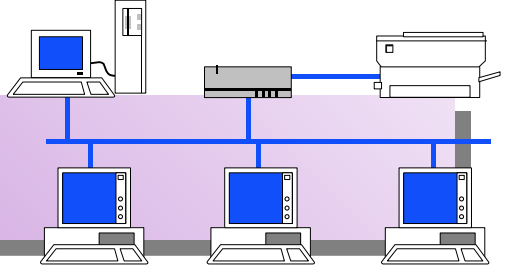
Unix Network Programming (2nd Edition)

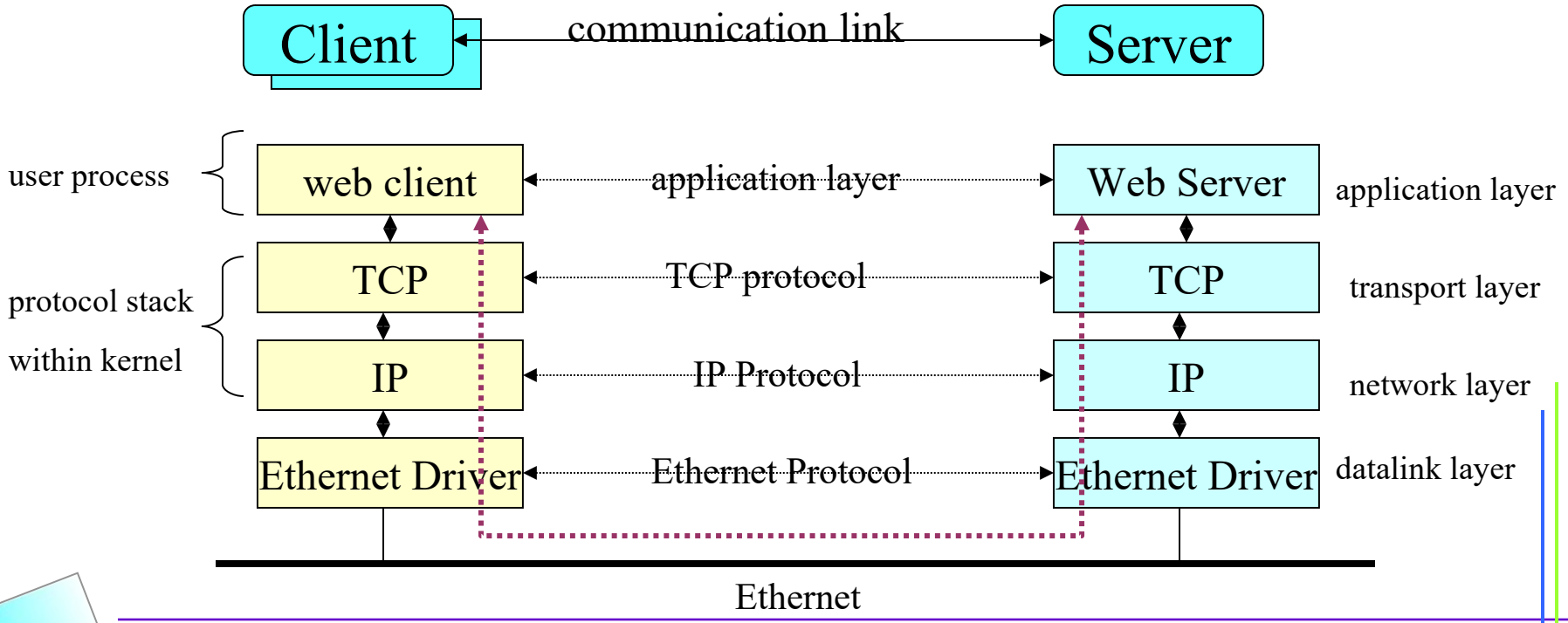
Part 1. Introduction and TCP/IP

Chapter 1. Introduction

Introduction



- **Most network application divided into**
  - a ***client*** and a ***server***



Simple Daytime Client

```
1.      #include  "unp.h"

2.      int
3.      main(int argc, char **argv)
4.      {
5.        int        sockfd, n;
6.        char       recvline[MAXLINE + 1];
7.        struct sockaddr_in  servaddr;

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

10.       if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
11.         err_sys("socket error");

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

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

19.       while ( (n = read(sockfd, recvline, MAXLINE)) > 0) {
20.         recvline[n] = 0;  /* null terminate */
21.         if (fputs(recvline, stdout) == EOF)
22.           err_sys("fputs error");
23.       }
24.       if (n < 0)
25.         err_sys("read error");

26.       exit(0);
27.     }
```

Create TCP Socket

Specify server's IP address and port

Establish connection with server

Read and display server's reply

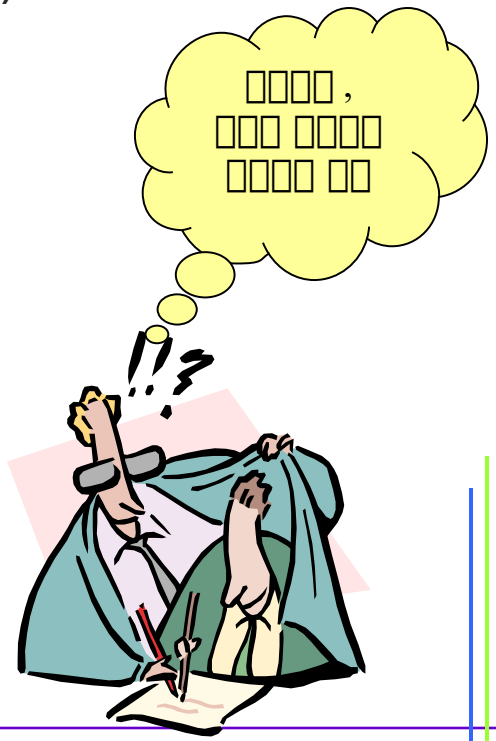
IPv6 Client

```
1.      #include  "unp.h"

2.      int
3.      main(int argc, char **argv)
4.      {
5.        int         sockfd, n;
6.        struct sockaddr_in6 servaddr;
7.        char        recvline[MAXLINE + 1];

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

10.       if ( (sockfd = socket(AF_INET6, SOCK_STREAM, 0)) < 0)
11.         err_sys("socket error");

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

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

19.       while ( (n = read(sockfd, recvline, MAXLINE)) > 0) {
20.         recvline[n] = 0;  /* null terminate */
21.         if (fputs(recvline, stdout) == EOF)
22.           err_sys("fputs error");
23.       }
24.       if (n < 0)
25.         err_sys("read error");

26.       exit(0);
27.     }
```

Error Handling (Wrapper Functions)

- **Performs the actual function call, tests the return value, and terminates on an error**
  - sockfd = Socket(AF_INET, SOCK_STREAM,0)

```
int
Socket(int family, int type, int protocol)
{
        int n;
        if (( n=socket(familty, type, protocol)) < 0)
                err_sys("Socket error");
        return(n);
}
```



A Simple Daytime Server

```
1.  #include    "unp.h"
2.  #include    <time.h>
3.  int
4.  main(int argc, char **argv)
5.  {
6.              int                 listenfd, connfd;
7.              struct sockaddr_in  servaddr;
8.              char                buff[MAXLINE];
9.              time_t              ticks;
10.             listenfd = Socket(AF_INET, SOCK_STREAM, 0);
11.             bzero(&servaddr, sizeof(servaddr));
12.             servaddr.sin_family      = AF_INET;
13.             servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
14.             servaddr.sin_port        = htons(13);    /* daytime
    server */
15.             Bind(listenfd, (SA *) &servaddr, sizeof(servaddr));
16.             Listen(listenfd, LISTENQ);
17.             for ( ; ; ) {
18.                     connfd = Accept(listenfd, (SA *) NULL,
    NULL);
19.         ticks = time(NULL);
20.         snprintf(buff, sizeof(buff), "%.24s\r\n", ctime(&ticks));
21.         Write(connfd, buff, strlen(buff));
22.             Close(connfd);
23.             }
24. }
```

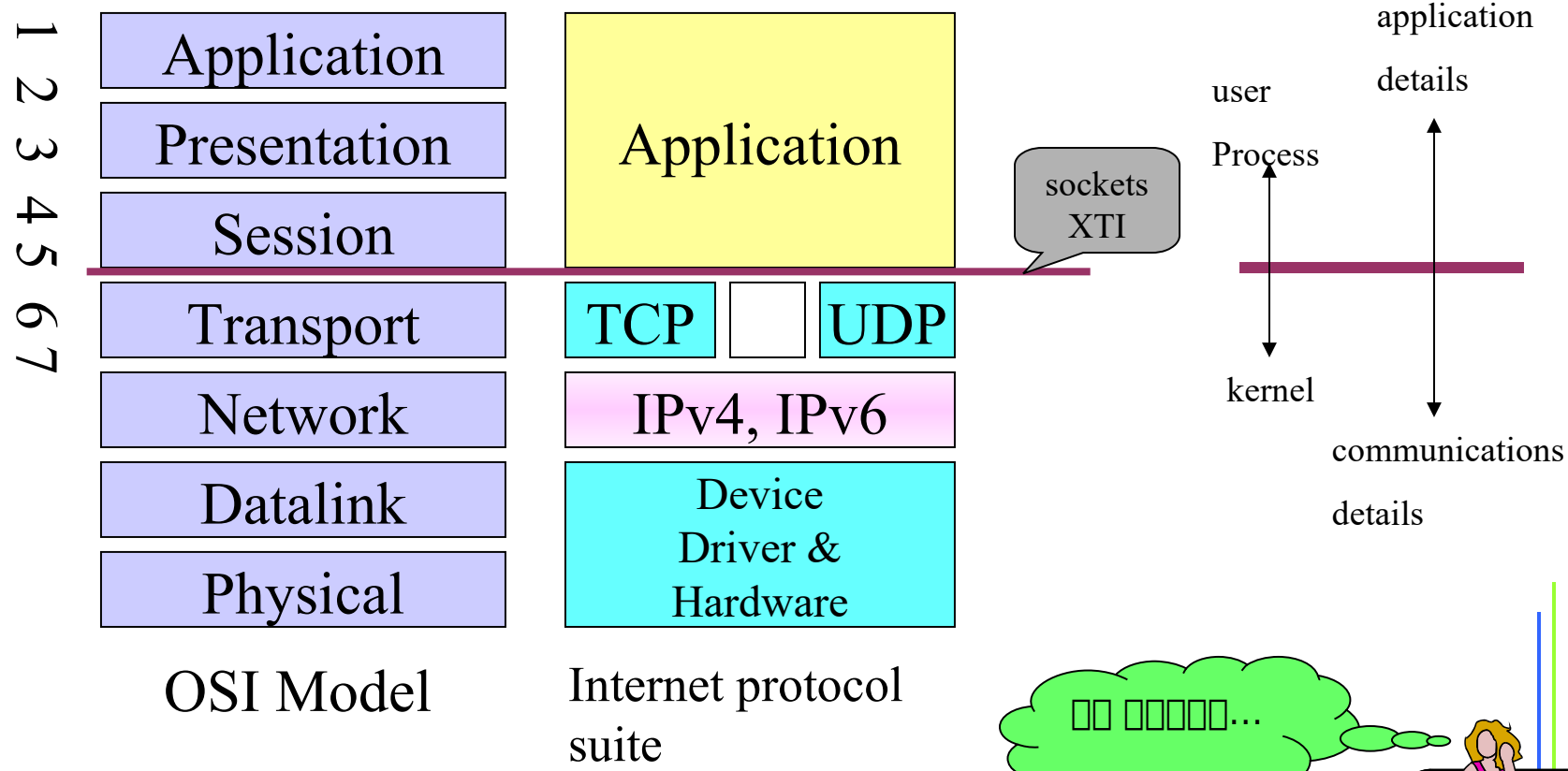
Create Socket

Bind server's well-known port to socket

Convert socket to listening socket

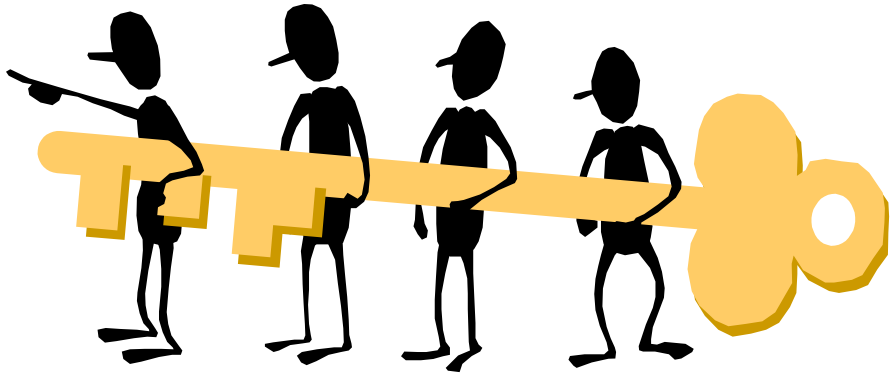
Accept client connection, send reply & close

OSI Model (Layers in OSI model and Internet protocol suite)



Unix Standards

- **POSIX**
  - acronym for “Portable Operation System Interface”
  - a family of standards being developed by the IEEE, also adopted as by ISO/IEC
  - http://www.pasc.org/standing/sd11.html
- **The Open Group**
  - X/Open Company and Open Software Foundations
  - XPG3(X/Open Portability Guide, Issue 3)
  - http://www.opengroup.org/public/tech/unix/version2
- **IETF(Internet Engineering Task Force)**
- **Unix Versions and Portability**
  - focus on Posix.1g



Unix Network Programming (2nd Edition)

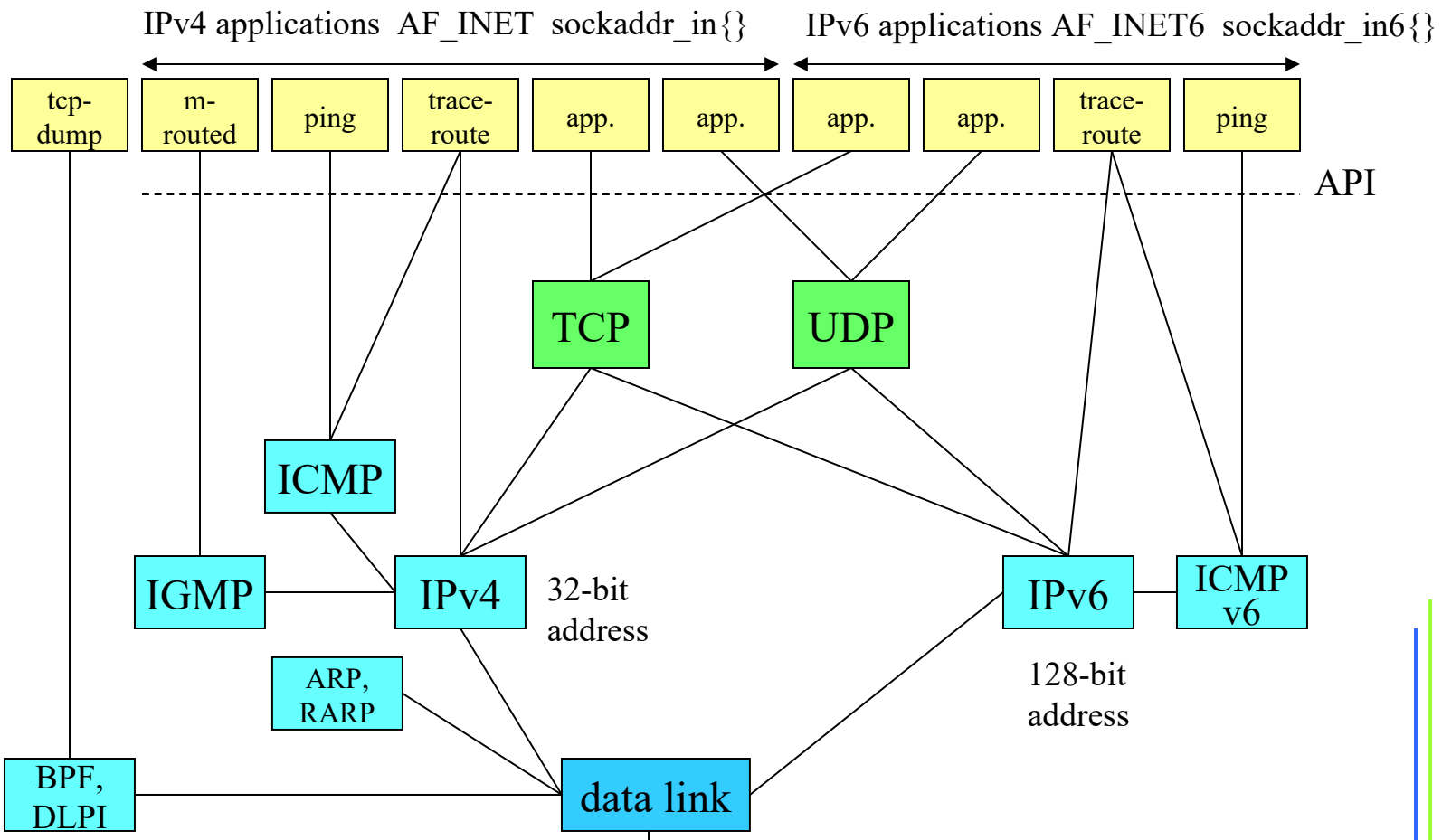
Part 1. Introduction and TCP/IP

Chapter 2.
The Transport Layer: TCP and UDP

Introduction

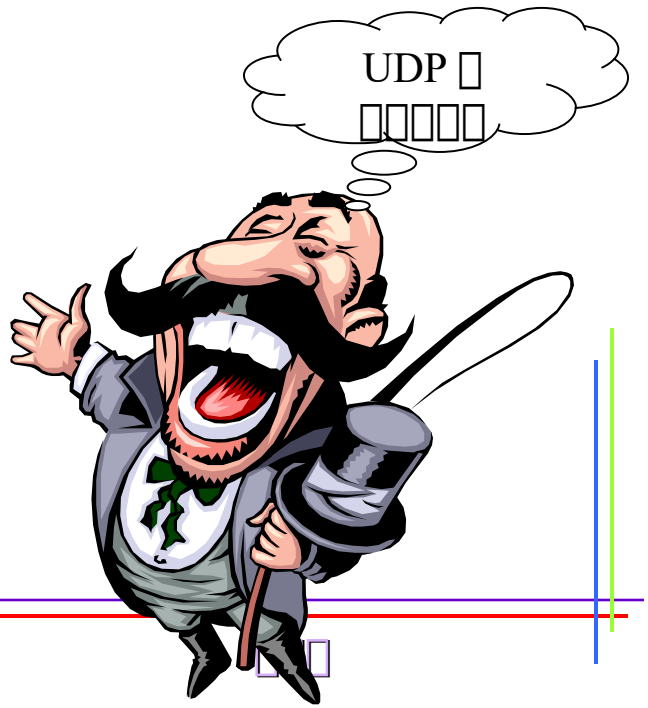
- **GOAL**
  - to provide enough ***detail to understand*** how to use the protocols from a network programming perspective
  - provide ***references to more detailed descriptions*** of the actual design, implementation, and history of the protocols
- **TCP**
  - to write robust clients and server
  - sophisticated, byte-stream protocol
- **UDP**
  - simple, unreliable, datagram protocol

The Big Picture



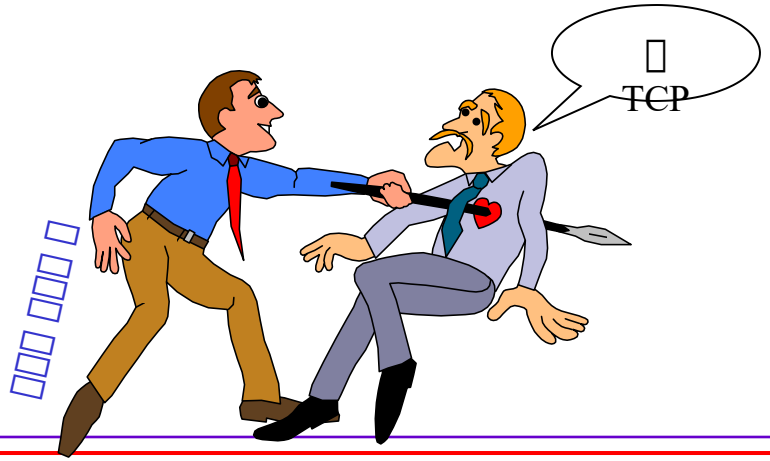
UDP (User Datagram Protocol)

- *simple* **transport-layer protocol**
- **App writes a** *datagram* **to a UDP socket**
- *no guarantee* **that a UDP datagram ever reaches its final destination**
- *lack of reliability*
- **UDP datagram has a** *length*
- **a** *connectionless* **services**

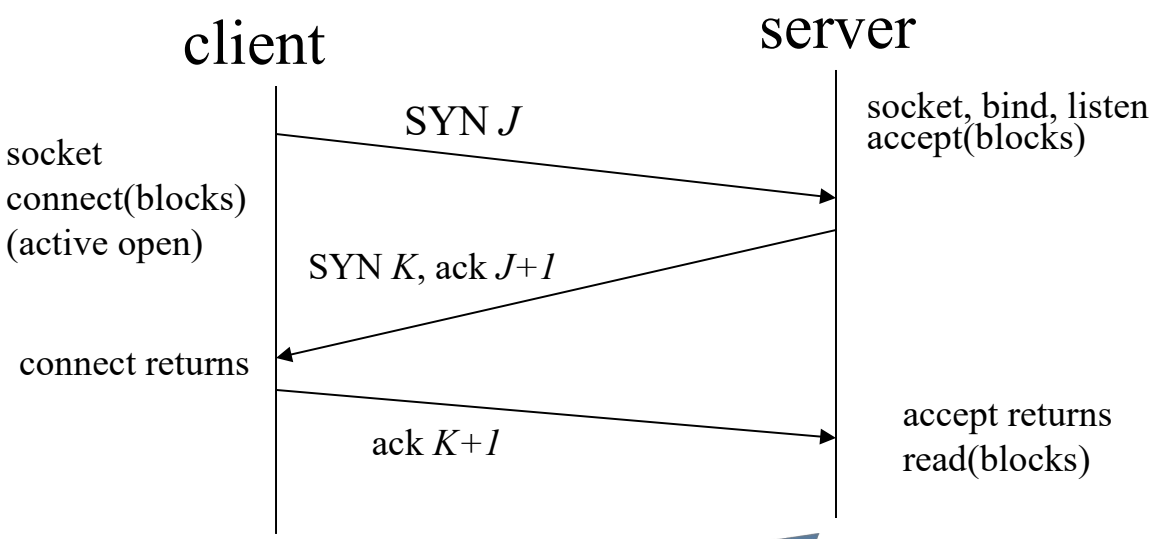


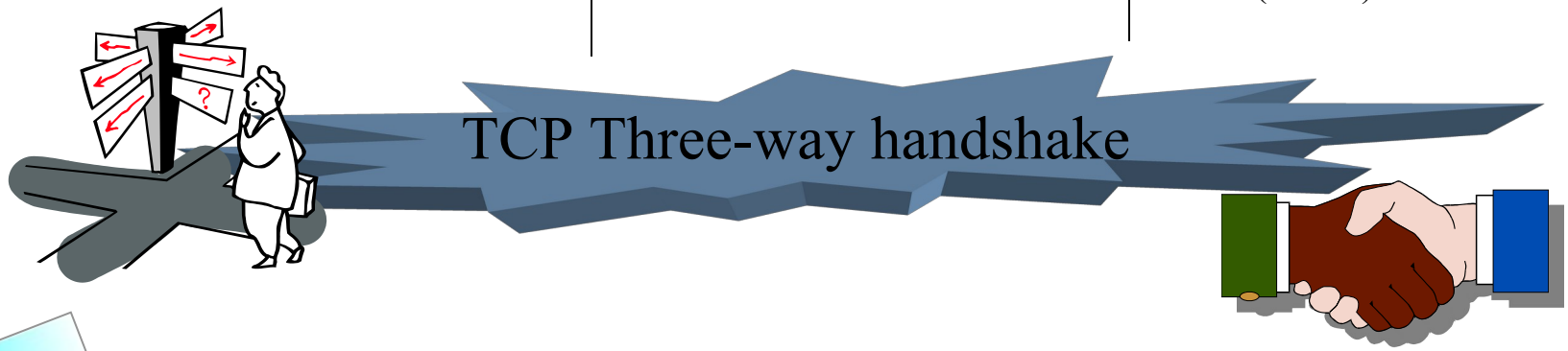
TCP (Transmission Control Protocol)

- **Provide** *connections* **between clients and servers**
- **Provide** *reliability*
- **also** *sequences* **the data by associating a sequence number with every byte**
- **Provide** *flow control*
- *full-duplex*

