

Name: K.Saikrishna

Reg-No: 192311106

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

Aim

To implement various memory allocation strategies in C, including **First Fit**, **Best Fit**, and **Worst Fit**, for allocating memory to processes.

Algorithm

1. **Input:**
 - Memory block sizes.
 - Process sizes.
2. For each process:
 - Apply the selected allocation strategy:
 - **First Fit:** Allocate the first block that fits the process.
 - **Best Fit:** Allocate the smallest block that fits the process.
 - **Worst Fit:** Allocate the largest block that fits the process.
3. Print the allocation results for each process.
4. **Output:** Process allocation details, indicating block numbers or unallocated processes.

Procedure

1. Define arrays for memory blocks and process sizes.
2. Use loops to simulate the allocation based on the chosen strategy.
3. Check block size availability and assign processes to blocks.
4. Print the results showing the process-to-block mapping or "Not Allocated" for unfit processes.

Code:

```
#include <stdio.h>
```

```
#define MAX 100
```

```
void firstFit(int blockSize[], int blocks, int processSize[], int processes) {
```

```
    int allocation[MAX] = {-1};
```

```
    for (int i = 0; i < processes; i++) {
```

```
        for (int j = 0; j < blocks; j++) {
```

```

        if (blockSize[j] >= processSize[i]) {

            allocation[i] = j;

            blockSize[j] -= processSize[i];

            break;

        }

    }

}

for (int i = 0; i < processes; 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 blocks, int processSize[], int processes) {

    int allocation[MAX] = {-1};

    for (int i = 0; i < processes; i++) {

        int bestIdx = -1;

        for (int j = 0; j < blocks; j++) {

            if (blockSize[j] >= processSize[i]) {

                if (bestIdx == -1 || blockSize[j] < blockSize[bestIdx])

                    bestIdx = j;

            }

        }

        if (bestIdx != -1)

            allocation[i] = bestIdx;

    }

}

```

```

    }

}

if (bestIdx != -1) {

    allocation[i] = bestIdx;

    blockSize[bestIdx] -= processSize[i];

}

}

for (int i = 0; i < processes; 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 blocks, int processSize[], int processes) {

    int allocation[MAX] = {-1};

    for (int i = 0; i < processes; i++) {

        int worstIdx = -1;

        for (int j = 0; j < blocks; 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];

    }

}

```

```

        }

    }

    if (worstIdx != -1) {

        allocation[i] = worstIdx;

        blockSize[worstIdx] -= processSize[i];

    }

}

for (int i = 0; i < processes; 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 blocks, processes;

    int blockSize[MAX], processSize[MAX];

    printf("Enter number of memory blocks: ");

    scanf("%d", &blocks);

    printf("Enter block sizes: ");

    for (int i = 0; i < blocks; i++) scanf("%d", &blockSize[i]);

    printf("Enter number of processes: ");

```

```
scanf("%d", &processes);

printf("Enter process sizes: ");

for (int i = 0; i < processes; i++) scanf("%d", &processSize[i]);


printf("\nFirst Fit Allocation:\n");

firstFit(blockSize, blocks, processSize, processes);


printf("\nBest Fit Allocation:\n");

bestFit(blockSize, blocks, processSize, processes);


printf("\nWorst Fit Allocation:\n");

worstFit(blockSize, blocks, processSize, processes);


return 0;

}
```

Output:

The screenshot shows a web application interface. On the left is a blue sidebar menu with the following items: 'Welcome, K Sai Krishna' with a bell icon, 'Create New Project', 'My Projects', 'Classroom' with a red 'new' badge, 'Learn Programming', 'Programming Questions', 'Upgrade', and 'Logout' with a dropdown arrow. The main area is a dark-themed code editor showing C code for memory allocation. The code includes functions for First Fit, Best Fit, and Worst Fit strategies. Below the code editor is a terminal window showing the output of the program. The terminal output displays the results for the First Fit strategy, followed by the headers for Best Fit and Worst Fit strategies.

```
105     firstFit(processes);
106
107     printf("\nMemory Allocation - Best Fit Strategy:\n");
108     initializeMemory();
109     bestFit(processes);
110
111     printf("\nMemory Allocation - Worst Fit Strategy:\n");
112     initializeMemory();
113     worstFit(processes);
114
115     return 0;
116 }
117
```

input

Memory Allocation - First Fit Strategy:
Process 1 allocated 150 KB from Block 2
Process 2 allocated 300 KB from Block 3
Process 3 allocated 50 KB from Block 1
Process 4 allocated 200 KB from Block 4
Process 5 allocated 400 KB from Block 5

Memory Allocation - Best Fit Strategy:

Memory Allocation - Worst Fit Strategy:

Result

1. The program prompts for memory block sizes and process sizes.
2. It outputs the allocation of processes using **First Fit**, **Best Fit**, and **Worst Fit** strategies.
3. Unallocated processes are displayed as "Not Allocated."