**EX13:Construct a C program for implementation of the various memory allocation strategies.**

**Aim:**

To construct a C program to implement and simulate various **memory allocation strategies** like **First Fit**, **Best Fit**, and **Worst Fit**.

**Algorithm:**

**1. Input Requirements:**

* Get the total memory size and the number of processes.
* For each process, get the memory size required.

**2. Define Strategies:**

* **First Fit**:
  + Allocate the first memory block that is large enough for the process.
* **Best Fit**:
  + Allocate the smallest available block that is large enough for the process.
* **Worst Fit**:
  + Allocate the largest available block for the process.

**3. Allocation Logic:**

* Traverse the list of memory blocks for each strategy to find a suitable block.
* Update memory block sizes after allocation.

**4. Output Results:**

* Display the allocation status and the remaining sizes of memory blocks

**Program:**

#include <stdio.h>

#include <stdlib.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("First Fit Allocation:\n");

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

if (allocation[i] != -1) {

printf("Process %d allocated to 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[bestIdx] > blockSize[j]) {

bestIdx = j;

}

}

}

if (bestIdx != -1) {

allocation[i] = bestIdx;

blockSize[bestIdx] -= processSize[i];

}

}

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

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

if (allocation[i] != -1) {

printf("Process %d allocated to 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[worstIdx] < blockSize[j]) {

worstIdx = j;

}

}

}

if (worstIdx != -1) {

allocation[i] = worstIdx;

blockSize[worstIdx] -= processSize[i];

}

}

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

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

if (allocation[i] != -1) {

printf("Process %d allocated to Block %d\n", i + 1, allocation[i] + 1);

} else {

printf("Process %d not allocated\n", i + 1);

}

}

}

int main() {

int blockSize[] = {100, 500, 200, 300, 600};

int processSize[] = {212, 417, 112, 426};

int m = sizeof(blockSize) / sizeof(blockSize[0]);

int n = sizeof(processSize) / sizeof(processSize[0]);

firstFit(blockSize, m, processSize, n);

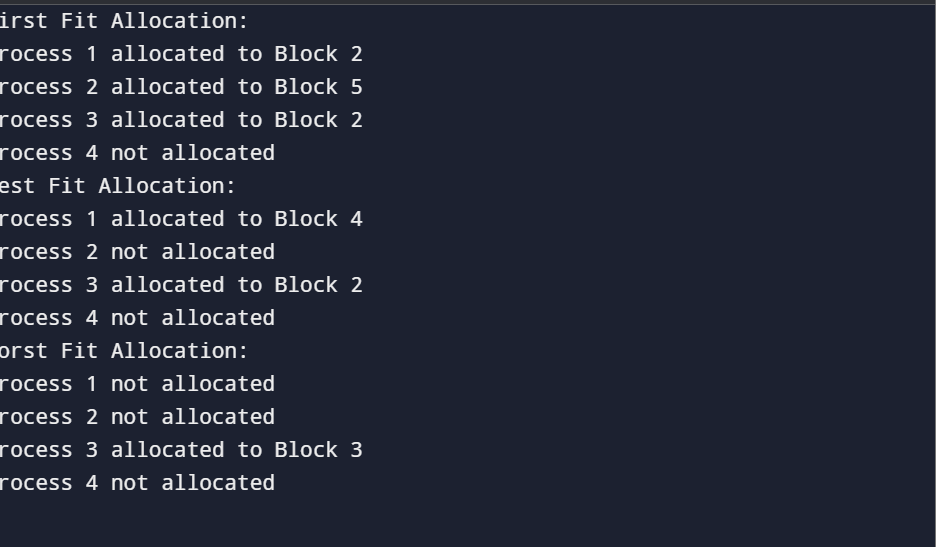
bestFit(blockSize, m, processSize, n);

worstFit(blockSize, m, processSize, n);

return 0;

}

**OUTPUT:**

****