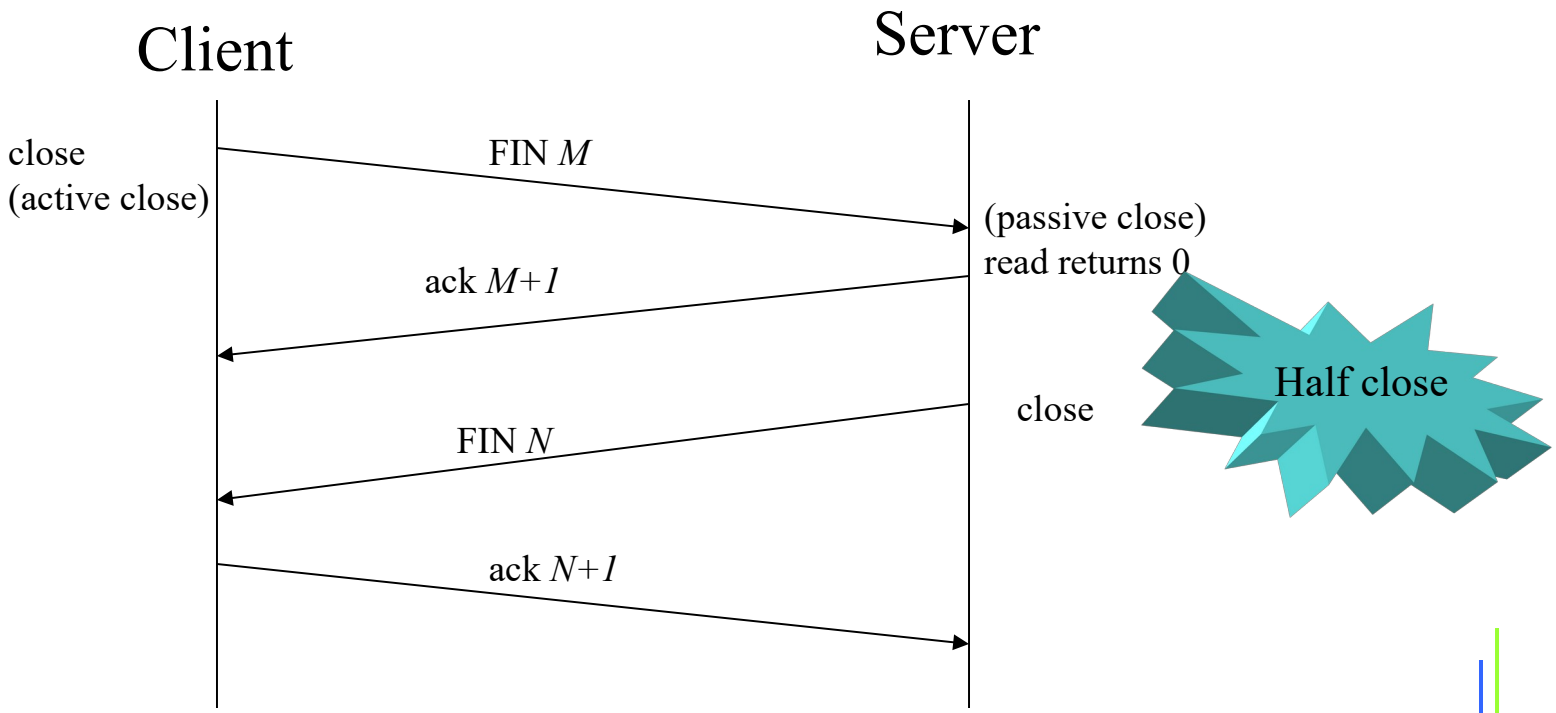


TCP Connection Establishment



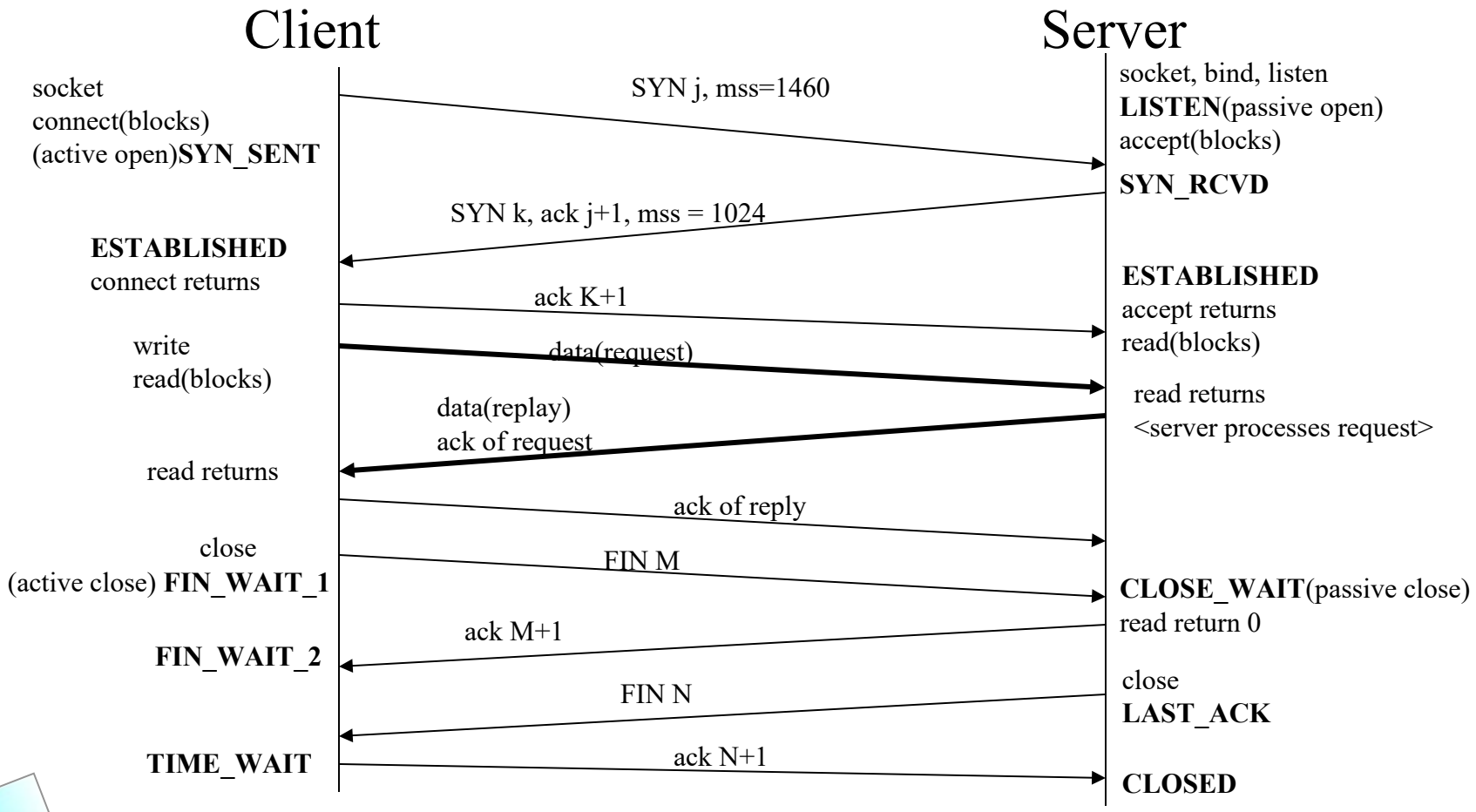


TCP Connection Termination

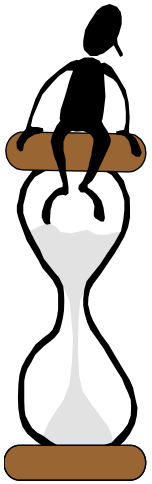


Packets exchanged when a TCP connection is close

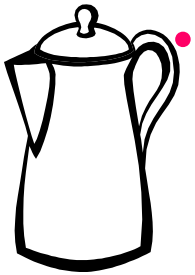
Watching the Packets



TIME_WAIT State & Port Number

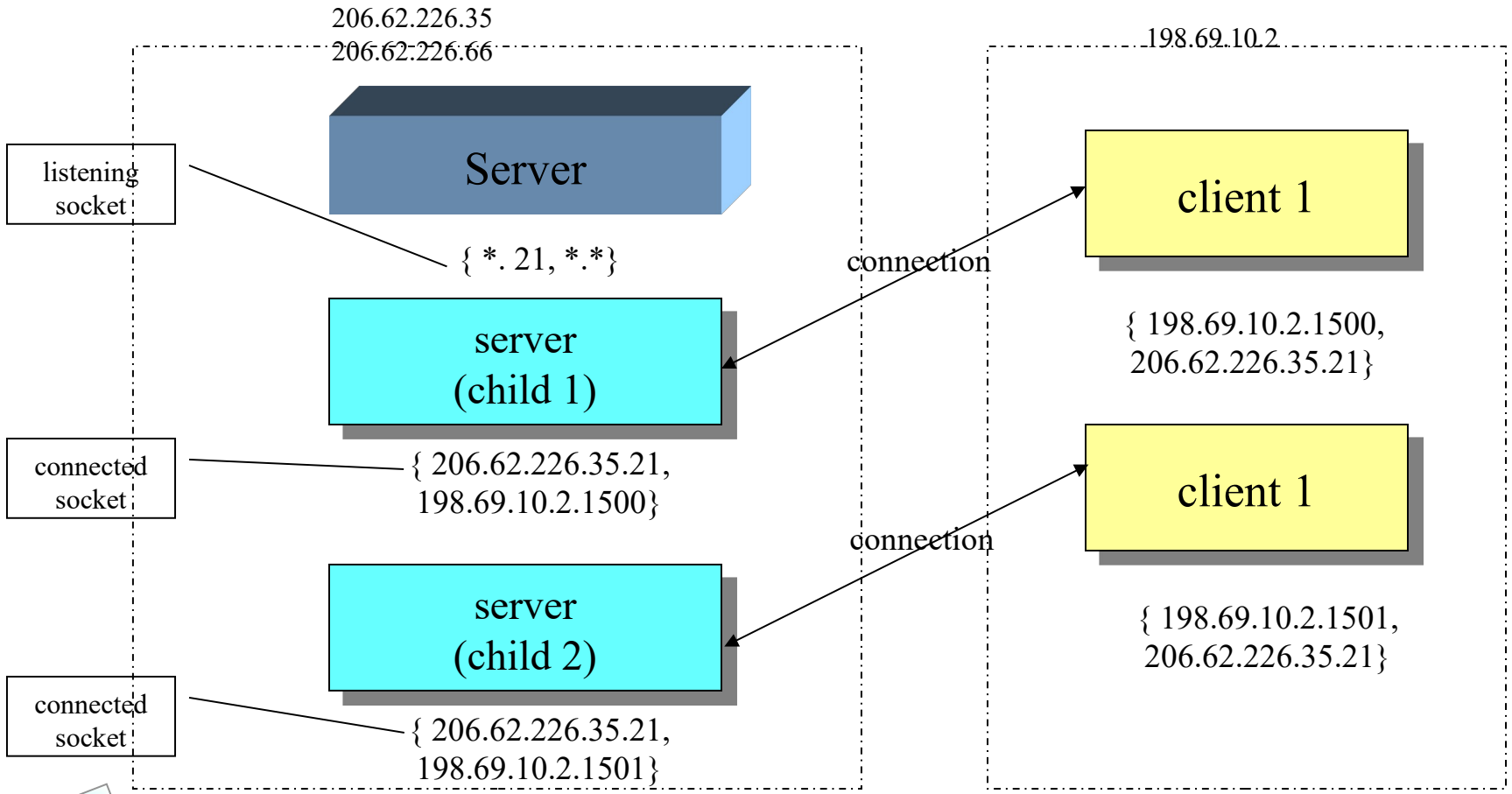


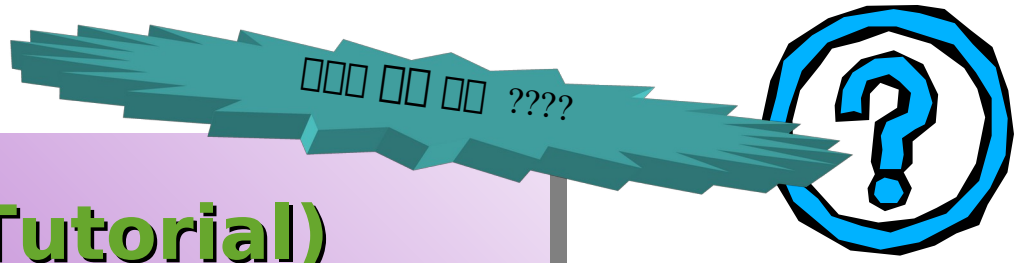
- **TIME_WAIT State**
  - MSL(Maximum segment life time)
    - † RFC 1122, about 2 min.
  - Two Reasons
    - † to implement TCP's full-duplex connection termination reliably
    - † to allow old duplicate segments to expire in the network



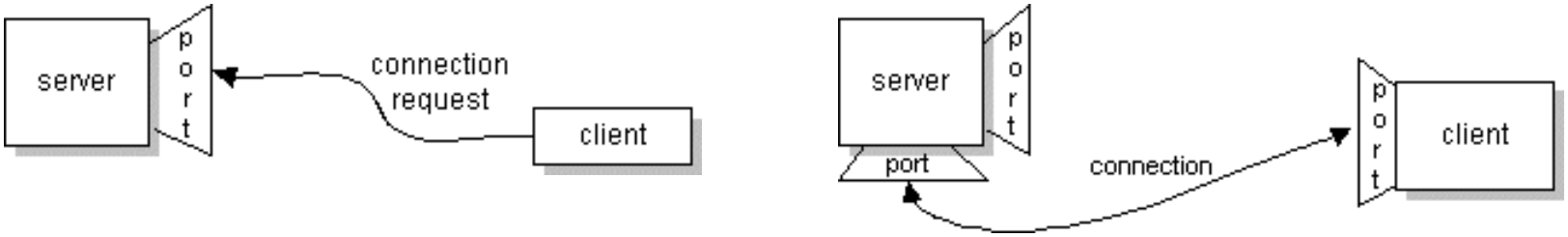
- **Port Number (IANA)**
  - well-known ports : 0 ~ 1023
  - registered port : 1024 ~ 49151
  - dynamic or private ports : 49152 ~ 65535
- **Socket Pair**
  - local IP add., local TCP port, foreign IP add., foreign port

Concurrent Server





□□□ (from Java Tutorial)



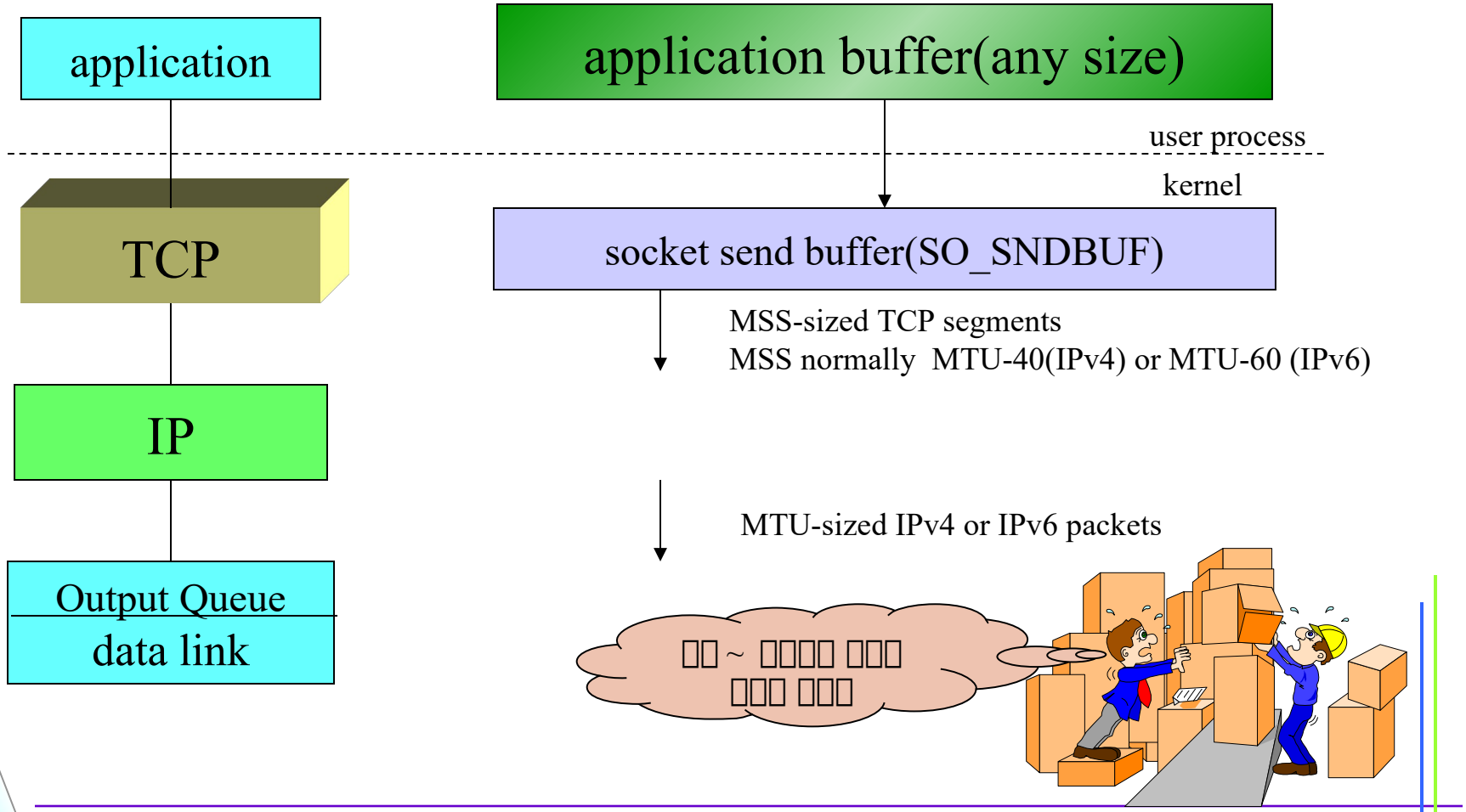
Normally, a server runs on a specific computer and has a socket that is bound to a specific port number. **The server just waits, listening to the socket for a client to make a connection request. On the client-side: The client knows the hostname of the machine on which the server is running and the port number to which the server is connected. To make a connection request, the client tries to rendezvous with the server on the server's machine and port.**

If everything goes well, the server accepts the connection**. Upon acceptance,** *the server gets a new socket bound to a different port*. ***It*** *needs a new socket* ***(and consequently a different port number) so that*** *it can continue to listen to the original socket for connection requests while tending to the needs of the connected client.*

On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server. Note that the socket on the client side is not bound to the port number used to rendezvous with the server. Rather, the client is assigned a port number local to the machine on which the client is running. The client and server can now communicate by writing to or reading from their sockets.

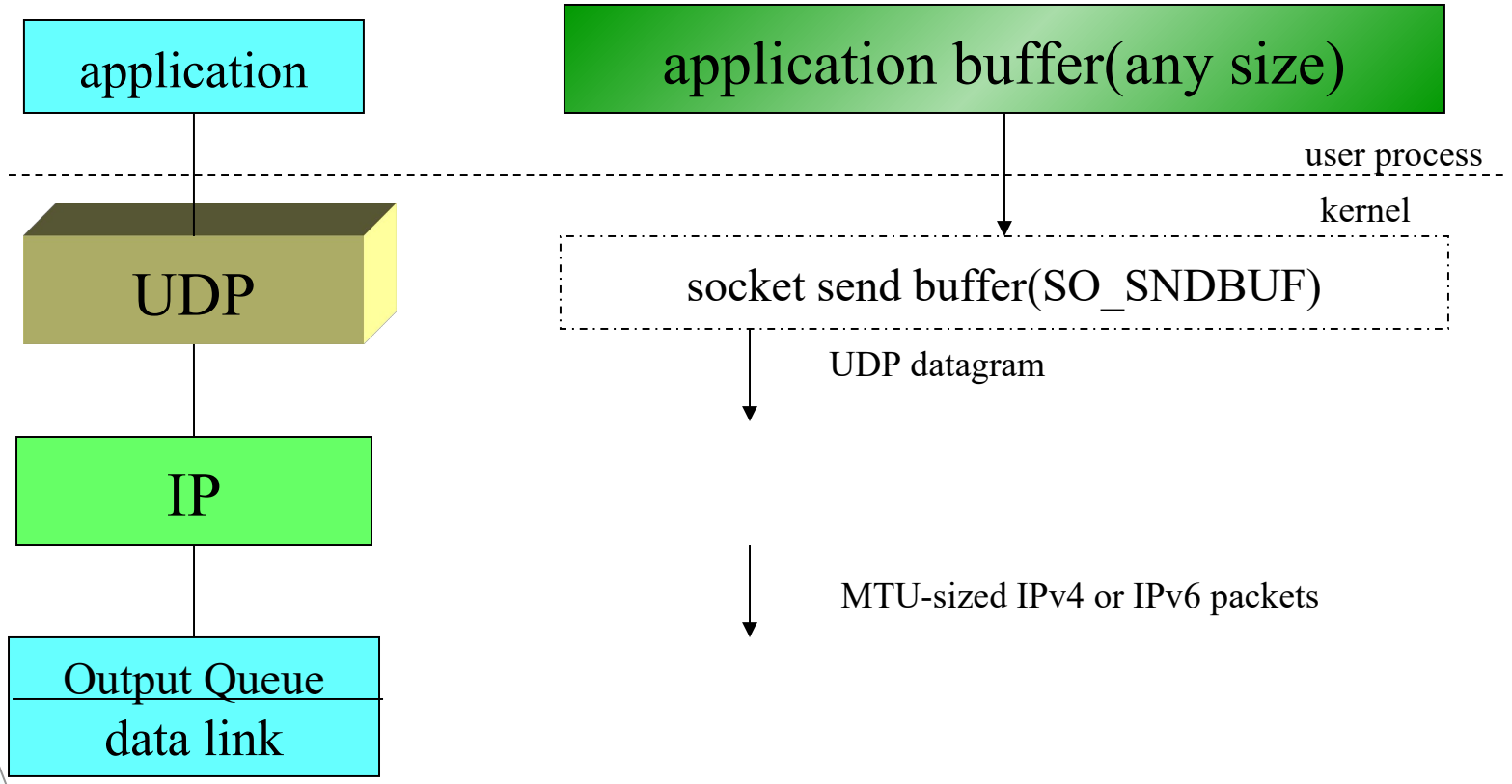
TCP Output

(Steps and buffers involved when application writes to a TCP socket)



UDP Output

(Steps and buffers involved when application writes to a UDP socket)



UNIX Network Programming (chapter 3 ~ 5)

Agenda

- Socket Address Structures
- Socket Functions
- TCP Socket
- POSIX □□□□
- Server Source Example
- Client Source Example

Socket Address Structures

Basic

```
struct sockaddr {
    uint8_t       sa_len;       /* for variable socket address structure */
    sa_family_t sa_family;      /* address family */
    char          sa_data[14];  /* protocol-specific address */
};
```

IPv4

```
struct in_addr {
    in_addr_t       s_addr;         /* 32 bit IPv4 */
};
struct sockaddr_in {
    uint8_t         sin_len;        /* value = 16 */
    sa_family_t     sin_family;
    in_port_t       sin_port;       /* 16 bit */
    struct in_addr  sin_addr;       /* 32 bit IPv4 */
    char            sin_zero[8];    /* unused */
};
```

IPv6

```
struct in6_addr {
    uint8_t         s6_addr[16];    /* 128 bit IPv6 */
};
struct sockaddr_in6 {
    uint8_t         sin6_len;       /* 16 */
    sa_family_t     sin6_family;
    in_port_t       sin6_port;
    uint32_t        sin6_flowinfo; /* priority & flow */
    struct in6_addr sin6_addr;      /* 128 bit IPv6 */
};
```

Socket Functions (1)

뒤에 자세히 봅시다 .

int socket(int family, int type, int protocol);

int bind(int sockfd, struct sockaddr *addrptr, int addrlen);

int connect(int sockfd, struct sockaddr *addrptr, int addrlen);

int listen(int sockfd, int backlog);

int accept(int sockfd, struct sockaddr *addrptr, int *addrlen);

int close(int sockfd);

Socket address 정보를 얻는 함수

int getsocketname(int sockfd, struct sockaddr *loacladdr, socklen_t *addrlen);

int getpeername (int sockfd, struct sockaddr *peeraddr, socklen_t *addrlen);

바이트 조작 함수

void bzero(void #dest, size_t nbytes);

void bcopy(const void *src, void *dest, size_t nbytes);

int bcmp(const void *ptr1, const void *ptr2, size_t nbytes);

주소 변환 함수

IPv4

int inet_aton(const char *strptr, struct in_addr *addrptr);

char *inet_ntoa(struct in_addr inaddr);

IPv4 and IPv6

int inet_pton(int family, const char *strptr, void *addrptr);

const char *inet_ntop(int family, const void *addrptr, char *strptr, size_t len);

Socket Functions (2)

바이트 순서 함수

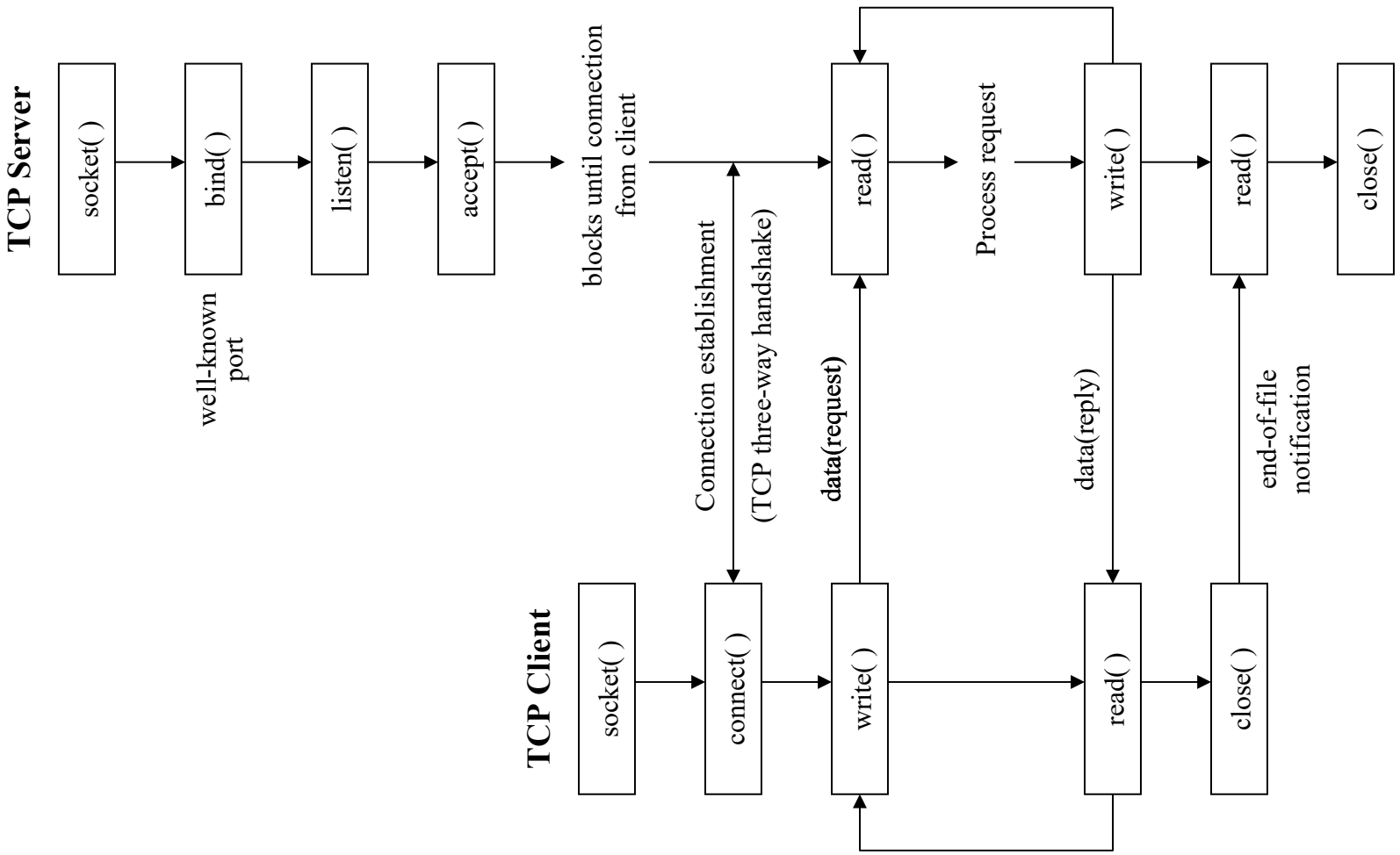
```
uint32_t htonl( uint32_t hostlong );
uint32_t ntohl( uint32_t netlong );
uint16_t htons( uint16_t hostshort );
uint16_t ntohs( uint16_t hostshort );
```

Network byte 순서는
Big-endian 이다 .

