CSE/EE 461: Programming Assignment 1, 461turnin

Neil Spring

Autumn 2000

In this assignment, you will write a program to turnin your future programming assignments. The intent of this assignment is to give you experience writing programs that use the network. This experience should include two pieces. First, the sequence of system calls used to intiate and teardown a network connection. Second, the use of messages in the form of structures in network byte order.

If you're unfamiliar with programming on a unix machine, this will be an exercise in using the on-line man pages and compiler.

The program you will write will do the following:

1. connect to the 461turnin server running on `poplar.cs.washington.edu`, using TCP at port 4610
2. send a message containing your names, CSE usernames, student ID numbers, which project and what file is being turned in.
3. wait for an acknowledgement stating that the server could parse your message
4. send the number of bytes in the file, followed by the file itself.
5. wait for an acknowledgment containing the number of test cases passed, which will represent your score on the assignment. (this will be more important in later assignments)

1 system calls

Use the `man` command to get additional information on each of the following calls, including function parameters and descriptions of the structures they use and return. You may need to use the section number, eg. `man 2 bind`, to get the correct entry.

The sequence of function calls should be roughly like:

1. socket

2. gethostbyname
3. connect
4. write
5. read
6. write
7. read
8. close

You may need to deviate from this, perhaps calling read or write a number of times before a message is complete.

You'll also need to use htonl to convert integers from host byte order (little endian on Intel machines) to network byte order (big endian) before sending those integers across the network. Character strings are already in the correct order.

2 message format

The initial request message:

```
int length;
int projectnumber;
int studentid1;
int studentid2;
char name1[30];
char name2[30];
char username1[10];
char username2[10];
char filename[25];
```

The initial response message:

```
int length;
int error;
```

If `error` is not zero, it represents the field of the request message that is incorrect. For example, if `error` is 1, the projectnumber field is wrong, or perhaps not in network order. If error is not zero, the connection is then closed by the server.

The upload message:

```
int length;
char content[];
```

The response message:

```
int length;
int passed;
int percentage;
```

`length` is the length of the message, since I may need to add additional fields. `passed` is the number of tests your program has successfully passed, and `percentage` is the percentage of tests that were successful, minus 20% per day the assignment is late. (although not yet implemented)

After the response message is sent, the connection is closed.

My implementation of the client is about 100 lines long.