OS EXPERIMENT 8

Name: Mohammed Arhaan Ali
Div: S11
Roll No: 02

Aim: Write a program to demonstrate the concept of dynamic partitioning placement algorithms i.e. Best Fit, First Fit, Worst Fit.

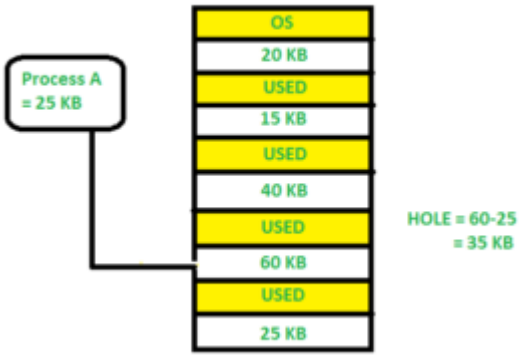
Theory:
In the operating system, the following are four common memory management techniques.
Single contiguous allocation: Simplest allocation method used by MS-DOS. All memory (except some reserved for OS) is available to a process.
Partitioned allocation: Memory is divided into different blocks or 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 into 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.
When it is time to load a process into the main memory and if there is more than one free block of memory of sufficient size then the OS decides which free block to allocate.
There are different Placement Algorithm:
A. First Fit

B. Best Fit
C. Worst Fit
D. Next Fit

1. First Fit: In the first fit, the partition is allocated which is the first sufficient block from the top of Main Memory. It scans memory from the beginning and chooses the first available block that is large enough. Thus it allocates the first hole that is large enough.

2. Best Fit Allocate the process to the partition which is the first smallest sufficient partition among the free available partition. It searches the entire list of holes to find the smallest hole whose size is greater than or equal to the size of the process.

3. Worst Fit Allocate the process to the partition which is the largest sufficient among the freely available partitions available in the main memory. It is opposite to the best-fit algorithm. It searches the entire list of holes to find the largest hole and allocate it to process.



4. Next Fit: Next fit is similar to the first fit but it will search for the first sufficient partition from the last allocation point.

Is Best-Fit really best?

Although best fit minimises the wastage space, it consumes a lot of processor time for searching the block which is close to the required size. Also, Best-fit may perform poorer than other algorithms in some cases. For example, see the exercise below.

Sl.No.	Partition Allocation Method	Advantages	Disadvantages
1.	Fixed Partition	Simple, easy to use, no complex algorithms needed	Memory waste, inefficient use of memory resources
2.	Dynamic Partition	Flexible, more efficient, partitions allocated as required	Requires complex algorithms for memory allocation
3.	Best-fit Allocation	Minimizes memory waste, allocates smallest suitable partition	More computational overhead to find smallest split

4.	Worst-fit Allocation	Ensures larger processes have sufficient memory	May result in substantial memory waste
5.	First-fit Allocation	Quick, efficient, less computational work	Risk of memory fragmentation

Comparison of Partition Allocation Methods:Exercise: Consider the requests from processes in given order 300K, 25K, 125K, and 50K. Let there be two blocks of memory available of size 150K followed by a block size 350K.
Which of the following partition allocation schemes can satisfy the above requests?
A) Best fit but not first fit.
B) First fit but not best fit.
C) Both First fit & Best fit.
D) neither first fit nor best fit.
Solution: Let us try all options.
Best Fit:
300K is allocated from a block of size 350K. 50 is left in the block.
25K is allocated from the remaining 50K block. 25K is left in the block.
125K is allocated from 150 K block. 25K is left in this block also.
50K can’t be allocated even if there is 25K + 25K space available.
First Fit:
300K request is allocated from 350K block, 50K is left out.
25K is allocated from the 150K block, 125K is left out.
Then 125K and 50K are allocated to the remaining left out partitions.
So, the first fit can handle requests.
So option B is the correct choice.

Program:
a)First Fit:

```
#include<stdio.h>

// Function to allocate memory to
// blocks as per First fit algorithm
```

```
void firstFit(int blockSize[], int m, int processSize[], int n)
{
    int i, j;
    // Stores block id of the
    // block allocated to a process
    int allocation[n];

    // Initially no block is assigned to any process
    for(i = 0; i < n; i++)
    {
        allocation[i] = -1;
    }

    // pick each process and find suitable blocks
    // according to its size ad assign to it
    for (i = 0; i < n; i++)  //here, n -> number of processes
    {
        for (j = 0; j < m; j++)  //here, m -> number of blocks
        {
            if (blockSize[j] >= processSize[i])
            {
                // allocating block j to the ith process
                allocation[i] = j;

                // Reduce available memory in this block.
                blockSize[j] -= processSize[i];

                break; //go to the next process in the queue
            }
        }
    }

    printf("\nProcess No.\tProcess Size\tBlock no.\n");
    for (int i = 0; i < n; i++)
    {
        printf(" %i\t\t\t", i+1);
        printf("%i\t\t\t\t", processSize[i]);
```

```
        if (allocation[i] != -1)
            printf("%i", allocation[i] + 1);
        else
            printf("Not Allocated");
        printf("\n");
    }
}

// Driver code
int main()
{
    int m; //number of blocks in the memory
    int n; //number of processes in the input queue
    int blockSize[] = {100, 500, 200, 300, 600};
    int processSize[] = {212, 417, 112, 426};
    m = sizeof(blockSize) / sizeof(blockSize[0]);
    n = sizeof(processSize) / sizeof(processSize[0]);

    firstFit(blockSize, m, processSize, n);

    return 0 ;
}
```

