Network Programming: Ch.1 Introduction

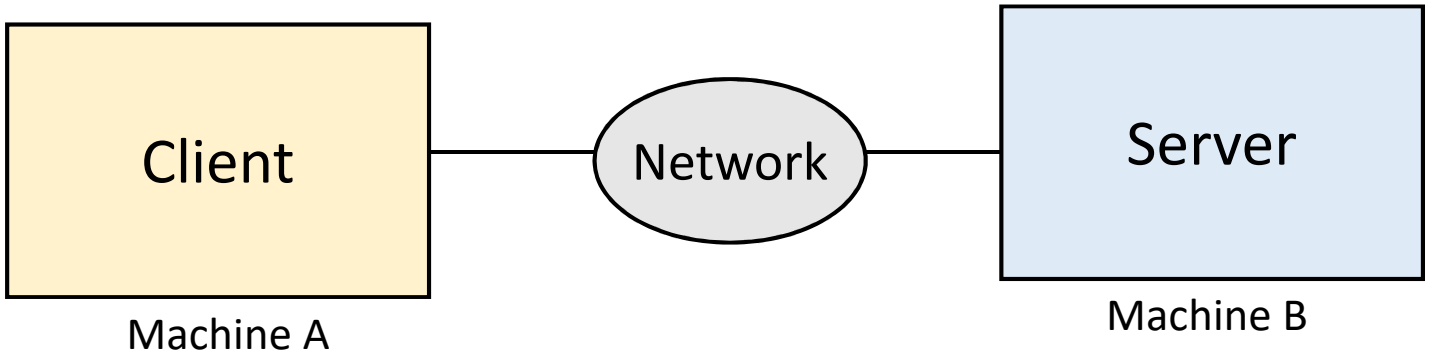
Li-Hsing Yen
NYCU

Ver. 1.0.1

Introduction

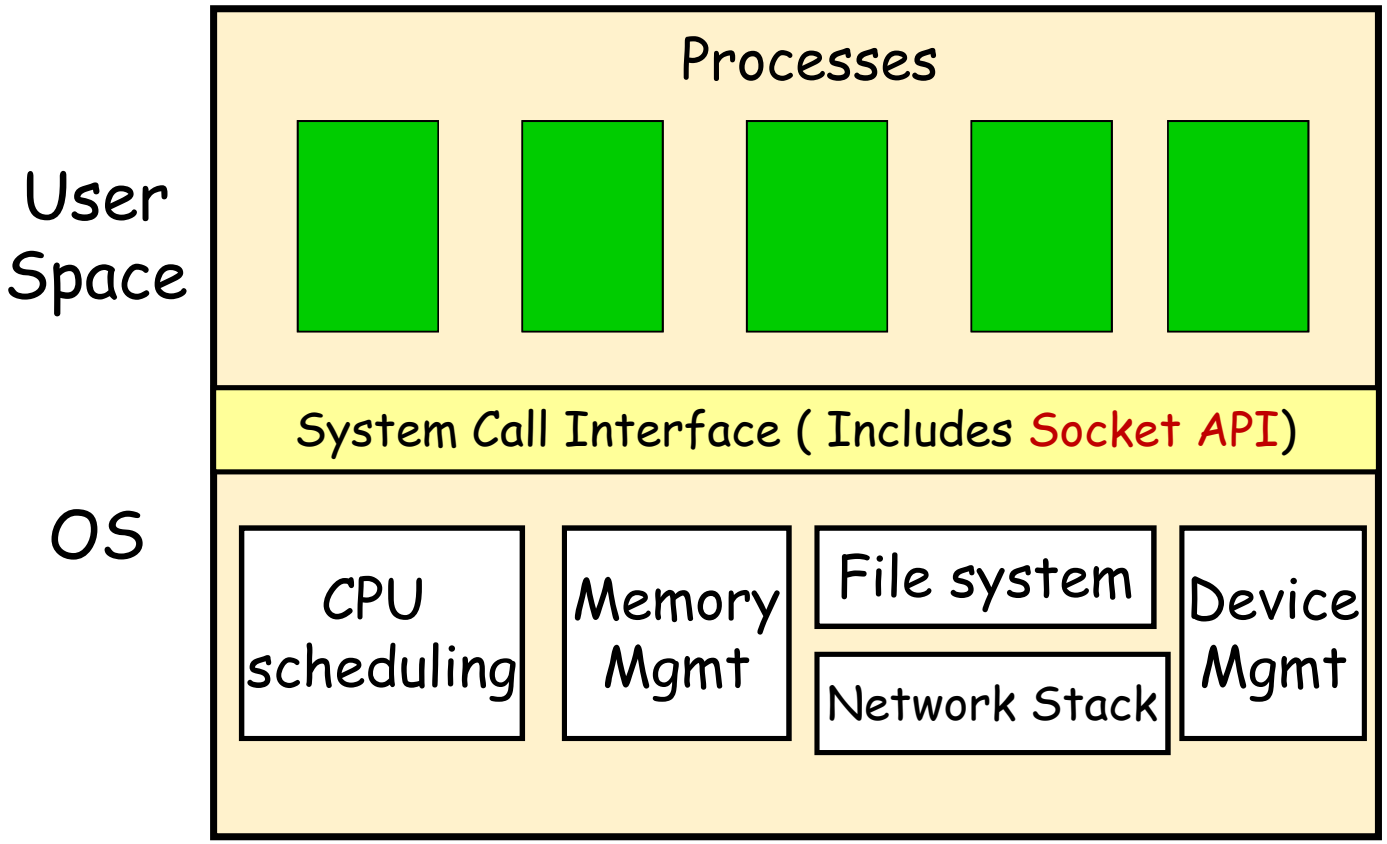
- A Simple Daytime Client
- A Simple Daytime Server
- Error handling: wrapper functions
- Types of Networking APIs
- BSD networking history
- Discover details of your local network

