ReadMe
Assignment 3
Mark Hirons and Kevin Daini

Base + Extension A

Overview

The contained files consist of a simple file server and an interface for a client program to connect and transfer files from a remote host. The five functions usable by the client are net-serverinit, netopen, netclose, netread, and netwrite. For full documentation on return values see the assignment description.

Libnetfiles.c

These functions are used by the clients to connect to and use files on the server.

Netserverinit connects the client to the server using a TCP socket and sets that socket as a global socket for use by all the net functions. This function also sets the filemode on the server through a parameter sent by the client. Modes are client specific.

Netopen, sends a file path to the server and opens the file on the server, along with he necessary flags for read/write permissions. The server returns the file descriptor for use by the client.

Netclose, takes a file descriptor from the client and closes the specified file on the server.

Netread, takes the file descriptor, a buffer, and an integer as parameters, for the necessary file, the destination of the bytes, and the number of bytes to be read from the server, respectively. The server sends back the requested bytes from the file and this data is written into the buffer.

Netwrite, takes a file descriptor from the client, a buffer to read from, and a number of bytes to write. The client sends this data to the server which then writes the bytes to the specified file.

Netfileserver.c

The server handles client requests and spawns a thread for each client. The files that are currently open on the server are stored in a global linked list called openFiles. When a file is opened, the client searches through the list to check if its already there. If it is, and the conditions for its mode are met, it updates the meta data held inside the node and opens the file. If it is not found, it creates the node and adds it to the list, setting the data accordingly.
Each client also has a list, called a clientList, that stores their currently open files. It is used to make sure the data for each file is maintained correctly when removing files. The file name and mode must be known, as well as the flags used when opening. Those are stored in this list, and they are created on a client to client basis. A client can only see their own list.

Main, calls waitforclients.

WaitForClients, uses a loop to continually listen for clients through a socket. When a connect is made, it spawns a thread that calls handleClient.

HandleClient, interprets the message sent by the client and calls the necessary server function which will handle the system calls. This is also where the clients list of currently open files is initialized and saved until they disconnect from server.

ServerOpen, receives the message from netOpen, along with the client's list and mode. as parameters. It then searches through the openFiles list and checks the flags and mode of the client. If the conditions are met or the file is not currently open by anyone, then the file is opened. Both the clientList and openFiles are updated.

ServerClose, receives the same parameters as open, but the message contains only the fd. The pathname and flags are stored in the clientList. It finds the file in openFiles and the client's clientList, and updates the data according to the conditions when the file was opened. If integer totalUsers is equal to 0 in the node, it is deleted from openFiles. If there are still users with the file open, then the node remains in the list with the updated data. The node corresponding to the file in client's clientList is also deleted.

ServerWrite extracts arguments and data sent by netwrite and calls write() with those arguments. Then it constructs a message with the results of open() back to the client.

ServerRead extracts arguments and data sent by netwrite and calls read() with those arguments. Then it constructs a message with the results of read() back to the client.

Messaging Protocol

Client Messages

The basic structure of client messages is a 4 byte prefix for the message length, a 1 byte function code so that the server can identify the system function to be called, followed by the arguments needed for that function. In the case of netopen, a 4 byte prefix is also needed before the pathname.

Server Message

The server does not use any prefixes in its messages. The first 4 bytes are the return value of the system call, then any data that the function might need to be sent to the client. The last 4 bytes are the errno set by the system call, or 0 if errno is not set.