EECS 489 - FA 24

Discussion 1

Introduction to Socket Programming

Outline

- Logistics
- Introduction to socket programming
- Socket programming demo

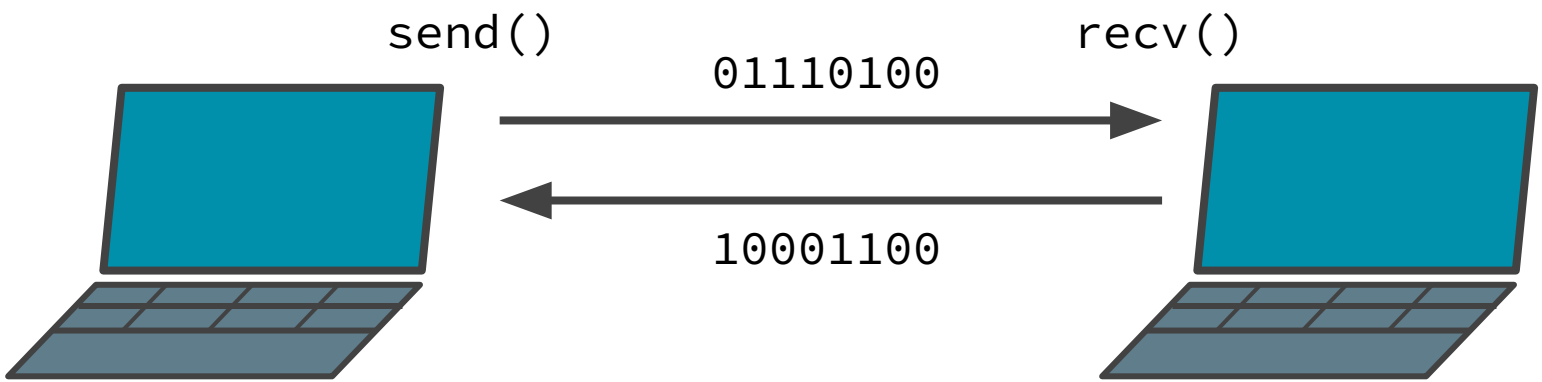
Logistics

- Course website: http://www.eecs489.org/
- Complete AWS setup exercise by tomorrow!
  - Ubuntu instances on AWS will be used for all assignments.
- A1 will be out soon! Due Monday, Sept. 18.
  - All assignments are submitted through https://g489.eecs.umich.edu.
  - A1 is individual, while the rest can be completed in groups of 2-3 people.
  - A1 focuses on creating a tool similar to `iPerf`, which measures the speed of a network connection.

What is a socket?

A socket is a **two-way communication channel** between two processes or machines.

Sockets abstract away many of the annoying parts about communicating over a network, providing a clean application programming interface (API) to exchange data.



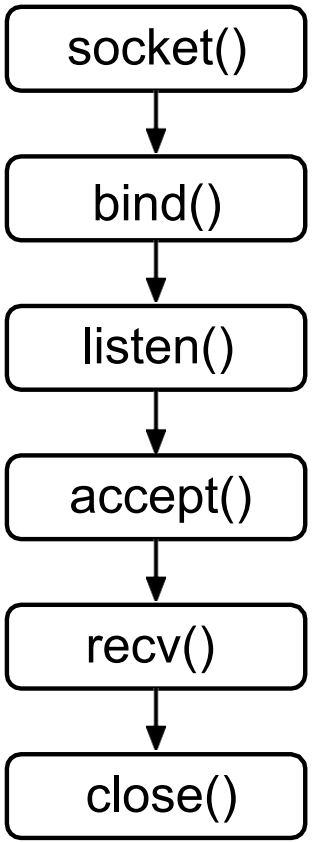
Socket API

```
int socket(domain, type, protocol);

int send(sockfd, buffer, size, flags);
int recv(sockfd, buffer, size, flags);

int bind   (sockfd, addr);
int listen (sockfd, backlog);
int accept (sockfd, 0, 0);
int connect(sockfd, addr);
int close  (sockfd);
```

Servers vs. Clients

- **Servers:** Actively listening for connections using sockets.
- **Clients:** Initiating connections using sockets.

Server-side sockets



Create a socket.

Bind the socket to a port.

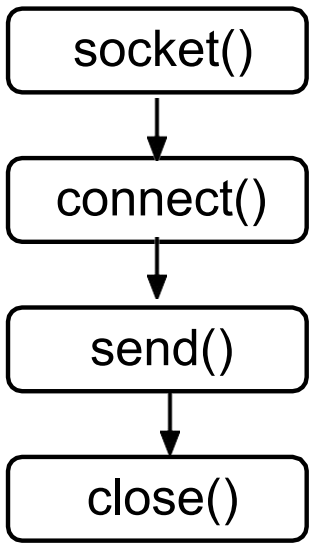
Begin listening for incoming connection requests.

Accept a connection request.

Receive a stream of bytes (can also send things)

Close the connection.

Socket Programming: Client Side



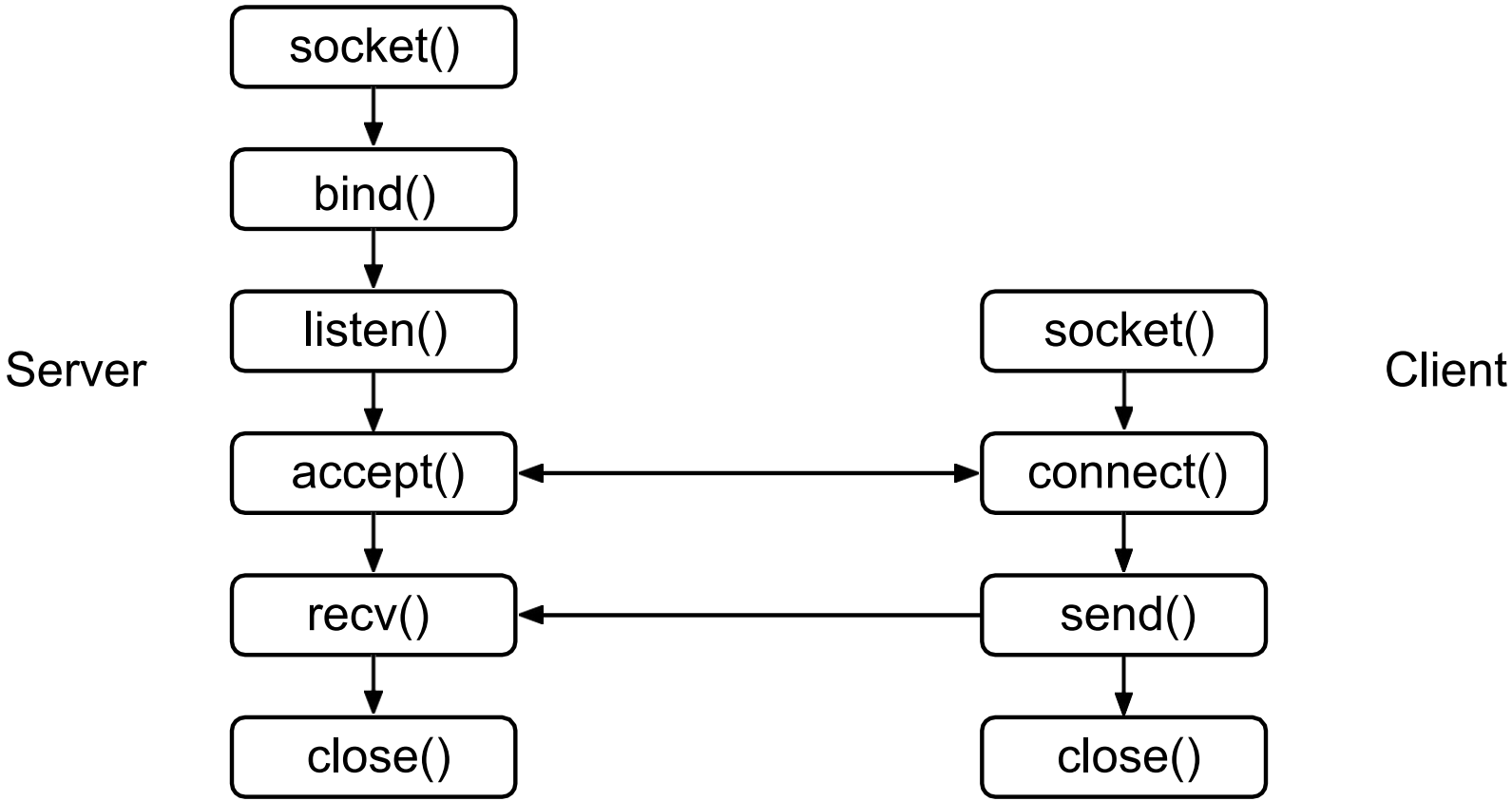
Create a socket.

Connect to a server.

Send some information (can also recv)

Close the connection.

Socket Programming: Complete Flow



socket() (for server and client)

```
int socket(int domain, int type, int protocol);
```

Creates a socket, returning a file descriptor that can be used to reference it.

- **domain:** specifies the communication domain; `AF_INET` is for IPv4 protocol.
- **type:** specifies the communication semantics; for a two-way communication socket, we use `SOCK_STREAM`
- **protocol:** specifies the protocol; this will be `IPPROTO_TCP` for our purposes.