Client-Server Model

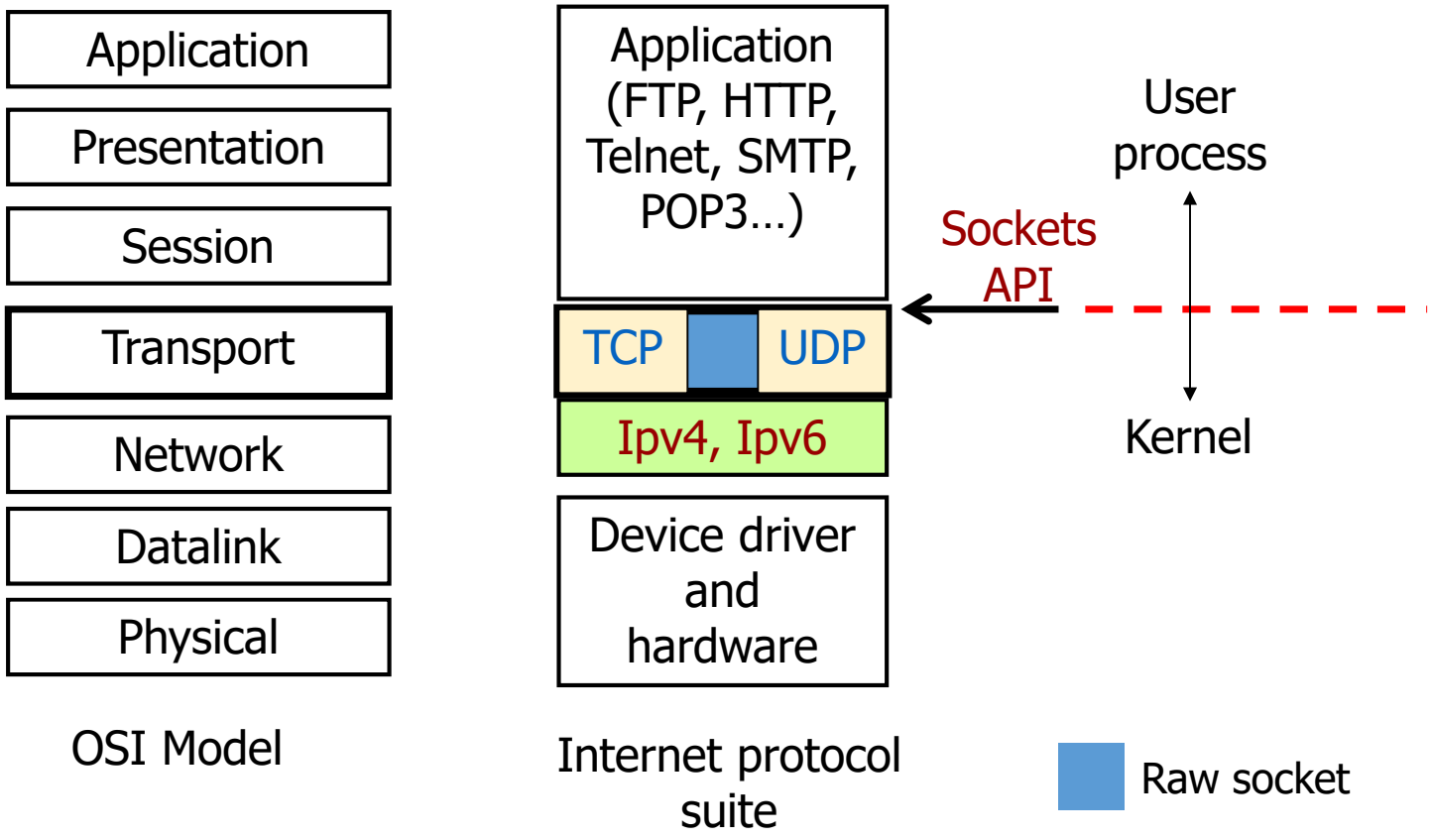


- Web browser and server
- FTP client and server
- Telnet client and server

Socket API Location in OS

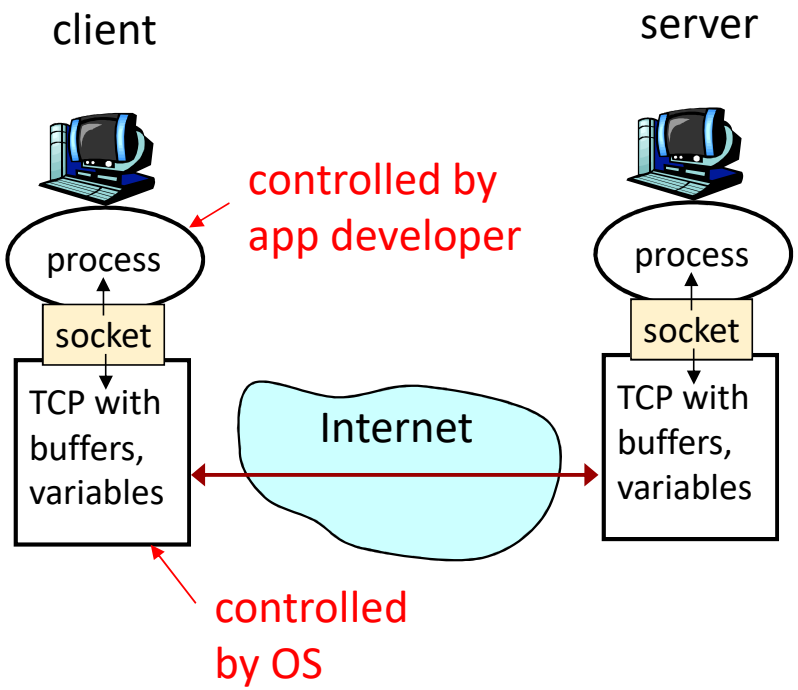


OSI and TCP/IP Model



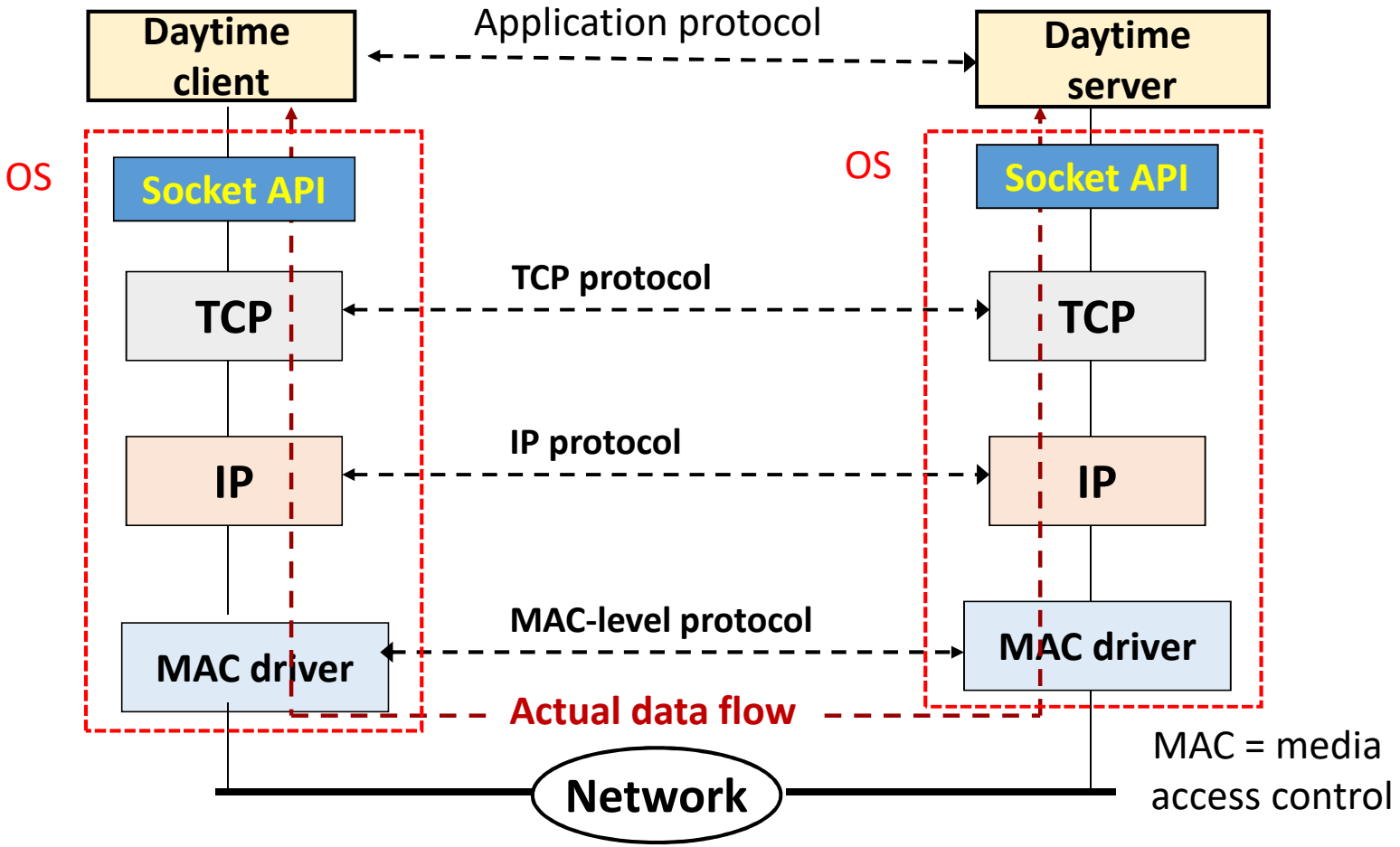
Sockets

- process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves message out door
