Hacettepe University
Department of Computer Science & Engineering

BBM203 Software Laboratory
Experiment IV

Subject : Data Structures and Algorithms (Lists), Linux
Submission Date : 09.12.2013
Design Rep. Due Date : 15.12.2013
Date Due : 29.12.2013
Environment : ANSI C, CentOS 6.4, GCC 4.4.7 20120313 (Red Hat 4.4.7-3)
Advisors : Dr. Mustafa Ege, Dr. Sevil Şen, R.A. Çağdaş Baş

INTRODUCTION

In this experiment you are expected to develop a defragmentation application. Your application will not deal with real file systems. You will develop a defragmentation application for an EXT2, FAT32 hybrid "file system file". Details of this so called file system will be explained.

Also you are expected to learn some basic Linux file operations. You will create the given file structure on the real file system of a Linux system.

BACKGROUND INFORMATION

ext2

The ext2 is the second version of the extended file system for Linux Systems. In this experiment a simplified and FAT32 like version of ext2 will be used. Every file or directory is represented by an i-node in the ext2 file system. Here is a simplified version of i-node structure for implementation:

Field	Size (in Bytes)
Own pointer	2
Parent pointer	2
Name	16
Attributes	2
User ID	2
Group ID	2
Data Pointer	2
Next Pointer	2
Size	2

Tablo 1 i-node structure

Attributes	Bit Index
Unused	15(MSB)-10
Is Directory	9
User Write	8
User Read	7
User Exec	6
Group Write	5
Group Read	4
Group Exec	3
Other Write	2
Other Read	1
Other Exec	0 (LSB)

Tablo 2 Node attributes

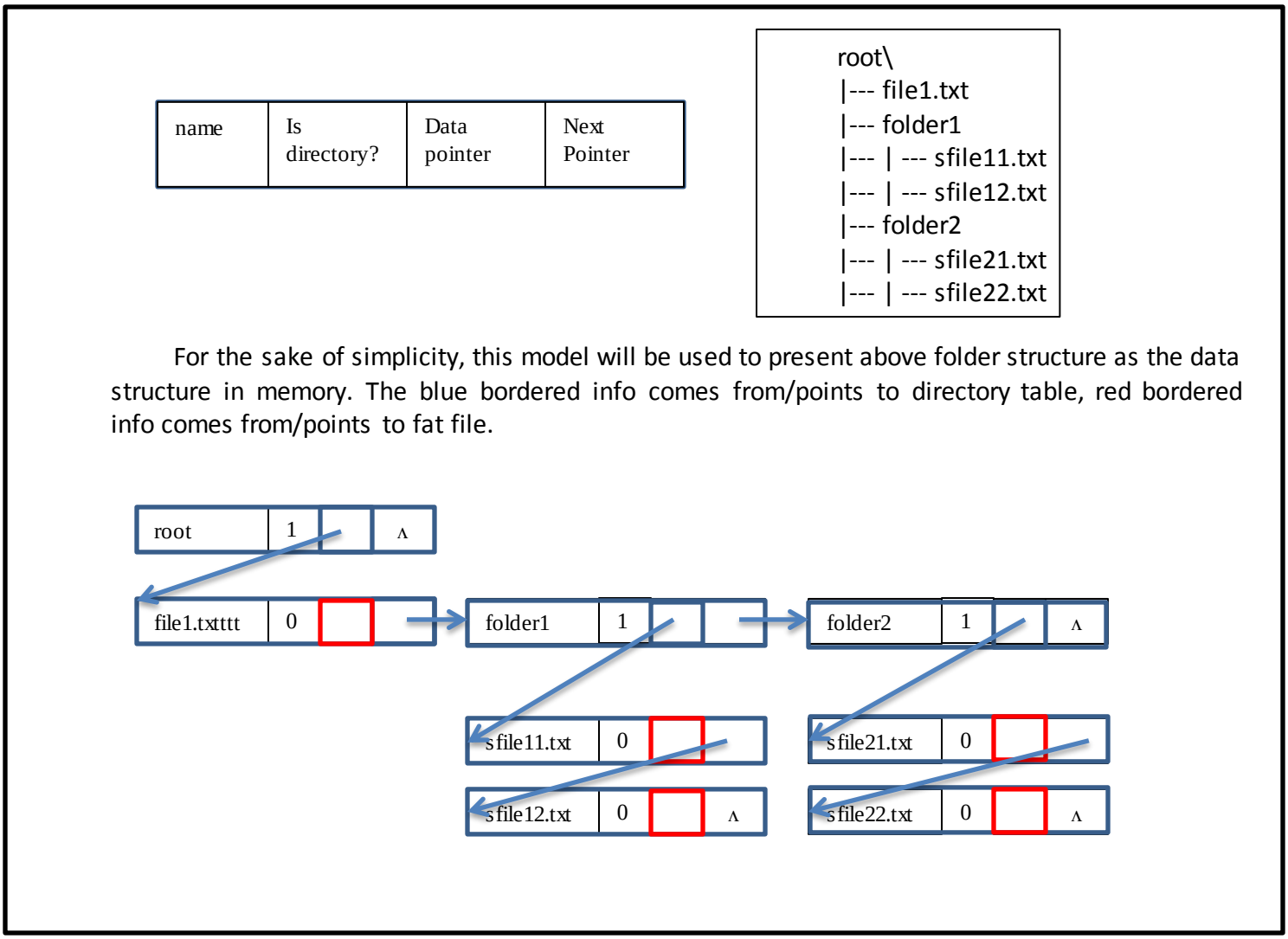
We will use a very similar ext2 i-node structure but with one exception. Our i-node doesn't have direct data link or any other. Our ext2 will store data and file allocation information like a standard FAT file system; we will come to that in FAT Section.

