2019 Computer Networks, Ai-Chun Pang

Ai-Chun Pang / Instructor

林偉哲 陳令原 / T.A.s

Assignment 2- TCP Socket Programming

Assignment 2 Announcement

Specification (1/4)

- In this assignment, you need to implement a simple Network Storage System, with these functions:
  - Client can **watch a ".mpg" video (streaming)** on the server
  - Client can **upload files** to server
  - Client can **download files** from server
  - Client can **list what files are in the folder of server**
- 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 by **the socket of TCP**.

Specification (2/4)

- You are required to write a Makefile for compilation. Thus, the command should be

```
$ make client          // for client code
$ make server          // for serve code
```

- After compilation, there should be 2 binary files named “client” and “server”
- When we launch the server app, we will enter

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

- When we launch the client app, we will enter

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

- After the launch, client or server is required to create their own folders while any folder doesn’t exist.

Specification (3/4)

- Server is required to support multiple connections. That is, there can be more than 1 client connecting to the server simultaneously.
- After building up of a connection, a user can enter the commands below on the client,

```
$ ls                    // Then, the client's will print out what
                           files is in the server's folder.

$ put <filename>        // Then, the client will upload the file
                           with the <filename> to the server's
                           folder

$ get <filename>        // Then, the client will download the
                           file with the <filename> to the
                           client's folder

$ play <videofile>      // Then, the client will play the
                           <videofile> from the server to the
                           client
```

Specification (4/4)

- 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 or getting 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."
- Client should be able to send another command after a command is finished.
- The multiple connections should be implemented with pthread.h, while the video player should be implemented with OpenCV.
- The implementation must be in **C or C++**.

Grading Policy (1/3)

- **This assignment accounts for 12% of the total score.**
- **Command Sending** **(10%)**
  - The client sends commands correctly (5%)
  - The client prints out responses correctly (5%)
- **Video Streaming** **(25%)**
  - Correctly playing a 720p Video (1280*720) (15%)
  - Correctly playing a resolution-unknown video (10%)

  (That is, client has no idea about the resolution of the video before requesting a video to play.)
- **File Transferring** **(25%)**
  - There would be 6 files with different sizes (5%*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)

- **Multiple Connections** **(20%)**
  - 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.
- **Report** **(20%)**
  - Draw a flowchart of the video streaming and explains how it works in detail. (5%)
  - Draw a flowchart of the file transferring and explains how it works in detail. (5%)
  - What is SIGPIPE? It is possible to happen to your code? If so, how do you handle it? (5%)
  - Is blocking I/O equal to synchronized I/O? Please give me some examples to explain it. (5%)

Grading Policy (3/3)

- **Submission**
  - Your report format must be in **“.pdf”** format and named “report.pdf”, or else **you will get zero point** in the part.
  - Please put all the files **into a folder** named **hw2_<student id>,** and compress the folder as a **.zip** file, and then submit the **.zip** file to here. The password is **<student id>** (alphabet is in uppercase).
  - If we **cannot compile or execute your code**, you will **have a chance to demo your results** in your own environment.
  - The penalty for **wrong format** is **10 points**.
  - **No plagiarism is allowed. A plagiarist will be graded zero.**
- **Deadline**
  - Soft Deadline: 23:59:59, November 12th, 2019
  - Hard Deadline: 23:59:59, November 19th, 2019 (get *0.9 points)
  - Penalty for late submission after hard deadline is “20 points per day” (after *0.9)

Environment Setup

Environment

- We provide a VirtualBox environment for you to run our binary code and you can run Wireshark on this environment.

- If you would like to setup the environment on your OS rather than our virtual machine, here is information of our environment
  - Ubuntu 16.04 x64
  - OpenCV 3.3.1 (will be also required in later Assignments)