  - transport infrastructure brings message to the door at receiving process

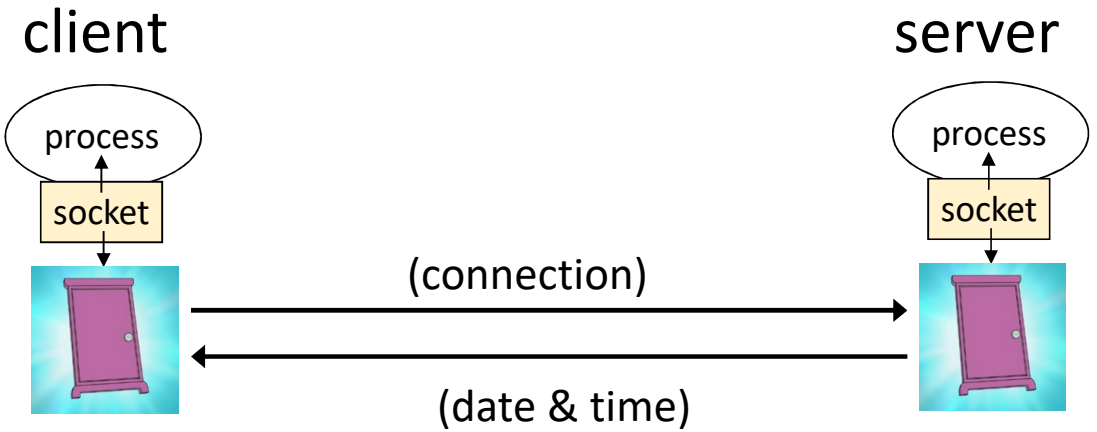


Socket API: (1) choice of transport protocol; (2) ability to fix many parameters.

A Daytime client/server using socket API



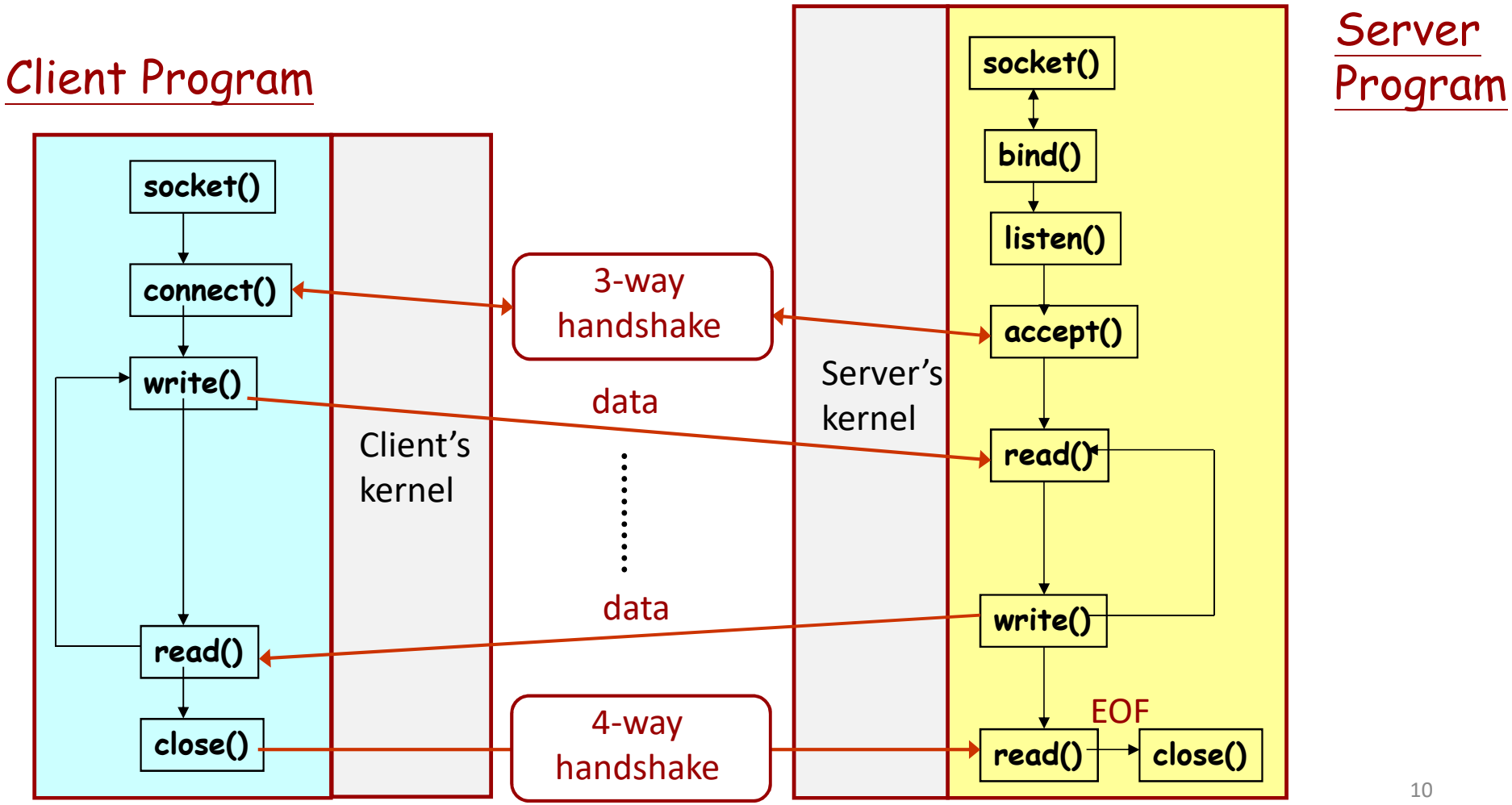
How Daytime Client and Server Processes Interact?



