

# CSE 333

## Lecture 18 -- server sockets

**Hal Perkins**

Department of Computer Science & Engineering

University of Washington



# Administrivia

Exercise covering client-side programming posted late yesterday, due Monday before class

Next exercise covers today's server-side code. Posted this morning in case people want to get started over the weekend, but not due until Wednesday before class.

HW4 posted now, files pushed after class, due last Thursday of the quarter (+ late days if you have them)

# 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 **listening** 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” or “in6addr\_any” -- 0.0.0.0 or :: (i.e., all 0’s)
- 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 (also handles IPv4 clients on POSIX systems)
- a local IP address
  - ▶ the special IP address **INADDR\_ANY** (“0.0.0.0”) means “all local IPv4 addresses of this host”
  - ▶ use **in6addr\_any** (instead of INADDR\_ANY) for IPv6
- 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

# Before we go...

*hw4 demo*

# 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 addresses 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 Monday!