Lab 2 - Socket Programming

Introduction to Computer Networks

Kuan-Wei Huang(黃冠維), Pei-Chieh Wu (吳沛潔), Cheng-Yuan Jian (簡呈原), Hsiang-Ting Huang (黃湘庭), Pham Ngoc Hoa (范玉花)

Purpose

- Learn what is Socket and its process
- Connect Socket to Wireshark
- Learn how TCP Checksum works

Prepare Windows Subsystem for Linux

- VScode WSL

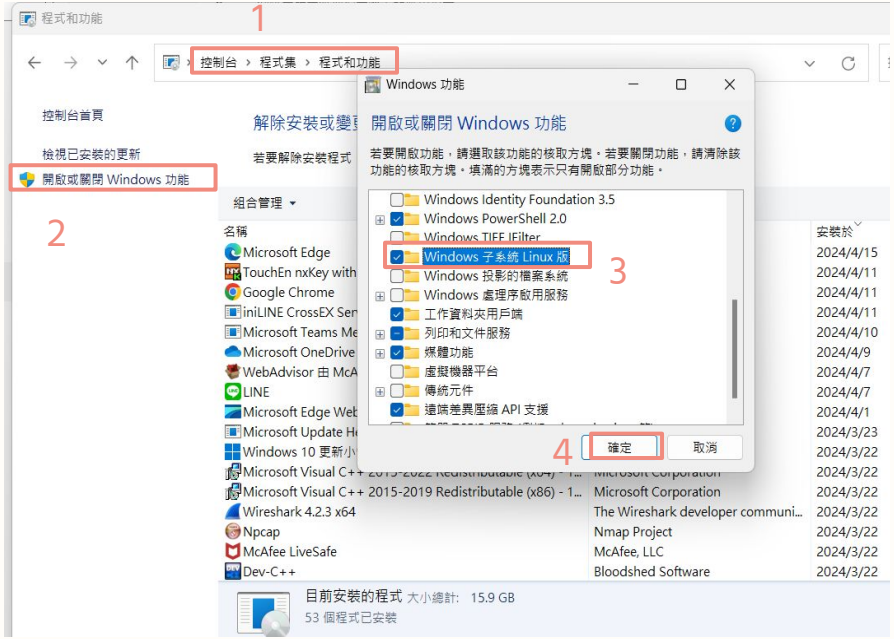
  https://code.visualstudio.com/docs/remote/wsl

  NOTE: MacOS doesn’t need any setting before directly use.

Prepare Windows Subsystem for Linux

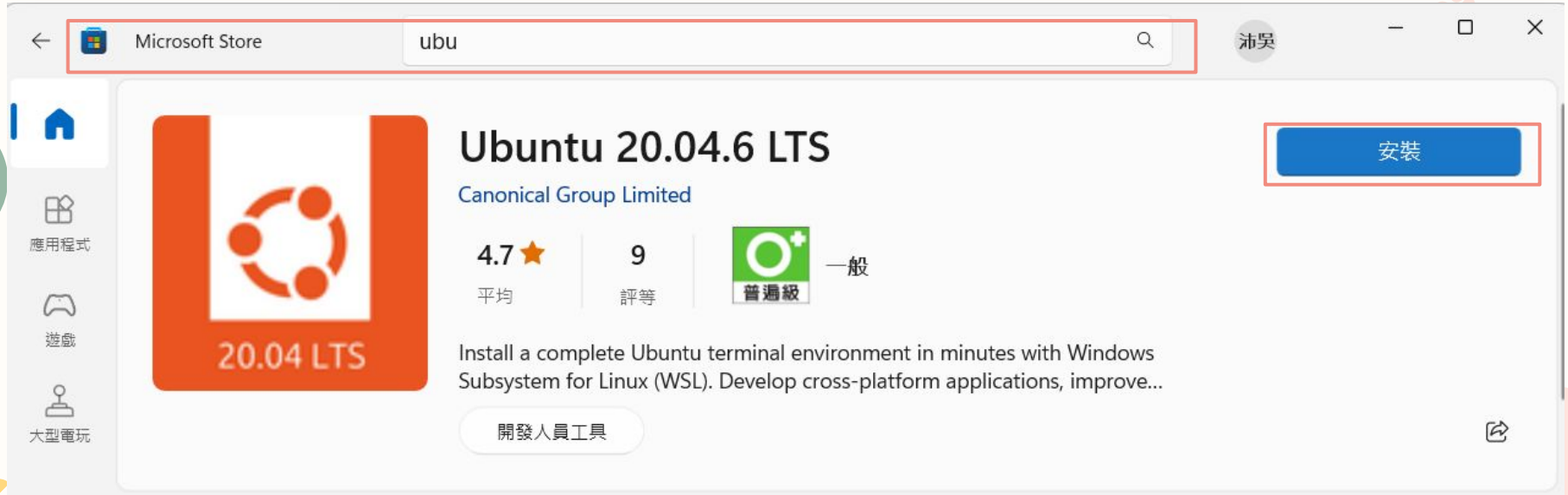
- Change the authentication to build WSL

