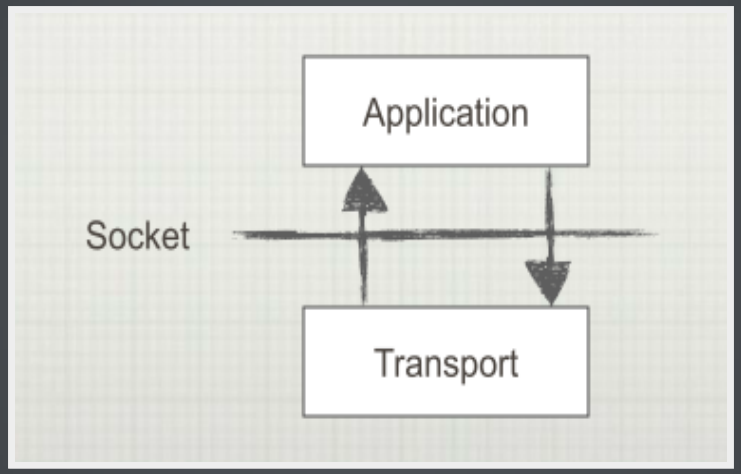
SOCKET PROGRAMMING

FOR COMPUTER NETWORK COURSE

ntu.cnta@gmail.com

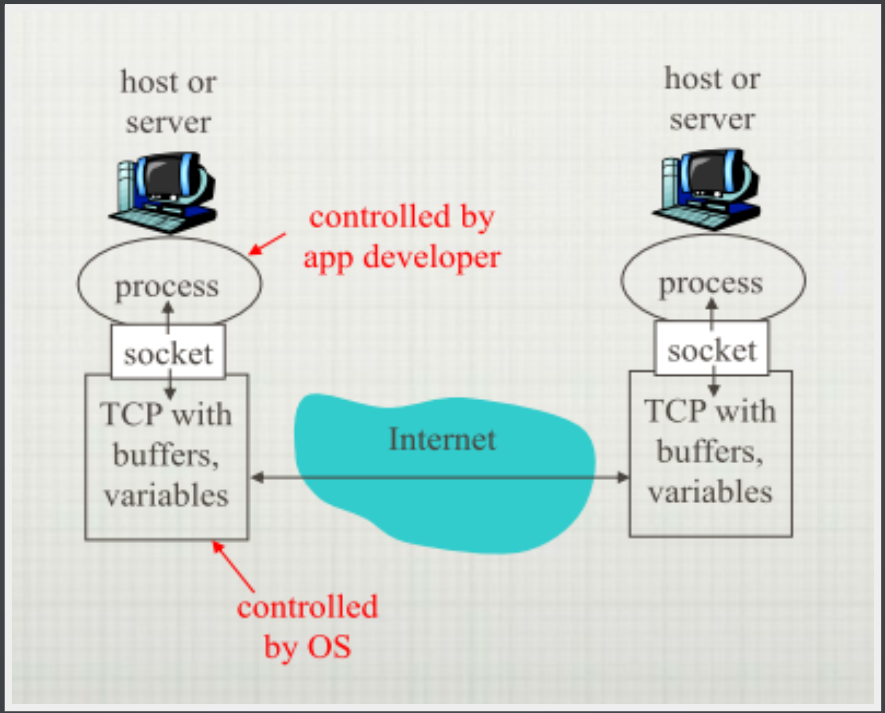
INTRODUCTION

- Socket is the API for the TCP/IP protocol stack.
- Provides communication between the Application layer and Transport layer



INTRODUCTION

Process sends/receives messages to/from its socket.



PROGRAMMER'S VIEW

- Socket is a file descriptor
- Socket allows application to...
  Send data to the network
  Receive data transmitted from other hosts

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

TCP SERVICE

- Connection-oriented
- Reliable transport
- Flow control
- Congestion control

TCP SERVICE

What is socket-address?

