File Space Allocation - Methods - Contiguous, Indexed, and Linked

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100

int disk[MAX_SIZE];

// Linked Allocation Structures
typedef struct node {
    int block_index;
    struct node *next;
} BlockNode;
BlockNode *free_list = NULL;

// Function to initialize disk
void initializeDisk() {
    for (int i = 0; i < MAX_SIZE; i++)
        disk[i] = -1; // -1 means free
}

// Contiguous Allocation
void contiguousAllocation(int start, int length) {
    int flag = 0;
    for (int i = start; i < start + length && i < MAX_SIZE; i++) {
        if (disk[i] != -1) {
            flag = 1;
            break;
        }
    }
    if (!flag) {
        for (int i = start; i < start + length && i < MAX_SIZE; i++)
            disk[i] = 1;
        printf("File allocated from %d to %d\n", start, start + length - 1);
    } else {
        printf("Contiguous blocks not available.\n");
    }
}

// Indexed Allocation
void indexedAllocation(int indexBlock, int blocks[], int n) {
    if (disk[indexBlock] != -1) {
        printf("Index block already allocated.\n");
        return;
```

```
    }
    disk[indexBlock] = 1;
    printf("Index block %d allocated. Points to blocks: ", indexBlock);
    for (int i = 0; i < n; i++) {
        if (blocks[i] < MAX_SIZE && disk[blocks[i]] == -1) {
            disk[blocks[i]] = 1;
            printf("%d ", blocks[i]);
        } else {
            printf("\nBlock %d not available.\n", blocks[i]);
        }
    }
    printf("\n");
}

// Linked Allocation
void buildFreeList() {
    BlockNode *temp = NULL;
    for (int i = 0; i < MAX_SIZE; i += 5) { // Simulate some blocks as free
        BlockNode *new_node = (BlockNode *)malloc(sizeof(BlockNode));
        new_node->block_index = i;
        new_node->next = temp;
        temp = new_node;
    }
    free_list = temp;
}

int allocateLinked(int num_blocks) {
    if (free_list == NULL) {
        printf("Error: Disk is full.\n");
        return -1;
    }
    int start_block = free_list->block_index;
    BlockNode *current = free_list;
    int count = 0;
    while (current != NULL && count < num_blocks) {
        disk[current->block_index] = 1;
        count++;
        current = current->next;
    }
    if (count < num_blocks) {
        printf("Error: Not enough free space.\n");
        return -1;
    }
    free_list = current;
```

```
    return start_block;
}

void displayDisk() {
    printf("Disk Status:\n");
    for (int i = 0; i < MAX_SIZE; i++)
        printf("%d ", disk[i]);
    printf("\n");
}

int main() {
    initializeDisk();
    buildFreeList();

    int choice;
   while(1)
   {
    printf("Select File Space allocation method:\n");
    printf("1. Contiguous\n2. Indexed\n3. Linked\nEnter choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1: {
            int start, length;
            printf("Enter starting block (0-%d): ", MAX_SIZE - 1);
            scanf("%d", &start);
            printf("Enter length: ");
            scanf("%d", &length);
            contiguousAllocation(start, length);
            break;
        }
        case 2: {
            int indexBlock, n;
            printf("Enter index block (0-%d): ", MAX_SIZE - 1);
            scanf("%d", &indexBlock);
            printf("Enter number of file blocks: ");
            scanf("%d", &n);
            int blocks[n];
            printf("Enter block numbers: ");
            for (int i = 0; i < n; i++)
                scanf("%d", &blocks[i]);
            indexedAllocation(indexBlock, blocks, n);
            break;
        }
```