1. Go to Control Panel>Programs>Programs & Features
2. Click “Turn Windows features on or off” on the left
3. Select “Windows Subsystem for Linux”
4. Press OK and restart your computer



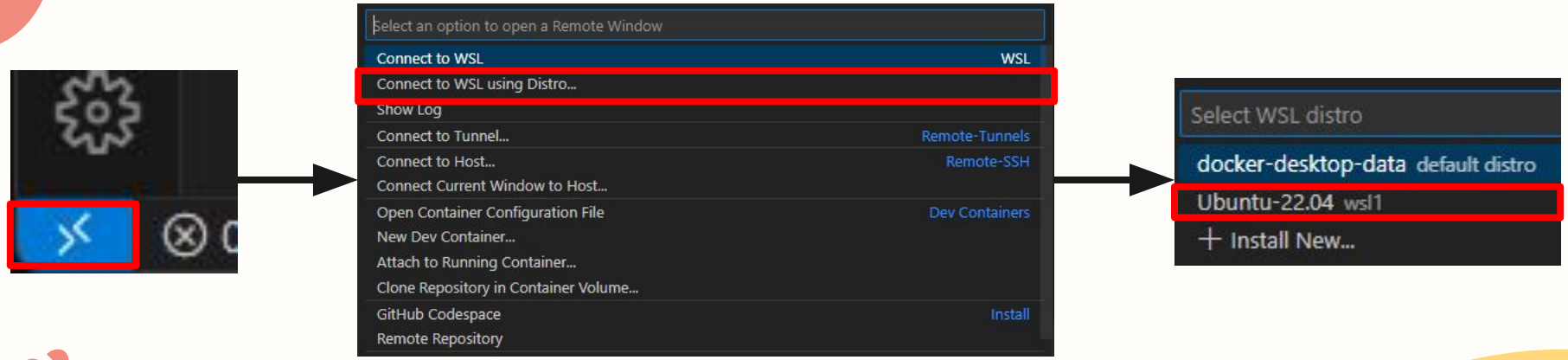
Install Ubuntu LTS

1. Open Microsoft store, search for ubuntu 22.04 LTS, download it, and activate it
2. Open ubuntu 22.04 LTS, create your own account → Finished !



Developing in WSL with VSCode (Optional)

