UCS 1411 - Operating Systems Lab

Exercise 12 – File Allocation Techniques

Mahesh Bharadwaj K - 185001089

To develop a C program to implement the various file allocation techniques.

Linked List Header File

```
typedef Block Data;

typedef struct Node
{
    Data d;
    struct Node *next;
} Node;

typedef Node *List;

extern void init_block(Block *const);

List createEmptyList()
{
    Node *head = (Node *)malloc(sizeof(Node));
    init_block(&(head -> d));
    head->next = NULL;
    return head;
}

void insertLast(List head, const Data d)
{
    Node *new = (Node *)malloc(sizeof(Node));
    new->d = d;
    Node *tmp = head;

    while (tmp->next)
        tmp = tmp->next;

    new->next = NULL;
    tmp->next = new;
}

void insertFirst(List head, const Data d)
{
    Node *new = (Node *)malloc(sizeof(Node));
    new->d = d;

    new->next = head->next;
    head->next = new;
}

Data delete (List prev)
```

```
{
    Data rVal;
    if (!prev)
        return rVal;
    if (!prev->next)
        return rVal;

    Node *tmp = prev->next;
    rVal = tmp->d;
    prev->next = prev->next->next;
    free(tmp);

    return rVal;
}

Data deleteFirst(List head)
{
    Data rVal;
    if (head->next == NULL)
    {
        printf(" Empty List!\n");
        return rVal;
    }

    delete (head);
}

Data deleteLast(List head)
{
    Data rVal;
    if (head->next == NULL)
    {
        printf(" Empty List!\n");
        return rVal;
    }

    Node *tmp = head;
    while (tmp->next->next != NULL)
        tmp = tmp->next;

    delete (tmp);
}

void display(List head)
{
    Node *tmp = head->next;

    if (tmp == NULL)
    {
        printf(" Empty!\n");
        return;
    }

    while (tmp)
    {
        printf(" BID: %-2d\tStatus: %d\n", tmp->d.id, tmp->d.status);
        tmp = tmp->next;
    }
```

```
}

int length(List head)
{
    Node *tmp = head->next;
    if (tmp == NULL)
        return 0;

    int count = 0;
    while (tmp)
    {
        tmp = tmp->next;
        count++;
    }
    return count;
}

Node* search(List head, const int id)
{

    if (head->next == NULL)
        return NULL;

    Node *tmp = head -> next;
    while (tmp)
    {
        if (tmp->d.id == id)
            return tmp;
        tmp = tmp->next;
    }

    return NULL;
}
```

Main Program

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

#define MAX 100
#define FREE 0

typedef struct File
{
    char name[21];
    int size;
    int start_block;
    int end_block;
    int *indices;
    int length;
} File;

void init_file(File *const);

typedef struct Directory
```

```
{
    File f[MAX];
    int size;
} Directory;

void init_dir(Directory *const);

typedef struct Block
{
    int id;
    unsigned status : 1;
    struct Block *next_file_blk;
} Block;

void init_block(Block *const);

#include "LinkedList.h"

void contiguous(File *const, const int, const int, const int);
void linked(File *const, const int, const int, const int);
void indexed(File *const, const int, const int, const int);

int main()
{
    int mem_size;
    int blk_size;
    int num_blks;
    int num_file;
    int choice;

    File f[MAX];

    printf(" Enter the size of memory: ");
    scanf("%d", &mem_size);
    printf(" Enter the size of block: ");
    scanf("%d", &blk_size);
    num_blks = mem_size / blk_size;

    printf(" Enter the number of files: ");
    scanf("%d", &num_file);
    getchar();

    for (int i = 0; i < num_file; i++)
    {
        printf(" Enter the name of file: ");
        scanf("%[^\n]", f[i].name);

        printf(" Enter the size of file: ");
        scanf("%d", &f[i].size);
        getchar();
    }

    while (1)
    {
        printf("\t\t\tFILE ALLOCATION TECHNIQUES\n");
        printf(" 1 - Contiguous\n");
        printf(" 2 - Linked\n");
        printf(" 3 - Indexed\n");
```

```
        printf(" 0 - Exit\n");
        printf(" ----------------------\n");
        printf(" Enter your choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
        case 0:
            exit(0);
        case 1:
            contiguous(f, num_file, blk_size, num_blks);
            break;
        case 2:
            linked(f, num_file, blk_size, num_blks);
            break;
        case 3:
            indexed(f, num_file, blk_size, num_blks);
            break;

        default:
            printf(" Invalid Input!\n");
        }
    }
}

void init_file(File *const f)
{
    strcpy(f->name, "");
    f->start_block = -1;
    f->end_block = -1;
    f->size = -1;
    f->indices = NULL;
    f->length = -1;
}

void init_dir(Directory *const d)
{
    d->size = 0;
    for (int i = 0; i < MAX; i++)
        init_file(&(d->f[i]));
}

void init_block(Block *const b)
{
    b->status = FREE;
    b->id = -1;
    b->next_file_blk = NULL;
}

void contiguous(File *const f, const int n_files, const int blk_size, const int
↪  num_blk)
{
    List list = createEmptyList();