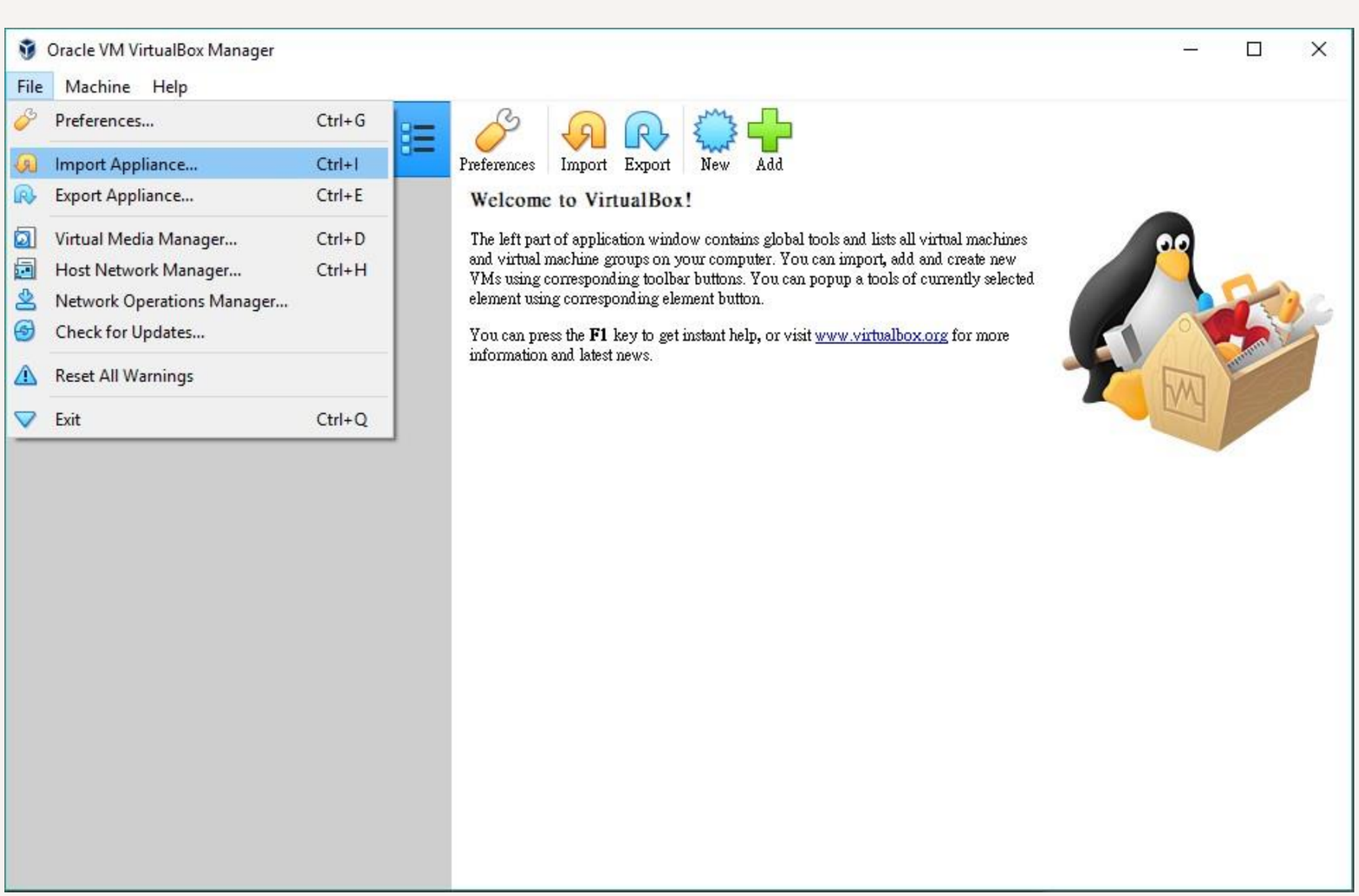
- You can install OpenCV 3.3.1 by following the instruction here.

