**PRACTICAL 13**

**OBJECTIVE: WRITE A PROGRAM TO IMPLEMENT FIRST-FIT , BEST-FIT AND WORST-FIT ALLOCATION STRATEGIES.**

**SOL:**

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

//function to enter values in array

void accept(int a[],int n)

{

int i;

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

{

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

}

}

//function to display array

void display(int a[],int n)

{

int i; printf("\n\n");

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

{

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

}

}

//function to sort given array

void sort(int a[],int n)

{

int i,j,temp;

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

{

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

{

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

{

temp=a[j];

a[j]=a[j+1];

a[j+1]=temp;

}

}

}

}

//reverse sort

void revsort(int a[],int n)

{

int i,j,temp;

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

{

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

{

if(a[j]<a[j+1])

{

temp=a[j];

a[j]=a[j+1];

a[j+1]=temp;

}

}

}

}

// first fit algo

void first\_fit(int psize[],int np,int msize[],int nm)

{

int i,j,in\_fr,ex\_fr,flag[30]={0}; // in\_fr : internal fragmebtation, ex\_fr : external

// fragmentation

in\_fr=ex\_fr=0;

for(i=0;i<np;i++) //loop to check space for process

{

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

{

/\*

\* the first fit option finds the "first largest hole" for a process that is given, hence if

\* process size is smaller than the memory hole size, just place it and move to next process

block.

\*/

if(flag[j]==0 && msize[j]>=psize[i]) //initially flag[] = NULL or 0, after every

// condition fullfilment flag[j]=1

{

flag[j]=1;

in\_fr=in\_fr+msize[j]-psize[i]; //calculate internal fragmentation (left

// over space after placing process) break;

}

}

if(j==nm) //upper loop breaks if there is no space for a particular process

printf("\n\nTHERE IS NO SPACE FOR PROCESS %d ",i);

}

for(i=0;i<nm;i++) //calculate external fragmenation, its is noting but the left over memory

// hole in the memory block.

{

if(flag[i]==0) ex\_fr=ex\_fr+msize[i];

}

printf("\n\nPROCESSES::"); display(psize,np); printf("\n\nMEMORY HOLES::"); display(msize,nm);

printf("\n\nTOTAL SUM OF INTERNAL FRAGMENTATION = %d ",in\_fr); printf("\n\nTOTAL SUM OF EXTERNAL FRAGMENTATION = %d ",ex\_fr);

}

void best\_fit(int psize[],int np,int msize[],int nm)

{

int i,j,in\_fr,ex\_fr,temp[30],flag[30]={0};

in\_fr=ex\_fr=0;

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

temp[i]=msize[i];

/\*

\* since we want to have best fit, sort the memory block, once

\* sorted we can start placing the process where we first find space for it.

\*/sort(temp,nm);

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

{

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

{

/\*

\* the best fit option uses a sorted array of memory holes for a process that is given,

hence if

\* process size is smaller than the memory hole size, just place it and move to next process

block.

\*/

if(flag[j]==0 && temp[j]>=psize[i])

{

flag[j]=1;

in\_fr=in\_fr+temp[j]-psize[i];

break;

}

}

if(j==nm) //rest same as first fit

printf("\n\nTHERE IS NO SPACE FOR PROCESS %d ",i);

}

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

{

if(flag[i]==0)

ex\_fr=ex\_fr+temp[i];

}

printf("\n\nPROCESSES::");

display(psize,np);

printf("\n\nMEMORY HOLES::");

display(temp,nm);

printf("\n\nTOTAL SUM OF INTERNAL FRAGMENTATION = %d ",in\_fr);

printf("\n\nTOTAL SUM OF EXTERNAL FRAGMENTATION = %d ",ex\_fr);

}

void worst\_fit(int psize[],int np,int msize[],int nm)

{

int i,j,in\_fr,ex\_fr,temp[30],flag[30]={0};

in\_fr=ex\_fr=0;

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

temp[i]=msize[i];

/\*

\* worst fit finds the biggest possible memory hole for the coming process,

\* hence let’s sort the memory block in reverse order. Now if 1st process comes,

\* it will get the biggest possible(first memory hole in reversed array) memory hole

\* and this will go on.

\*/ revsort(temp,nm);

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

{

/\* The array is sorted, so just start placing ;) \*/

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

{

if(flag[j]==0 && temp[j]>=psize[i])

{

flag[j]=1;

in\_fr=in\_fr+temp[j]-psize[i];

break;

}

}

if(j==nm) //rest same as first fit

printf("\n\nTHERE IS NO SPACE FOR PROCESS %d ",i);

}

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

{

if(flag[i]==0)

ex\_fr=ex\_fr+temp[i];

}

printf("\n\nPROCESSES::");

display(psize,np);

printf("\n\nMEMORY HOLES::");

display(temp,nm);

printf("\n\nTOTAL SUM OF INTERNAL FRAGMENTATION = %d ",in\_fr);

printf("\n\nTOTAL SUM OF EXTERNAL FRAGMENTATION = %d ",ex\_fr);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main()

{

int ch,np,nm,psize[30],msize[30];

printf("\nENTER NO OF PROCESSES::");

scanf("%d",&np);

printf("\n\nENTER SIZES OF PROCESSES::");

accept(psize,np);

printf("\nENTER NO MEMORY HOLES::");

scanf("%d",&nm);

printf("\n\nENTER SIZES OF MEMORY HOLES::");

accept(msize,nm); while(1)

{

printf("\n\n\t\t\*\*MAIN MENU\*\*");

printf("\n\n\tMEMORY MANAGEMENT");

printf("\n\n\t1.FIRST FIT");

printf("\n\n\t2.BEST FIT");

printf("\n\n\t3.WORST FIT");

printf("\n\n\t4.QUIT");

printf("\n\nENTER YOUR CHOICE::");

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("\n\nFIRST FIT::\n");

first\_fit(psize,np,msize,nm);

break;

case 2:

printf("\n\n\tBEST FIT::\n");

best\_fit(psize,np,msize,nm);

break;

case 3:

printf("\n\n\tWORST FIT::\n");

worst\_fit(psize,np,msize,nm);

break;

case 4:

exit(0);

default:

printf("\n\nPLEASE ENTER CORRECT CHOICE!!");

}

}

return 0;

}

OUTPUT:





