NAME: G. GANESH

REG NO: 192373008

EXERICSE-36

With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file. Each block contains a pointer to the next block. Design a C program to simulate the file allocation strategy.

AIM:

To design a C program to simulate the linked allocation file storage strategy, where files are represented as linked lists of disk blocks.

Algorithm:

1. Initialize the disk blocks and their availability status.
2. Create a structure to represent a file, containing:
    - File name
    - Pointer to the first block
    - Pointer to the last block
3. Create a structure for disk blocks, containing:
    - Block ID
    - Pointer to the next block
4. Implement functions for:
    - Allocating a file:
        - Input file name and required blocks.
        - Check the availability of blocks and allocate sequentially by linking them.
    - Displaying the file allocation table:
        - Traverse the linked blocks for each file and display the block allocation.
    - Releasing a file:
        - Deallocate blocks associated with the file and update availability.
5. Execute the operations and display the result.

Procedure:

1. Define the data structures for file and disk blocks.

2. Implement the functions for file allocation, release, and display.
3. Initialize the disk with a predefined number of blocks.
4. Allow the user to choose operations: allocate a file, release a file, or display allocations.
5. Simulate the operations and print results after each operation.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_BLOCKS 100
#define MAX_FILES 10
typedef struct Block {
    int id;
    struct Block* next;
} Block;
typedef struct File {
    char name[20];
    Block* start;
    Block* end;
} File;
Block disk[MAX_BLOCKS];
File files[MAX_FILES];
int blockStatus[MAX_BLOCKS];
int fileCount = 0;
void initializeDisk() {
    for (int i = 0; i < MAX_BLOCKS; i++) {
        disk[i].id = i;
        disk[i].next = NULL;
        blockStatus[i] = 0;
    }
}
```

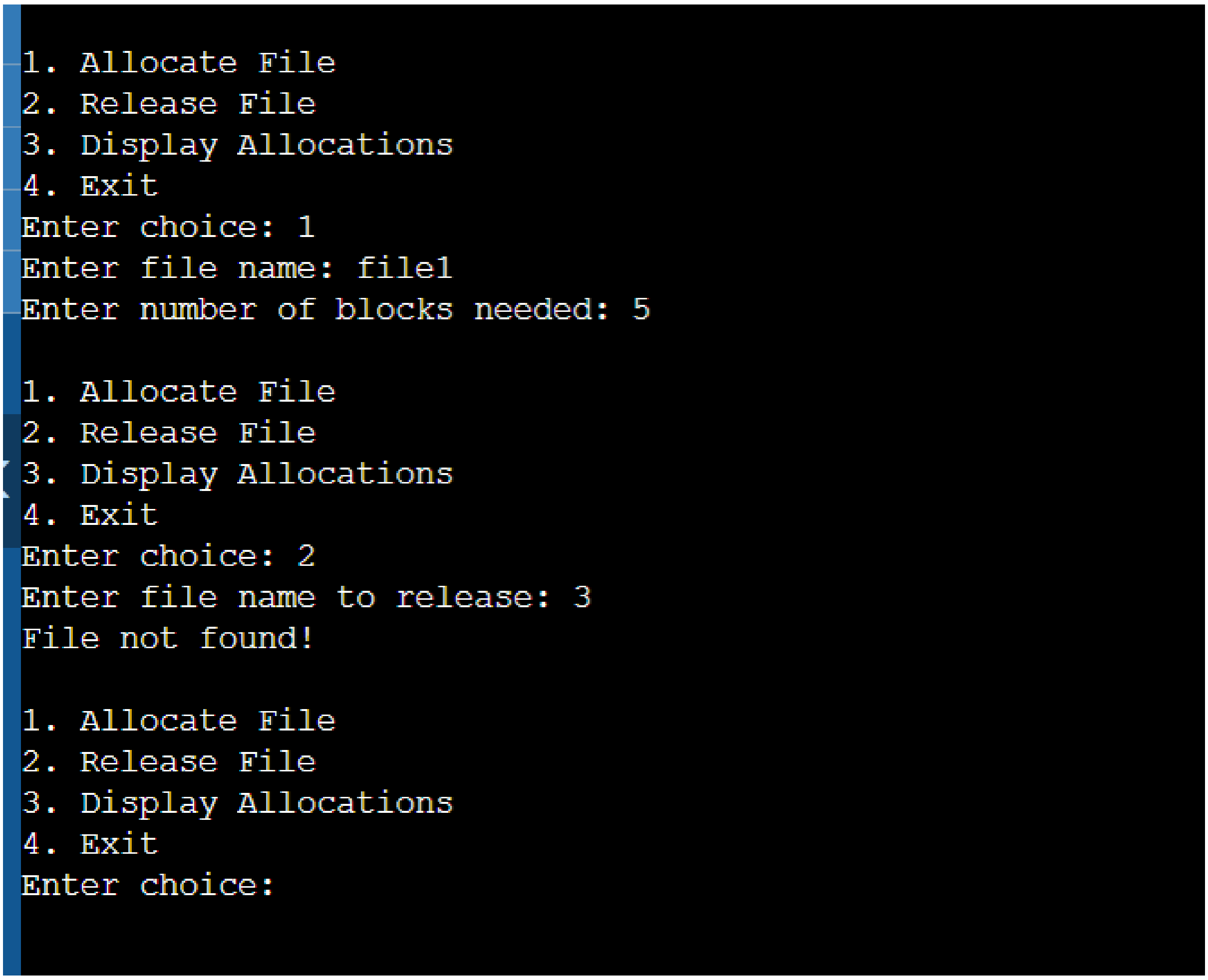
```
void allocateFile(char* fileName, int blocksNeeded) {
    if (fileCount >= MAX_FILES) {
        printf("File limit reached!\n");
        return;
    }
    int allocated = 0, firstBlock = -1, lastBlock = -1;
    for (int i = 0; i < MAX_BLOCKS && allocated < blocksNeeded; i++) {
        if (blockStatus[i] == 0) {
            blockStatus[i] = 1;
            if (firstBlock == -1) firstBlock = i;
            if (lastBlock != -1) disk[lastBlock].next = &disk[i];
            lastBlock = i;
            allocated++;
        }
    }
    if (allocated == blocksNeeded) {