VirtualBox Setup

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

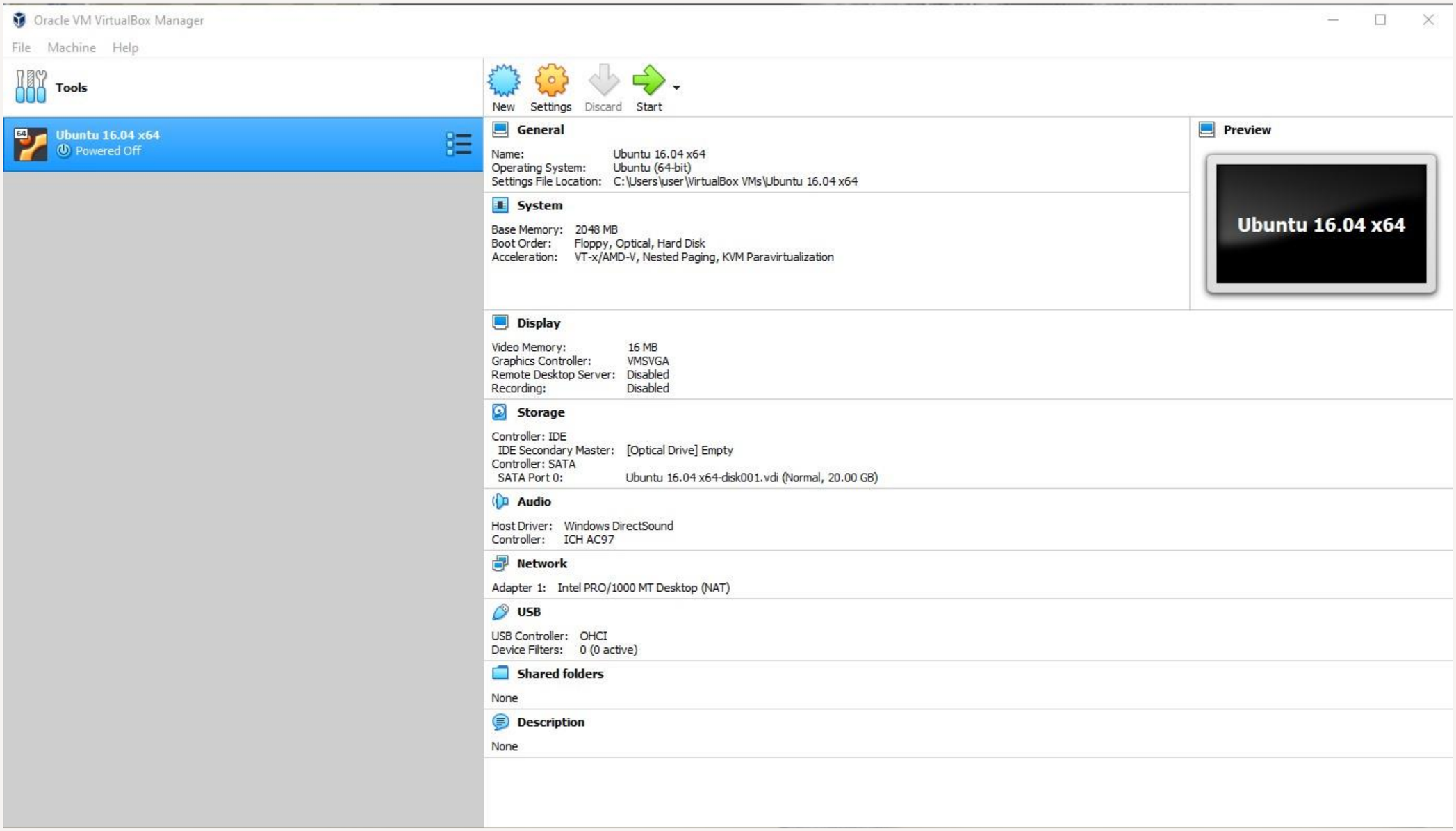
VirtualBox Setup

- Go to “File” and click “Import Appliance” to import the “CN-Ubuntu_16.04_x64.ova”



VirtualBox Setup

- Choose “Ubuntu 16.04 x64” and then start the machine.



Auxiliary Libraries

OpenCV

- Is an opensource 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.
- An example code is here. Here is an .mpg video file if you need.
- To compile code with OpenCV,

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

Pthread

- Pthead, 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 video with the same file name.
- An example code is here.
- To compile with Pthread,