Note: These constants are defined in sys/socket.h.

```
int sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
```

https://man7.org/linux/man-pages/man2/socket.2.html

bind() (for server)

```
int bind(int sockfd, const struct sockaddr *addr,
         socklen_t addrlen);
```

Binds the socket to a specific **port**, among other things.

- **sockfd:** The socket file descriptor from socket()
- **addr:** A sockaddr struct that specifies some parameters, including which port to be used (via `addr.sin_port`). If this is set to 0, the OS chooses a port for you. The `sockaddr` type is a generalization of `sockaddr_in`, which is the **in**ternet type.
- **addrlen:** `sizeof(sockaddr)`

Returns 0 on success, -1 on error.

https://man7.org/linux/man-pages/man2/bind.2.html

Aside: What is a port?

A number (usually a `uint16_t`) that uniquely identifies a connection endpoint.

The well-known ports (also known as *system ports*) are those numbered from 0 through 1023. The requirements for new assignments in this range are stricter than for other registrations.[2]

Notable well-known port numbers

Number	Assignment
20	File Transfer Protocol (FTP) Data Transfer
21	File Transfer Protocol (FTP) Command Control
22	Secure Shell (SSH) Secure Login
23	Telnet remote login service, unencrypted text messages
25	Simple Mail Transfer Protocol (SMTP) email delivery
53	Domain Name System (DNS) service
67, 68	Dynamic Host Configuration Protocol (DHCP)
80	Hypertext Transfer Protocol (HTTP) used in the World Wide Web
110	Post Office Protocol (POP3)
119	Network News Transfer Protocol (NNTP)
123	Network Time Protocol (NTP)
143	Internet Message Access Protocol (IMAP) Management of digital mail
161	Simple Network Management Protocol (SNMP)
194	Internet Relay Chat (IRC)
443	HTTP Secure (HTTPS) HTTP over TLS/SSL
546, 547	DHCPv6 IPv6 version of DHCP

The registered ports are those from 1024 through 49151. IANA maintains the official list of well-known and registered ranges.[3]

The dynamic or private ports are those from 49152 through 65535. One common use for this range is for ephemeral ports.

Aside: sockaddr_in

```c
int make_server_sockaddr(struct sockaddr_in *addr, int port) {
    // Step (1): specify socket family.
    // This is an internet socket.
    addr->sin_family = AF_INET;

    // Step (2): specify socket address (hostname).
    // The socket will be a server, so it will only be listening.
    // Let the OS map it to the correct address.
    addr->sin_addr.s_addr = INADDR_ANY;

    // Step (3): Set the port value.
    // If port is 0, the OS will choose the port for us.
    // Use htons to convert from local byte order to network byte
    addr->sin_port = htons(port);

    return 0;
}
```

```c
int make_client_sockaddr(struct sockaddr_in *addr, const char *hostname,
                         int port) {
    // Step (1): specify socket family.
    // This is an internet socket.
    addr->sin_family = AF_INET;

    // Step (2): specify socket address (hostname).
    // The socket will be a client, so call this unix helper function
    // to convert a hostname string to a useable `hostent` struct.
    struct hostent *host = gethostbyname(hostname);
    if (host == NULL) {
        fprintf(stderr, "%s: unknown host\n", hostname);
        return -1;
    }
    memcpy(&addr->sin_addr, host->h_addr, host->h_length);

    // Step (3): Set the port value.
    // Use htons to convert from local byte order to network byte order.
    addr->sin_port = htons(port);

    return 0;
}
```

Aside: errors

```
#include <sys/socket.h>

int bind(int sockfd, const struct sockaddr *addr,
         socklen_t addrlen);
```

RETURN VALUE top

On success, zero is returned. On error, -1 is returned, and *errno* is set to indicate the error.

NAME top

errno - number of last error

DESCRIPTION top

The *<errno.h>* header file defines the integer variable *errno*, which is set by system calls and some library functions in the event of an error to indicate what went wrong.

listen() (for server)

```
int listen(int sockfd, int backlog);
```

DESCRIPTION top

listen() marks the socket referred to by *sockfd* as a passive socket, that is, as a socket that will be used to accept incoming connection requests using accept(2).

The *sockfd* argument is a file descriptor that refers to a socket of type **SOCK_STREAM** or **SOCK_SEQPACKET**.

The *backlog* argument defines the maximum length to which the queue of pending connections for *sockfd* may grow. If a connection request arrives when the queue is full, the client may receive an error with an indication of **ECONNREFUSED** or, if the underlying protocol supports retransmission, the request may be ignored so that a later reattempt at connection succeeds.

