CS3008 Assessment 2012_2013

Official submission Friday 14 December 2012

In this assessment, you will implement a simple file system that allows you to manage files and directories in a virtual in-memory disk. The file system is to be based on simplified concepts of a File Allocation Table (FAT). Your task is to implement interface functions that allow simple read and write operations on this virtual disk. For this assessment, the virtual disk will be simulated by an array of memory blocks, where each block contains a fixed array of bytes. These blocks are allocated to files and directories, when write operations occur. The allocation of a new block to a file is recorded in the FAT – the FAT records block chains to indicate the sequence of blocks comprising a file. The FAT is an array of integers, a single FAT entry can either be set to UNUSED (indicating that the block on the virtual disk is free to be allocated), or to an array index in the FAT, which is the index of the next disk block for a file, or to ENDOFCHAIN, which indicates the end of a block chain.

For this assessment, we assume that the virtual disk consists of an array of 1024 blocks (MAXBLOCKS), where each block is an array of 1024 bytes (BLOCKSIZE). A block can be either (a) file data (an array of 1024 bytes), (b) directory data (a set of directory entries, as much as can fit within 1024 bytes), or (c) the FAT itself containing information about used and unused blocks. As there are 1024 disk blocks, the FAT has to have 1024 entries. For this assessment, we assume that a FAT entry is a short integer (2 bytes).

The virtual disk has the following layout:

- block 0 is reserved and can contain any information about the whole file system on the disk (e.g. volume name etc.); block 0 is left free
- block 1 and 2 will be occupied by the FAT (we need 2 blocks, because each entry is a short integer, occupying 2 bytes of disk space, and with 1024 entries in the FAT, the FAT needs 2 blocks of disk space
- block 3 is the root directory: a directory block has special structure, containing a list of directory entries

The rest of the virtual disk, blocks 4 – 1023, are either data or directory blocks.

There are two basic functions that are directly interacting with the virtual disk: writeblock() and readblock(). All other function (those you have to implement) are using these to read/write from/to the virtual disk.

The complete public interface of the file system to be implemented is the following (for each assessment step, you have to implement some of them):

void format()

- creates the initial structure on the virtual disk, writing the FAT and the root directory

MyFILE * myfopen (const char * filename, const car * mode) ;

- Opens a file on the virtual disk and manages a buffer for it of size BLOCKSIZE, mode may be either “r” for readonly or “w” for read/write/append (default “w”)

void myfclose (MyFILE * stream)

- closes the file, writes out any blocks not written to disk

Int myfgetc (MyFILE * stream)

- Returns the next byte of the open file, or EOF (EOF == -1)

void myfputc (int b, MyFILE * stream)

- Writes a byte to the file. Depending on the write policy, either writes the disk block containing the written byte to disk, or waits until block is full

void mymkdir (const char * path)

- this function will create a new directory, using path, e.g. mymkdir (“/first/second/third”) creates directory “third” in parent dir “second”, which is a subdir of directory “first”, and “first is a sub directory of the root directory

void myrmdir (const char * path)

- this function removes an existing directory, using path, e.g. myrmdir (“/first/second/third”) removes directory “third” in parent dir “second”, which is a subdir of directory “first”, and “first is a sub directory of the root directory

void mychdir (const char * path)

- this function will change into an existing directory, using path, e.g. mkdir (“/first/second/third”) creates directory “third” in parent dir “second”, which is a subdir of directory “first”, and “first is a sub directory of the root directory

mylistdir (const char * path)

- this function lists the content of a directory

Two files are provided for this assessment: Please copy them from here: http://www.abdn.ac.uk/~csc321/teaching/CS3008/assessment/abdn.only/

- The file filesys.h contains specifications of data structures you will use to implement the interface functions for this file system
- The file filesys.c contains the function writeblock() already implemented for you. Implement a function readblock() accordingly. It also contains a function writedisk() that allows you to save the virtual disk to a file on your real harddisk (such a file has to be submitted).

Your task is to extend filesys.c with additional interface functions (as outlined below). Also, implement a test program, called shell.c, that calls functions you have implemented. The files filesys.h and filesys.c are provided to give you the C structures needed for the implementation. You can also create your own

c structures to complete the assessment. Don't hesitate to extend or change structures in filesys.h, if you see a need for that in order to support your implementation (you may add additional parameters to the file descriptor MyFILE to record additional information, e.g. about the location of a file in the file system etc.).

