****

**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)

            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;

        }

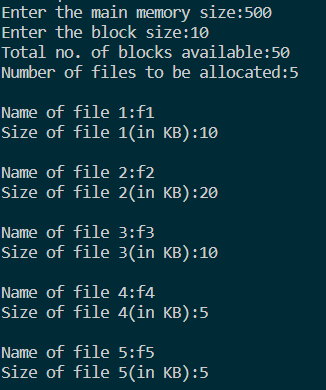
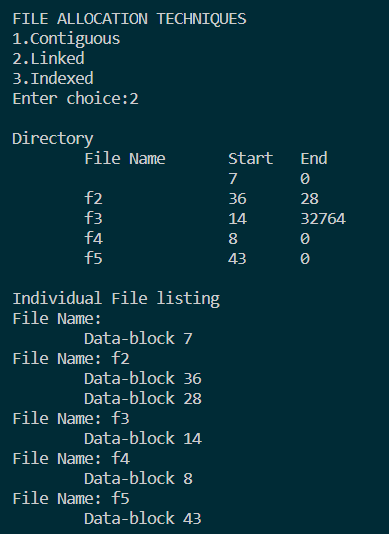
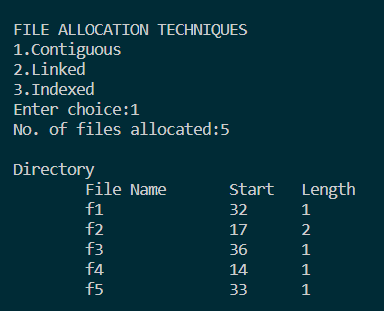
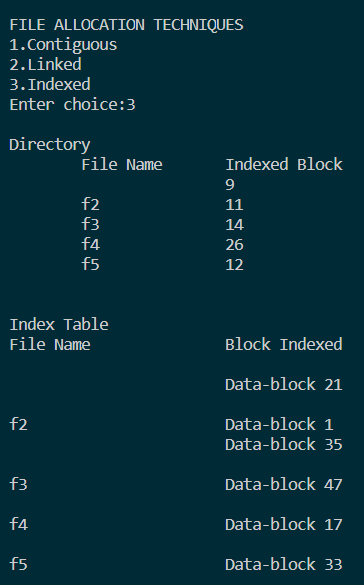
        printf("\nDo you want to continue?:");

        scanf("%s", &c);

    } while (c == 'y');

}

**Output:**

**Learning Outcome:**

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