    Block b;
    init_block(&b);

    Node *ptr, *tmp;
```

```
int blocks_visited, flag, id, counter, blk_req;
int start, end;

for (int i = 0; i < num_blk; i++)
{
    b.id = i;
    insertLast(list, b);
}

for (int i = 0; i < n_files; i++)
{
    blocks_visited = 0;
    flag = 0;
    blk_req = f[i].size / blk_size;
    if (f[i].size % blk_size)
        blk_req++;

    while (blocks_visited < num_blk && !flag)
    {
        id = random() % num_blk;
        ptr = search(list, id);
        if (ptr->d.status != FREE)
        {
            blocks_visited++;
            continue;
        }

        counter = 0;

        start = ptr->d.id;

        tmp = ptr;
        while (tmp)
        {
            if (tmp->d.status == FREE)
            {
                counter++;
                if (counter == blk_req)
                {
                    flag = 1;
                    break;
                }
            }
            else
                break;
            tmp = tmp->next;
        }

        if (flag)
        {
            f[i].start_block = start;
            f[i].length = blk_req;
            tmp = ptr;
            for (int i = 0; i < blk_req; i++)
            {
                tmp->d.status = 1;
                tmp = tmp->next;
```

```
                }
            }
            else
                blocks_visited++;
        }
        if (!flag)
            printf(" Unable to allocate file: %s\n!", f[i].name);
    }

    printf("\n\t\tDIRECTORY STRUCTURE\n");
    printf(" +----------------------+-------+--------+\n");
    printf(" |      File Name       | Start | Length |\n");
    printf(" +----------------------+-------+--------+\n");

    for (int i = 0; i < n_files; i++)
        if (f[i].length > 0)
            printf(" | %-20s | %-5d | %-6d |\n", f[i].name, f[i].start_block,
             ↪  f[i].length);
    printf(" +----------------------+-------+--------+\n");
}
void linked(File *const f, const int n_files, const int blk_size, const int num_blk)
{
    List list = createEmptyList();

    Block b;
    init_block(&b);

    Node *ptr, *tmp, *left, *right;

    int blocks_visited, flag, id, counter, blk_req;

    for (int i = 0; i < num_blk; i++)
    {
        b.id = i;
        insertLast(list, b);
    }

    for (int i = 0; i < n_files; i++)
    {
        counter = 0;
        blocks_visited = 0;
        flag = 0;
        blk_req = f[i].size / blk_size;
        if (f[i].size % blk_size)
            blk_req++;
        int *allocated = (int *)calloc(blk_req, sizeof(int));

        while (blocks_visited < num_blk && !flag)
        {
            id = random() % num_blk;
            ptr = search(list, id);

            if (ptr->d.status != FREE)
            {
                blocks_visited++;
                continue;
            }
            ptr -> d.status = 1;
```

```
            allocated[counter++] = id;

            if (counter == blk_req)
                flag = 1;
        }
        if (!flag){
            printf(" Unable to allocate file: %s\n", f[i].name);
            for(int i = 0; i < counter; i++){
                ptr = search(list, allocated[i]);
                ptr -> d.status = FREE;
            }
            free(allocated);
        }
        else
        {

            f[i].start_block = allocated[0];
            f[i].end_block = allocated[blk_req - 1];
            f[i].length = blk_req;
            for (int i = 0; i < blk_req - 1; i++)
            {

                left = search(list, allocated[i]);
                right = search(list, allocated[i + 1]);
                left->d.next_file_blk = &(right->d);
                left->d.status = 1;
            }
            right->d.next_file_blk = NULL;
            free(allocated);
        }
    }
    printf("\n\t\tDIRECTORY STRUCTURE\n");
    printf(" +----------------------+-------------+-----------+\n");
    printf(" |      File Name       | Start Block | End Block |\n");
    printf(" +----------------------+-------------+-----------+\n");
    for (int i = 0; i < n_files; i++)
        if (f[i].end_block >= 0)
            printf(" | %-20s |     %-2d      |    %-2d     |\n",
                   f[i].name, f[i].start_block, f[i].end_block);
    printf(" +----------------------+-------------+-----------+\n");

    printf("\n");
    for (int i = 0; i < n_files; i++)

        if (f[i].start_block >= 0)
        {
            printf("\n\n File Name: %s\n ",f[i].name);
            ptr = search(list, f[i].start_block);
            Block *b = &(ptr->d);
            while (b)
            {
                printf("%-2d ", b->id);
                b = b->next_file_blk;
            }
        }
}
void indexed(File *const f, const int n_files, const int blk_size, const int num_blk)
{
```

```
List list = createEmptyList();

Block b;
init_block(&b);

