**Q1 W AP to simulate the MFT memory management techniques**

#include<stdio.h>

main()

{

int ms, bs, nob, ef,n, mp[10],tif=0;

int i, p=0;

clrscr();

printf("Enter the total memory available (in Bytes) -- ");

scanf("%d",&ms);

printf("Enter the block size (in Bytes) -- ");

scanf("%d", &bs);

nob=ms/bs; ef=ms - nob\*bs;

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

scanf("%d",&n);

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

{

printf("Enter memory required for process %d (in Bytes)-- ",i+1);

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

}

printf("\nNo. of Blocks available in memory -- %d",nob);

printf("\n\nPROCESS\tMEMORY REQUIRED\t ALLOCATED\tINTERNAL FRAGMENTATION");

for(i=0;i<n && p<nob;i++)

{

printf("\n %d\t\t%d",i+1,mp[i]); if(mp[i] > bs)

printf("\t\tNO\t\t---");

else

{ printf("\t\tYES\t%d",bs-mp[i]);

tif = tif + bs-mp[i];

p++;

}

}

if(i<n)

printf("\nMemory is Full, Remaining Processes cannot be accomodated");

printf("\n\nTotal Internal Fragmentation is %d",tif);

printf("\nTotal External Fragmentation is %d",ef);

}

***INPUT***

Enter the total memory available (in Bytes) -- 1000

Enter the block size (in Bytes)-- 300

Enter the number of processes – 5

Enter memory required for process 1 (in Bytes) -- 275

Enter memory required for process 2 (in Bytes) – 400

Enter memory required for process 3 (in Bytes) – 290

Enter memory required for process 4 (in Bytes) – 293

Enter memory required for process 5 (in Bytes) -- 100

No. of Blocks available in memory -- 3

***OUTPUT***

PROCESS MEMORY REQUIRED ALLOCATED INTERNAL FRAGMENTATION

1 275 YES 25

2 400 NO 10

3 290 YES …..

4 293 YES 7

Memory is Full, Remaining Processes cannot be accommodated

Total Internal Fragmentation is 42

Total External Fragmentation is 100

Q2 W AP to simulate the MVT memory management techniques

#include<stdio.h>

main()

{

int ms,mp[10],i,

temp,n=0; char ch = 'y';

clrscr();

printf("\nEnter the total memory available (in Bytes)-- ");

scanf("%d",&ms);

temp=ms; for(i=0;ch=='y';i++,n++)

{

printf("\nEnter memory required for process %d (in Bytes) -- ",i+1);

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

if(mp[i]<=temp)

{

printf("\nMemory is allocated for Process %d ",i+1);

temp = temp - mp[i];

}

else

{

printf("\nMemory is Full"); break;

}

printf("\nDo you want to continue(y/n) -- ");

scanf(" %c", &ch);

}

printf("\n\nTotal Memory Available -- %d", ms);

printf("\n\n\tPROCESS\t\t MEMORY ALLOCATED ");

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

printf("\n \t%d\t\t%d",i+1,mp[i]);

printf("\n\nTotal Memory Allocated is %d",ms-temp);

printf("\nTotal External Fragmentation is %d",temp);

}

***INPUT***

Enter the total memory available (in Bytes) -- 1000

Enter memory required for process 1 (in Bytes) -- 400

Memory is allocated for Process 1

Do you want to continue(y/n) -- y

Enter memory required for process 2 (in Bytes) -- 275

Memory is allocated for Process 2

Do you want to continue(y/n) -- y

Enter memory required for process 3 (in Bytes) – 550

***OUTPUT***

Memory is Full

Total Memory Available – 1000

PROCESS MEMORY ALLOCATED

1. 400
2. 275

Total Memory Allocated is 675

Total External Fragmentation is 325

**Q3 2. Write a C program to simulate the following contiguous memory allocation techniques a)**

**Worst-fit b) Best-fit c) First-fit**

worst fit:

#include<stdio.h>

#define max

void main()

{ int frag[max],b[max],f[max],i,j,nb,nf,temp;

static int bf[max],ff[max];

clrscr();

printf("\n\tMemory Management Scheme - First Fit");

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

scanf("%d",&nb);

printf("Enter 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]);

}

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

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

{

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

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

}

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

{

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

{ if(bf[j]!=1)

