Green University of Bangladesh

Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

Semester: (Spring, Year:2023), BSc. in CSE (Day)

LAB REPORT - 03

Course Title: Operating Systems Lab

Course Code: CSE-310 **Section:** PC-201 DB

Student Details

Name		Students Id
1.	Md. Romzan Alom	201902144

Lab Date: 26-05-2023

Submission Date: 09-06-2023

Course Teacher's Name: Jarin Tasnim Tonvi

[For Teachers use only: Don't Write Anything inside this box]

Lab Report Status

Marks: ……………………………… **Signature:** …………………….

Comments: …………………………. **Date:** …………………………..

1. TITLE OF THE LAB EXPERIMENT

Implement best fit contiguous memory allocation algorithm using C programming language.

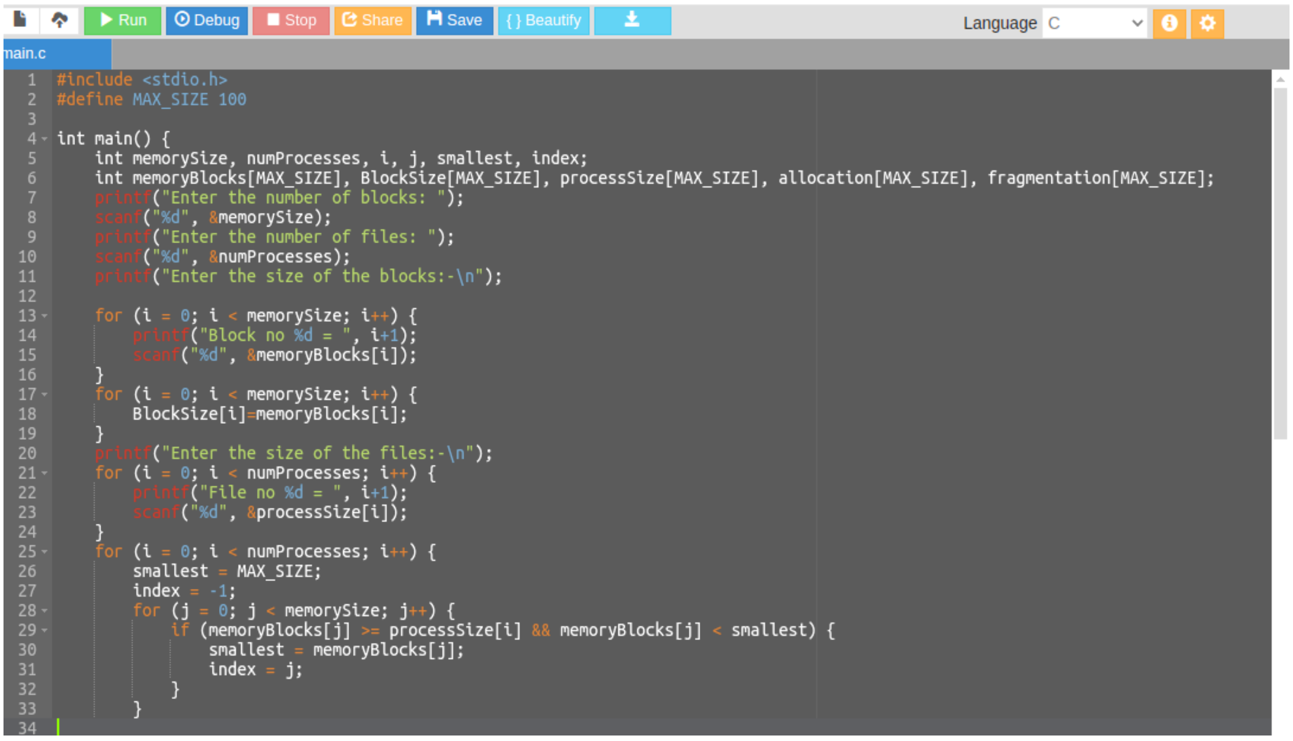
2. OBJECTIVES / AIM

- To gain a deeper understanding of the Best Fit contiguous memory allocation algorithm and its implementation.
- To learn how to simulate memory allocation using linked lists in C programming language.
- To evaluate the performance of the Best Fit contiguous memory allocation algorithm on a trace file of memory allocation requests.
- To compare the performance of the Best Fit contiguous memory allocation algorithm with other memory allocation algorithms.
- To gain practical experience in implementing algorithms and evaluating their performance.

