Lab2 – TCP Socket Programming

Introduction to Computer Networks

2023/04/20

Why socket programming?

- Socket programming is the foundation of network communication
- Helps to gain a deeper understanding of how networks work and how data is transmitted

Lab 2

- Socket programming
- Create TCP Header
  - Calculate the checksum

(This will be the requirements in the assignment!)

TCP vs UDP

UDP

- Connectionless, best-effort protocol.
- Target IP address and port number are required.

TCP

- Connection-oriented protocol
- Provides reliable data transmission

Transmission Control Protocol

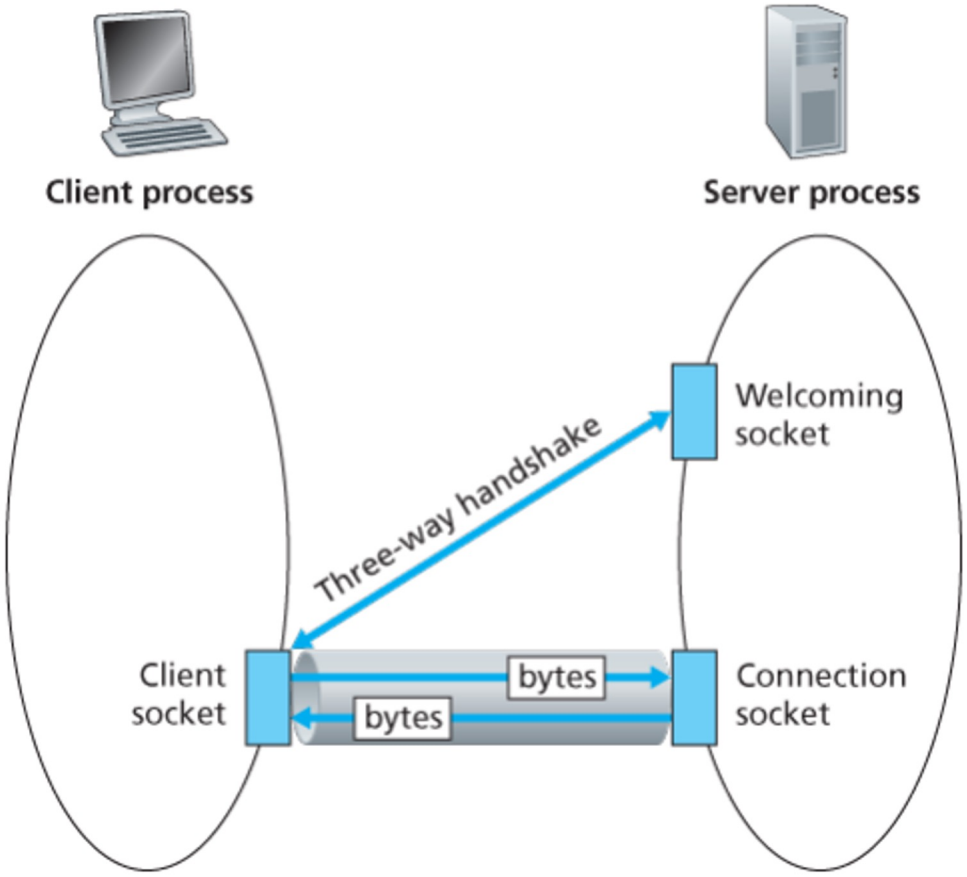
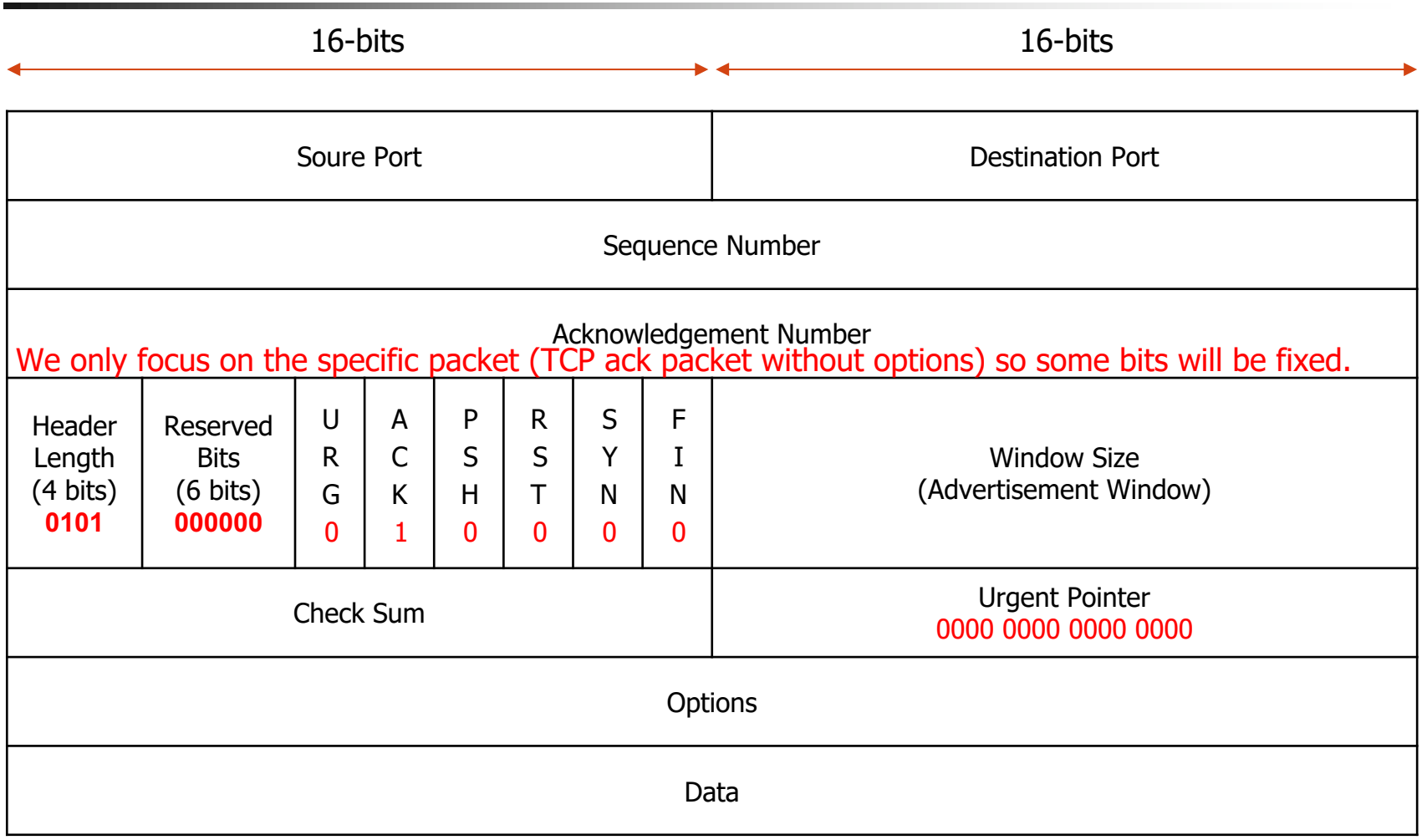


