SSN COLLEGE OF ENGINEERING, KALAVAKKAM
(An Autonomous Institution, Affiliated to Anna University, Chennai)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UCS1411 - OPERATING SYSTEMS LAB

LAB EXERCISE 12

File Allocation Techniques

Submission Date:30-05-2022

Name: Jayannthan P T *Dept: CSE 'A'* *Roll No.: 205001049*

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

Algorithm:

```
1. Get Main memory size and block size as input.
2. Create a Main memory with 'n' number of blocks of equal size.
3. Main memory is maintained as Linked List with structure containing block id, Free /
Filename, Link to next Memory block , Link to Next File block (only for Linked Allocation),
File block table( integer array to hold block numbers only for Indexed Allocation)
4. Get the number of files and their size as input.
5. Calculate the no. of blocks needed for each file.
6. Select the Allocation Algorithm. – For every algorithm display Directory information and
File information.
7. For Contiguous Allocation - For each file do the following
    i. Generate a random number between 1 to 'n'
    ii. Check for continuous number of needed file free blocks starting from that random
    block
    no.
    iii. If free then allot that file in those continuous blocks and update the directory
    structure.
    iv. else repeat step 1
    v. If no continuous blocks are free then 'no enough memory error'
    vi. The Directory Structure should contain Filename, Starting Block, length (no. of
    blocks)
8. For Linked Allocation- For each file do the following
    i. Generate a random number between 1 to 'n' blocks.
    ii. Check that block is free or not.
    iii. If free then allot it for file. Repeat step 1 to 3 for the needed number of blocks for
    file and create linked list in Main memory using the field "Link to Next File block".
    iv. Update the Directory entry which contains Filename, Start block number, Ending
    Block Number.
    v. Display the file blocks starting from start block number in Directory upto ending
    block number by traversing the Main memory Linked list using the field "Link to Next
    File block.
```

9.For Indexed Allocation - For each file do the following

i.Generate a random number between 1 to ‘n’ blocks for index block.

ii. Check if it is free else repeat index block selection

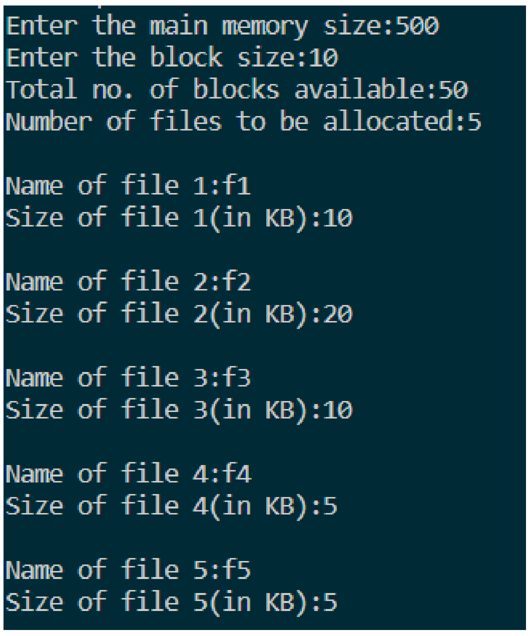
iii. Generate needed number of free blocks in random order for the file and store those block numbers in index block as array in File block table array.

Code:

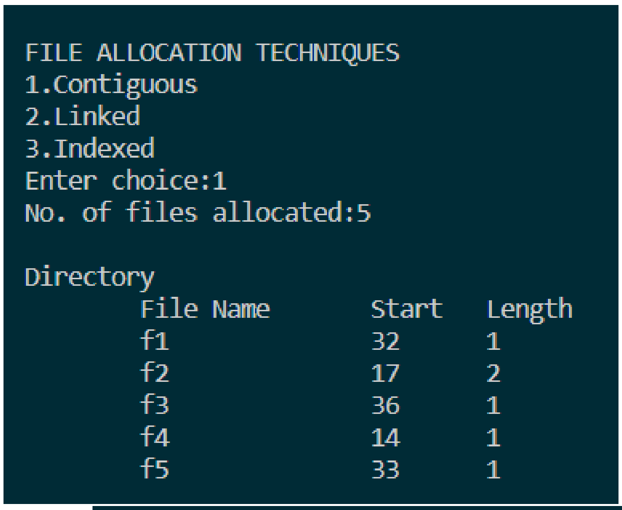
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <math.h>
typedef struct
{
    char fname[20];
    int start;
    int length;
    int end;
    struct dir *next;
} dir;
typedef struct
{
    char fname[20];
    int blockid;
    int fileblocktable[100];
    struct node *next;
    struct node *link;
} node;
typedef struct
{
    char fname[20];
    int fsize;
    int fileblocks;
} fileDetails;

