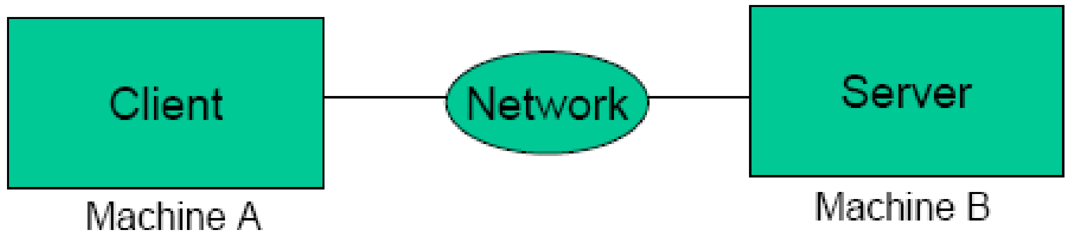
Computer Network Programming

Introduction to the Socket Program in Linux

Introduction

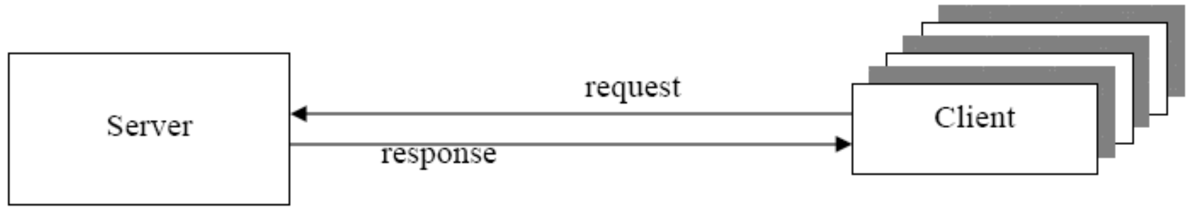
- A Simple Daytime Client
- A Simple Daytime Server
- Error handling: wrapper functions
- Types of Networking APIs
- BSD networking history
- Discover details of your local network

Client-Server Model



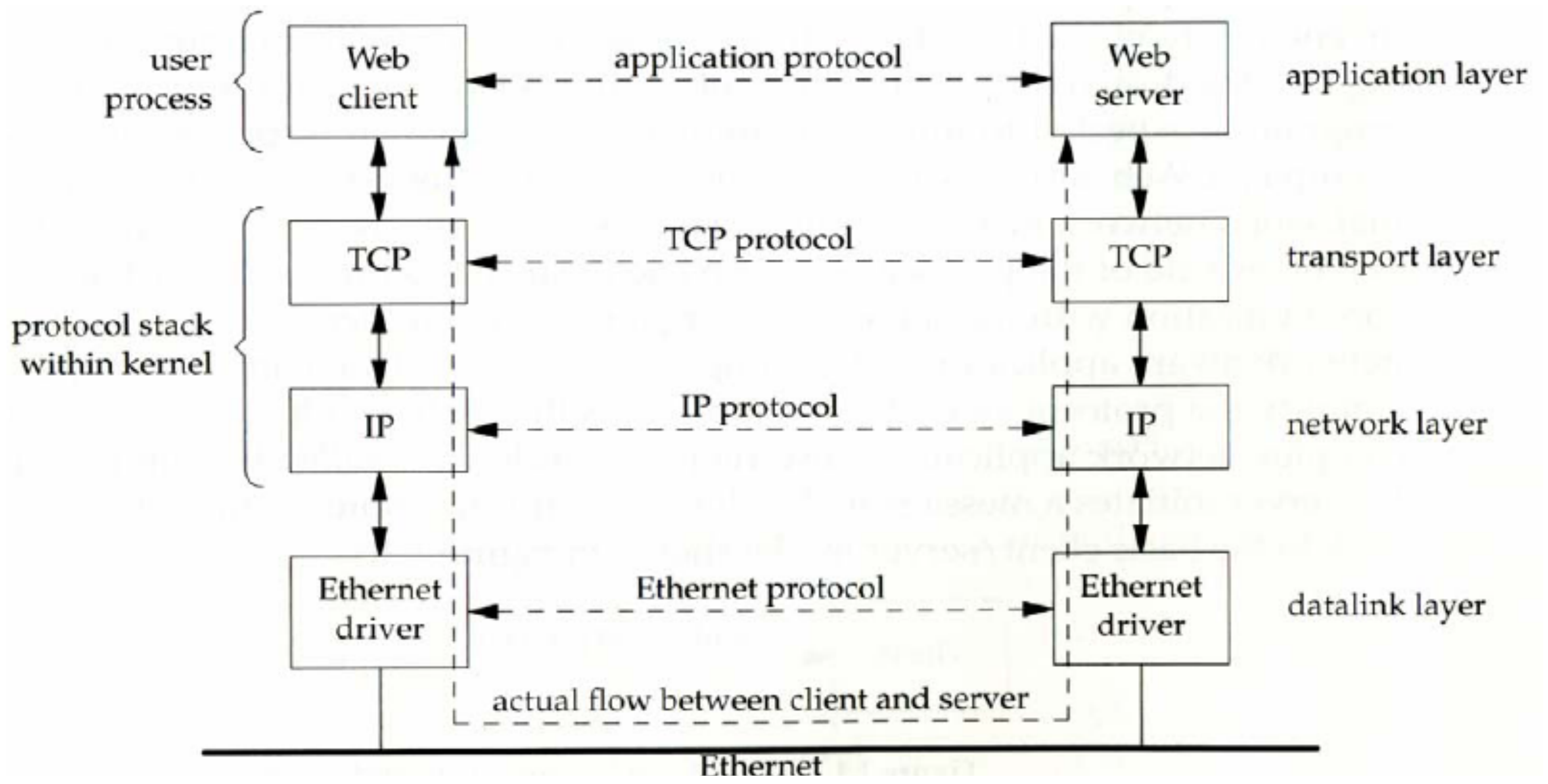
- Web browser and server
- FTP client and server
- Telnet client and server

The Client/Server Model

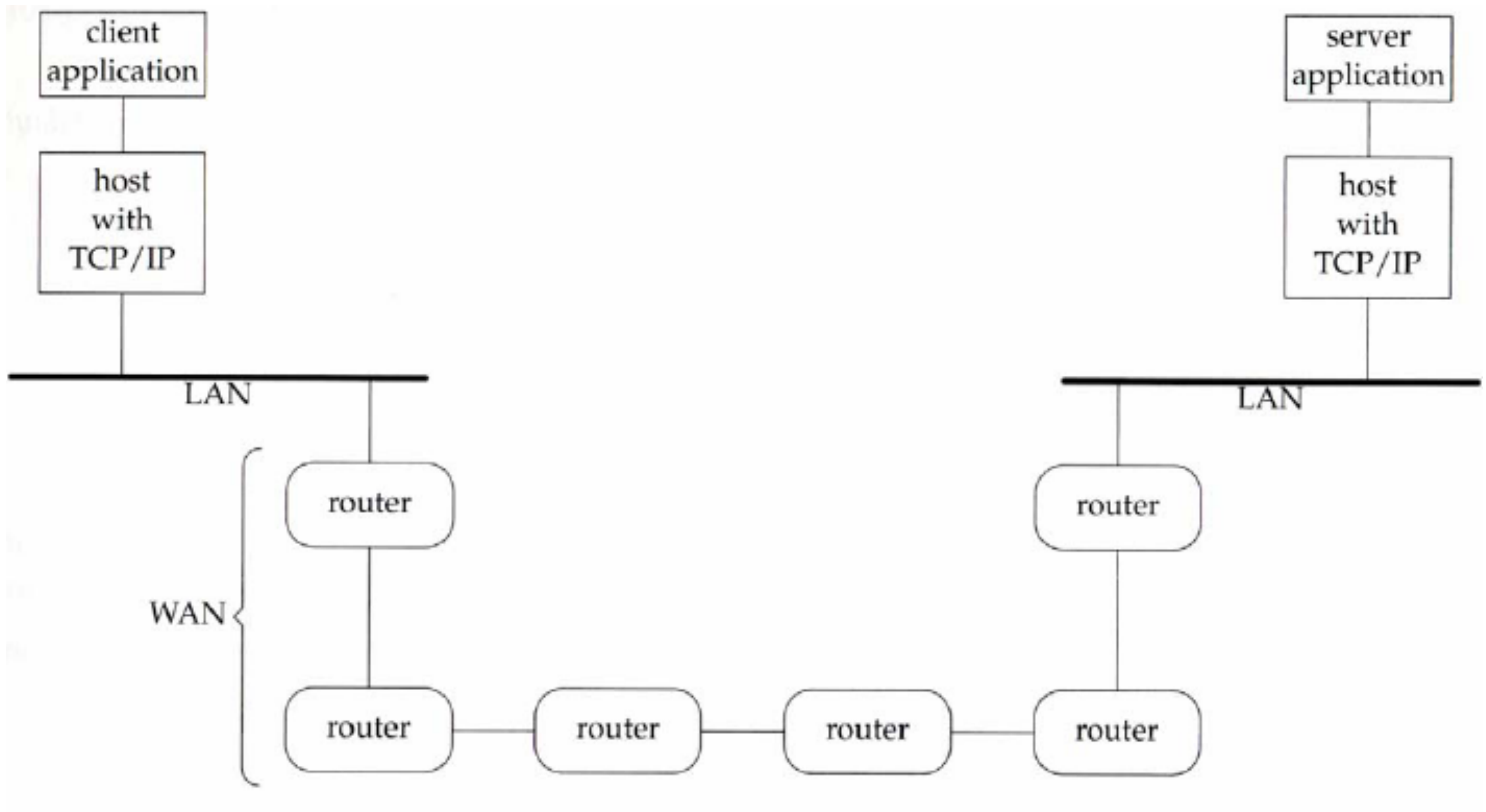


Starts and initializes.	Starts and initializes.
Goes to sleep waiting for client to connect.	Waits for user input.
When contacted by client, calls/creates a handler to handle. (Goes back to sleep: concurrent)	Upon receiving user request, contacts server, sends request on user's behalf.
Handles client request, sends back reply …	Waits for server reply, sends results to user, sends another user request …
Closes client connection. (Goes back to sleep: iterative).	Closes server connection.

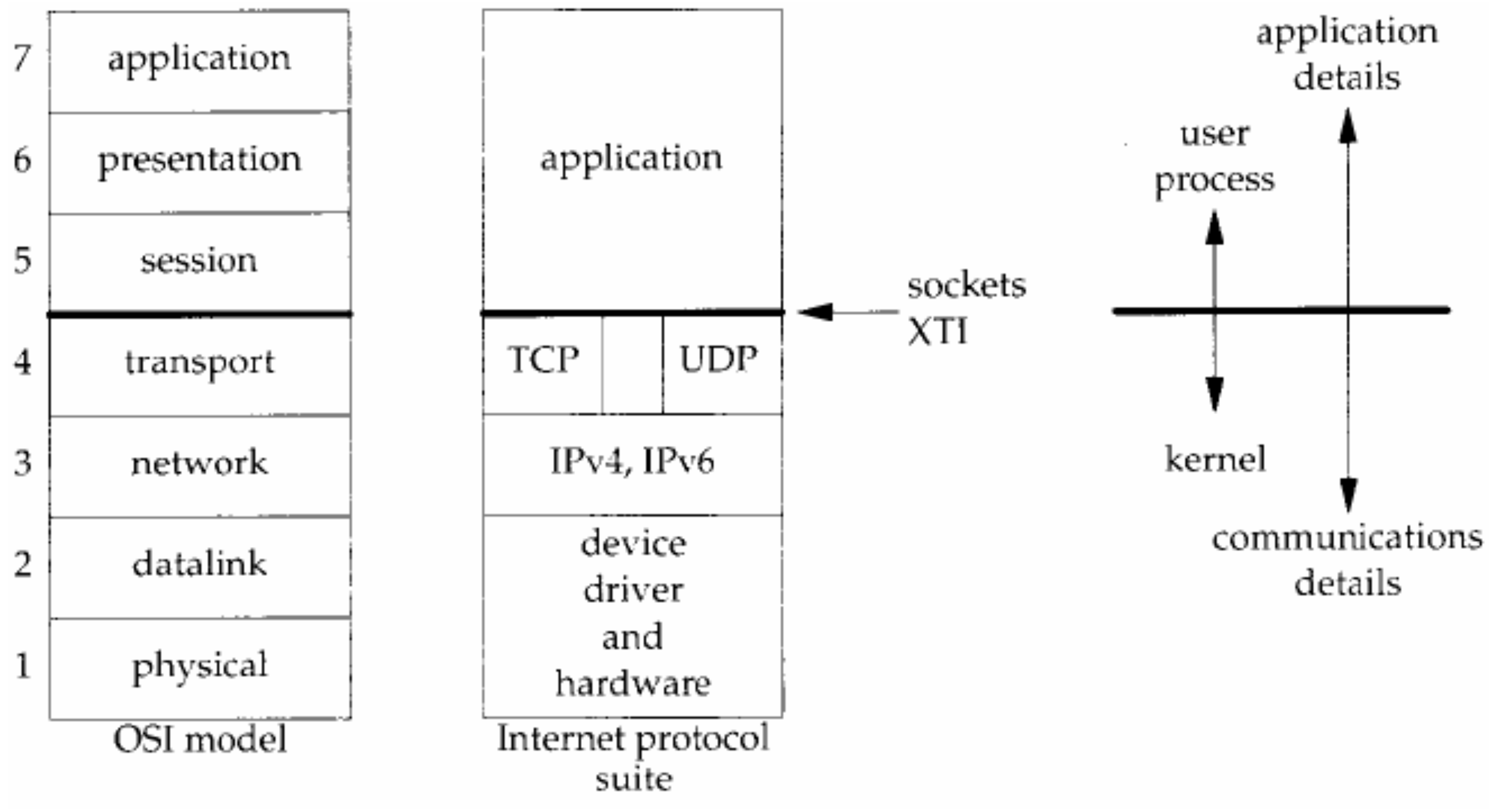
Client/Server on an LAN



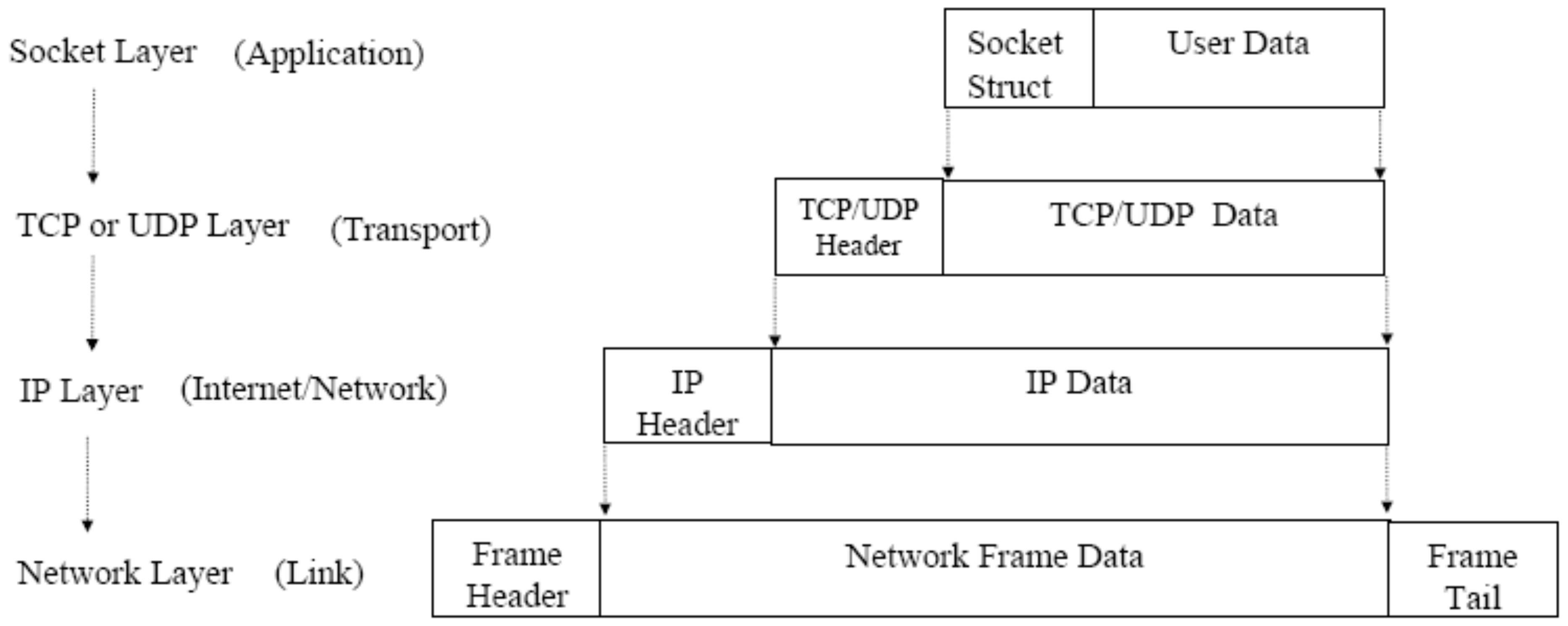
Client/Server via a WAN



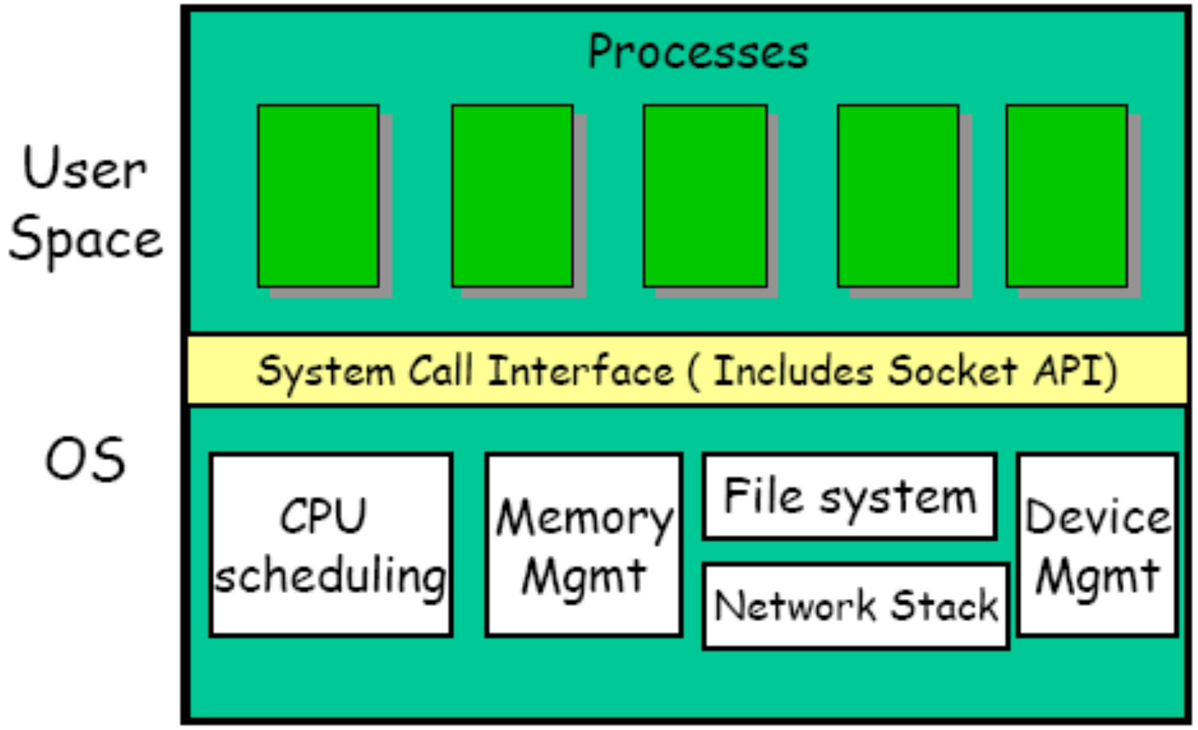
OSI vs. Internet Protocol Layers



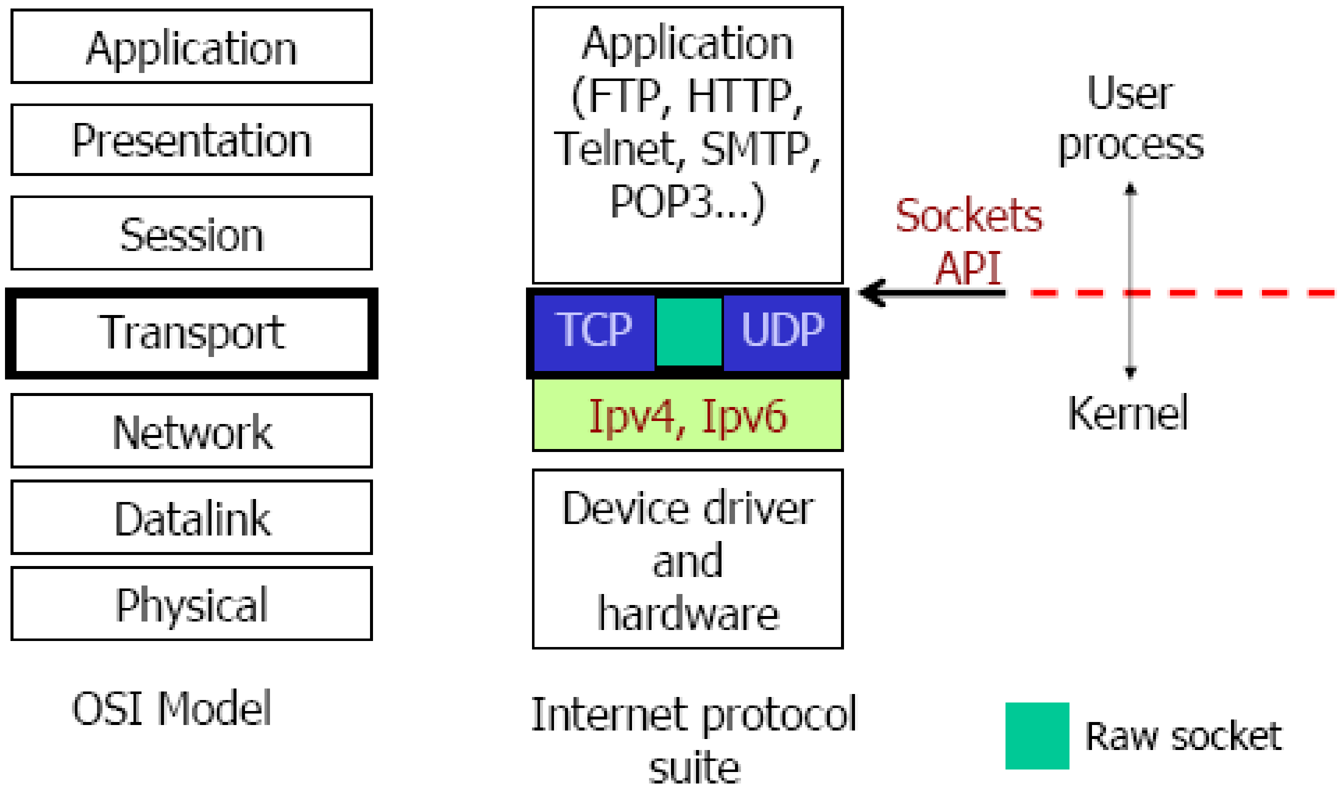
Protocol Data and Headers



Socket API Location in OS

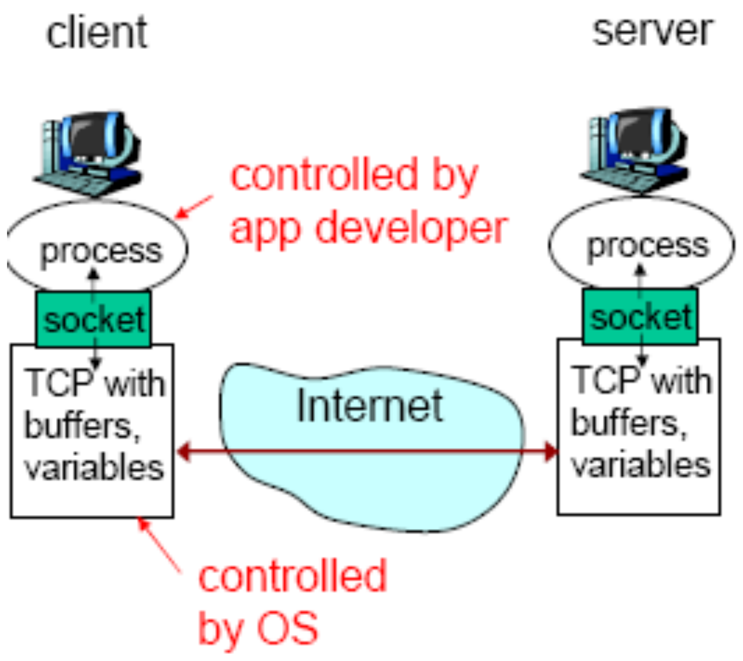


