192311035

M.RAVI SAI VINAY

EXPERIMENT-13

Construct a C program for implementation of the various memory allocation strategies.

AIM:-

To construct a C program to demonstrate the implementation of various memory allocation strategies such as First Fit, Best Fit, and Worst Fit.

ALGORITHM:-

1. **Input Memory Details:**
    - Input the total memory blocks and their sizes.
    - Input the number of processes and their memory requirements.
2. **First Fit Algorithm:**
    - Allocate the first available block that fits the process.
3. **Best Fit Algorithm:**
    - Allocate the smallest block that fits the process.
4. **Worst Fit Algorithm:**
    - Allocate the largest block that fits the process.
5. **Display Results:**
    - Show memory allocation for each strategy.

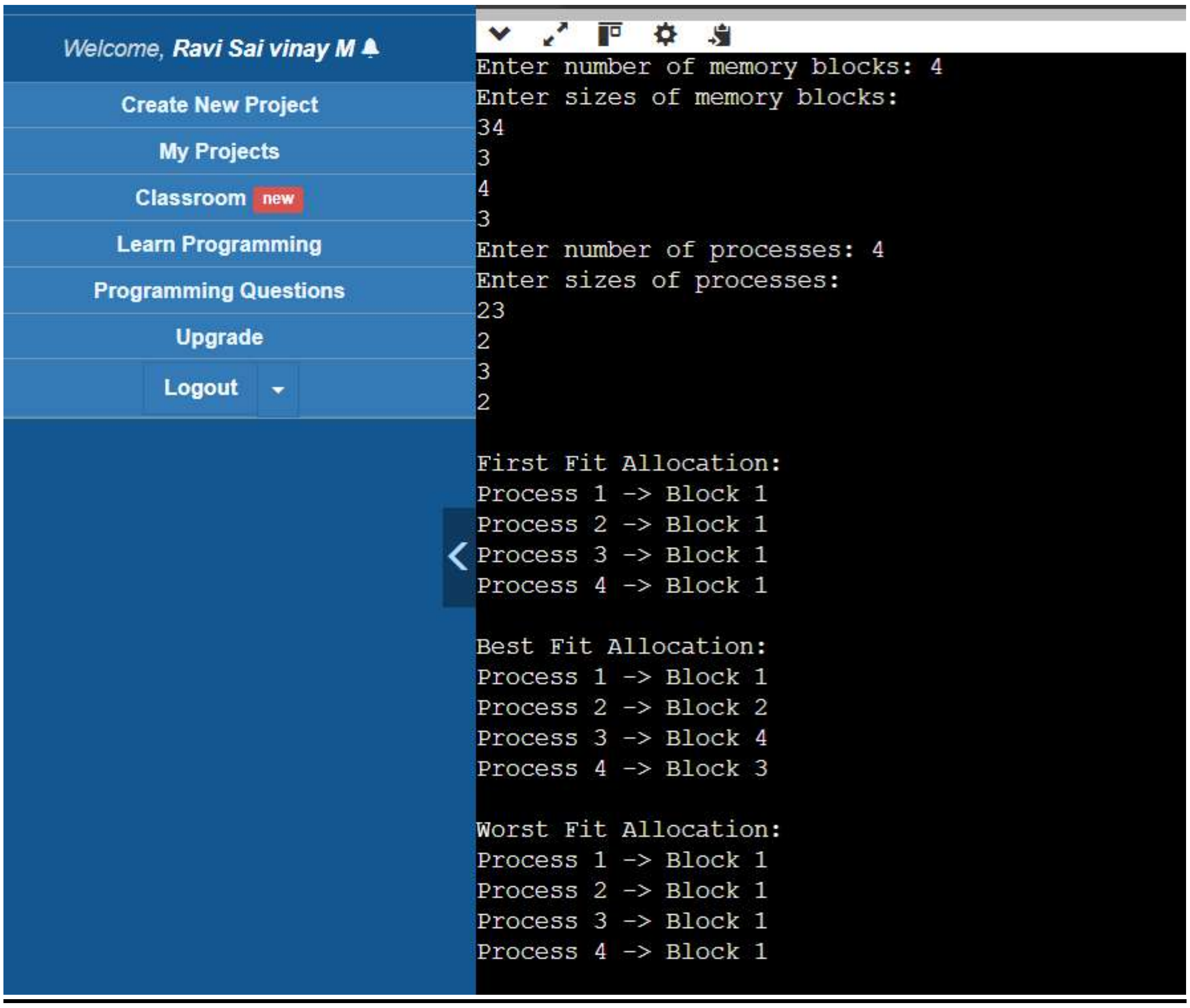
CODE:-

```
#include <stdio.h>

#include <limits.h>


void firstFit(int blockSize[], int m, int processSize[], int n) {
    int allocation[n];
    for (int i = 0; i < n; i++) allocation[i] = -1;

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (blockSize[j] >= processSize[i]) {
                allocation[i] = j;
                blockSize[j] -= processSize[i];
                break;
            }
        }
    }

    printf("\nFirst Fit Allocation:\n");
    for (int i = 0; i < n; i++) {
        if (allocation[i] != -1)
            printf("Process %d -> Block %d\n", i + 1, allocation[i] + 1);
        else
            printf("Process %d -> Not Allocated\n", i + 1);
    }
}

void bestFit(int blockSize[], int m, int processSize[], int n) {
    int allocation[n];
    for (int i = 0; i < n; i++) allocation[i] = -1;


    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];
        }
    }


    printf("\nBest Fit Allocation:\n");
    for (int i = 0; i < n; i++) {
        if (allocation[i] != -1)
            printf("Process %d -> Block %d\n", i + 1, allocation[i] + 1);
        else
            printf("Process %d -> Not Allocated\n", i + 1);
    }
}
```

```