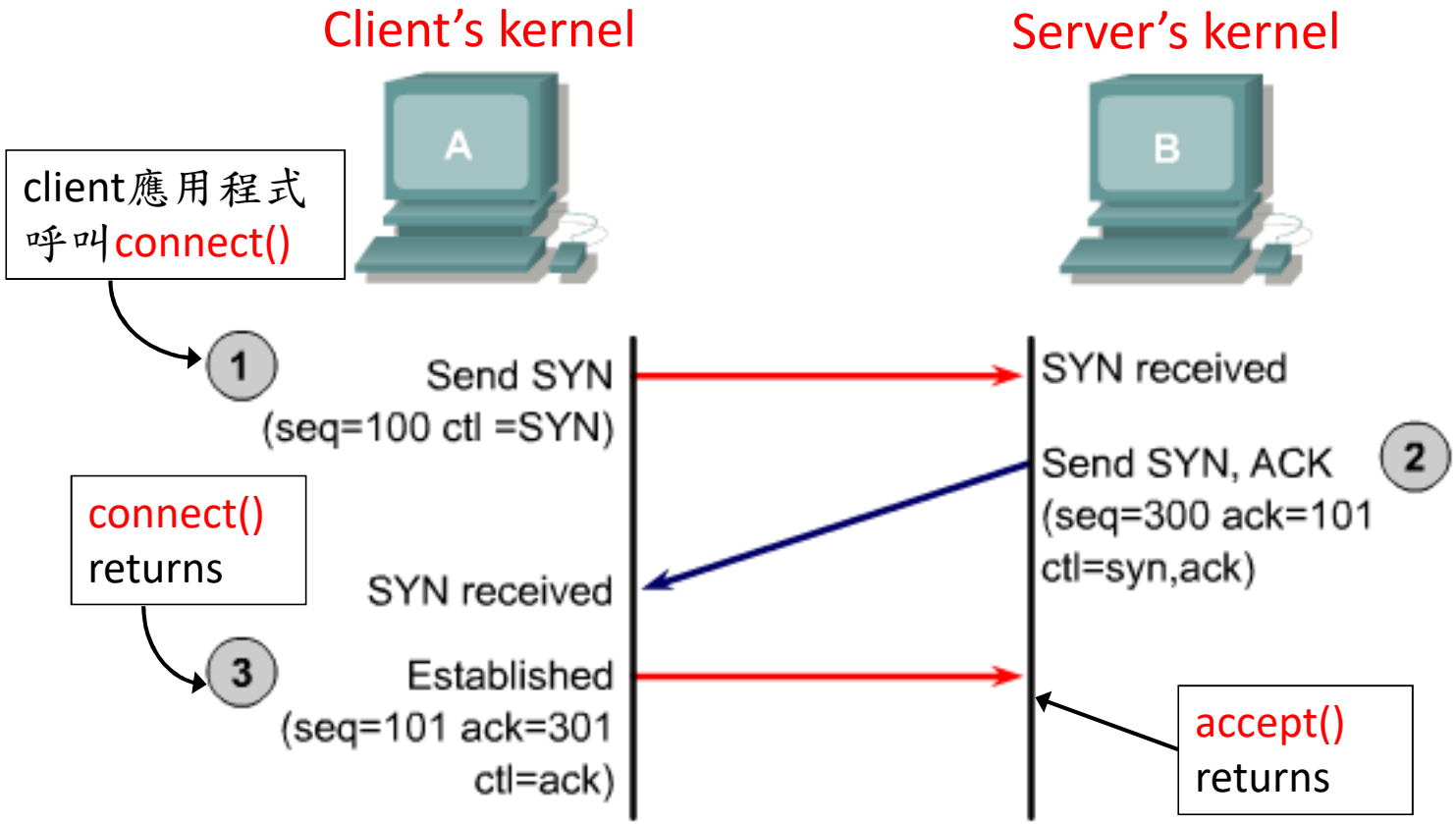
Five Steps to Create a Simple Daytime Client

1. Create TCP socket: get a file descriptor
2. Prepare server address structure: fill-in IP address and port number
3. Connect to the server: bind the file descriptor with the remote server
4. Read/write from/to server
5. Close socket

TCP client/server connection sequence



Three-Way Handshake of TCP (done by kernels)



intro/daytimetcpcli.c (1/2)

```
#include "unp.h"

int main(int argc, char **argv)
{
        int                     sockfd, n;
        char                    recvline[MAXLINE + 1];
        struct sockaddr_in      servaddr;

        if (argc != 2)
                err_quit("usage: a.out <IPaddress>");

        if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
                err_sys("socket error");
```

