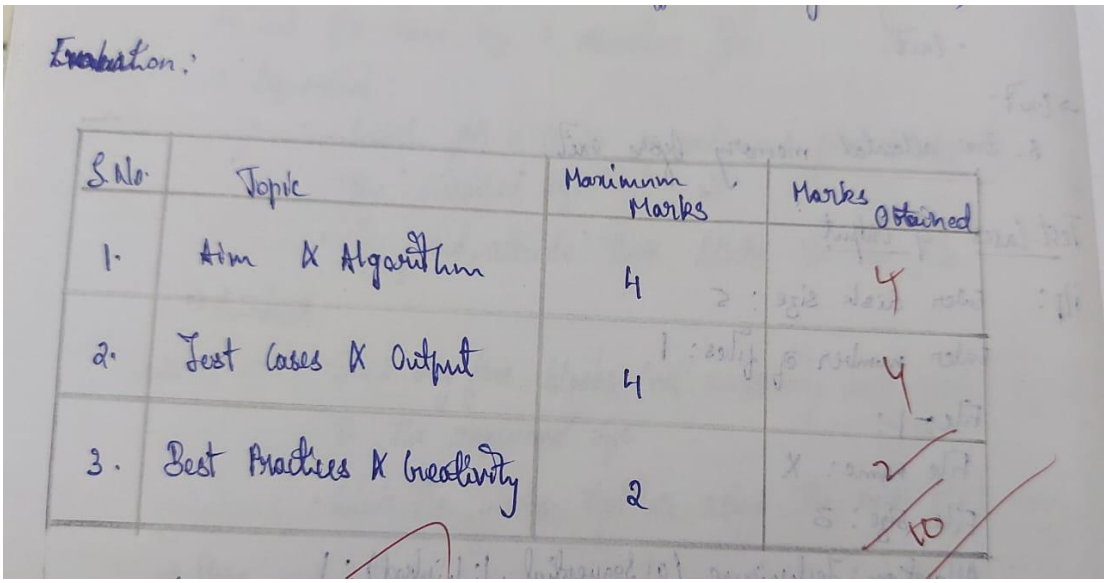
LMS Assignment Report

Department: Department of Computer Science & Engineering
Course: ICS1313 - Operating System Practices Laboratory
Assignment No: 11
Name: Simiyon Vinscent Samuel L
Reg No: 3122247001062

Assignment Title

Implementation of File Allocation Strategies (Sequential, Linked) (Exercise 11)

Evaluation Sheet



Evaluation:

S.No.	Topic	Maximum Marks	Marks Obtained
1.	Aim & Algorithm	4	4
2.	Test Cases & Output	4	4
3.	Best Practices & Creativity	2	2
			10

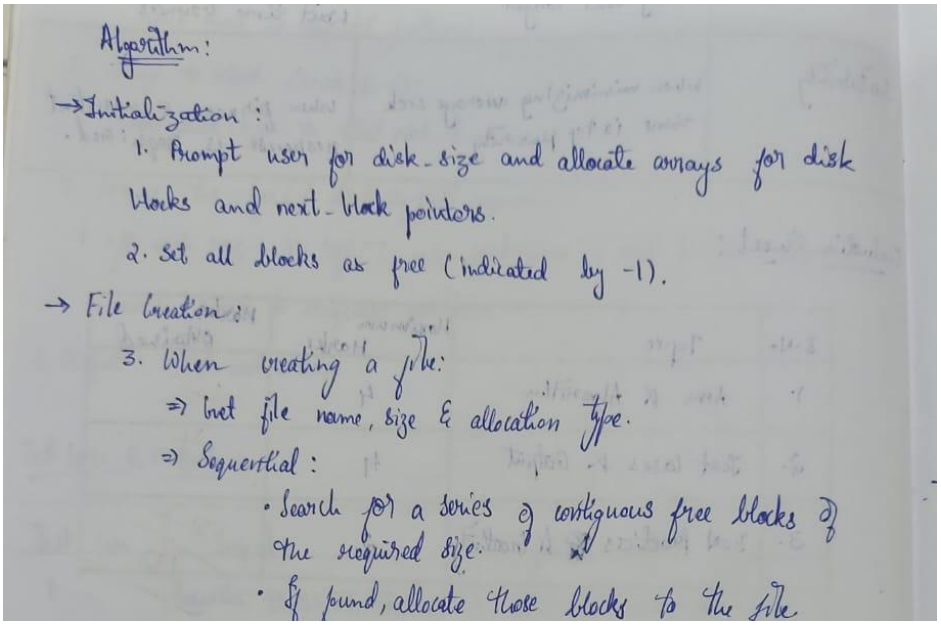
Question 1

File Allocation Methods Implementation

Aim

To implement and compare two file allocation strategies used in operating systems: Sequential (Contiguous) and Linked allocation methods.

