CSE421/521 Introduction to Operating System
Spring 2010
Project #3
Design and Implementation of File System and Disk Storage System

March 30, 2010

1 Objective

- Design and implement a basic disk-like secondary storage server.
- Design and implement a basic file system to act as a client using the disk services provided by the server designed above.
- Study and learn to use the *socket* API. Sockets will be used to provide the communication mechanism between (i) the client processes and the file system, and (ii) the file system and the disk storage server.

2 Problem Description

This project will be developed in steps to help you get a good understanding of the concepts, and to encourage good modular development. The first two parts will familiarize you with the socket-API. Sockets will serve as the communication backbone for the disk system that you will build in the later parts.It is important that you adhere to the specifications given.

1. (10 points) **Basic client-server:** Create two programs: a client and a server. Let the two communicate through **internet-domain sockets**.You will NOT use unix-domain sockets. Also each student has been assigned a range of 10 port numbers (you need only one), the starting port number is available on ublearns along with your grades. Do not use anyother port numbers than the ones assigned to you. Also refrain from using low port numbers since there are reserved for system services. The client will pass a string "Who are you?" to the server, the server replies with the string that contains "your name, current data and time" and returns it to the client which prints it out.The client also prints out its own the current time and data on the client side.Make sure you prefix printouts so that we know which is the output from the server and which is output from the client.

2. (10 points) **Directory listing server:** Next, extend part one to implement a directory listing server, and sample client. The directory listing server waits for clients to connect to it. The client provides the server with some data. The data is in the form of an array of parameters to the `ls` program (like the one you used in the first project). The server forks a child process to handle the request. In the child, using one of the `exec*` family of system calls, the `ls` command is executed with the appropriate parameters as indicated by the client. The output of the `ls` program should be written back via the socket to the client. At the client end, display the results transmitted by the server.

3. (35 points) **Basic disk-storage system:** Implement, as a inet-domain socket server, 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, passed as a command-line parameter to the disk server program. Also,

let the number of cylinders and number of sectors per cylinder be command line parameters. Assume the sector size is fixed at 128 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'll probably find that the `mmap(2)` system call provides you with the easiest way of manipulating the actual storage. However you are allowed to use file and file API to simulate the storage representing your disk)

The Disk Protocol

The server must understand the following commands, and give the following responses:

- `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 '1' followed by those 128 bytes of information, or '0' if no such block exists. (This will return whatever data happens to be on the disk in a given sector, even if nothing has ever been explicitly written there before.)
- `W` *c s l* `data`: write request for cylinder *c* sector *s*. *l* is the number of bytes being provided, with a maximum of 128. The `data` is those *l* bytes of data. The disk returns '1' to the client if it is a valid write request (legal values of *c*, *s* and *l*), or returns a '0' otherwise. In cases where $l < 128$, the contents of those bytes of the sector between byte *l* and byte 128 is undefined, use zero-fill.

The data format that you *must* use for *c s* and *l* above is a regular ASCII string, followed by a white-space (space, tab or newline) character. So, for example, a read request for the contents of sector 17 of cylinder 130 would look like: `R␣130␣17␣`. And then 129 bytes of data would be returned: the character `1`, followed immediately by the 128 bytes read from that sector.

Simple Disk Client

The clients that you need to write here are mostly for testing purposes. The real use of this "disk" will be the filesystem implemented in the later parts of this project. This is a command-line driven client that should work in a loop, having the user type commands in the format of the above protocol, send the commands to the disk server, and display the results to the user.

4. (35 points) **File system server:** Implement a flat filesystem that keeps track of files in a single directory (table). The filesystem should provide operations such as: initialize the filesystem, create a file, read the data from a file, write a file with given data, append data to a file, remove a file, etc..

   The following features are **not required** in your implementation:
   - The ability to create and use subdirectories.
   - Advanced access to the filesystem such as filesystem status report (free space, number of files, percentage of used space, fragmentation etc.).
   - Filesystem integrity checks.
   - File permissions.

   You will find that in order to provide the above file-like concepts, you will need to operate on more than just the raw block numbers with which your disk server provides you. You will need to keep track of things such as which blocks of storage are allocated to which file, and the free space available on the disk. A file allocation table (FAT) can be used to keep track of current allocation. Free space management involves maintaining a list of free blocks available on the disk. Two alternative designs are suggested: a bit vector (1 bit per block) or chain of free blocks. Associated with each block is a cylinder# and sector#. Writing to a file gets converted to writing into a specific cylinder# and sector#. Note that all this information needs to be stored on the disk, as the filesystem module could be shut down and restarted and the disk data should be persistent.

   Implement this file system server as another inet-domain socket server. So, this program will be a server for one inet-domain socket; and also be a client to the disk-server inet-domain socket from the previous parts.

The Filesystem Protocol

The server must understand the following commands, and give the following responses. (See the Appendix for some *suggested, but not required,* algorithms to implement some of these operations.)

- `F`: format. Will format the filesystem on the disk, by initializing any/all of tables that the filesystem relies on.
- `C` f: create file. This will create a file named f in the filesystem. Possible return codes: 0 = successfully created the file; 1 = a file of this name already existed; 2 = some other failure (such as no space left, etc.).
- `D` f: delete file. This will delete the file named f from the filesystem. Possible return codes: 0 = successfully deleted the file; 1 = a file of this name did not exist; 2 = some other failure.
- `L` b: directory listing. This will return a listing of the files in the filesystems. b is a boolean flag: if '0' it lists just the names of all the files, one per line; if '1' it includes other information about each file, such as file length, plus anything else your filesystem might store about it.
- `R` f: read file. This will read the *entire* contents of the file named f, and return the data that came from it. The message sent back to the client is, in order: a return code, the number of bytes in the file (in ASCII), a white-space, and finally the data from the file. Possible return codes: 0 = successfully read file; 1 = no such filename exists; 2 = some other failure.
- `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 previous 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. A return code is sent back to the client. Possible return codes: 0 = successfully written file; 1 = no such filename exists; 2 = some other failure (such as no space left, etc.).

For testing/demonstration purposes, you need to implement a command-line driven client, similar to the one that you wrote for the disk server. It should work in a loop, having the user type commands in the format of the above protocol, send the commands to the disk server, and display the results to the user.

5. (10 points) Technical Report. Provide a detailed printed technical report. This should include the performance graphs and discussion for the disk scheduler.

3 Material to be Submitted

1. Submit the **source code** for all the programs. Use meaningful names for the file so that the contents of the file is obvious from the name. Also provide a README file.
2. Test runs: It is very important that you show that your program works for all possible inputs. Submit a script file for each program clearly showing the working for correct input as well as graceful exit on error input.
3. You are required to submit a Makefile for your project. It should be set up so that just typing `make` in your submission directory should correctly compile all programs for all parts.
4. You need to submit a printed Technical Report. This Technical Report should be a professional looking document.It should contain:
   (a) A users manual, describing how to interact with your programs
   (b) A technical manual, describing your design disk, and of your filesystem. You must give complete technical details of the format of all data-structures used in your program, a class diagram, as well as the main algorithms, etc..

4 Due Date

4/24/2010. Submit on-line before midnight.