# Lab Session 9: Memory Allocation Algorithms

**11.1 Introduction**

In the operating system, the following are four common memory management techniques.

**Single contiguous allocation:** Simplest allocation method used by MS-DOS. All memory (except some reserved for OS) is available to a process.

**Partitioned allocation:** Memory is divided in different blocks or partitions. Each process is

allocated according to the requirement.

**Paged memory management**: Memory is divided into fixed sized units called page frames, used in a virtual memory environment.

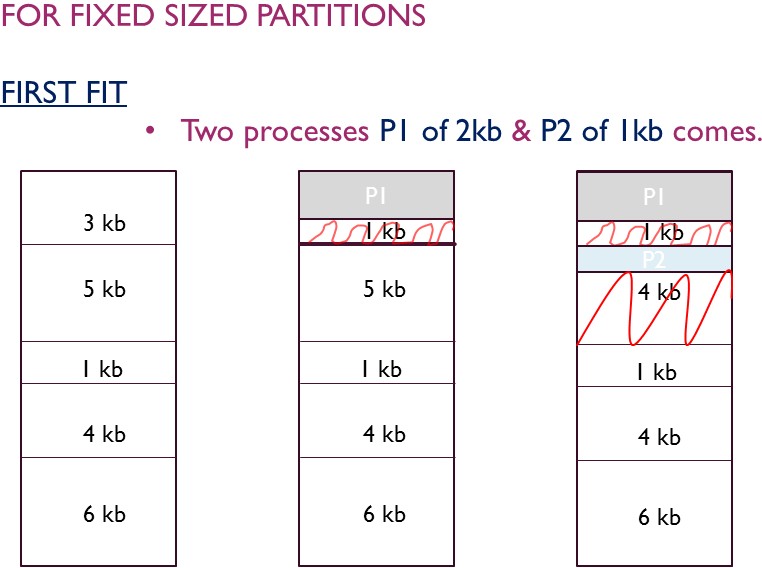
**Segmented memory management**: Memory is divided in different segments (a segment is a

logical grouping of the process’ data or code). In this management, allocated memory doesn’t have to be contiguous.

Most of the operating systems (for example Windows and Linux) use Segmentation with Paging. A process is divided into segments and individual segments have pages.

In Partition Allocation, when there is more than one partition freely available to accommodate a process’s request, a partition must be selected. To choose a particular partition, a partition allocation method is needed. A partition allocation method is considered better if it avoids internal fragmentation.

Below are the various partition allocation schemes:

* 1. ** First Fit**: In the first fit, the partition is allocated which is first sufficient block from the top of Main Memory.

Example:

//C program to implement First Fit Algorithms

#include<stdio.h>

void main()

{

int bsize[10], psize[10], bno, pno, flags[10], allocation[10], i, j;

for(i = 0; i < 10; i++)

{

flags[i] = 0;

allocation[i] = -1;

}

printf("Enter no. of blocks: "); scanf("%d", &bno);

printf("\nEnter size of each block: "); for(i = 0; i < bno; i++)

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

printf("\nEnter no. of processes: "); scanf("%d", &pno);

printf("\nEnter size of each process: "); for(i = 0; i < pno; i++)

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

for(i = 0; i < pno; i++) //allocation as per first fit for(j = 0; j < bno; j++)

if(flags[j] == 0 && bsize[j] >= psize[i])

{

allocation[j] = i; flags[j] = 1; break;

}

//display allocation details

printf("\nBlock no.\tsize\t\tprocess no.\t\tsize"); for(i = 0; i < bno; i++)

{

printf("\n%d\t\t%d\t\t", i+1, bsize[i]); if(flags[i] == 1)

printf("%d\t\t\t%d",allocation[i]+1,psize[allocation[i]]);