Command-Line Arguments

```
int main(int argc, char *argv[])
```

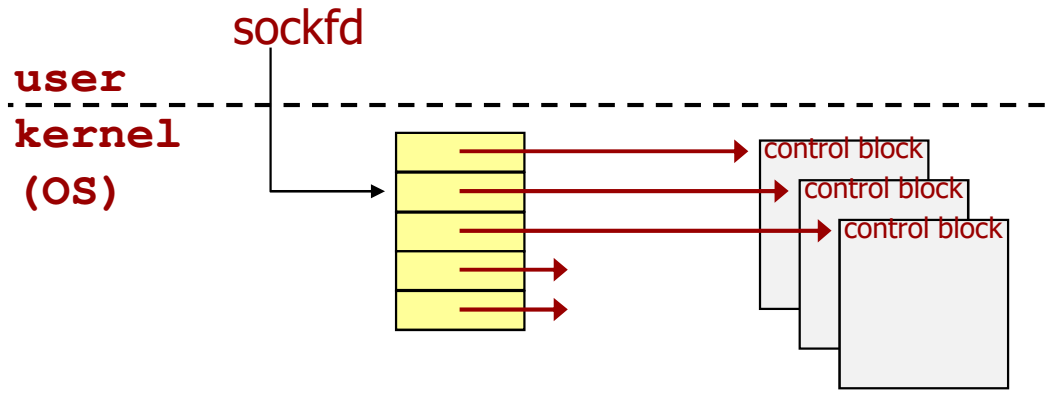
或 **argv

- `int argc`
  - Number of arguments passed
- `char *argv[]`
  - Array of strings
  - Has names of arguments in order
    - `argv[ 0 ]` is first argument
- Example: `$ mycopy input output`
  - `argc: 3`
  - `argv[ 0 ]: “mycopy"`
  - `argv[ 1 ]: "input"`
  - `argv[ 2 ]: "output"`

Step 1: Create A Socket

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

- Call to the function **`socket()`** creates a transport control block (hidden in kernel), and returns a reference to it (integer used as index)



Parameters of the `socket` Call

int socket(int *domain*, int *type*, int *protocol*);

returns a socket descriptor (or negative value on errors)

domain

predefined constant

family	Description
AF_INET	IPv4 protocols
AF_INET6	IPv6 protocols

type

predefined constant

type	Description
SOCK_STREAM	stream socket (TCP)
SOCK_DGRAM	datagram socket (UDP)

protocol

protocol	Description
0	Normally set to 0

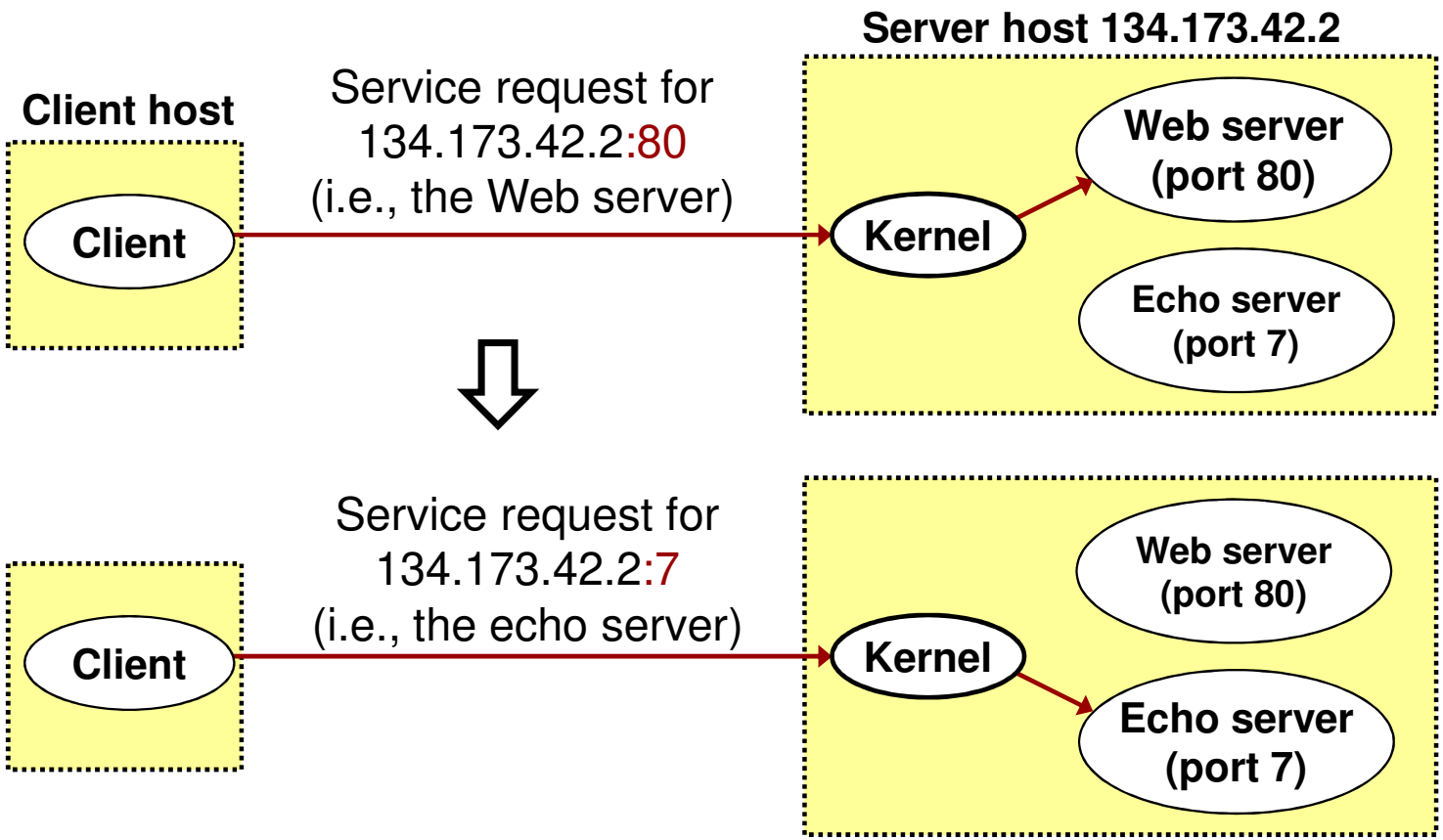
intro/daytimetcpcli.c (2/2)

```
bzero(&servaddr, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_port   = htons(13);  /* daytime server */
if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr) <= 0)
        err_quit("inet_pton error for %s", argv[1]);

        if (connect(sockfd, (SA *) &servaddr, sizeof(servaddr)) < 0)
                err_sys("connect error");

        while ( (n = read(sockfd, recvline, MAXLINE)) > 0) {
                recvline[n] = 0;        /* null terminate */
                if (fputs(recvline, stdout) == EOF)
                        err_sys("fputs error");
        }
        if (n < 0)
                err_sys("read error");
        exit(0);
}
```

How to identify clients to accept, and servers to contact?

