CS 43: Computer Networks

05: Socket Programming

September 12-14, 2018
Socket Programming

Client-Server communication

• Client:
  – initiates communication
  – must know the address and port of the server
  – active socket

• Server:
  – passively waits for and responds to clients
  – passive socket
TCP Socket Procedures: Client

- `socket()`
  - create a new communication endpoint

- `connect()`
  - actively attempt to establish a connection

- `send()`
  - receive some data over a connection

- `recv()`
  - send some data over a connection

- `close()`
  - release the connection
TCP Socket Procedures: Server

- **socket()**
  - create a new communication endpoint

- **bind()**
  - attach a local address to a socket

- **listen()**
  - announce willingness to accept connections

- **accept()**
  - block caller until a connection request arrives

- **recv()**
  - receive some data over a connection

- **send()**
  - send some data over a connection

- **close()**
  - release the connection
TCP socket connection

Server

- socket()
- bind()
- listen()
- accept()
- recv()
- send()
- close()

Client

- socket()
- connect()
- send()
- recv()
- close()
If the client sends a GET request to the server using `send()` but forgets to send the last `/r/n` which of the following can happen?

A. Server, Client both `recv()`
B. Server `send()`s, Client `recv()`s
C. Server `recv()`s, Client `send()`s
D. Some other combination
Create and close sockets

Server
- `socket()`
- `bind()`
- `listen()`
- `accept()`
- `recv()`
- `send()`
- `close()`

Client
- `socket()`
- `connect()`
- `send()`
- `recv()`
- `close()`
What is a socket?

An abstraction through which an application may send and receive data, in the same way as an open-file handle allows an application to read and write data to stable storage.
Create a TCP socket: socket()

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

int sock = socket(AF_INET, SOCK_STREAM, 0);

• domain: communication domain of the socket: generic interface.

• type of socket: reliable vs. best-effort

• end-to-end protocol: TCP for a stream socket -
  — 0: default E2E for specified protocol family and type.
Create a TCP socket: `socket()`

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

int sock = socket(AF_INET, SOCK_STREAM, 0);

/*
 * AF_INET: Communicate with IPv4 Address Family (AF),
 * SOCK_STREAM: Stream-based protocol
 * int sock: returns an integer-valued socket descriptor or handle
 */

if(sock < 0) {
    perror("socket");
    exit(1);
}
```
Close a socket: close()

```c
int close(int socket)
if (close(sock)) {
    perror("close");
    exit(1);
}
/* int socket: int socket descriptor is passed to close()*/
• Close operation similar to closing a file.
• initiate actions to shut down communication
• deallocate resources associated with the socket
• cannot send(), recv() after you close the socket.
```
connect()

• Before you can communicate, a connection must be established.
• Client Initiates, Server waits.
• Once connect() returns, socket is connected and we can proceed with send(), recv()

```c
int connect(int socket,
            const struct sockaddr *foreign Address,
            socklen_t addressLength)
```
int connect(int socket,
            const struct sockaddr *foreign Address,
            socklen_t addressLength)

struct sockaddr_in addr;
int res = connect(sock, (struct sockaddr*)&addr, sizeof(addr));

/* int socket: socket descriptor
   foreignAddress: pointer to sockaddr_in containing Internet address, port of server.
   addressLength: length of address structure */
send(), recv()
send(), recv()

Socket is connected when:
• client calls connect()
• connected socket is returned by accept() on server

ssize_t send(int socket, const void *msg, msgLength, int flags)
ssize_t recv (int socket, void *rcvBuffer, size_t bufferLength, int flags)

/* int socket: socket descriptor
    return: # bytes sent/received or -1 for failure.
send():
• by default send: blocks until data is sent

ssize_t send(int socket, const void *msg, msgLength, int flags)

/* int socket: socket descriptor
send(): msg: sequence of bytes to be sent
send(): msgLength: # bytes to send
send(), recv()

recv():

```c
ssize_t recv (int socket, void *rcvBuffer, size_t bufferLength, int flags)
int recv_count = recv(sock, buf, 255, 0);
```

/* int socket: socket descriptor

void *rcvBuffer: generally a char array

size_t bufferLength: length of buffer: max # bytes that can be received at once.

flags: setting flag to zero specifies default behavior.

⚠️ Place all send() and recv() calls in a loop, until you are left with no more bytes to send or receive. One call to send()/recv(), irrespective of the buffer does not necessarily mean all your data will be received at once.
Auxiliary functions: specifying addresses

Applications need to identify the remote end-host with which they communicate.

- Client must specify address + port number.
- Sockets API:
  - generic datatype: sockaddr
  - specify addresses associated with sockets
  - for IPv4: sockaddr_in
  - passing sockaddr_in to socket functions requires a cast:
    - struct sockaddr_in addr;
    - (struct sockaddr*)&(addr);
IPv4 address structure

struct sockaddr_in{
    sa_family_t sin_family;   //Internet protocol (AF_INET)
    in_port_t sin_port;      // Address port (16 bits)
    struct in_addr sin_addr; //IPv4 address (32 bits)
    char sin_zero[8];        // Not used
};

struct in_addr {
    uint32_t s_addr;        // that's a 32-bit int (4 bytes)
};
Auxiliary functions: getaddrinfo()

- getaddrinfo()
  - refer to hosts by name: demo.cs.swarthmore.edu.
    - easy to remember
    - insulation from IP address changes.
    - more on name resolution later.
  - however protocols use addresses (binary numbers) not names.
  - DNS: distributed database that maps domain names to Internet addresses.

- addr_in struct
Auxiliary functions: getaddrinfo()

int getaddrinfo(
  const char *node,   // e.g. demo.cs.swarthmore.edu
  const char *service, // e.g. "http" or port number
  const struct addrinfo *hints,
  struct addrinfo **res);

getaddrinfo(argv[1],0,0,&result)

- argv[1] valid host name
- service, hints set to 0: set to default sock type SOCK_STREAM (TCP)
- Function gives a pointer to a linked-list: res (results)
Auxiliary functions: inet_ntoa()

```
inet_ntoa(((struct sockaddr_in*)result->ai_addr)->sin_addr));
```

char *net_ntoa(struct in_addr in)

- convert IPv4 to an ASCII string in dotted-decimal notation: a.b.c.d

inet_aton()

- deprecated use inet_ntop()/ inet_pton()
Auxiliary functions: inet_ntoa()

```c
struct sockaddr_in antelope;
char *some_addr;

inet_aton("10.0.0.1", &antelope.sin_addr);
// store IP in antelope

some_addr = inet_ntoa(antelope.sin_addr);
// return the IP

printf("%s\n", some_addr); // prints "10.0.0.1"

// and this call is the same as the inet_aton() call, above:
antelope.sin_addr.s_addr = inet_addr("10.0.0.1");
```