입출력 함수

```
ssize_t readn( int filedes, void *buff, size_t nbytes );
ssize_t writen( int filedes, const void *buff, size_t nbytes );
ssize_t readline( int filedes, void buff, size_t maxlen );
```

TCP Socket


TCP Server
socket( )
bind( )
listen( )
accept( )
blocks until connection from client
read( )
Process request
write( )
read( )
close( )
well-known port
Connection establishment
(TCP three-way handshake)
data(request)
data(reply)
end-of-file notification
TCP Client
socket( )
connect( )
write( )
read( )
close( )

socket()

int socket(int family, int type, int protocol);

family	Description
AF_INET	IPv4 protocols
AF_INET6	IPv6 protocols
AF_LOCAL	Unix domain protocols
AF_ROUTE	Routing sockets
AF_KEY	Key sockets

< Address family >

type	Description
SOCK_STREAM	stream socket
SOCK_DGRAM	datagram socket
SOCK_RAW	raw socket

< Socket Type >

	AF_INET	AF_INET6	AF_LOCAL	AF_ROUTE	AF_KEY
SOCK_STREAM	TCP	TCP	Yes		
SOCK_DGRAM	UDP	UDP	Yes		
SOCK_RAW	IPv4	IPv6		Yes	Yes

☞ protocol argument 는 raw socket 을 제외하고는 0 으로 정한다 .

☞ key socket : kernel 의 암호화 key table 에의 interface 이다 .

☞ raw socket : TCP 나 UDP 의 header 없이 바로 IP header 를 붙여서 사용한다 .

connect(), bind()

int connect(int sockfd, const struct sockaddr *servaddr, socklen_t addrlen);

☞ client 가 remote socket 혹은 server 로 연결을 설정하기 위하여 사용

☞ three-way handshake 사용

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

☞ 생성된 socket 에 myaddr 구조체로 주어진 주소를 부여

☞ 보통 server 가 자신이 제공하는 service port 를 지정하기 위하여 사용

☞ client 의 경우 bind() 시스템 호출을 사용하지 않으면 kernel 이 사용하지 않는 port 를 자동적으로 할당

일반적으로 binding 하지 않는다 .

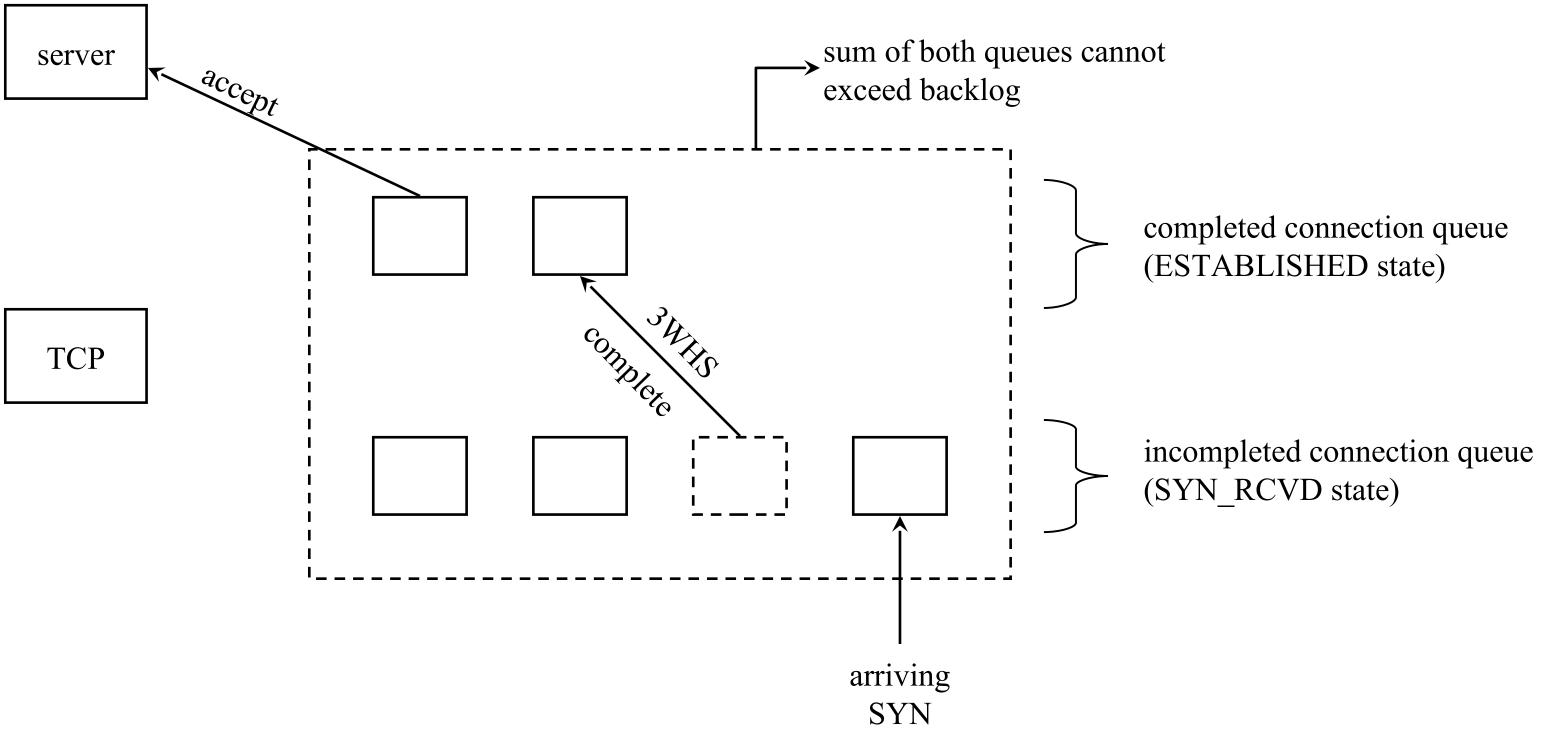
Process specifies		Result
IP address	port	
wildcard	0	kernel chooses IP address and port
wildcard	nonzero	kernel chooses IP address, process specifies port
local IP address	0	process specifies IP address, kernel chooses port
local IP address	nonzero	process specifies IP address and port

listen()

int listen(int sockfd, int backlog);

☞ server 의 경우만 사용

☞ server program 의 경우 자신이 server 로 동작하는 socket 을 사용하고 있음을 listen() system call 을 사용하여 kernel 에 알린다 .



accept(), close()

int accept(int sockfd, struct sockaddr *cliaddr, socklen_t *addrlen);

☞ incoming connection request 를 접수

☞ cliaddr 에 accept 된 client 의 주소를 받아온다 .

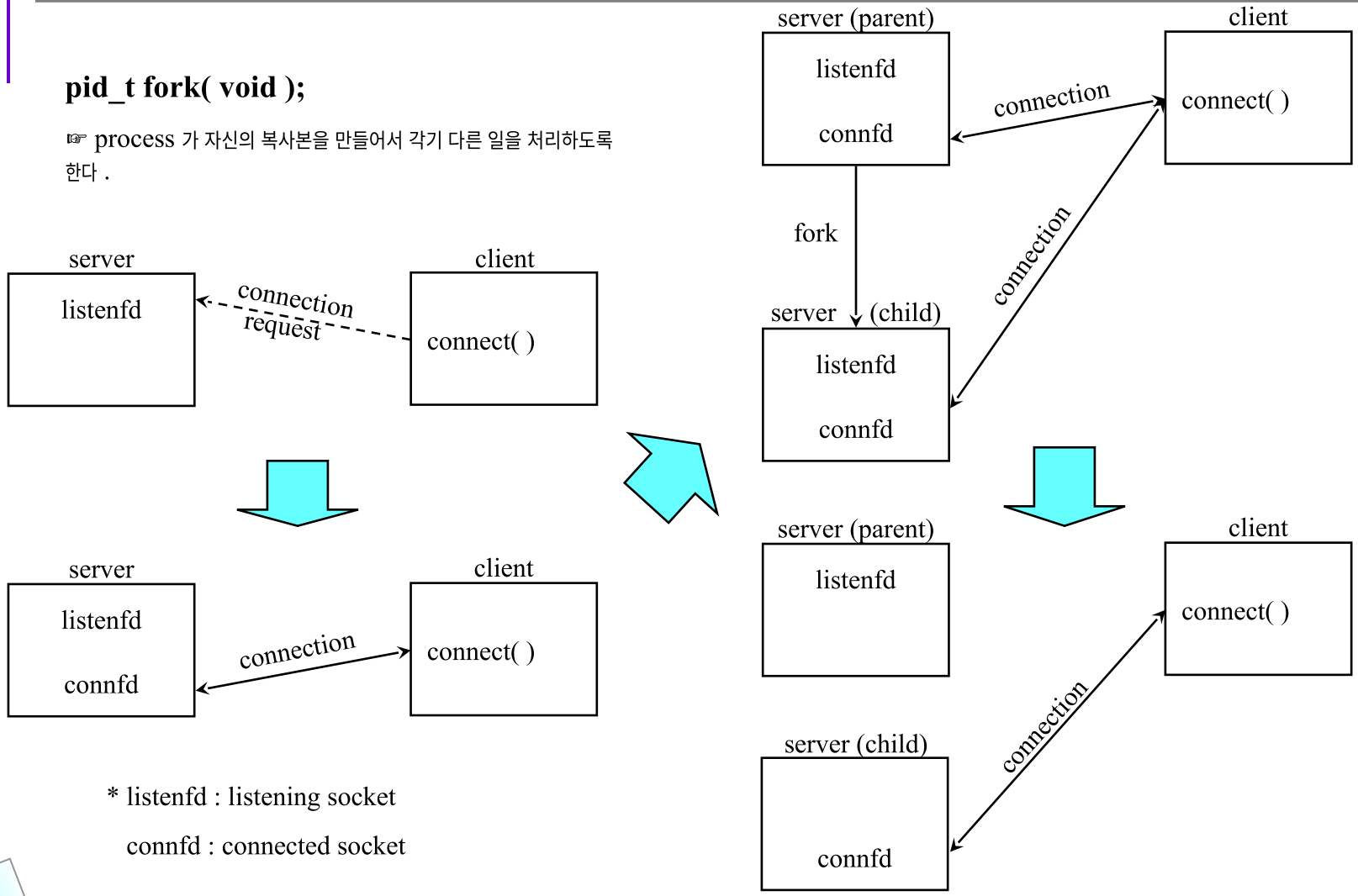
int close(int sockfd);

☞ socket 을 정상적인 종료 방식으로 종료한다 .

fork ()

pid_t fork(void);

☞ process 가 자신의 복사본을 만들어서 각기 다른 일을 처리하도록 한다 .



* listenfd : listening socket

connfd : connected socket

POSIX 신호처리

☞ 모든 신호는 자신과 연계된 disposition 을 가지며 , 때로는 이를 action 이라고도 부른다 .

☞ 신호의 disposition 을 sigaction 함수를 호출하여 설정한다 .

1) 특정 신호가 발생할 때마다 호출할 함수를 제공한다 . SIGKILL 과 SIGSTOP 은 포착할 수 없는 두 개의 신호이다 .

2) 계획을 SIG_IGN 으로 설정함으로써 신호를 무시할 수 있다 . SIGKILL 과 SIGSTOP 은 무시할 수 없다 .

3) 계획을 SIG_DFL 로 설정함으로써 신호의 default disposition 을 설정할 수 있다 . 기본은 보통 신호를 받으면 process 를 종료시키는 것이며 , 어떤 신호에서는 process 의 memory 내용을 그대로 현재의 working directory 에 옮겨놓기도 한다 .

```
typedef void Sigfunc( int );
Sigfunc *signal( int signo, Sigfunc *func );
{
                struct sigaction act, oact;

                act.sa_handler = func;
                sigemptyset( &act.sa_mask );
                act.sa_flags = 0;

                if ( sigaction( signo, &act, &oact ) < 0 )
                                return ( SIG_ERR );
                return ( oact.sa_handler );
}
```

signo = signal 이름

func = 함수를 가리키는 지시자 또는 SIG_IGN , SIG_DFL

SIGCHLD signal

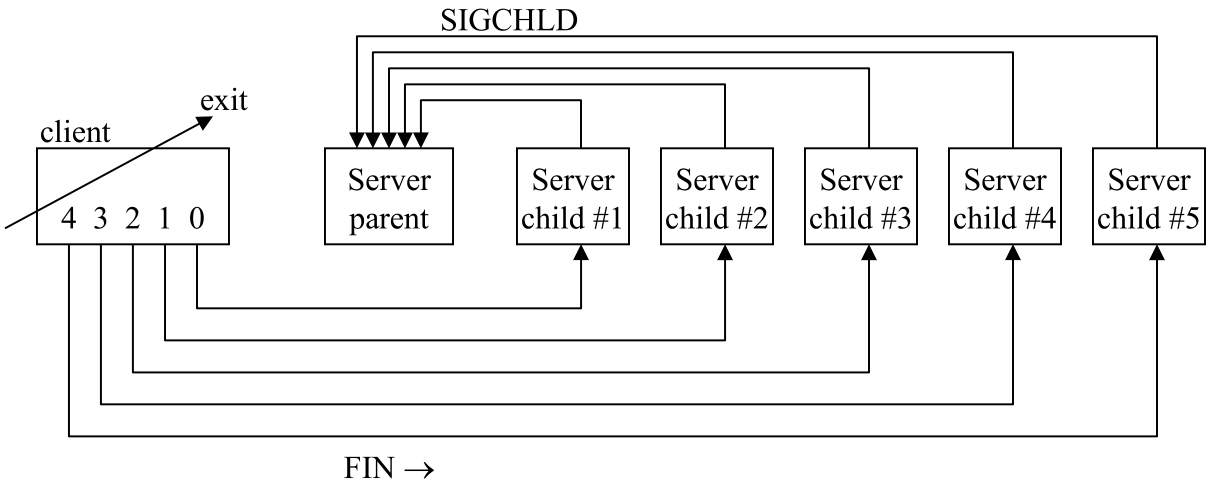
☞ Process 가 종료될 때마다 kernel 이 종료하는 process 의 parent 에게 보내는 것이다 .

☞ zombie 상태의 child 를 제거하기 위해서는 wait 함수를 수행해야 하고 만약 server 와 여러 연결을 설정한 경우는 waitpid 함수로 각각을 지정하여 종료되도록 한다 .

pid_t wait(int *statloc);

pid_t waitpid(pid_t pid, int *statloc, int options);

→ option 의 경우 " WNOHANG" 을 일반적으로 선택하는데 이는 kernel 에게 종료될 child process 가 없으면 blocking 하지 말도록 하며 , 아직 종료되지 않은 child process 가 있으면 waitpid 가 blocking 하지 않도록 한다 .



< wait 사용시 >

PID	TT	STAT	TIME	COMMAND
21282	p1	S	0:00.09	./tcpserv03
21284	p1	Z	0:00.00	(tcpserv03)
21285	p1	Z	0:00.00	(tcpserv03)
21286	p1	Z	0:00.00	(tcpserv03)
21287	p1	Z	0:00.00	(tcpserv03)

SIGPIPE signal

- ☞ sever 가 먼저 종료되어 버린 경우 server TCP 가 client 로부터 data 를 받으면 process 가 종료되었기 때문에 RST 을 보낸다 .
- ☞ process 가 RST 을 받은 socket 에 data 를 쓰면 write 는 EPIPE 를 돌려준다 .
- ☞ 이를 적절히 처리해 주지 않으면 “ server terminated prematurely” 와 같은 error message 와 함께 종료된다 ..

Server source example (1)

```
int main( int argc, char **argv )
{
        int listenfd, connfd;
        pid_t childpid;
        socklen_t clilen;
        struct sockaddr_in cliaddr, seraddr;
        void sig_chld( int );

        listenfd = Socket( AF_INET, SOCK_STREAM, 0 );

        bzero( &servaddr, sizeof(servaddr) );
        servaddr.sin_family = AF_INET;
        servaddr.sin_addr.s_addr = htonl( INADDR_ANY );
        servaddr.sin_port = htons( SERV_PORT );

        Bind( listenfd, (SA *)&servaddr, sizeof(servaddr) );
        Listen( listenfd, LISTENQ );
        Signal( SIGCHLD, sig_chld );

        for ( ; ; ) {
              clilen = sizeof (cliaddr );
```

server 의 주소등록

SERV_PORT 는 지정된 port number

child process 종료시 수신되는 SIGCHLD signal 을 하도록 설정

Server source example (2)

```
            if ( (connfd = accept( listenfd, (SA *)&cliaddr, &clilen ) ) < 0 ) {
                if ( errno == EINTR )
                    continue;
                else
                    err_sys( "accept error" );
            }
            if ( (childpid = Fork() ) == 0 ) {
                Close( listenfd );              /* child closes listening socket */
                str_echo( connfd );             /* do it all */
                exit(0);
            }
            Close( connfd );                    /* parent closes connected socket */
        }
    }

    void sig_chld( int signo )
    {
        pid_t pid;
        int stat;

        while ( ( pid = waitpid( -1, &stat, WNOHANG ) ) > 0 )
                printf( "child %d terminated\n", pid );
        return;
    }
```

slow system call(여기서는 accept) 이 EINTR(일시 중지된 시스템호출) 을 돌려주는데 그걸 받아서 처리

Client source example

```
int main( int argc, char **argv )
{
    int i, sockfd[5];
    struct sockaddr_in seraddr;

    if ( argc != 2 )
        err_quit( “usage: tcpcli <IPaddress>” );

    for ( i=0; i<5; i++ ) {
        sockfd[i] = Socket( AF_INET, SOCK_STREAM, 0 );

        bzero( &servaddr, sizeof(servaddr) );
        servaddr.sin_family = AF_INET;
        servaddr.sin_port = htons( SERV_PORT );
        Inet_pton( AF_INET, argv[1], &servaddr.sin_addr );

        Connect( sockfd[i], (SA *)&servaddr, sizeof( servaddr ) );
    }

    str_cli( stdin, sockfd[0] );                    /* do it all */

    exit(0);
}
```

□□□ server □□□ □□□□ .