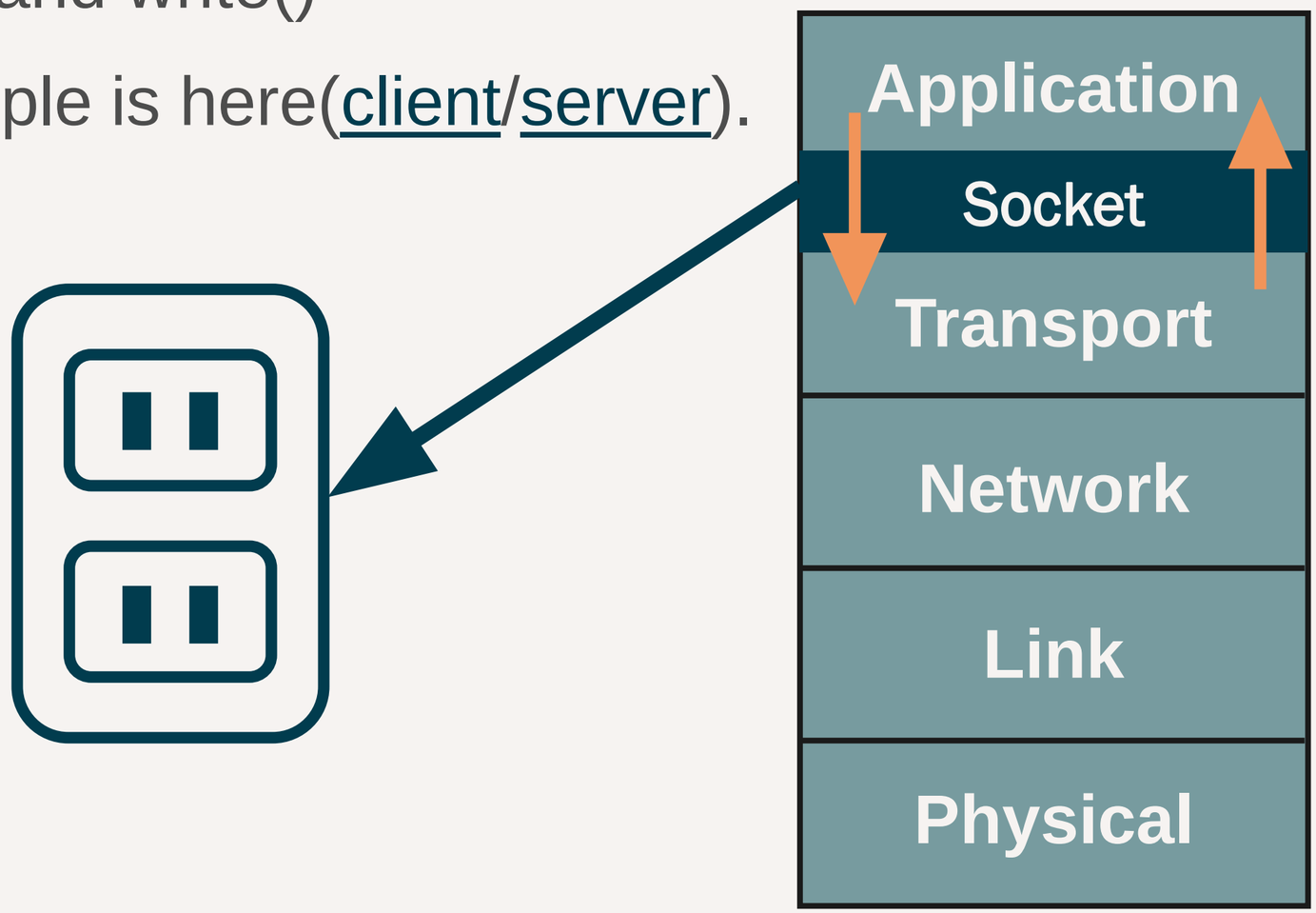
```
$ g++ <file name> -o <output name> -pthread
```

