Matthew Robinson
Faiq Raza

The server handles requests by forking every time a new connection comes in. This allows an easy mapping of connections to processes, which makes programming very easy. We don't have to lock files or think about what would happen when multiple connections access the same file because this is all handled at the OS layer. The client sends encoded messages following the following protocol:

```
All messages of the form
<size><message-id><payload>

size is 4 bytes ,big endian
message-id is 1 byte
payload varies in size

open:
size = 1 + length of filename
id = 0
payload = string of filename

open-response:
size = 5
id = 0
payload = 4 byte int (file descriptor)

read:
size = 5
id = 1
payload = 4 byte int (file descriptor)

read-response:
size = 1 + size of file
id = 1
payload = the file as raw bytes

write:
size = 5 + size of file
id = 2
payload = <file descriptor as 4 byte int><the file>

write-response:
size = 5
id = 2
payload = number of bytes written

stat:
size = 5
id = 3
payload = 4 byte int fd

stat-response:
size = 17
id = 3
payload = <size><creation><access><modification> all as 4 byte ints

close:
size = 5
id = 4
```

```
payload = file descriptor

close-response:
size = 5
id = 4
payload = 4 byte int (return value of close)

close-connection:
size = 1
id = 5
payload = nothing
```

Integers are sent as 4 bytes in big endian order. These bytes are then deserialized by the client and server. This is done through a byte_buffer struct and family of functions. They allow bytes, ints, and strings to be easily inserted into a character array by keeping track of an offset.

When the client has closed all of it's files, it sends a close-connection message and closes the socket. The server responds by closing its socket and then killing the associated process.

As for performance, we created a program that performed some basic read/write operations forever in a loop. We then ran multiple processes simultaneously. We tried up to 100, and did not run into any problems. This looks like a matter of the number of processes the operating system can handle and is very machine dependent. The only time problems occurred was when several processes attempted to open a socket at once. It is unknown if this is a bottleneck of the system or a design flaw on our part. Even with 100+ processes running, the files did not become mangled in any way.

Matthew Robinson did everything in the server and serialize directories. Faiq Raza did everything in the client directory. However, there was much cross-pollination in the work done, so this is only a rough estimate. Each contributed to the server and client to some degree, and both of us did bug-testing.