incomingStream[m];  
        }  
       
        printf("\n");  
        printf("%d\t\t\t",incomingStream[m]);  
        for(n = 0; n < frames; n++)  
        {  
            if(temp[n] != -1)  
                printf(" %d\t\t\t", temp[n]);  
            else  
                printf(" - \t\t\t");  
        }  
    }  
    printf("\nTotal Page Faults:\t%d\n", pageFaults);  
    return 0;  
}

**COMMANDS:**

skcet@sk-mca-507:~$ gedit fifo.c  
skcet@sk-mca-507:~$ cc fifo.c  
skcet@sk-mca-507:~$ ./a.out

**OUTPUT:**

Incoming Frame 1 Frame 2 Frame 3  
4 4 - -  
1 4 1 -  
2 4 1 2  
4 4 1 2  
5 5 1 2  
Total Page Faults: 4

**RESULT:**

Thus the C program to implement FIFO page replacement was implemented successfully and the output was verified.

**EXP.NO:**

**DATE:**

**LEAST RECENTLY USED(LRU) PAGE REPLACEMENT**

**AIM:**

The program to implement Least Recently Used page replacement for replacing the pages in the virtual memory and calculating the number of page faults.

**ALGORITHM:**

1. Start the process
2. Declare the size
3. Get the number of pages to be inserted
4. Get the value
5. Declare counter and stack
6. Select the least recently used page by counter value
7. Stack them according the selection.
8. Display the values
9. Stop the process

**PROGRAM:**

#include<stdio.h>  
int main()  
{  
int q[20],p[50],c=0,c1,d,f,i,j,k=0,n,r,t,b[20],c2[20];  
printf("Enter no of pages:");  
scanf("%d",&n);  
printf("Enter the reference string:");  
for(i=0;i<n;i++)  
            scanf("%d",&p[i]);  
printf("Enter no of frames:");  
scanf("%d",&f);  
q[k]=p[k];  
printf("\n\t%d\n",q[k]);  
c++;  
k++;  
for(i=1;i<n;i++)  
            {  
              c1=0;  
                  for(j=0;j<f;j++)  
                  {  
                       if(p[i]!=q[j])  
                            c1++;  
                  }  
                  if(c1==f)  
                  {  
                        c++;  
                        if(k<f)  
                        {  
                               q[k]=p[i];  
                               k++;  
                               for(j=0;j<k;j++)  
                               printf("\t%d",q[j]);  
                               printf("\n");  
                         }  
                         else  
                         {  
                               for(r=0;r<f;r++)  
                               {  
                                      c2[r]=0;  
                                      for(j=i-1;j<n;j--)  
                                      {  
                                        if(q[r]!=p[j])  
                                        c2[r]++;  
                                        else  
                                        break;  
                               }  
                         }  
                         for(r=0;r<f;r++)  
                         b[r]=c2[r];  
                         for(r=0;r<f;r++)  
                         {  
                            for(j=r;j<f;j++)  
                            {  
                              if(b[r]<b[j])  
                              {  
                                 t=b[r];  
                                 b[r]=b[j];  
                                 b[j]=t;  
                              }  
                            }  
                         }  
                         for(r=0;r<f;r++)  
                         {  
                           if(c2[r]==b[0])  
                            q[r]=p[i];  
                           printf("\t%d",q[r]);  
                         }  
                         printf("\n");  
                    }  
            }  
 }  
  printf("\nThe no of page faults is %d",c);  
 }

**COMMANDS:**

skcet@sk-mca-507:~$ gedit lru.c  
skcet@sk-mca-507:~$ cc lru.c  
skcet@sk-mca-507:~$ ./a.out

**INPUT:**

Enter no of pages:10  
Enter the reference string:7 5 9 4 3 7 9 6 2 1  
Enter no of frames:4

**OUTPUT:**

7  
7 5  
7 5 9  
7 5 9 4  
3 5 9 4  
3 7 9 4  
3 7 9 6  
2 7 9 6  
2 1 9 6  
  
The no of page faults is 9

**RESULT:**

Thus the C program to implement LRU page replacement was implemented successfully and the output was verified.

**EXP.NO:**

**DATE:**

**OPTIMAL PAGE REPLACEMENT ALGORITHM**

**AIM:**

The program to implement optimal page replacement for replacing the pages in the virtual memory and calculating the number of page faults.

**ALGORITHM:**

1. If referred page is already present, increment hit count.
2. If not present, find if a page that is never referenced in future.
3. If such a page exists, replace this page with new page.
4. If no such page exists, find a page that is referenced farthest in future.
5. Replace this page with new page.

**PROGRAM:**

#include<stdio.h>  
int main()  
{  
    int no\_of\_frames, no\_of\_pages, frames[10], pages[30], temp[10], flag1, flag2, flag3, i, j, k, pos, max, faults = 0;  
  
    printf("Enter number of frames: ");  
    scanf("%d", &no\_of\_frames);  
      
    printf("Enter number of pages: ");  
    scanf("%d", &no\_of\_pages);  
     
    printf("Enter page 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]){  
                   flag1 = flag2 = 1;  
                   break;  
               }  
        }  
         
        if(flag1 == 0){  
            for(j = 0; j < no\_of\_frames; ++j){  
                if(frames[j] == -1){  
                    faults++;  
                    frames[j] = pages[i];  
                    flag2 = 1;  
                    break;  
                }  
            }      
        }  
         
        if(flag2 == 0){  
         flag3 =0;  
         
            for(j = 0; j < no\_of\_frames; ++j){  
             temp[j] = -1;  
             
             for(k = i + 1; k < no\_of\_pages; ++k){  
             if(frames[j] == pages[k]){  
             temp[j] = k;  
             break;  
             }  
             }  
            }  
             
            for(j = 0; j < no\_of\_frames; ++j){  
             if(temp[j] == -1){  
             pos = j;  
             flag3 = 1;  
             break;  
             }  
            }  
             
            if(flag3 ==0){  
             max = temp[0];  
             pos = 0;  
             
             for(j = 1; j < no\_of\_frames; ++j){  
             if(temp[j] > max){  
             max = temp[j];  
             pos = j;  
             }  
             }              
            }  
frames[pos] = pages[i];  
faults++;  
        }  
         
        printf("\n");  
         
        for(j = 0; j < no\_of\_frames; ++j){  
            printf("%d\t", frames[j]);  
        }  
    }  
      printf("\n\nTotal Page Faults = %d", faults);  
       return 0;  
}

**COMMANDS:**

skcet@sk-mca-507:~$ gedit optimal.c  
skcet@sk-mca-507:~$ cc optimal.c  
skcet@sk-mca-507:~$ ./a.out

**INPUT:**

Enter number of frames: 3  
Enter number of pages: 10  
Enter page reference string: 1 2 3 4 3 2 1 1 4 3

**OUTPUT:**

1 -1 -1  
1 2 -1  
1 2 3  
4 2 3  
4 2 3  
4 2 3  
4 1 3  
4 1 3  
4 1 3  
4 1 3  
  
Total Page Faults = 5

**RESULT:**

Thus the C program to implement LRU page replacement was implemented successfully and the output was verified.