- Machine??
  - by its IP address (e.g., 140.127.234.180)
- Application/service/program??
  - by (IP address and) port number
  - standard applications have own, “well-known” port numbers
    - SSH: 22
    - Mail: 25
    - Web: 80
    - Look in /etc/services for more (for Unix OS)

Using Ports to Identify Services



Specifying Server's IPv4 Address and Port Number

- Filling in structure sockaddr_in

```
struct sockaddr_in {
  uint8_t        sin_len;      /* length of structure */
  sa_family_t    sin_family;   /* AF_INET */
  in_port_t      sin_port;     /* 16-bit port#, network byte order */
  struct in_addr sin_addr;     /* 32-bit IPv4 address, network byte order */
  char           sin_zero[8];  /* unused */
};
```

```
struct in_addr {
  in_addr_t   s_addr;        /* 32-bit IPv4 address, network byte order */
};
```

Data Types Required by Posix.1g

Data type	Description	Header
int8_t	signed 8-bit integer	<sys/types.h>
uint8_t	unsigned 8-bit integer	<sys/types.h>
int16_t	signed 16-bit integer	<sys/types.h>
uint16_t	unsigned 16-bit integer	<sys/types.h>
int32_t	signed 32-bit integer	<sys/types.h>
sa_family_t	address family of socket addr struct	<sys/types.h>
socklen_t	length of socket addr struct, uint32_t	<sys/types.h>
in_addr_t	IPv4 address, normally uint32_t	<sys/types.h>
in_port_t	TCP or UDP port, normally uint16_t	<sys/types.h>

儲存真正IPv4位址的資料型態

儲存位址結構類別(AF_INET, AF_INET6, AF_LOCAL, …) 的資料型態

`sockaddr_in6`: For IPv6

```
struct sockaddr_in6 {
  uint8_t       sin6_len;      /* length of this struct [24] */
  sa_family_t   sin6_family;   /* AF_INET6 */
  in_port_t     sin6_port;     /* port#, network byte order */
  uint32_t      sin6_flowinfo; /* flow label and priority */
  struct in6_addr sin6_addr;   /* IPv6 address, network byte order */
};
```

```
struct in6_addr {
  uint8_t    s6_addr[16];    /* 128-bit IPv6 address, network byte order */
};
```

Step 2-1: Clear the Structure

```
struct sockaddr_in      servaddr;
…
bzero(&servaddr, sizeof(servaddr));
```

- bzero 從位址 &servaddr 處開始將sizeof(servaddr) 位元組的空間清為0
- 亦可使用標準庫存函式memset或其它方法

```
memset(&servaddr,0x0,sizeof(servaddr));
```

Step 2-2: Set IP Addr And Port

```
servaddr.sin_family = AF_INET;
servaddr.sin_port   = htons(13);        /* port no. = daytime server */
if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr) <= 0)
        err_quit("inet_pton error for %s", argv[1]);
```

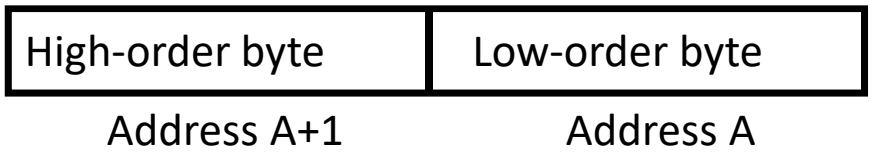
- htons將整數轉為network byte order
- inet_pton將位址字串轉換為內部儲存格式(傳回負值表轉換失敗)

  “197.15.22.131” ⇒ 1100010…011 ⇒ (network byte order)

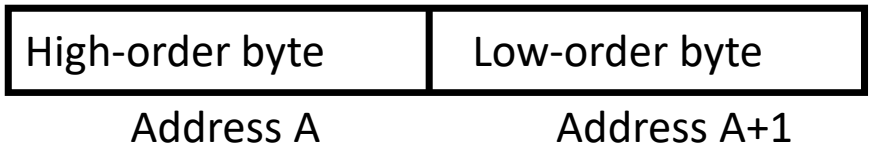
Network-byte Ordering

Two ways to store 16-bit/32-bit integers in an 8/16-bit computer

- Little-endian byte order (e.g. Intel)



- Big-endian byte order (E.g. Sparc)



Byte Order: IP address example

- IPv4 host address: represents a 32-bit address
  - written on paper (”dotted decimal notation”): **129.240.71.213**
  - binary in bits: **10000001 11110000 01000111 10001011**
  - hexadecimal in bytes: **0x81 0xf0 0x47 0x8b**
- Little-endian:
  - one 4 byte int on x86, StrongARM, XScale, …: 0x**81f0478b**
- Big-endian:
  - one 4 byte int on PowerPC, POWER, Sparc, …: 0x**8b47f081**
  - in network byte order: 0x**8b47f081**

Network-byte ordering (cont.)

- How do two machines with different byte-orders communicate?
  - Using network byte-order
  - Network byte-order = big-endian order
- Conversion macros **(<netinet/in.h>)**
  - **uint16_t htons (uint16_t n)**
  - **uint32_t htonl (uint32_t n)**

  host to network conversion (s: short; l: long)
  - **uint16_t ntohs (uint16_t n)**
  - **uint32_t ntohl (uint32_t n)**

  network to host conversion (s: short; l: long)

Function `inet_pton()`

- `inet_pton()` is new for IPv6 and may not exist yet (現在的Linux上沒問題).
- Oldest:

```
serveraddr.sin_addr.s_addr =
    inet_addr("129.240.65.193");
```

- Newer:

```
inet_aton("129.240.65.193",
    &serveraddr.sin_addr);
```

比inet_pton少一個參數(第一個AF_INET)

Step 3: Connect to the Server

```
int connect(int sockfd,
    struct sockaddr *serv_addr, socklen_t addrlen);
```

- **used by TCP client only**
- *sockfd* - socket descriptor (returned from socket())
- *serv_addr:* pointer to struct sockaddr (see next page)
- *addrlen* := sizeof(struct sockaddr)
- 傳回負值表連線建立失敗(如server不回應)

sockaddr: Generic (泛用) Socket Address Structure

```
struct sockaddr {                   /* only used to cast pointers */
  uint8_t       sa_len;
  sa_family_t   sa_family;          /* address family: AF_xxx value */
  char          sa_data[14];        /* protocol-specific address */
};
```

In unp.h

```
#define SA struct sockaddr
```

Casting The Pointer

```
int connect(int sockfd,
    struct sockaddr *serv_addr, socklen_t addrlen);
```