Requirements

CAS 9-11

Download the files filesys.c and filesys.h from here. Implement the function format() to create a structure for the virtual disk. Format has to create the FAT and the root directory. Write a test program containing the main() function, and call it "shell.c". Your shell.c program should do the following:

- call format() to format the virtual disk
- transfer the following text into block 0: "CS3008 Operating Systems Assessment 2012"
- write the virtual disk to a file (you can call it "virtualdisk9_11").Use the unix command "hexdump" to see what the file contains:

  hexdump –C virtualdisk9_11

CAS 12-14

Implement the interface functions myfopen(), myfputc(), myfgetc() and myfclose(). It is assumed that there is only a root directory and that all files are created there. Extend your test program shell.c

- open a file "testfile.txt" in your virtual disk: call myfopen ("testfile.txt", "w") to open this file
- write a text of size 4kb (4096 bytes) to this file, using the function myfputc() (you may first create a char array of 4 * BLOCKSIZE, fill it with text and then write it to the virtual file with myfputc())
- close the file with myfclose()
- write the virtual disk to a file "virtualdisk12_14"
- test myfgetc() by opening the file again and reading out its content, either to the screen or to a file on your real harddisk
- use the unix command hexdump to check the content of your virtual disk: hexdump –C virtualdisk12_14

CAS 15-17

Add a directory hierarchy on your virtualdisk, which allows the creation of subdirectories. Implement the interface function mymkdir(char * path). A directory can be specified as absolute or relative:

- absolute: "/mydirectory"
- relative: "mydirectory"

A directory may be specified with a path:

- absolute: "/firstlevel/secondlevel/mydirectory"
- relative: "somelevel/somelevelbeneath/mydirectory"

In order to create the directory “mydirectory”, all the subdirectories specified in the path must exist. If you call mymkdir (“/firstlevel/secondlevel/mydirectory”) in your test program shell.c, then the directory hierarchy consisting of Root->firstlevel->secondlevel must exist, before you can create “mydirectory” in the parent directory “secondlevel”. If these directories don’t exist, they have to be created.

Use strtok_r() from the C standard library to tokenize a path string (look up its usage).
Implement a function mylistdir (char * path) that lists the content of a directory.

CAS 18-20

Implement the interface function mychdir(char * path), using the global variable “currentDir” as specified in the header file. Change the creation of a new subdirectory: add two default entries (as we are used to under Unix etc.):

- the directory entry “.” points to the directory itself
- the directory entry “..” points to the parent directory

Implement two remove functions:

- remove a file: myremove(char * path) removes a file, the path can be absolute or relative
- remove a directory: myrmdir(char * path) removes a directory, if it is empty, the path can be absolute or relative

Look into fileys.h. A directory entry direntry_t uses a char array of 256 bytes for the file or directory name. Very few files may have such a long name. Try to implement directory entries with variable size.

Optional for extra points:
Saveguard the manipulation of the FAT table in a multithreaded application. Introduce a lock variable and store it in block 0 (you can introduce an extra struct for block 0 that contains, among other things, a volume name and this lock variable). The lock variable indicates either a LOCKED or UNLOCKED state of the virtual disk. Use mutexes to change the lock in a thread. Run tests by implementing a multithreaded shell.c.

Explanations

Please use filesys.h and filesys.c as a starting point. In filesys.h, the virtual disk is an array of blocks the size of BLOCKSIZE. There are only two functions that access this virtual disk directly:

- writeblock (diskblock_t * block, int blockIndex): write a complete block to the disk
- readblock (diskblock_t * block, int blockIndex) : reads a complete block from this virtual disk

For moving blocks from / to the virtual disk, two methods can be used

- byte-wise copying with a for loop:

  for (bytePos = 0; bytePos < BLOCKSIZE; bytePos++) virtualdisk[blockIndex].data[bytePos] = block->data[bytePos];

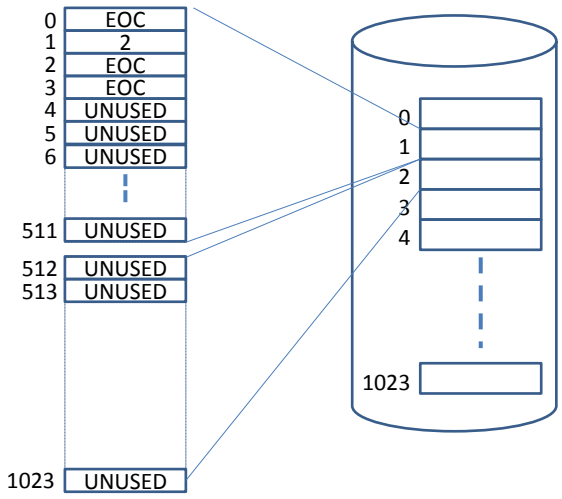
- using the system call memmove() :

```
memmove ( virtualdisk[blockIndex].data, block->data, BLOCKSIZE ) ;
```