void insert(node *head, node data)
{
    node *t;
    node *newnode;
    newnode = (node *)malloc(sizeof(node));
    newnode->blockid = data.blockid;
    strcpy(newnode->fname, data.fname);
    newnode->next = NULL;
    t = head;
    while (t->next != NULL)
    {
        t = t->next;
    }
    t->next = newnode;
}
void contiguous_alloc(node *mainmem, fileDetails filedata[])
{
    dir d[noOfFiles];
    int i, j;
    int alloc = 0;
    int randno;
    int occur[noOfBlocks + 1];
    node *t;
    node *start;
    int found;
    int count_rand;
    for (i = 0; i < noOfFiles; i++)
    {
        found = 0;
        count_rand = 0;
        for (j = 1; j <= noOfBlocks; j++)
            occur[j] = 0;
        while (count_rand != noOfBlocks)
        {
            randno = (rand() % noOfBlocks) + 1;
            while (occur[randno] != 0)
                randno = (rand() % noOfBlocks) + 1;
            count_rand++;
            occur[randno] = 1;
            t = mainmem;
            for (j = 0; j < randno; j++)
                t = t->next;
            start = t;
            found = 1;
            for (j = 0; j < filedata[i].fileblocks; j++)
            {
                if (t == NULL)
                    break;
                if (strcmp(t->fname, "free") == 0)
                {
                    t = t->next;
                    continue;
                }
                else
                {
                    found = 0;
                    break;
                }
            }
            if (found == 1)
            {
                d[alloc].start = start->blockid;
                d[alloc].length = filedata[i].fileblocks;
                strcpy(d[alloc].fname, filedata[i].fname);
                for (j = 0; j < filedata[i].fileblocks; j++)
                {
                    strcpy(start->fname, filedata[i].fname);
                    start = start->next;
                }
                break;
            }
        }
        if (found == 0)
```

```c
            printf("\nMEMORY UNAVAILABLE\n");
        else
            alloc++;
    }
    printf("No. of files allocated:%d\n", alloc);
    printf("\nDirectory\n");
    printf("\tFile Name\tStart\tLength\n");
    for (i = 0; i < alloc; i++)
        printf("\t%s\t\t%d\t%d\n", d[i].fname, d[i].start, d[i].length);
}
void linked_alloc(fileDetails filedata[], node *mainmem)
{
    node *temp, *start, *new;
    int i, j, k;
    int randno;
    int found;
    dir d[noOfFiles];
    int startpos;
    for (i = 0; i < noOfFiles; i++)
    {
        for (j = 0; j < filedata[i].fileblocks; j++)
        {
            found = 0;
            while (found == 0)
            {
                randno = (rand() % noOfBlocks) + 1;
                temp = mainmem;
                for (k = 0; k < randno; k++)
                    temp = temp->next;
                if (strcmp(temp->fname, "free") == 0)
                {
                    strcpy(temp->fname, filedata[i].fname);
                    found = 1;
                    if (j == 0)
                    {
                        new = temp;
                        strcpy(d[i].fname, filedata[i].fname);
                        d[i].start = temp->blockid;
                    }
                    else if (j == filedata[i].fileblocks - 1)
                    {
                        new->link = temp;
                        temp->link = NULL;
                        d[i].end = temp->blockid;
                    }
                    else
                    {
                        new->link = temp;
                        new = new->link;
                    }
                }
            }
        }
    }
    printf("\nDirectory\n");
    printf("\tFile Name\tStart\tEnd\n");
    for (i = 0; i < noOfFiles; i++)
        printf("\t%s\t\t%d\t%d\n", d[i].fname, d[i].start, d[i].end);
    printf("\nIndividual File listing\n");
    for (i = 0; i < noOfFiles; i++)
    {
        printf("File Name: %s\n", d[i].fname);
        startpos = d[i].start;
        temp = mainmem;
        for (j = 0; j < startpos; j++)
            temp = temp->next;
        printf("\tData-block %d\n", temp->blockid);
        temp = temp->link;
        while (temp != NULL)
        {
            printf("\tData-block %d\n", temp->blockid);
            temp = temp->link;
        }
    }
}
void indexed_alloc(fileDetails filedata[], node *mainmem)
{
    node *temp, *start, *indexblock;
    int i, j, k;
    int indexblockid;
    int randno;
    int found;
    dir d[noOfFiles];
    for (i = 0; i < noOfFiles; i++)
    {
        found = 0;
        while (found != 1)
        {
            randno = (rand() % noOfBlocks) + 1;
            temp = mainmem;
            for (k = 0; k < randno; k++)
                temp = temp->next;
            if (strcmp(temp->fname, "free") == 0)
            {
                found = 1;
                strcpy(temp->fname, filedata[i].fname);
            }
        }
        indexblock = temp;
        strcpy(d[i].fname, filedata[i].fname);
        d[i].start = indexblock->blockid;
        for (j = 0; j < filedata[i].fileblocks; j++)
        {
            found = 0;
            while (found != 1)
            {
                randno = (rand() % noOfBlocks) + 1;
                temp = mainmem;
                for (k = 0; k < randno; k++)
                    temp = temp->next;
                if (strcmp(temp->fname, "free") == 0)
                {
                    found = 1;
                    strcpy(temp->fname, filedata[i].fname);
                    indexblock->fileblocktable[j] = temp->blockid;
                }
            }
        }
    }
    printf("\nDirectory\n");
    printf("\tFile Name\tIndexed Block\n");
    for (i = 0; i < noOfFiles; i++)
        printf("\t%s\t\t%d\n", d[i].fname, d[i].start);
    printf("\n\nIndex Table\n");
    printf("File Name\t\tBlock Indexed\n");
    for (i = 0; i < noOfFiles; i++)
    {
        indexblockid = d[i].start;
        temp = mainmem;
        for (j = 0; j < indexblockid; j++)
            temp = temp->next;
        printf("\n%s", temp->fname);
        for (j = 0; j < filedata[i].fileblocks; j++)
            printf("\t\t\tData-block %d\n", temp->fileblocktable[j]);
    }
}

