Joseph Klaszky
Mohammad Memon

Asst 3

Preamble:

I only did the base program. I had hopes of at least doing extension A, but sadly a second round of late midterms killed that dream. Also, there were these odd “double freeing” issue that I simply couldn’t track down, so sadly I’m sure there are some memory leaks somewhere because I had a remove several free statements to make sure the server wouldn’t blow up.

Overview of how our code work:

- netfileserver.c:
The main() function makes a connection binds to port 42942 and simply listens for any incoming connections. When it gets a connection it creates a thread and passes the socket FD. That’s kind of it for main.

The threadMain() takes a newsockFD in the form of an int pointer. It then reads from the socket, parses some info and check which command it needs to run.

Server commands

- nopen(linked list, socketFD): attempts to open a local path, constructs a reply message with an errno code and the FD or -1. Sends message over socket.
- nclose(linked list, socketFD): attempts to close a local file, constructs a reply with the results and send message over socket.
- nread(linked list, socketFD): reads from a local file and constructs a return message with errno, the ammount of bytes read, and a buffer with all the of read bytes. Locks mutex while reading.
- nwrite(buffer, socketFD): parses the buffer differently than all the other functions. Makes a buffer with the size provided and attempts to write that buffer to a given local FD. Locks mutex while writing.

-libnetfiles.c/.h
Important functions:

- networkserverinit(hostname): gets the Ipaddres of the host with gethostbyname(hostname) and sets it as a global variable because every other function will use it. If its cool returns 0 or sets h_errno and returns -1.
- netopen(path, mode): does a quick error check on the mode, it its not one that I accept it just stops the code right there. If all is good writes the path and mode to a socket. Reads the return socket, parses the return message, sets errno and returns the FD or -1.
- netclose(fd): this one is pretty basic, sends the fd over a socket and gets a reply with an error code and an int representing success or failure
- netread(fd, buffer, numberOfBytes): this one probably took me the longest to do. It sends a message over a socket with a fd and a number of bytes. The server uses those to read them many bytes from a file server side. When netread gets a return message it parses the info, but often not all of the data is send in one read from the socket, so it need to keep pulling out data

until the stream ends. All of the data from this stream is stored in the buffer that was passed as a argument. It then finally returns the total number of bytes read from the server side.

- netwrite(fd, buffer, numberOfBytes): Very similar to netread(), but send over a buffer of data to the server instead of getting one back. Gets a return message with an errno and the number of bytes written.

Note: I made a function, errNoChk(errno), to output the errors to stderr when they happened. I'm not sure why I did this, there were bigger fish to fry, but I thought it would help me debug. Sorry if this gets annoying while you're trying to test the code.

Overall this project was awesome, but I just wish I had more time to work on it. I feel like the code isn't up to my normal standard, but when you're in a time crunch what can you do?

Testing: I tried everything I could think of to break it, but I don't know much about networking so I was a bit limited. I tried opening files that didn't exists, and open a file that did many, many times. I closes files that were never opened and closing files multiples times. I read more data than was in a given file. I read huge amounts of data at one time (the entire .txt book of the wizard of oz) and wrote huge amounts of text. I also wrote very time pieces of data to see if that would trip things up. I tried to open server different clients at once and do all of the above at the same time. Overall the code help up well against the things I could think of. I'm sure there was something that I missed though.