A disk block is defined as a union, so that it can be three things at the same time:

- a simple data block (an array of bytes of length BLOCKSIZE): each file has one or more of these blocks
- a directory block: each directory has one or more of these blocks, it has a structure, defined by the type direntry_t
- a FAT block: as things are defined in filesys.h, there are 1024 blocks, therefore the FAT has to have 1024 entries, each FAT entry is of type short, therefore the whole FAT is an array of short integers and we need 2 FAT blocks to store the whole table

The FAT shows which blocks belong to a file or a directory. If a file grows, it may need more space and additional blocks have to be allocated to the file. This only needs some manipulation of the FAT itself – a block chain has to be created. The same is true for a directory – if it has more entries than fit on a single block, additional blocks have to be allocated.



When a file is opened, the file descriptor MyFILE points to a buffer of BLOCKSIZE bytes. The functions myfputc() and myfgetc() operate on this buffer:

- in case of write operations: when this buffer becomes full, it has to be written (copied) to the virtual disk and a new block has to be allocated (an UNUSED block as indicated in the FAT)
- in case of read operations: the first block of the file has to be loaded (copied) from the virtual disk when opened; each read pushes a position pointer to the end of the buffer, when it

becomes BUFFERSIZE then the next block from the virtual disk has to be loaded; for this the chain in the FAT has to be followed to find the next block of the file

Starting the project

For the implementation of the format() function, you have to do three things

- initialize block 0:
  - define a variable diskblock_t block as a local variable in format() and fill it with '\0'
  - use strcpy(block.data, "CS3008 blabla etc") to copy the recommended text into block 0
  - use writeblock(&block, 0) to write this block to the virtual disk
- create the FAT and write it to the virtual disk, starting at block 1
  - the FAT itself needs more than one block, this is a chain that has to be put into the FAT iself
  - if the FAT needs two blocks on the disk, then it will occupy block 1 and 2, therefore there is a chain "block1->block2":
    - this is expressed by the following assignments
      FAT[1] = 2 ;
      FAT[2] = UNUSUED ;
  - If you decide to create a larger disk with more blocks, then you need a larger FAT, and the chain for the FAT is longer.
  - Write the FAT blocks to the virtual disk: for writing the FAT, you may implement an extra function writeFAT()
- Create one directory block for the root directory
  - A directory block has an array of directory entries (which can be files and directories) and a "next" pointer that points to the next free list element for a directory entry
  - Initialize the "next' pointer to 0
  - Write this block to the virtual disk

Use make files to compile your programs.

Submission Requirements

You are required to submit in electronic as well as in paper form:

1. one zip file called **`cs3008_assessment_<your_username>.zip`**
   Send this zip file to m.j.kollingbaum@abdn.ac.uk as an attachment.

2. A single printed copy of the source code plus a short one-page manual how to run your shell.c, and with a signed submission sheet.

All submissions should be documented, and this documentation must include your name and userid at the top of each file. Please submit (a) the complete source code (plus any necessary make files, text files, configuration files etc to compile and run your submission) and (b) a short report (one/two/more pages) describing your submission and how to operate your application. Make a zip file containing all this information and send it to the email address above.
For the printed version of your submission: To ensure that the printed source code is not excessively long, I suggest that you print the code in two-column landscape mode. The unix utility enscript provides this and more; for example, try out the following on some Java source file, e.g. shell.c, and look at the resulting PDF (shell.pdf): enscript -2 –r –Ejava –o – shell.c | ps2pdf – shell.pdf

DeadlineSubmission Procedure

The official submission date for handing in your in-course assessment is: 23:00pm, Friday 14 December 2012 (electronic submission).

An extension for submitting your in-course assessment is granted (no penalty) until Monday 7 January 2013, 12:00pm

You can submit any time before that date. The electronic submission counts. Standard lateness penalties for submissions beyond Monday 7 January 2013 apply.

Submission Procedure

You are required to submit your work electronically, using the following method:

- Send an email to m.j.kollingbaum@abdn.ac.uk with the following **exact** subject line:
  - "CS3008 Submission Assessment 2012_2013"
  - Attached to this email must be your zip file containing documentation and source code.