Name: Shawn Louis

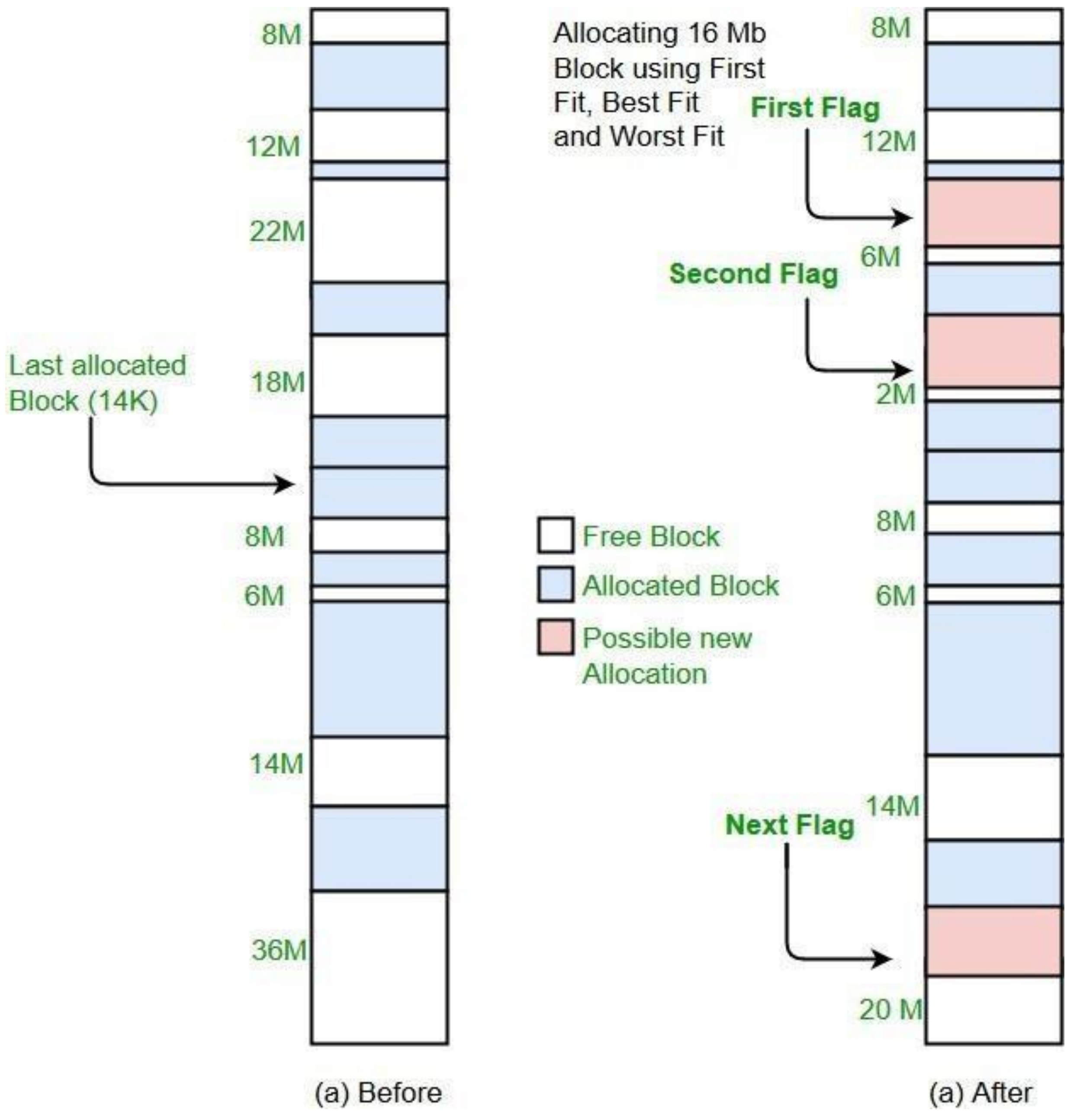
SE COMPS
Roll No : 31

Experiment no 11

Aim : Write a program to implement dynamic partitioning placement algorithms - Worst Fit algorithm.

Theory :

Worst Fit allocates a process to the partition which is largest sufficient among the freely available partitions available in the main memory. If a large process comes at a later stage, then memory will not have space to accommodate it.



Implementation:

1- Input memory blocks and processes with sizes.
2- Initialize all memory blocks as free.
3- Start by picking each process and find the minimum block size that can be assigned to current process i.e., find min(bockSize[1], blockSize[2],.....blockSize[n]) > processSize[current], if found then assign it to the current process.
5- If not then leave that process and keep checking the further processes.

