Assignment 2

TCP Socket Programming

Prof. Ai-Chun Pang
TA / Chun-Yu Lee, Wan-Chu Hsu

Assignment 2 Announcement

Specification (1/12)

- In this assignment, you need to implement a simple Network Storage System, with following functions:
  - Client can list all files in the folder of the server.
  - Client can upload files to the server.
  - Client can download files from the server.
  - Client can watch a “.mpg” video (streaming) on the server.

Specification (2/12)

- For video steaming, you don’t need to send audio. You can just send frames in RAW format.
- After upload and download, you need to ensure the files are identical between source and destination.
- In this assignment, all the transmission should be implemented in C/C++ and by TCP socket.

Specification – Commands (3/12)

- Server is required to support multiple connections. That is, there can be more than one client connecting to the server simultaneously.

- After building up of a connection, a client can enter the commands below:

```
$ ls
$ put <filename1> <filename2> … <filenameN>
$ get <filename1> <filename2> … <filenameN>
$ play <mpg_videofile>
```

Specification – ls (4/12)

- For the command **ls**, you should list all the files in the server's folder with the filename saparated by a newline.

```
$ ls
file1
file2
file3
...
video.mpg
```

Specification – put (5/12)

- For the command **put**, client should upload the files sepcified in the arguments to the server.

```
$ put  file1  file2  file99  file3
puting file1……
puting file2……
The file99 doesn’t exist.
puting file3……
```

- There won’t be multiple clients putting same file at the same time.
- About the error command

Specification – get (6/12)

- For the command **get**, client should download the files sepcified in the arguments from the server.

```
$ get  file1  file2  file99  file3
getting file1……
getting file2……
The file99 doesn't exist.
getting file3……
```

- There might be multiple clients getting same file at the same time. But a client will not get the file which is being put by another client.
- About the error command

Specification – play (7/12)

- For the command **play**, you should play the .mpg video files sepcified in the arguments.

```
$ play video1.mpg
playing the video...
```

- There might be multiple clients playing same video at the same time.
- A client will not play the video which is being put.
- When pressing **esc**, the client should terminate the video and be able to keep sending another commands.
- About the error command

Specification – Compilation (8/12)

- You are required to write a Makefile for compilation.

```
$ make client          // To compile client code
$ make server          // To compile server code
```

- After compilation, there should be 2 binary files named “**client**” and “server”

Specification – Usage (9/12)

- When we launch the server, we will type

```
$ ./server [port]                    // [port] will be determined
```

- After launching, a folder named **<student_id>_server_folder** for the server should be create.

```
|-- server
`-- b08902999_server_folder
    |-- file1
    |-- file2
```

- No matter what happend, the server should NOT be terminated.

Specification – Usage (10/12)

- When we launch the client, we will type

```
$ ./client  [client_id]  [ip]:[port]  // [client_id] is a integer
                                      // [ip] is the ip address of the server
                                      // [port] is determined by the command above
```

- After launching, a folder named **<student_id>_<client_id>_client_folder** for the client should be create.
- The client might be terminated at any time .

If we launched the server and 2 clients with id 1 and 5:

```
|-- server
|-- client
|-- b08902999_1_client_folder
|    |-- file10
|    |-- file11
|-- b08902999_5_client_folder
|    |-- file20
|    |-- file21
`-- b08902999_server_folder
     |-- file1
     |-- file2
```

Specification – Error Handling(11/12)

- All of your outputs should be printed to standard output (stdout) and end with a newline.
- If the command doesn't exist or the command format is wrong, please print out "Command not found." or "Command format error." on the client.
- If the file doesn't exist while putting, getting or playing a file, please print out "The '<filename>' doesn't exist." on the client.
- If the video file is not a ".mpg" file while playing a video file, please print out "The '<videofile>' is not a mpg file."

Specification (12/12)

- Server should output the file descriptor of new connection when a client connects to the server
- Client should be able to send another command after a command is finished.
- The multiple connections should be implemented with `pthread.h` or `select()`, while the video player should be implemented with OpenCV.
- The implementation must be in **C or C++**.
- File size will be less than 2GB

Specification – Multiple Connections

- There can be more than one client connecting to the server and send commands to the server simultaneously.
- Commands from different clients should be excuted concurrently. That is, command from one client cannot be blocked by other commands from other clients.

