Socket Programming

CPSC 441 - TUTORIAL 2

RACHEL MCLEAN

WINTER 2020

*Modified from slides by Reza Gholizadeh

What is a Socket?

The application wants to send and receive data via network

- A web browser

An interface is needed to pass data between the application (layer) and the network (layer)

Socket Types

TCP SOCKET

Type: SOCK_STREAM

Reliable delivery

In-order guaranteed

Connection-oriented

Bidirectional

UDP SOCKET

Type: SOCK_DGRAM

Unreliable delivery

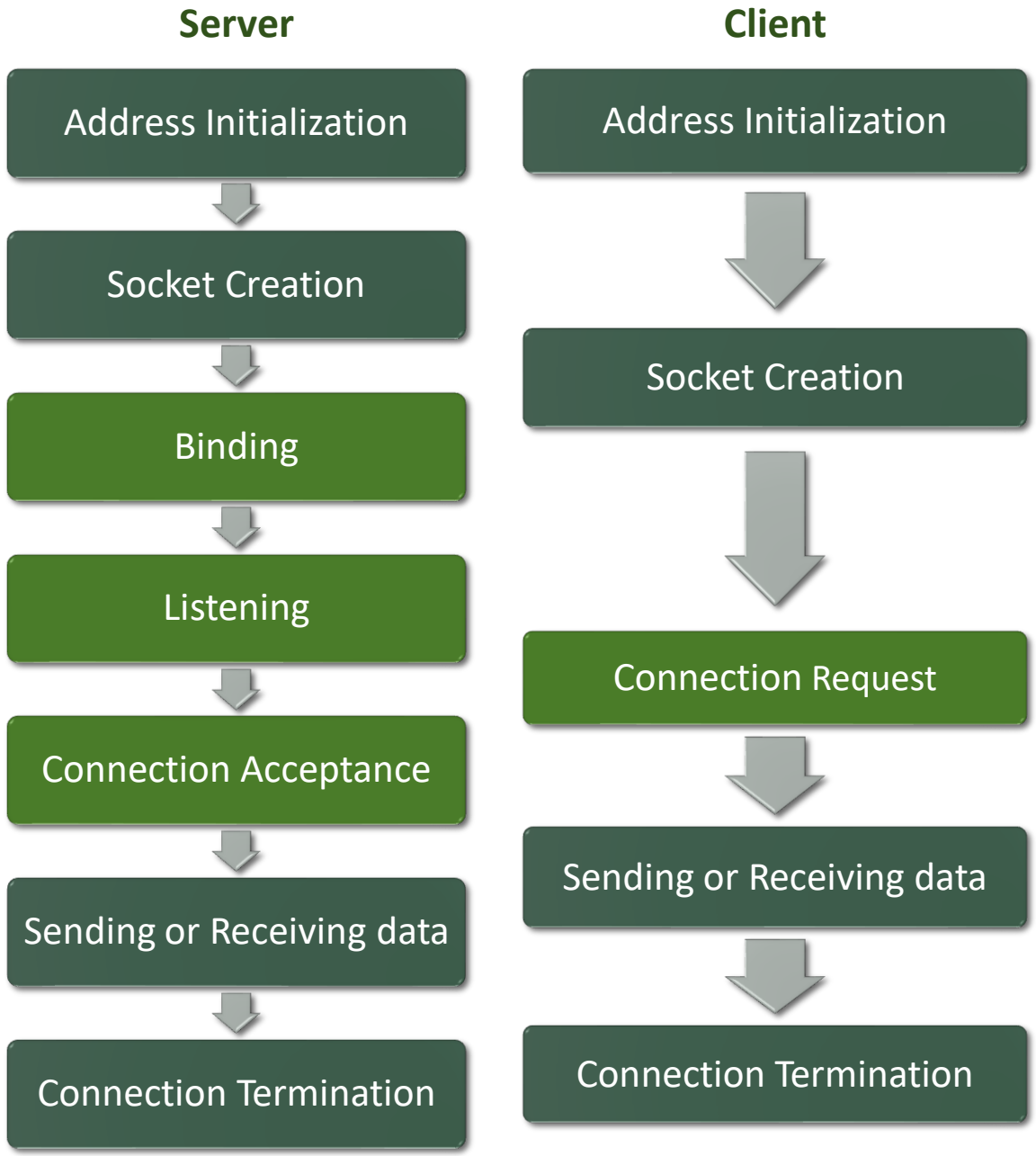
No order guaranteed

No notion of “connection” – application destination for each packet

Can send of receive

Socket Programming Process in C

- Two sides of socket programming:
  - **Server** side
  - **Client** side
- Proxy is **both** server & client



Address Initialization

```
struct sockaddr_in{
   short sin_family;
   u_short sin_port;
   struct in_addr sin_addr;
   char sin_zero[8];
}
```