Node *ptr, *tmp;

int blocks_visited, flag, id, counter, blk_req;
int start, end;

for (int i = 0; i < num_blk; i++)
{
    b.id = i;
    insertLast(list, b);
}

for (int i = 0; i < n_files; i++)
{
    blocks_visited = 0;
    flag = 0;
    blk_req = f[i].size / blk_size;
    if (f[i].size % blk_size)
        blk_req++;
    f[i].indices = (int *)calloc(blk_req + 1, sizeof(int));
    f[i].length = blk_req;
    counter = 0;
    while (blocks_visited < num_blk && !flag)
    {
        id = random() % num_blk;
        ptr = search(list, id);
        if (ptr->d.status == FREE)
        {
            f[i].indices[counter++] = id;
            if (counter == blk_req + 1)
            {
                flag = 1;
                break;
            }
        }
        else
            blocks_visited++;
    }
    if (!flag)
    {
        printf(" Unable to allocate memory for file: %s\n", f[i].name);
        free(f[i].indices);
        f[i].indices = NULL;
    }
}
printf("\n\t\tDIRECTORY STRUCTURE\n");
printf(" +----------------------+-------------+\n");
printf(" |      File Name       | Index Block |\n");
printf(" +----------------------+-------------+\n");
for (int i = 0; i < n_files; i++)
    if (f[i].indices)
        printf(" | %-20s |     %-2d      |\n", f[i].name, f[i].indices[0]);
printf(" +----------------------+-------------+\n");
```

```
    printf("\n\n");
    printf(" +----------------------+---------------+\n");
    printf(" |      File Name       | Blocks Indexed |\n");
    printf(" +----------------------+---------------+\n");
    for (int i = 0; i < n_files; i++)
    {
        if (f[i].indices)
        {
            for (int j = 1; j <= f[i].length; j++)
                printf(" | %-20s |       %-2d       |\n", ((j > 1) ? "" : f[i].name),
                ↪  f[i].indices[j]);
        }
        printf(" +----------------------+---------------+\n");
    }
}
```

Output

```
Enter the size of memory: 500
 Enter the size of block: 10
 Enter the number of files: 4
 Enter the name of file: file1.txt
 Enter the size of file: 53
 Enter the name of file: temp.bin
 Enter the size of file: 124
 Enter the name of file: output.pdf
 Enter the size of file: 32
 Enter the name of file: prog.c
 Enter the size of file: 22
FILE ALLOCATION TECHNIQUES
 1 - Contiguous
 2 - Linked
 3 - Indexed
 0 - Exit
 ----------------------
 Enter your choice: 1

DIRECTORY STRUCTURE
 +----------------------+-------+--------+
 |      File Name       | Start | Length |
 +----------------------+-------+--------+
 | file1.txt            | 33    | 6      |
 | temp.bin             | 15    | 13     |
 | output.pdf           | 43    | 4      |
 | prog.c               | 12    | 3      |
 +----------------------+-------+--------+
FILE ALLOCATION TECHNIQUES
 1 - Contiguous
 2 - Linked
 3 - Indexed
 0 - Exit
 ----------------------
 Enter your choice: 2

DIRECTORY STRUCTURE
 +----------------------+-------------+-----------+
 |      File Name       | Start Block | End Block |
```

```
+----------------------+-------------+-----------+
| file1.txt            |     27      |    22     |
| temp.bin             |     36      |    19     |
| output.pdf           |     43      |    21     |
| prog.c               |     34      |    48     |
+----------------------+-------------+-----------+



File Name: file1.txt
27 40 9  13 26 22

File Name: temp.bin
36 11 18 17 29 32 30 12 23 35 2  8  19

File Name: output.pdf
43 6  42 21

File Name: prog.c
34 37 48
           FILE ALLOCATION TECHNIQUES
1 - Contiguous
2 - Linked
3 - Indexed
0 - Exit
----------------------
Enter your choice: 3

DIRECTORY STRUCTURE
+----------------------+-------------+
|      File Name       | Index Block |
+----------------------+-------------+
| file1.txt            |     24      |
| temp.bin             |     6       |
| output.pdf           |     13      |
| prog.c               |     45      |
+----------------------+-------------+


+----------------------+----------------+
|      File Name       | Blocks Indexed |
+----------------------+----------------+
| file1.txt            |       15       |
|                      |       20       |
|                      |       13       |
|                      |       26       |
|                      |       41       |
|                      |       30       |
+----------------------+----------------+
| temp.bin             |       23       |
|                      |       12       |
|                      |       20       |
|                      |       46       |
|                      |       31       |
|                      |       5        |
|                      |       25       |
|                      |       34       |
|                      |       27       |
```

```
 |                      |       36       |
 |                      |       5        |
 |                      |       46       |
 |                      |       29       |
 +----------------------+----------------+
 | output.pdf           |       7        |
 |                      |       24       |
 |                      |       45       |
 |                      |       32       |
 +----------------------+----------------+
 | prog.c               |       14       |
 |                      |       17       |
 |                      |       34       |
 +----------------------+----------------+
FILE ALLOCATION TECHNIQUES
 1 - Contiguous
 2 - Linked
 3 - Indexed
 0 - Exit
 ----------------------
 Enter your choice: 0
```