2019 Computer Networks, Ai-Chun Pang

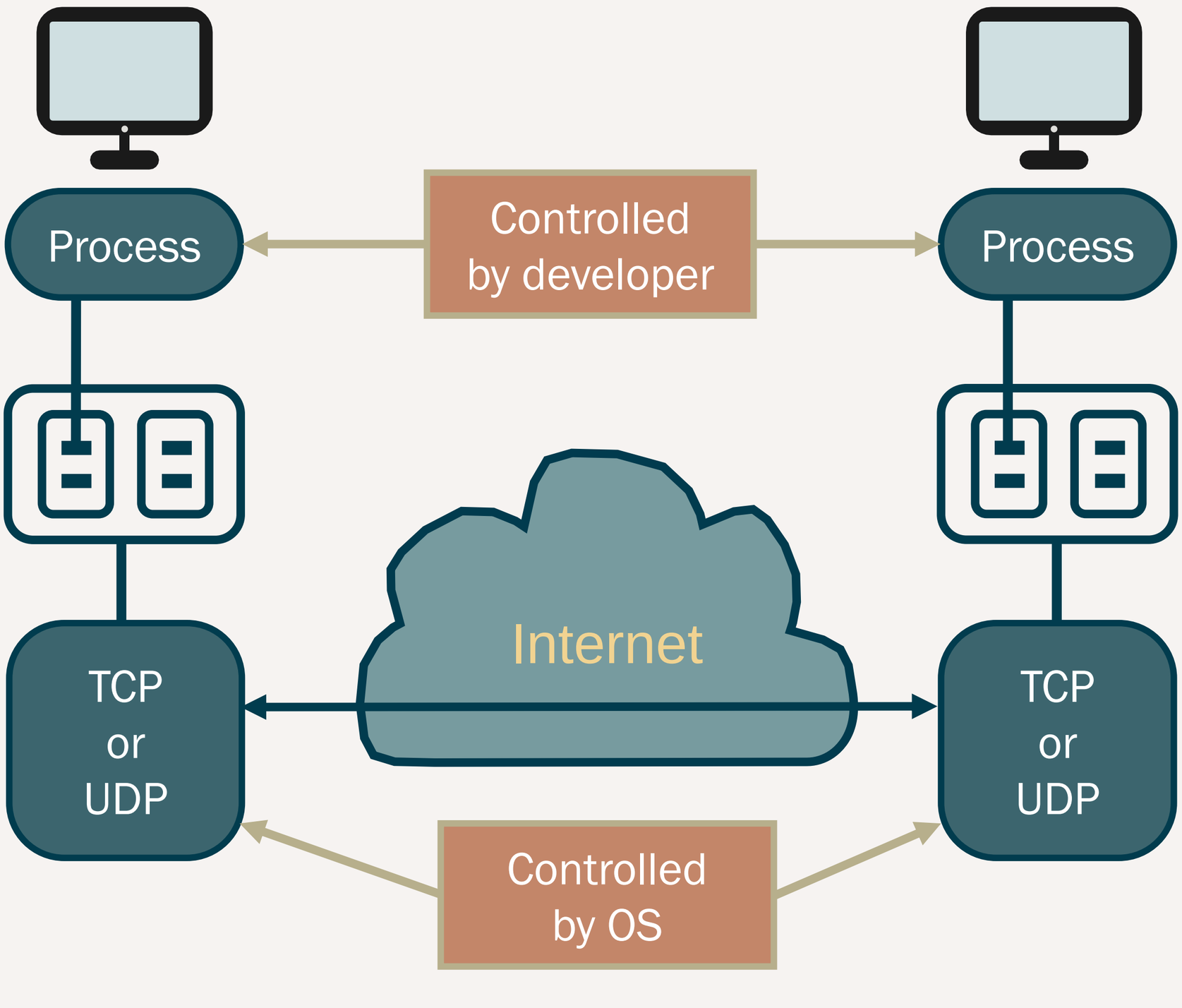
Socket Programming Tutorial

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()
- An example is here(client/server).