Chapter 6 *I/O Multiplexing: The* select *and* poll *Functions*

Introduction

- **I/O Multiplexing** 은 **select, poll, pselect** 함수를 통해 제공됨
- **I/O Multiplexing** 이 사용되는 예
  - client 가 여러 개의 descriptor(예를 들면 , interactive input 과 network socket) 을 관리하는 경우
  - client 가 동시에 여러 개의 socket 을 관리해야 하는 경우
  - TCP 서버가 listening socket 과 connected socket 을 관리하는 경우
  - 서버가 TCP 와 UDP 를 동시에 사용하는 경우
  - 서버가 동시에 여러 개의 서비스와 여러 개의 프로토콜을 관리해야 하는 경우 (예 : inetd)

I/O Models

- **blocking I/O**
- **nonblocking I/O**
- **I/O multiplexing (select and poll)**
- **signal-driven I/O (SIGIO)**
- **asynchronous I/O (Posix.1 aio_ functions)**

※ 각 **I/O** 모델에 대한 설명은 **Text** 의 **Fig 6.1~6.6** 을 참고

select Function

- **Allows the process to instruct the kernel to wait for any one of multiple events to occur and to wake up the process only when one or more of these events occurs or when a specified amount of time has passed.**

```
#include <sys/select.h>
#include <sys/time.h>

int select(int maxfdp1, fd_set *readset, fd_set  *writeset, fd_set *exceptset,
           const struct timeval *timeout);

                                    Returns: positive count of ready descriptors, 0 on timeout, -1 on error
```

1. Wait forever: timeout = NULL
2. Wait up to a fixed amount time: timeout specified in the timeval structure
3. Don’t wait at all: timeout = 0

select Function (cont'd)

- **fd_set** 제어 매크로
  - void FD_ZERO(fd_set *fdset);
  - void FD_SET(int fd, fd_set *fdset);
  - void FD_CLR(int fd, fd_set *fdset);
  - int FD_ISSET(int fd, fd_set *fdset);
- **Under What Conditions Is a Descriptor Ready?**
  - Ready for reading if…
    - † 소켓의 receive buffer 의 데이터 바이트수 ≥ low-water mark for receive buf. (디폴트 1) (read 연산의 return 값 > 0)
    - † 연결의 read-half 가 닫힌 경우 (read 연산의 return 값 = 0)
    - † listening socket 인 경우 , 완료된 연결의 수가 0 이상인 경우
    - † pending error 발생 (read 연산의 리턴 값 = -1)

select Function (cont'd)

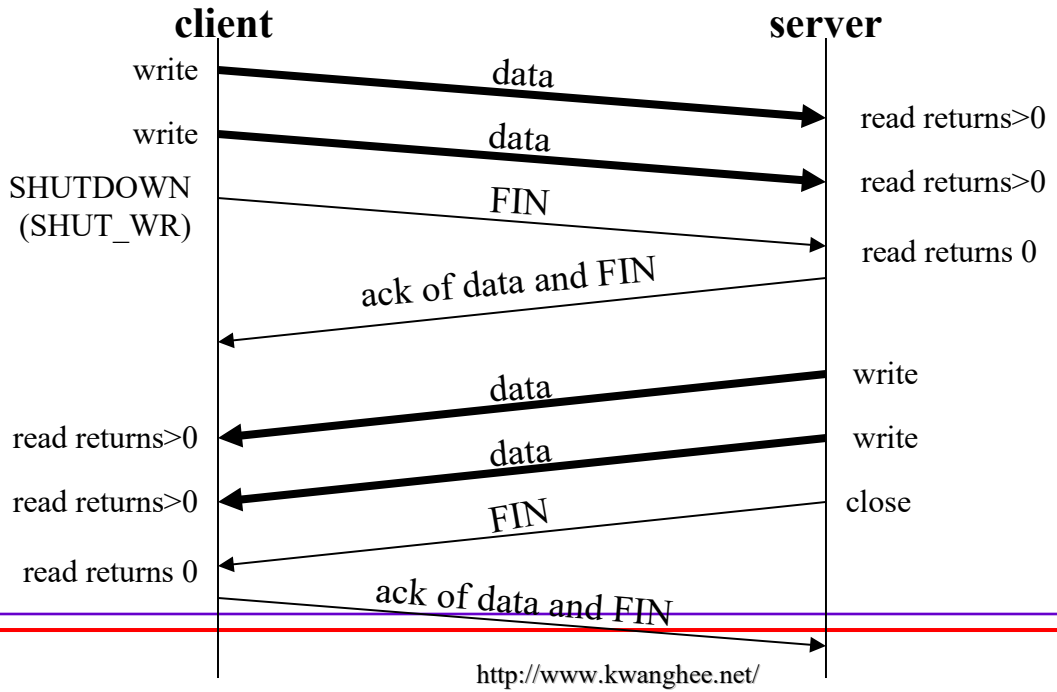
- Ready for writing if…
  - socket의 send buffer의 데이터 바이트수 ≥ low-water mark for send buf. (디폴트 2048)이고, 소켓이 연결 중이거나 UDP인 경우 (write 연산의 리턴 값 > 0)
  - 연결의 write-half가 닫힌 경우 (write 연산 때 SIGPIPE 발생)
  - pending error 발생 (write 연산의 리턴 값 = -1)
- Exception condition if…
  - 소켓에 Out-of-band data가 있는 경우 (Chap. 21에서 설명 ..)

shutdown Function

```
#include <sys/socket.h>

int shutdown (int sockfd, int howto);
```

Returns: 0 if OK, -1 on error

❖ **howto □□ : SHUT_RD / SHUT_WR / SHUT_RDWR**



Example (str_cli Function–Revisited)

```
 1 #include    "unp.h"
 2
 3 void
 4 str_cli(FILE *fp, int sockfd)
 5 {
 6     int         maxfdp1, stdineof;
 7     fd_set      rset;
 8     char        sendline[MAXLINE], recvline[MAXLINE];
 9
10     stdineof = 0;
11     FD_ZERO(&rset);
12     for ( ; ; ) {
13         if (stdineof == 0)
14             FD_SET(fileno(fp), &rset);
15         FD_SET(sockfd, &rset);
16         maxfdp1 = max(fileno(fp), sockfd) + 1;
17         Select(maxfdp1, &rset, NULL, NULL, NULL);
18
19         if (FD_ISSET(sockfd, &rset)) {  /* socket is readable */
20             if (Readline(sockfd, recvline, MAXLINE) == 0) {
21                 if (stdineof == 1)
22                     return;     /* normal termination */
23                 else
24                     err_quit("str_cli: server terminated prematurely");
25             }
```

Example (str_cli Function–Revisited)

```
26
27          Fputs(recvline, stdout);
28      }
29
30      if (FD_ISSET(fileno(fp), &rset)) {  /* input is readable */
31          if (Fgets(sendline, MAXLINE, fp) == NULL) {
32              stdineof = 1;
33              Shutdown(sockfd, SHUT_WR);  /* send FIN */
34              FD_CLR(fileno(fp), &rset);
35              continue;
36          }
37
38          Writen(sockfd, sendline, strlen(sendline));
39      }
40  }
41 }
```

pselect and poll Functions

```
#include <sys/select.h>
#include <signal.h>
#include <time.h>

int pselect (int maxfdp1, fd_set *readset, fd_set *writeset, fd_set *exceptset, const struct timespec
        *timeout, const sigset_t *sigmask);
```

Returns: count of ready descriptors, 0 on timeout, -1 on error

```
#include <poll.h>

int poll (struct pollfd *fdarray, unsigned long nfds, int timeout);
```

Returns: count of ready descriptors, 0 on timeout, -1 on error

Chapter 7 *Socket Options*

Socket Option 을 get/set 하는 함수

- **getsockopt and setsockopt functions**

```
#include <sys/socket.h>

int getsockopt(int sockfd, int level, int optname, void *optval, socklen_t *optlen);
int setsockopt(int sockfd, int level, int optname, const void *optval, socklen_t *optlen);

                                                    Both return: 0 if OK, -1 on error
```

- **fnctl function**
- **ioctl function**

※ getsockopt, setsockopt 함수가 사용 가능한 소켓 옵션 정리
(Fig. 7.1 of text)

Generic Socket Options

- **SO_BROADCAST**
  - enable/disable the ability of the processto send broadcast messages
  - supported for only datagram sockets
  - broadcast message 개념을 지원하는 네트웍 (Ethernet, Token ring 등) 에서만 지원됨
  - 이 옵션이 세팅되어 있지 않을 경우 , destination address 가 broadcast address 이면 EACCES 리턴됨
- **SO_DEBUG**
  - TCP 에서만 지원
  - 커널이 TCP 에 의해 보내고 / 받는 모든 패킷들에 대한 상세 정보를 circular buffer 에 기록
  - trpt program 으로 내용 조사 가능…

Generic Socket Options (cont’d)

- **SO_DONTROUTE**
  - 라우팅 테이블을 건너뛰게 함
  - destination 이 point-to-point link 나 , shared network 이 아닌 경우에는 ENETUNREACH 가 리턴됨
  - 각각의 datagram 에 MSG_DONTROUTE flag 를 세팅하여 send/sendto/sendmsg 함수로 보내는 것과 동일
- **SO_ERROR**
  - pending error 발생시 에러 값 (E*xxx*) 을 얻어오기 위함
  - pending error 가 인지되는 2 가지 방법
    - select call 에서 block 되어 있다가 – 1 값으로 return 함
    - signal-driven I/O 인 경우에는 SIGIO 시그널이 발생함
  - getsockopt 에 의해 리턴되는 정수 값이 socket 에 대한 pending error 이고 , 1 회 호출 후 커널 내의 so_error 의 값은 0 으로 리셋됨

Generic Socket Options (cont’d)

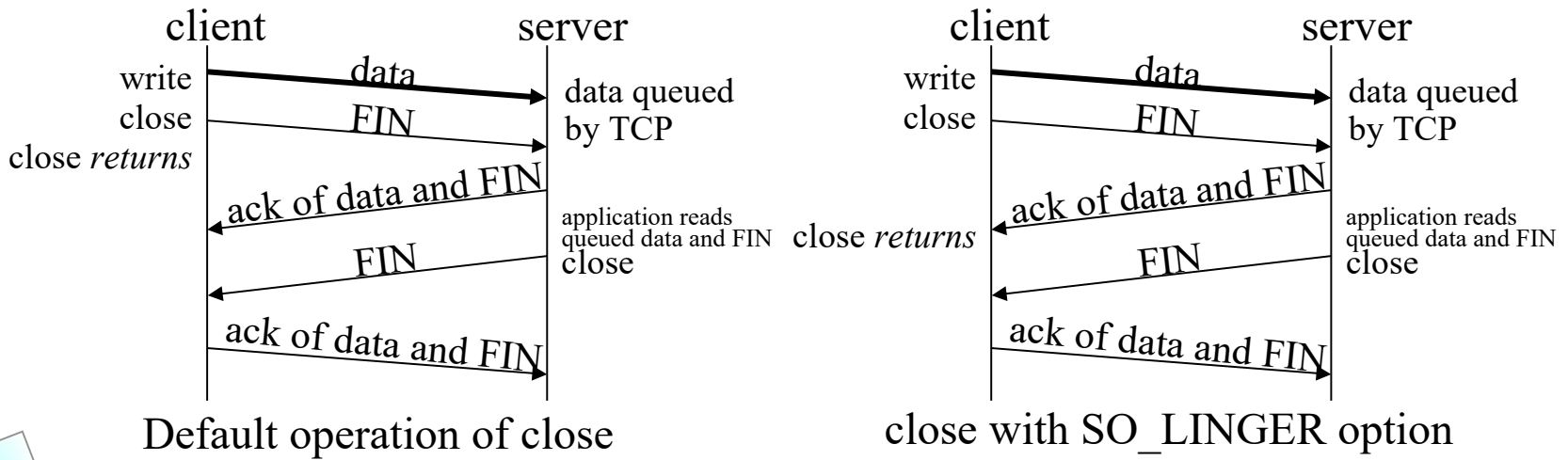
- **SO_KEEPALIVE**
  - 이 옵션이 세팅되어 있는 경우 , 한 방향으로 2 시간 동안 데이터가 전송되지 않는 경우 TCP 가 자동으로 keepalive probe 를 전송
    1. 상대방으로부터 기대한 ACK 를 받음 : Everything is OK!
    2. 상대방으로부터 RST 를 받음 : 상대편 호스트가 crash 후 reboot 됨 . pending error 가 ECONNRESET 으로 세팅되고 소켓이 닫힘
    3. 상대방으로부터 응답을 받지 못함 : 75 초마다 8 회의 probe 를 다시 보냄 . 첫 probe 로부터 11 분 15 초간 응답이 없으면 , pending error 가 ETIMEDOUT 으로 세팅되고 , 소켓이 닫힘
  - KEEPALIVE 파라미터를 수정 ?
    - † TCP_KEEPALIVE 옵션 : 구현되지 않은 경우가 많음
    - † TCPv1 의 Appendix E (pp. 530-535): socket 별로 설정은 불가능

```
# ndd /dev/tcp tcp_keepalive_interval
7200000    → default is 7200000 ms (2 hours)
# ndd –set /dev/tcp tcp_keepalive_interval value
```

Generic Socket Options (cont'd)

- **SO_LINGER**
  - Connection oriented protocol □□ close function □ □□□□ □□□ □□□

```
struct linger {
    int    l_onoff;     /* 0=off, nonzero=on */
    int    l_linger;    /* linger time, units as seconds */
```



Generic Socket Options (cont'd)

- **SO_OOBINLINE**
  - out-of-band data 에 대한 처리 : 이 옵션이 세팅되어 있으면 OOB data 를 normal input queue 에 둠
- **SO_RCVBUF / SO_SNDBUF**
  - 각 소켓의 send buffer 와 receive buffer 의 크기를 조절
  - **receive buffer** 의 크기는 **TCP** 의 **flow control** 에 필요
- **SO_RCVLOWAT / SO_SNDLOWAT**
  - select function 에서 사용되는 receive/send low-water mark 값을 설정함
  - receive low-water mark 의 default 값은 1
  - send low-water mark 의 default 값은 2048
- **SO_RCVTIMEO / SO_SNDTIMEO**
  - send/receive 시의 타임아웃 값을 설정
  - receive timeout affects: read, readv, recv, recvfrom, recvmsg
  - send timeout affects: write, writev, send, sendto, sendmsg

Generic Socket Options (cont'd)

- **SO_REUSEADDR / SO_REUSEPORT**
  - SO_REUSEADDR 옵션은 4 가지의 다른 용도로 사용됨

1. 한 포트로 전에 체결된 연결이 있더라도 listening server 가 그 포트로 binding 할 수 있게 함

시나리오
(a) listening server 가 시작
(b) 연결 요청이 도착하고 , 그 client 를 처리할 자식 프로세스가 생성됨
(c) listening server 가 종료되고 , 자식 프로세스는 기존 연결로 서비스를 계속함
(d) listening server 가 재시작
이와 같은 경우 SO_REUSEADDR 옵션이 설정되어 있지 않으면 , (d) 과정에서 bind 가 실패함

2. 각각의 instance 가 다른 로컬 **IP** 주소로 bind 하는 경우 , 같은 서버의 여러 개의 instance 가 존재할 수 있도록 허용

※ TCP 에서 completely duplicate binding 은 허용되지 않음 .

Generic Socket Options (cont’d)

3. 각각의 bind 가 다른 로컬 IP 주소를 지정하는 경우 , 하나의 프로세스가 여러 개의 소켓을 하나의 포트로 binding 하도록 함
4. UDP 소켓의 경우 completely duplicate binding 을 가능하도록 함

- SO_REUSEPORT (4.4BSD)
  - † multicasting 에 대한 지원이 추가 (4.4BSD)
  - † 같은 IP 와 포트로 binding 하고자 하는 모든 소켓이 SO_REUSEPORT 옵션을 세팅한 경우 , completely duplicate binding 을 허용
  - † IP 주소가 multicast address 이면 , SO_REUSEADDR 는 SO_REUSEPORT 와 동일

- **SO_TYPE**
  - 소켓 종류를 리턴함 (SOCK_STREAM, SOCK_DGRAM, etc)
- **SO_USELOOPBACK (AF_ROUTE** 에서만 사용 **)**
  - 소켓에서 보내지는 모든 패킷의 사본을 받게 됨

fnctl / ioctl Functions

- fnctl, ioctl, routing socket operation □ □□ □□

Operation	fnctl	ioctl	Routing socket	Posix.1g
set socket for nonblocking I/O	F_SETFL, O_NONBLOCK	FIONBIO		fnctl
set socket for signal-driven I/O	F_SETFL, O_ASYNC	FIOASYNC		fnctl
set socket owner	F_SETOWN	SIOCSPGRP or FIOSETOWN		fnctl
get socket owner	F_GETOWN	SIOCGPGRP or FIOGETOWN		fnctl
get #bytes in socket receive buffer		FIONREAD		
test for socket at out-of-bank mark		SIOCATMARK		sockatmark
obtain interface list		SIOCGIFCONF	sysctl	
interface operations		SIOC[GS]If*xxx*		
ARP cache operations		SIOC*x*ARP	RTM_*xxx*	
routing table operations		SIOC*xxx*RT	RTM_*xxx*	

fnctl □□ □□ □

- **Nonblocking I/O □ □□□□ □**

```
int flags;

            /* Set socket nonblocking */
if ( (flags = fnctl (fd, F_GETFL, 0)) < 0 )
            err_sys("F_GETFL error");
flags |= O_NONBLOCK;
if ( (fnctl(fd, F_SETFL, flags) < 0 )
            err_sys("F_SETFL error");
```

- **Nonblocking I/O □ □□□□ □**

```
......
flags &= ~O_NONBLOCK;
if ( (fnctl(fd, F_SETFL, flags) < 0 )
            err_sys("F_SETFL error");
```

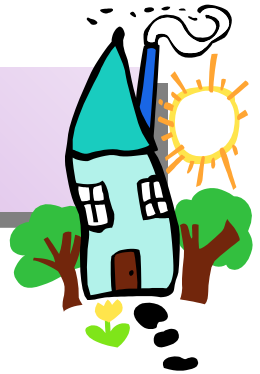
Unix Network Programming

Chapter 8
Elementary UDP Socket

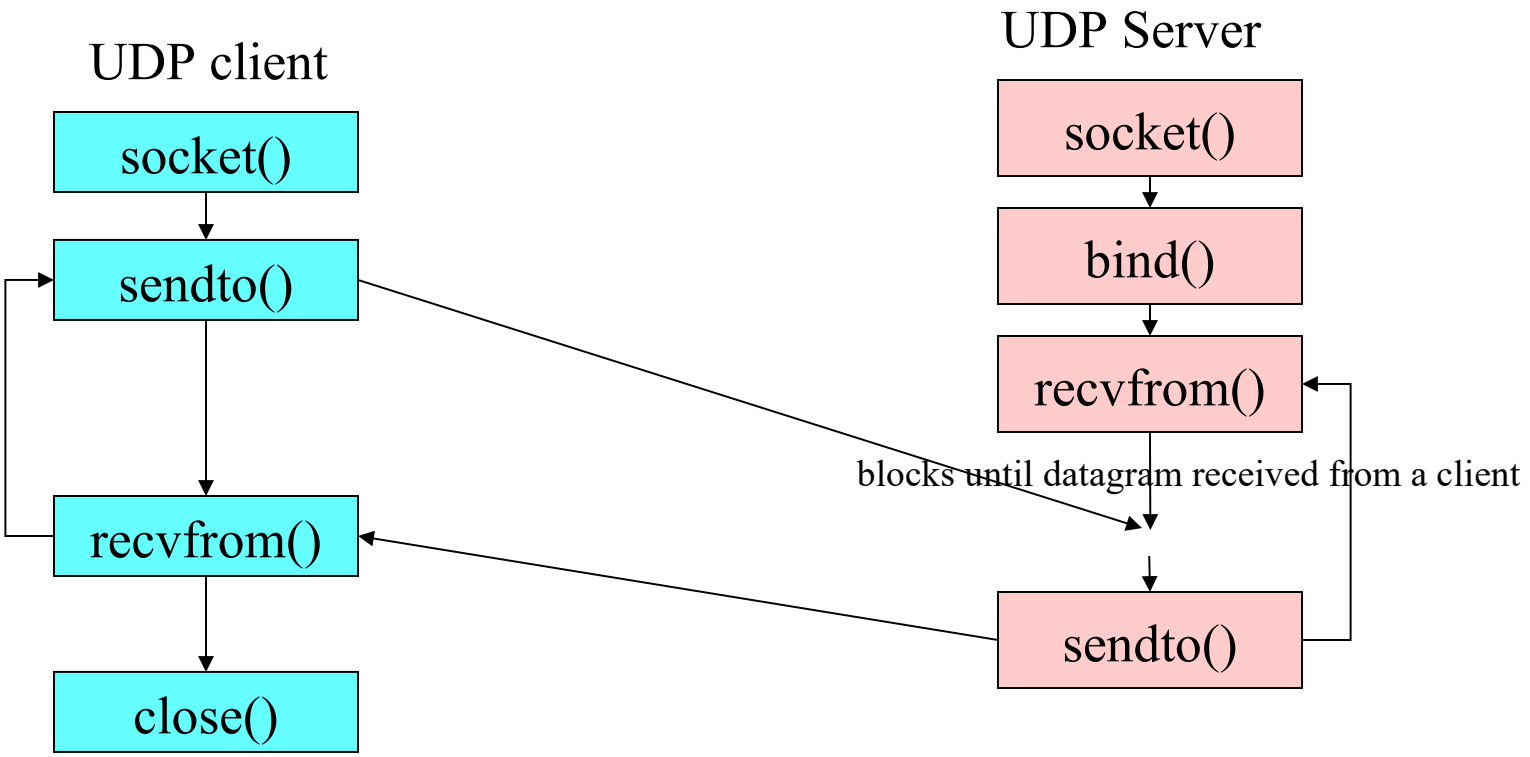
Contents

- Introduction
- *recvfrom* and *sendto* functions
- UDP Echo Server : main Function
- UDP Echo Server : dg_echo Function
- UDP Echo Client : main Function
- UDP Echo Client : dg_cli Function
- Lost Diagrams
- Verifying Received Response
- Server Not Running
- Summary of UDP example
- connect Function with UDP
- dg_cli Function(Revisited)
- Lack of Flow Control with UDP
- Determining Outgoing Interface with UDP
- TCP and UDP Echo Server Using select
- Summary

Introduction



- **connectionless, unreliable, datagram protocol**



recvfrom and *sendto* Functions

- **#include <sys/socket.h>**

13 □□□ □□ . recv, send, recvmsg & sendmsg □□ 0 □□□ □□

- **ssized_t recvfrom(int sockfd, void *buff, size_T nbytes, int flags,**
  struct sockaddr *from, socklen_t *addrlen);

accept

- **ssized_t sendto(int sockfd, const void *buff, size_t nbytes, int flags,**
  const struct sockaddr *to, socklen_t addrlen);

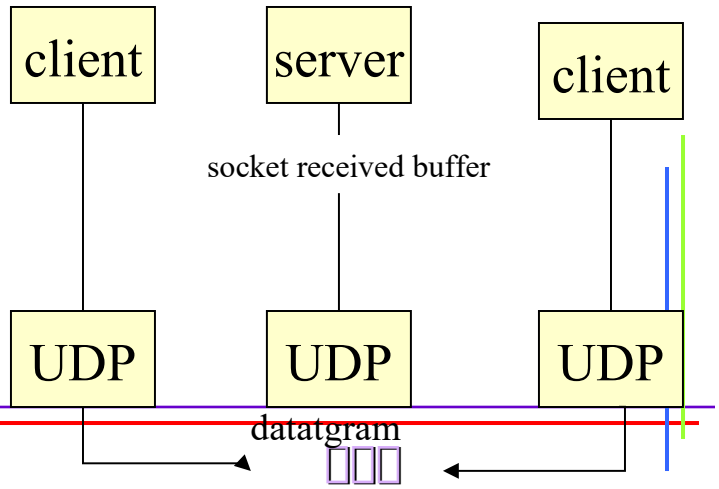
connect

both return : # of byte read or written if OK, -1 on error

UDP Echo Server

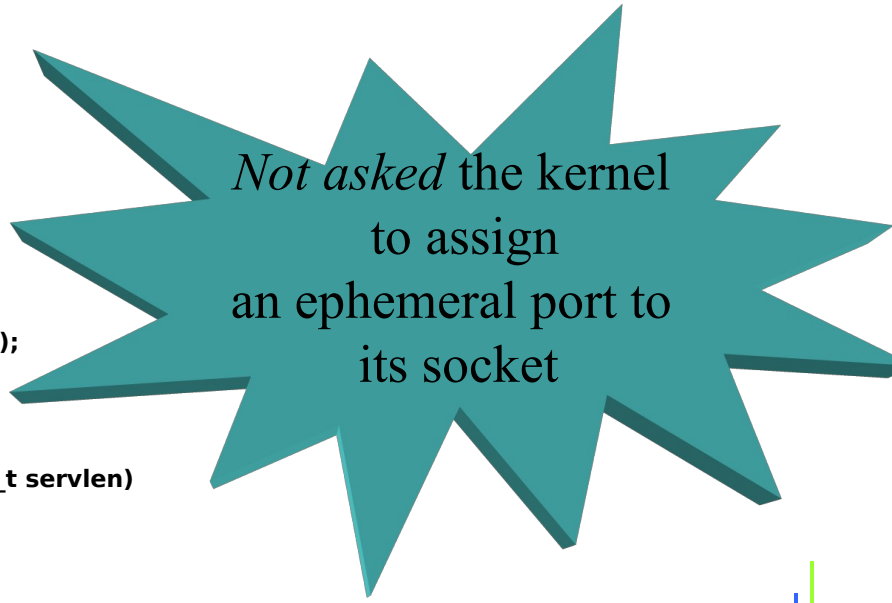
```
1.    #include  "unp.h"
2.    int main(int argc, char **argv)
3.    {  int        sockfd;
4.      struct sockaddr_in  servaddr, cliaddr;
5.      sockfd = Socket(AF_INET, SOCK_DGRAM, 0);
6.      bzero(&servaddr, sizeof(servaddr));
7.      servaddr.sin_family      = AF_INET;
8.      servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
9.      servaddr.sin_port        = htons(SERV_PORT);
10.     Bind(sockfd, (SA *) &servaddr, sizeof(servaddr));
11.     dg_echo(sockfd, (SA *) &cliaddr, sizeof(cliaddr)); }
12.   void dg_echo(int sockfd, SA *pcliaddr, socklen_t clilen){
13.     int     n;
14.     socklen_t len;
15.     char    mesg[MAXLINE];
16.     for ( ; ; ) {
17.       len = clilen;
18.       n = Recvfrom(sockfd, mesg, MAXLINE, 0, pcliaddr, &len);
19.       Sendto(sockfd, mesg, n, 0, pcliaddr, len);
20.     }}
```

Never terminated
Iterative Server
SO_RCVBUF Socket Option



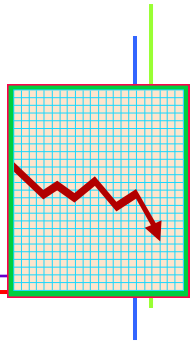
UDP Echo Client

```
1.      #include  "unp.h"
2.      int main(int argc, char **argv)
3.      {
4.        int        sockfd;
5.        struct sockaddr_in  servaddr;
6.        if (argc != 2)
7.          err_quit("usage: udpcli <IPaddress>");
8.        bzero(&servaddr, sizeof(servaddr));
9.        servaddr.sin_family = AF_INET;
10.       servaddr.sin_port = htons(SERV_PORT);
11.       Inet_pton(AF_INET, argv[1], &servaddr.sin_addr);
12.       sockfd = Socket(AF_INET, SOCK_DGRAM, 0);
13.       dg_cli(stdin, sockfd, (SA *) &servaddr, sizeof(servaddr));
14.       exit(0);
15.     }
16.     void
17.     dg_cli(FILE *fp, int sockfd, const SA *pservaddr, socklen_t servlen)
18.     {
19.       int n;
20.       char  sendline[MAXLINE], recvline[MAXLINE + 1];
21.       while (Fgets(sendline, MAXLINE, fp) != NULL) {
22.       Sendto(sockfd, sendline, strlen(sendline), 0, pservaddr, servlen);
23.         n = Recvfrom(sockfd, recvline, MAXLINE, 0, NULL, NULL);
24.         recvline[n] = 0;  /* null terminate */
25.         Fputs(recvline, stdout);
26.       }   }
```