void worstFit(int blockSize[], int m, int processSize[], int n) {
    int allocation[n];
    for (int i = 0; i < n; i++) allocation[i] = -1;


    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];
        }
    }


    printf("\nWorst Fit Allocation:\n");
    for (int i = 0; i < n; i++) {
        if (allocation[i] != -1)
            printf("Process %d -> Block %d\n", i + 1, allocation[i] + 1);
        else
```

```
            printf("Process %d -> Not Allocated\n", i + 1);
    }
}

int main() {
    int m, n;

    printf("Enter number of memory blocks: ");
    scanf("%d", &m);
    int blockSize[m];
    printf("Enter sizes of memory blocks:\n");
    for (int i = 0; i < m; i++) scanf("%d", &blockSize[i]);

    printf("Enter number of processes: ");
    scanf("%d", &n);
    int processSize[n];
    printf("Enter sizes of processes:\n");
    for (int i = 0; i < n; i++) scanf("%d", &processSize[i]);

    int blockSize1[m], blockSize2[m], blockSize3[m];
    for (int i = 0; i < m; i++) {
        blockSize1[i] = blockSize[i];
        blockSize2[i] = blockSize[i];
        blockSize3[i] = blockSize[i];
    }

    firstFit(blockSize1, m, processSize, n);
    bestFit(blockSize2, m, processSize, n);
    worstFit(blockSize3, m, processSize, n);

    return 0;
}
```

OUTPUT:-



```
Enter number of memory blocks: 4
Enter sizes of memory blocks:
34
3
4
3
Enter number of processes: 4
Enter sizes of processes:
23
2
3
2

First Fit Allocation:
Process 1 -> Block 1
Process 2 -> Block 1
Process 3 -> Block 1
Process 4 -> Block 1

Best Fit Allocation:
Process 1 -> Block 1
Process 2 -> Block 2
Process 3 -> Block 4
Process 4 -> Block 3

Worst Fit Allocation:
Process 1 -> Block 1
Process 2 -> Block 1
Process 3 -> Block 1
Process 4 -> Block 1
```

RESULT:-

The program successfully implements the First Fit, Best Fit, and Worst Fit memory allocation strategies, demonstrating their working with different process sizes and memory block configurations. Each strategy allocates processes to memory blocks based on its specific rules