As you can see every node has an attribute field. This field defines a node's various attributes. If a bit is 1 then this means given attribute is true. The most important bit is the 9th bit (6th from left), the directory bit. If this bit is set to "1" then this means this node belongs to a directory structure, if "User write" bit is "1" then the user is allowed to write, and so on.

If a node is a directory node, then Data pointer points its sub contents. If node represents a file node this means "Data Pointer" points a sector address from the file allocation table.

Next pointer always represents next file/folder node. For instance if there are two files under a directory, let's say "f1" and "f2" is under "folder1"; folder1's directory will be "1", "folder1"'s data pointer will point "f1"'s node. "f1"'s data pointer will point a cluster number from FAT table, and "f1"'s next pointer will point "f2"'s node because these files are in the same level under "folder1.

You can examine an example outline of FS in the figure illustrated below:



For the sake of simplicity, this model will be used to present above folder structure as the data structure in memory. The blue bordered info comes from/points to directory table, red bordered info comes from/points to fat file.

Figure 1.

FAT32

FAT32 is a basic file-system developed for MS-DOS and Microsoft Windows and also floppy disks. It has a directory element of every file's first cluster and for files bigger than just one cluster it uses the File Allocation Table. Simply, File Allocation Table stores the next cluster's number for a file. End of file is shown with an invalid value. In this experiment we will use a different and simplified version for our needs.

Our File Allocation Table would be similar to FAT32 with little changes. Our FAT will only include files in the root directory; no directory information will be stored for simplicity. The position of a FAT element corresponds to a cluster on the hard drive and the values in FAT represent the next cluster's location. FFFFFFFF signifies the end of file.

00000001	00000002	00000005	00000004	00000006
00000009	FFFFFFFF	0000000A	00000000	FFFFFFFF
0000000C	FFFFFFFF	0000000D	FFFFFFFF	00000000
00000000	00000000	00000000	00000000	00000000

File 1: 0, 1, 2, 5, 9
File 2: 3, 4, 6
File 3: 7, A, C, D
File 4: B

Figure 2: Sample FAT, with cluster chains and file explanation.

File system fragmentation

In computing, file system fragmentation is the inability of a file system to lay out related data sequentially (contiguously). File system fragmentation increases disk head movement or seeks, which are known to hinder throughput. The correction to existing fragmentation is to reorganize files and free space back into contiguous areas, a process called defragmentation. In this experiment you will implement simple defragmentation software for our hybrid file system.

You can see above file system is highly fragmented. We wish it would be like this:

00000001	00000002	00000003	00000004	FFFFFFFF
00000006	00000007	FFFFFFFF	00000009	0000000A
0000000B	FFFFFFFF	FFFFFFFF	00000000	00000000
00000000	00000000	00000000	00000000	00000000

File 1: 0, 1, 2, 3, 4
File 2: 5, 6, 7
File 3: 8, 9, A, B
File 4: C

Figure 3: Defragmented Sample FAT.

