

Operating System

**LAB-11**

**Name \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Roll No \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Date \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_­\_**

**Marks Obtained \_\_\_\_\_\_\_\_\_\_\_\_**

**Signature\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Memory Management Techniques

**Objective:**

Write a C program to simulate the MVT and MFT memory management techniques.

Write a C program to simulate the following contiguous memory allocation techniques

* Worst-fit
* Best-fit
* First-fit

**TASK1:**

**DESCRIPTION**

MFT (Multiprogramming with a Fixed number of Tasks) is one of the old memory management techniques in which the memory is partitioned into fixed size partitions and each job is assigned to a partition. The memory assigned to a partition does not change. MVT (Multiprogramming with a Variable number of Tasks) is the memory management technique in which each job gets just the amount of memory it needs. That is, the partitioning of memory is dynamic and changes as jobs enter and leave the system. MVT is a more ``efficient'' user of resources. MFT suffers with the problem of internal fragmentation and MVT suffers with external fragmentation.

**PROGRAM**

**MFT MEMORY MANAGEMENT TECHNIQUE**

#include<stdio.h>

#include<conio.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); getch();

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| } |  |  |  |  |  |
| ***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 | ----- |
| 3 | 290 |  |  | YES | 10 |
| 4 | 293 |  |  | YES | 7 |
| Memory is Full, Remaining Processes cannot be accommodated | | | | |  |
| Total Internal Fragmentation is | | 42 |  |  |  |
| Total External Fragmentation is | | 100 |  |  |  |

**MVT MEMORY MANAGEMENT TECHNIQUE**

#include<stdio.h>

#include<conio.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);

getch();

|  |  |  |  |
| --- | --- | --- | --- |
| } |  |  |  |
| ***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 |  |

**TASK2:**

**DESCRIPTION**

One of the simplest methods for memory allocation is to divide memory into several fixed-sized partitions. Each partition may contain exactly one process. In this multiple-partition method, when a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process. The operating system keeps a table indicating which parts of memory are available and which are occupied. Finally, when a process arrives and needs memory, a memory section large enough for this process is provided. When it is time to load or swap a process into main memory, and if there is more than one free block of memory of sufficient size, then the operating system must decide which free block to allocate. Best-fit strategy chooses the block that is closest in size to the request. First-fit chooses the first available block that is large enough. Worst-fit chooses the largest available block.

**PROGRAM**

**FIRST-FIT**

#include<stdio.h>

#include<conio.h> #define max 25

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]);

getch();

}

***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>

#include<conio.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]);

getch();

}

***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 | 2 | 2 | 1 |
| 2 | 4 | 1 | 5 | 1 |

Lab Task:

* You are required to make changes in the above programs and introduce the use of compaction where required.
* Write code to simulate Worst-Fit Algorithm