...for a complete human being

Texas International College

Lab No: 7 **Date: 2082/**

Title: Write a program to display the process allocation for user input free blocks and incoming process using Best fit allocation.

The Best Fit algorithm is a memory management strategy used in operating systems to allocate memory to processes. It assigns a process to the smallest block of memory that is large enough to hold the process. This helps reduce wasted space (fragmentation) in each allocated block.

Algorithm:

Step 1: A request for memory of size S arrives.

Step 2: Search the list of free memory blocks to find all blocks with size ≥ S.

Step 3: Among those blocks, select the block with the smallest size that can fulfill the request.

Step 4: Allocate the selected block to the process:

- If the block size is exactly S, allocate the entire block.
- If the block size is larger than S, allocate S units and keep the leftover as a smaller free block.

Step 5: Update the free memory list to reflect the allocation and any leftover fragments.

Step 6: If no suitable block is found, fail the allocation (memory insufficient).

Step 7: End.

Language: C++

IDE: VS Code

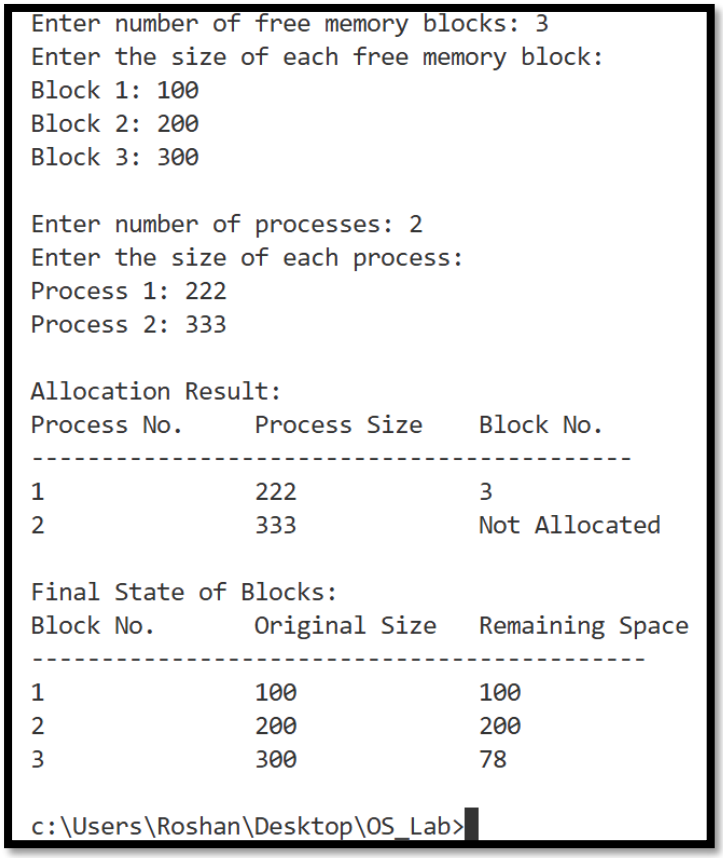
Code:

```
#include <iostream>
#include <vector>
using namespace std;

void bestFit(vector<int>& blockSize, const vector<int>& processSize) {
    int n = processSize.size(); // number of processes
    int m = blockSize.size();   // number of blocks
    vector<int> allocation(n, -1);
    // Keep original block sizes for later display
    vector<int> blockSizeBefore = blockSize;
    for (int i = 0; i < n; i++) {
        int bestIdx = -1;
        for (int j = 0; j < m; j++) {
            if (blockSize[j] >= processSize[i]) {
                if (bestIdx == -1 || blockSize[j] < blockSize[bestIdx]) {
                    bestIdx = j;
                }
            }
        }
        if (bestIdx != -1) {
            allocation[i] = bestIdx;
            blockSize[bestIdx] -= processSize[i]; // Allocate
        }
    }
    // Final output without Remaining Space
    cout << "\nAllocation Result:\n";
    cout << "Process No.\tProcess Size\tBlock No.\n";
    cout << "----------------------------------------\n";
    for (int i = 0; i < n; i++) {
```

```
            cout << i + 1 << "\t\t" << processSize[i] << "\t\t";
            if (allocation[i] != -1) {
                cout << allocation[i] + 1;
            } else {
                cout << "Not Allocated";
            }
            cout << endl;
        }
        // Show final state of all blocks
        cout << "\nFinal State of Blocks:\n";
        cout << "Block No.\tOriginal Size\tRemaining Space\n";
        cout << "-----------------------------------------\n";
        for (int j = 0; j < m; j++) {
            cout << j + 1 << "\t\t" << blockSizeBefore[j] << "\t\t" << blockSize[j] << endl;
        }
    }

    int main() {
        int m, n;
        cout << "Enter number of free memory blocks: ";
        cin >> m;
        vector<int> blockSize(m);
        cout << "Enter the size of each free memory block:\n";
        for (int i = 0; i < m; i++) {
            cout << "Block " << i + 1 << ": ";
            cin >> blockSize[i];
        }
        cout << "\nEnter number of processes: ";
        cin >> n;
        vector<int> processSize(n);
```

```
    cout << "Enter the size of each process:\n";
    for (int i = 0; i < n; i++) {
        cout << "Process " << i + 1 << ": ";
        cin >> processSize[i];
    }
    bestFit(blockSize, processSize);
    return 0;
}
```

Output:



Conclusion:

The Best Fit algorithm was implemented successfully, minimizing wasted memory by allocating the smallest suitable block. It improves memory utilization but may increase allocation time and can still face external fragmentation.