```
        case 3: {
            int num_blocks;
            printf("Enter number of blocks for linked allocation: ");
            scanf("%d", &num_blocks);
            int allocated = allocateLinked(num_blocks);
            if (allocated != -1)
                printf("Linked allocation successful. Starting block: %d\n", allocated);
            break;
        }
        default:
            printf("Invalid choice.\n");
    }
    displayDisk();
}
    return 0;
}
```

Description

File Allocation Methods

The allocation methods define how the files are stored in the disk blocks.

There are three main disk space or file allocation methods.

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

The main idea behind these methods is to provide Efficient disk space utilization & Fast access to the file blocks.

Contiguous Allocation

In this scheme, each file occupies a contiguous set of blocks on the disk. This means that given the starting block address and the length of the file (in terms of blocks required), we can determine the blocks occupied by the file. The directory entry for a file with contiguous allocation contains:

- Address of starting block
- Length of the allocated portion.
- **Example:** If a file requires n blocks and is given a block b as the starting location, then the blocks assigned to the file will be: *b, b+1, b+2,.....b+n-1.*

Advantages

- Both the Sequential and Direct Accesses are supported by this. For direct access, the address of the kth block of the file which starts at block b can easily be obtained as
- $(b+k)$
- $(b+k)$.
- This is extremely fast since the number of seeks are minimal because of contiguous allocation of file blocks.

Disadvantages

- This method suffers from both internal and external fragmentation. This makes it inefficient in terms of memory utilization.

- Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.

Linked Allocation

In this scheme, each file is a linked list of disk blocks which need not be contiguous. Here,

- The disk blocks can be scattered anywhere on the disk. The directory entry contains a pointer to the starting and the ending file block.