不用 struct sockaddr_in 或 struct sockaddr_in6是為了使connect可同時適用於IPv4與IPv6的位址結構

真正呼叫時傳入的還是指向 struct sockaddr_in 或 struct sockaddr_in6的指標，此指標需被強制形態轉換(cast)成指向 struct sockaddr的指標

```
connect(sockfd, (SA *) &servaddr, sizeof(servaddr)
```

Step 4: Read/Write from/to Server

```
while ( (n = read(sockfd, recvline, MAXLINE)) > 0) {
        recvline[n] = 0;        /* null terminate */
        if (fputs(recvline, stdout) == EOF)
                err_sys("fputs error");
}
if (n < 0)
        err_sys("read error");
```

- 使用read讀取從server送過來的資料，並用fputs輸出到螢幕上 (此例client並沒有送資料到server)

`read` Function

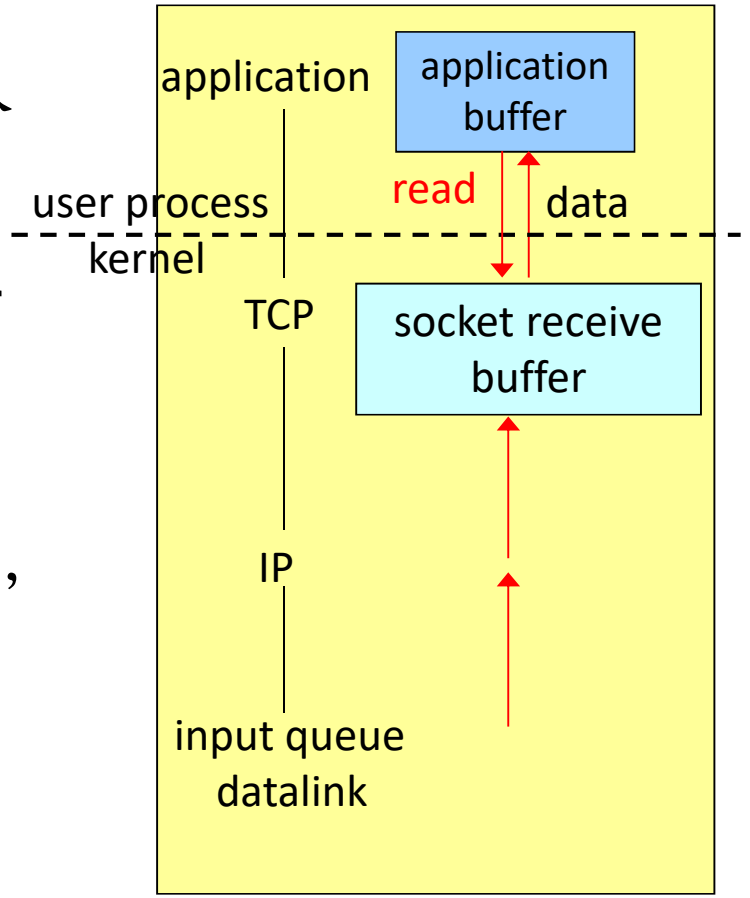
```
int read(int sockfd, char *buf, int maxlen);
```

- *sockfd* - socket descriptor (returned from socket())
- *buf*: buffer (in your program) to store received data
- *maxlen*: maximum number of bytes to receive
- Returns: # of bytes received (≤ *maxlen*) if OK,
  0 if connection has been gracefully closed,
  negative number on error

  (The received data are not null-terminated.)

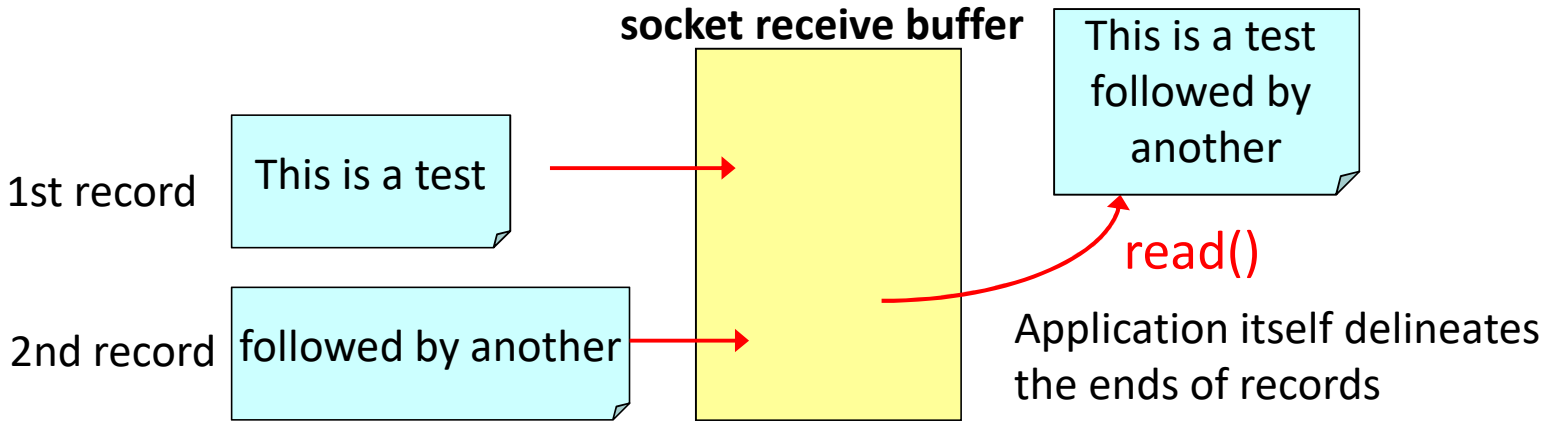
關於使用`read`讀取資料

- 從server送過來的data是先放在位於kernel的socket receive buffer
- client應用程式呼叫read()將此buffer的資料取回放於自己程式的buffer `recvline`
- 如socket receive buffer中沒有資料，則read呼叫會被block (暫時無法return)