Figure 2.28 *The TCPServer* process has two sockets

TCP Header


16-bits
16-bits
Soure Port
Destination Port
Sequence Number
Acknowledgement Number
We only focus on the specific packet (TCP ack packet without options) so some bits will be fixed.
Header Length (4 bits) 0101
Reserved Bits (6 bits) 000000
URG 0
ACK 1
PSH 0
RST 0
SYN 0
FIN 0
Window Size (Advertisement Window)
Check Sum
Urgent Pointer 0000 0000 0000 0000
Options
Data

Checksum

- What is checksum?
  - An error detection method used by upper layer protocols.
- How to calculate checksum?
  - Ref: https://www.geeksforgeeks.org/calculation-of-tcp-checksum/

Checksum

2. TCP checksum

```
▼ Internet Protocol Version 4, Src: 10.5.4.107, Dst: 10.8.9.237
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 48
    Identification: 0xcc61 (52321)
  ▶ Flags: 0x02 (Don't Fragment)
    Fragment offset: 0
    Time to live: 64
    Protocol: TCP (6)
  ▶ Header checksum: 0x4c02 [validation disabled]
    Source: 10.5.4.107
    Destination: 10.8.9.237
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
▼ Transmission Control Protocol, Src Port: 62429 (62429), Dst Port: 3283 (3283), Seq: 3657103398, Len: 0
    Source Port: 62429
    Destination Port: 3283
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 3657103398
    Acknowledgment number: 0
    Header Length: 28 bytes
  ▶ Flags: 0x002 (SYN)
    Window size value: 65535
    [Calculated window size: 65535]
  ▶ Checksum: 0x8ee9 [validation disabled]
    Urgent pointer: 0
  ▶ Options: (8 bytes), Maximum segment size, SACK permitted, End of Option List (EOL)
0000  00 22 83 9e 50 8d a4 5e  60 b7 d7 03 08 00 45 00   ."..P..^ `.....E.
0010  00 30 cc 61 40 00 40 06  4c 02 0a 05 04 6b 0a 08   .0.a@.@. L....k..
0020  09 ed f3 dd 0c d3 d9 fa  f8 26 00 00 00 00 70 02   ........ .&....p.
0030  ff ff 8e e9 00 00 02 04  05 b4 04 02 00 00         ........ ......
```

TCP封包資料

Checksum

(1) Pseudo Header: Source IP + Destination IP + Protocol + TCP header length
0a05 + 046b + 0a08 + 09ed + 0006 + 001c
= 2287

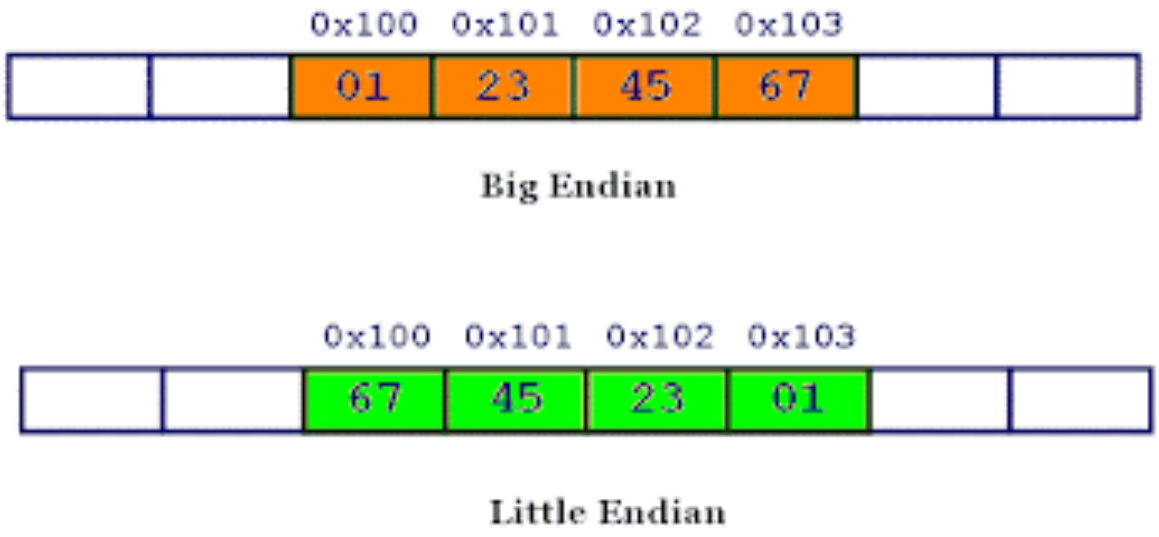
(2) TCP header
Sum the data in groups of 2 bytes (except the checksum field)
f3 dd 0c d3 d9 fa f8 26 00 00 00 00 70 02 ff ff 8e e9 00 00 02 04 05 b4 04 02 00 00
f3dd + 0cd3 +d9fa +f826 + 7002 + ffff + 0204 + 05b4 + 0402
= 44e8b