        strcpy(files[fileCount].name, fileName);
        files[fileCount].start = &disk[firstBlock];
        files[fileCount].end = &disk[lastBlock];
        fileCount++;
    } else {
        printf("Not enough blocks available!\n");
        for (int i = firstBlock; i != -1 && i < MAX_BLOCKS; i = disk[i].next ? disk[i].next->id : -1)
            blockStatus[i] = 0;
    }
}
void releaseFile(char* fileName) {
    for (int i = 0; i < fileCount; i++) {
        if (strcmp(files[i].name, fileName) == 0) {
            Block* current = files[i].start;
            while (current) {
                blockStatus[current->id] = 0;
                current = current->next;
            }
            for (int j = i; j < fileCount - 1; j++)
                files[j] = files[j + 1];
            fileCount--;
            return;
        }
    }
    printf("File not found!\n");
}
void displayAllocations() {
    for (int i = 0; i < fileCount; i++) {
        printf("File: %s -> ", files[i].name);
        Block* current = files[i].start;
        while (current) {
            printf("%d ", current->id);
            current = current->next;
        }
        printf("\n");
    }
}
int main() {
    initializeDisk();
    int choice, blocks;
    char fileName[20];
    while (1) {
        printf("\n1. Allocate File\n2. Release File\n3. Display Allocations\n4. Exit\nEnter choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter file name: ");
                scanf("%s", fileName);
                printf("Enter number of blocks needed: ");
                scanf("%d", &blocks);
                allocateFile(fileName, blocks);
                break;
            case 2:
                printf("Enter file name to release: ");
                scanf("%s", fileName);
                releaseFile(fileName);
                break;
            case 3:
                displayAllocations();
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice!\n");
        }
    }
    return 0;
}
```

Result:

1. Files are allocated and managed as linked lists of disk blocks, with dynamic allocation and release operations successfully performed.

2. The disk block allocations and deallocations are displayed as per user input, verifying the linked allocation strategy.

Output:



```
1. Allocate File
2. Release File
3. Display Allocations
4. Exit
Enter choice: 1
Enter file name: file1
Enter number of blocks needed: 5

1. Allocate File
2. Release File
3. Display Allocations
4. Exit
Enter choice: 2
Enter file name to release: 3
File not found!

1. Allocate File
2. Release File
3. Display Allocations
4. Exit
Enter choice:
```