Dhirubhai Ambani
Institute of Information and Communication Technology

LAB 8

File System Implementation III

File System Design Document

Group No.: 28 (2020 batch)

Name	IDs
Noopur Chaudhary	202001091
Karan Padhiyar	202001162

Course Instructor Name: Prof. Sanjay Shrivastava

Overview:

A file system is a set of files, directories, and other structures. File systems maintain information and identify where a file or directory's data is located on the disk. In addition to files and directories, file systems contain a superblock, free blocks-map, and one or more allocation groups. An allocation group contains data blocks - fragments. Each file system occupies one logical volume.

Constraints on the file system:

- The system supports 100 files.
- A file's max size is 10 KB, which cannot span more than ten blocks.
- The max size for a filename is 100 characters.
- The file name uniquely identifies files for now. So, this means two files with the same file name are discouraged.
- The size of a data block is 4096 bytes or 4KB.
- The file system only supports one directory, which means only files can exist on the disk.

Structures:

1. **Struct file:**

- It provides the information of the particular file.
- It contains all exist files names, and how many blocks are acquired by each of them.
- There, all the files have maximum 10 blocks, the structure also contains how many blocks are filled with data and how many are just empty using (Data Blocks) a vector or an array.

Code:

```
struct file{
   int isCreated;
   char fileName[fileNameLimit];
   int dataBlocks[maxDataBlocksPerFile];
   // dataBlocks values:
   // Block is quired by the file and flled with data: 1
   // Block is acquired by the file and emty: -1
   //  Block is not acquired by the file and also empty: 0
   int numBlocks;
};
```

2. Struct Superblock:

- This Block Stores metadata like the total size of the disk, the size of a data block, the bitmap, the total number of data blocks, and the number of full and empty blocks.

Code:

```
struct superBlock{
        int totalSize;
        int blockSize;
        int totalNumBlocks;
        int free[maxNumDataBlocks];
                // Bitmap vector
        int numEmptyBlocks;
        int numFullBlocks;
        // It is a single directory structure so it contains no directory, only files
};
```

3. Struct Directory:

- This stores the File table (which in turn stores the name of the file, the number of blocks this file takes on the disk, and the list of blocks in sequential order where this file exists), max number of files allowed, and the max size of the file allowed.

Code:

```
struct directory{
   struct file files[maxNumOfFiles]; // This is the file Allocation table
   int maxFileSize;       // Maximum number of blocks a file is allowed to occupy
   int maxNumFiles;       // Maximum number of files allowed on the disk
};
```

4. Struct datablock:

- It contains the data of all the filled blocks.
- Block sizes may range from 512 bytes to 4K or larger; there we used 4KB block size, because usually virtual page is 4096 bytes.
    - **Code:**

```
struct dataBlock{
   char data[blockSizeLimit];
};
```

System Calls and description:

1. File system module initialization (mount function)

- Prototype: void mount ();

- Function description:

➢ This program initializes the data structures and resources used by the file system, and if it detects that the disk is already formatted, the program calls this function and mounts the disk (FileSystem.data). The disk is not always necessarily formatted.

➢ This function is the only NON-SYSTEM CALL function you are required to implement.

Pseudo Code:

```
Mount ()
{
        print "1. Format the disk or 2. Use the disk as it is."
        if (choice = 1)
        {
                It releases all the data from the disk;
                Create new super block structure;
                Assign filled bocks = max blocks;
                Assign Empty bocks = 0;
                Assign all zeros to free vector or array (bitmap vector);
                Create new directory structure;
                And write both the structures to the disk;
        }
        else
        {
                It opens the data file;
                And assign the superblock and directory structure from the
disk (both the structures are stored on the disk);
        }
}
```

2. Create a file:

- Prototype: void createFiles()

Pseudo Code:

```
CreateFile ()
{
        Finding an empty space in file allocation table for the new file
        Takes a new file name and check if the file name is existed or not.
        Initialize the block list for the new file.
        Then take a user input of the file  on data block structure;
        Update the filledblock += consumedBlocks
        Update the Emptyblock -= comsumedBlocks.
        Update the superblock and directory structures on the disk.
        Store the data blocks on the FileSystem.data disk
}
```

3. Read a file:

- Prototype: void readFile()

Pseudo Code:

```
readFile ()
{
        Take file name as a user input,
        Search the particular file,
        If (file not exists)
        {
                Prints that file not exist.
        }
```

Else

{

Set a file pointer to the first block of the particular file,

Prints all the filled blocks of the file from the data block structure.

