Zachary Blanco
Mike Botti
CS214 - Systems Programming

Assignment 3: Fileserver + client

Extensions

We attempted to implement the Extensions A, B, C, and D on top of the base program.

The libnetfiles.h library provides functions to open files on a remote server. The functions used to perform such actions are *netserverinit(), netopen(), netread(), netwrite(), and netclose()*. Typically they will be used in that same order.

The actions will only be able to open or read files which reside on the fileserver and are able to be accessed by the process which is running the fileserver.

The netfileserver can be run using the Makefile calling "make server"

We tested many different cases for reading, writing, opening, and closing files - all of which can be found in **test_files.c** - This is a file full of unit tests for our functions which assert different conditions for certain operations in order to ensure that our code is working properly.

Implementation Details

The behavior of our client is such that we open a new socket connection for every operation on the server. While this may cause overhead it helps to ease network congestion. Usually transfers over the network are not made to be small anyways. Typically they'll be at least a couple hundred bytes which can offset the amount of overhead is takes to open and close many different sockets.

The extension A is implemented via a check in the netopen method on the serverside. This will check out global state of the opened files and check for any files which are still open with specific permissions and will send an error if it finds conflict in permissions issues.

The way we keep track of the global state is by a linked list which holds the data for all opened files. This is used on all operations to make sure they are allowable - else an error is sent back to the client.

Threading for the multiplexing is done via an extra exchange between the client and server where the client will first calculate necessary metadata such as the number of ports. From there is is sent to the client where the client follows a strict protocol for sending or receiving the multiplexed data. In order to do this we need to know how much data we're reading and sending. We also need to keep track of the global state of the multiplex threads and ensure that there are now more than 10 threads are open to multiplex at once. In order to accomplish this we use global state variables with mutexes in order to lock the structures when we want to modify the global state of the available metadata for the multiplexed threads.

useful Makefile targets:

- make test
  - compile and run the server
- make debug_client
  - compile our test cases and libnet and run all unit tests in gdb.
- make clean
  - clear all compiled files
- make libnet
  - compile all files

The following files are included
libnetfiles.h
libnetfiles.c
netfileserver.h
netfileserver.c
Makefile
test_files.c

test_files.c is a file containing unit tests for many different functions of the the library.

And example piece of code which tests multiple reading and writing to the server:

```
netserverinit("localhost", NFS_UN);
  int fd1 = netopen("./test.txt", O_RDWR);
  int fd2 = netopen("./test.txt", O_WRONLY);
  int fd3 = netopen("./test.txt", O_RDONLY);

  assert(fd1 != -1, "Descriptor 1 should be good");
  assert(fd2 != -1, "Descriptor 2 should be good");
  assert(fd3 != -1, "Descriptor 3 should be good");
  assert(netwrite(fd1, "test1", 5) == 5, "Should be able to write 5 successful bytes");
  assert(netread(fd3, &buf, 5) == 5, "Should read 5 bytes");
  assert(netwrite(fd2, "test2", 5) == 5, "Should write 5 successful bytes with fd2");
  int nr = netread(fd1, buf, 2);
  printf("NR: %i", nr);
  assert( nr == 2, "read 2 successful bytes");

  netclose(fd2);
  netclose(fd3);
  netclose(fd1);
```

where things.txt is a file that exists in the same directory as the server.