EXPERIMENT

There are two different sections of this experiment. In the first section you will read 3 input file and defragment disk. In the second part you will create directories and files in the real file system. You will have 3 input files named "dir.txt" which includes directory tables; "fat.txt" which includes file allocation tables; "harddisk.txt" which includes real data. Hard disk size will not be larger than 1KB.

Section 1.

Directory Table: dir.txt

This file will include directory info in a binary mode which means you will use binary file operations to read and write. First sector (8B) will always be the root directory's i-node. You will create your generalized list from beginning to root. Remember, an i-node is 4 sectors long. Data pointers for directories or next pointers for all i-nodes will show you the location of the next i-node, which is located in dir.txt. You can simply multiply your pointer's value to 8 then you will reach desired i-node's beginning.

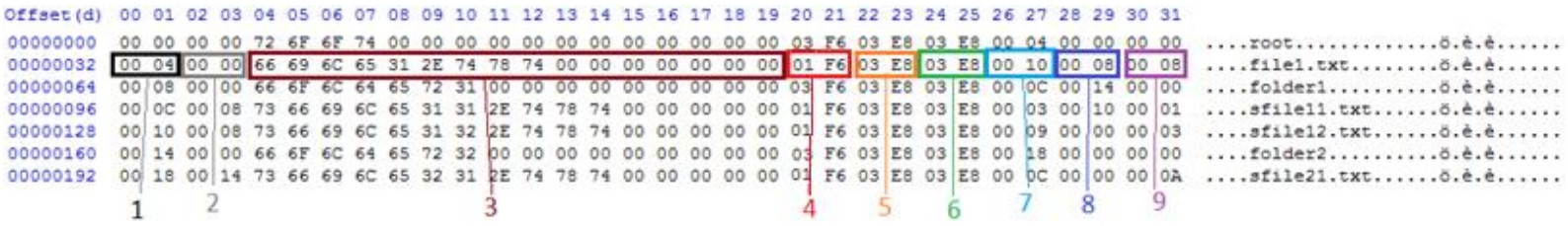


Figure 4: Example of directory file. Offset means the address, middle block is data and right block is it's ASCII correspondence (e.g. 66="f") There are dots for non-readable bytes (e.g. 08 = BS in ASCII table but not a readable byte.)

As you can see, these are 9 different parts of the i-node.

1. Is the self-pointer which gives us the starting address of the i-node in the directory file. (4. Sector means 4*8=32th byte)
2. Is the parent's address which gives us the starting address of its parent's i-node in the directory file. (0 means 0th sector, the root node.)
3. Is the name of the file. It will always be an ASCII coded string and will not exceed 16 bytes.
4. Is the node attributes. You can see which bit is used for what in the first section. For example 01F6 means 000000 0 111 110 110 as bit array which means this attribute is a file's attribute (7th bit from left is 0) and file's permissions are 766.
5. Is the user id. You will just convert hexadecimal number to decimal number and set the file's owner according to this.
6. Is the group id. You will just convert hexadecimal number to decimal number and set the file's group according to this.
7. Is the data pointer. If a node represents a directory then this sector number will give you an address in directory table. Otherwise it will show you an address in the fat table. (It will also give you the same address in the hard disk file.)
8. Is the next pointer. It will give you the next node's address in the same level (next file/folder in the same directory.)
9. Is the size of the file. If a node represents a directory then size will be zero. It only exists to guide you in the fat file. It depends on your design but you may never use it.

FAT Table: fat.txt

This file will also be in binary mode. You will read these in binary mode as well. This file includes your linked list of data for every file. A file's linked list starting sector exists in its i-node. You will learn the next sector's address from that node and so on. Read the sector and it will show you the next sector's address. "F" terminator will be the last sector of the file. After you read these sectors, you will acquire a linked list of your actual data's addresses.

Real Data: harddisk.txt

This file will include real data for your file system and will be in binary mode also. After you extracted your linked list for a file, you will simply travel through the linked list from beginning to the end and get real data from this file. Remember the sequence is important when accessing data. You have to access the data in the same order you read the list from FAT table. **Remember: A sector in FAT file points you to another sector, not a byte. FAT contains sector indexes not byte indexes.**