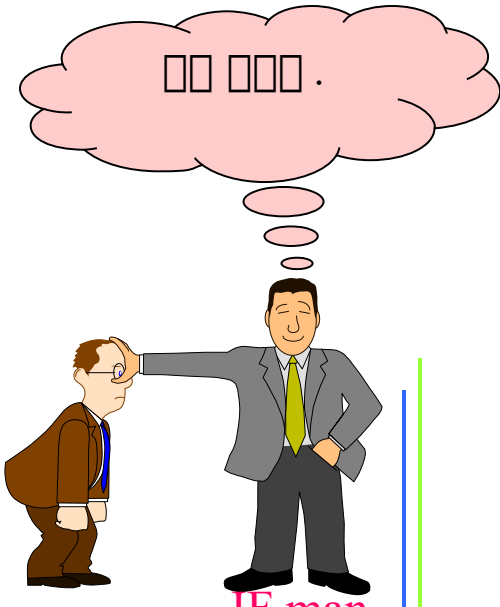
Lost Datamgrams

- **Our UDP client-server example is *not reliable*.**
- **If a client is lost, the client will *block forever* in its call to *recvfrom* in the function dg_cli, waiting for a server reply.**
- **Only way to prevent this is to *place a timeout* on the client's call to recvfrom.(13.2)**
  - connent with a Timeout Using SIGALRM
  - recvfrom with a Timeout Using SIGALRM
- **Reliability to a UDP client-server in 20.5 p542**



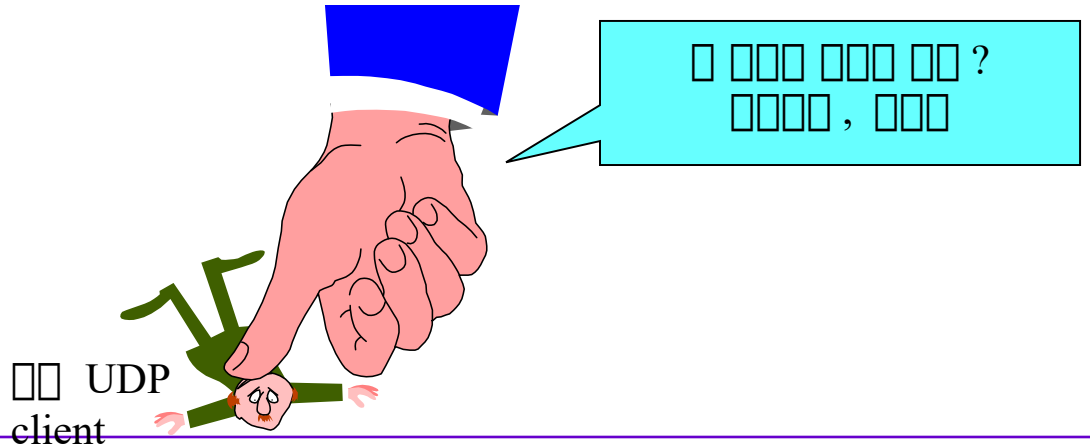
Verifying Received Response

```
1.      #include  "unp.h"
2.      void dg_cli(FILE *fp, int sockfd, const SA *pservaddr, socklen_t servlen)
3.      {
4.        int      n;
5.        char     sendline[MAXLINE], recvline[MAXLINE + 1];
6.        socklen_t   len;
7.        struct sockaddr *preply_addr;
8.        preply_addr = Malloc(servlen);
9.        while (Fgets(sendline, MAXLINE, fp) != NULL) {
10.         Sendto(sockfd, sendline, strlen(sendline), 0, pservaddr, servlen);
11.         len = servlen;
12.         n = Recvfrom(sockfd, recvline, MAXLINE, 0, preply_addr, &len);
13.         if (len != servlen || memcmp(pservaddr, preply_addr, len) != 0) {
14.           printf("reply from %s (ignored)\n",
15.               Sock_ntop(preply_addr, len));
16.           continue;
17.         }
18.         recvline[n] = 0;  /* null terminate */
19.         Fputs(recvline, stdout);
20.       }
21.     }
```



Server Not Running

- **Asynchronous error**
  - errors that are reported some time ***after*** the packet was sent.
  - ICMP port unreachable
  - TCP : always report these errors to the application.

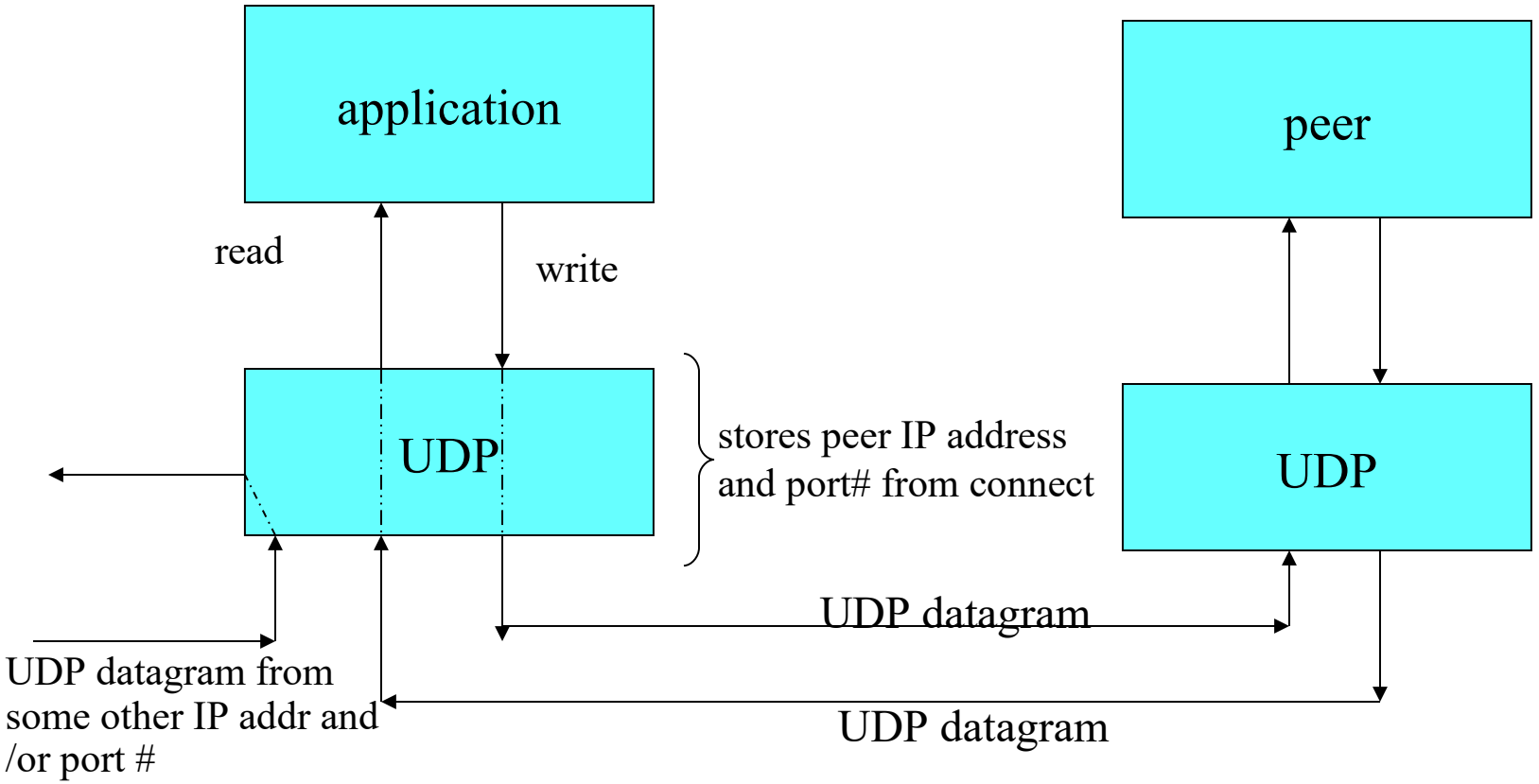


Connect Function with UDP(1)

- **unconnected UDP Socket**
  - the default when we create a UDP socket
- **connected UDP Socket**
  - the result of calling connect on a UDP socket

1. Does *not* use ***sendto*** but use ***write*** or ***send*** instead.
2. Does *not* use ***recvfrom*** but use ***read*** or ***recv*** instead.
3. ***Asynchronous errors*** are returned to the process for a connected UDP socket. The corollary, as we previously described, is that an unconnected UDP socket does not receive any asynchronous errors

connect Function with UDP(2)

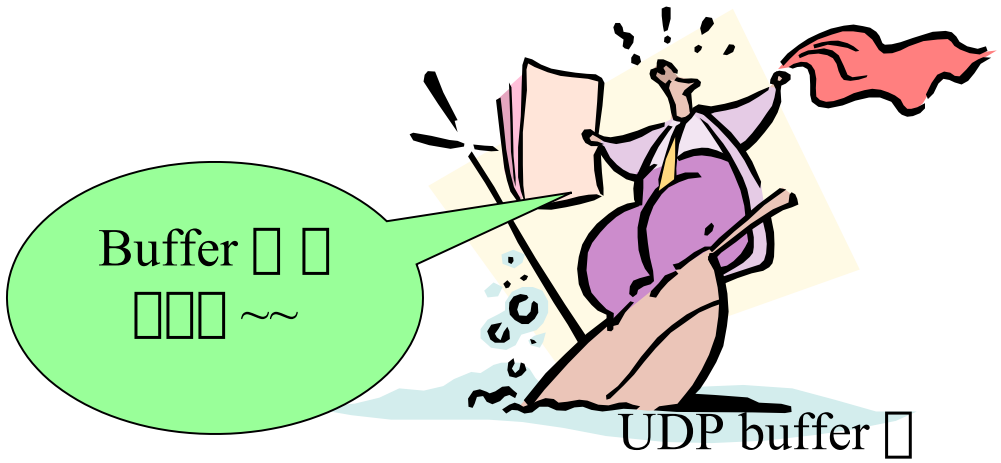


dg_client Function

```
1.      #include  "unp.h"

2.      void
3.      dg_cli(FILE *fp, int sockfd, const SA *pservaddr, socklen_t servlen)
4.      {
5.        int   n;
6.        char  sendline[MAXLINE], recvline[MAXLINE + 1];

7.        Connect(sockfd, (SA *) pservaddr, servlen);

8.        while (Fgets(sendline, MAXLINE, fp) != NULL) {

9.          Write(sockfd, sendline, strlen(sendline));

10.         n = Read(sockfd, recvline, MAXLINE);

11.         recvline[n] = 0;  /* null terminate */
12.         Fputs(recvline, stdout);
13.       }
14.     }
```

Lack of Flow Control with UDP

- **UDP has no flow control and unreliable**
  - ex) UDP sender to overrun the receiver.
- **UDP Socket Receive Buffer**
  - can change receive Buffer : SO_RCVBUF
  - default : 41,600 bytes(BSD/OS)
    - † actual limit : 246,723 bytes



TCP and UDP Echo Server Using select

```
1.  int  main(int argc, char **argv) {
2.    int        listenfd, connfd, udpfd, nready, maxfdp1;
3.    char       mesg[MAXLINE];
4.    pid_t      childpid;
5.    fd_set     rset;
6.    ssize_t    n;
7.    socklen_t  len;
8.    const int  on = 1;
9.    struct sockaddr_in  cliaddr, servaddr;
10.   void       sig_chld(int);
11.     /* create listening TCP socket */
12.   listenfd = Socket(AF_INET, SOCK_STREAM, 0);
13.   bzero(&servaddr, sizeof(servaddr));
14.   servaddr.sin_family      = AF_INET;
15.   servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
16.   servaddr.sin_port        = htons(SERV_PORT);
17.   Setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR,
      &on, sizeof(on));
18.   Bind(listenfd, (SA *) &servaddr, sizeof(servaddr));
19.   Listen(listenfd, LISTENQ);
20.     /* create UDP socket */
21.   udpfd = Socket(AF_INET, SOCK_DGRAM, 0);
22.   bzero(&servaddr, sizeof(servaddr));
23.   servaddr.sin_family      = AF_INET;
24.   servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
25.   servaddr.sin_port        = htons(SERV_PORT);
26.   Bind(udpfd, (SA *) &servaddr, sizeof(servaddr));
```

```
28.   /* include udpservselect02 */
29.     Signal(SIGCHLD, sig_chld);  /* must call waitpid() */
30.     FD_ZERO(&rset);
31.     maxfdp1 = max(listenfd, udpfd) + 1;
32.     for ( ; ; ) {
33.       FD_SET(listenfd, &rset);
34.       FD_SET(udpfd, &rset);
35.       if ( (nready = select(maxfdp1, &rset, NULL, NULL, NULL)) < 0)
      {
36.         if (errno == EINTR)       continue;   /* back to for() */
37.         else       err_sys("select error");    }
38.       if (FD_ISSET(listenfd, &rset)) {
39.         len = sizeof(cliaddr);
40.         connfd = Accept(listenfd, (SA *) &cliaddr, &len);
41.           if ( (childpid = Fork()) == 0) {  /* child process */
42.           Close(listenfd);  /* close listening socket */
43.           str_echo(connfd); /* process the request */
44.           exit(0);      }
45.         Close(connfd);      /* parent closes connected socket */    }
46.       if (FD_ISSET(udpfd, &rset)) {
47.         len = sizeof(cliaddr);
48.         n = Recvfrom(udpfd, mesg, MAXLINE, 0, (SA *) &cliaddr,
      &len);
49.         Sendto(udpfd, mesg, n, 0, (SA *) &cliaddr, len);
50.       }
51.     }
52.   }
```

/etc/services □□□

- **tcpmux 1/tcp**
- *echo 7/tcp*
- *echo 7/udp*
- **discard 9/tcp sink null**
- **discard 9/udp sink null**
- **systat 11/tcp users**
- *daytime 13/tcp*
- *daytime 13/udp*
- **netstat 15/tcp**
- **chargen 19/tcp ttytst source**
- **chargen 19/udp ttytst source**
- **ftp-data 20/tcp**
- **ftp 21/tcp**
- **telnet 23/tcp**
- *smtp 25/tcp* **mail**
- *time 37/tcp timserver*
- *time 37/udp* **timserver**
- *name 42/udp nameserver*
- **whois 43/tcp nicname # usually to sri-nic**
- *domain 53/udp*
- *domain 53/tcp*
- **bootps 67/udp # BOOTP/DHCP serve**
- **bootpc 68/udp # BOOTP/DHCP client**

1 port □ UDP/TCP □□□ □□□□ □□ □□□□ □□

Unix Network Programming

Chapter 9.

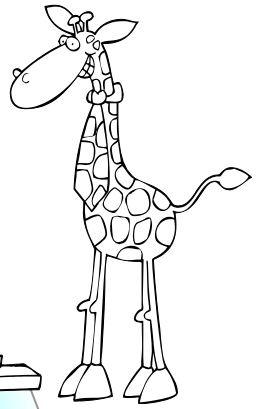
Elementary Name and Address Conversions

Contents

- Introduction
- Domain Name System
- *gethostbyname* Functions
- RES_USE_INET6 Resolver Option
- *gethostbyname2* Function and IPV6 Support
- *gethostbyaddr* Function
- *uname* Function
- *gethostname* Function
- *getservbyname* and *getservbyport* Functions
- Other Networking Information
- Summary

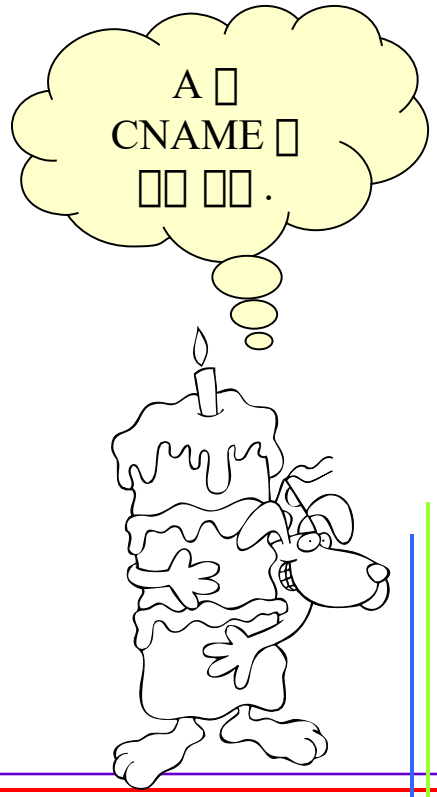
Introduction

- **So far, we used numeric addresses for the host**
- **User names instead of numbers for numerous reasons**
  - names are easier to remember
    - † gethostbyname, gethostbyaddr (hostnames -> IP addr)
    - † getservbyname, getservbyport(service names -> port)

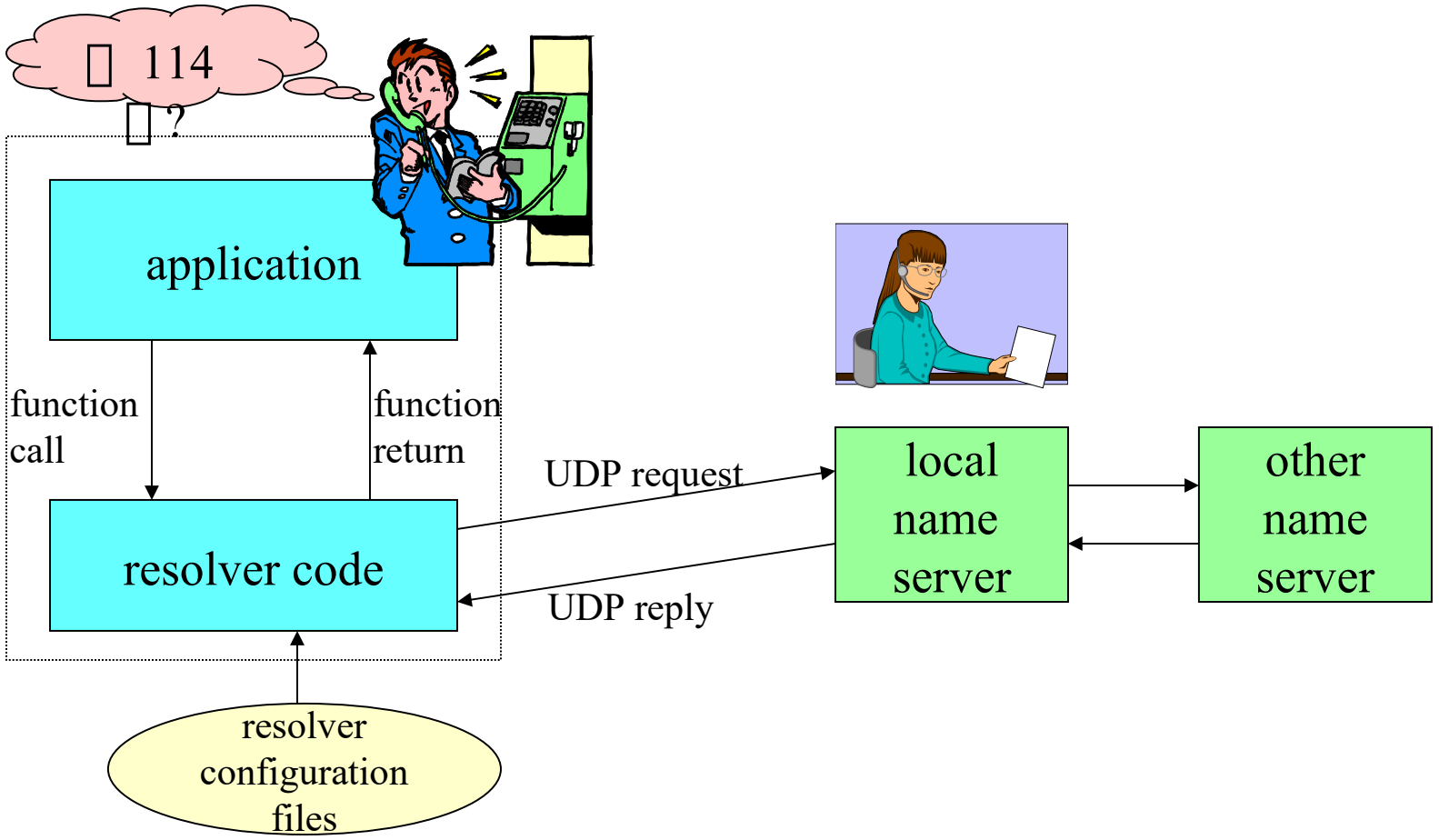


Domain Name System

- **DNS is used primarily to** *map* **between** *hostnames* **and** *IP addresses***.**
- **Resource Records**
  - A : 32bit IPv4 address
  - AAAA : 128bit IPv6 address
  - PTR : pointer records
  - MX : mail exchanger
  - CNAME : canonical name (ftp, www,..)



Typical arrangement of clients, resolvers, and name servers



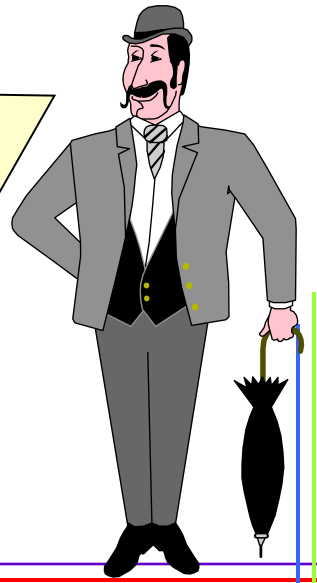
gethostbyname function

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

Returns : nonull pointer if OK, NULL on error with h_errorno set

```
struct hotent {
        char    *h_name;
        char    **h_aliases;
        int     h_addrtype;
        int     h_length;
        char    **h_addr_list;
};
#define h_addr h_addr_list[0];
```

page 242 를 보시면 ,
구조체가 어떻게
생겼는지 알 수 있슴다 .
IPv6 용 9.4, 9.5 절은
관심있는 분은
개별적으로
읽어보세여 .



gethostbyaddr Function / uname Function

AF_INET or AF_INET6

- **#include <netdb.h>**
- **struct hostent *gethostbyaddr(const char *addr, size_t len, int family)**

Returns : nonull pointer if OK, NULL on error with h_errno set

- #include <sys/utsname.h>
- int uname(struct utsname *name)

Returns : nonnegative value is OK, -1 on error

```
#define _UTS_NAMESIZE  16
#define _UTS_NODESIZE  256
struct   utsname{
        char sysname[_UTS_NAMESIZE]; /* os name */
        char nodename[_UTS_NODESIZE]; /* node name */
        char release[_UTS_NAMESIZE]; /* O.S. release level */
        char version[_UTS_NAMESIZE]; /* O.S. version level */
        char machine[_UTS_NAMESIZE]; /*hardware type */
}
```

uname() □□□ local IP □ □□□□

getservbyname / getservbyport

- **/etc/services** 을 참조해서 , **local machine** 의 제공되는 **server** 의 이름이나 , 그것이 사용하는 **port** 를 출력해준다 .