- Tutorial: https://code.visualstudio.com/docs/remote/wsl



Install Packages

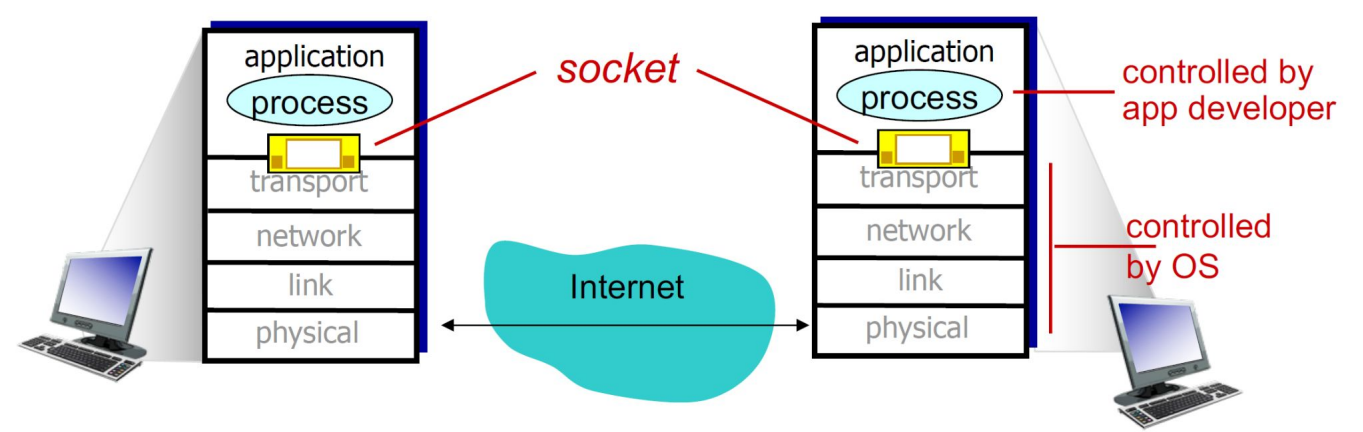
1. sudo apt update
2. sudo apt install build-essential
3. sudo apt install gcc
4. sudo apt install make

```
root@DESKTOP-CM77LAB:/home# gcc --version
gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0
Copyright (C) 2021 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

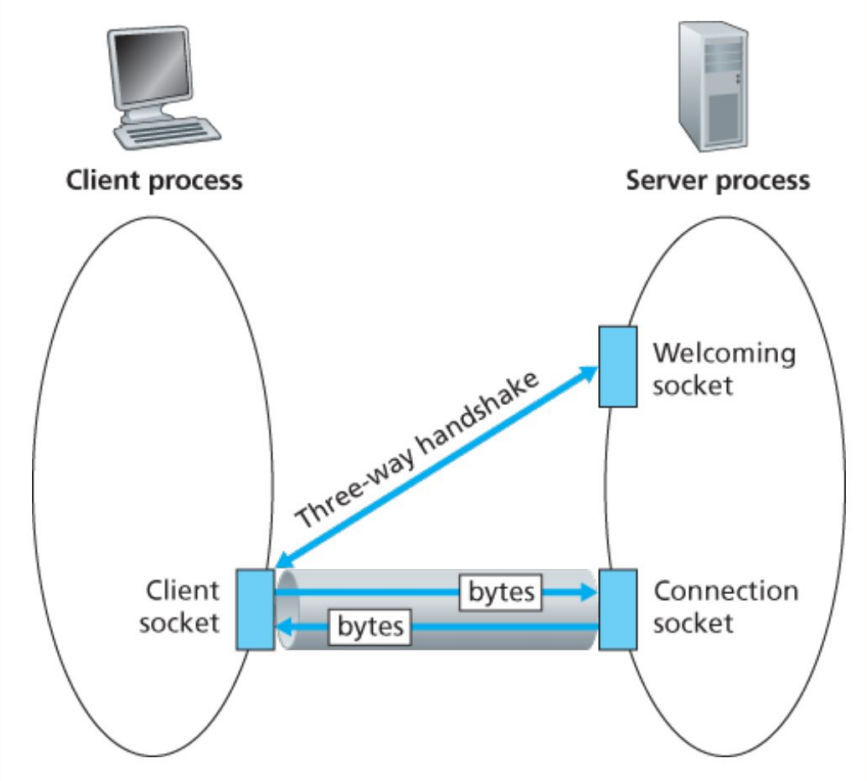
What is Socket

Socket

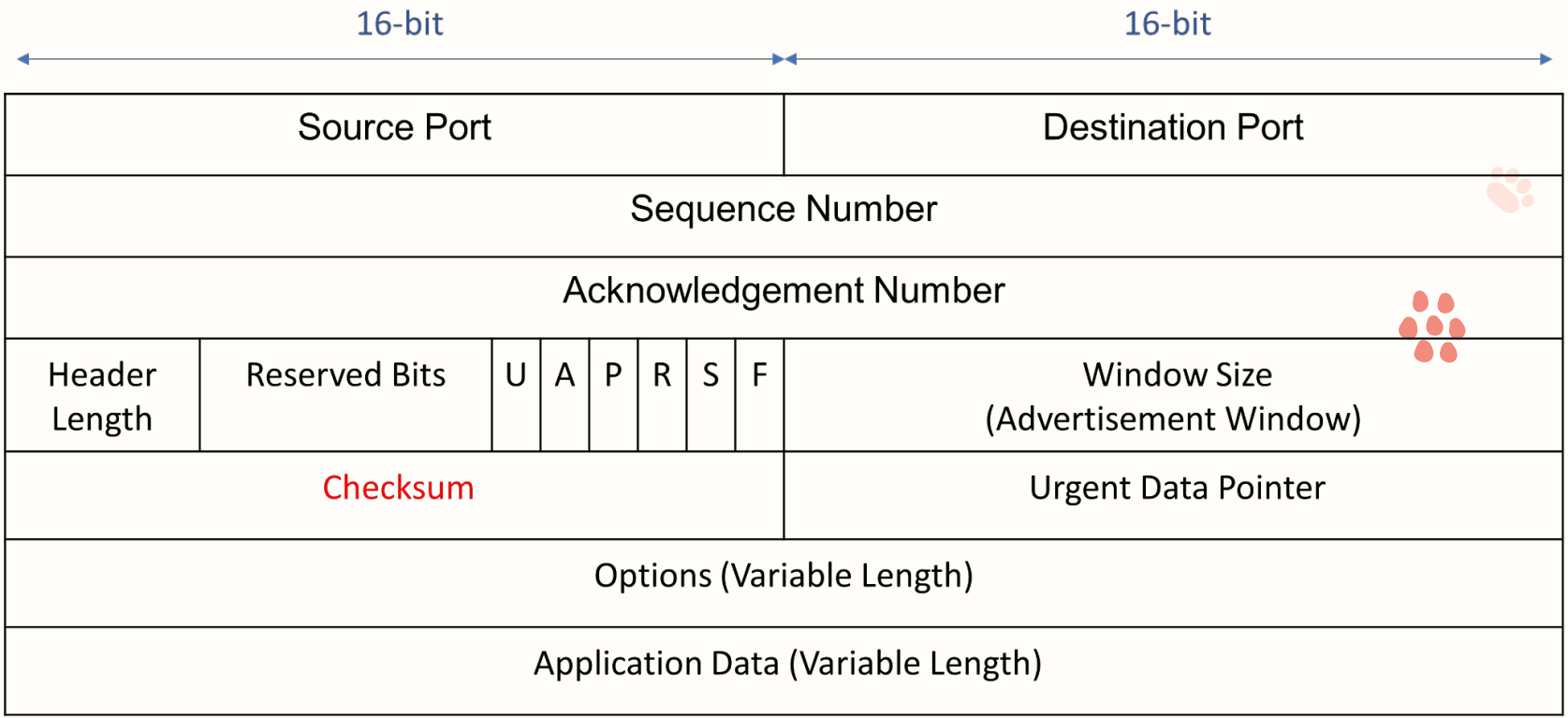
- Socket: Door between application process and end-end-transport protocol
- Socket types for two transport services:
  - UDP: unreliable datagram
  - TCP: reliable, byte stream-oriented



TCP (Transmission Control Protocol)



TCP Header


16-bit
16-bit
Source Port
Destination Port
Sequence Number
Acknowledgement Number
Header Length
Reserved Bits
U
A
P
R
S
F
Window Size (Advertisement Window)
Checksum
Urgent Data Pointer
Options (Variable Length)
Application Data (Variable Length)

Connect Socket to Wireshark

Wireshark - Capture Packets from local to local

- For WSL
  - Since WSL (Windows Subsystem for Linux) establishes a virtual network layer on Windows for communication, and Windows wireshark does not support localhost (127.0.0.1) traffic by default.
  - Solution : `sudo tcpdump -i lo -w {student_id}.pcap`
    - `tcpdump` : versatile tool that enables you to capture and inspect network traffic in real-time
    - `-i lo` : loopback interface
    - `-w` : written into a capture file
    - `{student_id}.pcap` : the file name you capture your file.
    - For MacOS, run `sudo tcpdump -i lo0 -w {student_id}.pcap` instead

TCP Checksum

TCP Checksum

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

TCP 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

TCP 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)

be careful to skip these 8 bytes