- **sin_family**: specifies the address family – e.g. AF_INET
- **sin_port**: specifies the port number – 0 to 65535 (Use 8000 to 9000)
- **sin_addr**: specifies the IP address
- **sin_zero**: unused

```
1.  struct sockaddr_in address;
2.  memset(&address, 0, sizeof(address));
3.  address.sin_family = AF_INET;
4.  address.sin_port = htons(port);
5.  address.sin_addr.s_addr = htonl(INADDR_ANY)
```

address: the variable that you define for initializing the sockaddr_in

memset(): initialize the address to zero

AF_INET: corresponds to IPv4 protocol as communication domain

htons(): convert 16-bit host-byte-order (little endian) to network-byte-order (big endian)

htonl(): convert 32-bit host-byte-order (little endian) to network-byte-order (big endian)

INADDR_ANY: any IP address on the local machine

Socket Creation

socket(domain, type, protocol)

- **domain**: communication domain – integer
- **type**: socket type – SOCK_STREAM or SOCK_DGRAM
- **protocol**: network protocol – 0 is default (TCP)
- other parameters: rarely used

```
1.  int mysocket1;
2.  mysocket1 = socket(AF_INET, SOCK_STREAM, 0);
3.  if(mysocket1 == -1){
4.      printf("socket() call failed");
5.  }
```

mysocket1: socket descriptor (sockid) – integer

AF_INET: corresponds to IPv4 protocol as communication domain

SOCK_STREAM: TCP socket type

0: TCP network protocol

UNIVERSITY OF CALGARY

Binding (server only)

bind(sockid, &addrport, size)

- **sockid**: socket descriptor - integer
- **addrport**: the IP address and port number of the machine stored in `sockaddr struct`
- **size**: the size of the `sockaddr struct`

```
1.  int status;
2.  status = bind(mysocket1, (struct sockaddr *)
3.  &address, sizeof(struct sockaddr_in));
4.  if(status==-1){
5.      printf("bind() call failed");
6. }
```

status: error status of `bind()` function
- `-1` if bind failed
- `>0` if bind succeed

mysocket1: previously created socket

`(struct sockaddr *)&address`: reference of initialized server address struct casted to `sockaddr struct` pointer

`sizeof(struct sockaddr_in)`: size of `sockaddr_in struct`

UNIVERSITY OF CALGARY

Listening (server only)

`listen``(sockid, queuelen)`

- **`sockid`**: socket descriptor - integer
- **`queuelen`**: the number of active participants that can wait for a connection

```
1.  int status;
2.  status = listen(mysocket1,5);
3.  if(status==-1){
4.      printf("listen() call failed");
5.  }
```

`status`: error status of `listen()` function

- `-1` if bind failed
- `>0` if bind succeed

`mysocket1`: previously created socket

`5`: five participants can wait for a connection if the server is busy

UNIVERSITY OF CALGARY

Connection request (client only)

```
1.  int status;
2.  status = connect(mysocket1, (struct sockaddr
    *)&address, sizeof(struct sockaddr_in));
3.  if(status==-1){
4.      printf("connect() call failed");
5.  }
```

connect(sockid, &addr, addrlen)

- **sockid**: socket descriptor - integer
- **addr**: address of server which is stored in `struct sockaddr`
- **addrlen**: size of `addr`

status: error status of `connect()` function
- `-1` if bind failed
- `>0` if bind succeed

mysocket1: previously created socket

`(struct sockaddr *)&address`: reference of initialized server address struct casted to `sockaddr struct` pointer

`sizeof(struct sockaddr_in)`: size of `sockaddr_in struct`

*Note: the connect process is **blocking** and waits for the server to accept the connection

Connection Acceptance (server only)

```
1.  int mysocket2;
2.  mysocket2 = accept(mysocket1, NULL, NULL);
3.  if(mysocket2 == -1){
4.      printf("accept() call failed");
5.  }
```

accept(sockid, &addr, addrlen)

- **sockid**: listening socket descriptor - integer
- **addr**: address of active participant will be stored in addr in struct sockaddr format
- **addrlen**: size of addr

mysocket1: listening socket descriptor (sockid) – integer

mysocket2: new socket descriptor for accepted connection – integer

If you don't want to store clients you can use **NULL** as the second and third parameters of the accept() function

*Note: the accept process is **blocking** and waits for connection before retuning

Send and Receive

recv(sockid, &buff, len, flags)

- **sockid**: socket descriptor – integer
- **buff**: stores received bytes
- **Len**: length of buffer (in bytes) – integer
- **flags**: special options, usually set to 0 – integer

send(sockid, &buff, len, flags)

- **sockid**: socket descriptor – integer
- **buff**: buffer to be transmitted
- **Len**: length of buffer (in bytes) – integer
- **flags**: special options, usually set to 0 – integer

```
1.  int count;
2.  char snd_message[100] = {"hello"};
3.  char rcv_message[100];
4.  count = send(mysocket2, snd_message, 5, 0);
5.  if(count == -1){
6.      printf("send() call failed.")
7.  }
8.  count = recv(mysocket2, rcv_message, 100, 0);
9.  if(count == -1){
10.     printf("recv() call failed.")
11. }
```

mysocket2: the socket that holds the established connection

5: the length of **snd_message** which is equal to number of characters in "hello"

100: the maximum length for **rcv_message** is used here

Connection termination

close(sockid)

- **sockid**: listening socket descriptor – integer
- Closing the socket will close the connection and the port used by it will be freed up

```
1.  close(mysocket2);
2.  close(mysocket1);
```

*Note: always close the socket after you are done using it

Server and Client Interactions