Information	Data file	Structure	keyed lookup function
hosts	/etc/hosts	hostent	gethostbyaddr, gethostbyname
networks	/etc/networks	netent	getnetbyaddr, getnetbyname
protocols	/etc/protocols	protoent	getprotobyname, getprotobynumber
services	/etc/services	servent	getservbyname, getservbyport

Summary

- **resolver**
  - to convert a hostname into an IP address and vice versa
- **gethostbyname / getthostbyaddr**
  - IPV6 □ : gethostname2(), RES_USE_INET6
- **getservbyname**
  - commonly used function dealing with services names and port

UNIX Network Programming
(chap 10 - chap 11)

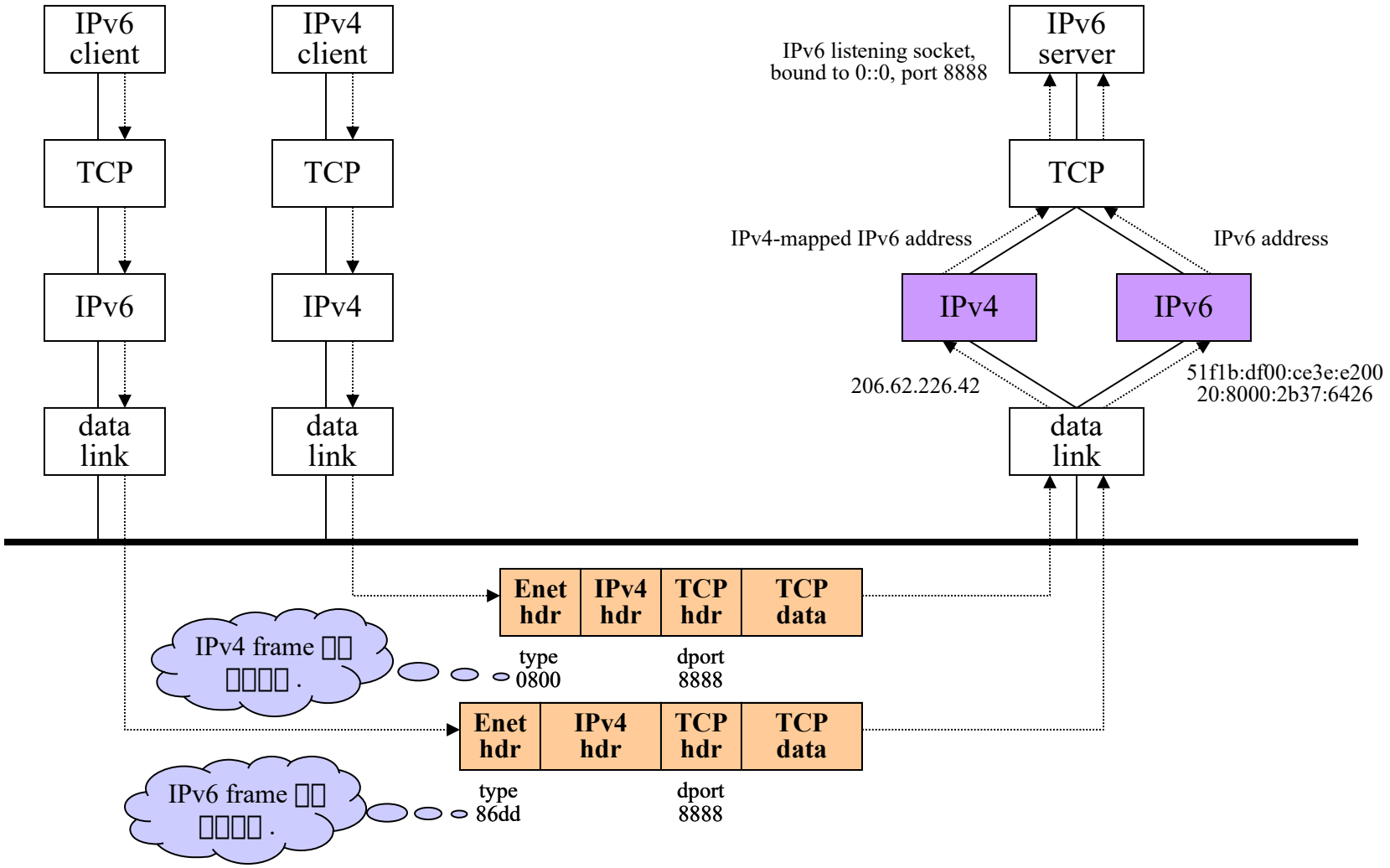
Agenda

- IPv4 and IPv6 Interoperability (chap.10)
  - IPv6 Server on dual-stack host
  - Processing of sever on dual-stack host
  - Dual stack host
  - Processing of client requests
  - IPv4-mapped IPv6

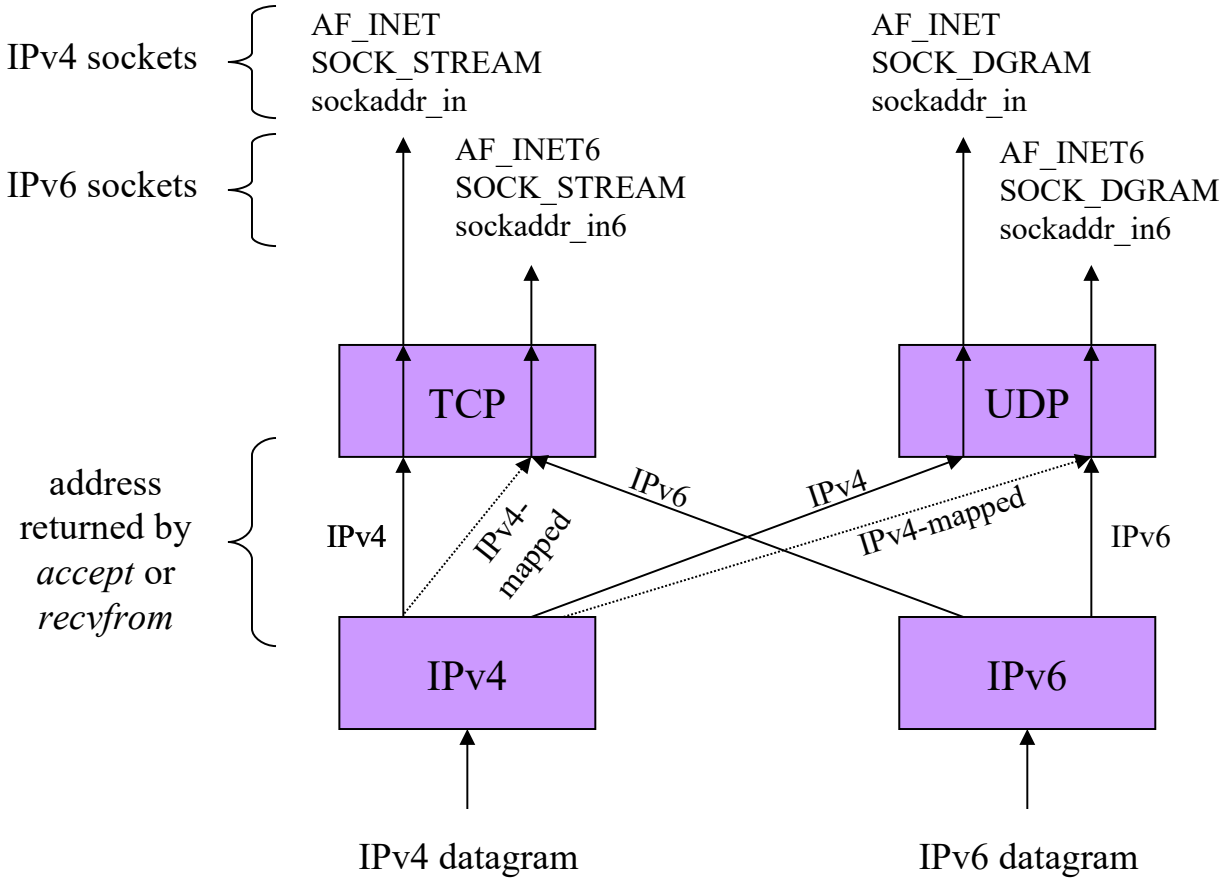
Agenda

- Advanced Name and Address Conversions (chap.11)
  - Why use *getaddrinfo( ) & getnameinfo( )* ?
  - *getaddrinfo( )*
  - Action and Result of *getaddrinfo( )*
  - *getnameinfo( )*
  - Reentrant functions

IPv6 Server on dual-stack host



Processing of server on dual-stack host

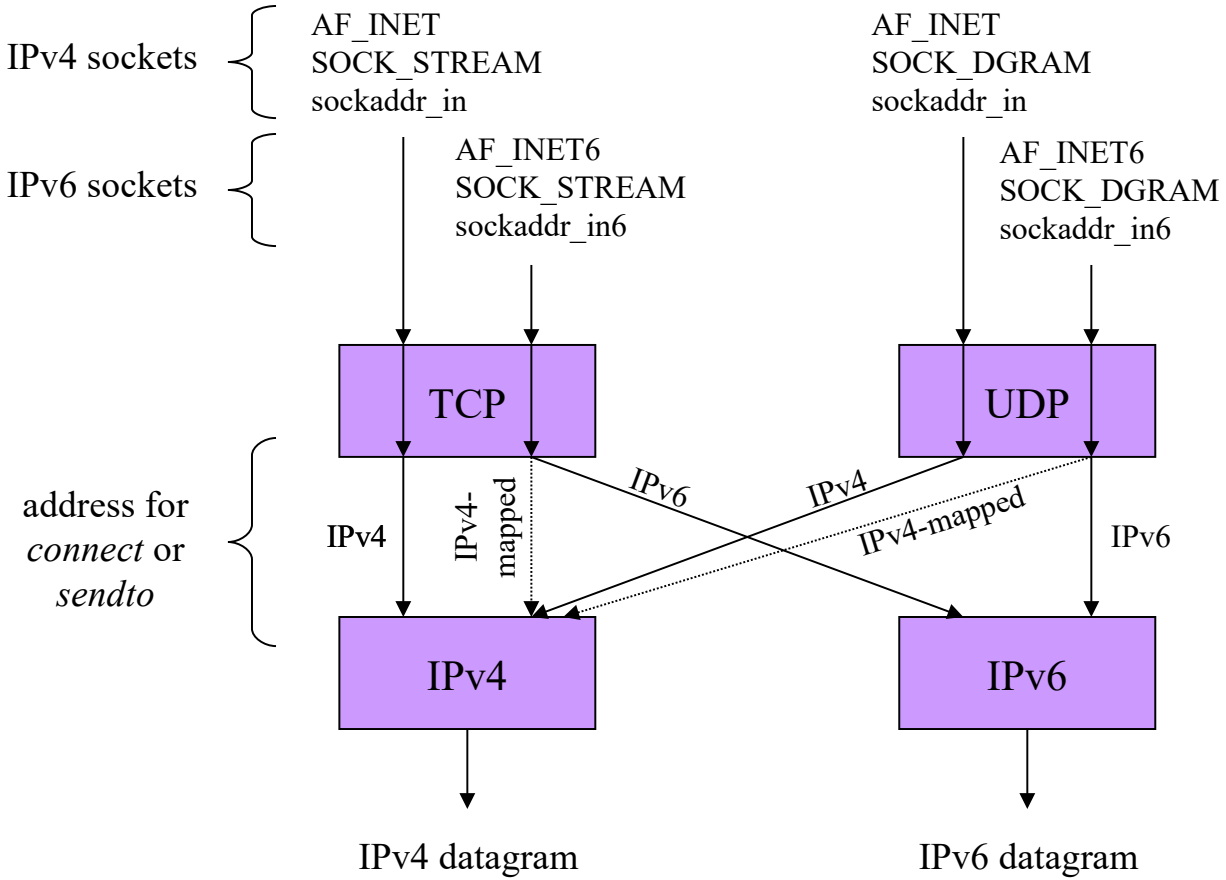


Dual-stack host

📖 Listening socket rules

1. A listening IPv4 socket can accept incoming connections from only IPv4 clients.
2. If a server has a listening IPv6 socket that has bound the wildcard address, that socket can accept incoming connections from either IPv4 clients or IPv6 clients. For a connection from an IPv4 client the server's local address for the connections will be the corresponding IPv4-mapped IPv6 address.
3. If a server has a listening IPv6 socket that has bound an IPv6 address other than an IPv4-mapped IPv6 address, that socket can accept incoming connections from IPv6 clients only

Processing of client requests



IPv4-mapped IPv6

- IPv6 server

  IPv4 → IPv4-mapped IPv6 : kernel 에 의해서 수행된다 .
  accept 또는 recvfrom 에 의해서 application 으로 투명하게 전달된다 .

- IPv6 client

  IPv6 → IPv4-mapped IPv6 : resolver 에 의해서 수행된다 .
  (resolver ex: gethostbyname, gethostbyaddr etc)
  application 에 의해 connect 또는 sendto 로 투명하게 전달된다 .

- DNS records example

```
solaris     IN  A       206.62.226.33
            IN  AAAA    5f1b:df00:c33e:e200:0020:0800:2078:e3e3

aix         IN  A       206.62.226.43
            IN  A       5f1b:df00:c33e:e200:0020:0800:5afc:2b36

bsdi2       IN  A       206.62.226.34
```

- IPv4-mapped IPv6 address example

  return address - 0::ffff:206.62.226.34

Summary of interoperability

	IPv4 sever IPv4-only host (A only)	IPv6 sever IPv6-only host (AAAA only)	IPv4 sever dual-stack host (A and AAAA)	IPv6 sever dual-stack host (A and AAAA)
IPv4 client, IPv4-only host	IPv4	X	IPv4	IPv4
IPv6 client, IPv6-only host	X	IPv6	X	IPv6
IPv4 client, dual-stack host	IPv4	X	IPv4	IPv4
IPv6 client, dual-stack host	IPv4	IPv6	X*	IPv6

Why use *getaddrinfo( ) & getnameinfo( )* ?

gethostbyname( ) 과 *gethostbyaddr( )* 은 protocol 에 의존적이다 .

⏩ address family 를 인수로서 가진다 .

Because of !!
protocol independence
of our application

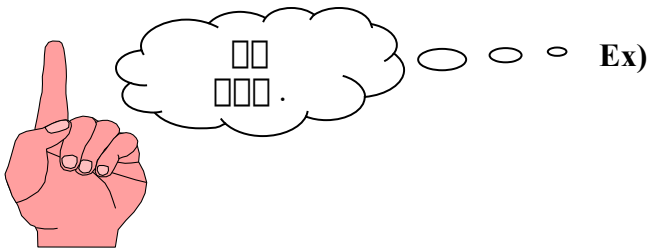
```
int getaddrinfo( const char *hostname, const char *service,
                 const struct addrinfo *hints, struct addrinfo **result );

int getnameinfo( const struct sockaddr *sockaddr, socklen_t addrlen,
                 char *host, size_t hostlen,
                 char *serv, size_t servlen, int flags );
```

getaddrinfo( )

hostname : host name or address string

service : service name or decimal port number string

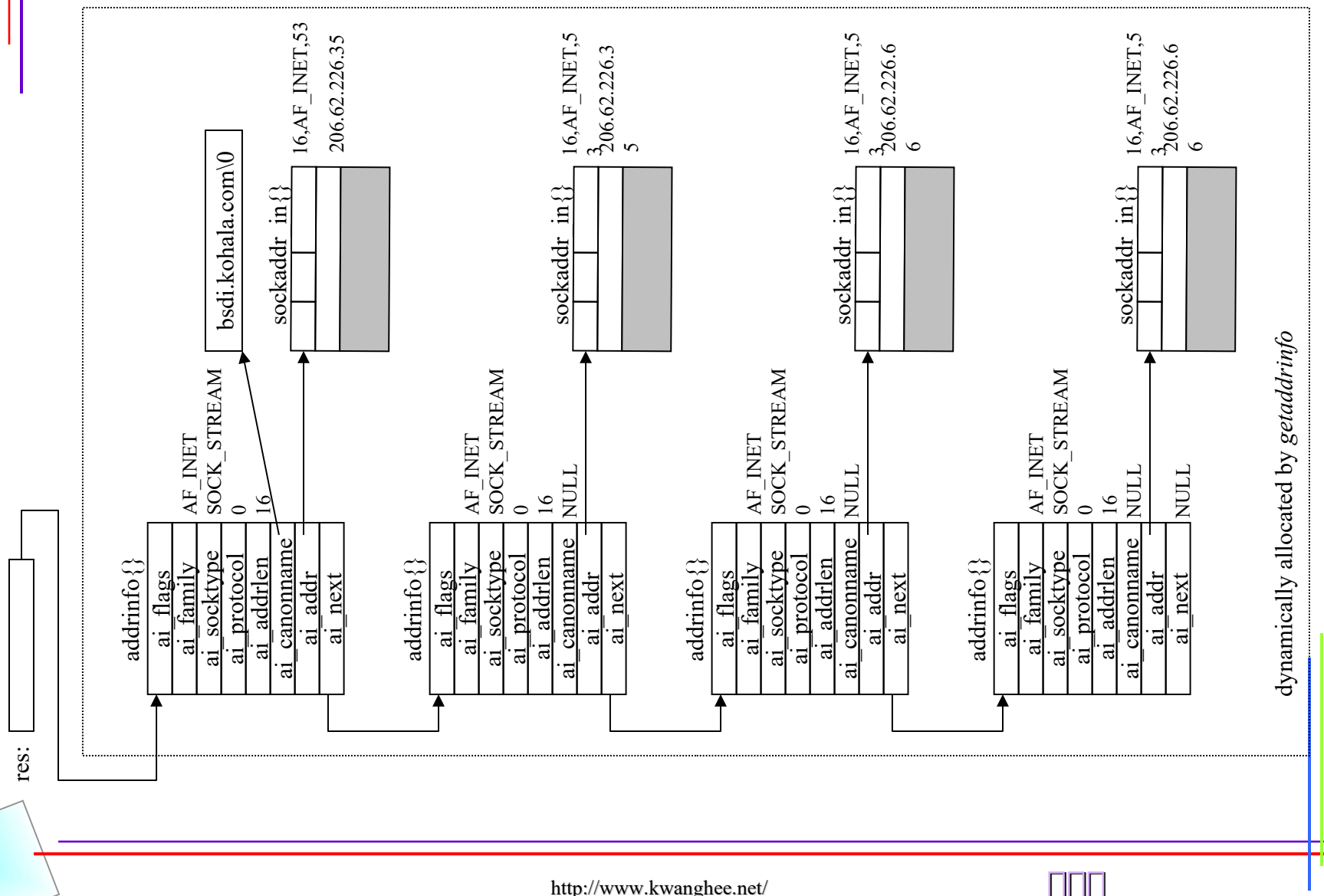
```
struct addrinfo {                       // define at <netdb.h>
    int         ai_flags;               /* AI_PASSIVE, AI_CANONNAME */
    int         ai_family;              /* AF_xxx */
    int         ai_socktype;            /* SOCK_xxx */
    int         ai_protocol;            /* 0 or IPPROTO_xxx, TCP or UDP */
    size_t      ai_addrlen;             /* length of ai_addr, IPv4:16, IPv6:24 */
    char        *ai_canonname;          /* ptr to canonical name for host */
    struct      sockaddr    *ai_addr;   /* ptr to socket address structure */
    struct      addrinfo    *ai_next;   /* ptr to next structure in linked list */
};
```



Ex)

```
struct addrinfo      hints, *res;
bzero( &hints, sizeof(hints) );
hints.ai_flags = AI_CANONNAME;
hints.ai_family = AF_INET;
getaddrinfo( “bdsi”, “domain”, &hints, &res );
```

□□□ IP address □□ □□□ □□

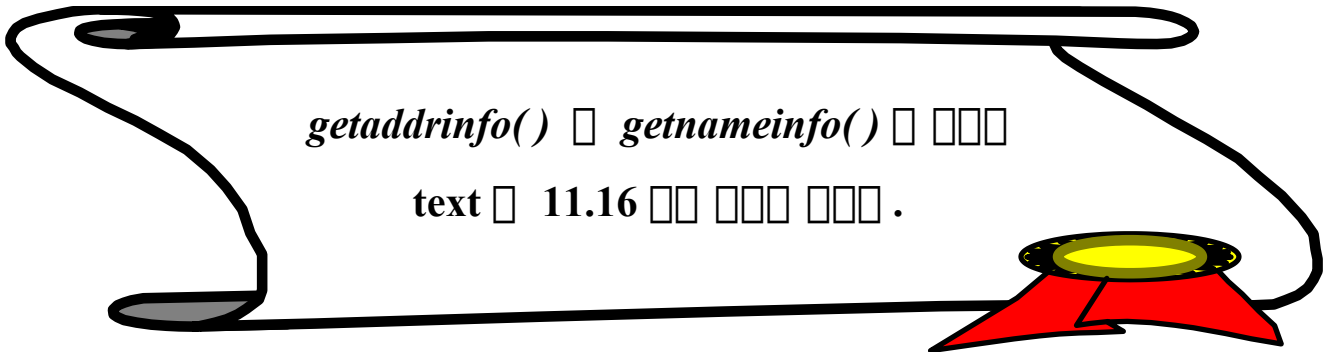

res:
addrinfo{}
ai_flags
ai_family
ai_socktype
ai_protocol
ai_addrlen
ai_canonname
ai_addr
ai_next
AF_INET
SOCK_STREAM
0
16
NULL
bsdi.kohala.com\0
sockaddr_in{}
16,AF_INET,53
206.62.226.35
16,AF_INET,5
206.62.226.3
5
206.62.226.6
6
dynamically allocated by getaddrinfo

Action and Result of *getaddrinfo( )*

Hostname specified by caller	Address family specified by caller	Hostname string contains	Result	Action
nonnull hostname string; active or passive	AF_UNSPEC	hostname	all AAAA records returned as sockaddr_in6{}s and all A records returned as sockaddr_in{}s	two DNS searches : gethostbyname2(AF_INET6) with RES_USE_INET6 off gethostbyname2(AF_INET) with RES_USE_INET6 off
		hex string	one sockaddr_in6{}	inet_pton(AF_INET6)
		dotted decimal	one sockaddr_in{}	inet_pton(AF_INET)
	AF_INET6	hostname	all AAAA records returned as sockaddr_in6{}s **else** all A records returned as IPv4-mapped IPv6 as sockaddr_in{}s	gethostbyname() with RES_USE_INET6 **on**
		hex string	one sockaddr_in6{}	inet_pton(AF_INET6)
		dotted decimal	error : EAI_ADDRFAMILY	
	AF_INET	hostname	all A records returned as sockaddr_in{}s	gethostbyname() with RES_USE_INET6 off
		hex string	error : EAI_ADDRFAMILY	
		dotted decimal	one sockaddr_in{}	inet_pton(AF_INET)

Continue ...

Hostname specified by caller	Address family specified by caller	Hostname string contains	Result	Action
null hostname string; passive	AF_UNSPEC	implied 0::0 implied 0.0.0.0	one sockaddr_in6{}s and one sockaddr_in{}s	inet_pton(AF_INET6) inet_pton(AF_INET)
	AF_INET6	implied 0::0	one sockaddr_in6{}	inet_pton(AF_INET6)
	AF_INET	implied 0.0.0.0	one sockaddr_in{}	inet_pton(AF_INET)
null hostname string; active	AF_UNSPEC	implied 0::1 implied 127.0.0.1	one sockaddr_in6{}s and one sockaddr_in{}s	inet_pton(AF_INET6) inet_pton(AF_INET)
	AF_INET6	implied 0::1	one sockaddr_in6{}	inet_pton(AF_INET6)
	AF_INET	implied 127.0.0.1	one sockaddr_in{}	inet_pton(AF_INET)



getnameinfo( )

Socket address 를 가지고 host 와 service 에 대한 string 을 각각 돌려준다 .

sock_ntop

DNS 와 무관하게 나타낼 수 있는 IP address
(IPv4 에서는 dotted decimal, IPv6 에서는 hex string) 와
port number 를 돌려준다 .

getnameinfo

host 와 service 에 대한 name or numeric 값을 돌려준다 .

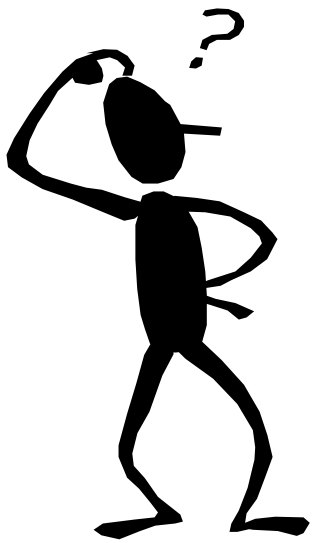
Constant	Description
NI_DGRAM	inform datagram service
NI_NAMEREQD	return an error if name cannot be resolved from address
NI_NOFQDN	return only hostname portion of FQDN
NI_NUMERICHOST	return numeric string for hostname
NI_NUMERICSERV	return numeric string for service name

< *flags* for *getnameinfo( )* >

Reentrant Functions (1)

? Problem

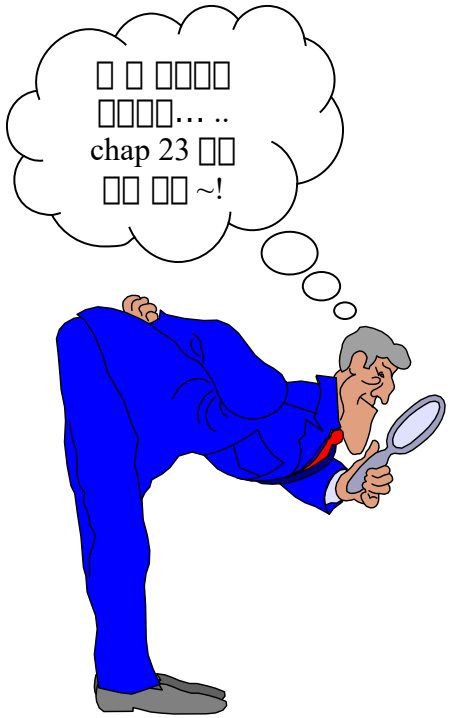
이 static 변수 (host) 의 공유로 인하여 문제가 발생될 수 있다 .



```
static struct hostent host; /* result stored here */

struct hostent *gethostbyname( const char *hostname )
{
        return( gethostbyname2( hostname, family ) );
}

struct hostent *gethostbyname2( const char *hostname, int family )
{
        /* call DNS functions for A or AAAA query */
        /* fill in host structure */
        return( &host );
}

struct hostent *gethostbyaddr( const char *addr, size_t len, int family )
{
        /* call DNS functions for PTR query in in-addr.arpa domain */
        /* fill in host structure */
        return( &host );
}
```

Reentrant Functions (2)

▯ Method



1. static structure 를 채우고 돌려주는 대신에 , 호출자는 structure 에 대한 memory 를 먼저 할당하고 reentrant function 이 호출자의 구조를 채운다 .

   - getnameinfo 가 사용하는 기법

   - 기법이 아주 복잡하고 많은 다른 정보를 사용하고 관리해야 한다 .