Add (1) and (2) together
2287 + 44e8b = 47112

End-around carry
4 + 7112 = 7116 (0111 0001 0001 0110)

1's complement of the result
1000 1110 1110 1001 -> 8e e9 (final result)

Little Endian & Big Endian



- The TCP header is Big Endian (MSB in Low Memory Address)
- Our computer is Little Endian (MSB in High Memory Address)

When creating the header, you need to be aware of this problem!

The Assignment

The Assignment

In lab 2, you will each get a zip file containing:

1. client.c
2. server.c
3. header.h
4. header.o
5. makefile

Linux Socket Programming

```
struct sockaddr_in {
      sa_family_t     sin_family;  /* address family: AF_INET */
      in_port_t         sin_port;     /* stores port in network byte
order
                                          htons(port)*/
      struct in_addr sin_addr;    /* internet address
                                          INADDR_ANT for server
                                          inet_addr(IP) for client*/
};

/* Internet address */
struct in_addr {
      uint32_t          s_addr;      /* address in network byte order
*/
};
```

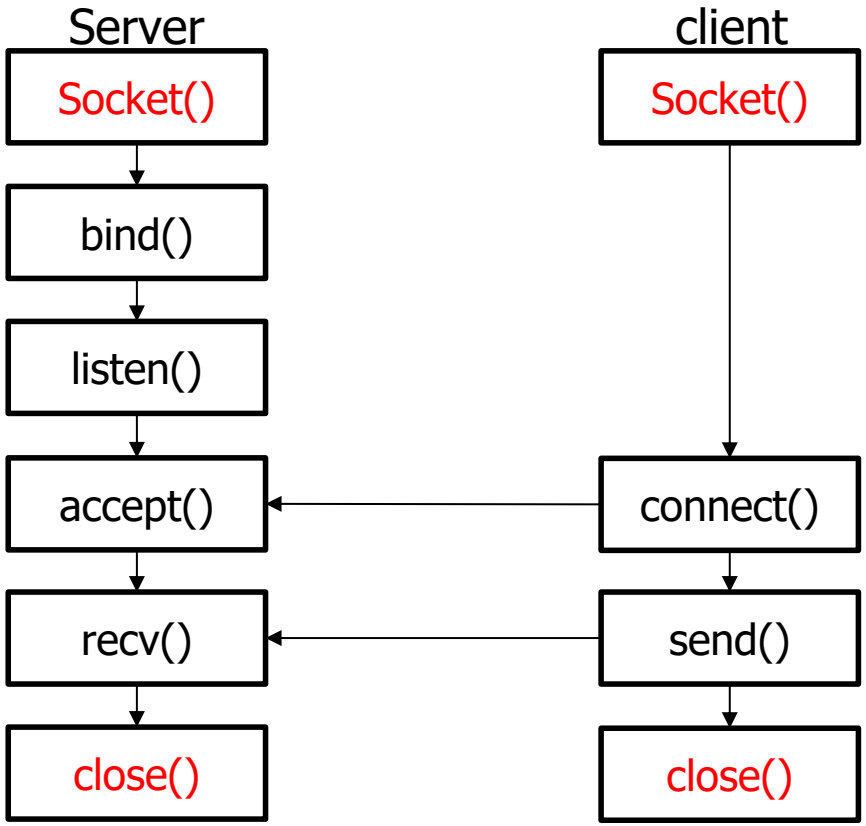
Linux Socket Programming

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

int socket(int domain, int type, int protocol);
int close(int fd);

