K NARENDRA-192321162

1. 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, which are used for dynamic memory management in operating systems.

# Algorithm:

* 1. **First-Fit**: Allocate memory to the first available block that is large enough.
  2. **Best-Fit**: Allocate memory to the smallest block that can accommodate the request.
  3. **Worst-Fit**: Allocate memory to the largest block available.
  4. Each strategy will keep track of memory blocks, and when a request for memory is made, it will try to find the best suitable block using the strategy.
  5. After allocation, the program should display the memory blocks, and when freeing memory, it should merge adjacent free blocks if necessary.

# Procedure:

1. Define a structure for memory blocks.
2. Implement functions for First-Fit, Best-Fit, and Worst-Fit strategies.
3. Maintain a list of memory blocks with their status (allocated or free).
4. Allocate memory using the chosen strategy.
5. Free memory and merge adjacent free blocks.
6. Display memory allocation status after each operation.

CODE:

#include <stdio.h> #include <stdlib.h>

#define MEMORY\_SIZE 100

typedef struct Block { int size;

int is\_allocated;

} Block;

Block memory[MEMORY\_SIZE];

void initialize\_memory() {

for (int i = 0; i < MEMORY\_SIZE; i++) { memory[i].size = 0;

memory[i].is\_allocated = 0;

}

}

int first\_fit(int size) {

for (int i = 0; i < MEMORY\_SIZE; i++) {

if (!memory[i].is\_allocated && memory[i].size >= size) { memory[i].is\_allocated = 1;

return i;

}

}

return -1;

}

int best\_fit(int size) { int best\_idx = -1;

for (int i = 0; i < MEMORY\_SIZE; i++) {

if (!memory[i].is\_allocated && memory[i].size >= size) {

if (best\_idx == -1 || memory[i].size < memory[best\_idx].size) { best\_idx = i;

}

}

}

if (best\_idx != -1) {

memory[best\_idx].is\_allocated = 1;

}

return best\_idx;

}

int worst\_fit(int size) { int worst\_idx = -1;

for (int i = 0; i < MEMORY\_SIZE; i++) {

if (!memory[i].is\_allocated && memory[i].size >= size) {

if (worst\_idx == -1 || memory[i].size > memory[worst\_idx].size) { worst\_idx = i;

}

}

}

if (worst\_idx != -1) {

memory[worst\_idx].is\_allocated = 1;

}

return worst\_idx;

}

void free\_block(int index) {

memory[index].is\_allocated = 0;

}

void display\_memory() {

printf("\nMemory Blocks: \n");

for (int i = 0; i < MEMORY\_SIZE; i++) {

printf("Block %d: Size = %d, Allocated = %s\n", i, memory[i].size, memory[i].is\_allocated ? "Yes" : "No");

}

}

int main() {

initialize\_memory();

memory[0].size = 50; // First block of size 50 memory[1].size = 30; // Second block of size 30 memory[2].size = 70; // Third block of size 70 memory[3].size = 60; // Fourth block of size 60

int choice, size, block\_index;

while (1) {

printf("\nChoose memory allocation strategy:\n");

printf("1. First Fit\n2. Best Fit\n3. Worst Fit\n4. Exit\n"); scanf("%d", &choice);

if (choice == 4) break;

printf("Enter the size of the block to allocate: ");

scanf("%d", &size);

switch (choice) { case 1:

block\_index = first\_fit(size); break;

case 2:

block\_index = best\_fit(size); break;

case 3:

block\_index = worst\_fit(size); break;

default:

printf("Invalid choice.\n"); continue;

}

if (block\_index != -1) {

printf("Memory allocated at block %d\n", block\_index);

} else {

printf("No suitable block found for the requested size.\n");

}

display\_memory();

}

return 0;

}

OUTPUT:

