BUILDING A UNIX-TYPE FILE SYSTEM

OVERVIEW

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

THE PROGRAM

Create a program that sets up and formats a file system using a simulated disk and performs specific testing functions. The simulated disk reads and writes 1 KB binary data block from a random access file set up on your actual file system. Your file system consists of a superblock, a list of i-node blocks, and data blocks. The superblock is the first block (block 0) on the 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 superblock 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 superblock 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.cpp* or *filesystem.hpp* for functions that you must implement. These 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 that your file system must have as many i-nodes as you want files to be created.

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 must:

1) Display the information in the superblock, the number of i-nodes free and in use, and the number of disk blocks free and in use.
2) Perform consistency checking. This means that the test code must determine that the sum of the number of free disk blocks and the number of used disk blocks for data, superblock, and i-nodes is equal to the total number of disk blocks of the disk.
3) Create a binary file using your file system, 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 must 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 reading operations if they don't exist.
    d) Files exist in the directory 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. 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 *Canvas*. Files *disk.cpp* and *disk.hpp* implement the simulated disk for reading and writing binary data blocks. Files *filesystem.cpp* and *filesystem.hpp* implement a rudimentary file system. The header files *superblock.hpp* and *inode.hpp* describe the superblock and i-node data structures. Both the superblock and a single i-node must each fit within a single disk block.

DELIVERABLES

Your project submission must follow the instructions below. I will not grade any submissions that do not follow the stated requirements.

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

The README should describe any issues you may have encountered. If you do not include comments in your source code and refactor your code adequately, points will be deducted.

TESTING & EVALUATION

Your program will be evaluated on the Department's public Linux servers according to the steps shown below. Notice that warnings and errors are not permitted and will make grading quick!

1. Program compilation with Makefile. The options –*g* and –*Wall* must be enabled in the Makefile. See the sample Makefile that I uploaded in *eLearning*.
   - If errors occur during compilation, there will be a substantial deduction. The instructor will not fix your code to get it to compile.
   - If warnings occur during compilation, there will be a deduction. The instructor will test your code.
2. Perform an evaluation run. At a minimum, the test run addresses the following questions.
   - Does your file system format the disk and perform a correct consistency check?
   - Does your file system allow users to create new files and open existing files?
   - Does your file system read large files?
   - Does your file system correctly read binary files?
   - Bonus: Does your file system write large files, including binary files?

DUE DATE

The project is due as indicated in the schedule in the syllabus and calendar in *Canvas*. Upload your complete solution to the dropbox. I will not accept submissions emailed to me or the grader. Upload ahead of time, as last-minute uploads may fail.

GRADING

This project is worth 100 points in total. The rubric used for grading is included below. Keep in mind that there will be deductions if your code does not compile, has memory leaks, or is otherwise poorly documented or organized.

Submission	**Perfect**	**Deficient**		
Canvas	5 points individual files have been uploaded	0 points files are missing		
Compilation	**Perfect**	**Good**	**Attempted**	**Deficient**
Makefile	5 points makefile works; includes clean rule	3 points make files compiles code; missing clean rule	2 points missing rules; doesn't compile project	0 points makefile is missing
compilation	10 points no errors, no warnings	7 points some warnings	3 points many warnings	0 points many errors
Documentation & Program Structure	**Perfect**	**Good**	**Attempted**	**Deficient**
documentation & program structure	10 points follows documentation and code structure guidelines	7 points follows mostly documentation and code structure guidelines; minor deviations	3 points some documentation and/or code structure lacks consistency	0 points missing or insufficient documentation and/or code structure is poor; review sample code and guidelines

File System Implementation	Perfect	Good	Attempted	Deficient
creates i-nodes and superblock on disk	15 points correct, completed	11 points minor errors	4 points incomplete	0 points missing or does not compile or program does not run
implements free space management	20 points correct, completed	14 points minor errors	6 points incomplete	0 points missing or does not compile or program does not run
implements system call openf(), readf(), writef(), and closef()	20 points correct, completed	14 points minor errors	6 points incomplete	0 points missing or does not compile or program does not run
implements test code	15 points correct, completed	11 points minor errors	6 points incomplete	0 points missing or does not compile or program does not run
Bonus: implement system call writef()	10 points correct, completed	7 points minor errors	3 points Incomplete	0 points missing or does not compile or program does not run

I will evaluate your solution as attempted or insufficient if your code does not compile. This means, if you submit your solution according to my instructions, document and structure your code properly, and provide a makefile, but the submitted code does not compile or crashes immediately, you can expect at most 20 out of 100 points. So be sure your code compiles and executes properly.