README Assignment 3

Azizur Rahman, Renard Tumboken

December 13, 2016

Note: Only did “Base Program” ONLY (no extension)

We have a makefile that compiles and makes .out files: libnetfiles.out, netfileserver.out

You can use our code by including the following lines of code:

For Client:

#include “libnetfiles.h”

netserverinit(char* hostname)

netopen(const char* pathname, int flags)

netread(int fildes, void* buf, size_t nbyte)

netwrite(int fildes, const void* buf, size_t nbyte)

netclose(int fd)

Before Compiling:

netfileserver.out (server) needs to be executed before running libnetfiles.out (client). You need to call netserverinit() in libnetfiles.c before doing any other net functions. You can change the port number of both client and server in the constant PORTNO variable (which needs to be equivalent) which we defaulted to 35972 in libnetfiles.h and netfileserver.c. The server and client code also has a max buffer (which needs to be equivalent) under the constant BUFFERSIZE which we defaulted to 256 in libnetfiles.h and netfileserver.c.

Note: When compiling netfilesserver.c, there will be a warning "pointer targets in passing arguement 3 of 'accept' differ in signedness." Ignore this as everything works as intended.

Sample Test Case:

Executing main() in libnetfiles.c will give you a SAMPLE series of all the net connections to the server. The client will prompt for a hostname where the server is hosted at and a testfile (test.c is provided) that MUST exist. Also, the port number 35972 for both client and server should not be currently occupied. It will read a given hardcoded number of characters and write "[WRITE HERE]" to the file.

Implementation:

We used a 1 to 1 to 1 mapping between socket, connection, and thread. Between the server and client, we use a struct called “structToSend” to pass information back and forth. This struct contains the file descriptor, string buffer, bytes to read, and file operation. Parts of the struct may be sent as blank/0/NULL if the information is unrequired for the file operation.

libnetfiles.c:

netserverinit(): Checks if given hostname and port are valid. If valid, connect to server.

netopen(): Open file with given flag. Return file descriptor used for other net operations. Set file cursor to start of file.

netread(): Read file via file descriptor. Read nybtes in file and save what was read to buf. It moves the file curser forward to how many nbytes read.

netwrite(): Write to file via file descriptor. Write nbytes of buf in file. Note that the nbytes written should NOT include the null terminator byte or the file will turn into hexadecimal. If file cursor is not at the end of the file, it will overwrite the characters in the file.

netclose(): Close file via file descriptor and close the connection to the server.

Note: if any error happens to the file operation server side, the server will return the appropriate errno, which libnetfiles.c will set and print client side.

netserver.c:

main(): Waits for a client to connect. If client connects, make a new socket for a new thread to handle client file operations. While the thread happens, main() continues to wait for other clients to connect.

clientFileHandler(): Connects to client via new socket. Reads the client and execute its given file operation. Close the socket when client tells it to close.