}

}

4. Edit a file:

- Prototype: void editFile()

Pseudo Code:

editFile ()

{

Take file name as a user input,

Search the particular file,

If (file not exists)

{

Prints that file not exist.

}

Else

{

Print “choice 1. Edit the blocks or 2. Remove the blocks”

If(choice 1)

{

It first set a pointer at the beginning of the file or at the first block of the file.

Then clear the all the blocks, update free vector,

Update filled blocks value and empty blocks value,

Update the superblock structure, and directory structure,

Take a user input data,

Set file pointer to the first block of the file and store all the data on the disk,

Update the data block structure,

Update the Superblock and directory structures,

Store the both the structures on the disk,

}

Else

{

Clear all the data of the file using above method;

Update and store the all the structures and its values on the disk;

}

}

}

5. Delete a file:

- It will delete a file if the file exist, update the free vector, data blocks vector, file structure and other structures on the disk.
- Prototype: void createFiles()

Pseudo Code:

deleteFile ()

{

Take file name as a user input,

Search the particular file,

If (file not exists)

{

Prints that file not exist.

}

Else

{

It first set a pointer at the beginning of the file or at the first block of the file.

Then clear the all the blocks, update free vector and data blocks vector's values to Zero,

Update filled blocks value and empty blocks value,

Update and store the superblock structure, directory structure and dataBlock structure on the disk;

}

}

File System Code:

```
// The system supports a maximum of 100 files
// The max size a file can have is 10 KB that is a file cannot span more than 10 blocks
// The max size for a file name is 100 characters
// Files are uniquely identified by the file name for now. So this means two files with same file name are discouraged.

// This file system uses linked list approach in order to store the data to data blocks. This helps use all the free datablocks on the disk.
// The size and the number of datablocks required for a file are automatically calculated based on the data user enters for a file.
// This file system also supports empty files. Empty files take no data blocks.

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

```
#define fileNameLimit 100
#define maxDataBlocksPerFile 10
#define maxNumDataBlocks 1000
#define totalSizeofDisk 4096000
#define blockSizeLimit 4096
#define maxNumOfFiles 100


void mainMenu();
void listFiles();
void createFile();
void readFile();
void unMount();


struct file{
   int isCreated;
   char fileName[fileNameLimit];
   int dataBlocks[maxDataBlocksPerFile];
   // dataBlocks values:
   // Block is quired by the file and flled with data: 1
   // Block is acquired by the file and emty: -1
   //  Block is not acquired by the file and also empty: 0
   int numBlocks;
};

struct superBlock{
   int totalSize;
   int blockSize;
   int totalNumBlocks;
   int free[maxNumDataBlocks];  // Bitmap vector
   int numEmptyBlocks;
   int numFullBlocks;
   // It is a single directory structure so it contains no directory, only files
};

struct directory{
   struct file files[maxNumOfFiles]; // This is the file Allocation table
   int maxFileSize;       // Maximum number of blocks a file is allowed to occupy
   int maxNumFiles;       // Maximum number of files allowed on the disk
};

struct dataBlock{
   char data[blockSizeLimit];
};
```

```
FILE* disk;
struct superBlock sb;
struct directory dir;



void mainMenu(){
   printf("Main Menu:\n\n");
   printf("1. List files in the disk\n");
   printf("2. Create file\n");
   printf("3. Read file\n");
   printf("4. Edit a file\n");
   printf("5. Delete a file\n");
   printf("-. Any other number to exit\n\n");
   printf("Enter choice: ");
}

void mount(){

   printf("Trying to mount disk named FileSystemDisk.data\n");
   printf("Do you want to format the disk or use it as it is?(1 or 2): ");
   int format;
   scanf("%d", &format);

   if(format==1){
      disk= fopen("FileSystemDisk.data", "w+");
      if(!disk){
         printf("Error mounting the disk! Exiting\n");
         exit(1);
      }
      else
         printf("Mount successful!\n\n");