Grading Policy (1/3)

- This assignment accounts for 15% of the total score.
- **Command Sending (10%)**
  - The client and server handle commands correctly (5%)
  - The client prints out responses correctly (5%)
- **Basic File Transferring (30%)**
  - There would be 5 test cases (6% * 5)
  - (You will get 0 point in a testcase if the transfer of a testcase is terminated or halts before finished, or the files are not identical between source and destination after the transfer.)

Grading Policy (2/3)

- **Video Streaming (20%)**
  - Correctly playing a resolution-fixed video (960*540) (6%)
  - Correctly playing a resolution-unknown video (7%)
    (Client has no idea about the resolution of the video)
  - Playing video while others transmitting files (7%)
- **Multiple Connections (20%)**
  - There will be 5 test cases
  - Use **<pthread.h>** to achieve this function (basic) (10%)
  - Use **select()** to achieve this function (advance) (20%)
  - You just need to choose one of above to implement.

Grading Policy (3/3)

- **Report (5% * 4)**
  - Draw a flowchart of the file transferring and explains how it works in detail.
  - Draw a flowchart of the video streaming and explains how it works in detail.
  - What is SIGPIPE? It is possible that SIGPIPE is sent to your process? If so, how do you handle it?
  - Is blocking I/O equal to synchronized I/O? Please give some examples to explain it.

Submission

- Requirements
  - Your report must be a **.pdf** file and named “**report.pdf**”.
  - Please put all the file into a folder named **<studentID>_hw2** and compress the folder as a **.zip** file. Submit your **.zip** file to **NTU COOL**.
    - e.g. B08902999_hw2.zip
  - The penalty for wrong format is 10 points.
  - No plagiarism is allowed. A plagiarist will be graded zero.
- Deadline
  - Due Date : 23:59:59, November 23th, 2021
  - Penalty for late submission is **20 points per day**.

Sample Codes

- We will provide sample codes for your reference
  - *server.c* - Default port number is 8787
  - *client.c* - Default IP address is 127.0.0.1, port is 8787
  - *pthread.c*
  - *openCV.cpp* - It will play video.mpg
  - *video.mpg*
  - *Makefile*
  - *install_opencv.sh* (for Ubuntu 20.04)

Environment Setup

Environment

- We provide a VirtualBox VM for you to run our example code.
- If you choose to setup the environment on your OS rather than using our VM, here is information of our environment
  - Ubuntu 20.04 x64
  - OpenCV 3.4.15 (will be also required in later assignments)
- You can install OpenCV 3.4.15 by following the instruction here or via our shell script:

```
$ sh inatall_opencv.sh
```

VirtualBox Setup

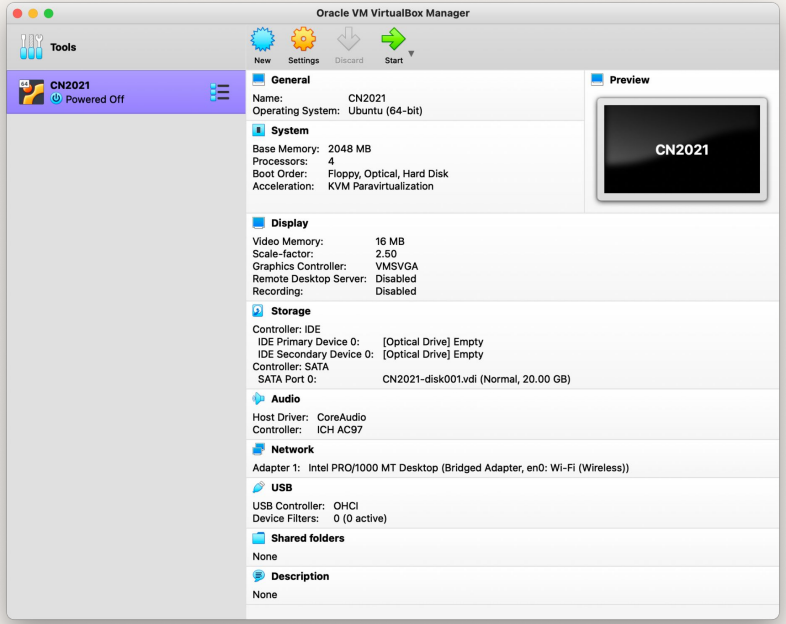
- Download the VM from
  - our Google Drive
- Install Virtualbox (natively installed on the computers of Lab R204).