```
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

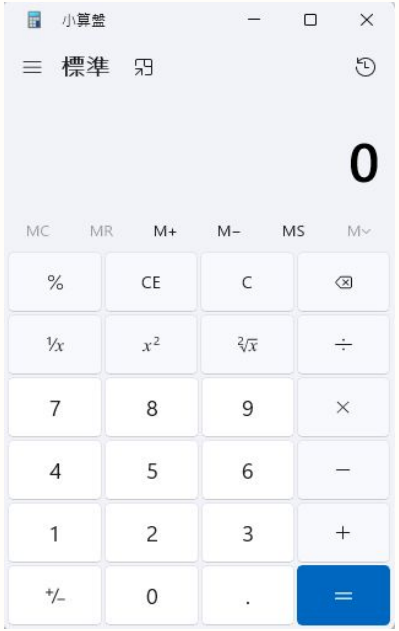
Add (1) and (2) together : 2287 + 44e8b = 47112

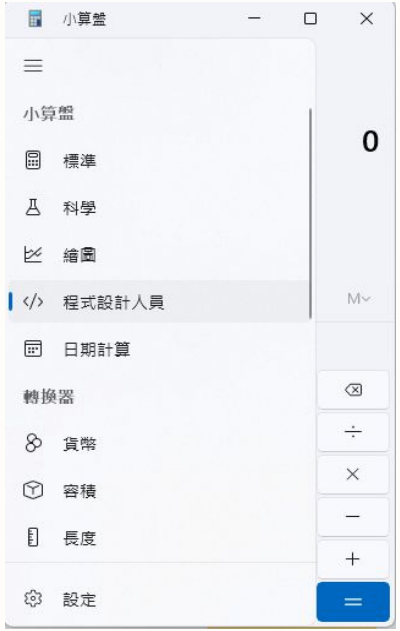
End-around carry : 4 + 7112 = 7116 (0111 0001 0001 0110)

