File Systems

Overview

This assignment focuses on file systems and their implementation. You will write a minimum file system using a simulated disk. Your file system will setup the data blocks to manage general information about the files, the i-nodes for file creation, and the free space.

The Program

Create a program that sets up and formats a file system using a simulated disk as well as performs specific testing functions. The simulated disk reads and writes 1 KB binary data blocks from a random access file that is setup on your actual file system. Your file system consists of a super block, a list of i-node blocks, and data blocks. The super block is the first block (block 0) on disk. It contains information about the number of blocks on the file system, the number of i-nodes, and the index of the first block in the free list. The list of i-node blocks following the super block is needed to implement files. The following picture illustrates the layout of the disk:

superblock	i-node blocks	data blocks

Disk Formatting

Formatting a disk requires the following operations: 1) a super block is written to disk, 2) a list of empty i-nodes is created and written to disk, and 3) a free-space list is setup to indicate the data blocks that are free. Review the code in *filesystem.c* and *filesystem.h* for functions that you need to implement. The files contain the needed support to read and write integers from and to a byte array. In your implementation, you must choose the number of i-nodes that you want to store on the file system. Keep in mind that each file must have an i-node that describes the file's data blocks. This means, your file system must have as many i-nodes as you want files to exist.

Free-Space Management

For free-space management, you will use a linked-list technique as discussed in class that links free blocks together. For example, if 100 is the first free block (as indicated by the superblock), the block contains the address (an integer) of the next free block, and so forth. The last free block contains -1 as the address of the free block to indicate the end of the free list.

Test Code

Write test code that prints formatting information to the screen after completion of the formatting process. Your test code should display the information in the superblock, the number of i-nodes free and in use, the number of disk blocks free and in use. In addition, your test code should perform consistency checking. This means that the test code must determine that the number of free disk blocks plus the number of used disk blocks for data, superblock, and i-nodes is equal to the total number of disk blocks of the disk. Finally, your test program must create a binary file, write and read data of arbitrary size to the file, and close the file. The IO operations should closely resemble the equivalent IO operation of a regular file system. This means the following:

a) files need to be created when they are opened for write operations and don't exist.
b) open files grow in size with every write operation allowing the data to be appended to the existing data in the file,
c) files cannot be opened for read operations if they don't exist,
d) files exist after they are closed,
e) all open files are managed in a system-wide open file table.

Keep in mind that several independent write operations to a file must ensure that the file pointer moves along as bytes are written to the file indicating after every complete write operation the end of the file where the next byte can be written. And as data is written into the blocks, the last block of a file may only be partially filled.

IMPLEMENTATION SUGGESTIONS

To start this project, download the C files in the course shell in *eLearning*. File *disk.c* and *disk.h* implements the simulated disk for reading and writing binary data blocks. File *filesystem.c* and *filesystem.h* implements a rudimentary file system. The header files s*uperblock.h* and *inode.h* describe the superblock and i-nodes data structures. Both superblock and an i-node must be written into a single disk block.

DELIVERABLES

Your project submission should follow the instructions below. Any submissions that do not follow the stated requirements will not be graded.

1. Follow the submission requirements of your instructor as published on *eLearning* under the Content area.
2. You should have at a minimum the following files for this assignment:
    a. `disk.c`, `disk.h`
    b. `filesystem.c`, `filesystem.h`
    c. `superblock.h`, `inode.h`
    d. `main.c`
    e. Makefile
    f. README

The README should describe the test program, how to run the program, and any issues you encountered. If you do not include comments in your source code and refactor your code adequately, points will be deducted.

GRADING

This project is worth 100 points total. The points will be given based on the following criteria:

- 60 points for implementation of file system
- 25 points for test code
- 10 points for appropriate refactoring and documentation of source code
- 5 points for README

DUE DATE

The project is due Wednesday, November 20th by 3:00 pm to the Dropbox of project 6 in *eLearning*. I will not accept submissions emailed to me or the grader. Upload ahead of time, as last minute uploads may fail. Please review the policy in the syllabus regarding late work.