Algorithm



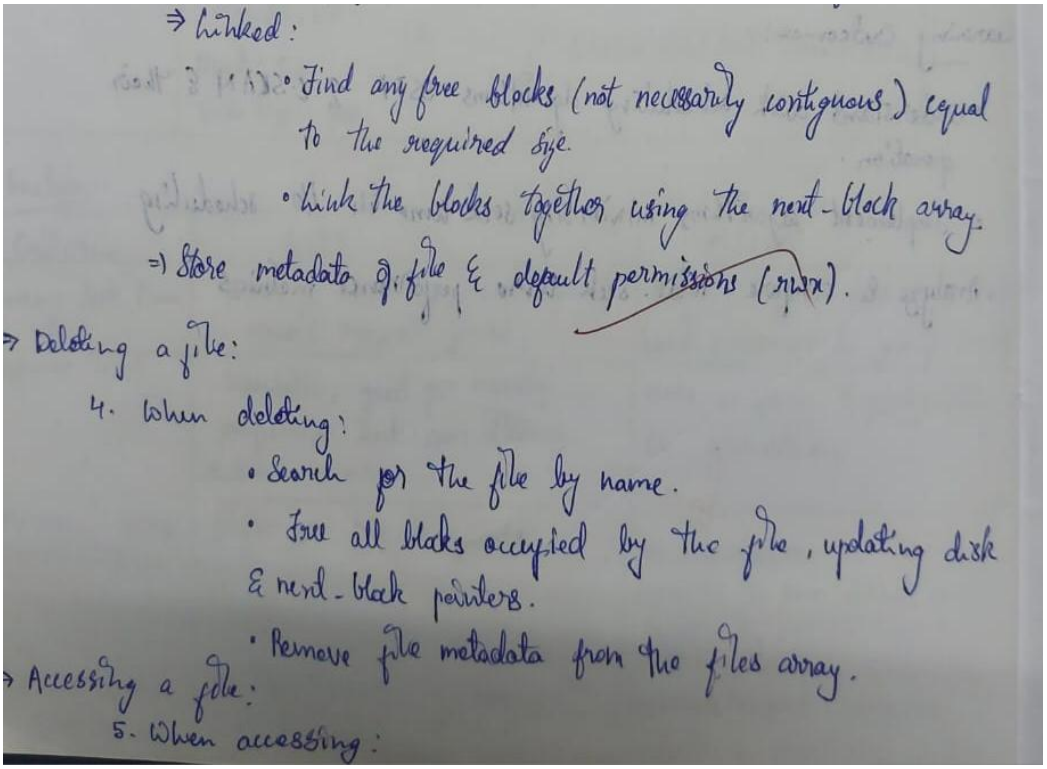
Algorithm:

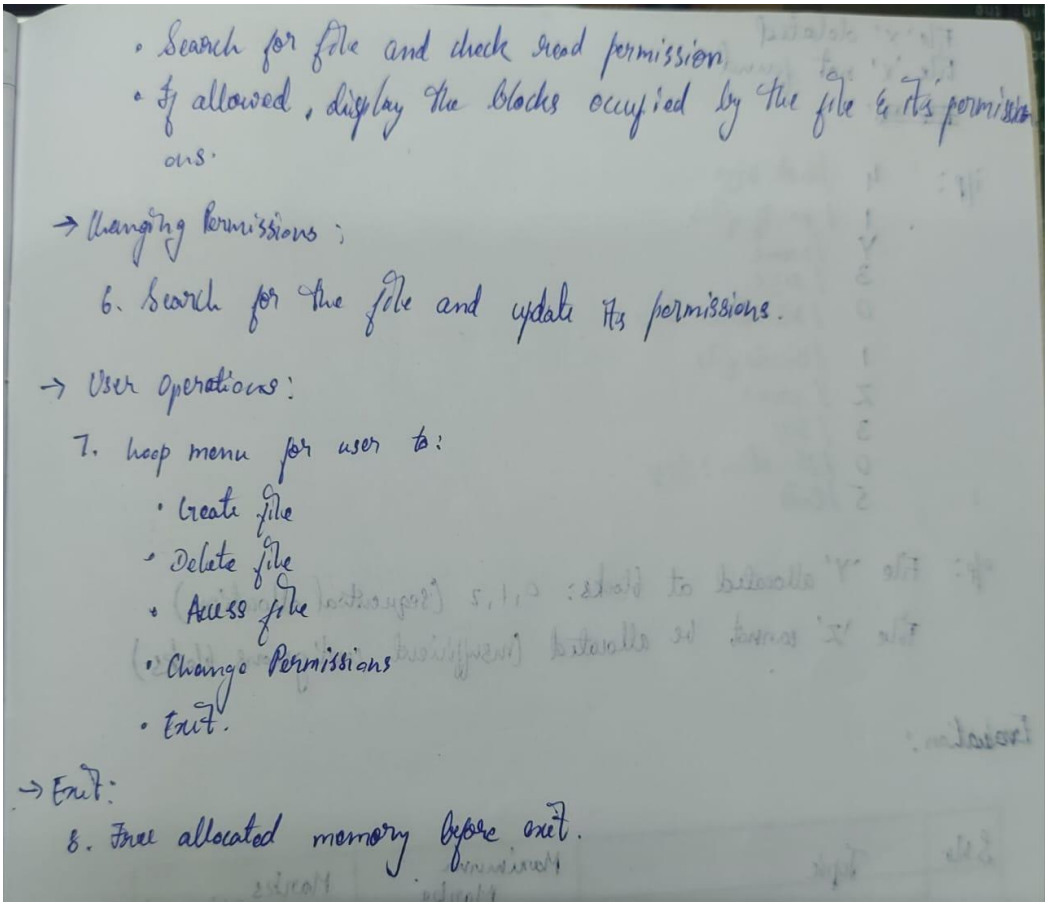
→ Initialization:

1. Prompt user for disk_size and allocate arrays for disk blocks and next_block pointers.
2. Set all blocks as free (indicated by -1).

→ File Creation:

3. When creating a file:
   ⇒ Get file name, size & allocation type.
   ⇒ Sequential:
   - Search for a series of contiguous free blocks of the required size.
   - If found, allocate those blocks to the file





Sequential Allocation

1. Input file name, starting block, and number of blocks.
2. Verify consecutive blocks from start are free.
3. Mark blocks occupied and record file entry.
4. Display allocated blocks.

Linked Allocation

1. Input file name and number of blocks.
2. Read block numbers, ensure each is free.
3. Allocate each block and link via pointers.
4. Store head pointer in directory.
5. Display linked block sequence.

Test Cases

Test Case	Method	Input	Expected Output
1	Sequential	File: F1, Start: 5, Blocks: 3	Blocks allocated: 5 6 7
2	Sequential	File: F2, Start: 8, Blocks: 4	Blocks allocated: 8 9 10 11
3	Linked	File: L1, Blocks: 4; Blocks: 2 5 9 12	Linked allocation done; 2→5→9→12
4	Linked	File: L2, Blocks: 3; Blocks: 7 14 20	Linked allocation done; 7→14→20

Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_FILES 10
#define MAX_NAME 20

typedef struct
{
    char name[MAX_NAME];
    int size;
    int start_block;
    int allocation_type; // 0: sequential, 1: linked
    int permissions;     // bitmask: 1=read, 2=write, 4=execute
} File;

int disk_size;
int *disk;
int *next_block;
File files[MAX_FILES];
int file_count = 0;

void init_disk();
int find_free_blocks(int num, int *blocks);
int allocate_sequential(int size, int *start);
int allocate_linked(int size, int *start);
int create_file(char *name, int size, int type);
void delete_file(char *name);
void access_file(char *name);
void display_permissions(File *f);
void change_permissions(char *name, int perm);

void init_disk()
{
```

```
    disk = (int *)malloc(disk_size * sizeof(int));
    next_block = (int *)malloc(disk_size * sizeof(int));
    for (int i = 0; i < disk_size; i++)
    {
        disk[i] = -1; // free
        next_block[i] = -1;
    }
}

int find_free_blocks(int num, int *blocks)
{
    int count = 0;
    for (int i = 0; i < disk_size && count < num; i++)
    {
        if (disk[i] == -1)
        {
            blocks[count++] = i;
        }
    }
    return count == num;
}

int allocate_sequential(int size, int *start)
{
    for (int i = 0; i <= disk_size - size; i++)
    {
        int contiguous = 1;
        for (int j = 0; j < size; j++)
        {
            if (disk[i + j] != -1)
            {
                contiguous = 0;
                break;
            }
        }
        if (contiguous)
        {
            *start = i;
            for (int j = 0; j < size; j++)
            {
                disk[i + j] = file_count;
            }
            return 1;
        }
    }
    return 0;
}

int allocate_linked(int size, int *start)
{
    int blocks[size];
    if (!find_free_blocks(size, blocks))
        return 0;
    *start = blocks[0];
```