VirtualBox Setup

- click “**Import**” to import the “**CN2021.ova**”

VirtualBox Setup

- Choose “**CN2021**” and then start the machine.



Auxiliary Libraries

OpenCV

- An open source library for computer vision.
- ***Mat*** is an image container to load an image so that you can easy to do image processing, recognition, etc.
- In this assignment, we use this library to get frames from videos on server, and show frames on client.
- We will provide a sample code and a .mpg file for you.
- To compile code with OpenCV,

```
$ g++ <file name> -o <output name> \ `pkg-config --cflags --libs opencv`
```

Pthread

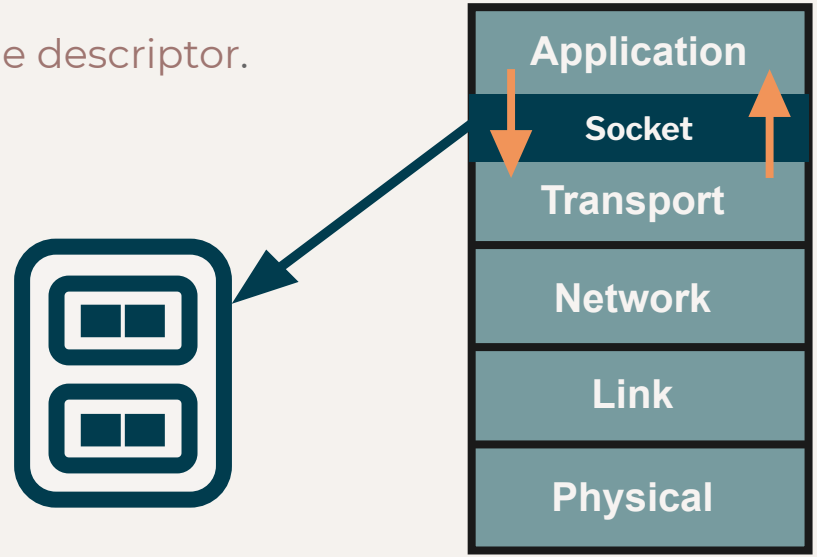
- Pthread, i.e., POSIX Thread, is used to implement multi-thread parallelization in POSIX environment.
- You can use pthread to achieve multiple connections.
- You don’t need to deal with synchronization issues, i.e., in our testcases, it won’t put a file with the same file name.
- We will provide a sample code for you.
- To compile with Pthread,

```
$ g++ <file name> -o <output name> -pthread
```

