NAME: G. GANESH

REG NO: 192373008

EXERICSE-35

Consider a file system that brings all the file pointers together into an index block. The ith entry in the index block points to the ith block of the file. Design a C program to simulate the file allocation strategy.

AIM:

To design a C program that simulates the **indexed file allocation strategy**, where all the file pointers are organized into an index block, and each entry in the index block points to the corresponding block of the file.

Algorithm:

1. **Initialization:**
   - Define a structure to represent a file and its index block.
   - Create a simulated memory structure for file blocks and index blocks.
2. **File Creation:**
   - Allocate blocks to a file using an index block.
   - Store the pointers to the allocated blocks in the index block.
3. **File Access:**
   - Use the index block to access any block of the file directly.
4. **Display:**
   - Display the contents of the index block and the data stored in each block.

Procedure:

1. Define a structure to represent an index block and file blocks.
2. Write functions to create a file, allocate blocks, and read blocks using the index block.
3. Use an array to simulate memory blocks.
4. Implement error handling for cases like insufficient memory blocks.

Code:

```
#include <stdio.h>

#include <stdlib.h>

#define MAX_BLOCKS 100
```

```
#define MAX_FILES 10
#define BLOCK_SIZE 512
typedef struct {
    char name[20];
    int size;
    int index_block;
    int blocks[MAX_BLOCKS];
} File;
int memory[MAX_BLOCKS] = {0};
File files[MAX_FILES];
int file_count = 0;
void create_file(char *name, int size) {
    if (file_count >= MAX_FILES) {
        printf("Error: Maximum number of files reached.\n");
        return;
    }
    int index_block = -1;
    for (int i = 0; i < MAX_BLOCKS; i++) {
        if (memory[i] == 0) {
            index_block = i;
            memory[i] = 1;
            break;
        }
    }
    if (index_block == -1) {
        printf("Error: No free blocks available for index block.\n");
        return;
    }
    int blocks_needed = (size + BLOCK_SIZE - 1) / BLOCK_SIZE;
    int allocated_blocks = 0;
```

```
    int allocated[MAX_BLOCKS];
    for (int i = 0; i < MAX_BLOCKS && allocated_blocks < blocks_needed; i++) {
        if (memory[i] == 0) {
            allocated[allocated_blocks++] = i;
            memory[i] = 1;
        }
    }
    if (allocated_blocks < blocks_needed) {
        printf("Error: Insufficient blocks for file allocation.\n");
        memory[index_block] = 0; // Release the index block
        for (int i = 0; i < allocated_blocks; i++) {
            memory[allocated[i]] = 0; // Release allocated blocks
        }
        return;
    }
    File file;
    strcpy(file.name, name);
    file.size = size;
    file.index_block = index_block;
    for (int i = 0; i < blocks_needed; i++) {
        file.blocks[i] = allocated[i];
    }
    files[file_count++] = file;
    printf("File '%s' created successfully with size %d bytes.\n", name, size);
}
void display_files() {
    if (file_count == 0) {
        printf("No files to display.\n");
        return;
    }
```

```
    for (int i = 0; i < file_count; i++) {
        printf("File Name: %s\n", files[i].name);
        printf("File Size: %d bytes\n", files[i].size);
        printf("Index Block: %d\n", files[i].index_block);
        printf("Allocated Blocks: ");
        int blocks_needed = (files[i].size + BLOCK_SIZE - 1) / BLOCK_SIZE;
        for (int j = 0; j < blocks_needed; j++) {
            printf("%d ", files[i].blocks[j]);
        }
        printf("\n");
    }
}
int main() {
    int choice;
    char name[20];
    int size;
    while (1) {
        printf("\n1. Create File\n2. Display Files\n3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter file name: ");
                scanf("%s", name);
                printf("Enter file size (in bytes): ");
                scanf("%d", &size);
                create_file(name, size);
                break;
            case 2:
                display_files();
```

```
            break;
        case 3:
            exit(0);
        default:
            printf("Invalid choice.\n");
      }
   }
   return 0;
}
```

Result:

The program successfully simulates the **indexed file allocation strategy**, storing file pointers in an index block and allocating corresponding file blocks in memory.

Output: