

Matthew Mann  
Samrat Darisipudi  
Systems Programming  
4/30/2018

## **Design**

The client side program creates the appropriate message and sends them to and parses messages sent from the server. The server parses through the message sent from the client and checks on whether to open, read, write, or close and takes permissions into account. It also implements the monitor thread.

### ***libnetfiles.c***

#### ***netserverinit()***

This method takes in a hostname and verifies if the host exists. First it checks if the hostname is valid, and if it isn't it, it sets the appropriate errno value. If it is valid, it creates a new socket, sets mysocket1 to the correct value and it connects.

#### ***netopen()***

It takes in the pathname of the file and the flags needed to open it. First the method uses mysocket1 to check if there's a valid connection to the host. Once it checked if the connection and the flags were valid, it would create and send the message to the host with the mode, address and flag. It then receives the message and prints out the length and size of the message it received. It interprets the message and it returns the new file descriptor.

#### ***netread()***

It takes in the file descriptor, a buffer pointer, and number of bytes. It creates and sends a message to the server dictating that it wants to read a certain number of bytes from the file destination. It then receives the bytes that it asked for, and this is where we attempted to implement Extension B. If the number of bytes it is asked to read is over 4000, it creates a socket for every 4000 bytes, receives the chunks of data and recombines it, interprets it, and stores it into the buffer. It then returns the number of bytes read.

#### ***netwrite()***

It takes in the file descriptor, a buffer pointer, and number of bytes. It creates and sends a message to the server dictating that it wants to write a certain number of bytes from the buffer to the file destination. It interprets the file, and it then returns the number of bytes that were completely written to the file.

#### ***netclose()***

It takes in the file descriptor. It creates a message for the server dictating that it wants to close the file descriptor. It then makes the connection to the server and sends the message to the server. It returns 0 on success.

*main()*

This is where the `netopen()`, `netread()`, `netwrite()`, and `netclose()` functions are run.

***netfiles.c***

*sigHandler()*

It takes in an int, a `siginfo_t` and a void pointer. The purpose of this function is for us to try and implement D, where it is to check every 3 seconds for connections that have been waiting for more than 2 seconds and invalidates them. It does the second part by iterating through a queue and checks if any connection has been waiting for more than 2 seconds.

*enqueue()*

It takes in an integer and a queue. It creates a node with that value and adds it to the queue.

*dequeue()*

It takes in a queue. It pops off a node off of the queue.

*threadFunc()*

It takes in an argument. It starts off with our attempt to prepare for extension D, by setting up the first by having it set up `sigHandler()` and using a timer to have it run every 3 seconds. It then takes in the message sent in by the client, and reads it in and makes sure it is the proper format. It first checks if it is a close command, in which case it closes the file descriptor, and sends a message to the client. It then removes the client from the linked list of open files. Then it checks if it is not an open command, in which case it assumes it is a read or write. It determines the size of the buffer and is also an attempt at implementing extension B again, and makes sure it splits reads or writes over 4000 bytes. It sends the appropriate response to the client and connects to different sockets and creates various threads to split it up. If it is read, it reads the appropriate number of bytes and sends it to the client. If it is write, it writes the appropriate bytes and sends it to the client. If it is open, then it implements extension A, and if it's been initialized, checks that the client has the appropriate permissions. If it hasn't been initialized, it creates a new node and adds it to the queue. It sends the appropriate message to the client and returns null.

*main()*

It sets up the server and listens for new connections, creating threads when necessary.

## **Tests**

Each set of files has their own purpose. `libnetfiles.c` and `netfiles.c` test `netwrite`. `libnetfiles1.c` and `netfiles1.c` test `netopen`. `libnetfiles2.c` and `netfiles2.c` test `netread`. `libnetfiles3.c` and `netfiles3.c` test `netopen` and checks if the correct error messages are passed. `libnetfiles4.c` and `netfiles4.c` attempt to implement and test extension C.