      // Initialization for superblock
      sb.totalSize= totalSizeofDisk;   // in Bytes
      sb.blockSize= blockSizeLimit;      // in Bytes
      sb.totalNumBlocks= sb.totalSize/sb.blockSize;

      for(int i=0; i<sb.totalNumBlocks; i++)
         sb.free[i]=0;

      sb.numEmptyBlocks= maxNumDataBlocks;
      sb.numFullBlocks= 0;

      // Initialization for directory
      dir.maxFileSize= maxDataBlocksPerFile;     // This is 10 blocks or 10*4096 Bytes
      dir.maxNumFiles= sb.totalNumBlocks/dir.maxFileSize;
```

```
        for(int i=0; i<dir.maxNumFiles; i++)
            dir.files[i].isCreated=0;

        // Writing the fresh super block and the directory block on the disk
        fwrite(&sb, sizeof(struct superBlock), 1, disk);
        fwrite(&dir, sizeof(struct directory), 1, disk);

    }
    else{
        // opening the disk for read and write in append mode
        disk= fopen("FileSystemDisk.data", "a+");
        if(!disk){
            printf("Error mounting the disk! Exiting\n");
            exit(1);
        }
        else
            printf("Mount successful!\n\n");

        // Reading the superblock and directory block from the disk
        fread(&sb, sizeof(struct superBlock), 1, disk);
        fread(&dir, sizeof(struct directory), 1, disk);
    }
}

void listFiles(){
    if(sb.numFullBlocks==0)
        printf("Disk empty. No files on the disk!\n");
    else{
        printf("\n---------------------------FileSystemDisk.data-------------------------\n\n");
        printf("File No.\tFile Name\tNumber of Blocks Occupied\n");
        for(int i=0; i<dir.maxNumFiles; i++){
            if(dir.files[i].isCreated)
                printf("%d\t       %s\t %d\n", i+1, dir.files[i].fileName, dir.files[i].numBlocks);
        }
    }
    printf("\n");
}

void createFile(){

    // Finding an empty space in file allocation table for the new filer
    int fileIndex;
    for(fileIndex=0; fileIndex<dir.maxNumFiles; fileIndex++){
        if(!dir.files[fileIndex].isCreated)
            break;
    }
```

```
    // Getting file name of the new file from the user
    printf("Enter the name of the file to create: ");
    char removeSpace;
    scanf("%c", &removeSpace);
    scanf("%s", dir.files[fileIndex].fileName);

    //Initializing the block list for the newly created file
    for(int i=0; i<10; i++)
        dir.files[fileIndex].dataBlocks[i]=-1;

    printf("\n---------Enter data (input # when finished)----------\n\n");

    // Removing any unwanted characters in the input stream
    scanf("%c", &removeSpace);

    char inputChar='s';
    int numBlocks=1;
    int blockIndex=0;

    while(inputChar!='#'){

        // Finding a free datablock on disk
        for(; blockIndex<sb.totalNumBlocks; blockIndex++){
            if(!sb.free[blockIndex])
                break;
        }

        // Moving the file pointer to the appropriate position
        fseek(disk, (blockIndex-(ftell(disk) - sizeof(struct superBlock) - sizeof(struct
directory)))*sb.blockSize, SEEK_CUR);

        // Getting user input for file data
        struct dataBlock db;
        int i;
        for(i=0; i<sb.blockSize; i++){
            //Getting input from the user
            if( (inputChar=getchar())!='#' )
                db.data[i]=inputChar;
            else
                break;
        }
        if(inputChar=='#')
            db.data[i]='#';

        // The only data user entered was # so this is an empty file with 0 blocks allocated to
it
```

```
        if(numBlocks==1 && strcmp(db.data, "#")==0)
            dir.files[fileIndex].numBlocks=0;

        // For non empty files
        else{
            // Writing the user input to the file
            fwrite(&db, sizeof(struct dataBlock), 1, disk);