OSI and TCP/IP Model

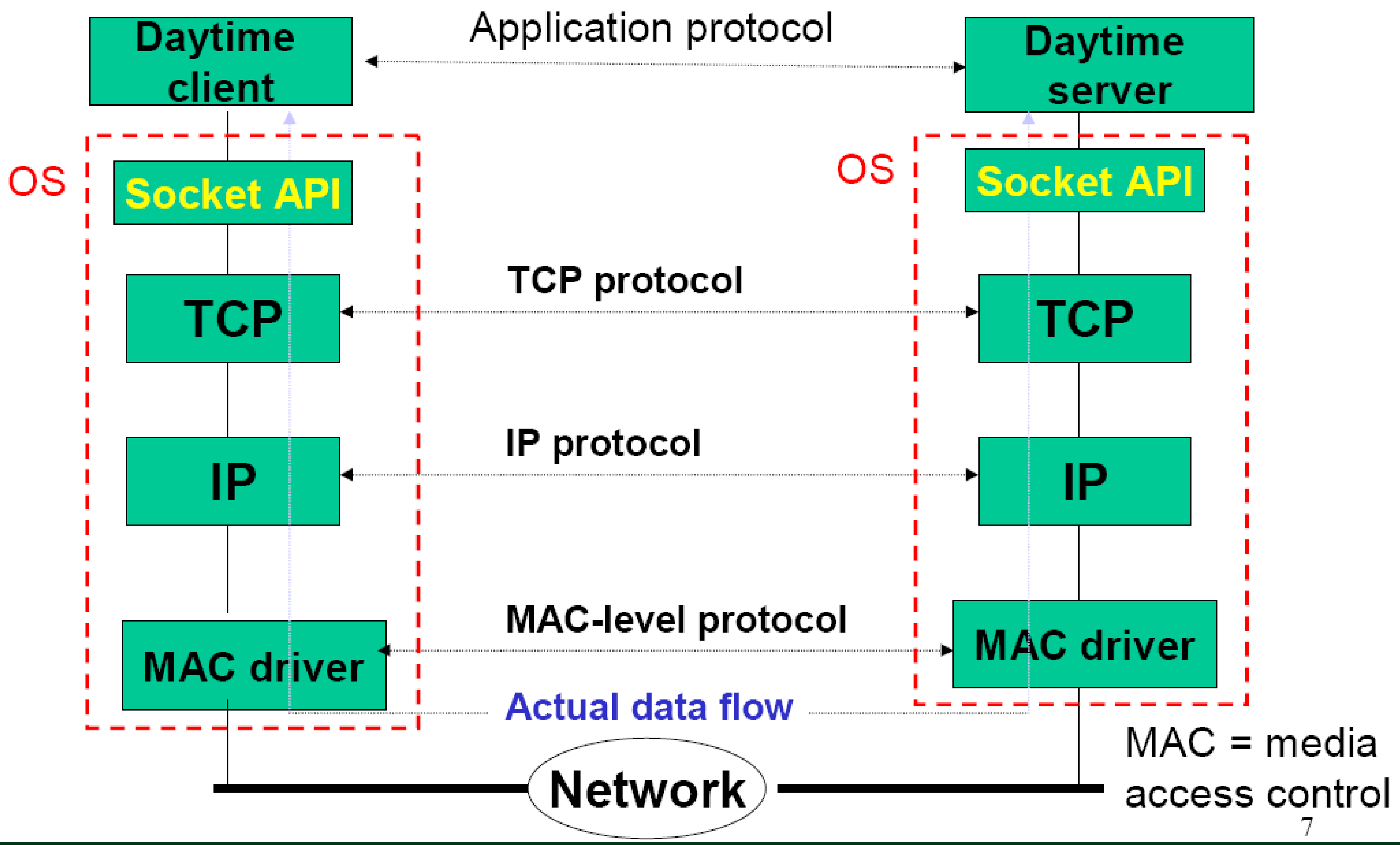


Sockets

- 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

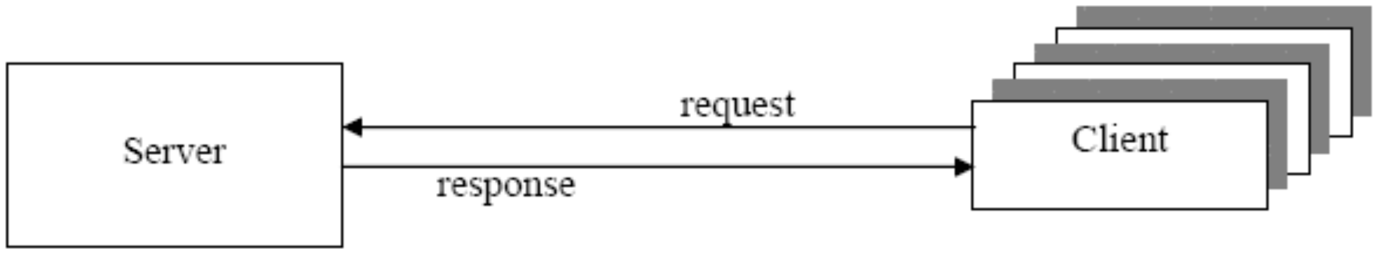

client
server
process
socket
TCP with buffers, variables
controlled by app developer
Internet
process
socket
TCP with buffers, variables
controlled by OS

A Daytime client/server using socket API

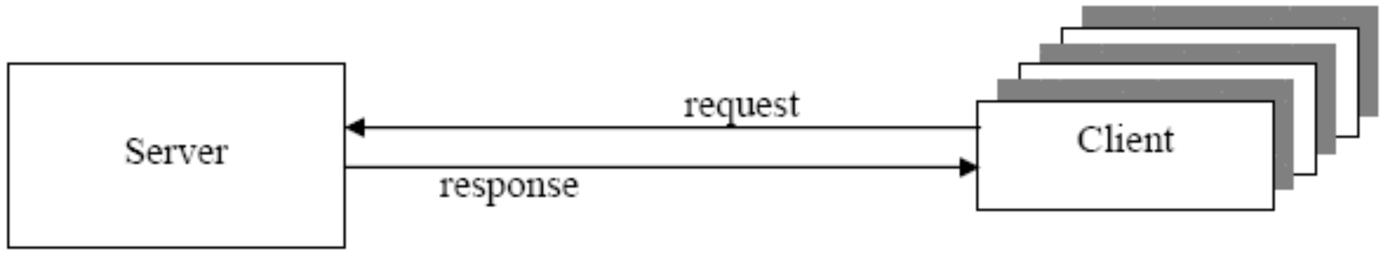


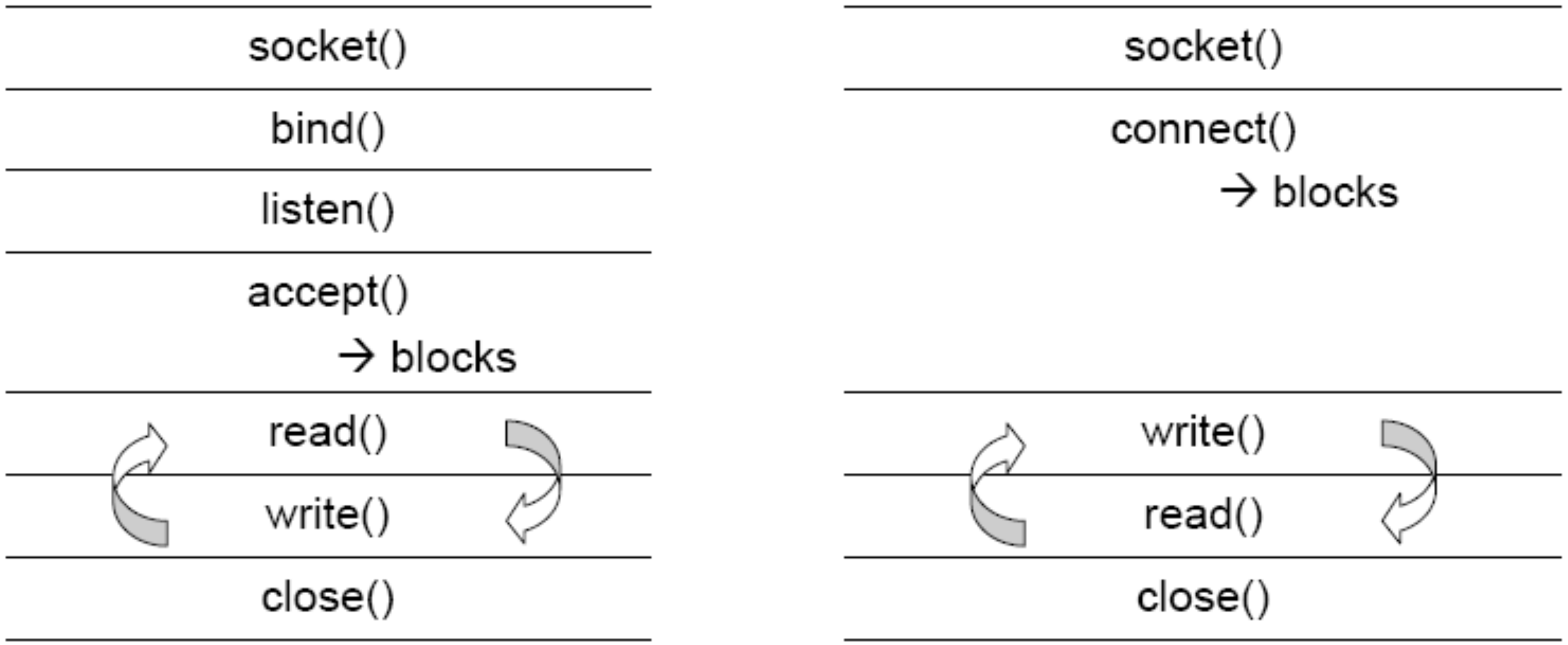
7

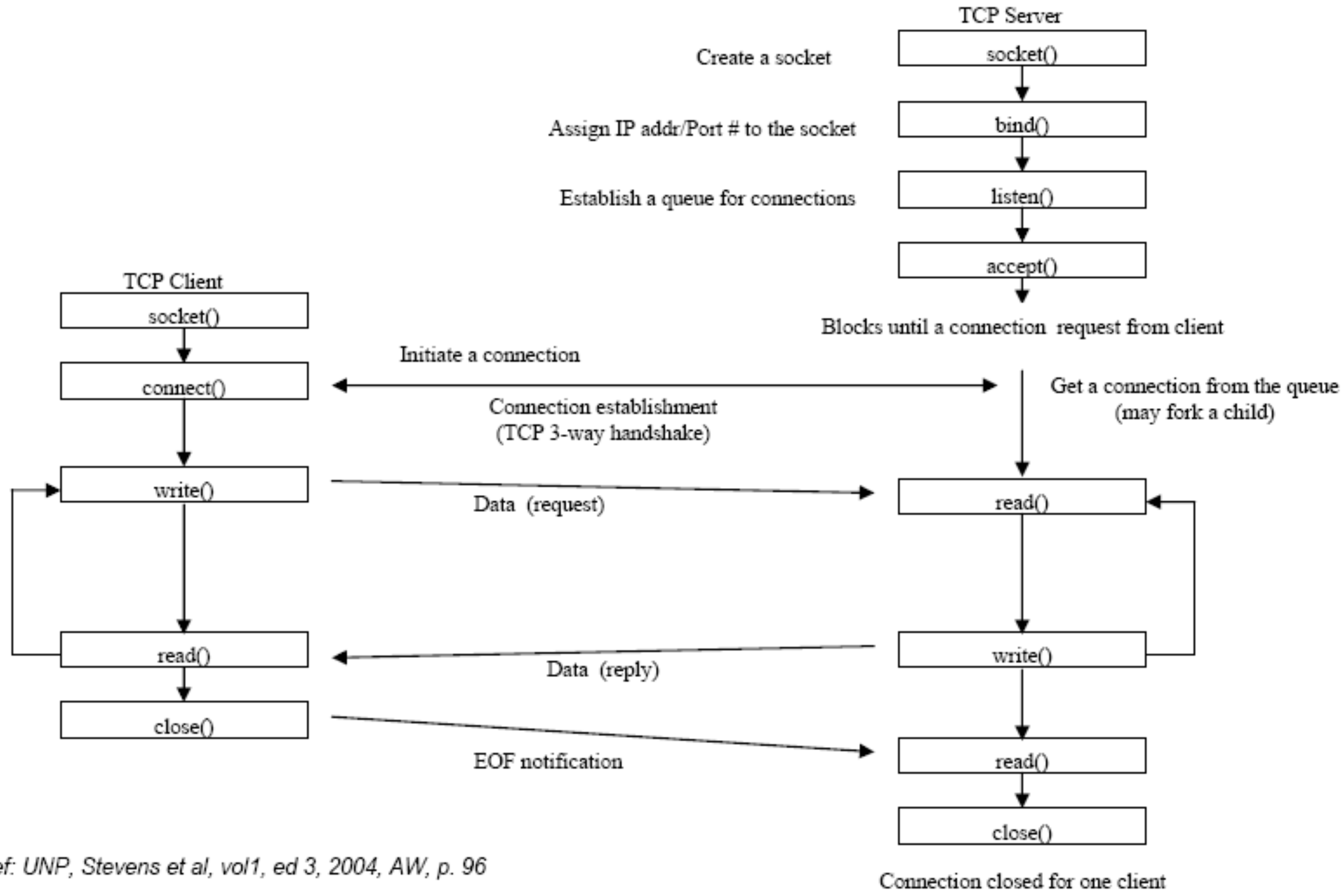
Connectionless: Functions Used



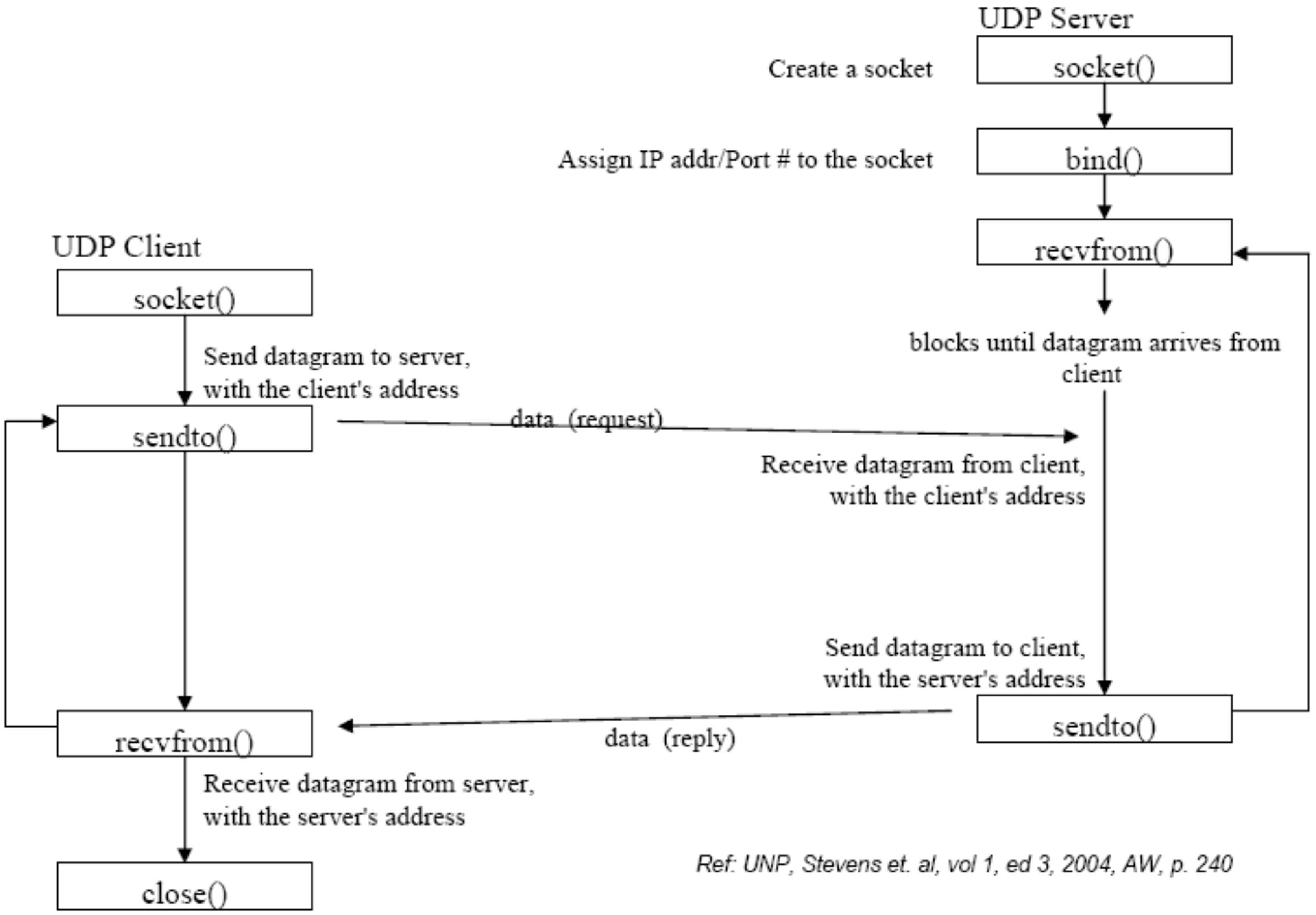
socket()	socket()
bind()	bind() → optional
recvfrom() → blocks	sendto()
sendto()	recvfrom() → blocks


request
Server
Client
response


socket()
bind()
listen()
accept()
→ blocks
read()
write()
close()
socket()
connect()
→ blocks
write()
read()
close()



Ref: UNP, Stevens et al, vol1, ed 3, 2004, AW, p. 96

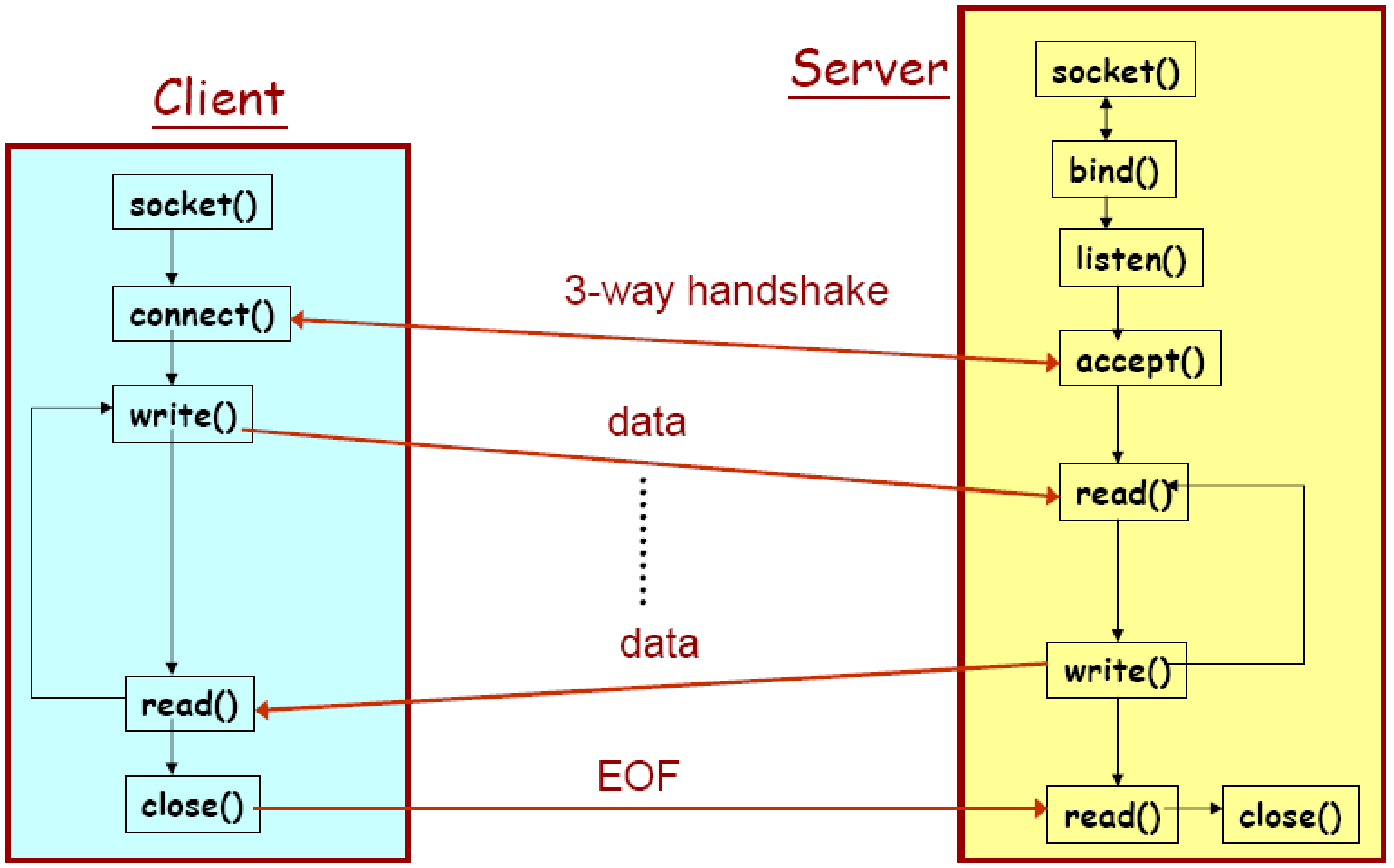

UDP Server
Create a socket
socket()
Assign IP addr/Port # to the socket
bind()
recvfrom()
blocks until datagram arrives from client
UDP Client
socket()
Send datagram to server, with the client's address
sendto()
data (request)
Receive datagram from client, with the client's address
Send datagram to client, with the server's address
sendto()
data (reply)
recvfrom()
Receive datagram from server, with the server's address
close()

Ref: UNP, Stevens et. al, vol 1, ed 3, 2004, AW, p. 240

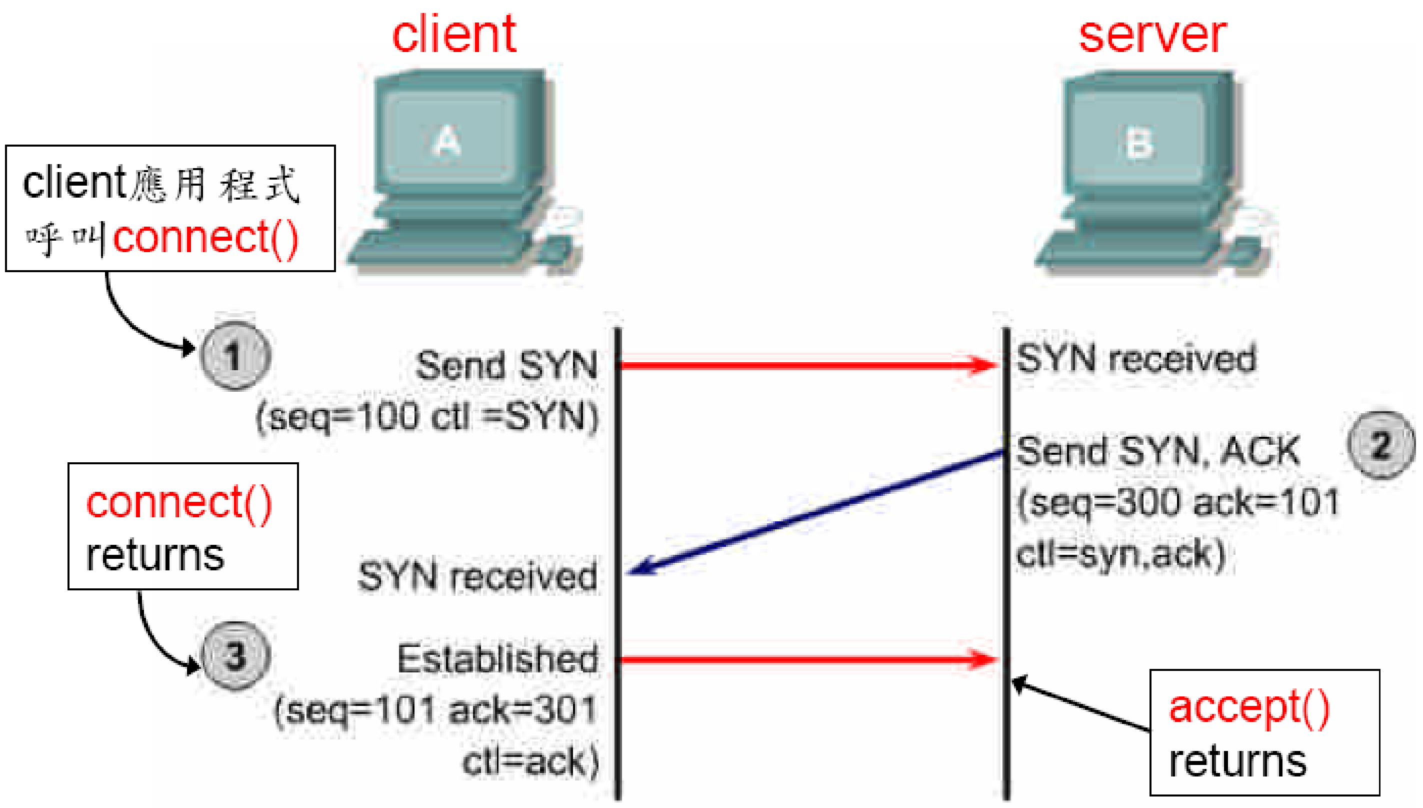
Five Steps to Create a Simple Daytime Client

1. Create TCP socket: get a file descriptor
2. Prepare server address structure: filling 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



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

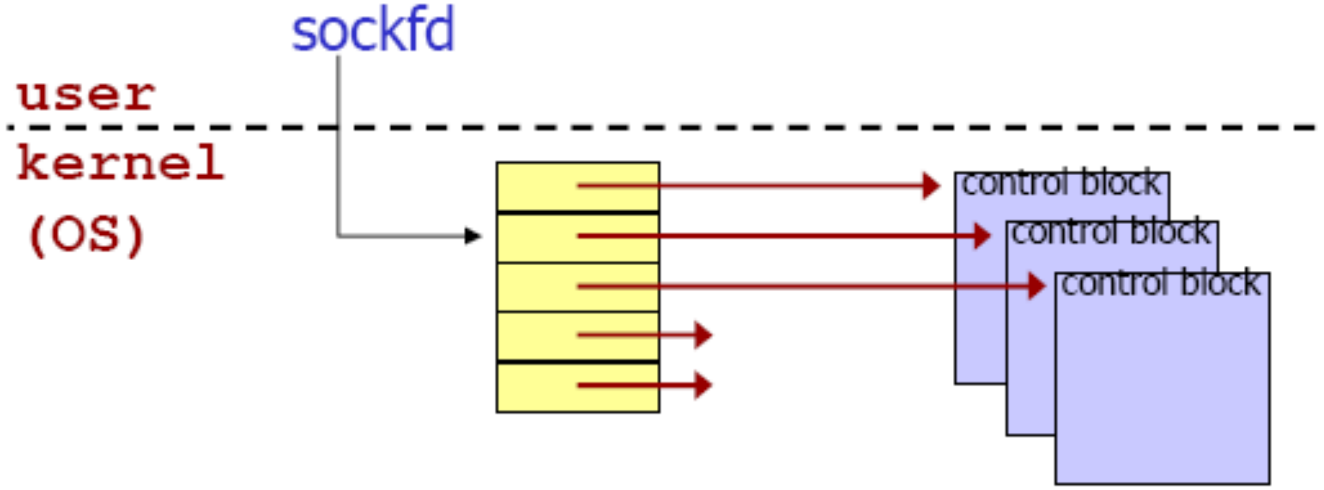
```
int main(int argc, char *argv[])
```

- 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

Socket functions

- listenfd = Socket(AF_INET, SOCK_STREAM, 0); //建立TCP套接口描述字
  bzero(&servaddr, sizeof(servaddr)); //地址結構清零
  servaddr.sin_family = AF_INET; //IPv4協議
  servaddr.sin_addr.s_addr = htonl(INADDR_ANY); //內核指定地址
  servaddr.sin_port = htons(SERV_PORT); //總所諸知端口#9877
  // 允許啟動一個監聽服務器並捆綁其眾所周知端口，即使以前建立的將此端口用作它們的本地端口的連接仍存在。
  Setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));
  Bind(listenfd, (SA *) &servaddr, sizeof(servaddr)); //綁定本機地址和端口
  Listen(listenfd, LISTENQ); //監聽

intro/daytimetcpcli.c (2/2)

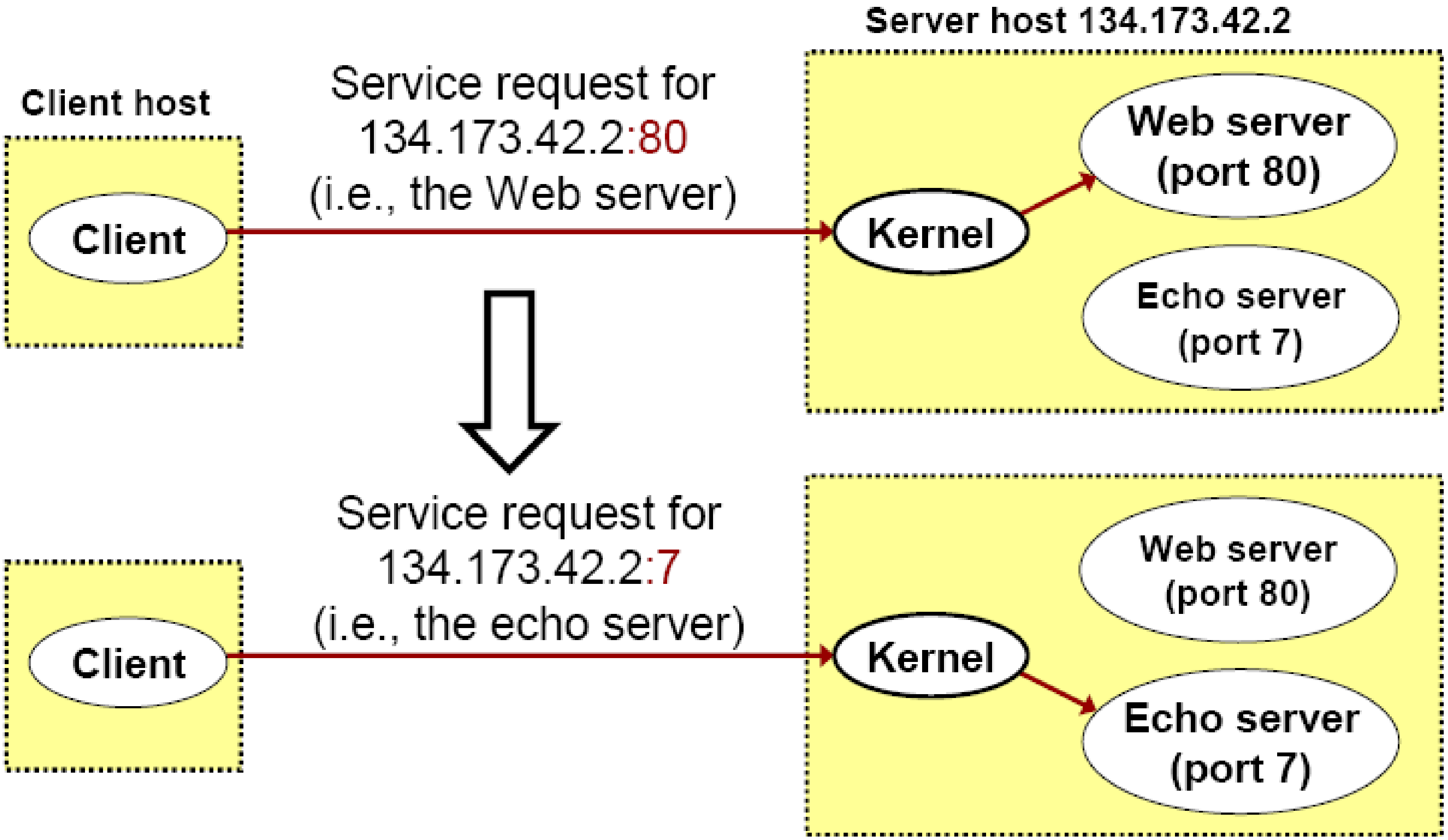
```
bzero(&servaddr, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(13); /* daytime server */
if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr) <= 0)
    err_quit("inet_pton error for %s", argv[1]);

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



Port Numbers

- A (protocol) port is an abstraction used by TCP/UDP to distinguish applications on a given host
  - A port is identified by a 16-bit integer known as the *port number*
- Three ranges of port numbers :
  - Well-known ports
  - Registered ports
  - Dynamic ports

Well-known Ports

- Port numbers ranging from 0 to 1,023
  - A set of pre-assigned port numbers for specific uses
- Port numbers are managed by ICANN.
  - Short for the ***I***nternet ***C***orporation for ***A***ssigned ***N***ames and ***N***umbers (ICANN)
  - Used to be controlled solely by IANA (**I**nternet **A**ssigned **N**umbers **A**uthority)

Some Well-known Ports

Port	Keyword	Description
0		Reserved
7	ECHO	Echoes a received datagram to the sender
13	DAYTIME	Returns the date and the time
20	FTP-DATA	File Transfer Protocol (data)
21	FTP	File Transfer Protocol (control)
22	SSH	Secure Shell
23	TELNET	Terminal Connection
25	SMTP	Simple Mail Transport Protocol
53	DNS	Domain Name Server
67	BOOTP	Bootstrap Protocol
79	FINGER	Finger
80	HTTP	HyperText Transfer Protocol
101	HOSTNAME	NIC Host Name Server
103	X400	X.400 Mail Service
110	POP3	Post Office Protocol Vers. 3

Registered Ports

- Port numbers ranging from 1,024 to 49,151
- Not assigned or controlled by ICANN; however their uses need to be registered via an ICANN-accredited registrar to prevent duplications

Dynamic Ports

- Port numbers ranging from 49,152 to 65,535
- Neither assigned or registered. They can be used by anyone
  - These are ephemeral ports
  - Also known as private ports

Specifying Server's IP Address and Port Number

- Filling in structure sockaddr_in

