CSC 351 Operating Systems
KPA Semester Project Description
Fall 2023
Dr. Wheat

This is a semester-long team project that is broken into two phases with five deliverables per phase. A rubric associated with this project is posted on D2L.

Phase 1 is to implement an ext2 file system simulation, as described in section 10.6.3 of the textbook.

Phase 2 is to implement a shell program, similar in behavior as the Bourne shell to exercise the phase 1 implementation.

Each phase has the following five deliverables:

1. Requirements document
2. Design Document
3. Gannt Chart for the development of the project
4. A demonstration of the program working properly
5. A teaming assessment.

Each team member is responsible for submitting each deliverable item. The first four deliverables shall be the same packages for each team member on the same team. The fifth deliverable is unique to each team member.

NOTE: The two phases overlap in deliverables. I.e., phase 2 starts long before phase 1 is completed.

Assignment Description

Overview

The filesystem (FS) shall be implemented in a user-level C++ program consisting of an appropriate number of source and header files. The FS shall be sized at 2GB of raw storage, and shall be implemented with in a file on the computer. The program shall have the have the ability to create the file system, upon request. Otherwise, the program shall service file operations as requested by another program, the Shell. The two programs shall communicate via TCP sockets. When the FS is told to shutdown, it the FS program shall terminate cleanly. Upon starting again, the FS shall be in the same state as it was when it shut down.

Phase 1 Description

The FS shall support at least the fields noted in Figure 10-33 of the textbook. Other fields can be created/used but must be fully documented. The FS shall NOT implement the file-descriptor table as described in the textbook. The FS shall use 4KB block size and, for each time a file needs to be lengthened, it shall allocate 8 more blocks.

The FS shall support the system calls noted in Figure 10-27, except that it shall NOT implement the pipe() and fcntl() calls. However, the function names shall be prepended with “my_”; e.g., create() will be called my_creat().

Figure 10-28 shows the fields to be returned by the mystat() function. The device name (Figure 10-28) shall be “mydev”.

Similarly, the FS shall implement all the system calls in Figure 10-29.

The demonstration of the FS will require a client application to be developed, much along the same lines as the Phase 2 shell. It will be a shell that allows the following calls to be made:

- ls – provide the equivalent of a “ls -l” long output of the current directory.
- cd – change directory to the given directory
- mkdir – make a directory with the name given.
- lcp – copy a host file to the current directory in the FS
- Lcp – copy a FS file from the current directory to the current directory in the host system
- shutdown – shutdown and terminate the FS and the shell.
- exit – causes the Shell to terminate, allowing the FS to make a connection to yet another shell.

The FS process shall be given a file name for its backing store. If the file does not exist, the FS shall initialize the FS per specification. If the file exists, it is an existing image and the FS shall open it and use it. For a newly created FS, it shall have a single directory owned by 0:0 and have the permission bits set to 777. It is up to you to develop how to know when the FS is being created versus when the FS already exists and is being opened for service.

For any file or directory creation, use the same uid/gid bits as the user has on the host system.

This first shell may be a prototype of the Phase 2 shell.

The FS demonstration needs to show each of these commands working for small and large files, with multiple directories.

The teaming report shall be done in a Excel spreadsheet. A template is provided in D2L.

Phase 2 – The Shell

The shell is a client to the FS program, to which it will communicate via sockets. It is up to you as to how you coordinate which sockets are used.

The shell shall provide a prompt to the user; the prompt shall be “current_directory->” where current_directory is the full path of the current working directory.

It shall take only one command per input line.

Each command is a key word followed by zero or more arguments.

Each command is done without dash options; for example, there will be no -R to chown.

The arguments can be absolute or relative paths. There will be no need for wildcards in filenames.

The commands are:

- Those in Phase 1, but now allowing paths
- chown
- rmdir

- cp – copy one or more FS files to a destination. When only one file is being copied, the destination can be either a filename or a directory name.
- mv – similar to cp, but moves the file(s)
- rm – removes one or more files
- ln – create a hard link; no need to support symbolic links.
- cat – print the contents of one or more files to the screen; I/O redirection does not need to be provided.

It is recommended that Lex and Yacc (or flex and bison) be considered to develop the shell.

The teaming report shall be done in a Excel spreadsheet. A template is provided in D2L.

Deliverables

For each phase, the deliverables shall be:

- All the software files (.cpp, .h) and a makefile to build both the FS and the shell
- The requirements, design, and mpp documents.
- The teaming report.
- A zoom link containing a video demonstration of the FS in action, narrated by the team. The demonstration shall include every step, from the making of the programs, to the initial creation of the file system, to the use of the shell (with many examples), and at least two demonstrations showing the shutdown of the filesystem with subsequent resumptions, clearly showing no loss of data.