            // Updating the bit map in the super block
            sb.free[blockIndex]=1;

            // Adding the location of this datablock to the list of data blocks for this file
            dir.files[fileIndex].dataBlocks[numBlocks-1]=blockIndex;
            blockIndex++;

            if(inputChar!='#')
                numBlocks++;
        }
    }

    // Updating the number of blocks this file takes up in the directory block
    dir.files[fileIndex].numBlocks=numBlocks;

    // Moving file pointer to the beginning of the disk
    fseek(disk, 0, SEEK_SET);

    // Updating the number of Full and Empty datablocks in the superblock
    sb.numEmptyBlocks-=dir.files[fileIndex].numBlocks;
    sb.numFullBlocks+=dir.files[fileIndex].numBlocks;

    //Writing the updates superblock back to the disk
    fwrite(&sb, sizeof(struct superBlock), 1, disk);

    // Updating the file table in directory
    dir.files[fileIndex].isCreated=1;

    //Writing the updated directory block back to the disk
    fwrite(&dir, sizeof(struct directory), 1, disk);
}

void readFile(){

    // Getting filename of the file that the user wants to read
    printf("Enter the name of the file to read: ");
    char removeSpace;
    scanf("%c", &removeSpace);
```

```
    char inputName[100];
    scanf("%s", inputName);

    // Looking for the required file in the directory
    int fileIndex;
    for(fileIndex=0; fileIndex<dir.maxNumFiles; fileIndex++){
        if(strcmp(dir.files[fileIndex].fileName, inputName)==0)
            break;
    }

    if(fileIndex==dir.maxNumFiles)
        printf("No such file present on the disk!\n");

    else{
        printf("\n-----------------------------%s----------------------------\n\n",
dir.files[fileIndex].fileName);
        for(int i=0; i<10; i++){

            // If this is the end of the data block list for this file then nothing left to read,
come out of the loop
            if(dir.files[fileIndex].dataBlocks[i]==-1)
                break;

            //else
            // Move the file pointer to this data block
            fseek(disk, (dir.files[fileIndex].dataBlocks[i]-(ftell(disk)- sizeof(struct superBlock)
- sizeof(struct directory)))*sb.blockSize, SEEK_CUR);

            //Read the data in the data block
            struct dataBlock db;
            fread(&db, sizeof(struct dataBlock), 1, disk);

            // Print the read data character by character
            for(int i=0; i<sb.blockSize && db.data[i]!='#'; i++)
                printf("%c", db.data[i]);
        }
        printf("\n");

        // Moving the file pointer back to the start of the first data block
        fseek(disk, sizeof(struct superBlock)+sizeof(struct directory), SEEK_SET);
    }
}

void editFile(){

    // Getting filename of the file that the user wants to read
    printf("Enter the name of the file to edit: ");
```

```
    char removeSpace;
    scanf("%c", &removeSpace);

    char inputName[100];
    scanf("%s", inputName);

    // Looking for the required file in the directory
    int fileIndex;
    for(fileIndex=0; fileIndex<dir.maxNumFiles; fileIndex++){
        if(strcmp(dir.files[fileIndex].fileName, inputName)==0)
            break;
    }

    if(fileIndex==dir.maxNumFiles)
        printf("No such file present on the disk!\n");

    else{
        printf("\n\n1. Edit the block\n-. Any other value to Remove the block\n\n----------
----------------------------------------\n\n");
        int i;
        for(i=0; i<10; i++){
            if(dir.files[fileIndex].dataBlocks[i]==-1)
                break;

            // Moving the file pointer to this data block
            fseek(disk, (dir.files[fileIndex].dataBlocks[i]-(ftell(disk)- sizeof(struct superBlock)
- sizeof(struct directory)))*sb.blockSize, SEEK_CUR);

            //Reading the data in the data block
            struct dataBlock db;
            fread(&db, sizeof(struct dataBlock), 1, disk);

            // Print the read data character by character
            for(int i=0; i<sb.blockSize && db.data[i]!='#'; i++)
                printf("%c", db.data[i]);
            printf("\n\n");

            printf("Enter your choice: ");
            int input;
            scanf("%d", &input);