{ temp=b[j]-f[i]; if(temp>=0)

{ ff[i]=j; break;

}

}

}

frag[i]=temp;

bf[ff[i]]=1;

}

printf("\nFile\_no:\tFile\_size :\tBlock\_no:\tBlock\_size:\tFragement");

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

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);

}

***INPUT***

Enter the number of blocks: 3

Enter the number of files: 2

Enter the size of the blocks:-

Block 1: 5

Block 2: 2

Block 3: 7

Enter the size of the files:-

File 1: 1

File 2: 4

***OUTPUT***

File No File Size Block No Block Size Fragment

1 1 1 5 4

2 4 3 7 3

**BEST-FIT**

#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];

clrscr();

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

scanf("%d",&nb);

printf("Enter 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]);

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

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

{

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

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

}

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

{

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

{

if(bf[j]!=1)

{

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

if(temp>=0)

if(lowest>temp)

{

ff[i]=j;

lowest=temp;

}

}

}

frag[i]=lowest;

bf[ff[i]]=1;

lowest=10000;

}

printf("\nFile No\tFile Size \tBlock No\tBlock Size\tFragment");

for(i=1;i<=nf && ff[i]!=0;i++)

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);

}

INPUT

Enter the number of blocks: 3

Enter the number of files: 2

Enter the size of the blocks:-

Block 1: 5

Block 2: 2

Block 3: 7

Enter the size of the files:-

File 1: 1

File 2: 4

OUTPUT

File No File Size Block No Block Size

1 1 2 2

2 4 1 5

**FIRST-FIT**

#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];

clrscr();

printf("\n\tMemory Management Scheme - Worst Fit");

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

scanf("%d",&nb);

printf("Enter 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]);

}

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

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

{

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

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

}

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

{

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

{

if(bf[j]!=1) //if bf[j] is not allocated

{

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

if(temp>=0)

if(highest<temp)

{

ff[i]=j;

highest=temp;

}

}

}

frag[i]=highest;

bf[ff[i]]=1;

highest=0;

}

printf("\nFile\_no:\tFile\_size :\tBlock\_no:\tBlock\_size:\tFragement");

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

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);

}

INPUT

Enter the number of blocks: 3

Enter the number of files: 2

Enter the size of the blocks:-

Block 1: 5

Block 2: 2

Block 3: 7

Enter the size of the files:-

File 1: 1

File 2: 4

OUTPUT

File No File Size Block No Block Size Fragment

1 1 3 7 6

2 4 1 5 1

**4. Write a C program to simulate paging technique of memory management.**

#include<stdio.h>

main()

{

int ms, ps, nop, np, rempages, i, j, x, y, pa, offset;

int s[10], fno[10][20];

clrscr();

printf("\nEnter the memory size -- ");

scanf("%d",&ms);

printf("\nEnter the page size -- ");

scanf("%d",&ps);

nop = ms/ps;

printf("\nThe no. of pages available in memory are -- %d ",nop);

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

scanf("%d",&np);

rempages = nop;

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

{

printf("\nEnter no. of pages required for p[%d]-- ",i);

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

if(s[i] >rempages)

{

printf("\nMemory is Full");

break;

}

rempages = rempages - s[i];

printf("\nEnter pagetable for p[%d] --- ",i);

for(j=0;j<s[i];j++)

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

}

printf("\nEnter Logical Address to find Physical Address ");

printf("\nEnter process no. and pagenumber and offset -- ");

scanf("%d %d %d",&x,&y, &offset);

if(x>np || y>=s[i] || offset>=ps)

printf("\nInvalid Process or Page Number or offset");

else

{

pa=fno[x][y]\*ps+offset;

printf("\nThe Physical Address is -- %d",pa);

}

}

INPUT

Enter the memory size – 1000

Enter the page size --100

The no. of pages available in memory are -- 10

Enter number of processes --3

Enter no. of pages required for p[1] --4

Enter pagetable for p[1] ---8 6 9 5

Enter no. of pages required for p[2] -- 5

Enter pagetable for p[2] --- 1 5 4 5 7 3

Enter no. of pages required for p[3] -- 5

OUTPUT

Memory is Full

Enter Logical Address to find Physical Address

Enter process no. and pagenumber and offset – 2 3 60

The Physical Address is -- 760