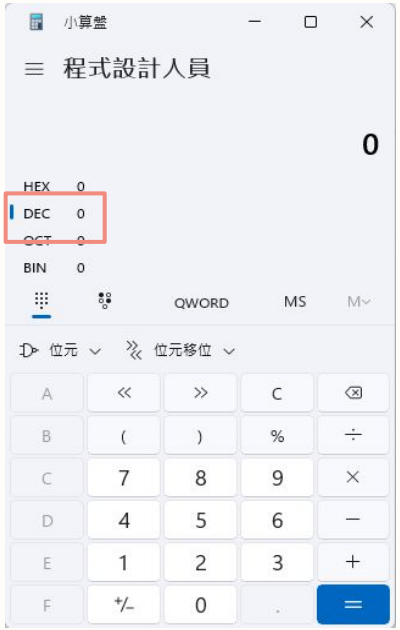
1's complement of the result : 1000 1110 1110 1001 -> <u>8e e9</u> (final result)

How to calculate decimal number manually

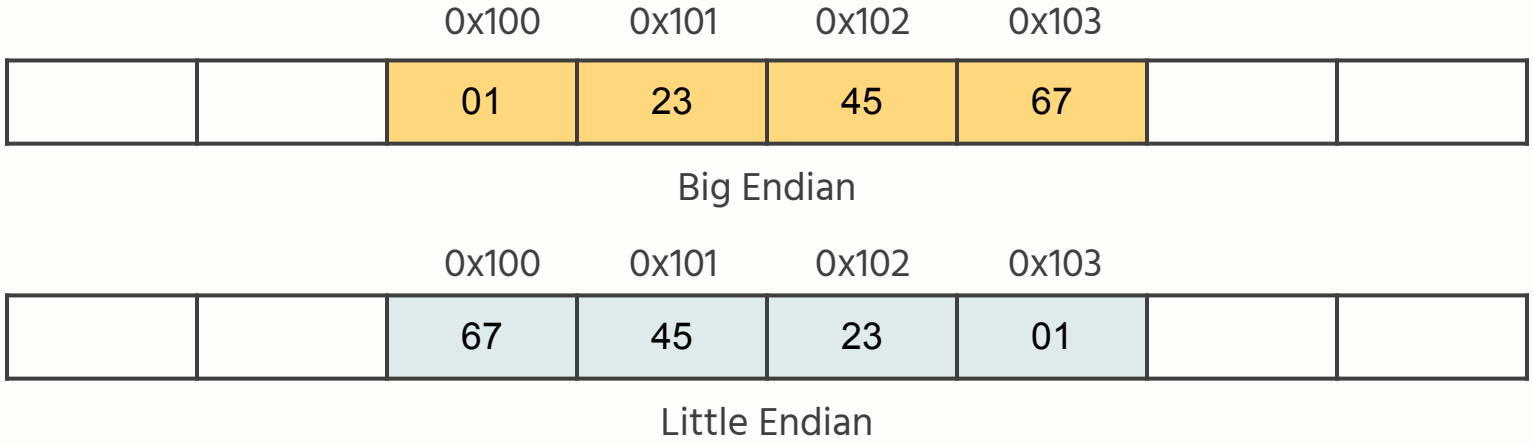
- Open Calculator app in your Windows system
- Change the mode from “standard” to “Programmer” and select “DEC”







