CS 214: Systems Programming
Assignment 3: Network File Server

By Christopher Gong and Eric Giovannini

Base Program

We made a program that allows users to connect to a remote file system and modify files on the server. The clients would be able to modify the server using functions such as “netopen”, “netread”, “netwrite”, and “netclose”. In cases of failure, the proper errno value will be set on the client side as well as a descriptive message being printed out on both the client and server side. As well as handling files, multithreading will be utilized to give the program concurrency, allowing clients to change files one at a time and to change different files independently of other files/clients. This requires the implementation of different mutexes.

Message Parsing

In order to affectively pass messages between the client and server, we separated parts of the messages by delimiters. In our case, we chose a “(“ as our delimiter. The first parameter of the message will always be the number of bytes of the rest of the message. The reason behind this is so that we can read a fixed number of bytes, because when receiving messages from the server/client there’s no way to know how many bytes is the message initially. But then, after reading the rest of the message, we can separate it into the parameters by looking for other instances of the delimiter.

```
( 1 0 ( o ( t e s t . t x t
        ^               ^
        |---------------|
            10 bytes
```

Note that the last parameter, we have chosen to omit the closing delimiter because the last parameter has the possibility of containing a delimiter as part of its actual content.

Netserverinit

Before performing any of the other net functions, we have to make sure the server hostname is valid. Also, since the port will be hardcoded, the global port variable will be set here as well as the global hostname variable.

Netopen

This function asks the client for a path to a file located on the server as well as a read-write permission. Then, the client is returned a network file descriptor to the file or -1 in the case of an error. The purpose of a network file descriptor is to give each client a unique identifier by which to address a file on the server. Otherwise, each client would receive the same local file descriptor.

Netread

This function asks the user for a network file descriptor, a buffer, as well as the number of bytes the user wants to read from the file. The function returns the number of bytes successfully read from the server, or -1 in the case of failure. Like the regular read function, even if the user enters in more bytes than contained in the file, netread will eventually reach the end of the file (EOF) and prematurely end before reading garbage value from the file. The bytes read are then copied into the buffer inputted by the client.

Netwrite

This function asks the user for a network file descriptor, a buffer, as well as the number of bytes the user wants to write to the file. The function returns the number of bytes successfully written to the server, or -1 in the case of failure. Like the regular write function, even if the user enters in more bytes than contained in the buffer, netwrite will only write up to the end of the buffer and prematurely end before reading garbage value from the buffer. The bytes from the buffer are copied into the file associated with the network file descriptor.

Netclose

If the user no longer needs to make changes to a file or feels the need to change permissions, the client can close the network file descriptor, making it unusable. Returns 0 on success, -1 on failure.

Multithreading

In order to handle multiple clients, threads, specifically POSIX threads, have to be utilized. A thread takes in a function pointer than runs with inputted arguments. As of now, a thread is created per each operation aka for each message sent from the client to the server. Every time a thread is created, it handles the incoming client message by decoding it, determining which function was called by the client, and handling the output appropriately.

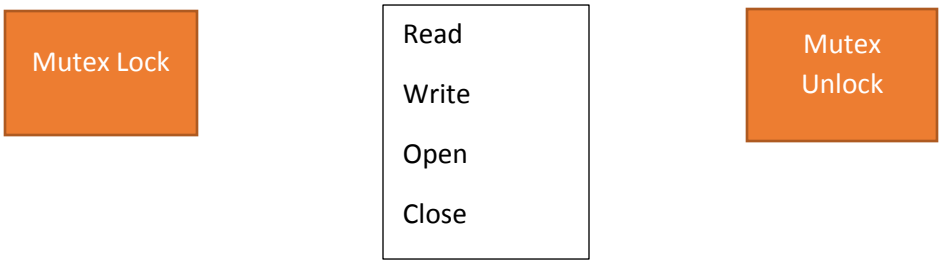
Mutexes

Since this is a networking assignment, we have to make sure there is concurrency and parallelism among threads, no deadlocks. If one client is currently reading to file “chris.txt”, but another client wants to write to file “chris.txt”, the other client must wait until the first client finishes reading. In order to do this, we need to maintain a linked list of structs containing network file descriptors and their associated file pathes. Note that this linked list is more relevant to gathering local file descriptors from network files descriptors. This linked list uses a global mutex so that changes to this linked list can only happen one at a time from clients. Then, for file operation mutexes, we maintain another linked list of structs containing mutexes and file paths. Similar to the other linked list, this linked list also uses a global mutex to ensure changes happen in parallel fashion. When performing read and write operations on files, their associated mutex is searched in the linked list.

Mutex lock while searching through linked list of file operations mutexes

Mutex unlock once done searching

Mutex lock using newly found mutex from linked list during read/write/open/close operation



Mutex unlock once read/write/open/close operation finishes

Extension A – Different File Modes

Here's the decision tree used to determine whether or not a file can be opened in a certain file mode depending on whether or not other clients have opened the same file in specific file modes,