

Experiment no. 7

Aim: Write a C program for the memory management algorithm.

Theory: Memory Management is the process of controlling and coordinating computer memory, assigning portions known as blocks to various running programs to optimize the overall performance of the system.

It is the most important function of an operating system that manages primary memory. It helps processes to move back and forward between the main memory and execution disk. It helps OS to keep track of every memory location, irrespective of whether it is allocated to some process or it remains free.

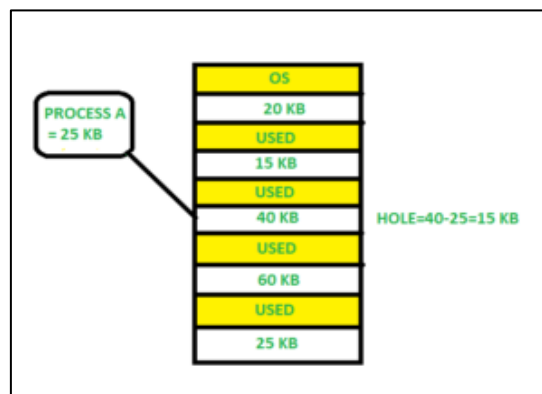
In **Partition Allocation**, when there is more than one partition freely available to accommodate a process's request, a partition must be selected. To choose a particular partition, a partition allocation method is needed. A partition allocation method is considered better if it avoids internal fragmentation.

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.

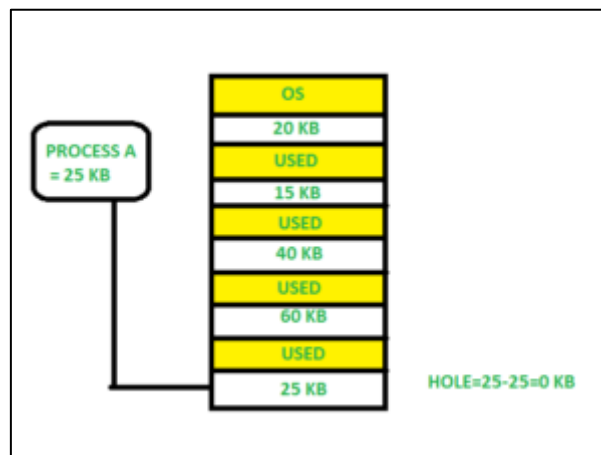
There are different Placement Algorithm:

- A. First Fit
- B. Best Fit
- C. Worst Fit
- D. Next Fit

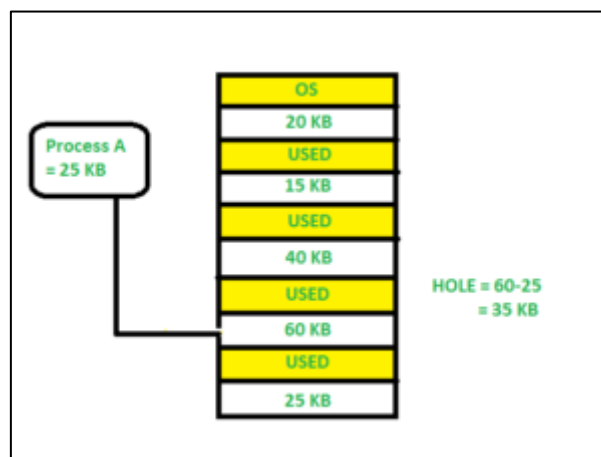
1. First Fit: In the first fit, the partition is allocated which is the first sufficient block from the top of Main Memory. It scans memory from the beginning and chooses the first available block that is large enough. Thus, it allocates the first hole that is large enough.



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.

Input for the algorithm:

Input: blockSize [] = {100, 500, 200, 300, 600};

processSize [] = {212, 417, 112, 426};

First Fit:**Code:**

```
#include <stdio.h>

void firstFit(int blockSize[],int m,int processSize[],int n)
{
    //Block allocations for process
    int allocation[n];
    //Initially allocate the index of -1 to all allocation blocks
    for (int i=0;i<n;i++)
    {
        allocation[i]=-1;
    }
    //Fetching the process
    for (int i=0;i<n;i++)
    {
        //Setting the index with first available block
        int best_idx=-1;
        //Running for-loop over block sizes
        for (int j=0;j<m;j++)
        {
            //Block has been found
            if (blockSize[j]>=processSize[i])
            {
                best_idx=j;
                break;
            }
        }
        //Cannot be allotted
        if (best_idx==-1)
```

```

        continue;

//At the end allot the block to process and re-calculate free-space
blockSize[best_idx]-=processSize[i];
allocation[i]=best_idx;
}
//Now printing the allocation array
printf("\nProcess No.\tProcess Size\tBlock no.\n");

for(int i=0;i<n;i++)
{
    if (allocation[i]!=-1)

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

    else

        printf("\n %d \t\t %d \t\t cannot be allocated",i+1,processSize[i]);

}

printf("\n\nEnd of process\n");
}
int main()
{
    int blockSize[] = {100, 500, 200, 300, 600};
    int processSize[] = {212, 417, 112, 426};
    int m = sizeof(blockSize)/sizeof(blockSize[0]);
    int n = sizeof(processSize)/sizeof(processSize[0]);

    firstFit(blockSize, m, processSize, n);
    return 0 ;
}

