FORM NO. SSGMCE/FRM/32 B

	SHRI SANT GAJANAN MAHARAJ COLLEGE ENGG. SHEGAON		**LABORATORY MANUAL**
	PRACTICAL EXPERIMENT INSTRUCTION SHEET		
	EXPERIMENT TITLE: **Best Fit Contiguous Memory Allocation**		
EXPERIMENT NO.: SSGMCE/WI/ASH/01/1A6/01		ISSUE NO.:	**ISSUE DATE:**26.08.23
REV. DATE: 26.08.23	REV. NO.: 00	DEPTT. : INFORMATION TECHNOLOGY.	
LABORATORY: OPERATING SYSTEMS		SEMESTER : IV	PAGE NO. 01 OF 05

Date:

BEST FIT CONTIGUOUS MEMORY ALLOCATION

01. AIM :

Write a program to implement Best Fit contiguous memory allocation algorithms.

02. FACILITIES :

Hardware: A Linux-based system (e.g., Ubuntu, Fedora, or any other distribution).

Software: C programming language, GCC compiler for compiling and running the program.

IDE/Editor: Any text editor or IDE that supports C programming (e.g., Visual Studio Code, Vim, Sublime Text, or Eclipse).

03. SCOPE :

This experiment aims to simulate memory allocation in an operating system using the **Best Fit** algorithm. The scope includes:

- Allocating memory blocks to processes using the Best Fit strategy.
- Understanding memory fragmentation and allocation in contiguous memory allocation schemes.

04 THEORY:

What is Memory Management?

Memory management mostly involves management of main memory. In a multiprogramming computer, the Operating System resides in a part of the main memory, and the rest is used by multiple processes. The task of subdividing the memory among different processes is called Memory Management.

Memory management is a method in the operating system to manage operations between main memory and disk during process execution. The main aim of memory management is to achieve efficient utilization of memory.

Why is Memory Management Required?

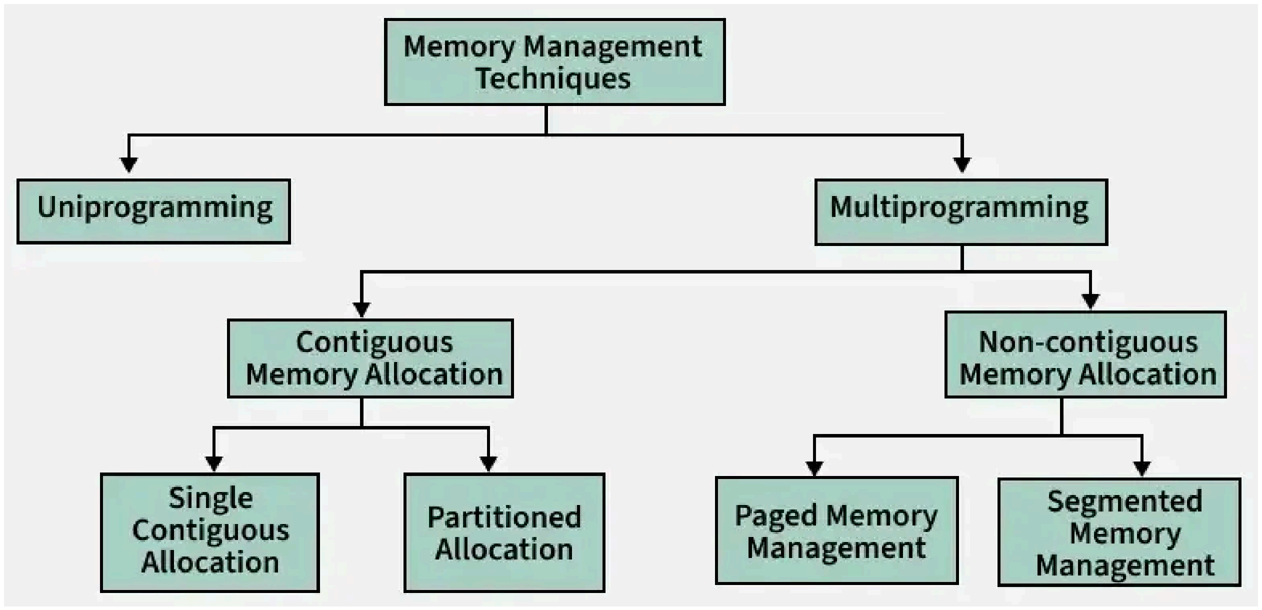
1. Allocate and de-allocate memory before and after process execution.
2. To keep track of used memory space by processes.
3. To minimize fragmentation issues.

4. To proper utilization of main memory.
5. To maintain data integrity while executing the process.

Memory Management Techniques:

Memory management is a crucial aspect of operating systems. It involves managing computer memory, ensuring that memory is allocated efficiently and appropriately to processes, thus optimizing system performance.

Memory management techniques can be broadly categorized into several strategies based on how the memory is allocated and utilized.


Memory Management Techniques
Uniprogramming
Multiprogramming
Contiguous Memory Allocation
Non-contiguous Memory Allocation
Single Contiguous Allocation
Partitioned Allocation
Paged Memory Management
Segmented Memory Management

Contiguous Memory Allocation:

Contiguous memory allocation is a memory allocation strategy. As the name implies, we utilize this technique to assign contiguous blocks of memory to each task. Thus, whenever a process asks to access the main memory, we allocate a continuous segment from the empty region to the process based on its size.

In this technique, memory is divided into fixed-sized partitions, and each partition can hold a single process. This means that a process occupies a block of memory that is continuous, without any gaps. The operating system uses a memory allocation strategy to assign these blocks of memory to processes based on certain criteria. The most common methods for assigning contiguous memory are **First Fit**, **Best Fit**, and **Worst Fit**.

Best-Fit Memory Allocation:

Best-Fit Allocation is a memory allocation technique used in operating systems to allocate memory to a process. In Best-Fit, the operating system searches through the list of free blocks of memory to find the block that is closest in size to the memory request from the process.

Once a suitable block is found, the operating system splits the block into two parts: the portion that will be allocated to the process, and the remaining free block.

How Best-Fit Memory Allocation Works:

1. **Memory Block Search:** The free memory blocks are sorted based on size, either in ascending or descending order.
2. **Process Request:** When a process requests memory, the allocator searches for the smallest free block that can satisfy the process's memory needs.
3. **Memory Assignment:** The smallest block of memory that can accommodate the process is selected and allocated.
4. **Remaining Space:** If there is remaining space in the selected block after allocating memory to the process, it remains as a new free block.

Advantages of Best-Fit Allocation:

1. Memory Efficient. The operating system allocates the job, minimum possible space in the memory, making memory management very efficient.
2. To save memory from getting wasted, it is the best method.
3. Improved memory utilization
4. Reduced memory fragmentation
5. Minimizes external fragmentation

Disadvantages of First-Fit Allocation:

1. It is a Slow Process. Checking the whole memory for each job makes the working of the operating system very slow. It takes a lot of time to complete the work.
2. Increased computational overhead
3. May lead to increased internal fragmentation
4. Can result in slow memory allocation times.

Example:

Imagine a system with memory blocks of sizes [100KB, 300KB, 500KB, 200KB]. If a process requests 250KB:

1. The system scans the available blocks and finds that the first block that is large enough but not too large is 300KB.
2. The 250KB is allocated to this block, and the remaining 50KB is left free.
3. The new free block of 50KB is added to the list of available blocks.