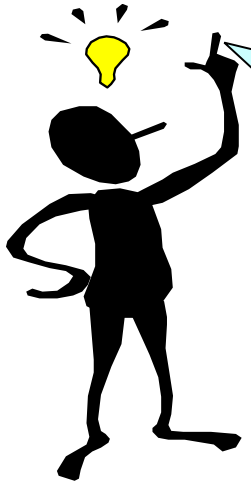
2. reentrant function 은 malloc 을 호출하여 memory 를 동적으로 할당한다 .

   - getaddrinfo 가 사용하는 기법

   - 동적으로 할당된 memory 를 해제하기 위해서 freeaddrinfo() 를 호출해서 메모리 누출을 방지해야 한다 .

Continue example ….

```
#include <netdb.h>

struct hostent *gethostbyname_r( const char *hostname,
                        struct hostent *result, char *buf, int buflen, int *h_errnop );

struct hostent *gethostbyaddr_r( const char *addr, int len, int type,
                        struct hostent *result, char *buf, int buflen, int *h_errnop );

                                both return : nonnull pointer if OK, NULL on error
```



result 는 호출자에 의해 메모리가 할당된 후
함수 내에서 값이 채워진다 .
이 값을 return 받는다 .

buf 는 호출자가 할당한 버퍼로 실제 값들을
가지고 있으며 result 의 pointer 들은 이 버퍼
안의 값들을 가리키게 된다 .
Buffer 의 크기는 최근 8192 byte 가 적당하다고
한다 .

Chap. 12 *Daemon Processes and* inetd *Superserver*

What is Daemon?

- **Daemon is ...**
  - a process that runs in the background
  - independent of control from all terminals
- **Daemon** 을 작동시키는 방법
  - 시스템의 startup 시에 system initialization script 를 통해서 작동 (/etc 또는 /etc/rc* 디렉토리)
  - inetd superserver 를 통해 작동
  - cron daemon 에 의해 수행
  - at command 로부터 지정되어 작동
  - user terminal 로부터 작동

daemon_init Function

```
1 #include    "unp.h"
2 #include    <syslog.h>
3 #define MAXFD   64
4 extern int  daemon_proc;    /* defined in error.c */
5 void
6 daemon_init(const char *pname, int facility)
7 {
8     int     i;
9     pid_t   pid;
10    if ( (pid = Fork()) != 0)
11        exit(0);          /* parent terminates */
12    /* 1st child continues */
13    setsid();             /* become session leader */
14    Signal(SIGHUP, SIG_IGN);
15    if ( (pid = Fork()) != 0)
16        exit(0);          /* 1st child terminates */
17    /* 2nd child continues */
18    daemon_proc = 1;      /* for our err_XXX() functions */
19    chdir("/");           /* change working directory */
20    umask(0);             /* clear our file mode creation mask */
21    for (i = 0; i < MAXFD; i++)
22        close(i);
23    openlog(pname, LOG_PID, facility);
24 }
```

process 가 foreground 로 수행된 경우 자동으로 background 로 전환되어 수행

새로운 그룹의 그룹 리더가 됨과 동시에 , controlling terminal 이 없는 상태가 된다 .

The purpose of this 2nd fork is to guarantee that the daemon can't automatically acquire a controlling terminal should it open a terminal device in the future

syslogd 를 통해 logging 이 가능하도록 한다 .

Example: Daytime Server as a Daemon

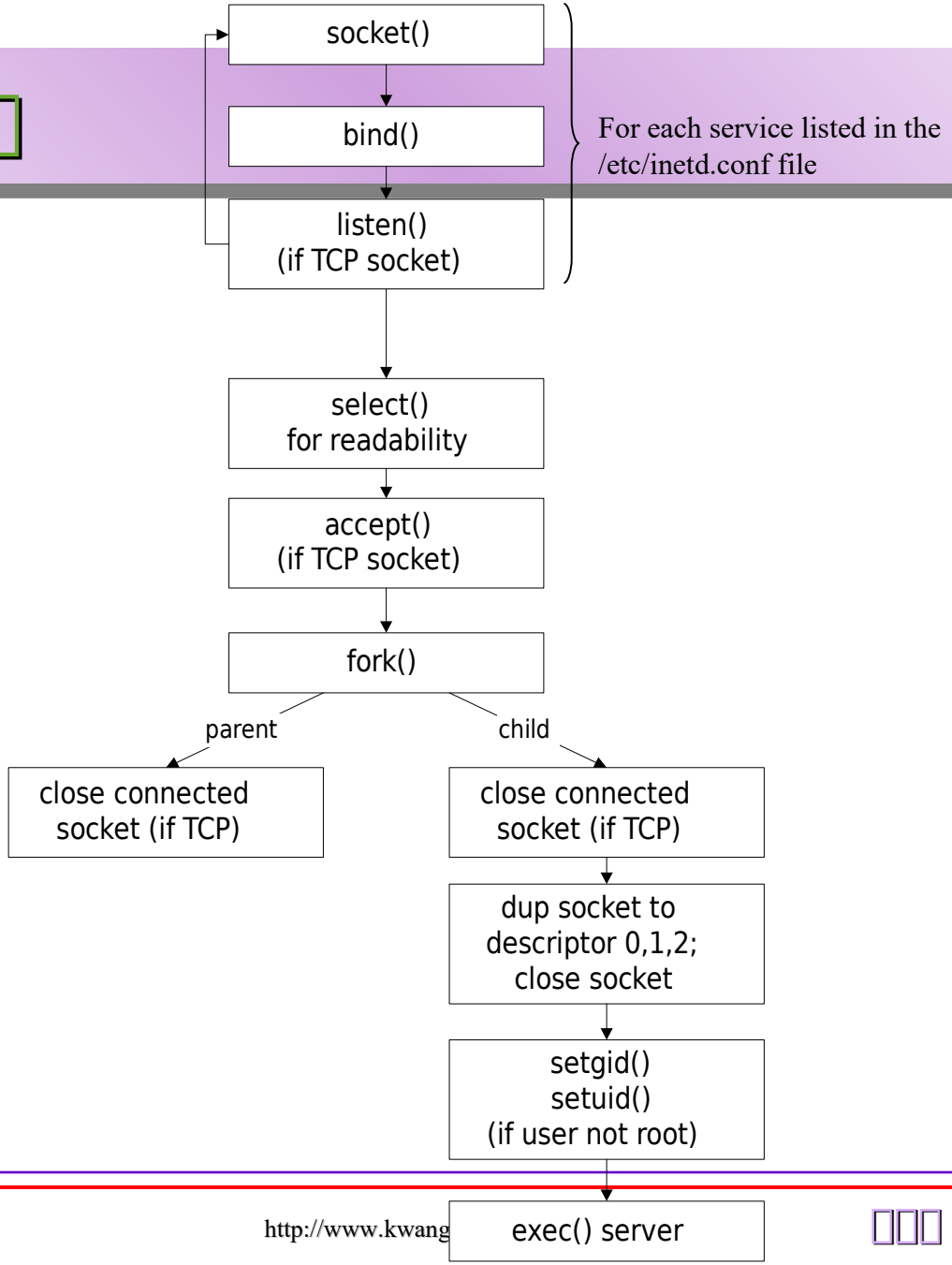
```
1 #include    "unp.h"
2 #include    <time.h>
3 int main(int argc, char **argv)
4 {
5     int             listenfd, connfd;
6     socklen_t       addrlen, len;
7     struct sockaddr *cliaddr;
8     char            buff[MAXLINE];
9     time_t          ticks;
10    daemon_init(argv[0], 0);
11    if (argc == 2)
12        listenfd = Tcp_listen(NULL, argv[1], &addrlen);
13    else if (argc == 3)
14        listenfd = Tcp_listen(argv[1], argv[2], &addrlen);
15    else
16        err_quit("usage: daytimetcpsrv2 [ <host> ] <service or port>");
17    cliaddr = Malloc(addrlen);
18    for ( ; ; ) {
19        len = addrlen;
20        connfd = Accept(listenfd, cliaddr, &len);
21        err_msg("connection from %s", Sock_ntop(cliaddr, len));
22        ticks = time(NULL);
23        snprintf(buff, sizeof(buff), "%.24s\r\n", ctime(&ticks));
24        Write(connfd, buff, strlen(buff));
25        Close(connfd);
26    }       27 }
```

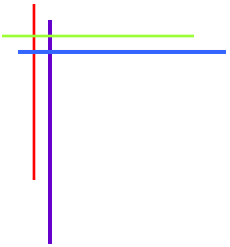
getaddrinfo() + socket() + bind() + listen()

inetd Daemon

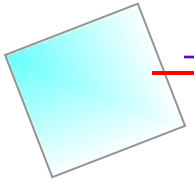
- **inetd daemon = internet superserver**
- 장점
  - inetd 에 의해 daemon 초기화에 관련된 작업을 수행하므로 각각의 daemon 프로그램이 간략해짐
  - 여러 서비스들에 대한 client request 의 대기를 하나의 프로세스 (inetd) 에서 수행하여 , 시스템의 총 프로세스 수를 감소시킬수 있음
- **/etc/inetd.conf file**
  - ftp stream tcp nowait root /usr/sbin/in.ftpd in.ftpd
  - telnet stream tcp nowait root /usr/sbin/in.telnetd in.telnetd
  - login stream tcp nowait root /usr/sbin/in.rlogind in.rlogind
  - tftp dgram udp wait root /usr/sbin/in.tftpd in.tftpd -s /tftpboot
- **/etc/services file**

inetd □ □□


socket()
bind()
listen()
(if TCP socket)
For each service listed in the
/etc/inetd.conf file
select()
for readability
accept()
(if TCP socket)
fork()
parent
child
close connected
socket (if TCP)
close connected
socket (if TCP)
dup socket to
descriptor 0,1,2;
close socket
setgid()
setuid()
(if user not root)
exec() server



Chap. 13
Advanced I/O Functions



Socket Timeouts

- **I/O operation** 에서 **timeout** 을 이용하는 **3** 가지 방법
  - alarm 함수 이용 : 특정 시간 후에 SIGALRM 시그널 발생
  - select 함수 이용 : Block waiting for I/O
  - SO_RCVTIMEO, SO_SNDTIMEO 소켓 옵션 이용
- **connect with a Timeout Using SIGALRM**

```
int connect_timeo(int sockfd, const SA *saptr, socklen_t salen, int nsec)
{
    …
    sigfunc = Signal(SIGALRM, connect_alarm);
    if (alarm(nsec) != 0)
        err_msg("connect_timeo: alarm was already set");

    if ( (n = connect(sockfd, (struct sockaddr *) saptr, salen)) < 0) {
        close(sockfd);
        if (errno == EINTR)
            errno = ETIMEDOUT;
    }
    alarm(0);                   /* turn off the alarm */
    Signal(SIGALRM, sigfunc);   /* restore previous signal handler */
    return(n);
}
```

```
static void
connect_alarm(int signo)
{
    return;     /* just interrupt the connect() */
}
```

recv and send Functions

```
#include <sys/socket.h>
ssize_t recv(int sockfd, void *buff, size_t nbyte, int flags);
ssize_t send(int sockfd, const void *buff, size_t nbytes, int flags);
```

Both return: number of bytes read or written if OK, -1 on error

※ flags 인수를 제외하고는 read / write 함수와 동일

flags	Description	recv	send
MSG_DONTROUTE	bypass routing table lookup		●
MSG_DONTWAIT	only this operation is nonblocking	●	●
MSG_OOB	send or receive out-of-band data	●	●
MSG_PEEK	peek at incoming message	●	
MSG_WAITALL	wait for all the data	●	

readv and writev / recvmsg and sendmsg Functions

```
#include <sys/socket.h>
ssize_t readv(int filedes, const struct iovec *iov, int iovcnt);
ssize_t writev(int filedes, const struct iovec *iov, int iovcnt);
```

Both return: number of bytes read or written if OK, -1 on error

```
#include <sys/socket.h>
ssize_t recvmsg(int sockfd, struct msghdr *msg, int flags);
ssize_t sendmsg(int sockfd, struct msghdr *msg, int flags);
```

Both return: number of bytes read or written if OK, -1 on error

```
struct iovec {
   void *iov_base;   /* starting addr. of buf. */
   size_t iov_len;   /* size of buffer */
};
```

```
struct msghdr {
   void *msg_name;           /* protocol address */
   socklen_t msg_namelen;    /* size of protocol addr. */
   struct iovec *msg_iov;    /* scatter/gather array */
   size_t msg_iovlen;        /* # elements in msg_iov */
   void *msg_control;        /* ancillary data */
   socklen_t msg_controllen; /* len. of ancillary data */
   int msg_flags             /* flags returned by recvmsg() */
};
```

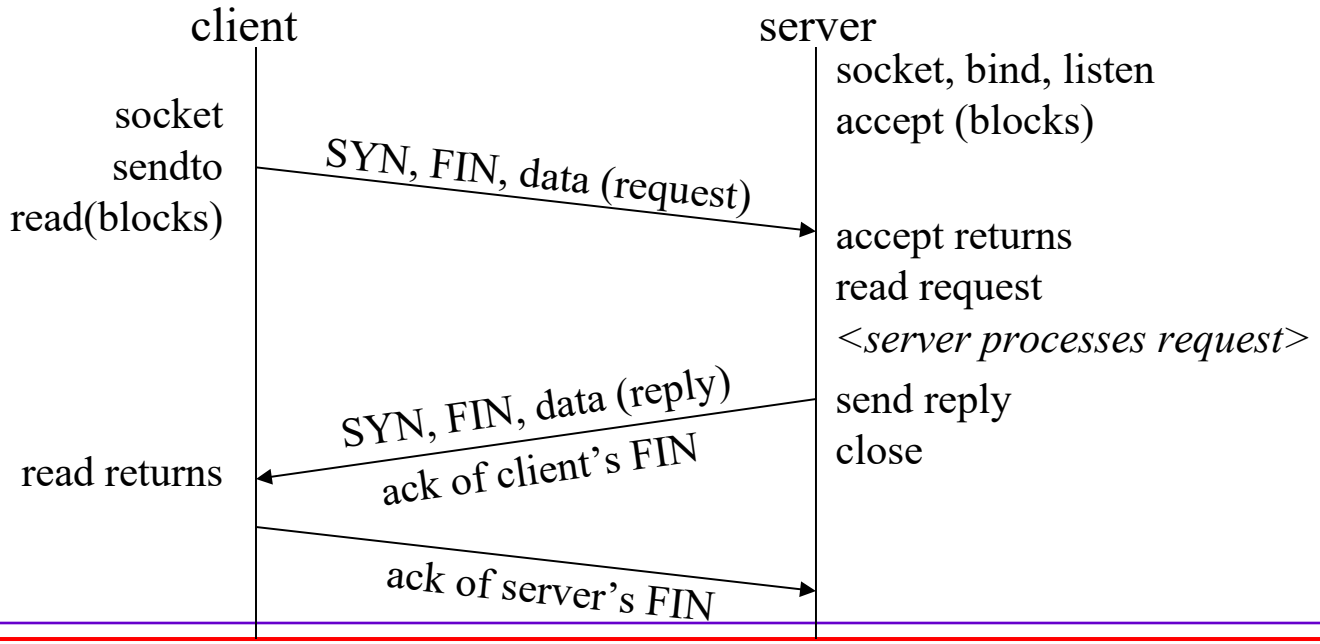
Sockets and Standard I/O

- **Example: str_echo Function Using Standard I/O**

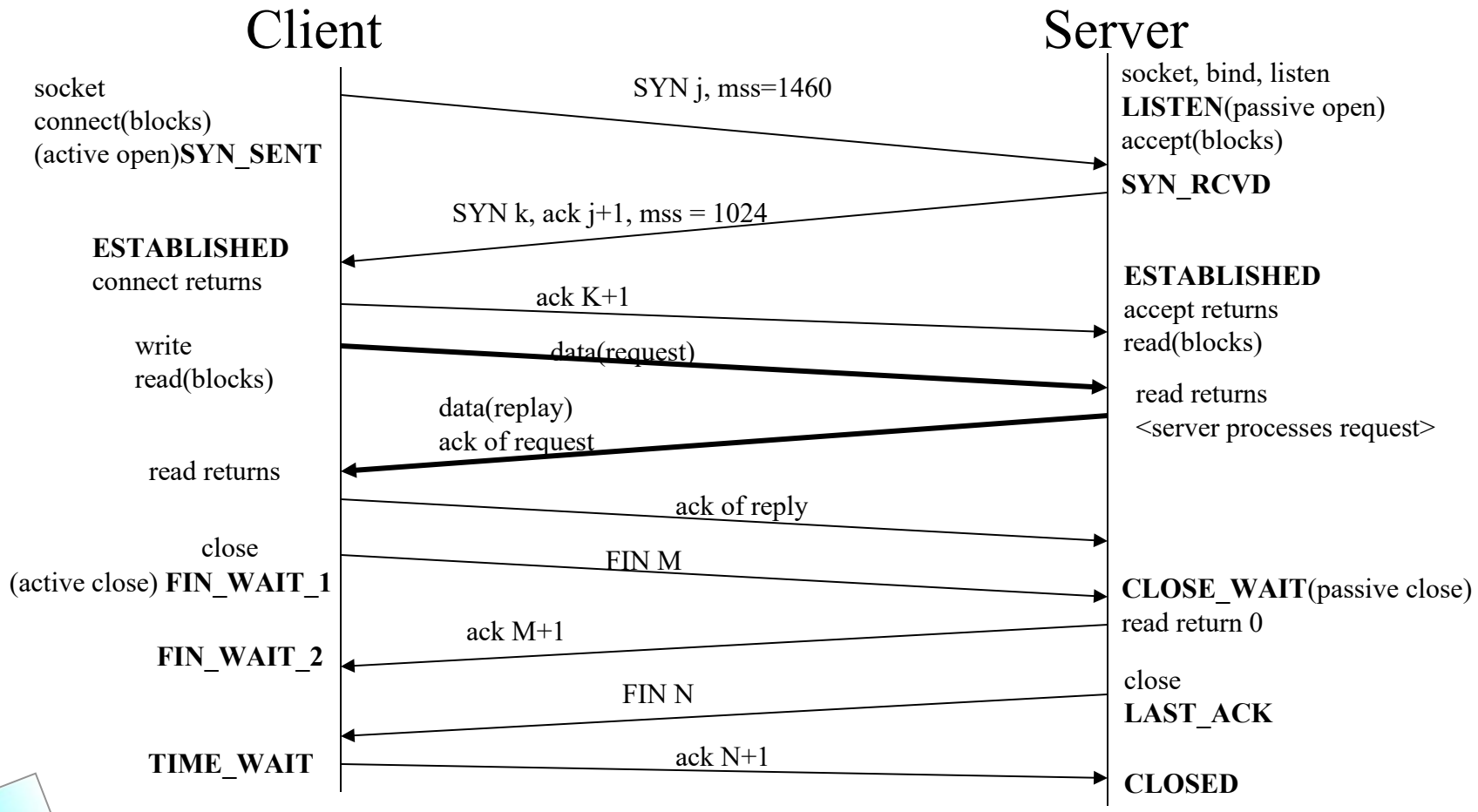
```
1 #include    "unp.h"
2
3 void
4 str_echo(int sockfd)
5 {
6     char        line[MAXLINE];
7     FILE        *fpin, *fpout;
8
9     fpin = Fdopen(sockfd, "r");
10    fpout = Fdopen(sockfd, "w");
11
12    for ( ; ; ) {
13        if (Fgets(line, MAXLINE, fpin) == NULL)
14            return;     /* connection closed by other end */
15
16        Fputs(line, fpout);
17    }
18 }
```

T/TCP: TCP for Transactions

- **can avoid 3-way handshake between hosts that have communicated with each other recently**
- **can combine the SYN, FIN, and data into a single segment**



T/TCP: comparison to general TCP



Chap. 14
Unix Domain Protocols

Unix Domain Protocols

- **Unix domain protocol** 은 **IPC** 대용으로 사용할 수 있음
  - stream socket (similar to TCP)
  - datagram socket (similar to UDP)
- **Unix Domain Socket Address Structure**

```
struct sockaddr_un {
    uint8_t         sun_len;
    sa_family_t     sun_family;     /* AF_LOCAL */
    char            sun_path[104];  /* null-terminated pathname */
};
```

Unix Domain Steram Protocol Echo Server

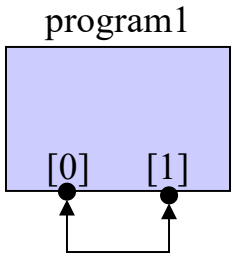
```
1 #include "unp.h"
2 int
3 main(int argc, char **argv)
4 {
                … /* □□ □□□ □□ □□  */
10    listenfd = Socket(AF_LOCAL, SOCK_STREAM, 0);
11    unlink(UNIXSTR_PATH);
12    bzero(&servaddr, sizeof(servaddr));
13    servaddr.sun_family = AF_LOCAL;
14    strcpy(servaddr.sun_path, UNIXSTR_PATH);
15    Bind(listenfd, (SA *) &servaddr, sizeof(servaddr));
16    Listen(listenfd, LISTENQ);
17    Signal(SIGCHLD, sig_chld);
18    for ( ; ; ) {
19      clilen = sizeof(cliaddr);
20      if ( (connfd = accept(listenfd, (SA *) &cliaddr, &clilen)) < 0) {
21        if (errno == EINTR)
22          continue;      /* back to for() */
23        else
24          err_sys("accept error");
25      }
26      if ( (childpid = Fork()) == 0) { /* child process */
27        Close(listenfd);  /* close listening socket */
28        str_echo(connfd); /* process the request */
29        exit(0);
30      }
31      Close(connfd);       /* parent closes connected socket */
32    }
33 }
```

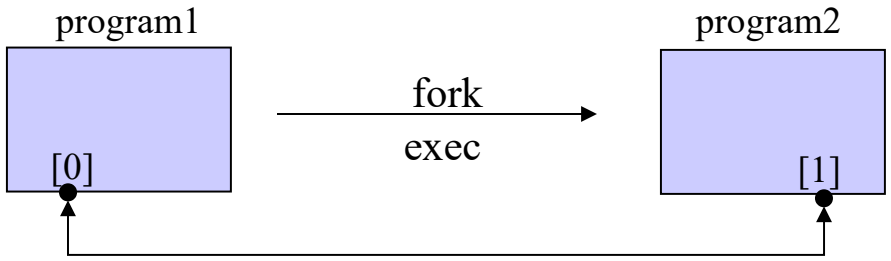
Unix Domain Steram Protocol Echo Client

```
1 #include    "unp.h"
2
3 int
4 main(int argc, char **argv)
5 {
6     int                 sockfd;
7     struct sockaddr_un  servaddr;
8
9     sockfd = Socket(AF_LOCAL, SOCK_STREAM, 0);
10
11    bzero(&servaddr, sizeof(servaddr));
12    servaddr.sun_family = AF_LOCAL;
13    strcpy(servaddr.sun_path, UNIXSTR_PATH);
14
15    Connect(sockfd, (SA *) &servaddr, sizeof(servaddr));
16
17    str_cli(stdin, sockfd);     /* do it all */
18
19    exit(0);
20 }
```

Unix Domain Protocol Examples

- **Passing Descriptors**





Unix Network Programming

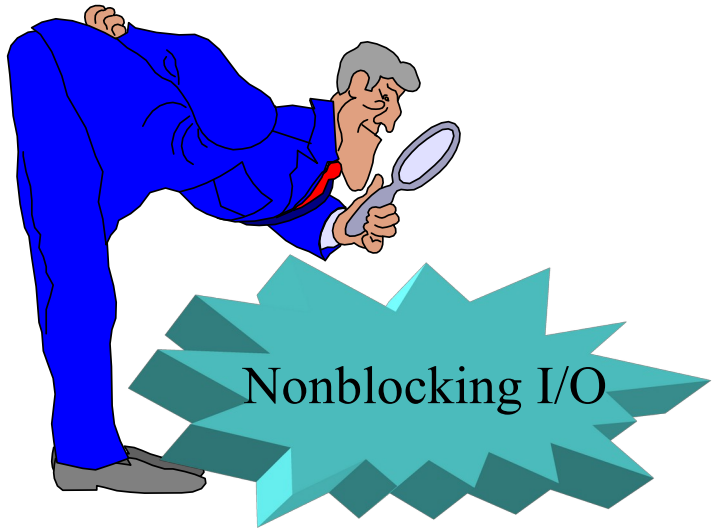
Chapter 15
Nonblocking I/O

Contents

- **Introduction**
- **Nonblocking Reads and Writes :**
  - str_cli function
- **Nonblocking connect**
  - Daytime Client
  - Web Client
  - accept
- **Summary**

Introduction

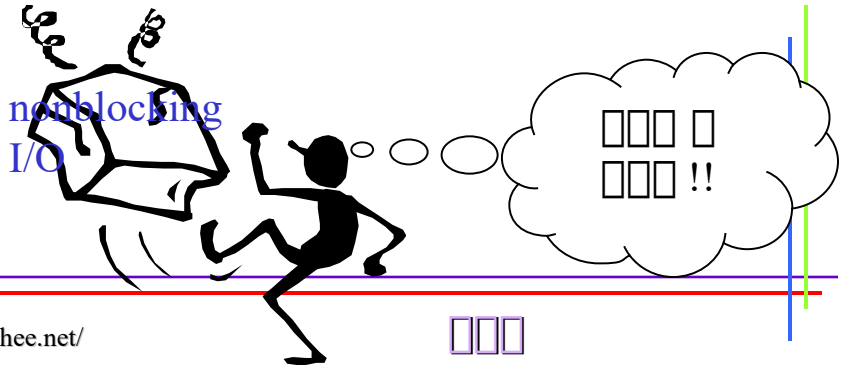
- **By default, sockets are blocking.**
- **□□ blocking □□ □□**
  - Input Operations
    - † read(), readv(), recv(), recvfrom()
  - Output operations
    - † write(), writev(), send(), sendto()
  - Accepting incoming connections
    - † accept()
  - Initiating outgoing connections
    - † connect()
- **Chapter 6 : various I/O model was learned**



Nonblocking Read and Writes

- **Goal in this section is to develop a version of this function that uses nonblocking I/O**
- **Unfortunately the addition of nonblocking I/O** *complicates the function's buffer management* **noticeably.**
- **Is it *worth the effort to code an application using nonblocking I/O*, given the complexity of the resulting code?**

Writer recommend the simple Approach !!!!