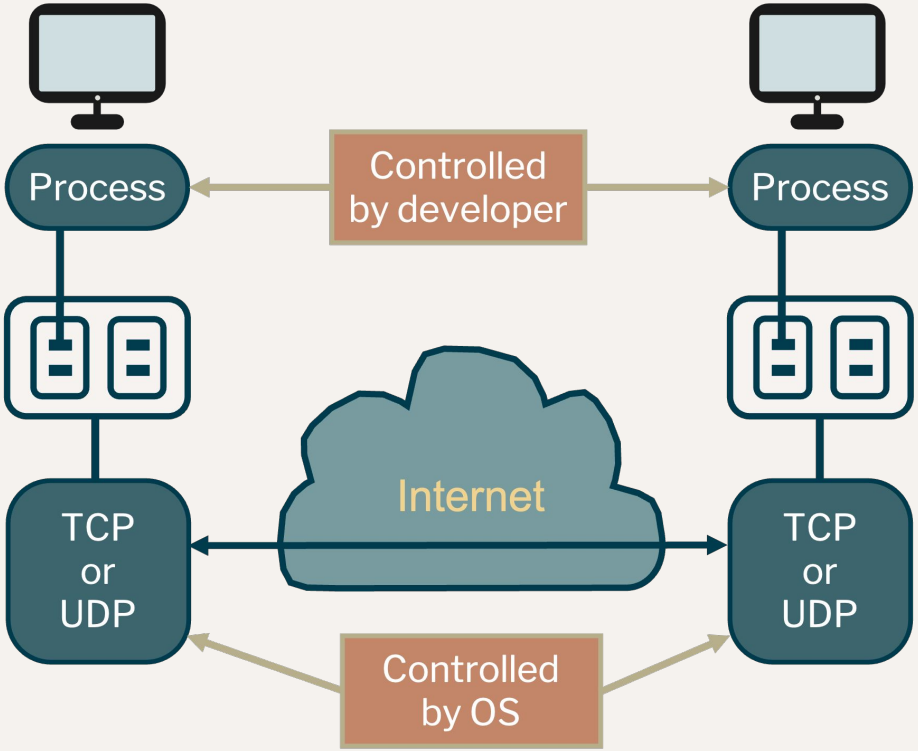
Socket Programming Tutorial

What is Socket?

- Socket is the API for the TCP/IP protocol stack.
- Provides communication between the Application layer and Transport layer.
- Make internet communication like a file descriptor.
    - read() and write()
- We will provide a sample code for you.



What is Socket?



File Descriptors

- When we open an existing file or create a new file, the kernel return a file descriptor to the process.
- If we want to read or write a file, we identify the file with the file descriptor.

Interger vaule	Name	<unistd.h> symbolic constant	<stdio.h> file stream
0	Standard input	STDIN_FILENO	stdin
1	Standard output	STDOUT_FILENO	stdout
2	Standard error	STDERR_FILENO	stderr

```
FILE *fp = fopen("this.txt","w");
fprintf(fp, "Happy Coding.");
fclose(fp);
```

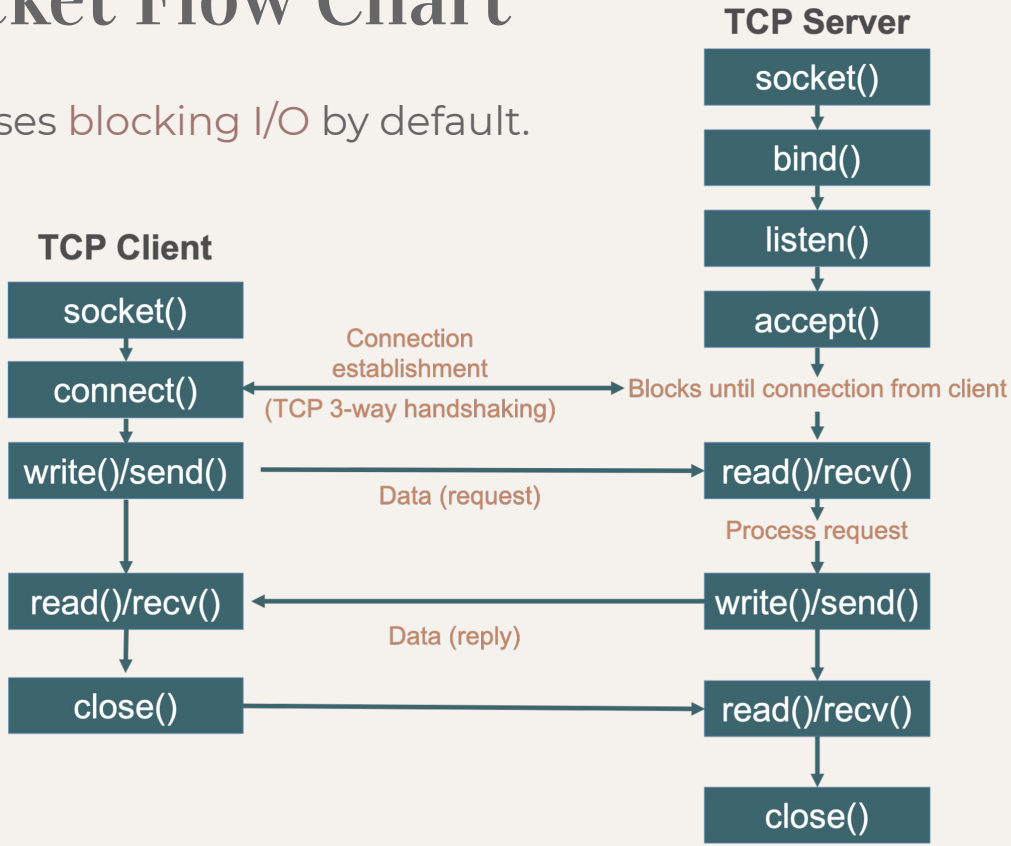
```
printf("This is Computer Networking.\n");
fprintf(stdout," This is Computer Networking.\n");
```