3. PROBLEM STATEMENT

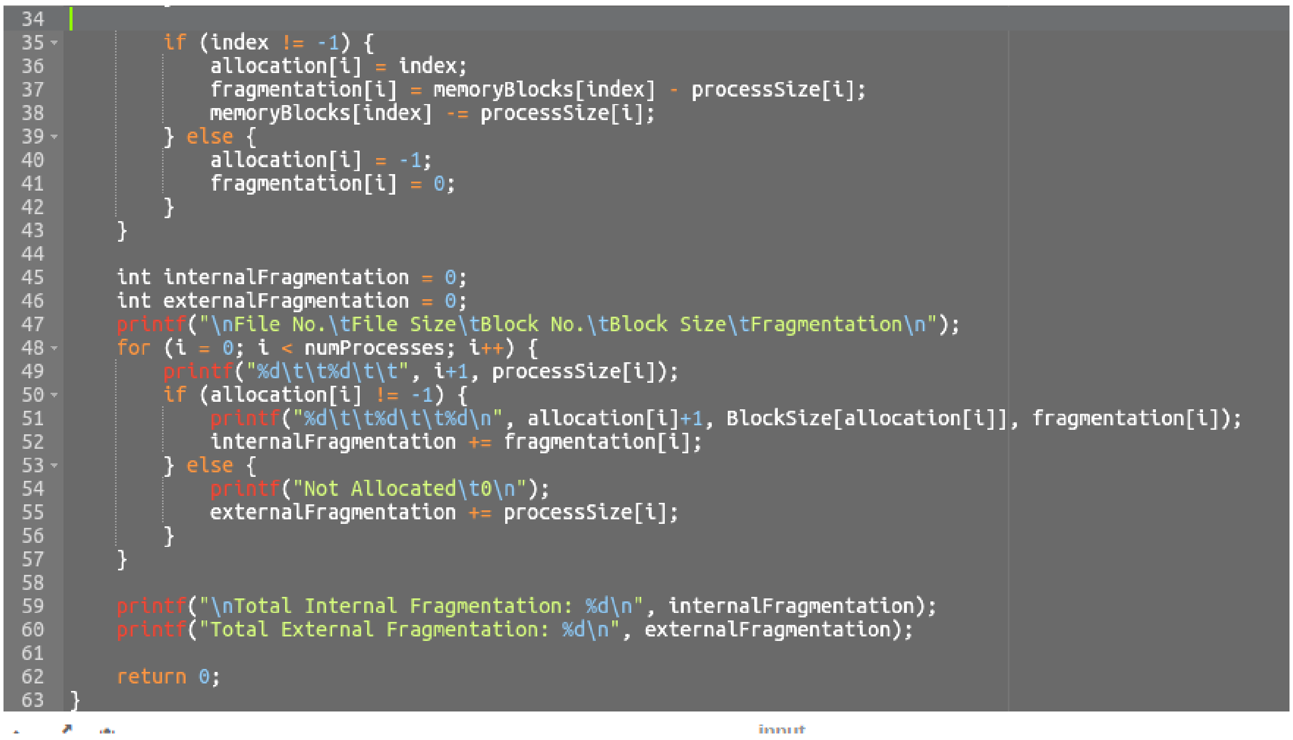
The Best Fit contiguous memory allocation algorithm was implemented by creating a data structure and using a linked list to track free memory segments. To evaluate the performance, a trace file was used to simulate the algorithm and record the number of memory allocations, block no, block size and fragmentation.

4. IMPLEMENTATION



```c
#include <stdio.h>
#define MAX_SIZE 100

int main() {
    int memorySize, numProcesses, i, j, smallest, index;
    int memoryBlocks[MAX_SIZE], BlockSize[MAX_SIZE], processSize[MAX_SIZE], allocation[MAX_SIZE], fragmentation[MAX_SIZE];
    printf("Enter the number of blocks: ");
    scanf("%d", &memorySize);
    printf("Enter the number of files: ");
    scanf("%d", &numProcesses);
    printf("Enter the size of the blocks:-\n");

    for (i = 0; i < memorySize; i++) {
        printf("Block no %d = ", i+1);
        scanf("%d", &memoryBlocks[i]);
    }
    for (i = 0; i < memorySize; i++) {
        BlockSize[i]=memoryBlocks[i];
    }
    printf("Enter the size of the files:-\n");
    for (i = 0; i < numProcesses; i++) {
        printf("File no %d = ", i+1);
        scanf("%d", &processSize[i]);
    }
    for (i = 0; i < numProcesses; i++) {
        smallest = MAX_SIZE;
        index = -1;
        for (j = 0; j < memorySize; j++) {
            if (memoryBlocks[j] >= processSize[i] && memoryBlocks[j] < smallest) {
                smallest = memoryBlocks[j];
                index = j;
            }
        }