```
    for (int i = 0; i < size; i++)
    {
        disk[blocks[i]] = file_count;
        if (i < size - 1)
        {
            next_block[blocks[i]] = blocks[i + 1];
        }
        else
        {
            next_block[blocks[i]] = -1;
        }
    }
    return 1;
}

int create_file(char *name, int size, int type)
{
    if (file_count >= MAX_FILES)
        return 0;
    int start;
    int success = 0;
    if (type == 0)
    { // sequential
        success = allocate_sequential(size, &start);
    }
    else
    { // linked
        success = allocate_linked(size, &start);
    }
    if (success)
    {
        strcpy(files[file_count].name, name);
        files[file_count].size = size;
        files[file_count].start_block = start;
        files[file_count].allocation_type = type;
        files[file_count].permissions = 7; // default rwx
        file_count++;
        printf("File '%s' allocated at blocks: ", name);
        int current = start;
        for (int i = 0; i < size; i++)
        {
            printf("%d", current);
            if (i < size - 1)
                printf(", ");
            if (type == 1)
                current = next_block[current];
            else
                current++;
        }
        printf(" (%s Allocation)\n", type == 0 ? "Sequential" : "Linked");
    }
    else
    {
```

```
        printf("File '%s' cannot be allocated (insufficient %s blocks)\n", name, type == 0
? "contiguous" : "");
    }
    return success;
}

void delete_file(char *name)
{
    int idx = -1;
    for (int i = 0; i < file_count; i++)
    {
        if (strcmp(files[i].name, name) == 0)
        {
            idx = i;
            break;
        }
    }
    if (idx == -1)
    {
        printf("File '%s' not found\n", name);
        return;
    }
    // Collect blocks first
    int blocks[files[idx].size];
    int current = files[idx].start_block;
    for (int i = 0; i < files[idx].size; i++)
    {
        blocks[i] = current;
        if (files[idx].allocation_type == 1)
            current = next_block[current];
        else
            current++;
    }
    // Free the blocks
    for (int i = 0; i < files[idx].size; i++)
    {
        disk[blocks[i]] = -1;
        next_block[blocks[i]] = -1;
    }
    // shift files
    for (int i = idx; i < file_count - 1; i++)
    {
        files[i] = files[i + 1];
    }
    file_count--;
    printf("File '%s' deleted\n", name);
}

void access_file(char *name)
{
    int idx = -1;
    for (int i = 0; i < file_count; i++)
    {
        if (strcmp(files[i].name, name) == 0)
```

```
        {
            idx = i;
            break;
        }
    }
    if (idx == -1)
    {
        printf("File '%s' not found\n", name);
        return;
    }
    if (!(files[idx].permissions & 1))
    {
        printf("Access denied: no read permission\n");
        return;
    }
    printf("File '%s' allocated at blocks: ", name);
    int current = files[idx].start_block;
    for (int i = 0; i < files[idx].size; i++)
    {
        printf("%d", current);
        if (i < files[idx].size - 1)
            printf(", ");
        if (files[idx].allocation_type == 1)
            current = next_block[current];
        else
            current++;
    }
    printf(" (%s Allocation)\n", files[idx].allocation_type == 0 ? "Sequential" :
"Linked");
    display_permissions(&files[idx]);
}

void display_permissions(File *f)
{
    printf("Permissions: ");
    if (f->permissions & 1)
        printf("r");
    else
        printf("-");
    if (f->permissions & 2)
        printf("w");
    else
        printf("-");
    if (f->permissions & 4)
        printf("x");
    else
        printf("-");
    printf("\n");
}

void change_permissions(char *name, int perm)
{
    int idx = -1;
    for (int i = 0; i < file_count; i++)
```

