Sockets and Beginning Network Programming

Networking
CS 3470, Section 1
Sarah Diesburg
Network Software

- Having a network is nice. But if there is no software to transmit information on top of the network, what is the point?

- Operating systems implement the core networking stack
  - As application writers, we learn how to use the operating system **networking API** to send and receive packets
Network Software

- Operating systems are crucial
  - Implement drivers for networking cards
Network Software

- Receiving a packet
  - Accept packets from a network
Network Software

- Receiving a packet
  - Process the packet
    - Take off headers, reassemble multiple packets together in the right order, etc.
Network Software

- Receiving a packet
  - Hands the information in the packet to an application running on the machine
Network API - Sockets

- Socket Interface was originally provided by the Berkeley distribution of Unix
  - Now supported in virtually all operating systems

- Each protocol provides a certain set of services, and the API provides a syntax by which those services can be invoked in this particular OS
**Socket**

- **What is a socket?**
  - The point where a local application process attaches to the network
  - An interface between an application and the network
  - An application creates the socket

- **The interface defines operations for**
  - Creating a socket
  - Attaching a socket to the network
  - Sending and receiving messages through the socket
  - Closing the socket
Socket

Socket Family
- PF_INET denotes the Internet family
- PF_UNIX denotes the Unix pipe facility
- PF_PACKET denotes direct access to the network interface (i.e., it bypasses the TCP/IP protocol stack)

Socket Type
- SOCK_STREAM is used to denote a byte stream
- SOCK_DGRAM is an alternative that denotes a message oriented service, such as that provided by UDP
Creating a Socket

int sockfd = socket(address_family, type, protocol);

- The socket number returned is the socket descriptor for the newly created socket

- int sockfd = socket (PF_INET, SOCK_STREAM, 0);
- int sockfd = socket (PF_INET, SOCK_DGRAM, 0);

  The combination of PF_INET and SOCK_STREAM implies TCP
Client-Server Model

- The server is passively waiting for connections
- The client queries the server for information, the server responds to the client
Client-Server Model

■ Server
  - Passive open
  - Prepares to accept connection, does not actually establish a connection

■ Server invokes

```c
int bind (int socket, struct sockaddr *address,
         int addr_len)

int listen (int socket, int backlog)

int accept (int socket, struct sockaddr *address,
            int *addr_len)
```
Client-Server Model with TCP

- **Bind**
  - Binds the newly created socket to the specified address i.e. the network address of the local participant (the server)
  - Address is a data structure which combines IP and port

- **Listen**
  - Defines how many connections can be pending on the specified socket
Client-Server Model with TCP

- **Accept**
  - Carries out the passive open
  - Blocking operation
    - Does not return until a remote participant has established a connection
    - When it does, it returns a new socket that corresponds to the new established connection and the address argument contains the remote participant’s address
Client-Server Model with TCP

Client
- Application performs active open
- It says who it wants to communicate with

Client invokes

```c
int connect (int socket, struct sockaddr *address,
            int addr_len)
```

Connect
- Does not return until TCP has successfully established a connection at which application is free to begin sending data
- Address contains remote machine’s address
In practice

- The client usually specifies only remote participant’s address and let’s the system fill in the local information.
- Whereas a server usually listens for messages on a well-known port.
- A client does not care which port it uses for itself, the OS simply selects an unused one.
What is a port??

- When the operating system receives data from a packet, which listening application receives it?
  - Web server (HTTP)?
  - Mail server (SMTP)?
  - SSH server?

- Each packet is addressed to a certain port number
What is a port??

- Some port numbers are “well-known” numbers
  - 25 is almost always a mail server
  - 22 is almost always SSH
- Applications know to use these numbers by default
  - But many allow you to change the port number if a service is running on an abnormal port
- Take a look at the /etc/services file
  - $> nano /etc/services
/etc/services

# Network services, Internet style
#
# Note that it is presently the policy of IANA to assign a single well-known
# port number for both TCP and UDP; hence, officially ports have two entries
# even if the protocol doesn't support UDP operations.
#
# Updated from http://www.iana.org/assignments/port-numbers and other
# sources like http://www.freebsd.org/cgi/cvsweb.cgi/src/etc/services .
# New ports will be added on request if they have been officially assigned
# by IANA and used in the real-world or are needed by a debian package.
# If you need a huge list of used numbers please install the nmap package.

# TCP port service multiplexer

tcpmux 1/tcp
echo 7/tcp
echo 7/udp
discard 9/tcp  sink null
discard 9/udp  sink null
systat 11/tcp
daytime 13/tcp
daytime 13/udp
netstat 15/tcp
qotd 17/tcp
msp 18/tcp
msp 18/udp
chargen 19/tcp  ttypst source
chargen 19/udp  ttypst source
ftp-data 20/tcp
tftp 21/tcp
ftp 21/udp
fasp 21/udp
ssh 22/tcp
ssh 22/udp
telnet 23/tcp
smtp 25/tcp  mail

# message send protocol

quote

# SSH Remote Login Protocol

fspd
Once a connection is established, the application process invokes two operations

```c
int send (int socket, char *msg, int msg_len,
        int flags)

int recv (int socket, char *buff, int buff_len,
          int flags)
```
Control Flow

TCP Client

Socket()
connect()  Connection establishment
send()
recv()
close()

TCP Server

socket()
bind()  Well-known port
listen()
accept()
blocks until connection from client
recv()
process request
send()
read()
close()
For now, use “localhost” for “host” when invoking the client

- “localhost” resolves to IP address 127.0.0.1, which is a shortcut for meaning “this machine”
- Can change this when more machines are in network
Introduction to Program 1

- [Here I will introduce Program 1]
Program 1 Hints

- If you are not familiar with C, play with test files
- Download simpletalk textbook example program and play with it (step 1)
  - Remember, change SERVER_PORT to your UNI ID number so no one else uses it at the same time!!
- Think about how you must modify the design to fit the program 1 specifications