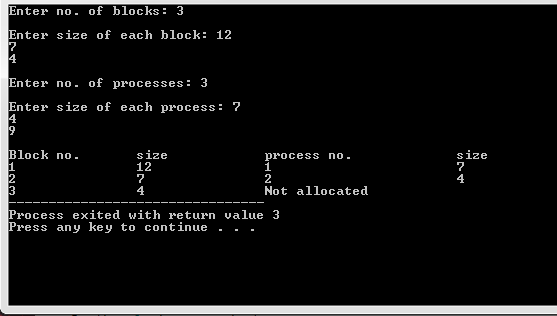
else

}

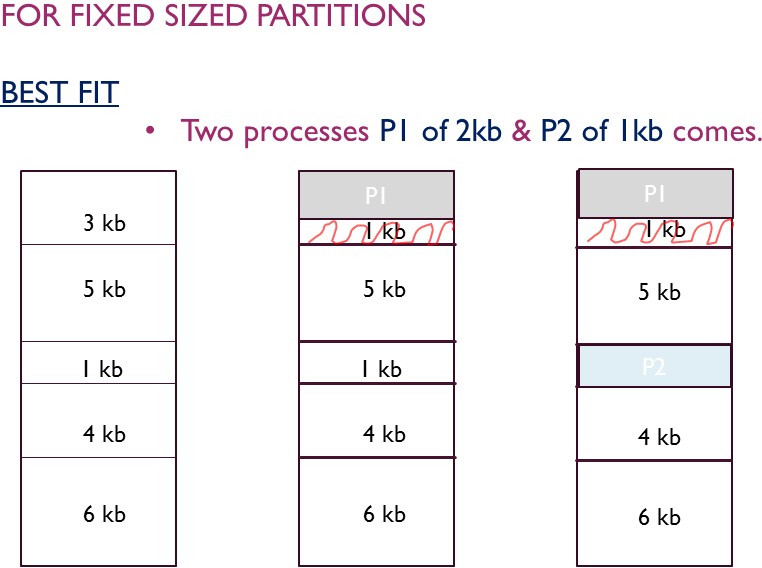
}

printf("Not allocated");}

Output:



* 1. **Best Fit** Allocate the process to the partition which is the first smallest sufficient partition among the free available partition.



**Example:**

#include<stdioh>

void main()

{

int fragment[20],b[20],p[20],i,j,nb,np,temp,lowest=9999;

static int barray[20],parray[20];

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

printf("\nEnter the number of blocks:"); scanf("%d",&nb);

printf("Enter the number of processes:"); scanf("%d",&np);

printf("\nEnter the size of the blocks:-\n"); for(i=1;i<=nb;i++)

printf("Block no.%d:",i);

{

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

}

printf("\nEnter the size of the processes :-\n"); for(i=1;i<=np;i++)

{

printf("Process no.%d:",i);

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

}

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

{

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

{

if(barray[j]!=1)

{

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

if(lowest>temp)

{

parray[i]=j; lowest=temp;

}

}

}

fragment[i]=lowest; barray[parray[i]]=1; lowest=10000;

}

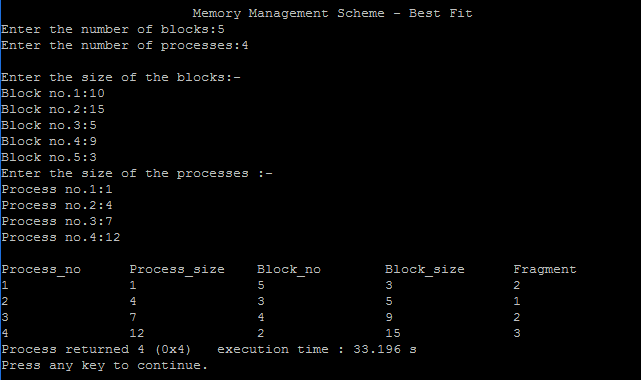
printf("\nProcess\_no\tProcess\_size\tBlock\_no\tBlock\_size\tFragment");

for(i=1;i<=np && parray[i]!=0;i++)