```

Output:

The screenshot displays an online C compiler interface. The top section shows the source code for a program implementing the First Fit memory management algorithm. The code defines an array of block sizes and an array of process sizes, then iterates through the processes to find the first available block that can accommodate each process. The output section shows the execution results, including a table of process allocations and a message indicating that the program finished with exit code 0.

```
51 }
52 int main()
53 {
54     printf("First Fit Memory management Algorithm : \n");
55     int blockSize[] = {100, 500, 200, 300, 600};
56     int processSize[] = {212, 417, 112, 426};
57     int m = sizeof(blockSize)/sizeof(blockSize[0]);
58     int n = sizeof(processSize)/sizeof(processSize[0]);
59
60     firstFit(blockSize, m, processSize, n);
61     return 0;
62 }
63
```

input

First Fit Memory management Algorithm :

Process No.	Process Size	Block no.
1	212	2
2	417	5
3	112	2
4	426	cannot be allocated

End of process

...Program finished with exit code 0
Press ENTER to exit console.

Next Fit:

Code:

```
#include <stdio.h>

void nextFit(int blockSize[],int m,int processSize[],int n)
{
    //Block allocations for process

    int allocation[n];

    //Initially allocate the index of -1 to all allocation blocks

    for (int i=0;i<n;i++)
    {
        allocation[i]=-1;
    }

    //Pointer to keep track of where last allocation was made

    int j=0;

    //Counter to keep track of how many blocks have been visited

    //Fetching the process and allocating block

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

        int count=0;

        // Do not start from beginning

        while (count < m)

        {

            if (blockSize[j] >= processSize[i])
```

```

{
    // allocate block j to p[i] process

    allocation[i] = j;

    // Reduce available memory in this block.

    blockSize[j] -= processSize[i];

    break;
}

// mod m will help in traversing the blocks from

    // starting block after we reach the end.

    j = (j + 1) % m;

    count+=1;

}

}

//Now printing the allocation array

printf("\nProcess No.\tProcess Size\tBlock no.\n");

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

{

    if (allocation[i]!=-1)

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

    else

        printf("\n %d \t\t %d \t\t cannot be allocated",i+1,processSize[i]);

}

```

```

        printf("\n\nEnd of process\n");
    }

    int main()

    {int blockSize[] = {100, 500, 200, 300, 600};

        int processSize[] = {212, 417, 112, 426};

        int m = sizeof(blockSize)/sizeof(blockSize[0]);

        int n = sizeof(processSize)/sizeof(processSize[0]);

        nextFit(blockSize, m, processSize, n);

        return 0 ;

    }

```

Output:

The screenshot shows an online C compiler interface with the following code in the editor:

```

50 }
51 printf("\n\nEnd of process\n");
52 }
53 int main()
54 { printf("Next Fit Memory management Algorithm :\n");
55   int blockSize[] = {100, 500, 200, 300, 600};
56   int processSize[] = {212, 417, 112, 426};
57   int m = sizeof(blockSize)/sizeof(blockSize[0]);
58   int n = sizeof(processSize)/sizeof(processSize[0]);
59   nextFit(blockSize, m, processSize, n);
60   return 0 ;
61 }
62

```

The output window displays the following text:

```

Next Fit Memory management Algorithm :

Process No.    Process Size    Block no.
1             212          2
2             417          5
3             112          5
4             426          cannot be allocated

End of process

...Program finished with exit code 0
Press ENTER to exit console.

```

The Windows taskbar at the bottom shows the system time as 10:35 on 23-05-2022.