domain:  AF_INET IPv4 Internet protocols
type:       SOCK_STREAM TCP
protocol: 0
```

TCP flow


Server
client
Socket()
Socket()
bind()
listen()
accept()
connect()
recv()
send()
close()
close()

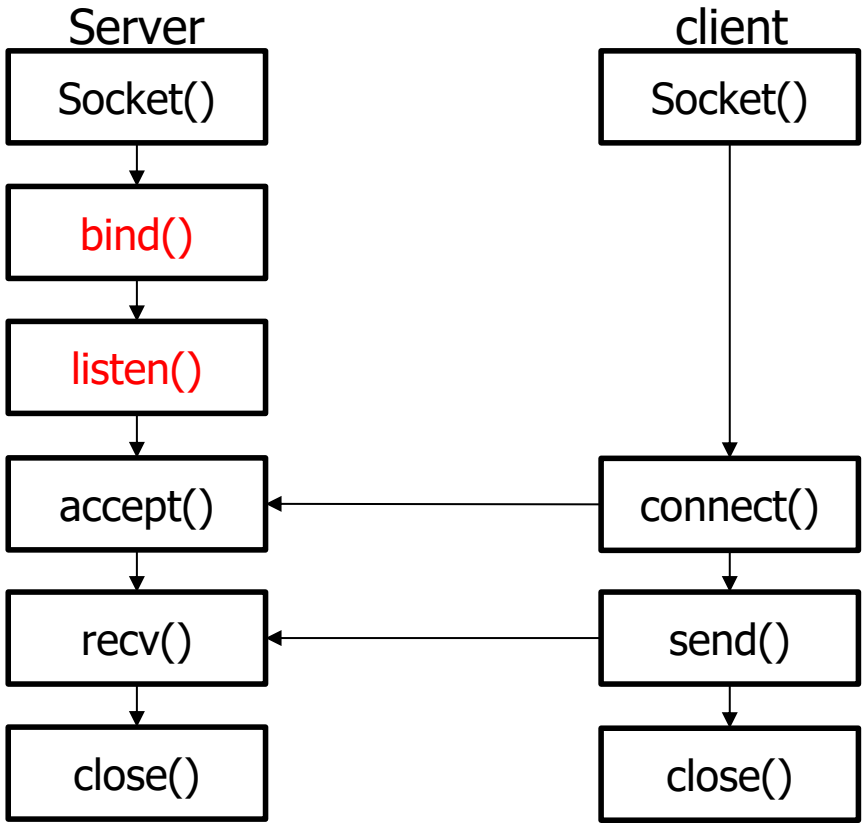
Linux Socket Programming

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

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

sockfd: return value of socket()
addr: sockaddr_in for IPv4
addrlen : sizeof(addr)

TCP flow

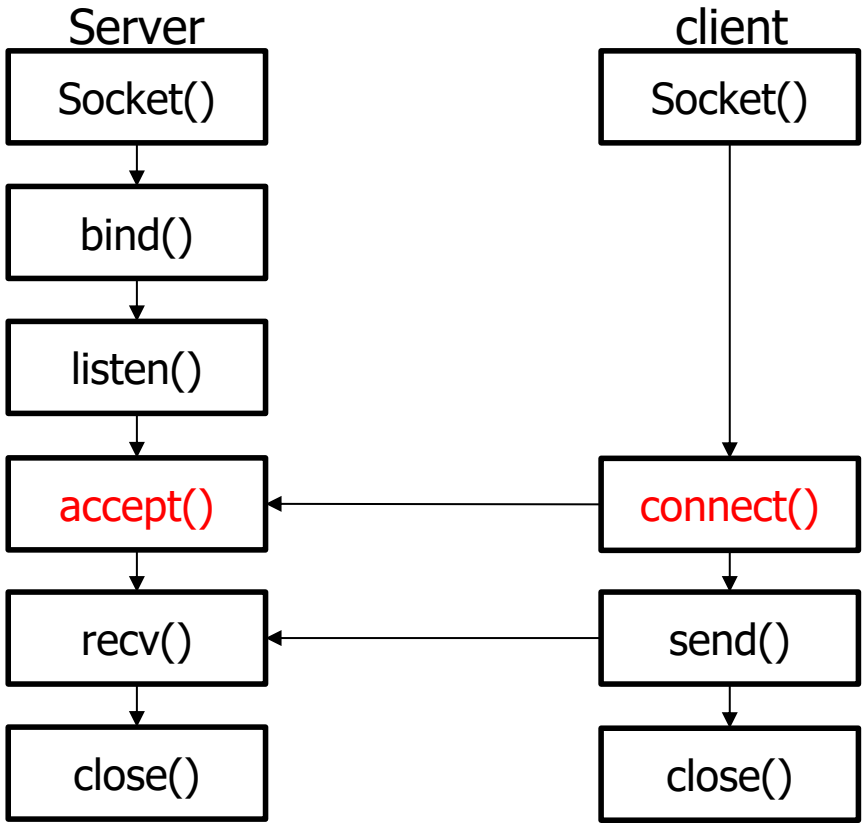

Server
client
Socket()
Socket()
bind()
listen()
accept()
connect()
recv()
send()
close()
close()

Linux Socket Programming (TCP)

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

int connect(int sockfd, const struct sockaddr *addr,
                socklen_t addrlen);
int accept(int sockfd, struct sockaddr *restrict addr,
                socklen_t  *restrict addrlen);
ssize_t send(int sockfd, const void *buf, size_t len,
                int flags);
ssize_t recv(int sockfd, void *buf, size_t len, int flags);
```