IP address + port number

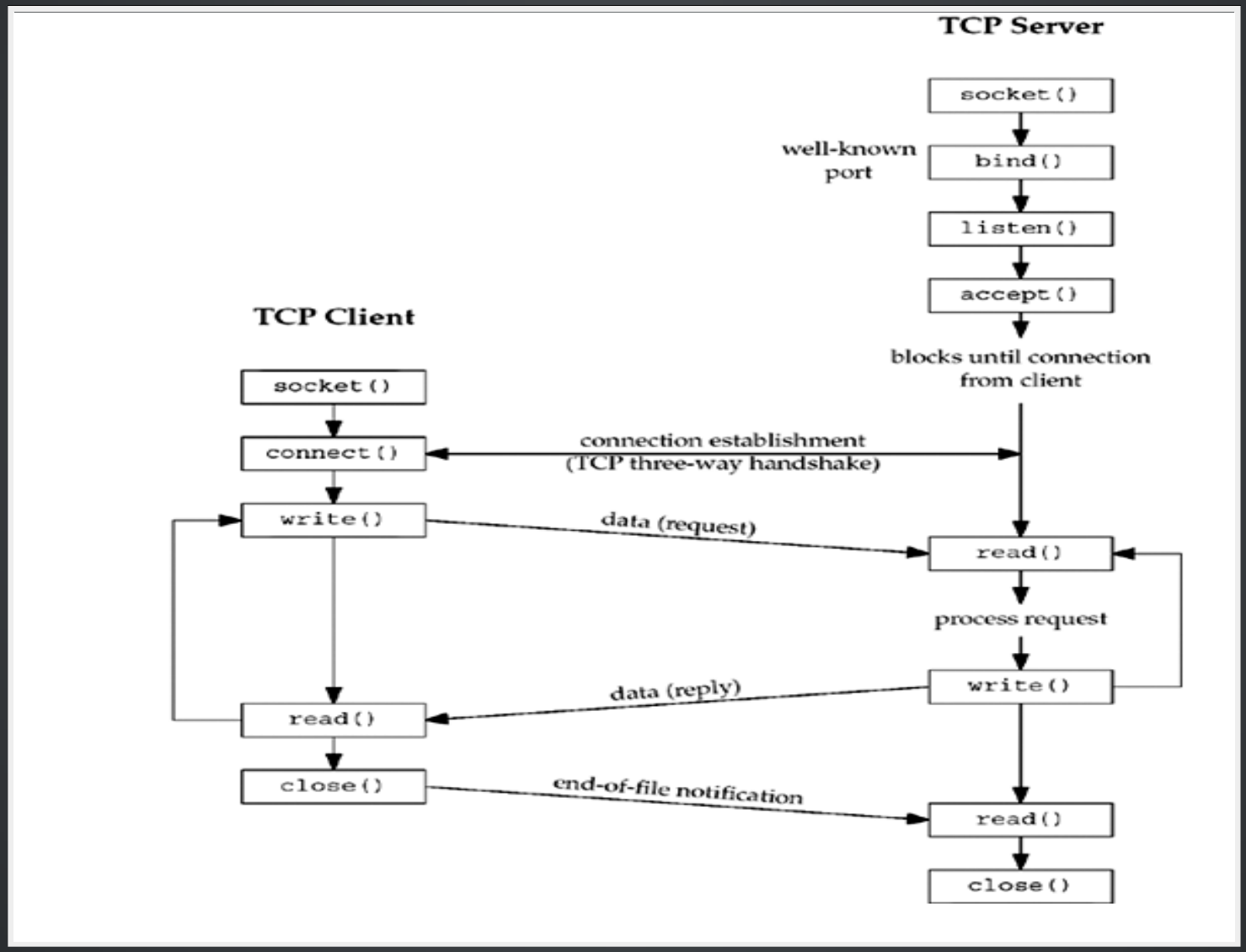
- IP address : *address the machine*
- port number : *address the process*

PORT NUMBER

port number for well-known process

Service	Port Number	Description
FTP	21	FILE Transfer Protocol
ssh	22	Secure Shell
telnet	23	
SMTP	25	Simple mail Transfer Protocol
DNS	53	Domain Name Server
HTTP	80	Hyper Text Transfer Protocol
POP3	110	Post Office Protocol
RPC	135	Remote Procedure Call

TCP CONNECTION



USEFUL FUNCTION

CONNECTION ESTABLISHMENT

SOCKET

create the endpoint for the connection

```
#include <sys/types.h, sys/socket.h>
int socket(int domain, int type, int protocol)
                              //return  file descriptor on OK; -1 otherwise
```

SOCKET

- domain : *indicate whether IPV4 or IPV6 is used*
  IPV4 : AF_INET
- type : *indicate communication type*
  TCP : SOCK_STREAM
  UDP : SOCK_DGRAM
- protocol : *defined in /etc/protocols, usually set to 0.*
- returned : *socket file descriptor*

BIND

bind the address to the socket

```
#include <sys/types.h, sys/socket.h>
int bind(int sockfd, struct sockaddr *addr, socklen_t len)
                                        //return 0 on OK; -1 otherwise
```

BIND

- sockfd : *specifies the socket file descriptor*
- addr : *specifies the socket address(`struct sockaddr_in`) to be associated with sockfd*
- len : *specifies the size of addr.(`sizeof(struct sockaddr)`)*