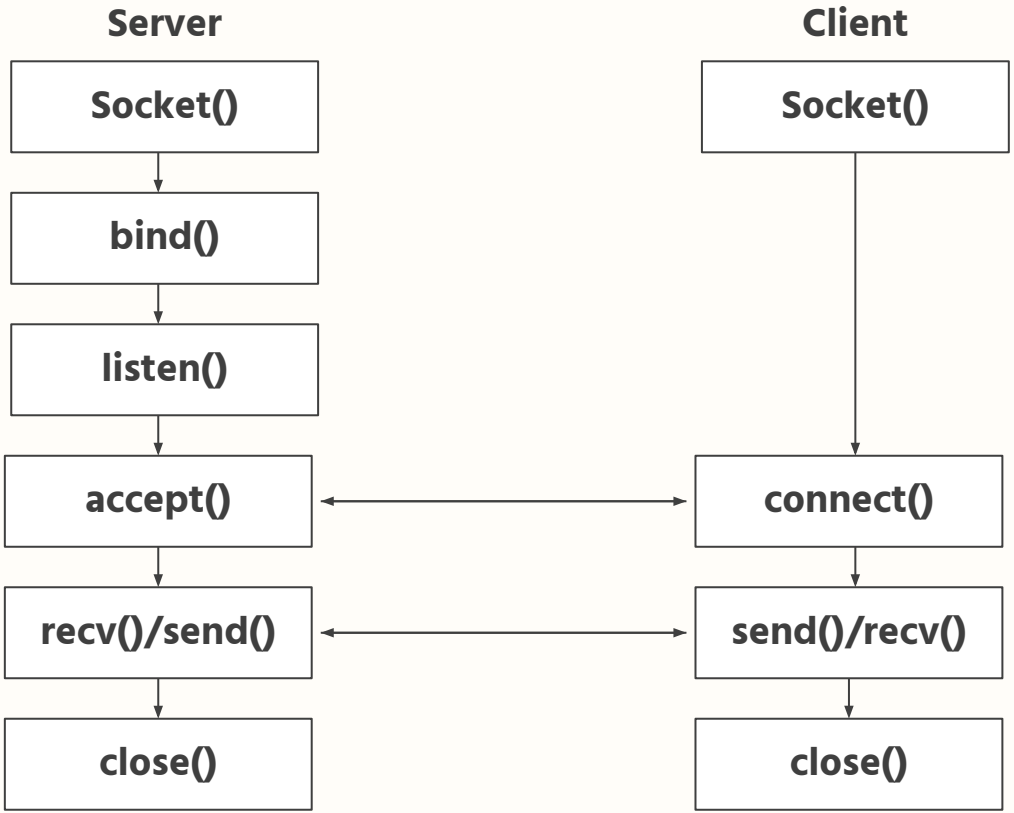
Little Endian & Big Endian



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

TCP working flow

TCP flow


Server
Client
Socket()
Socket()
bind()
listen()
accept()
connect()
recv()/send()
send()/recv()
close()
close()

TCP flow

Server

Socket()

```
//Create TCP socket
int socket_fd = socket(PF_INET , SOCK_STREAM , 0);
```

bind()

```
//Bind socket to the address.
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

listen()

```
//Listening the socket.
int listen(int sockfd, int backlog);
```

accept()

```
//Accept the connect request.
int accept(int sockfd, struct sockaddr *restrict addr, socklen_t *restrict addrlen);
```

recv()/send()

```
//Send message to client
ssize_t send(int sockfd, const void *buf, size_t len, int flags);
```

close()

```
//Close the connection
int close(int fd);
```

TCP flow

Client

Socket()

```
//Create TCP socket
int socket_fd = socket(PF_INET , SOCK_STREAM , 0);
```

connect()

```
//Accept the connect request.
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

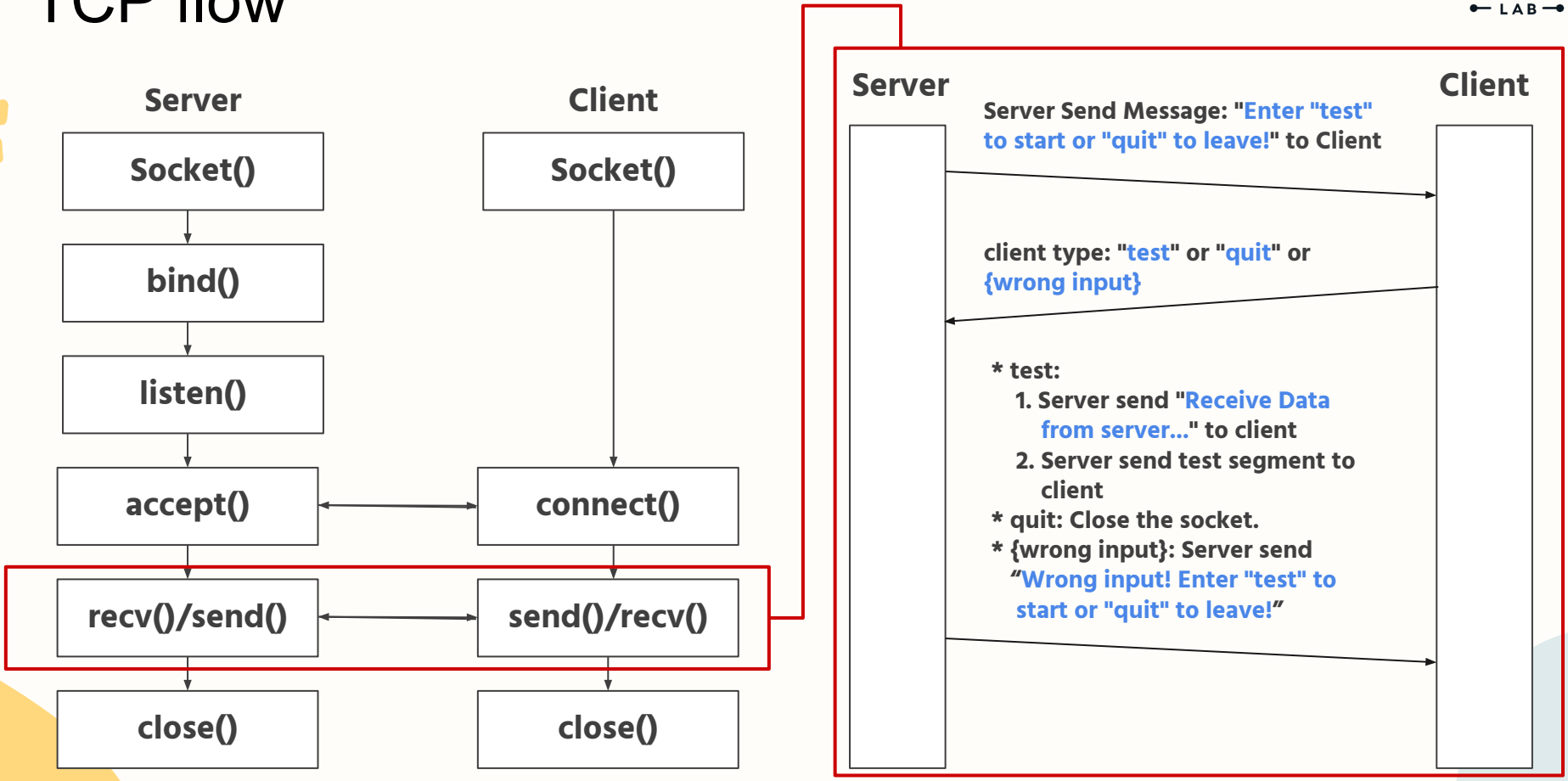
send()/recv()

```
//Receive message from server
ssize_t recv(int sockfd, void *buf, size_t len, int flags);
```