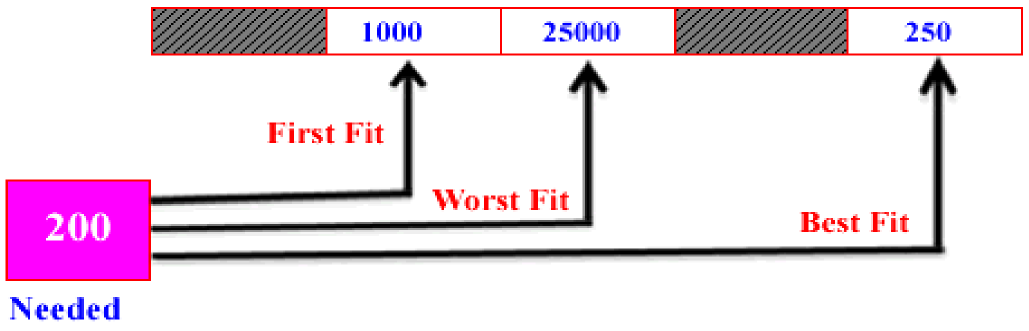
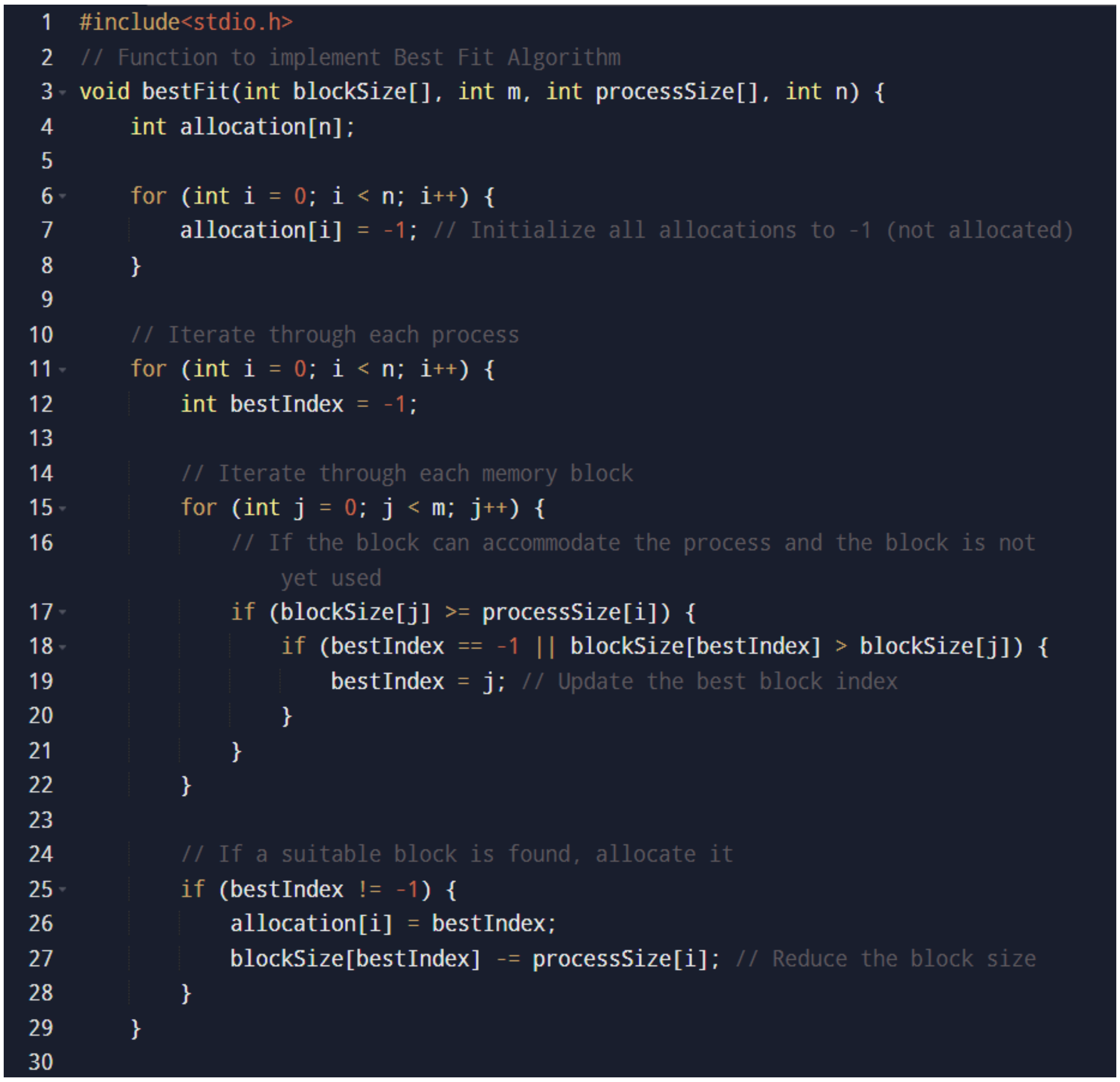


Fig: Showing First Fit, Worst Fit and Best Fit Memory Allocations

05 PROGRAM:



```
#include<stdio.h>
// Function to implement Best Fit Algorithm
void bestFit(int blockSize[], int m, int processSize[], int n) {
    int allocation[n];

    for (int i = 0; i < n; i++) {
        allocation[i] = -1; // Initialize all allocations to -1 (not allocated)
    }

    // Iterate through each process
    for (int i = 0; i < n; i++) {
        int bestIndex = -1;

        // Iterate through each memory block
        for (int j = 0; j < m; j++) {
            // If the block can accommodate the process and the block is not
                yet used
            if (blockSize[j] >= processSize[i]) {
                if (bestIndex == -1 || blockSize[bestIndex] > blockSize[j]) {
                    bestIndex = j; // Update the best block index
                }
            }
        }

        // If a suitable block is found, allocate it
        if (bestIndex != -1) {
            allocation[i] = bestIndex;
            blockSize[bestIndex] -= processSize[i]; // Reduce the block size
        }
    }

```

```c
    // Display the allocation result
    printf("\nBest Fit Allocation:\n");
    for (int i = 0; i < n; i++) {
        if (allocation[i] != -1)
            printf("Process %d allocated to block %d\n", i + 1, allocation[i] + 1);
        else
            printf("Process %d not allocated\n", i + 1);
    }
}

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;
}
```

06 OUTPUT

```
Best Fit Allocation:
Process 1 allocated to block 4
Process 2 allocated to block 2
Process 3 allocated to block 3
Process 4 allocated to block 5
```

08 CONCLUSION:

In conclusion, the Best Fit algorithm reduces fragmentation by placing data in the smallest available space. Though it improves memory usage, it can be slower due to the search for the optimal fit. This experiment highlights its effectiveness in minimizing fragmentation, with the trade-off of increased computational overhead.

09 VIVA QUESTIONS:

PREPARED BY **H. P. Amle**	**APPROVED BY** **HOD**