CSC209 Summer 2014

Software Tools and Systems Programming

Assignment 4 (10%): Sockets and Select

Due Date: 11:59pm on Friday, Aug 1, 2014

Objective

In this assignment you will develop a client/server service that allows a user to save files on their server, and automatically synchronize the files between the server and a number of devices including computers, phones, and tablets. The user installs client software on their computer, which takes care of automatically synchronizing files between the client and the server.

Important Notes:

- You must submit your solution by the due date on MarkUs.
- All programs are to be written using *C* and compiled by *gcc*.
- Your programs should be tested on *CDF* before being submitted. Failure to test your programs on *CDF* will result in a mark of 0 being assigned.
- **Code that does not compile will get 0 marks!**
- To get full marks, your code must be well-documented.
- If your work as a **team**, include member information (student ID, full name, login) inside **all** C programs as comments.

What To Submit

When you have completed the assignment, move or copy in a directory named **lightdropbox** all source code files, header files, and a **Makefile** that will build two programs: `dbclient` and `dbserver`.

Submit these files into MarkUs. **Make sure to submit the correct files, and that the files are the latest version. Submitting the wrong file is no excuse!**

Light Dropbox

Dropbox (https://www.dropbox.com) is a service that allows a user to save files on their server, and automatically synchronize the files between the server and a number of devices including computers, phones, and tablets. The user installs client software on their computer, which takes care of automatically synchronizing files between the client and the server.

Your task in this assignment is to implement a light version of Dropbox, using the `file`, `socket` and `select` system calls you have been learning in class. While you might find it interesting to read more about Dropbox itself, the specifications in this document define your task for the assignment.

dbclient

When the `dbclient` is run, it will first establish a socket connection to the `dbserver`, and will send a message containing the userid, and the name of the directory that the client wants to synchronize. We will restrict synchronization to files in one directory and will not handle subdirectories. Every *N* seconds, the client will initiate a synchronization operation. A synchronization operation uses the following algorithm.

For each file in the local directory

- get the last modified time and size
- send a `sync_message` request to the server containing the filename, last modified time, and size.
- read a `sync_message` response from the server containing filename, the server's last modified time for the file, and the size of the file on the server.
- if the last modified time in the message from the server is more recent that the local last modified time
    - the client will read the file from the server one CHUNKSIZE chunk at a time and will replace the local file with the contents from the server.
- otherwise, the client will send the local file one CHUNKSIZE chunk at a time to the server.

After iterating over all files in the local directory, the client checks to see if the server has any new files to send by sending an "empty" `sync_message` (a message with an empty string for the file name, a last modified time of 0, and a size of 0). It reads the response message from the server. If the server's response message is also an empty message, then the client knows there are no new files.

If the server's response message has a non-empty file name, or a non-zero last modified time, then the client will read the file from the server as above. After it has completed reading the file, it will repeat the process of sending a new file request (by sending an empty `sync_message`) and reading new files from the server until it receives an empty message from the server.

dbserver

The server is a little more complicated. It will create a directory for each user under the directory name provided by the user. If two users use the same directory name, we will assume they are sharing the directory. Because the server is handling multiple connections, we don't want the server to block on a read or accept from any client. To solve this problem, we will use `select` to multiplex between clients. This also means that we will need to keep track of the type of message that the server expects to receive from a client depending on the state of the client. When the server is notified (via select), that a message is available to read from the client, there are several possible options. The server may be expecting:

- a `login_message`, if it hasn't read anything from the client yet.
- a CHUNKSIZE chunk of a file, if the server has realized that a file is newer on the client than on the server.
- a `sync_message`, otherwise.

The server will store the following information for each client:

- The user id of the client
- The socket file descriptor
- The directory name where the client's files are stored
- A list of the files that the client has synchronized with the server. Information about each file includes:
    - The file name
    - The last modified time sent by the client.

A partial algorithm that describes the communication between the server and a single client follows. Note that the server only acts in response to a message from the client. Aside from the initial setup, it doesn't initiate any operations.

```
handle a read from a client
    case LOGIN:
        read login message and store userid and dir in client array
        create dir if doesn't exist
        set state = SYNC
    case SYNC:
        read sync_message
        if sync_message is an empty message
           figure out how to identify a file on the server that is
           not on the client, and add it to the client list
        else
            find filename in client's files array
```

```
        write response (if there are no new files to send, then send
         an empty response)
        if file's mtime is newer in clients' files array on server
         than in sync_message
            write file to client
            set state = SYNC
        else
            set state GETFILE
            (keep track of which file we are expecting to read)
    case GETFILE:
        read chunk
        write chunk
        if finished reading file (keep track of size vs bytes read)
            set state SYNC
```

Part of the algorithm is missing. In particular, the server needs to handle an empty `sync_message` from the client, which is an indication that the client is checking if the server has a file that the client does not. You will need to figure out a strategy to identify the files that the server is storing in the appropriate directory that are not in the client's list of files, so that they can be added to the client's list.

Tips:

- Take advantage of the example code shown in class and given on the course web site. The example code will help you set up the sockets and the select call.
- Write the code incrementally! Make sure each step compiles and runs before moving on. It will save you time! The first step might be to write a client that just sends the login message, and the server that receives only the login message stores the client info. An intermediate step might be one where no file data is transferred, but the `sync_messages` are sent. Write and test a function that fills in a `sync_message` for a file.
- If you are anxious to get started on the assignment immediately, you might want to focus first on being able to iterate over files in a directory, retrieving their size and last modified times.
- Note that lots of shortcuts with the starter code have been taken. In particular, the number of clients and the number of files are artificially limited. One would not write code this way, but it simplifies this part of the assignment.
- You will be required to use the structs found in `message.h` for communicating between the client and the server.
- Also note that file deletion on the server is not allowed. Once a file has been synchronized to the server, it stays there.
- Be careful to handle the case where a client connection is closed.