STRUCT SOCKADDR_IN

the structure for the socket address

```
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)
                                        //return 0 on OK; -1 otherwise
```

LISTEN

- sockfd : *specifies the socket descriptor*
- backlog : *specifies the number of users allowed in queue*
  Linux typically add 3 to the number specified
  Other OS has different implementations

ACCEPT

Accept the connection on a socket .
After accepting the connection , it create the new fd for the cilent . The original socket is not affacted.

```
#include <sys/types.h, sys/socket.h>
int accept(int sockfd, struct sockaddr *sockaddr, socklen_t *len)
                                  //return file descriptor on OK; -1 otherwise
```

ACCEPT

- Blocking until a user `connect()` call is received
- sockfd : *specifies the socket file descriptor*
- addr : *specifies the cilent address*
- return : *a new socket descriptor (for client)*

CONNECT

connect to the socket

```
#include <sys/types.h, sys/socket.h>
int connect(int sockfd, struct sockaddr *addr, socklen_t len)
                                        //return 0 in OK , -1 otherwise
```

CLOSE

```
#include <unistd.h>
int close(int sockfd)
                                        //return 0 on OK; -1 otrwise
```

USEFUL FUNCTION

COMMUNICATION I/O

READ

read data from the socket file descriptor

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t count)
                        //return : number of bytes read on OK, -1 otherwise
```

READ

- fd : *specifies the socket file descriptor to read data from*
- buf : *specifies the buffer to contain the received data*
- count : *specifies the size of buf.(`sizeof(buf)`)*
- sometimes **reading may block** . reading data from file may ...
  1 . succeeds
  2 . get ends of file(return = 0)

WRITE

write data to the socket file descriptor

```
#include <unistd.h>
ssize_t write(int fd, struct void *buf, size_t count)
                        //return : number of bytes sent on OK, -1 otherwise
```

WRITE

- fd : *specifies the socket file descriptor to send data to*
- buf : *specifies the buffer to contain the data to transmit*
- count : *specifies the size of buf.(`strLen(buf)`)*
- sometimes **writing may block** . writing data to file may ...
  1 . succeeds (return > 0)
  2 . connection closed (return = 0)
  3 . error (return < 0)

USEFUL FUNCTIONS

CONVERSION

BYTE ORDERING

- Address and port numbers are stored as integers
  Different machines implements different endian
  They may communicate with each other on the network
- IP address are usually hard to remember
  We need to translate IP address to hostname

CONVERSION

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)*

CONVERSION

Converting IP address between network and human-readble format

- `inet_ntop()` : *network -> presentation*
- `inet_pton()` : *presentation -> network*

GETHOSTBYNAME

Translate a hostname to IP address

name specifies the hostname

```
#include <netdb.h>
struct hostent *gethostbyname(const char *name)
                          //return : host environment on OK,null otherwise
```

EXAMPLE

HELLO WORLD SERVER/CLIENT

HELLO WORLD SERVER

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <sys/types.h>
int main(int argc, char *argv[])
{
    char buffer[50];
    int listenfd, connfd;
    socklen_t length;
    struct sockaddr_in serverAddress, clientAddress;
```

HELLO WORLD SERVER

```
    listenfd = socket(AF_INET, SOCK_STREAM, 0);
    bzero(&serverAddress, sizeof(serverAddress));
    serverAddress.sin_family = AF_INET;
    serverAddress.sin_port = htons(5000);
    serverAddress.sin_addr.s_addr = htonl(INADDR_ANY);
    bind(listenfd, (struct sockaddr*)&serverAddress, sizeof(serverAddress));
    listen(listenfd, 1);
    while(1)
    {
        length = sizeof(clientAddress);
        connfd = accept(listenfd, (struct sockaddr*)&clientAddress, &length);
        write(connfd, "Hello World!\n", 13);
        close(connfd);
    }
}
```

HELLO WORLD CLIENT

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <sys/types.h>
int main(int argc, char *argv[])
{
    char buffer[50];
    int sockfd;
    struct sockaddr_in serverAddress;
```

HELLO WORLD CLIENT

```
    sockfd = socket(AF_INET, SOCK_STREAM, 0);

    bzero(&serverAddress, sizeof(serverAddress));
    serverAddress.sin_family = AF_INET;
    serverAddress.sin_port = htons(5000);
    inet_pton(AF_INET, argv[1], &serverAddress.sin_addr);

    connect(sockfd, (struct sockaddr *)&serverAddress, sizeof(serverAddress));

    bzero(buffer, sizeof(buffer));
    read(sockfd, buffer, sizeof(buffer));
    printf("%s", buffer);
    close(sockfd);
}
```

END

Please email to TA if you have any question.