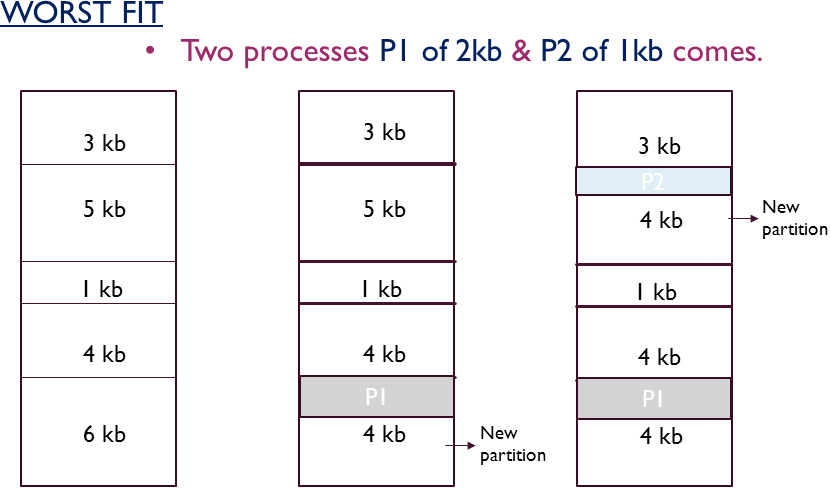
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,p[i],parray[i],b[parray[i]],fragment[i]);

}

Output:



* 1. **Worst Fit** Allocate the process to the partition which is the largest sufficient among the freely available partitions available in the main memory.



**Example:**

//C program to implement Worst Fit Algorithms

#include<stdio.h>

int main()

{

int fragments[10], blocks[10], files[10];

int m, n, number\_of\_blocks, number\_of\_files, temp, top = 0;

static int block\_arr[10], file\_arr[10];

printf("\nEnter the Total Number of Blocks:\t");

scanf("%d",&number\_of\_blocks);

printf("Enter the Total Number of Files:\t");

scanf("%d",&number\_of\_files); printf("\nEnter the Size of the Blocks:\n");

for(m = 0; m < number\_of\_blocks; m++)

{

printf("Block No.[%d]:\t", m + 1);

scanf("%d", &blocks[m]);

}

printf("Enter the Size of the Files:\n"); for(m = 0; m < number\_of\_files; m++)

{

printf("File No.[%d]:\t", m + 1);

scanf("%d", &files[m]);

}

for(m = 0; m < number\_of\_files; m++){ for(n = 0; n < number\_of\_blocks; n++)

{

if(block\_arr[n] != 1)

{

temp = blocks[n] - files[m]; if(temp >= 0)

{

if(top < temp)

{

file\_arr[m] = n; top = temp;

}

}

}

fragments[m] = top; block\_arr[file\_arr[m]] = 1;

top = 0;

}

}

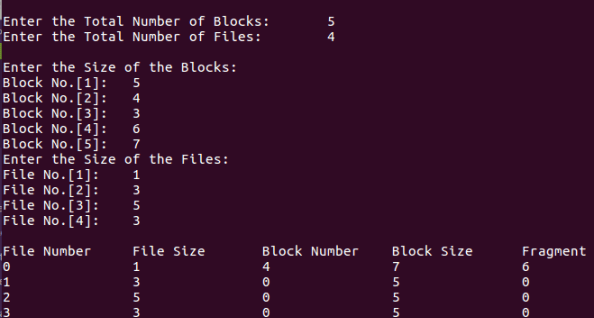
printf("\nFile Number\tFile Size\tBlock Number\tBlock Size\tFragment"); for(m = 0; m < number\_of\_files; m++)

{

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", m, files[m], file\_arr[m], blocks[file\_arr[m]], fragments[m]);

}

printf("\n"); return 0;}

Output:

**LAB EXERCISE:**

Q1. Briefly explain following:

1. First fit Memory Algorithm
2. Worst fit Memory Algorithm

Q2.Write code for First fit Memory Algorithm using Python language?

Q3.Write code for Best fit Memory Algorithm using Python language?