Offset(h)	00	01	02	03	04	05	06	07	
00000000	00	00	00	00	00	00	00	0D	....
00000008	00	00	00	00	00	00	00	0B	....
00000010	FF	FF	FF	FF	FF	FF	FF	FF	˙˙˙˙˙˙˙˙
00000018	FF	FF	FF	FF	FF	FF	FF	FF	˙˙˙˙˙˙˙˙
00000020	00	00	00	00	00	00	00	0A	....
00000028	00	00	00	00	00	00	00	13	....
00000030	FF	FF	FF	FF	FF	FF	FF	FF	˙˙˙˙˙˙˙˙
00000038	00	00	00	00	00	00	00	02	....
00000040	00	00	00	00	00	00	00	11	....
00000048	00	00	00	00	00	00	00	07	....
00000050	00	00	00	00	00	00	00	0E	....
00000058	00	00	00	00	00	00	00	14	....
00000060	00	00	00	00	00	00	00	05	....
00000068	FF	FF	FF	FF	FF	FF	FF	FF	˙˙˙˙˙˙˙˙
00000070	00	00	00	00	00	00	00	00	....
00000078	00	00	00	00	00	00	00	15	....
00000080	00	00	00	00	00	00	00	01	....
00000088	00	00	00	00	00	00	00	04	....
00000090	00	00	00	00	00	00	00	06	....
00000098	00	00	00	00	00	00	00	08	....
000000A0	00	00	00	00	00	00	00	0F	....
000000A8	00	00	00	00	00	00	00	12	....

Offset(h)	00	01	02	03	04	05	06	07	
00000000	73	66	69	6C	65	32	31	38	sfile218
00000008	66	69	6C	65	5F	5F	5F	31	file___1
00000010	73	66	69	6C	65	31	32	32	sfile122
00000018	73	66	69	6C	65	31	31	30	sfile110
00000020	73	66	69	6C	65	32	31	35	sfile215
00000028	73	66	69	6C	65	32	31	31	sfile211
00000030	66	69	6C	65	5F	5F	5F	37	file___7
00000038	73	66	69	6C	65	31	32	31	sfile121
00000040	73	66	69	6C	65	32	31	33	sfile213
00000048	73	66	69	6C	65	31	32	30	sfile120
00000050	73	66	69	6C	65	32	31	36	sfile216
00000058	66	69	6C	65	5F	5F	5F	32	file___2
00000060	73	66	69	6C	65	32	31	30	sfile210
00000068	73	66	69	6C	65	32	31	39	sfile219
00000070	73	66	69	6C	65	32	31	37	sfile217
00000078	66	69	6C	65	5F	5F	5F	34	file___4
00000080	66	69	6C	65	5F	5F	5F	30	file___0
00000088	73	66	69	6C	65	32	31	34	sfile214
00000090	66	69	6C	65	5F	5F	5F	36	file___6
00000098	73	66	69	6C	65	32	31	32	sfile212
000000A0	66	69	6C	65	5F	5F	5F	33	file___3
000000A8	66	69	6C	65	5F	5F	5F	35	file___5

Figure 5: Example of FAT (left) and Hard disk (right)file.

Defragmentation

After you read all your file structure from directory table, create all file's data lists from FAT table and read all data from hard disk file to memory, you will simply write this data to "harddisk.txt" file in an order, update file's data lists from FAT table and change the file's starting sector (data pointer) from the directory table. You don't have to use any special defragmentation

algorithm. Start from the root and travel through files. **You have to write all files within a folder AND you can go into subdirectories of that folder.**

Section 2.

In this section you will just create directories which were already in the memory. You examined the folder structure and its memory reflection in figure 1. You will just travel through your list and create directories, write file contents and set necessary attributes. You must use Linux file operation functions to do this. Every file and folder has the information about user and group ID and permissions in i-nodes. You can create folders, set permissions, users and groups with Linux Standard Libraries. You do not have to install any additional libraries.

You will implement your program in C. You have to perform file and directory operations using UNIX system calls. You will/may use the following system calls functions:

```
open(), close(), lseek(), read(), fcntl(),write()
stat(), fstat(), lstat(), unlink()
utime(), chdir(), getcwd(), opendir(), readdir(),
closedir(),rmdir(),mkdir()
```

In order to make you familiar with UNIX system programming API and to use appropriate libraries, we put some restrictions on library usage as follows:

- Use dirent.h ,stat.h libraries and related libraries for file and directory operations.