TCP flow


Server
client
Socket()
Socket()
bind()
listen()
accept()
connect()
recv()
send()
close()
close()

Linux Socket Programming

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

recvfrom(int sockfd, void *restrict buf, size_t len, int flags,
                struct sockaddr *restrict src_addr,
                 socklen_t *restrict addrlen);
```

sockfd: return value of socket()
buf: data transmitted
len: sizeof(buf)
flags: 0
src_addr: sockaddr_in for IPv4
addrlen : sizeof(src_addr)

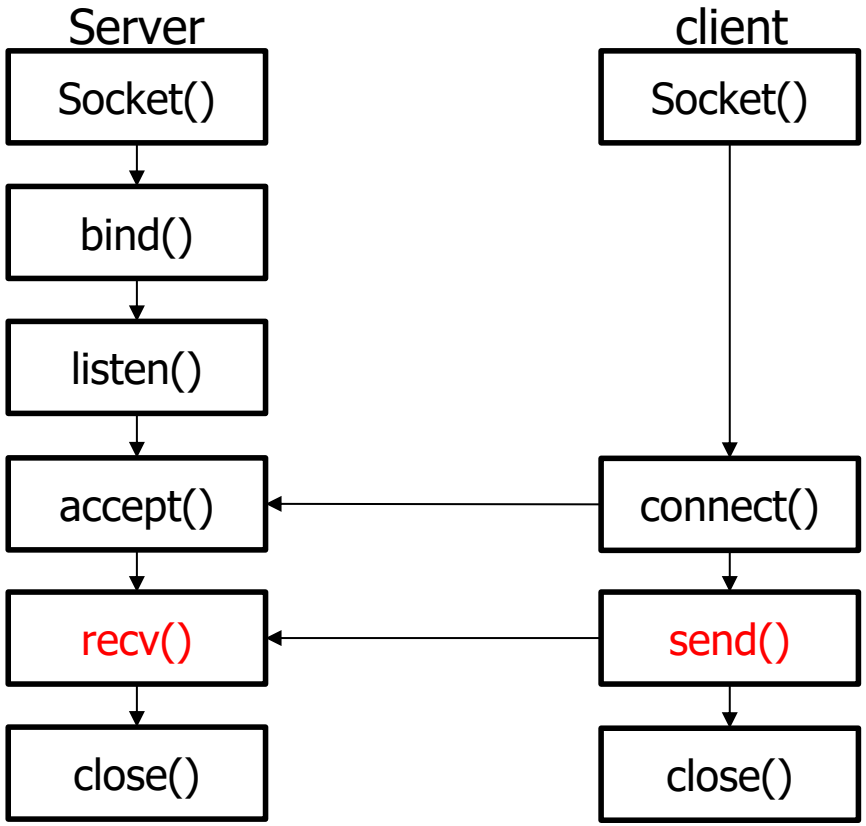
Linux Socket Programming

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

ssize_t sendto(int sockfd, const void *buf, size_t len, int
flags,
                       const struct sockaddr *dest_addr,
                       socklen_t addrlen);
```

sockfd: return value of socket()
buf: data transmitted
len: sizeof(buf)
flags: 0
dest_addr: sockaddr_in for IPv4
addrlen : sizeof(dest_addr)