TCP Service

- **TCP (Transmission Control Protocol)**
  - **Connection-oriented**
  - **Reliable transport**
  - Flow control
  - Congestion control
- What is Socket-Address?
  - **IP address + Port number**
  - IP address: To find out the machine (Network Layer)
  - Port number: To find out the process (Transport Layer)

TCP Socket Flow Chart

- Socket uses blocking I/O by default.



socket()

- Create the endpoint for connection.

```
#include <sys/types.h, sys/socket.h>
int socket (int domain, int type, int protocol);
```

- domain
  - AF_UNIX/AF_LOCAL: communication between 2 processes on a host, so that they can share a file system.
  - AF_INET, AF_INET6: communication between processes on different hosts through the Internet. AF_INET is for IPv4, where AF_INET6 is for IPv6.

socket()

- Create the endpoint for connection.

```
#include <sys/types.h, sys/socket.h>
int socket (int domain, int type, int protocol);
```

- type
  - SOCK_STREAM: sequential and connection-oriented (TCP)
  - SOCK_DGRAM: datagram (UDP)
- protocol: defined in /etc/protocols, usually set to 0
- return: socket file descriptor (an integer)

bind()

- Bind the address to the socket.

```
#include <sys/types.h, sys/socket.h>
int bind (int sockfd, struct sockaddr *addr, socklen_t len);
```

- sockfd: specifies the socket file descriptor to bind.
- sockaddr
  - specifies the socket address to be associated with the sockfd
  - You can use “struct sockaddr_in*” defined in <netinet/in.h>, and then cast it into “struct sockaddr*”
- len: specifies the size of sockaddr (=sizeof(struct sockaddr))

bind()

```
struct sockaddr_in {
        short sin_family;                   // address family. EX:AF_INET
        unsigned short sin_port;            // port number for network
        struct in_addr sin_addr;            // IP address for network
        unsigned char sin_zero[8];          // pad to sizeof(struct sockaddr)
}
```

listen()

- Listen for connections on a socket.

```
#include <sys/types.h, sys/socket.h>
int listen (int sockfd, int backlog);      // returns 0 if it's success; -1 otherwise
```

- sockfd: specifies the socket file descriptor to listen.
- backlog: specifies the number of users allowed in queue.
    - Linux typically add 3 to the number specified, while other OS has different implementations.

accept()

- Accept the connection on a socket.

```
#include <sys/types.h, sys/socket.h>
int accept (int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

- After accepting the connection, it creates a new file descriptor for the client. The original socket is not affected.
- **sockfd**: specifies the socket being listened.
- **addrlen**: pointer to the length of sockaddr.
- Blocking until a user **connect()** call is received.
- Format is the same as **socket()**.

connect()

- Connect to the socket from client to server.

```
#include <sys/types.h, sys/socket.h>
int connect (int sockfd, struct sockaddr *addr, socklen_t len);
```

- Format is the same as bind().

close()

- Close the file descriptor.

```
#include <unistd.h>
int close (int sockfd);        // returns 0 if it's success; -1 otherwise
```

read()/recv()

- Read data from socket file descriptor.

```
#include <unistd.h>
ssize_t read (int fd, void *buf, size_t count);
ssize_t recv (int fd, void *buf, size_t len, int flag);
```

- fd: specifies the socket file descriptor to read data from
- buf: specifies the buffer to contain the received data
- count/len: specifies the size to receive
- flag: (**read()** has no this parameter.) It's about some details like blocking/nonblocking.

read()/recv()

- Read data from socket file descriptor.

