

# Programming with Network Sockets

ECE 650  
Systems Programming & Engineering  
Duke University, Spring 2017

## Sockets

- We've looked at shared memory vs. message passing
  - All on a single system (meaning running under a single OS)
- What about communication across distributed processes
  - Running on different systems
  - Assume systems are connected by a network (e.g. the internet)
- We can program using network sockets
  - For creating connections and sending / receiving messages
  - Often follows a client / server pattern
- We will assume basic network knowledge
  - E.g. what is an IP address
  - We will cover the networking stack in more detail in next lectures

ECE 650 – Fall 2017

2

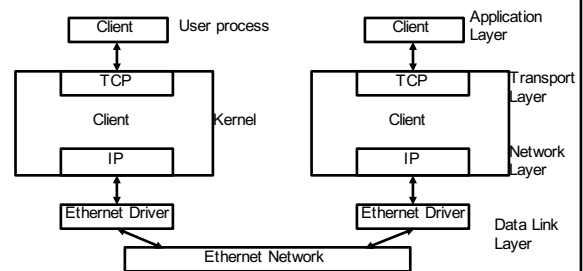
## Client-Server Model

- Common communication model in networked systems
  - Client typically communicates with a server
  - Server may connect to multiple clients at a time
- Client needs to know:
  - Existence of a server providing the desired service
  - Address (commonly IP address) of the server
- Server does not need to know either about the client

ECE 650 – Fall 2017

3

## Client-Server Overview



- Client and Server communicating across Ethernet using TCP/IP

ECE 650 – Fall 2017

4

## TCP – Connection-oriented Service

- Transmission Control Protocol
  - Designed for end-to-end byte stream over unreliable network
  - Robust against failures and changing network properties
- TCP transport entity
  - e.g. Library procedure(s), user processes, or a part of the kernel
  - Manages TCP streams and interfaces to the IP layer
  - Accepts user data streams from processes
  - Breaks up into pieces not larger than 64KB
    - Often 1460 data bytes to fit in 1 Ethernet frame w/ IP + TCP headers
  - Sends each piece separately as IP datagram
  - Destination machine TCP entity reconstructs original byte stream
  - Handles retransmissions & re-ordering
- Connection-oriented transport layer
  - Provides error-free, reliable communication
  - Can think of communication between two processes on different machines as just like UNIX pipes or fifos
    - One process puts data in one end, other process takes it out

ECE 650 – Fall 2017

5

## Network Sockets

- Network interface is identified by an IP address
  - Or a hostname, which translates into an IP address
  - E.g. 127.0.0.1, localhost or login.oit.duke.edu
- Interface has 65536 ports (0-65535)
- Processes attach to ports to use network services
  - Port attachment is done with "BIND" operation
- Allows application-level multiplexing of network services
  - E.g. SSH vs. Web vs. Email may all use different ports
  - Many ports are standard (e.g. 80 for web server, 22 for SSH)
  - You may have seen URLs like <http://127.0.0.1:4444>
    - 127.0.0.1 is the IP, 4444 is the port

ECE 650 – Fall 2017

6

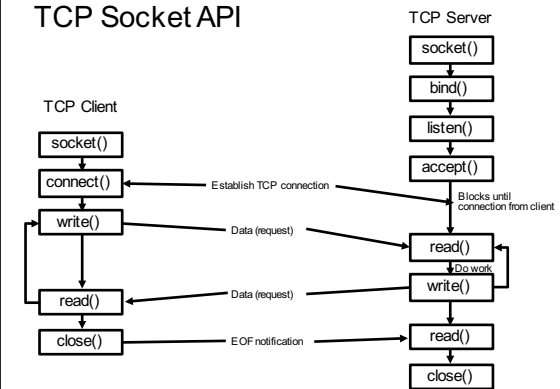
## TCP Service Model

- TCP service setup as follows:
  - Two endpoint processes create endpoints (sockets)
  - Each socket has an address: IP address of host + 16-bit **port**
  - API functions used to create & communicate on sockets
- Ports
  - Numbers below 1024 called “well-known ports”
    - Reserved for standard services, like FTP, HTTP, SMTP
    - <http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml>
  - But not all services usually used & active all at once
    - Don't want them all active, just waiting for incoming connections
  - Special daemon: inetd (Internet daemon)
    - Attaches to multiple ports
    - Waits for incoming connection
    - fork()'s of the new, appropriate process to handle that connection

ECE 650 – Fall 2017

7

## TCP Socket API



ECE 650 – Fall 2017

8

## Example – UNIX TCP sockets

- Let's look at example code...
- Here is a great reference for use of socket-related calls
  - <http://beej.us/guide/bgnet/>

Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Attach a local address to a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Block the caller until a connection attempt arrives
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECV	Receive some data from the connection
CLOSE	Release the connection

ECE 650 – Fall 2017

9

## Server-Side Structure

- Often follows a common pattern to serve incoming requests

```

pid_t pid;
int listenfd, connfd;
listenfd = socket(...);

/*fill the socket address with server's well known port*/
bind(listenfd, ...);
listen(listenfd, ...);

for ( ; ; ) {
    connfd = accept(listenfd, ...); /* blocking call */
    if ( (pid = fork()) == 0 ) { /* create a child process to service */
        close(listenfd); /* child closes listening socket */

        /*process the request doing something using connfd */
        /* ..... */

        close(connfd);
        exit(0); /* child terminates */
    }
    close(connfd); /*parent closes connected socket*/
}
    
```

ECE 650 – Fall 2017

10