```
struct sockaddr_in {
unsigned short sin_family; /* address family (always AF_INET) */
unsigned short sin_port; /* port num in network byte order */
struct in_addr sin_addr; /* IP addr in network byte order */
unsigned char sin_zero[8]; /* pad to sizeof(struct sockaddr) */
};
```

Client程式須在此結構的欄位中填入server的IP address和port number

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));
```

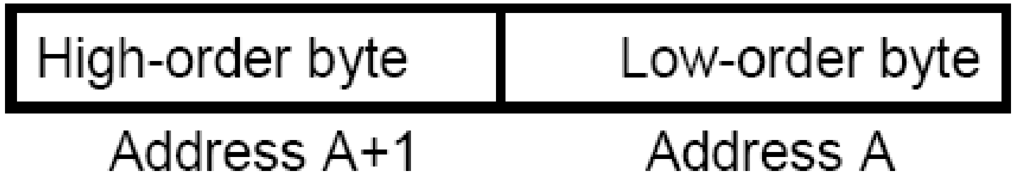
Step 2-2: Set IP Addr And Port

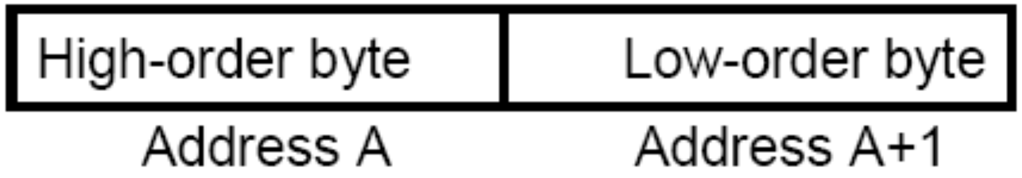
```
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(13);/* port no=daytime server */
if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr) <= 0)
    err_quit("inet_pton error for %s", argv[1]);
```

- 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)

Network-byte ordering

- Two ways to store 16-bit/32-bit integers
  - Little-endian byte order (e.g. Intel)
  - Big-endia





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**
- Network byte order uses big-endian ordering
- Little-endian:
  - one 4 byte int on x86, StrongARM, XScale, …: 0x8b47f081
- Big-endian:
  - one 4 byte int on PowerPC, POWER, Sparc, … 0x81f0478b

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)**

  $\Big\}$ **host to network conversion (s: short; l: long)**

  - **uint16_t ntohs (uint16_t n)**
  - **uint32_t ntohl (uint32_t n)**

  $\Big\}$ **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_in *serv_addr, int addrlen);
```

- **used by TCP client only**
- *sockfd* - socket descriptor (returned from socket())
- *serv_addr:* socket address, struct sockaddr_in is used
- *addrlen* := sizeof(struct sockaddr_in)
- 傳回負值表連線建立失敗(如server不回應)

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");
```

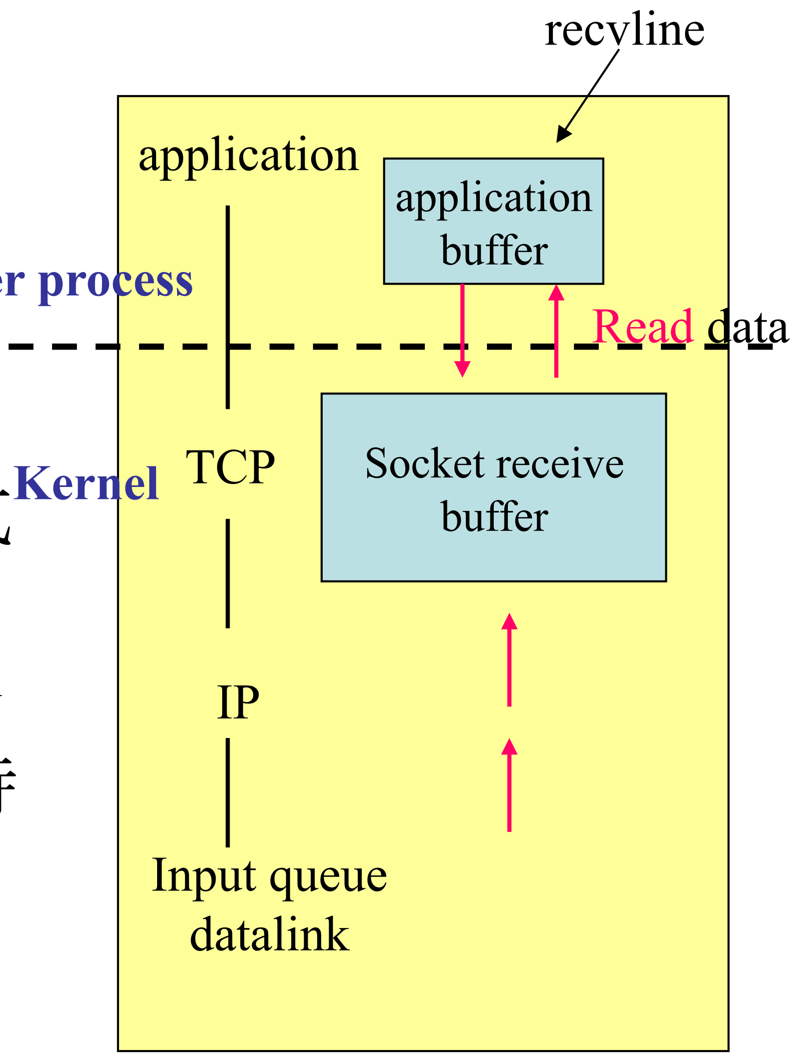
read Function

```
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

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)



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

本機

```
ws1 [unpv12e/intro]% ./daytimetcpcli 127.0.0.1
Mon Mar 5 15:36:27 2007
ws1 [unpv12e/intro]%
```

程式的**output**

關於程式執行

- “./”表示執行目前這個目錄中的…
- 測試時本機必須有執行Daytime server。有些系統如Ubuntu預設沒有Daytime server，則必須先啟動Daytime server
- Daytime server的程式碼稍後介紹

Error Handling: Wrappers

lib/wrapsock.c

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

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);
```

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 */

      大寫
Bind(listenfd, (SA *) &servaddr, sizeof(servaddr));
      大寫
Listen(listenfd, LISTENQ);

for ( ; ; ) {
                              大寫
        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
   }
```

bind Function

```
int bind(int sockfd, const struct sockaddr_in *name,
           int namelen );
```

- ***sockfd***: Descriptor identifying an unbound socket. (returned by the ***socket*** function.)
- ***name*****:** A pointer to a protocol-specific address
- ***namelen***: Length of the value in the ***name***
- 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, prefork, 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)

Family	Description
SOCK_STREAM	stream socket (TCP)
SOCK_DGRAM	datagram socket (UDP)
SOCK_SEQPACKET	sequenced packet socket (SCTP)
SOCK_RAW	raw socket (talk to IP directly)

Some Relevant Socket System Calls and Header Files

```
int socket(int family, int type, int protocol);
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
int listen(int sockfd, int backlog);
int accept(int sockfd, struct sockaddr *cliaddr, socklen_t *cliaddrlen);
int connect(int sockfd, struct sockaddr *servaddr, socklen_t *servaddrlen);
ssize_t recvfrom(int sockfd, void *buff, size_t len, int flags,
                struct sockaddr *from, socklen_t *fromlen);
ssize_t sendto(int sockfd, void *msg, size_t len, int flags,
              struct sockaddr *to, socklen_t tolen);
struct hostent gethostbyname(const char *name);
int shutdown(int sockfd, int howto);
<sys/socket.h>
<netinet/in.h>
<netdb.h>
```