TCP flow


Server
client
Socket()
Socket()
bind()
listen()
accept()
connect()
recv()
send()
close()
close()

header.h

Segment (struct)

```
typedef struct Segment{
    char header[20];
    char pseudoheader[12];
    L3info l3info;
    L4info l4info;
}Segment;
```

(All the information will store in it for you to create TCP header!)

header.h

Some **predefined** functions:

```
void serverfunction(int clientfd);
void receivedata(int sockfd,Segment* s);
void sendheader(int sockfd, char* header);
```

(Be careful not to change this part!)

Makefile

After you finish writing your code,
compile your code using the command “make”.

```
(base) alice@Alices-MacBook-Air lab2 % make
```

Requirements

1. Implementation (70%)

- TASK 1 (70%)
  - Connect server and client with TCP socket and successfully send a message.
- TASK 2 (18%)
  - Create TCP header (without checksum) using l4info.
- TASK 3 (12%)
  - Complete the header (checksum).

2. Report (30%)

- Questions are in the next slide.

Lab 2 percentage: 10%

Report

1. What does INADDR_ANY mean? (10pts)
2. What's the difference between bind() and listen()? (10pts)
3. Usually, we set up the server's port and exclude the client's. Who determines the client's port and what's the benefit? (20pts)
4. What is little endian and big endian? Why do most network byte order use big endian? (10pts)
5. Why do we need a pseudoheader ? (10pts)
6. For the code below, what's difference between client_fd and socket_fd ? (10pts)

   ```
   client_fd = accept(socket_fd, (struct sockaddr *)&clientAddr, (socklen_t*)&client_len);
   ```

7. When using the send() function and recv() function, why do we not need the address? (10pts)
8. Write about what you have learned from Lab 2. (20pts)

Submission

- Submit a zip file that consists of
  - client.c
  - server.c
  - makefile
  - header.o
  - header.h
  - report.pdf
- Name the zip file Lab2_{studentID}_{compiler}.zip (e.g Lab2_11001234_gcc/intel_clang/m1_clang.zip)
- Submission deadline: 5/11 (Thu) 11:59 PM

Some commands

- make #run Makefile to compile
  - Remember save the updated code before you make.
- ./server #run the server
- ./client #run the client
- CTRL + C #exit server
- tcp.flags.ack && frame.len==54 #Filter of Wireshark to find the packet we use in this lab.

Some problems

- If you cannot execute the server. Try use the command
  - lsof -i tcp:45525 (Find the process using the port)
  - kill <PID> (kill the proccess)
  - For example :

```
years@years-System-Product-Name:~/Desktop/CNlab-student-$ ./server
years@years-System-Product-Name:~/Desktop/CNlab-student-$ ./server
years@years-System-Product-Name:~/Desktop/CNlab-student-$ lsof -i tcp:45525
COMMAND    PID  USER   FD   TYPE  DEVICE SIZE/OFF NODE NAME
server  110764 years    3u  IPv4 1048399      0t0  TCP *:45525 (LISTEN)
years@years-System-Product-Name:~/Desktop/CNlab-student-$ kill 110764
years@years-System-Product-Name:~/Desktop/CNlab-student-$
```

- If you have any questions, please post it on eeclass first!
- If you fail to compile (if the program can run but with some compiler version warning, please ignore it.) make sure you download the correct zip file (only for MacOS and Ubuntu.)
- If you still fail to compile, e-mail us with the screenshot.