```
#include <unistd.h>
ssize_t read (int fd, void *buf, size_t count);
ssize_t recv (int fd, void *buf, size_t len, int flag);
```

- Reading data from file may be
  - Successful, return the number of bytes received
  - EOF (end of file) (i.e., return = 0)
  - Failed, errno is set to indicate the error
- It may be blocked. (block I/O)

write()/send()

- Write data to socket file descriptor.

```
#include <unistd.h>
ssize_t write (int fd, const void *buf, size_t count);
ssize_t send (int fd, const void *buf, size_t len, int flags);
```

- fd: specifies the socket file descriptor to send data to
- buf: specifies the buffer to contain the data to be transmitted
- count/len: specifies the size to send
- flag: (write() has no this parameter.) It's about some details.

write()/send()

- Write data to socket file descriptor.

```
#include <unistd.h>
ssize_t write (int fd, const void *buf, size_t count);
ssize_t send (int fd, const void *buf, size_t len, int flags);
```

- Writing data to file may be
  - Successful, return the number of bytes written
  - Failed, errno is set to indicate the error
- It may be blocked. (block I/O)

Useful Functions

- Address and port numbers are stored as integers.

  Different machines implements different endian.

  They may communicate with each other on the network.
- IP address is usually hard to remember.

  We need to translate hostname to IP address.

Useful Functions

- Converting IP address and port number
  - **htonl()**: for IP address (host -> network)
  - **ntohl()**: for IP address (network -> host)
  - **htons()**: for port number (host -> network)
  - **ntohs()**: for port number (network -> host)
- Translate a hostname to IP address

```
#include <netdb.h>
struct hostent *gethostbyname (const char *name);
```

Supplementary Materials

How to Simulate a Bad Network

- Some bugs may occur when the network is not good.
- We can simulate a bad network to test our program in advance.
- Linux
    - Get the network interfaces list in your machine

```
$ ifconfig
```

    - Then, to make your personal internet slow, you can enter

```
$ sudo tc qdisc add dev <interface> root netem delay 500ms
```

      In this way, the delay will be 500ms.

```
$ sudo tc qdisc del dev <interface> root netem
```

How to Trace Kernel

- Sometimes, the service may terminate without any error message.
- It happens usually because of some kernel issues.
- To trace the interactions between your code and kernel, we can use strace.
- To run your program with strace, you can enter

```
$ strace ./<name>
```

Behavior of send() and recv()

- In fact, a **send()** doesn't imply all the data in the buffer are sent.
- In addition, a **send()** doesn't imply all the data in the buffer are sent in a packet, and even 2 **send()** don't imply they are in different packets.
- You are required to design a protocol so that each receive has the same size as the send in respect to it.

select()

- **select()** provides you to supervise multiple sockets, telling you which is able to read or write, etc.
- With **select()**, it is possible to achieve Asynchronous Blocking I/O.
- If you want to implement this assignment with **select()**, please refer to this website.

select()

- Monitor whether there is at least one fd available.

```
#include <unistd.h>
int select (int nfds, fd_set*, readfds, fd_set* writefds, fd_set* exceptfds, struct timeval* timeout);
```

- nfds: specifies number of file descriptors to monitor.
- readfds: specifies the pointer to read file descriptor list.
- writefds: specifies the pointer to write file descriptor list.
- exceptfds: specifies the pointer to error file descriptor list.
- timeout: deadline for **select()**.

select()

```
void FD_SET (int fd, fd_set *set);
void FD_CLR (int fd, fd_set *set);
int FD_ISSET (int fd, fd_set *set);  // return: 1 if it's available, else: 0
void FD_ZERO (fd_set *set);
```

- FD_SET(): Add the file descriptor into the set.
- FD_CLR(): Remove the file descriptor from the set.
- FD_ISSET(): Check if the file descriptor is available.
- FD_ZERO(): Clear the set.

Reference

- Beej's Guide to Network Programming (中文)
- Beej's Guide to Network Programming (English)
- Linux manual page

Contact us if you have any problem. •ω•)ค

TA Email: ntu.cnta@gmail.com