b)Best Fit:

```
#include<stdio.h>

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])
;
}
```

c) Worst Fit:

```
#include <stdio.h>

void implimentWorstFit(int blockSize[], int blocks, int processSize[], int
processes)
{
   // This will store the block id of the allocated block to a process
   int allocation[processes];
   int occupied[blocks];

   // initially assigning -1 to all allocation indexes
   // means nothing is allocated currently
   for(int i = 0; i < processes; i++){
      allocation[i] = -1;
   }

   for(int i = 0; i < blocks; i++){
      occupied[i] = 0;
   }

   // pick each process and find suitable blocks
   // according to its size ad assign to it
   for (int i=0; i < processes; i++)
   {
   int indexPlaced = -1;
   for(int j = 0; j < blocks; j++)
   {
      // if not occupied and block size is large enough
      if(blockSize[j] >= processSize[i] && !occupied[j])
         {
            // place it at the first block fit to accomodate process
            if (indexPlaced == -1)
               indexPlaced = j;

            // if any future block is larger than the current block where
            // process is placed, change the block and thus indexPlaced
            else if (blockSize[indexPlaced] < blockSize[j])
```

```
                indexPlaced = j;
            }
        }

        // If we were successfully able to find block for the process
        if (indexPlaced != -1)
        {
            // allocate this block j to process p[i]
            allocation[i] = indexPlaced;

            // make the status of the block as occupied
            occupied[indexPlaced] = 1;

            // Reduce available memory for the block
            blockSize[indexPlaced] -= processSize[i];
        }
    }

    printf("\nProcess No.\tProcess Size\tBlock no.\n");
    for (int i = 0; i < processes; i++)
    {
        printf("%d \t\t\t %d \t\t\t", i+1, processSize[i]);
        if (allocation[i] != -1)
            printf("%d\n",allocation[i] + 1);
        else
            printf("Not Allocated\n");
    }
}

// Driver code
int main()
{
    int blockSize[] = {100, 50, 30, 120, 35};
    int processSize[] = {40, 10, 30, 60};
    int blocks = sizeof(blockSize)/sizeof(blockSize[0]);
    int processes = sizeof(processSize)/sizeof(processSize[0]);
```

```
    implimentWorstFit(blockSize, blocks, processSize, processes);

    return 0;
}
```

Output:

a)First Fit:

```
PS C:\Users\arhaa\OneDrive\Desktop\OS> cd "c:\Users\arhaa\OneDrive\Desktop\OS\" ; if ($?)
 { .\firstfit }

Process No.     Process Size    Block no.
 1                      212                             2
 2                      417                             5
 3                      112                             2
 4                      426                             Not Allocated
PS C:\Users\arhaa\OneDrive\Desktop\OS>
```

b)Best Fit:

```
PS C:\Users\arhaa\OneDrive\Desktop\OS> cd "c:\Users\arhaa\OneDrive\Desktop\OS\"
 .\bestfit }

                        Memory Management Scheme - Best Fit
Enter the number of blocks:4
Enter the number of processes:3

Enter the size of the blocks:-
Block no.1:10
Block no.2:20
Block no.3:30
Block no.4:50

Enter the size of the processes :-
Process no.1:5
Process no.2:12
Process no.3:9

Process_no      Process_size    Block_no        Block_size      Fragment
1               5               1               10              5
2               12              2               20              8
3               9               3               30              21
PS C:\Users\arhaa\OneDrive\Desktop\OS>
```

c)Worst Fit:

```
PS C:\Users\arhaa\OneDrive\Desktop\OS> cd "c:\Users\arhaa\OneDrive\Desktop\OS\" ;
 .\bestfit }

                        Memory Management Scheme - Best Fit
Enter the number of blocks:4
Enter the number of processes:3

Enter the size of the blocks:-
Block no.1:10
Block no.2:20
Block no.3:30
Block no.4:50

Enter the size of the processes :-
Process no.1:5
Process no.2:12
Process no.3:9

Process_no      Process_size    Block_no        Block_size      Fragment
1               5               1               10              5
2               12              2               20              8
3               9               3               30              21
PS C:\Users\arhaa\OneDrive\Desktop\OS>
```

Conclusion: Thus we have successfully implemented dynamic partitioning placement algorithms.