void main()
{
    int mem_size, choice, i;
    node data;
    node *mainmem;
    node *temp;
    fileDetails filedata[100];
    char c;
    mainmem = malloc(sizeof(node));
    mainmem->next = NULL;
    printf("Enter the main memory size:");
    scanf("%d", &mem_size);
    printf("Enter the block size:");
    scanf("%d", &block_size);
    noOfBlocks = (int)mem_size / block_size;
    printf("Total no. of blocks available:%d\n", noOfBlocks);
    for (i = 0; i < noOfBlocks; i++)
    {
        data.blockid = i + 1;
        strcpy(data.fname, "free");
        insert(mainmem, data);
    }
    printf("Number of files to be allocated:");
    scanf("%d", &noOfFiles);
    for (i = 0; i < noOfFiles; i++)
    {
        printf("\nName of file %d:", i + 1);
        scanf("%s", filedata[i].fname);
        printf("Size of file %d(in KB):", i + 1);
        scanf("%d", &filedata[i].fsize);
        filedata[i].fileblocks = ceil((float)filedata[i].fsize / (float)block_size);
    }
    do
    {
        printf("\n\nFILE ALLOCATION TECHNIQUES\n");
        printf("1.Contiguous\n");
        printf("2.Linked\n");
        printf("3.Indexed\n");
        printf("Enter choice:");
        scanf("%d", &choice);
        temp = mainmem->next;
        while (temp != NULL)
        {
            strcpy(temp->fname, "free");
            temp = temp->next;
        }
        srand(time(NULL));
        switch (choice)
        {
        case 1:
            contiguous_alloc(mainmem, filedata);

