...for a complete human being

Texas International College

Lab No: 8 **Date: 2082/**

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

The Worst Fit memory allocation algorithm allocates the largest available memory block to a process request. The idea is to leave relatively large leftover blocks that may be useful for future allocations, potentially reducing external fragmentation compared to smaller leftover fragments. However, it may not always use memory efficiently and can lead to underutilization.

Algorithm:

Step 1: Start.

Step 2: Input the sizes of all available memory blocks.

Step 3: Input the sizes of all processes that need memory.

Step 4: Set allocation for all processes to "Not Allocated" initially.

Step 5: For each process (one by one):

Step 5.1: Search all memory blocks.

Step 5.2: From the blocks that can fit the process, find the largest block.

Step 5.3: If a suitable block is found:

Assign the process to that block.

Reduce the block size by the process size.

Step 5.4: If no suitable block is found, keep the process as Not Allocated.

Step 6: Display the final allocation table.

Step 7: Stop.

Language: C++

IDE: VS Code

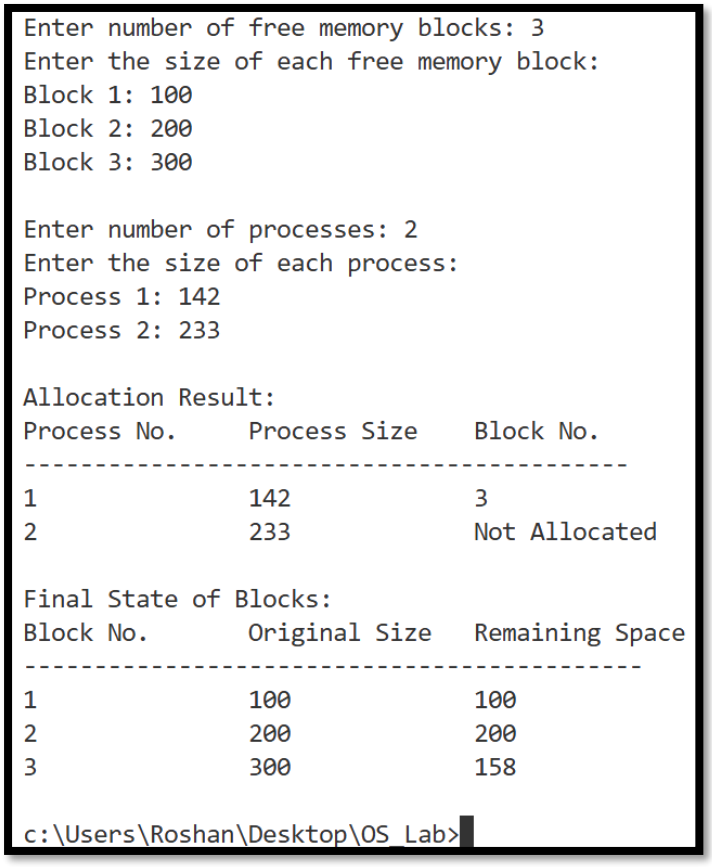
Code:

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

void worstFit(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 worstIdx = -1;
        for (int j = 0; j < m; j++) {
            if (blockSize[j] >= processSize[i]) {
                if (worstIdx == -1 || blockSize[j] > blockSize[worstIdx]) {
                    worstIdx = j;
                }
            }
        }
        if (worstIdx != -1) {
            allocation[i] = worstIdx;
            blockSize[worstIdx] -= 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];
    }
    worstFit(blockSize, processSize);
    return 0;
}
```

Output:



Conclusion:

The Worst Fit algorithm allocates the largest suitable block, leaving bigger leftover spaces. It can reduce small fragments but may be less efficient in overall memory utilization.