            // Edit the block
            if(input==1){
                printf("\nEnter the new data for this block (# to stop taking the input for this
block)\n");
                char removeSpace;
                scanf("%c", &removeSpace);
```

```
            fseek(disk, dir.files[fileIndex].dataBlocks[i]*sb.blockSize+sizeof(struct
superBlock)+sizeof(struct directory), SEEK_SET);
            char inputChar;
            struct dataBlock db;
            int j;
            for(j=0; j<sb.blockSize; j++){
               if((inputChar=getchar())=='#' || inputChar=='*')
                  break;
               db.data[j]=inputChar;
            }

            // The user left the block empty so there is not data for this block, we can mark
it as free
            if(strcmp(db.data, "#")==0){
               sb.free[dir.files[fileIndex].dataBlocks[i]]=0;
               for(int j=i; dir.files[fileIndex].dataBlocks[j]!=-1; j++)
                  dir.files[fileIndex].dataBlocks[j]=dir.files[fileIndex].dataBlocks[j+1];
               dir.files[fileIndex].numBlocks--;
               sb.numFullBlocks--;
               sb.numEmptyBlocks++;
            }
            // Otherwise write the new data for this block
            else{
               if(inputChar=='#')
                  db.data[j]='#';
               fwrite(&db, sizeof(struct dataBlock), 1, disk);
            }
         }

         // Deleting the block
         else{
            // Marking this block as free in the bit map
            sb.free[dir.files[fileIndex].dataBlocks[i]]=0;
            // Removing this block from the file's data block list
            for(int j=i; dir.files[fileIndex].dataBlocks[j]!=-1; j++)
               dir.files[fileIndex].dataBlocks[j]=dir.files[fileIndex].dataBlocks[j+1];
            // Decrementing the number of blocks for this file
            dir.files[fileIndex].numBlocks--;
            // Updating the number of Empty and Full blocks in the superBlock
            sb.numFullBlocks--;
            sb.numEmptyBlocks++;
         }
      }
      printf("\n");

      // Moving the file pointer back to the start of the first data block
      fseek(disk, sizeof(struct superBlock)+sizeof(struct directory), SEEK_SET);
```

```
    }
}

void deleteFile(){

    // Getting filename of the file that the user wants to delete
    printf("Enter the name of the file to delete: ");
    char removeSpace;
    scanf("%c", &removeSpace);

    char inputName[100];
    scanf("%s", inputName);

    // Looking for the required file in the directory
    int fileIndex;
    for(fileIndex=0; fileIndex<dir.maxNumFiles; fileIndex++){
        if(strcmp(dir.files[fileIndex].fileName, inputName)==0)
            break;
    }

    if(fileIndex==dir.maxNumFiles)
        printf("No such file present on the disk!\n");

    else{
        for(int i=0; i<10; i++){
            if(dir.files[fileIndex].dataBlocks[i]!=-1)
                sb.free[dir.files[fileIndex].dataBlocks[i]]=0;
            else
                break;
        }
        dir.files[fileIndex].isCreated=0;
        sb.numEmptyBlocks+=dir.files[fileIndex].numBlocks;
        sb.numFullBlocks-=dir.files[fileIndex].numBlocks;
        printf("\nFile named %s deleted\n", dir.files[fileIndex].fileName);
    }
}

void unMount(){
    if(disk!=NULL){
        fclose(disk);
        printf("\nUnmount successful!\n");
    }
}
```

```c
int main(){
    mount();

    char retToMainMenu;

    do{
        mainMenu();
        int choice;
        scanf("%d", &choice);

        if(choice==1)
            listFiles();
        else if(choice==2)
            createFile();
        else if(choice==3)
            readFile();
        else if(choice==4)
            editFile();
        else if(choice==5)
            deleteFile();
        else
            break;
        printf("Do you want to return to main menu?(y/n): ");
        scanf(" %c", &retToMainMenu);

    }while(retToMainMenu=='y');

    unMount();
    return 0;
}
```

Note: FileSystemDisk.data file is attached with the zip file.