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”);
```

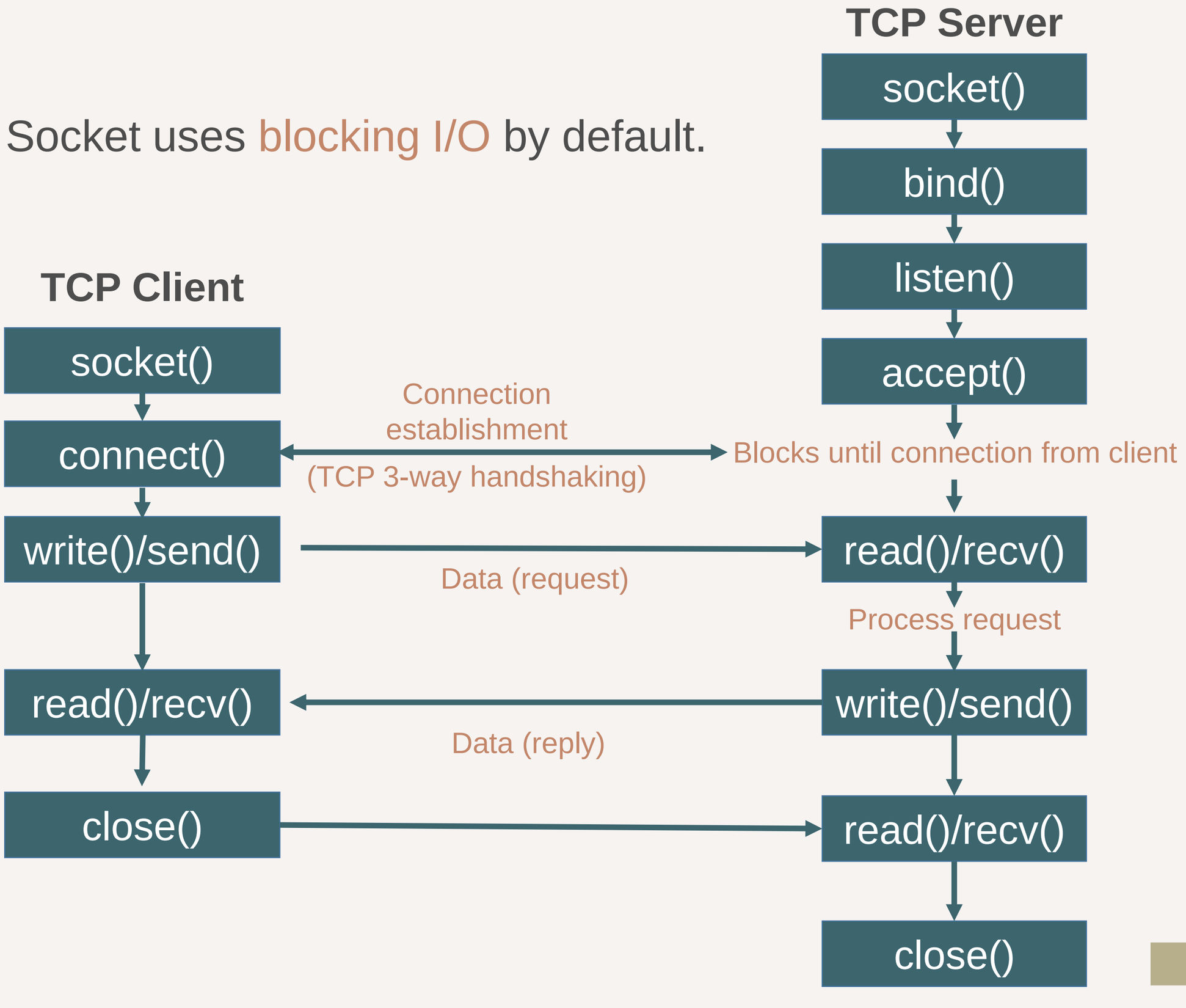
TCP Service

- **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);
                                  // returns a file descriptor if it's success; -1 otherwise
```

- 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
- 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);
                                        // returns 0 if it's success; -1 otherwise
```

- sockfd: specifies the socket file descriptor to bind.
- sockaddr
  - specifies the socket address to be associated with 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))

```
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. After accepting the connection, it creates a new file descriptor for the client. The original socket is not affected.

```
#include <sys/types.h, sys/socket.h>
int accept (int sockfd, struct sockaddr *addr, socklen_t *addrlen);
                                    // returns a file descriptor if it's success; -1 otherwise
```

- 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);
                                        // returns 0 if it's success; -1 otherwise
```

- 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  recv (int fd, void *buf, size_t len, int flag);
                        //return : number of bytes read if it's success, -1 otherwise
```

- fd: specifies the socket file descriptor to read data from
- buf: specifies the buffer to contain the received data
- count: specifies the size to receive
- flag: (read() has no this parameter.) It's about some details like blocking/nonblocking.
- Reading data from file may be
  - Success
  - EOF (end of file) (i.e., return = 0)
- It may be blocked. (block I/O)

write()/send()

- Write data to socket file descriptor

```
#include <unistd.h>
ssize_t  send (int fd, void *buf, size_t len, int flag);
                    //return : number of bytes written if it's success, -1 otherwise
```

- fd: specifies the socket file descriptor to send data to
- buf: specifies the buffer to contain the data to transmit
- count: specifies the size to send
- flag: (write() has no this parameter.) It's about some details.
- Reading data from file may be
  - Success
  - EOF (end of file) (i.e., return = 0)
  - Error (i.e., return<0)
- 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);
                    // return: host environment if it's success, NULL otherwise
```

Supplementary Materials

How to Simulate a Bad Network

- Some bugs may occur when the network is not good.
- In case of bad network, we should 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.

  - You can turn it off with

```
$ 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.
- For more details about I/O, you can go to this website. (Report 4)
- 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);     //return : 1 if it's success, -1 otherwise
```

- 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)