CSE 333

Lecture 18 -- server sockets

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 **listen**ing 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 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