Lecture 22: Network Programming

Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com

Some slides adapted (and slightly modified) from:
• Clark Barrett
• Jinyang Li
• Randy Bryant
• Dave O’Hallaron
Clients and server are processes running on different machines or the same machine
• Transport layer provides two connection types:
  – UDP: unreliable datagram delivery btw two processes on different hosts
  – TCP: reliable byte-stream btw two processes w/ congestion control
How to identify a connection?

- A connection consists of a pair of endpoint processes.

How to identify an endpoint?
- IP address?
- IP address: port, e.g. 216.165.108.10:80

A connection is uniquely identified by the pair of socket addresses of its endpoints:
- (cliaddr:cliport, servaddr:servport)

Port is a 16-bit integer that identifies a process on a given machine.
Anatomy of a Connection

Connection socket pair

Client socket address: 128.2.194.242:51213

Server socket address: 216.165.108.10:80

51213 is an ephemeral port allocated by the kernel

80 is a well-known port used by Web servers
Clients and Servers

• Client programs
  – E.g. Firefox, Chrome, Safari, ssh, dig,…

• Servers are long-running processes (daemons)
  – Run continuously waiting to serve clients’ requests
  – E.g. Apache web server, Mail server, sshd

• How does a client find the server?
  – The IP address in the server socket identifies the host
  – The (well-known) port in the server socket identifies the server process
  – Examples of well known ports
    • Port 22: sshd server
    • Port 25: Mail server
    • Port 80: Web server

See /etc/services for a comprehensive list of the port mappings on a Linux machine
What is a socket?

• An abstract interface provided to the application programmer
  – File descriptor, allows apps to read/write to the network
• Allows to processes on remotely connected computers to talk to each other
Programming with Socket API

- UNIX networking is modeled after file I/O
  - To the kernel, a socket is an endpoint of communication
  - To an application, a socket is a file descriptor that lets the application read/write from/to the network

```
clientfd = socket(AF_INET, SOCK_STREAM, 0)
```

The sockets interface is a set of functions that are used in conjunction with the Unix I/O functions to build network applications
Sockets

• The IP address and port number are always stored in network (big-endian) byte order, even if the machine is little-endian.

• Internet socket addresses are stored in 16-byte structures of the type sockaddr_in
Overview of the Sockets Interface

Client

- Create socket, obtain clientfd
- Connect client’s socket to server’s IP:port
- Network IO via read/write calls
- Closing clientfd shuts down connection

```c
int socket(int domain, int type, int protocol);
#include <sys/socket.h>
int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);

TCP connection start packet (SYN)
TCP connection finished packet (FIN)
```
Overview of the Sockets Interface

Client

- `socket`
- `connect`
- `write`
- `read`
- `close`

Server

- `socket`
- `bind`
- `listen`
- `accept`
- `read`
- `write`
- `Read EOF or write error`
- `close`

Create socket, obtain serverfd
Bind serverfd to a specific port
Start to listen for client connections
Blocks till a connection is accepted
Overview of the Sockets Interface

- **Office Telephone Analogy for Server**
  - **Socket:** Buy a phone
  - **Bind:** Tell phone company what number you want to use
  - **Listen:** Plug the phone in
  - **Accept:** Answer the phone when it rings
Important

• Clients are active entities that initiate connection requests.

• Servers are passive entities that wait for connection requests from clients.

• By default, the kernel assumes that a descriptor created by the socket function corresponds to an active socket that will live on the client end of a connection.

• A server calls the listen function to tell the kernel that the descriptor will be used by a server instead of a client.
Questions

• **On the server, why are there two descriptors (listenfd and connfd)?**
  – Listenfd
    • End point for client connection requests
    • Created once and exists for lifetime of the server
  – Connfd
    • End point of the connection between client and server
    • A new descriptor is created when server accepts a client connection
    • Exists only as long as it takes to service client

• **Why the distinction?**
  – Allows for concurrent servers that can communicate over many client connections simultaneously
    • E.g., Each time we receive a new request, we fork a child to handle the request
Example program: Echo Client and Server

On Client

ranma> ./echoserver 9999

linux> ./echoclient 216.165.108.10 9999

server accepted a connection 128.122.213.29:64690

type: hello there

server received 12 bytes

echo: HELLO THERE

type: ^D

Connection closed

On Server
Echo Server: accept Illustrated

1. Server blocks in `accept`, waiting for connection request on listening descriptor `listenfd`.

2. Client makes connection request by calling and blocking in `connect`.

3. Server returns `connfd` from `accept`. Client returns from `connect`. Connection is now established between `clientfd` and `connfd`.
Questions

• Can the server serve more than one client at a time?

• How to build a concurrent server using what we’ve learnt so far?
TCP packet exchange btw client/server

Client

SYN, seqno=34
SYN, seqno=66, ack=35
ack = 67
seq=35 “hello\n”
ack = 41
seq=67 “hello\n”
ack = 73
FIN, seqno=41, ack=41
FIN, seqno=73, ack=42
ack = 74

Server

accept returns
write
read returns
write
read returns EOF
close(connfd)

Connection established
write
read returns
connect
close(clientfd)
time
TCP handles retransmission

Sender
Packet 1
Packet 2
Packet 3
Packet 4
Packet 5
Packet 6
Retransmit packet 3

Receiver
ACK 1
ACK 2
ACK 2
ACK 2
ACK 6
ACK 2

Fast Retransmit
Based on three duplicate ACKs
Network diagnosis tool (tcpdump)

org@orgmachine$ sudo tcpdump -i lo -n port 9999

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on lo, link-type EN10MB (Ethernet), capture size 96 bytes
14:04:14 IP 127.0.0.1.43771 > 127.0.0.1.9999: Flags [S], seq 3700971665, ...
14:04:14 IP 127.0.0.1.9999 > 127.0.0.1.43771: Flags [S.], seq 3699955226, ack 3700971666, ...
14:04:14 IP 127.0.0.1.43771 > 127.0.0.1.9999: Flags [], ack 1, ...
14:04:16 IP 127.0.0.1.43771 > 127.0.0.1.9999: Flags [], seq 1:7, ack 1, ...
14:04:16 IP 127.0.0.1.9999 > 127.0.0.1.43771: Flags [], ack 7,...
14:04:16 IP 127.0.0.1.9999 > 127.0.0.1.43771: Flags [], seq 1:7, ack 7, ...
14:04:16 IP 127.0.0.1.43771 > 127.0.0.1.9999: Flags [], ack 7, ...
14:04:41 IP 127.0.0.1.43771 > 127.0.0.1.9999: Flags [F], seq 7, ack 7, ...
14:04:42 IP 127.0.0.1.9999 > 127.0.0.1.43771: Flags [F], seq 7, ack 8, ...
14:04:42 IP 127.0.0.1.43771 > 127.0.0.1.9999: Flags [], ack 8, ...

Packet sniffing tools with GUI: Wireshark, Ethereal
For More Information

  – The network programming bible

• Unix Man Pages
  – Linux> man bind
Conclusions

• Client-server is a de facto programming model for network programming.
• Client and server are processes that can run on the same or different machines.
• Communication between a client and a server is established using sockets (IP:port).