```

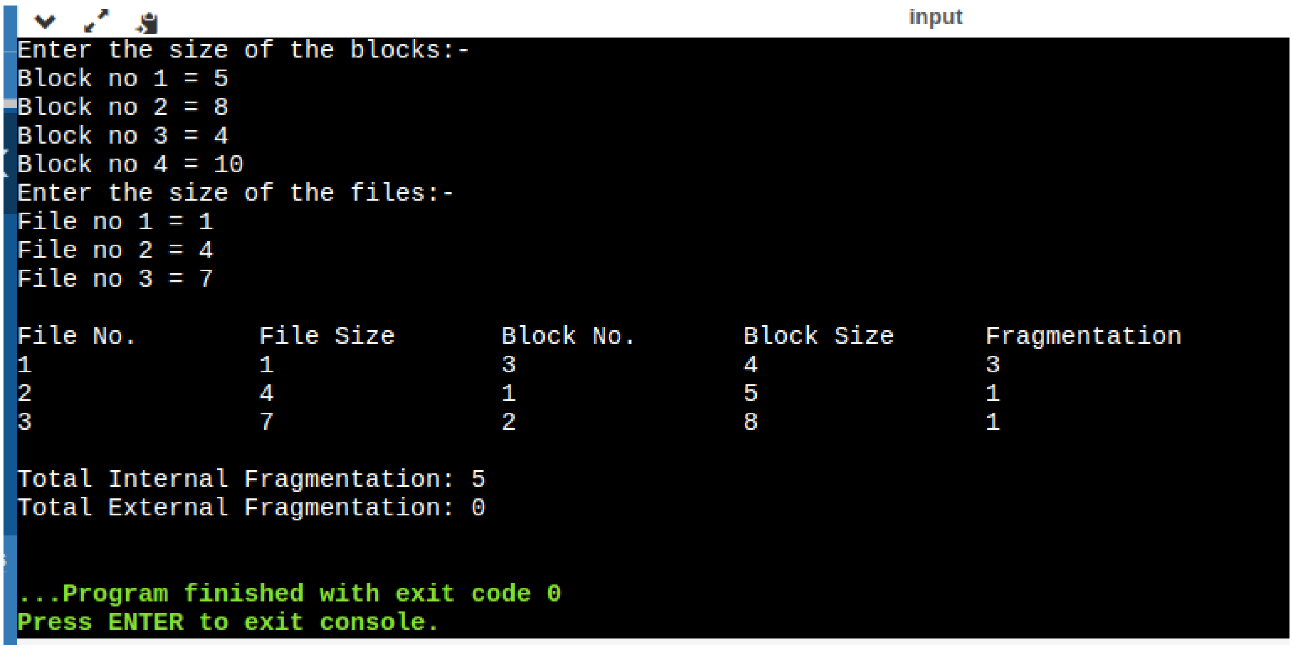
Figure_01: Code of Best Fit part-1



```
        if (index != -1) {
            allocation[i] = index;
            fragmentation[i] = memoryBlocks[index] - processSize[i];
            memoryBlocks[index] -= processSize[i];
        } else {
            allocation[i] = -1;
            fragmentation[i] = 0;
        }
    }

    int internalFragmentation = 0;
    int externalFragmentation = 0;
    printf("\nFile No.\tFile Size\tBlock No.\tBlock Size\tFragmentation\n");
    for (i = 0; i < numProcesses; i++) {
        printf("%d\t\t%d\t\t", i+1, processSize[i]);
        if (allocation[i] != -1) {
            printf("%d\t\t%d\t\t%d\n", allocation[i]+1, BlockSize[allocation[i]], fragmentation[i]);
            internalFragmentation += fragmentation[i];
        } else {
            printf("Not Allocated\t0\n");
            externalFragmentation += processSize[i];
        }
    }

    printf("\nTotal Internal Fragmentation: %d\n", internalFragmentation);
    printf("Total External Fragmentation: %d\n", externalFragmentation);

    return 0;
}
```

Figure_02: Code of Best Fit part-2

5. TEST RESULT



```
Enter the size of the blocks:-
Block no 1 = 5
Block no 2 = 8
Block no 3 = 4
Block no 4 = 10
Enter the size of the files:-
File no 1 = 1
File no 2 = 4
File no 3 = 7

File No.        File Size       Block No.       Block Size      Fragmentation
1               1               3               4               3
2               4               1               5               1
3               7               2               8               1

Total Internal Fragmentation: 5
Total External Fragmentation: 0


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

Figure_03: Input & output for Best Fit

According to this figure here we see user can put the number of block and file, size of block and file as input then using Best Fit to find the place of file in block and the fragmentation as output. From this figure we can see file 1's place in 3rd block and the fragmentation is 4-1 = 3. Similarly, file 2 and 3 are place in 1st and 2nd block. And final fragmentation is 5 (internal). There are no external fragmentation.

5. ANALYSIS AND DISCUSSION

This experiment mainly based on C program. It may have some compiler error. An efficient approach for managing memory in operating systems is the Best Fit contiguous memory allocation algorithm. It makes sure that processes are given the most effective memory allocation possible, minimizing fragmentation and enhancing system performance. To keep track of the free memory segments, the method needs additional overhead, which might have an influence on system performance. Furthermore, the approach might not work effectively when there are several little memory allocation requests since it can leave numerous small pieces.

6. SUMMARY

In this lab, we used the C programming language to develop the Best Fit contiguous memory allocation method and assess its performance using a trace file. The algorithm ran rapidly and produced fragments with a small average size. Overall, the C programming language may be used to create the Best Fit contiguous memory allocation method, which is a useful approach for managing memory in operating systems.