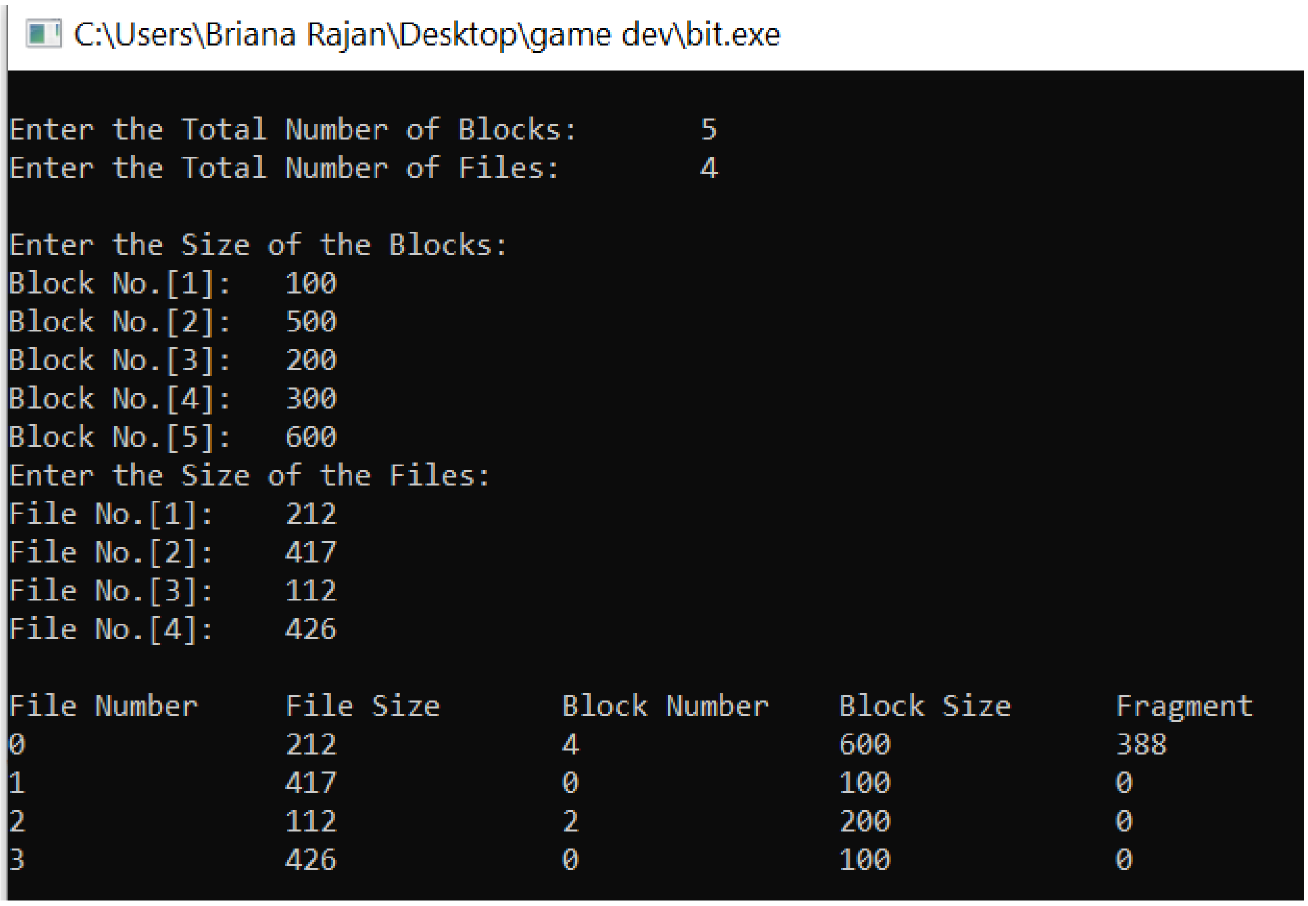
Program :

```
#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 :



```
C:\Users\Briana Rajan\Desktop\game dev\bit.exe

Enter the Total Number of Blocks:       5
Enter the Total Number of Files:        4

Enter the Size of the Blocks:
Block No.[1]:   100
Block No.[2]:   500
Block No.[3]:   200
Block No.[4]:   300
Block No.[5]:   600
Enter the Size of the Files:
File No.[1]:    212
File No.[2]:    417
File No.[3]:    112
File No.[4]:    426

File Number     File Size       Block Number    Block Size      Fragment
0               212             4               600             388
1               417             0               100             0
2               112             2               200             0
3               426             0               100             0
```

Conclusion : Thus worst fit

algorithm was studied and

implemented in C language.

As the name suggests it is the

worst of all the three- best fit,

first fit and worst fit

algorithms as the biggest

block available is dedicated

to the processes that come in

which may leave a bigger

sized process entered at a

later stage with no available

blocks.