close()

```
//Close the connection
int close(int fd);
```

TCP flow


Server
Client
Socket()
Socket()
bind()
listen()
accept()
connect()
recv()/send()
send()/recv()
close()
close()
Server
Client
Server Send Message: "Enter "test" to start or "quit" to leave!" to Client
client type: "test" or "quit" or {wrong input}
* test:
1. Server send "Receive Data from server..." to client
2. Server send test segment to client
* quit: Close the socket.
* {wrong input}: Server send "Wrong input! Enter "test" to start or "quit" to leave!"

Assignment

Assignment

- In lab 2, you will each get a zip file containing :
  1. client.c
  2. server.c
  3. header.h
  4. header.c
  5. makefile
  6. sample_input.txt

Assignment

TCP Header Value

- We only focus on the specific packet **TCP ack packet without options**

<table>
<tr><td colspan="8">Source Port</td><td>Destination Port</td></tr>
<tr><td colspan="9">Sequence Number</td></tr>
<tr><td colspan="9">Acknowledgement Number</td></tr>
<tr><td>Header Length
0101</td><td>Reserved Bits
000000</td><td>U
0</td><td>A
1</td><td>P
0</td><td>R
0</td><td>S
0</td><td>F
0</td><td>Window Size
(Advertisement Window)</td></tr>
<tr><td colspan="8">Checksum</td><td>Urgent Data Pointer
0000 0000 0000 0000</td></tr>
</table>

Makefile

- After you finish writing your code, compile your code using the command “`make`” under the lab2 folder
  - `wupei@wupeideMacBook-Pro lab2 % make`

- `make` #run Makefile to compile
- `./server {sample_input.txt}` #run the server
  - E.g. `./server sample_input.txt` : Use the sample_input to run the server
- `./client` #run the client
- CTRL+C #exit server

Sample_input format

1. Source IP
2. Destination IP
3. Source port
4. Destination port
5. Seq number
6. Ack number
7. Window size

sample_input.txt

```
53.34.79.160
199.151.130.68
42961
35171
2121043209
1167626588
16324
```

Assignment

- Implementation (70%)
  - Task 1 (60%)
    - Connect server and client with TCP socket and successfully send a message.
  - Task 2 (15%)
    - Create TCP header (without checksum) using l4info.
  - Task 3 (15%)
    - Complete the header (with checksum).
  - Task 4 (10%)
    - Screenshot the TCP Packet contains “Server {Your_student_ID}” *(Put your result in the report file)*
- Report (30%)
  - Questions are in the next slide

Implementation (70%)

