Dhirubhai Ambani
Institute of Information and Communication Technology

IT-215, Systems Software

Prof. Sanjay Chaudhary

Note-Sync Application

Aditya Basu
2010-01-096

Rishabh Jain
2010-01-092

Contents

1 Problem Definition **2**
- 1.1 Approach . . . 2
- 1.2 Functionalities . . . 2
- 1.3 Concepts Used . . . 3

2 Kit **3**
- 2.1 Essential Tools to Compile and Build from source . . . 3
- 2.2 Compiling and Building . . . 4
- 2.3 Code in Action . . . 4
- 2.4 Internals . . . 4
- 2.5 Logging . . . 4

3 Testings **5**
- 3.1 Operating Systems . . . 5
- 3.2 Editors . . . 5

4 Others **5**
- 4.1 Important Assumptions . . . 5
- 4.2 Source Code Organization . . . 5

---

1 Problem Definition

Make an application which syncs a document (a text file) across all clients which are registered and connected to the server.

1.1 Approach

Make a server which listens to client requests and when a client wants to connect to the server with proper credentials, register the client and start syncing. The username, password and data are not encrypted. It will be plainly sent to and fro the server and client.

1.2 Functionalities

- The application generates a patch file on the client side and then pushes it to the server.

- The server uses the patch file to update its copy and then pushes it to all other clients (other than the one which sent it the patch).
- Our own command interpreter which interprets pre-defined command sequences.
- A parser which parses a *configuration file* before start-up, so that settings can be loaded dynamically before each start-up.

1.3 Concepts Used

- Network programming with sockets *for connection*
- Process Management *for managing clients*
- Threading *in client for 2 sockets*
- Signal Handling *for server to child notification*

2 Kit

2.1 Essential Tools to Compile and Build from source

- make
- makedepend
- diff
- patch
- gcc *for compiling C files*
- POSIX Thread Building Library (*pthread*)
- kqueue *on Mac OS*
- inotify *on Linux*

2.2 Compiling and Building

The code is written in *C programming* language. The application uses *make* to simplify the building process. To compile, open the terminal and change the directory to the base directory of the project. Type *make* and the program would be compiled. To build from a clean source, type *make clean*. This would remove all traces of previous compilations and then you can again compile using *make*.

2.3 Code in Action

Compilation would yield 2 executable images named *client* and *server*. The server does not need any command line options, while the client takes the *hostname/IP* of the server computer. If nothing is specified then, the hostname is taken as *localhost*. Each of these executables requires that the *settings.txt* file is in the same directory as that of the executable or else the program would crash at runtime. All essential configurations are placed in this file.

There are 2 folders named *pool-client* and *pool-server*. These are the working directories of the executables client and server respectively. As soon as the client gets connected to the server and authentication is successful (*if enabled*), files named *note.txt* and *cache.db* are created. **Do Not Modify** the *cache.db* file. Changes to the file - *note.txt* are synced across all clients.

2.4 Internals

As soon as the file - *note.txt* is modified, the client raises a SIGUSR1 signal which triggers a push to the server. If you don not have the notification modules (*inotify* or *kqueue*), you can manually send a SIGUSR1 to trigger a push to the server.

The child of the server which receives the push, sends a SIGUSR1 signal back to the parent after taking proper actions. Then, the server sends a SIGUSR2 signal to all members in the process group to notify for a push event. Then all children of the server push a patch to their respective clients to update changes.

2.5 Logging

The name of the log file is of the pattern *<executable>.<pid>.log*. Separate logs are generated for the server and the child, along with the necessary time-stamps.

3 Testings

3.1 Operating Systems

The code is checked to work on Mac OS 10.7 (*recommended*) and on Ubuntu 11.10. The source code is *conditionally compiled* to suite the OS needs. The Mac OS version of the executable uses *kernel queues* to detect changes to the file. This is more efficient than the *inotify* sub-system used in the Linux version of the executable.

3.2 Editors

The following text editors have been used for testing the functionality of the application. Other editors which do not change the inode of the file (*note.txt*) would also work.

- vi — *command-line*
- TextEdit — *GUI, Mac OS*
- gEdit — *GUI, Ubuntu*

4 Others

4.1 Important Assumptions

1. Two connections are established between the server and the client. We assume that no other connection request comes when the second connection (*control connection*) between the client and the server is made.
2. No two clients simultaneously update the *note.txt* file. Anyone client can change it, at a given instant of time.

4.2 Source Code Organization

include All header files are put in this directory. All header files except *server.h* and *client.h* are put in the header *all.h*. So including *all.h*, includes all files in the *lib* directory.

lib All support files are (our own library - *libsupport*) placed in here, which are required by both the client and the server executable images.