Project 3: Design and Implementation of File System

April 10, 2023

1 Objectives

1. Design and implement a basic disk-like secondary storage server.
2. Design and implement a basic file system to act as a client using the disk services provided by the server designed above.
3. Use socket API.

2 Problem Statement

This project will be developed in several steps to help you understand the concepts and encourage good modular development. It is very important that you adhere to the specifications given.

Step 1: Design a basic disk-storage system

Description: Implement a simulation of a physical disk. The simulated disk is organized by cylinder and sector. You should include in your simulation something to account for track-to-track time. (using usleep(3C), nanosleep(3R) etc.). Let this be a value in microseconds (may not be an integer), passed as a command-line parameter to the disk-storage system. Also, let the number of cylinders and the number of sectors per cylinder be command line parameters. Assume the sector size is fixed at 256 bytes. Your simulation should store the actual data in a real disk file, so you'll want a filename for this file as another command line option. (You can use file and file API to simulate the storage representing your disk, or you can simpily use mmap(2) system call to manipulate the actual storage.)

Protocol: Here, we provide a protocol. Your program must understand and response to the following commands.

- I: Information request. The disk returns two integers representing the disk geometry: the number of cylinders and the number of sectors per cylinder.
- **R c s:** Read request for the contents of cylinder *c* sector *s*. The disk returns **Yes** followed by a writespace and those 256 bytes of information, or **No** if no such block exists.
- **W c s data:** Write a request for cylinder *c* sector *s*. The disk returns **Yes** and writes the *data* to cylinder *c* sector *s* if it is a valid write request or returns **No** otherwise.
- **E:** Exit the disk-storage system.

Test: You can prepare some testdata to test your disk-storage system. You need to read requests from **STDIN** and write a log file (**disk.log**) when the disk-storage system is running.

- Input Format
  Input contains only requests, with exactly one request per line. Each request is in the format of **I**, **R c s**, **W c s data**, or **E**. The request **E** only appears in the last line. You can assume the input is legal, so you don't need to report any errors.

- Output Format
  For each request, output one line.
  For **I**, output two integers, separate by a writespace, which denotes the number of cylinders and the number of sectors per cylinder.
  For **R c s**, output **No** if no such block exists. Otherwise, output **Yes** followed by a writespace and those 256 bytes of information.
  For **W c s data**, Write **Yes** or **No** to show whether it is a valid write request or not.
  For **E**, output **Goodbye!**.

Command line:
./disk <#cylinders> <#sector per cylinder> <track-to-track delay> <#disk-storage-filename>

Step 2: Design a basic file system

Description: Implement a file system. The file system should provide operations such as: initialize the file system, create a file, read the data from the file, write a file with given data, append data to a file, change data in a file, remove a file, create directories, etc.

Protocol: Here, we provide a protocol. Your program must understand and response to the following commands.

- **f:** Format. This will format the file system on the disk, by initializing any/all of the tables that the file system relies on.
- **mk *f*:** Create file. This will create a file named f in the file system.
- **mkdir *d*:** Create directory. This will create a subdirectory named d in the current directory.
- **rm *f*:** Delete file. This will delete the file named f from the current directory.
- **cd *path*:** Change directory. This will change the current working directory to the path. The path is in the format in Linux, which can be a relative or absolute path. When the file system starts, the initial working path is /. You need to handle **.** and **..** .
- **rmdir *d*:** Delete directory. This will delete the directory named d in the current directory.
- **ls:** Directory listing. This will return a listing of the files and directories in the current directory. You are also required to output some other information, such as file size, last update time, etc.
- **cat *f*:** Catch file. This will read the file named *f*, and return the data that came from it.
- **w *f l data*:** Write file. This will overwrite the contents of the file named f with the l bytes of data. If the new data is longer than the data previously in the file, the file will be made longer. If the new data is shorter than the data previously in the file, the file will be truncated to the new length.
- **i *f pos l data*:** Insert to a file. This will insert into the file at the position before the pos^{th} character(0-indexed), with the l bytes of data. If the *pos* is larger than the size of the file. Just insert it at the end of the file.
- **d *f pos l*:** Delete in the file. This will delete contents from the *pos* character (0-indexed), delete l bytes, or till the end of the file.
- **e:** Exit the file system.

Test: You can prepare some testdata to test your file system. You need to read requests from **STDIN**, and write a log file (**fs.log**) when the file system is running.

- Input Format
  Input contains only requests, with exactly one request per line. Each request is in the format of **f**, **mk *f***, **mkdir *d***, **rm *f***, **rmdir *d***, **ls**, **cat *f***, **w *f l data***, **i *f pos l data***, **d *f pos l***, **cd *path***, or **e**. The request e only appears in the last line. You can assume the input is legal, so you don't need to report any errors.