- Task 1: Connect server and client with TCP socket and successfully send a message.(60%)
  - Server side

```
root@DESKTOP-EFQ5EAV:~/2024/lab2# ./server sample_input.txt
New connection
```

  - Client side

```
root@DESKTOP-EFQ5EAV:~/2024/lab2# ./client
Hi, I'm server 112062571...
```

Implementation (70%)

- Task 2: Create TCP header (without checksum) using l4info.(15%)
  - Client side: “test”

```
server: Enter "test" to start or "quit" to leave!
test
Receive Data from server...
Receive information:

            Layer 3 information:
            Source IP: 53.34.79.160 , Destinatio  Ip: 199.151.130.68
            Protocol: 6 (TCP)

            Layer 4 information:
            Source port: 42961 , Destination port: 35171
            Seq number: 2121043209 , Ack number: 1167626588
            Header length: 5 (bytes) , FLAG: 0x10 (ACK)
            Window size: 16324
```

Implementation (70%)

- Task 2: Create TCP header (without checksum) using l4info.(15%)
  - Client side: “quit”

```
server: Enter "test" to start or "quit" to leave!
quit
close Socket
```

Implementation (70%)

- Task 2: Create TCP header (without checksum) using l4info.(15%)
  - Client side: {wrong_input}
    - If wrong input, it should be re-entered!

```
server: Enter "test" to start or "quit" to leave!
112062571
server: Wrong input! Enter "test" to start or "quit" to leave!
# can type again!
```

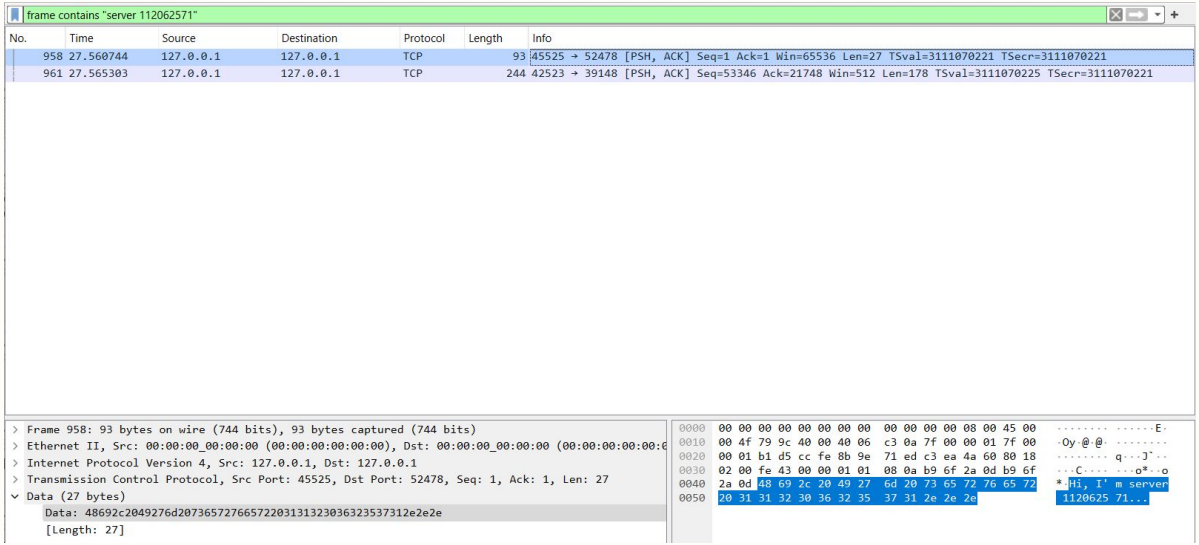
Implementation (70%)

- Task 3: Complete the header (with checksum) (15%).
  - Client side:

```
The header is:
A7 D1 89 63 7E 6C 8D 09 45 98 91 5C 50 10 3F C4 8D D2 00 00
```

Implementation (70%)

- Task 4: Screenshot the TCP Packet contains “Server {Your_student_ID}” (10%).



Report (30%)

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 pseudo header ? (10pts)
6. For the code below, what's difference between client_fd andsocket_fd ? (10pts)

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

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

Name the report file as: report.pdf

Requirement

- Compress all files into one.
  - client.c
  - server.c
  - header.h
  - header.c
  - makefile
  - Sample_input.txt
  - report.pdf
- Name the file Lab2_studentID.zip
  - (e.g. Lab2_112062571.zip)
- Upload to eeclass before **May 9th**.

- We will run your file
  - `Make`
  - `./server test_input.txt`
  - `./client`

Penalty

- **Plagiarism will get 0 point**
- Late submission before **May 16th** only get **70%** of the original score.
- Late submission after **May 16th** will **not** be accepted