```
    {
        if (strcmp(files[i].name, name) == 0)
        {
            idx = i;
            break;
        }
    }
    if (idx == -1)
    {
        printf("File '%s' not found\n", name);
        return;
    }
    files[idx].permissions = perm;
    printf("Permissions for '%s' changed to: ", name);
    display_permissions(&files[idx]);
}

int main()
{
    printf("Enter disk size: ");
    scanf("%d", &disk_size);
    init_disk();
    int num_files;
    printf("Enter number of files: ");
    scanf("%d", &num_files);
    for (int i = 0; i < num_files; i++)
    {
        char name[MAX_NAME];
        int size, type;
        printf("File %d:\n", i + 1);
        printf("File name: ");
        scanf("%s", name);
        printf("File size: ");
        scanf("%d", &size);
        printf("Allocation Technique (0: Sequential, 1: Linked): ");
        scanf("%d", &type);
        create_file(name, size, type);
    }
    // Operations loop
    while (1)
    {
        printf("\nOperations:\n1. Create File\n2. Delete File\n3. Access File\n4. Change
Permissions\n5. Exit\n");
        int choice;
        scanf("%d", &choice);
        if (choice == 1)
        {
            char name[MAX_NAME];
            int size, type;
            printf("File name: ");
            scanf("%s", name);
            printf("File size: ");
            scanf("%d", &size);
            printf("Allocation Technique (0: Sequential, 1: Linked): ");
```

```
            scanf("%d", &type);
            create_file(name, size, type);
        }
        else if (choice == 2)
        {
            char name[MAX_NAME];
            printf("File name to delete: ");
            scanf("%s", name);
            delete_file(name);
        }
        else if (choice == 3)
        {
            char name[MAX_NAME];
            printf("File name to access: ");
            scanf("%s", name);
            access_file(name);
        }
        else if (choice == 4)
        {
            char name[MAX_NAME];
            int perm;
            printf("File name: ");
            scanf("%s", name);
            printf("New permissions (bitmask: 1=r, 2=w, 4=x, e.g. 7=rwx): ");
            scanf("%d", &perm);
            change_permissions(name, perm);
        }
        else if (choice == 5)
        {
            break;
        }
    }
    free(disk);
    free(next_block);
    return 0;
}
```

Output Screenshots

```
ks_vijay-1401@DESKTOP-J8G3TP8:~$ cc file_allocation.c -o file
ks_vijay-1401@DESKTOP-J8G3TP8:~$ ./file
Enter disk size: 10
Enter number of files: 3
File 1:
File name: A
File size: 4
Allocation Technique (0: Sequential, 1: Linked): 0
File 'A' allocated at blocks: 0, 1, 2, 3 (Sequential Allocation)
File 2:
File name: B
File size: 2
Allocation Technique (0: Sequential, 1: Linked): 1
File 'B' allocated at blocks: 4, 5 (Linked Allocation)
File 3:
File name: C
File size: 2
Allocation Technique (0: Sequential, 1: Linked): 0
File 'C' allocated at blocks: 6, 7 (Sequential Allocation)

Operations:
1. Create File
2. Delete File
3. Access File
4. Change Permissions
5. Exit
2
File name to delete: B
File 'B' deleted
```

```
Operations:
1. Create File
2. Delete File
3. Access File
4. Change Permissions
5. Exit
3
File name to access: B
File 'B' not found

Operations:
1. Create File
2. Delete File
3. Access File
4. Change Permissions
5. Exit
3
File name to access: A
File 'A' allocated at blocks: 0, 1, 2, 3 (Sequential Allocation)
Permissions: rwx

Operations:
1. Create File
2. Delete File
3. Access File
4. Change Permissions
5. Exit
4
File name: A
New permissions (bitmask: 1=r, 2=w, 4=x, e.g. 7=rwx): 6
Permissions for 'A' changed to: Permissions: -wx
```

```
Operations:
1. Create File
2. Delete File
3. Access File
4. Change Permissions
5. Exit
3
File name to access: A
Access denied: no read permission

Operations:
1. Create File
2. Delete File
3. Access File
4. Change Permissions
5. Exit
1
File name: B
File size: 4
Allocation Technique (0: Sequential, 1: Linked): 1
File 'B' allocated at blocks: 4, 5, 8, 9 (Linked Allocation)

Operations:
1. Create File
2. Delete File
3. Access File
4. Change Permissions
5. Exit
5
ks_vijay-1401@DESKTOP-J8G3TP8:~$
```

Learning Outcomes

- Implemented sequential and linked file allocation methods in C.
- Understood contiguous vs non-contiguous block management.
- Learned dynamic memory and pointer usage for linked lists.
- Gained insights into directory entries and block tracking.
- Enhanced skills in OS disk space allocation techniques.