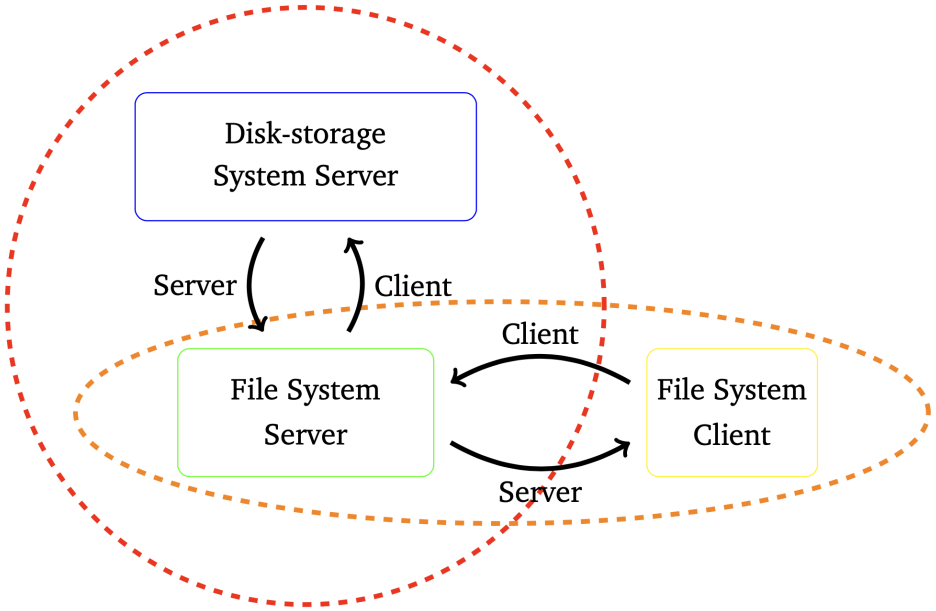

Figure 1: File System Architecture

- Output Format
  For each request, output one line.
  For **f**, output Done.
  For **mk** ***f***, **mkdir** ***d***, **rm** ***f***, **rmdir** ***d***, **w** ***f l data***, **i** ***f pos l data***, **d** ***f pos l***, and **cd** ***path***, output **Yes**, if successful, or **No** otherwise.
  For **cat** ***f***, if the file *f* exists, output its contents. Otherwise, output **No**.
  For **ls**, output all the files and directories in the current directory. First, output all the files, separate by a space, in lexicography order. Then output all the directories, separate by a space, in lexicography order. Output an '&' between the files and directories.
  For **e**, output **Goodbye!**.

Command line: ./fs

Step 3: Work together

In this part, we will combine the disk-storage and file systems.

First, change the disk-storage system to a disk-storage server, and let the file system be the client of the server. Your server needs to output the track-to-track time between two operations. In *Step 2*, the information of your file system is stored in the memory. Now you are required to store them on the disk, using the disk-storage system. Note that all the information, including files and the structure of your file system, needs to be stored on the disk, as the file system module could be shut down and restarted, and the data should be **persistent**.

Second, treat the file system as a network file system server, and write a client for this server. You can output some debugging information or logs on the file system server to let you know the states your server.

Command line:
./disk <#cylinders> <#sectors per cylinder> <track-to-track delay> <#disk-storage-filename> <DiskPort>
./fs <DiskPort> <FSPort>
./client <FSPort>

Step 4: Submit your project

Your project **must** be organized as the following structure.

- Prj3+StudentID.tar

- step1
  - disk.c
  - makefile
  - ...
- step2
  - fs.c
  - makefile
  - ...
- step3
  - disk.c
  - fs.c
  - client.c
  - makefile
  - ...
- report.pdf
- Prj3README
- ...

3 Implementation Details

In general, the execution of any of the programs above is carried out by specifying the executable program name followed by the command line arguments.

1. Use Ubuntu Linux and GNU C Compiler to compile and debug your programs. GCC-inline-assembly is allowed.
2. See the man pages for more details about specific system or library calls and commands: sleep(3), gcc(1), mmap(2), usleep(3), nanosleep(3), etc.
3. When using system or library calls, you have to make sure that your program will exit gracefully when the requested call cannot be carried out.
4. One of the dangers of learning about forking processes is leafing unwanted processes active, which wastes system resources. So please ensure that each process/thread is terminated cleanly when the program exits. A parent process should wait until its child processes finish, print a message, and then quit.
5. Your program should be robust. If any call fails, it should print an error message and exit with an appropriate error code. So please always check for failure when invoking a system or library call.

4 Material to be submitted

1. Compress the source code of the programs into Prj3+StudentID.tar file. Use meaningful names for the file so that the contents of the file are obvious. A single makefile that makes the executables out of any of the source code should be provided in the compressed file. Enclose a README file that lists the files you have submitted along with a one-sentence explanation. Call it Prj3README.
2. Please state clearly the purpose of each program at the start of the program. Add comments to explain your program.
3. Test runs: It is very important that you show that your program works for all possible inputs. Submit online a single typescript file clearly showing the working of all the programs for correct input as well as graceful exit on error input.

4. Submit a well-organized report. Since this project is really complicated, your report will be the **most important** thing to help TA grade your project this time.

5. Send your Prj3+StudentID.tar file to Canvas. (e.g. Prj3+5108888888.tar)

6. Due date: May 26, 2023, submit on-line before midnight.

7. Demo slots: May 27-28, 2023. Demo slots will be posted on the shared document. Please sign your name in one of the available slots.

8. You are encouraged to present your design of the project optionally. The presentation time is 8-9:40, May 30, 2023. Please pay attention to the course website.