RETURN VALUE top

On success, zero is returned. On error, −1 is returned, and *errno* is set to indicate the error.

https://man7.org/linux/man-pages/man2/listen.2.html

connect() (for client)

```
int connect(int sockfd, const struct sockaddr *addr,
            socklen_t addrlen);
```

The **connect**() system call connects the socket referred to by the file descriptor *sockfd* to the address specified by *addr*. The *addrlen* argument specifies the size of *addr*. The format of the address in *addr* is determined by the address space of the socket *sockfd*; see socket(2) for further details.

https://man7.org/linux/man-pages/man2/connect.2.html

accept() (for server)

```
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

The **accept**() system call is used with connection-based socket types (**SOCK_STREAM**, **SOCK_SEQPACKET**). It extracts the first connection request on the queue of pending connections for the listening socket, *sockfd*, creates a new connected socket, and returns a new file descriptor referring to that socket. The newly created socket is not in the listening state. The original socket *sockfd* is unaffected by this call.

This one is tricky!

- **addr**: parameter used for returning; it will be filled with the address of the peer socket.
- **addrlen**: a value-return parameter; when called, it should have the size of addr. When returned, it will contain the actual size of what is stored in addr.

Returns **a connection file descriptor** (i.e. a new socket) that will be used for communicating with this specific client.

```
int connfd = accept(sockfd, NULL, NULL);
```

https://man7.org/linux/man-pages/man2/accept.2.html

send()

```
ssize_t send(int sockfd, const void *buf, size_t len, int flags);
```

Sends the message described by **buf** and **len** over the connection described by sockfd.

- **Server:** `sockfd` should be the `connfd` (from accept) used to communicate with the client.
- **Client:** `sockfd` can just be the actual `sockfd`.

Returns number of bytes actually sent; this may not be all the bytes in the message. -1 on error.

```
inline int send_data(int connectionfd, const char *message,
                     size_t message_len) {
    size_t sent = 0;
    do {
        const ssize_t n =
            send(connectionfd, message + sent, message_len - sent, 0);
        sent += n;
    } while (sent < message_len);
    return 0;
}
```

https://man7.org/linux/man-pages/man2/send.2.html

recv()

```
ssize_t recv(int sockfd, void *buf, size_t len, int flags);
```

Receives up to **len** bytes from **sockfd** and stores them in **buf.**

- **Server:** `sockfd` should be the `connfd` (from accept) used to communicate with the client.
- **Client:** `sockfd` can just be the actual `sockfd`.

Returns the number of bytes actually received; this can be less than the maximum number specified in len.

Use the `MSG_WAITALL` flag to receive the whole message (this can still be less than len).

```
ssize_t num_bytes_recv = recv(sockfd, data, 1000, MSG_WAITALL);
```

https://man7.org/linux/man-pages/man2/recvmsg.2.html

close()

```
int close(int fd);
```

Closes the file descriptor passed.

Returns 0 on successful close, -1 on error.

https://man7.org/linux/man-pages/man2/close.2.html

Note: Byte Order

Internet Convention: Big Endian
Host: Machine-dependent

```
int val = 0x0A0B0C0D;
/* little-endian: 0D 0C 0B 0A */
/* big-endian:    0A 0B 0C 0D */
```

Function	Description
`htons()`	`h`ost `to` `n`etwork `s`hort
`htonl()`	`h`ost `to` `n`etwork `l`ong
`ntohs()`	`n`etwork `to` `h`ost `s`hort
`ntohl()`	`n`etwork `to` `h`ost `l`ong

General Programming Advice

Always check the return values of functions:

```
// (1) Create socket
int sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_IP);
if (sockfd == -1) {
    cout << "Error opening stream socket" << endl;
    return;
}
```

Use assertions if you expect something to never fail:

```
int close_retval = close(sockfd);
assert(close_retval == 0);
```

Resources & Advice

- Beej's Guide to Network Programming
- Example code: https://github.com/wlfuh/bgreeves.sockets.starter/tree/master
  - Feel free to copy-paste portions of this code for your Assignment 1 as necessary!
- Use existing documentation (i.e. man pages) – these are generally very complete!
  - Nice UI wrapper: https://man7.org/linux/man-pages/
- StackOverflow and ChatGPT can be very helpful for socket programming
  - Many of these things are formulaic; others likely have encountered similar problems.
  - Understanding code instead of pure copy-paste can help prevent further questions down the line.

Useful Libraries

```
#include <arpa/inet.h>      // htons(), ntohs()
#include <netdb.h>      // gethostbyname(), struct hostent
#include <netinet/in.h>     // struct sockaddr_in
#include <stdio.h>      // perror(), fprintf()
#include <string.h>     // memcpy()
#include <sys/socket.h>     // getsockname()
#include <unistd.h>      // stderr
#include <time.h>      //time(&time_t)
```

Credit

Slides adapted from EECS 482 discussion materials and EECS 489 discussion slides from previous semesters.