            break;
        case 2:
            linked_alloc(filedata, mainmem);
            break;
        case 3:
            indexed_alloc(filedata, mainmem);
            break;
        }
        printf("\nDo you want to continue?:");
        scanf("%s", &c);
    } while (c == 'y');
}
```

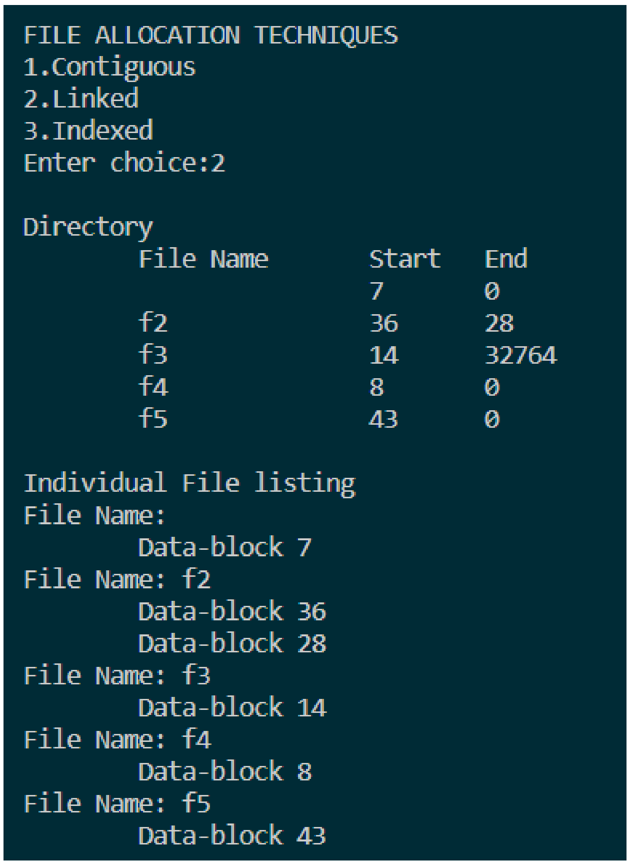
Output:



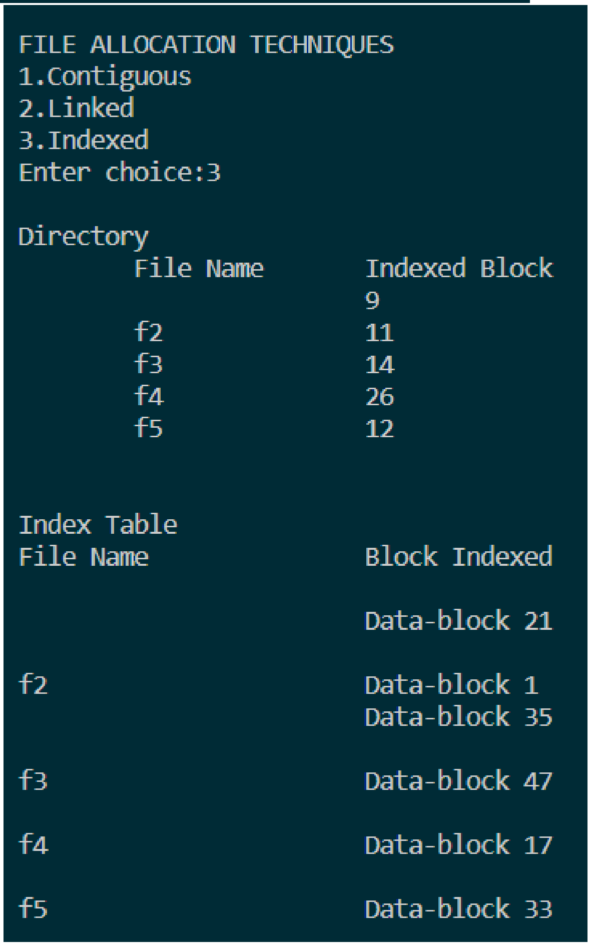
```
Enter the main memory size:500
Enter the block size:10
Total no. of blocks available:50
Number of files to be allocated:5

Name of file 1:f1
Size of file 1(in KB):10

Name of file 2:f2
Size of file 2(in KB):20

Name of file 3:f3
Size of file 3(in KB):10

Name of file 4:f4
Size of file 4(in KB):5

Name of file 5:f5
Size of file 5(in KB):5
```



```
FILE ALLOCATION TECHNIQUES
1.Contiguous
2.Linked
3.Indexed
Enter choice:1
No. of files allocated:5

Directory
        File Name       Start   Length
        f1              32      1
        f2              17      2
        f3              36      1
        f4              14      1
        f5              33      1
```



```
FILE ALLOCATION TECHNIQUES
1.Contiguous
2.Linked
3.Indexed
Enter choice:2

Directory
        File Name       Start   End
                        7       0
        f2              36      28
        f3              14      32764
        f4              8       0
        f5              43      0

Individual File listing
File Name:
        Data-block 7
File Name: f2
        Data-block 36
        Data-block 28
File Name: f3
        Data-block 14
File Name: f4
        Data-block 8
File Name: f5
        Data-block 43
```



```
FILE ALLOCATION TECHNIQUES
1.Contiguous
2.Linked
3.Indexed
Enter choice:3

Directory
        File Name       Indexed Block
                        9
        f2              11
        f3              14
        f4              26
        f5              12


Index Table
File Name               Block Indexed

                        Data-block 21

f2                      Data-block 1
                        Data-block 35

f3                      Data-block 47

f4                      Data-block 17

f5                      Data-block 33
```

Learning Outcome:

- Learnt about the various file allocation techniques
- Learnt to implement the c program for the various file allocation techniques