RULES

- All your input files will have a fixed size of 1Kb(1024 Bytes).
- Your cluster size would be 8 bytes.
- An i-node is 4 sector long and a data point is 1 sector.
- The files would include 16 bytes of strings named by filename and a number for easy to control their content. For example; “sample.avi” includes “sampl__1sampl__2sampl__3”.
- If a node represents a directory than its data pointer will point to another i-node which is a subfolder or a sub-file under it.
- Data or next pointers are an integer which is the i-node’s first SECTOR’S INDEX.
- If a node represents a file, then its data pointer will point a sector in FAT table. This way you will read a data list in order to access the real data.
- Defragmentation will be done level by level. If you will not write a directory’s content to disk, you cannot descend to subfolder’s content.

INPUT

An input file including given operations and an output file will be taken from the command line. Example input and output files will be put on FTP, but firstly design your application independent of files. You can create your own inputs in dev machine. After connecting with ssh to dev, create your own directory structure. Execute “*203defrag your_root_path output_path*” and 3 input files will be created in the desired directory.

OUTPUT

In the first section you will overwrite input files, so format will be the same. For the second part you will create the root directory to the current directory of executable and create others in it.

MAKEFILE

All students have to supply a valid Makefile to compile their experiments. A Makefile is a basic rule file to correctly compile and create executables. You will create a basic one, only "all" rule is enough. Please create an executable named "defrag". It will only be two lines. **Experiments without a proper Makefile will not be graded.**

REPORTS

1. **DESIGN REPORT**

You are expected to write a design report and submit it until **19.12.2013 23:59**. Your design report will include your approach to the problem and will give a brief look of the problem. Design report will include solutions to problems in an algorithmic way. Design report will be graded on 10 points. I will try to grade your design reports within 2-3 days but no promises.

2. **REPORT**

There are no special requirements for the report. If you cite a document, code, solution vs. you have to give a proper citation. If you properly cite your code or algorithm block, it will be not considered as cheating but you will get 0 points for the consecutive part. The report will be graded on 20 points. Please inspect lab report format from these link:
ftp://ftp.cs.hacettepe.edu.tr/pub/dersler/genel/FormatForLabReports.doc

NOTES

- Your experiments will be executed in DEV machine, please make it work on dev before submitting.
- Input file names will be fixed, you can hardcode them into your program.
- Your design report should include a class diagram and your brief solutions for every operation.
- Give your report in the necessary details.
- Save all your work until the assignment is graded.
- The assignment must be original, individual work. Duplicate or very similar assignments are both going to receive 0 (zero). General discussion of the problem is allowed, but do not share answers, algorithms or source codes. Keep in mind that you will get points for every little thing you do.
- You can ask your questions about the experiment with sending your mails to BBM203 newsgroup or Moodle.
- You have to give your experiment files in the given directory structure below;

```
<Student id>
        <src>
                *.c
                *.h
                Makefile
        <report>
```

report.pdf

Experiments which don't have the wanted directory structure get -5 points.

- You can get punishment points if you don't follow the rules defined above.
- You have to supply a valid Makefile and your Makefile have to create an executable named "defrag".
- Your assignment will not be graded unless you supply a valid Makefile.

REFERENCES

- Database Management Systems, Raghu Ramakrishnan, 2nd Edition, Chapter 3: Storing Data: Disks and Files
- Understanding FAT32 File Systems:
  - http://www.csee.umbc.edu/courses/undergraduate/CMSC391/summer04/burt/pjrc_2004_02_07/tech/8051/ide/fat32.html
- Understanding ext2 Filesytems: URL and Bilgisayar İşletim Sistemleri, Ali Saatçi, Bıçaklar Kitabevi, Chapter 6: Kütük Yönetimi
- About defragmentation URL
- For an introduction to the GNU/Linux operating system refer to the following guide:
  - http://www.tldp.org/LDP/intro-linux/html/index.html
- For the whole system API documentation of GNU/Linux operating system refer to the following manual:
  - http://www.gnu.org/software/libc/manual/html_mono/libc.html
- For the whole ANSI C standard library reference refer to the following document:
  - http://www.infosys.utas.edu.au/info/documentation/C/CStdLib.html
- For lots of Turkish documentation about GNU/Linux system usage and programming refer to the following site:
  - http://www.belgeler.org
- For an overview of file and directory operations on UNIX systems refer to the following tutorial:
  - http://users.actcom.co.il/~choo/lupg/tutorials/handling-files/handling-files.html
- For an introduction to automating program compilation with Makefile refer to the following tutorial:
  - http://users.actcom.co.il/~choo/lupg/tutorials/writing-makefiles/writing-makefiles.html.