CSE 333

Lecture 21 -- server sockets

Steve Gribble

Department of Computer Science & Engineering

University of Washington

Administrivia

HW4 out either Friday or Monday

- planning on letting you write more of the code

Bonus questions in HW2, HW3, HW4

- your grade will be completely unaffected if you don't do any of the bonus questions

HW3 q1

How long did it take you?

- 10-12: 1
- 12-14: 1
- 14-16: 3
- 16-20: 3
- 20-25: 7
- 25-35: 4
- 35+: 3

HW3 q2

Confidence in your work

- low [tons ‘o bugs]: 1
- medium [a few bugs]: 6
- high [code pretty much correct]: 17
- supreme [> Gribble’s]: 0

HW3 q3

Worthwhile

- True; even better than hw1/hw2: 8
- True; just as good as hw1/hw2: 14
- True; < hw1/hw2: 1
- False: 1

HW3 q4

We provided [...] code:

- Too little: 1
- Just right: 18
- Too much: 4
- Waaay too much, next time don't provide any: 1

HW3 q5

I spent my time:

- on stupid C++ compiler errors: 1
- on memory-related bugs/errors: 3
- bugs in my on-file index format: 17
- learning STL: 0
- other: 3

HW3 q6

I like that you give us optional parts:

- True: 17
- False: 7

Q7

Other feedback?

- make the bonus due **after** the assignment [agreed]
  - I like the range of difficulty
- thanks for providing libhw1 and libhw2
  - but fix your unit tests for HW1 and HW2! [agreed!!!]
- liked the free format for filesearchshell
- I'd like to learn how to write C++ unit tests
- wish our test suite worked, but great project for learning C++

Q7

Other feedback?

- found that the homeworks are equivalent to giving an artist color-by-numbers. Very little thinking.
  - great potential, but please leave more to the student.
- filling out the table readers was repetitive, but it was a nice balance to the free-form QueryProcessor
- part D was *much* easier in C++ than in C [yay!]
- consider giving us better tools for debugging index file errors
- for next assignment, give us *less* (specs, few statements)?

Q8:

Favorite game

Numeric value	*Answer*	*Frequency*	*Percentage*
1	Bubble bobble	3	12.50%
2	Contra	2	8.33%
3	Donkey Kong	1	4.17%
4	Dr. Mario	0	0.00%
5	Golf	0	0.00%
6	The Legend of Zelda	5	20.83%
7	Lemmings	0	0.00%
8	Lifeforce	1	4.17%
9	Mario Bros.	1	4.17%
10	Mega Man	1	4.17%
11	Pac-Man	1	4.17%
12	Super Mario Bros.	3	12.50%
13	Tennis	0	0.00%
14	Tetris	4	16.67%
15	Tetris 2	0	0.00%
16	Zelda II -- The Adventure of Link	2	8.33%

Today

Network programming

- server-side programming

Servers

Pretty similar to clients, but with additional steps

- there are seven steps:
  1. figure out the address and port on which to listen
  2. create a socket
  3. **bind** the socket to the address and port on which to listen
  4. indicate that the socket is a **listen**ing socket
  5. **accept** a connection from a client
  6. **read** and **write** to that connection
  7. **close** the connection

Accepting a connection from a client

Step 1. Figure out the address and port on which to listen.

Step 2. Create a socket.

Step 3. **Bind** the socket to the address and port on which to listen.

Step 4. Indicate that the socket is a **listening** socket.

Servers

Servers can have multiple IP addresses

- "multihomed"
- usually have at least one externally visible IP address, as well as a local-only address (127.0.0.1)

When you bind a socket for listening, you can:

- specify that it should listen on all addresses
  - by specifying the address "INADDR_ANY" -- 0.0.0.0
- specify that it should listen on a particular address

bind( )

The “bind( )” system call associates with a socket:

- an address family
  - AF_INET: IPv4
  - AF_INET6: IPv6
- a local IP address
  - the special IP address INADDR_ANY (“0.0.0.0”) means “all local IP addresses of this host”
- a local port number

listen( )

The “listen( )” system call tells the OS that the socket is a listening socket to which clients can connect

- you also tell the OS how many pending connections it should queue before it starts to refuse new connections
  - you pick up a pending connection with “accept( )”
- when listen returns, remote clients can start connecting to your listening socket
  - you need to “accept( )” those connections to start using them

Server socket, bind, listen

see server_bind_listen.cc

Accepting a connection from a client

Step 5. Accept a connection from a client.

Step 6. read( ) and write( ) to the client.

Step 7. close( ) the connection.

accept( )

The “accept( )” system call waits for an incoming connection, or pulls one off the pending queue

- it returns an active, ready-to-use socket file descriptor connected to a client
- it returns address information about the peer
  - use inet_ntop( ) to get the client’s printable IP address
  - use getnameinfo( ) to do a **reverse DNS lookup** on the client

Server accept, read/write, close

see server_accept_rw_close.cc

Something to note...

Our server code is not concurrent

- single thread of execution
- the thread blocks waiting for the next connection
- the thread blocks waiting for the next message from the connection

A crowd of clients is, by nature, concurrent

- while our server is handling the next client, all other clients are stuck waiting for it

Exercise 1

Write a program that:

- creates a listening socket, accepts connections from clients
  - reads a line of text from the client
  - parses the line of text as a DNS name
  - does a DNS lookup on the name
  - writes back to the client the list of IP addrsses associated with the DNS name
  - closes the connection to the client

Exercise 2

Write a program that:

- creates a listening socket, accepts connections from clients
  - reads a line of text from the client
  - parses the line of text as a DNS name
  - connects to that DNS name on port 80
  - writes a valid HTTP request for "/"
    - see next slide for what to write
  - reads the reply, returns the reply to the client

Exercise 2 continued

Here's a valid HTTP request to server www.foo.com

- note that lines end with '\r\n', not just '\n'

```
GET / HTTP/1.0\r\n
Host: www.foo.com\r\n
Connection: close\r\n
\r\n
```

See you on Friday!