TCP Is Stream-Based

- TCP provides no record markers
- If two records of data arrive after the last read, the **`read()`** may return one block of data



fputs 與 err_sys

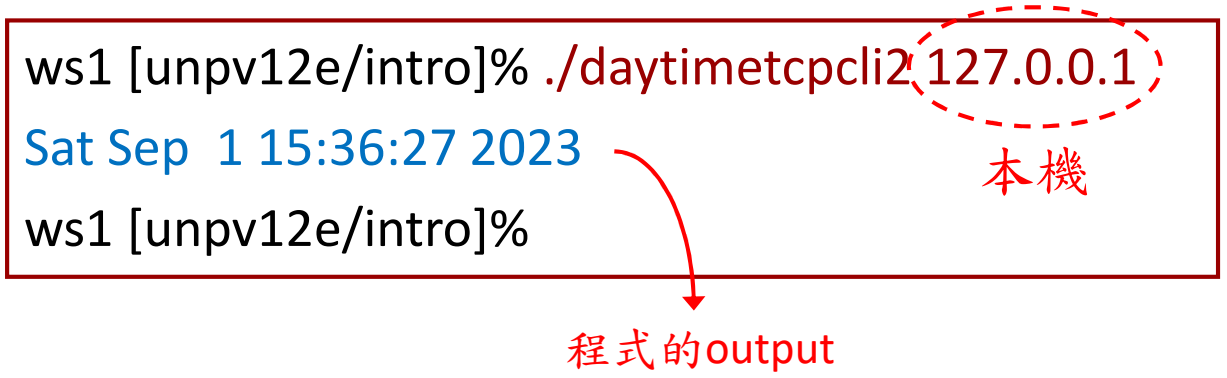
- **fputs**是C的標準庫存函數，功能是將字串str寫到file pointer *stream中

```
int fputs( const char *str, FILE *stream );
```

- 成功傳回非負整數值；否則傳回EOF
- **stdout**: File pointer for standard output stream. Automatically opened when program execution begins.
- **err_sys**是作者自行定義的函數(需先include “unp.h” 方可使用)

Test the Program

- 從課程網頁下載原始程式檔
- 依課程網頁的說明解壓縮、設定、及編譯程式
- 若可出現類似下列執行結果即表示成功



關於程式執行

- “./”表示執行目前這個目錄中的…
- 測試時本機必須先執行Daytime server daytimetcpsrv2
- Daytime server的程式碼稍後介紹

intro/daytimetcpcliv6.c (1/2)

(for IPv6)

```
#include "unp.h"

int main(int argc, char **argv)
{
        int                     sockfd, n;
        char                    recvline[MAXLINE + 1];
        struct sockaddr_in6     servaddr;

        if (argc != 2)
                err_quit("usage: a.out <IPaddress>");

        if ( (sockfd = socket(AF_INET6, SOCK_STREAM, 0)) < 0)
                err_sys("socket error");
```

intro/daytimetcpcliv6.c (2/2)

```
 bzero(&servaddr, sizeof(servaddr));
 servaddr.sin6_family = AF_INET6;
 servaddr.sin6_port   = htons(13);/* daytime server */
 if (inet_pton(AF_INET6, argv[1], &servaddr.sin6_addr) <= 0)
        err_quit("inet_pton error for %s", argv[1]);

        if (connect(sockfd, (SA *) &servaddr, sizeof(servaddr)) < 0)
                err_sys("connect error");

        while ( (n = read(sockfd, recvline, MAXLINE)) > 0) {
                recvline[n] = 0;        /* null terminate */
                if (fputs(recvline, stdout) == EOF)
                        err_sys("fputs error");
        }
        if (n < 0)
                err_sys("read error");
        exit(0);
}
```

A Simple Daytime Server

- Create TCP socket: get a file descriptor
- Bind the socket with its local port
- Listen: convert the socket to a listening descriptor
- Accept blocks to sleep
- Accept returns a connected descriptor
- Read/write
- Close socket

Error Handling: Wrappers

lib/wrapsock.c

```
/* include Socket */
int
Socket(int family, int type, int protocol)
{
        int             n;

        if ( (n = socket(family, type, protocol)) < 0)
                err_sys("socket error");
        return(n);
}
/* end Socket */
```

大寫

將函數傳回值的檢查包裝在以大寫字母開頭的同名函數中

intro/daytimetcpsrv.c (1/2)

```
#include"unp.h"
#include<time.h>

int
main(int argc, char **argv)
{
    int                 listenfd, connfd;
    struct sockaddr_in  servaddr;
    char                buff[MAXLINE];
    time_t              ticks;

    listenfd = Socket(AF_INET, SOCK_STREAM, 0);
```

大寫

intro/daytimetcpsrv.c

```
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family= AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port= htons(13);   /* daytime server */

    Bind(listenfd, (SA *) &servaddr, sizeof(servaddr));
    Listen(listenfd, LISTENQ);
    for ( ; ; ) {
        connfd = Accept(listenfd, (SA *) NULL, NULL);

        ticks = time(NULL);
        snprintf(buff, sizeof(buff), "%.24s\r\n", ctime(&ticks));
        Write(connfd, buff, strlen(buff));
        Close(connfd);
    }
}
```

大寫

大寫

大寫

大寫

`bind` Function

```
int bind(int sockfd, const struct sockaddr *myaddr,
  socklen_t addrlen);
```

- *sockfd*: Descriptor identifying an unbound socket. (returned by the *socket* function.)
- *myaddr*: A pointer to a protocol-specific address structure
- *addrlen*: Length of the value in the *myaddr* parameter, in bytes.
- returns: 0 if OK, negative number on error

`listen` Function

int listen(int *sockfd*, int *backlog*);

- *sockfd*: Descriptor identifying a bound socket. (returned by the *socket* function.)
- *backlog*: how many connections we want to queue

Problems with the the Simple Server

- Protocol dependency on IPv4
- Iterative server: no overlap of service times of different clients (一次只能服務一個client)
- Need for concurrent server: fork, pre-fork, or thread
- Need for daemon: background, unattached to a terminal

Types of Networking APIs

- TCP socket
- UDP socket
- raw socket over IP (bypass TCP/UDP)
- datalink (bypass IP)
  - BPF (BSD Packet Filter)
  - DLPI (SVR4 Data Link Provider Interface)

BSD Networking History

- BSD releases:
  - Licensed: 4.2 BSD with TCP/IP and socket APIs(1983), 4.3 BSD (1986), 4.3 BSD Tahoe (1988), 4.3 Reno (1990), 4.4 BSD (1993)
  - Non-licensed: Net/1 (Tahoe) (1989), Net/2 (Reno) (1991), 4.4 BSD-Lite (Net/3) (1994), 4.4 BSD-Lite2 (1995) ---> BSD/OS, FreeBSD, NetBSD, OpenBSD
- System V Release 4: Solaris and Linux
- UNIX Standards: POSIX and The Open Group

Discovering Details of Your Local Network

- No longer preinstalled in Linux, should manually install
  - **`sudo apt install net-tools`** (Ubuntu)
  - **`sudo dnf install net-tools`** (Red Hat)
- To find out interfaces: netstat -ni
- To find out routing table: netstat -rn
- To find out details of an interface: ifconfig
- To discover hosts on a LAN: ping