Assignment 3: Remote Files Base + Ext A

By Derek Schatel

Libnetfiles.c and Netfileserver.c

Netfileserver must be started up before any client connections are made. On the netfileserver side, the program will create a new socket, bind it to a port and listen for remote connections. It will then enter a while loop where it will continually accept connections and spawn new threads to handle those connections.

On the libnetfiles side, the client must first initialize the connection with netserverinit. This function takes a hostname and a global mode. These modes are as follows: 0 for unrestricted, 1 for exclusive, and 2 for transactional. Netserverinit will attempt to use getaddrinfo to resolve the hostname, and if it finds it it will store it in a global variable for later use. If it cannot find the host, it will set h_errno to HOST_NOT_FOUND. Furthermore, if the mode type is invalid, it will set errno to INVALID_FILE_MODE, which is defined in libnetfiles.h. If the mode is valid, it will store the mode integer in a global variable that will later be used for all netopen requests.

After initializing the server with netserverinit, the client may use the other library functions – netopen, netread, netwrite and netclose. In each function, the client will create a socket and attempt to connect to the remote server given the stored address info. It will then take the inputs and use snprintf to store them in a formatted buffer with semicolons as delimiters. Each buffer consists of the following format: opcode;arg1;arg2;...argn. For netopen, opcode is 1, up to 4 for netclose. If the socket connection is successful, the function will then write the buffer to the server. On the server side, the spawned thread will use strtok to tokenize the input and copy the various segments of data into separate variables. It will run a switch statement on the opcode and run a server-side function depending on whether netopen, netread, netwrite or netclose was called client-side.

For open on the server side, the formatted buffer will include a pathname, file mode and permission flag. As part of Extension A, the server open function will parse a linked list of structs that are indexed by a unique network file descriptor. This struct consists of the following information: a network FD, corresponding real file FD, full pathname, opened mode, and permission flag. Upon parsing the linked list, if the server open function finds a conflict between an existing file entry and the requested open flags, it will disallow opening of the file and report an error code when it writes back to the client. If there is no conflict, the server open function will try to open the file. If an error is reported, the server will store the errno code in a formatted buffer for sending back to the client. Otherwise, if all is successful, the server open function will generate a new struct, insert it into the list, and pass a unique network FD back to the client.

Netwrite and netread operate similarly. The user passes the unique network FD to the server through a formatted buffer, along with other data – for netread, a buffer and the number of bytes to be read, and for netwrite a buffer to write and a number of bytes to write. The formatted buffer will pass to the server via write, which will read it, tokenize it, and perform the necessary operations. If anything goes wrong, an error code will be reported instead. To find the real file descriptor, the server-side functions parse the linked list and find the entry with the associated network FD. In both cases, the client-side functions return the number of bytes read or written, respectively, or it reports an error.

Netclose locates the server-side real FD based on the passed network FD, closes the file descriptor, then removes the struct node from the linked list. Client side, netclose returns 0 if successful.

All sections of the server-side code that involve traversing or modifying the linked list of structs, as well as any sections dealing with reading and writing to files have been mutex locked in order to provide thread safety.

A client.c file has been provided with a driver program in order to assist with testing functionality.