Worst Fit:

Code:

```
#include <stdio.h>

void worstFit(int blockSize[],int m,int processSize[],int n)
{
    //Block allocations for process

    int allocation[n];

    //Initially allocate the index of -1 to all allocation blocks

    for (int i=0;i<n;i++)
    {
        allocation[i]=-1;
    }

    //Fetching the process

    for (int i=0;i<n;i++)
    {
        //Setting the index with maximal wastage

        int worst_idx=-1;

        //Keeping the track of free space in best block

        //Running for-loop over block sizes

        for (int j=0;j<m;j++)
        {
            if (blockSize[j]>=processSize[i])
```

```

    {
        //Choosing first block as the best allocated block
        if (worst_idx==-1)
        {
            worst_idx=j;
        }
        else
        {
            //Check if this free space is less than that of current best
            if (blockSize[worst_idx]<blockSize[j])
            {
                worst_idx=j;
            }
        }
    }

    if (worst_idx==-1)
        continue;

//At the end allot the block to process and re-calculate free-space

    blockSize[worst_idx]-=processSize[i];

    allocation[i]=worst_idx;
}

```

```

//Now printing the allocation array

printf("\nProcess No.\tProcess Size\tBlock no.\n");

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

{

    if (allocation[i]!=-1)

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

    else

        printf("\n %d \t\t%d \t\tcannot be allocated",i+1,processSize[i]);

}

printf("\n\nEnd of process\n");

}

int main()

{

    int blockSize[] = { 100, 500, 200, 300, 600};

    int processSize[] = { 212, 417, 112, 426};

    int m = sizeof(blockSize)/sizeof(blockSize[0]);

    int n = sizeof(processSize)/sizeof(processSize[0]);

    worstFit(blockSize, m, processSize, n);

    return 0 ;

}

```

Output:

The screenshot shows an online C compiler interface with the following components:

- Editor:** Contains C code for a memory management algorithm. The code defines an array of block sizes and a list of processes. It implements a 'Worst Fit' algorithm where processes are allocated to the largest available block that can accommodate them. Process 4 (size 426) cannot be allocated as no block is large enough.
- Output Console:** Displays the program's execution. It prints the title 'Worst Fit Memory management Algorithm :', followed by a table of process allocations, and ends with a confirmation message.
- Table:** Summarizes the allocation results for each process.
- System Tray:** Shows system status including temperature (29°C), time (10:36), and date (23-05-2022).

```
54 {
55     if (allocation[i] != -1)
56         printf("\n %d \t\t %d \t\t %d", i+1, processSize[i], allocation[i]+1);
57     else
58         printf("\n %d \t\t %d \t\t cannot be allocated", i+1, processSize[i]);
59 }
60 printf("\n\nEnd of process\n");
61 }
62 int main()
63 { printf("Worst Fit Memory management Algorithm :\n");
64   int blockSize[] = {100, 500, 200, 300, 600};
65   int processSize[] = {212, 417, 112, 426};
66   int m = sizeof(blockSize)/sizeof(blockSize[0]);
```

input

Worst Fit Memory management Algorithm :

Process No.	Process Size	Block no.
1	212	5
2	417	2
3	112	5
4	426	cannot be allocated

End of process

...Program finished with exit code 0
Press ENTER to exit console.

Type here to search

U: 0.02 MB/s
D: 0.00 MB/s

29°C Mostly clear

10:36
23-05-2022

Best Fit:

Code:

```
#include <stdio.h>

void bestFit(int blockSize[],int m,int processSize[],int n)
{
    //Block allocations for process

    int allocation[n];

    //Initially allocate the index of -1 to all allocation blocks

    for (int i=0;i<n;i++)
    {
        allocation[i]=-1;
    }

    //Fetching the process

    for (int i=0;i<n;i++)
    {
        //Setting the index with minimal wastage

        int best_idx=-1;

        //Keeping the track of free space in best block

        int diff;

        //Running for-loop over block sizes

        for (int j=0;j<m;j++)
        {
```

```
if (blockSize[j]>=processSize[i])
{
    //Choosing first block as the best allocated block
    if (best_idx==-1)
    {
        diff=blockSize[j]-processSize[i];

        best_idx=j;
    }
    else
    {
        int inter=blockSize[j]-processSize[i];

        //Check if this free space is less than that of current best
        if (diff>inter)
        {
            diff=inter;

            best_idx=j;
        }
    }
}

if (best_idx==-1)
```

```

        continue;

//At the end allot the block to process and re-calculate free-space

        blockSize[best_idx]-=processSize[i];

        allocation[i]=best_idx;

    }

//Now printing the allocation array

printf("\nProcess No.\tProcess Size\tBlock no.\n");

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

{

    if (allocation[i]!=-1)

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

    else

        printf("\n %d \t\t %d \t\t cannot be allocated",i+1,processSize[i]);

}

printf("\n\nEnd of process\n");

}

int main()

{

    int blockSize[] = {100, 500, 200, 300, 600};

    int processSize[] = {212, 417, 112, 426};

    int m = sizeof(blockSize)/sizeof(blockSize[0]);

    int n = sizeof(processSize)/sizeof(processSize[0]);

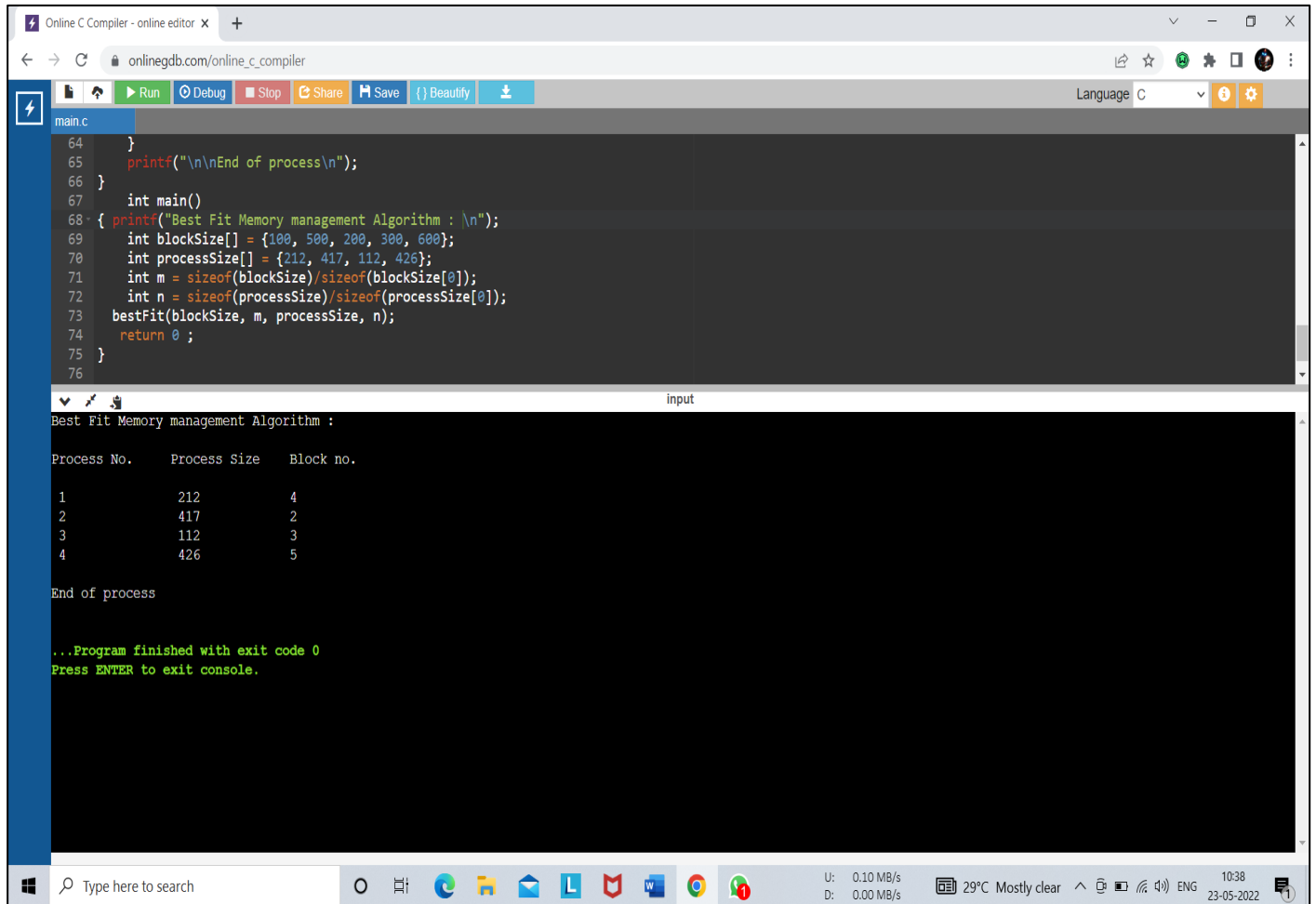
```

```
bestFit(blockSize, m, processSize, n);
```

```
return 0 ;
```

```
}
```


Output:



```
main.c
64 }
65 printf("\n\nEnd of process\n");
66 }
67 int main()
68 { printf("Best Fit Memory management Algorithm : \n");
69   int blockSize[] = {100, 500, 200, 300, 600};
70   int processSize[] = {212, 417, 112, 426};
71   int m = sizeof(blockSize)/sizeof(blockSize[0]);
72   int n = sizeof(processSize)/sizeof(processSize[0]);
73   bestFit(blockSize, m, processSize, n);
74   return 0 ;
75 }
76 }
```

input

```
Best Fit Memory management Algorithm :

Process No.    Process Size    Block no.

1              212          4
2              417          2
3              112          3
4              426          5

End of process

...Program finished with exit code 0
Press ENTER to exit console.
```

Conclusion:

Here, we performed memory management algorithm like First fit, Next fit, Worst fit, Best fit practical using C programming language.