Computer Networks

Lisa Frye, Instructor
frye@kutztown.edu
Kutztown University

Sockets

- Stream
- Unrelated process communication
- *client-server* model
- Two types
  - SOCK_STREAM
  - SOCK_DGRAM

Socket Pair

- the 4-tuple that defines the two endpoints of the connection
- uniquely identifies every TCP connection on an internet

Socket Programming: UDP

- No connection
- Unreliable

- What if packet doesn't reach server?

- What if client requires a response?

Network Byte Order

- Little endian
  - Most significant bit on right
  - 2.1.0.192
- Big endian
  - Most significant bit on left
  - 192.0.1.2

Socket Programming - TCP

- Must establish connection
  - Client program
- Requires reliability – handled by TCP

Establish a TCP connection

- server prepared to accept incoming connection
  - *passive open*
- client issues an *active open*
  - causes client to send packet to server
- server acknowledges the client and sends back a packet

Terminate a TCP connection

- one application calls **close** first – this end performs the *active close*; this end sends a packet to other process
- second process receives packet and performs a *passive close*. This packet is passed to application as an EOF
- process that received the EOF will eventually **close** its socket; packet is sent to other process

Socket Libraries

- Most languages require the use of additional libraries for sockets

Typical TCP client-server

- Flowchart

Name and Port Number

- server process must provide that socket with a “name” so client programs can access it
- Port numbers
  - Well-known
  - Ephemeral

Wait for a connection

- Server must notify OS when it is ready to accept connections from clients
- When socket is created by **socket** function, it is assumed to be an *active* socket
- The **listen** function converts an unconnected socket into a *passive* socket

Accept a connection

- Server creates “listening socket”
- Accepts client connection

Client

- Connects to server
- Binds to socket → Connected state

Concurrent servers

- Usually need a concurrent server to handle requests
  - won't hold up other requests
- server calls **fork**
- child process services the client
- parent waits for another connection and closes connected socket

Transferring data

- *stream-based* connection
  - **read**
  - **write**

Destroy communications channel

- one way to close socket is to use the **close** function
- if socket refers to a stream-based socket, the close will block until all data has been transmitted

Destroy communications channel

- **int shutdown(int s, int how);**
- s – communications channel to shut down either or both sides
- how – tells what should be shut down
  - 0 socket is shut down for reading
  - 1 socket is shut down for writing
  - 2 both sides of socket are shut down and it becomes useless