- Each block contains a pointer to the next block occupied by the file.

Advantages

- This is very flexible in terms of file size. File size can be increased easily since the system does not have to look for a contiguous chunk of memory.
- This method does not suffer from external fragmentation. This makes it relatively better in terms of memory utilization.

Disadvantages

- Because the file blocks are distributed randomly on the disk, a large number of seeks are needed to access every block individually. This makes linked allocation slower.

- It does not support random or direct access. We can not directly access the blocks of a file. A block k of a file can be accessed by traversing k blocks sequentially (sequential access) from the starting block of the file via block pointers.
- Pointers required in the linked allocation incur some extra overhead.

Indexed Allocation

In this scheme, a special block known as the Index block contains the pointers to all the blocks occupied by a file. Here,

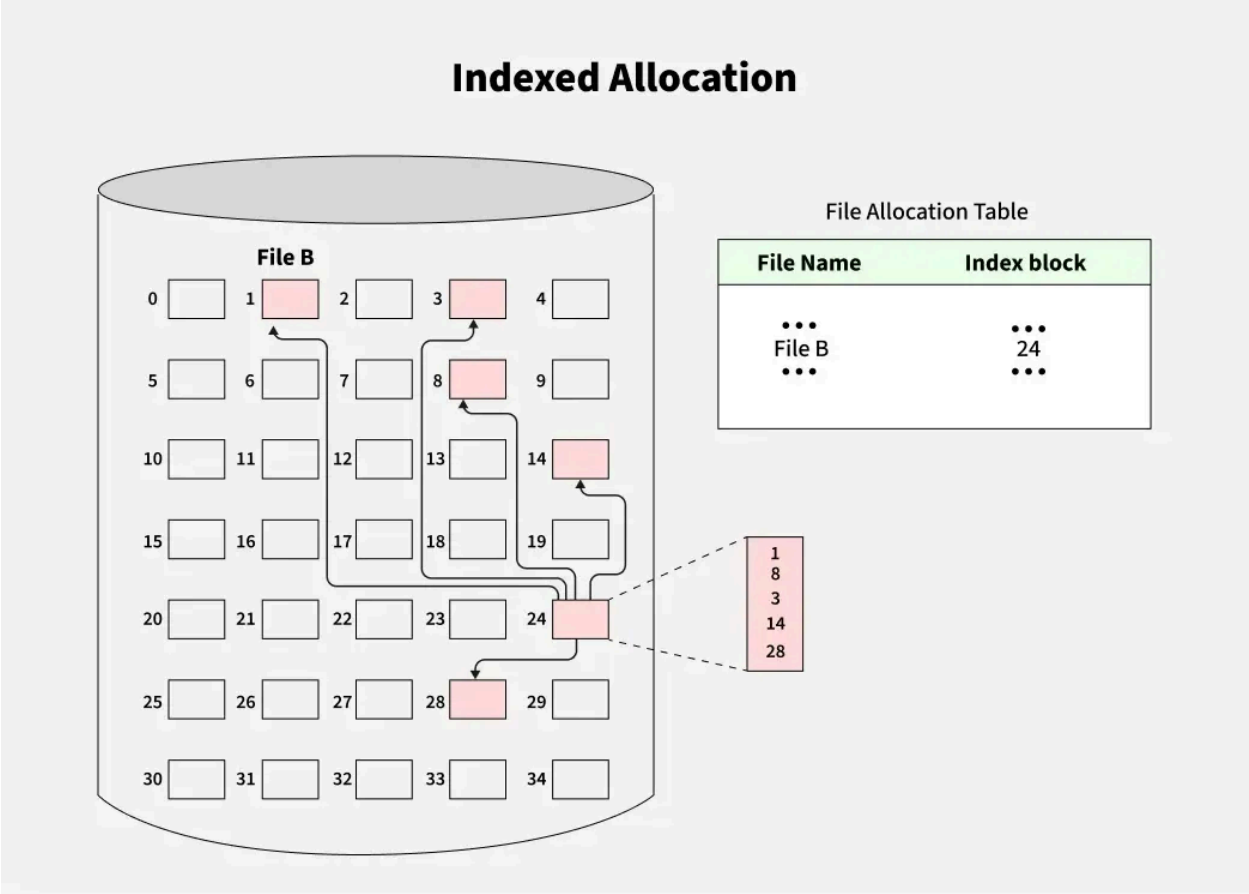
- Each file has its own index block.
- The i^{th} entry in the index block contains the disk address of the i^{th} file block.
- The directory entry contains the address of the index block as shown in the image.

Advantages

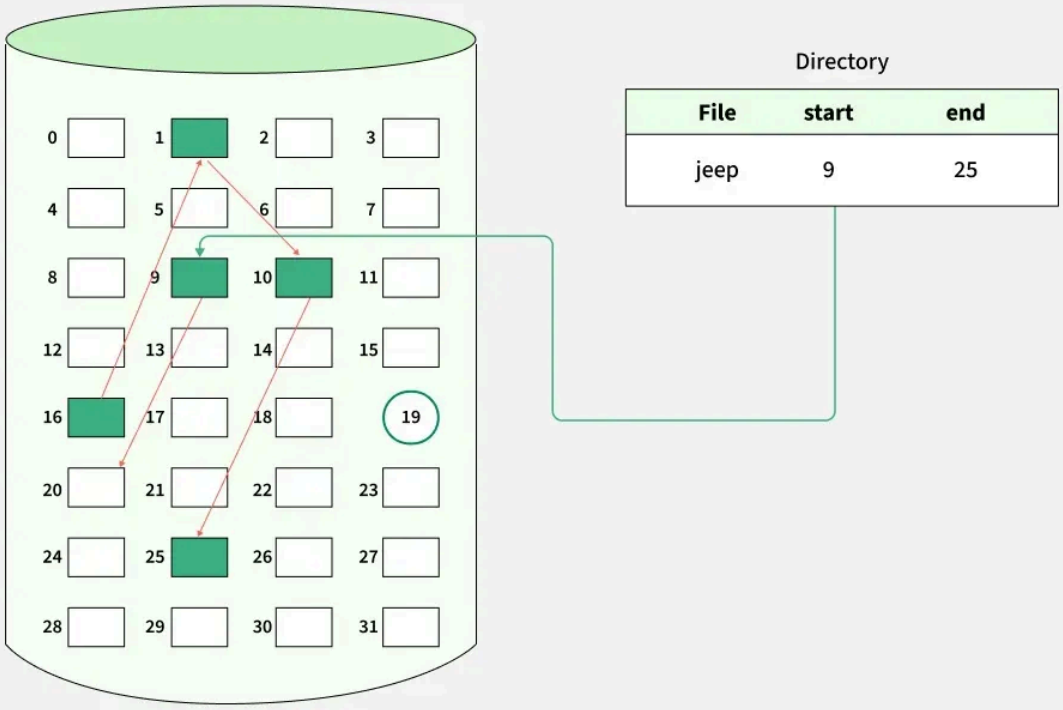
- This supports direct access to the blocks occupied by the file and therefore provides fast access to the file blocks.
- It overcomes the problem of external fragmentation.

Disadvantages

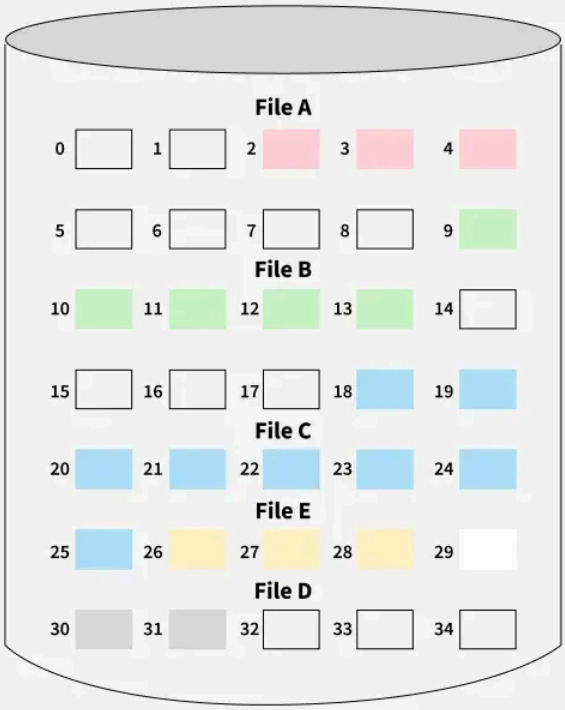
- The pointer overhead for indexed allocation is greater than linked allocation.
- For very small files, say files that expand only 2-3 blocks, the indexed allocation would keep one entire block (index block) for the pointers which is inefficient in terms of memory utilization. However, in linked allocation we lose the space of only 1 pointer per block.



Linked Allocation



Continuous Allocation



File Name	Star block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3