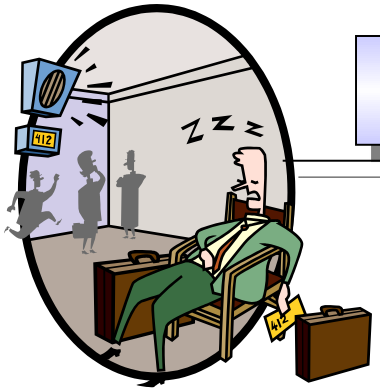


Nonblocking connect

- **There are 3 uses for nonblocking connect**
  - Overlap other processing with the 3-way handshaking
  - Establish multiple connections at the same time
  - Specify a time limit for *select()*

Must handled !!!

- Even though the socket is nonblocking, if the server to which we are connecting is on the same host, the connection establishment normally takes place immediately when we call connect()
- Berkeley-derived implementations have the following 2 rules regarding select() and nonblocking connects (1) when the connection completes successfully, the descriptor becomes writable, and (2) when the connection establishment encounters an error, the descriptor becomes both readable and writable



Daytime Client

```
1.   int  connect_nonb(int sockfd, const SA *saptr, socklen_t salen, int nsec)
2.   {  int      flags, n, error;   socklen_t   len;  fd_set      rset, wset;  struct
     timeval  tval;
3.     flags = Fcntl(sockfd, F_GETFL, 0);
4.     Fcntl(sockfd, F_SETFL, flags | O_NONBLOCK);
5.     error = 0;
6.     if ( (n = connect(sockfd, (struct sockaddr *) saptr, salen)) < 0)
7.       if (errno != EINPROGRESS)
8.         return(-1);
9.     /* Do whatever we want while the connect is taking place. */
10.    if (n == 0)
11.      goto done;  /* connect completed immediately */
12.    FD_ZERO(&rset);
13.    FD_SET(sockfd, &rset);
14.    wset = rset;
15.    tval.tv_sec = nsec;
16.    tval.tv_usec = 0;
17.    if ( (n = Select(sockfd+1, &rset, &wset, NULL, nsec ? &tval : NULL)) == 0) {
18.      close(sockfd);    /* timeout */
19.      errno = ETIMEDOUT;
20.      return(-1);   }
21.    if (FD_ISSET(sockfd, &rset) || FD_ISSET(sockfd, &wset)) {
22.      len = sizeof(error);
23.      if (getsockopt(sockfd, SOL_SOCKET, SO_ERROR, &error, &len) < 0)
24.        return(-1);     /* Solaris pending error */    } else
25.      err_quit("select error: sockfd not set");
26.  done:
27.    Fcntl(sockfd, F_SETFL, flags);  /* restore file status flags */
28.    if (error) {
29.      close(sockfd);    /* just in case */
30.      errno = error;
31.      return(-1);  }
32.    return(0);  }
```

Set socket noblocking

Overlap processing with connection establishment

Check for immediate completion

Call select

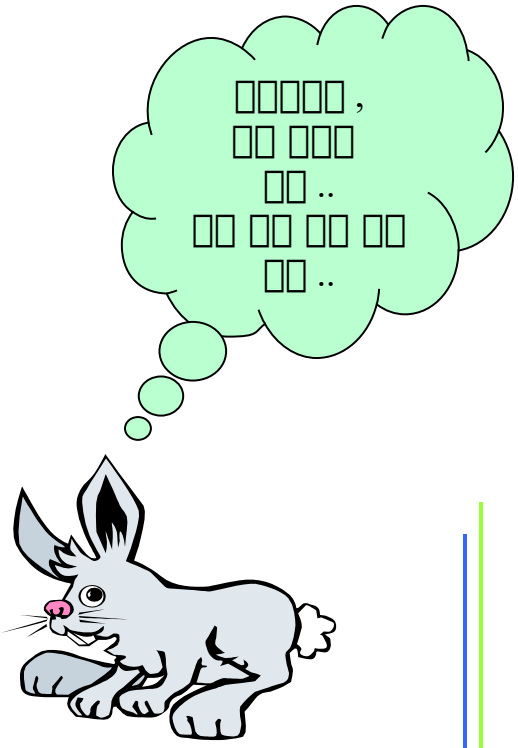
Check for readability or writabilty

Turn off nonblocking and return

WebClient

- **Performance of simulation Connections**

# simultaneous connections	clock time (sec) nonblocking	Clock time threads
1	6.0	6.3
2	4.1	4.2
3	3.0	3.1
4	2.8	3.0
5	2.5	2.7
6	2.4	2.5
7	2.3	2.3
8	2.2	2.3
9	2.0	2.2

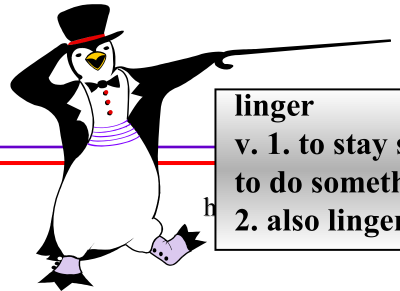


Nonblocking accept

```
1.      #include  "unp.h"
2.      int
3.      main(int argc, char **argv)
4.      {
5.        int         sockfd;
6.        struct linger   ling;
7.        struct sockaddr_in  servaddr;
8.        if (argc != 2)
9.          err_quit("usage: tcpcli <IPaddress>");
10.       sockfd = Socket(AF_INET, SOCK_STREAM, 0);
11.       bzero(&servaddr, sizeof(servaddr));
12.       servaddr.sin_family = AF_INET;
13.       servaddr.sin_port = htons(SERV_PORT);
14.       Inet_pton(AF_INET, argv[1], &servaddr.sin_addr);
15.       Connect(sockfd, (SA *) &servaddr, sizeof(servaddr));
16.       ling.l_onoff = 1;   /* cause RST to be sent on close() */
17.       ling.l_linger = 0;
18.       Setsockopt(sockfd, SOL_SOCKET, SO_LINGER, &ling, sizeof(ling));
19.       Close(sockfd);
20.       exit(0);
21.     }
```

tcp 문제점은 연결을 하려는 듯하다가 연결을 하지 않을 경우 , 연결이 끊어졌는데도 불구 계속해서 3way 악수를 하기때문에 , nonblocking I/O 를 할때 다음과 같이 socket option 을 이용해야 함다 .

Set SO_LINGER socket Option



linger
v. 1. to stay somewhere longer or to take longer to do something than usual
2. also linger on to be slow to disappear

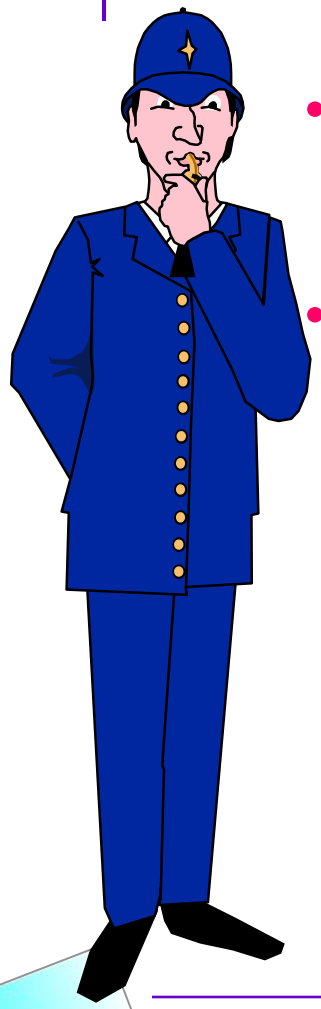
Unix Network Programming

Chapter 16
ioctl Operation

Contents

- Introduction
- ioctl Function
- Socket Operations
- File Operations
- Interface Configuration
- get_fif_info Function
- Interface Operations
- ARP Cache Operations
- Routing Table Operations
- Summary

Introduction

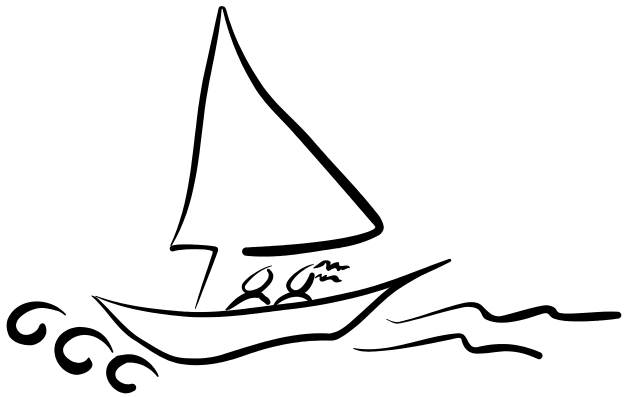


- **ioctl() has traditionally been the system interface used for everything that didn't fit into some other nicely defined category.**
- **This remain for implementation-dependent features related to network programming**
  - obtaining the interface information
  - accessing the the routing table
  - accessing the ARP cache

ioctl Function

```
#include <unistd.h>
 int  ioctl(int fd, int request, …./*void *arg */ );
                                        Returns : 0 if OK, -1 on error
```

- **socket operations**
- **file operations**
- **interface operations**
- **ARP cache operations**
- **routing table operations**
- **stream system(Chapter 33)**



Socket Operation

- **SIOCATMARK**
  - socket's read pointer is currently at the out-of-band mark, or a zero value if the read pointer is not at the out-of-band mark.
- **SIOCGPGRP**
  - PID or the process group ID
- **SIOCSPGRP**
  - set either the PID or the process tourp ID

chapter 7
fcntl() □□

File Operations

- **FIONBIO**
  - nonblocking flag for the socket is cleared or turned on, depending whether the third argument to ioct() points to a zero or nonzero value
- **FIOASYNC**
  - governs the receipt of asynchronous I/O signals(SIGIO)
- **FIONREAD**
  - the number of bytes in the socket receive buffer
- **FIOSETOWN**
  - same as SIOCSPGRP
- **FIOGETOWN**
  - same as SIOCGPGRP

Interface Operations

- **SIOCGIFADDR**
  - Return the unicast address
- **SIOCSIFADDR**
  - Sets the interface address
- **SIOCGIFFLAGS**
  - Return the interface flags
- **SIOCSIFFLAGS**
  - Set the interface flags
- **SICGIFDSTTADDR**
  - Return the point-to-point address
- **SICSIFDSTADDR**
  - Set the point-to-point address
- **SIOCSIFBRDADDR**
  - Set the broadcast address
- **SIOCGIFNETMASK**
  - Return the subnet mask
- **SIOCSIFNETMASK**
  - Set the subnet mask
- **SIOCGIFMETRIC**
  - Return the interface metric
- **SIOCIFMETRIC**
  - Set the interface routing metric

ARP Cache Operations

```
1.      #include  <net/if_arp.h>
2.      int main(int argc, char **argv)
3.      {
4.       int        family, sockfd;
5.       char       str[INET6_ADDRSTRLEN];
6.       char       **pptr;
7.       unsigned char   *ptr;
8.       struct arpreq   arpreq;
9.       struct sockaddr_in  *sin;
10.      pptr = my_addrs(&family);
11.      for ( ; *pptr != NULL; pptr++) {
12.        printf("%s: ", Inet_ntop(family, *pptr, str, sizeof(str)));
13.        switch (family) {
14.        case AF_INET:
15.          sockfd = Socket(AF_INET, SOCK_DGRAM, 0);
16.          sin = (struct sockaddr_in *) &arpreq.arp_pa;
17.          bzero(sin, sizeof(struct sockaddr_in));
18.          sin->sin_family = AF_INET;
19.          memcpy(&sin->sin_addr, *pptr, sizeof(struct in_addr));
20.          Ioctl(sockfd, SIOCGARP, &arpreq);
21.          ptr = &arpreq.arp_ha.sa_data[0];
22.          printf("%x:%x:%x:%x:%x:%x\n", *ptr, *(ptr+1),
23.              *(ptr+2), *(ptr+3), *(ptr+4), *(ptr+5));
24.          break;
25.        default:
26.          err_quit("unsupported address family: %d", family);
27.        }  }  exit(0);}
```

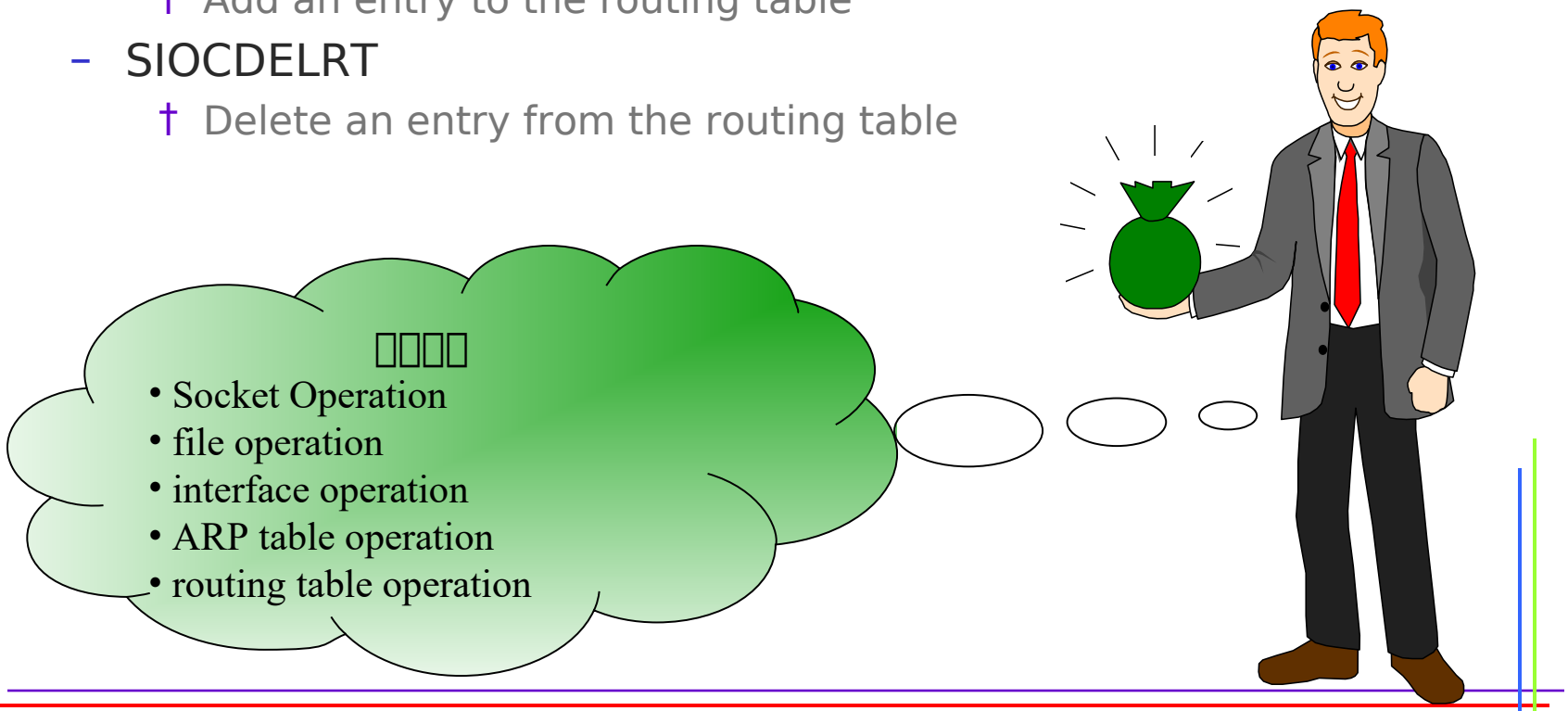
get list of address and loop through each one

Print IP address

Issue ioctl and print hardware address

Routing Table Operations

- **TWO ioctl() requests provided**
  - SIOCADDRT
    - † Add an entry to the routing table
  - SIOCDELRT
    - † Delete an entry from the routing table



UNIX Network Programming
(chap 17 - chap 18)

Contents

Chap.17 - Routing Socket

- Routing table □□ □□ □□
- Message Types
- Structure Types
- Constants in routing messages
- RMT_GET example
- sysctl Operations

We must study … ?

📖 Network 상태를 관리하기 위한 Routing table 로의 접근방법 (2 가지) 과 사용법

1. Routing socket 사용 (superuser)
2. *sysctl* 사용

Routing table 로의 접근 방법

1. Use ioctl command

2. Use netstat program

3. Use routing daemon (only superuser)
   - monitor ICMP redirection message by creating a raw ICMP socket
   - routing domain 에서 지원되는 유일한 socket type 은 raw socket 이다 .

4. Use sysctl command

Message Types

☞ Routing table □ □□ / □□ □□□ Message type - <net/route.h>

Message type	To kernel	From kernel	Description	Structure type
RTM_ADD	•	•	add route	rt_msghdr
RTM_CHANGE	•	•	change gateway, metrics, or flags	rt_msghdr
RTM_DELADDR		•	address being removed from interface	ifa_msghdr
RTM_DELETE	•	•	delete route	rt_msghdr
RTM_GET	•	•	report metrics and other route information	rt_msghdr
RTM_IFINFO		•	interface going up, down etc.	if_msghdr
RTM_LOCK	•	•	lock specified metrics	rt_msghdr
RTM_LOSING		•	kernel suspects route is failing	rt_msghdr
RTM_MISS		•	lookup failed on this address	rt_msghdr
RTM_NEWADDR		•	address being added to interface	ifa_msghdr
RTM_REDIRECT		•	kernel told to use different route	rt_msghdr
RTM_RESOLVE		•	request to resolve destination to link layer address	rt_msghdr

```
typedef struct rt_msghdr {
        ushort_t rtm_msglen;    /* to skip over non-understood messages */
        uchar_t  rtm_version;   /* future binary compatibility */
        uchar_t  rtm_type;      /* message type */
        ushort_t rtm_index;     /* index for associated ifp */
        int      rtm_flags;     /* flags, incl. kern & message, e.g. DONE */
        int      rtm_addrs;     /* bitmask identifying sockaddrs in msg */
        pid_t    rtm_pid;       /* identify sender */
        int      rtm_seq;       /* for sender to identify action */
        int      rtm_errno;     /* why failed */
        int      rtm_use;       /* from rtentry */
        uint_t   rtm_inits;     /* which metrics we are initializing */
        struct   rt_metrics rtm_rmx; /* metrics themselves */
} rt_msghdr_t;
```

```
typedef struct if_msghdr {
        ushort_t ifm_msglen;    /* to skip over non-understood messages */
        uchar_t  ifm_version;   /* future binary compatability */
        uchar_t  ifm_type;      /* message type */
        int      ifm_addrs;     /* like rtm_addrs */
        int      ifm_flags;     /* value of if_flags */
        ushort_t ifm_index;     /* index for associated ifp */
        struct   if_data ifm_data; /* statistics and other data about if */
} if_msghdr_t;
```

```
typedef struct ifa_msghdr {
        ushort_t ifam_msglen;   /* to skip over non-understood messages */
        uchar_t  ifam_version;  /* future binary compatability */
        uchar_t  ifam_type;     /* message type */
        int      ifam_addrs;    /* like rtm_addrs */
        int      ifam_flags;    /* route flags */
        ushort_t ifam_index;    /* index for associated ifp */
        int      ifam_metric;   /* value of ipif_metric */
} ifa_msghdr_t;
```

Constants in routing messages

Bitmask		Array index		Socket address structure containing
constant	value	constant	value	
RTA_DST	0x01	*RTAX_DST*	0	destination address
RTA_GATEWAY	0x02	*RTAX_GATEWAY*	1	gateway address
RTA_NETMASK	0x04	*RTAX_NETMASK*	2	network mask
RTA_GENMASK	0x08	*RTAX_GENMASK*	3	cloning mask
RTA_IFP	0x10	*RTAX_IFP*	4	interface name
RTA_IFA	0x20	*RTAX_IFA*	5	interface address
RTA_AUTHOR	0x40	*RTAX_AUTHOR*	6	author of redirect
RTA_BRD	0x80	*RTAX_BRD*	7	broadcast or point-to-point destination address
		RTAX_MAX	8	Max #elements

RMT_GET example (1)

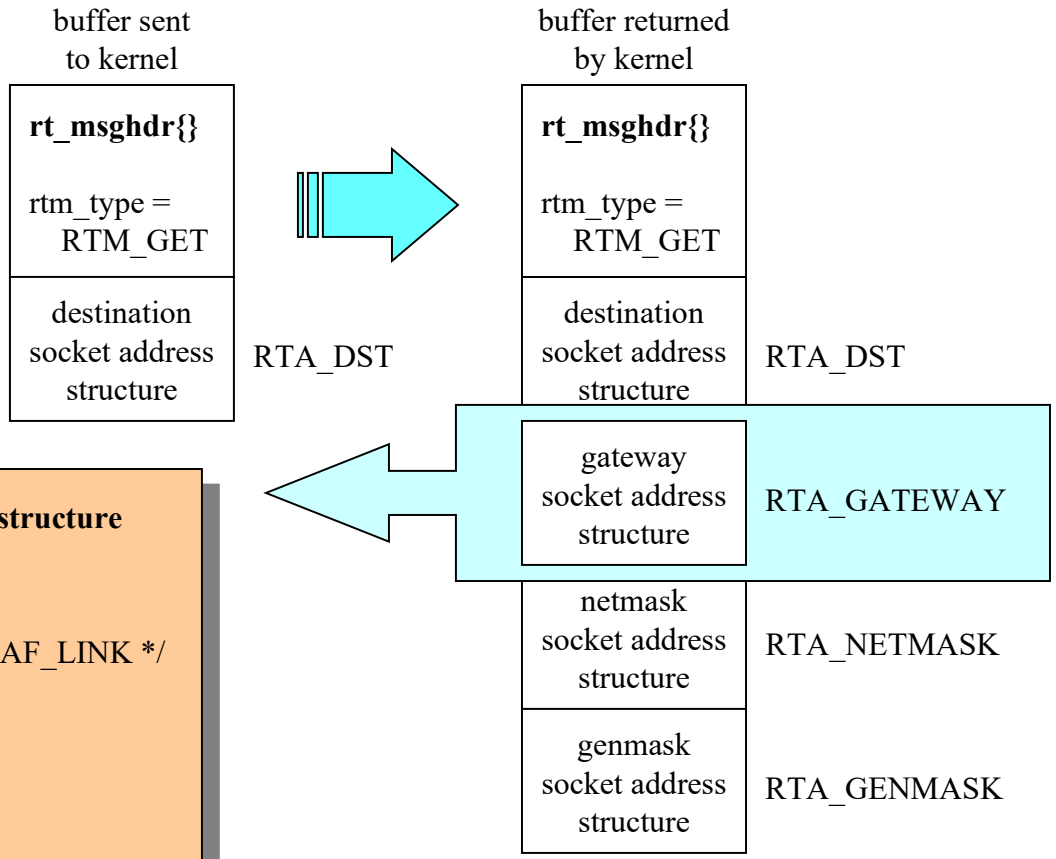
```
#include       "unproute.h"

#define BUFLEN  (sizeof(struct rt_msghdr) + 512)
                        /* 8 * sizeof(struct sockaddr_in6) = 192 */
#define SEQ             9999

int
main(int argc, char **argv)
{
        int                     sockfd;
        char                    *buf;
        pid_t                   pid;
        ssize_t                 n;
        struct rt_msghdr        *rtm;
        struct sockaddr         *sa, *rti_info[RTAX_MAX];
        struct sockaddr_in      *sin;

        if (argc != 2)
                err_quit("usage: getrt <IPaddress>");

        sockfd = Socket(AF_ROUTE, SOCK_RAW, 0); /* need superuser privileges */

        buf = Calloc(1, BUFLEN);        /* and initialized to 0 */

        rtm = (struct rt_msghdr *) buf;
        rtm->rtm_msglen = sizeof(struct rt_msghdr) + sizeof(struct sockaddr_in);
        rtm->rtm_version = RTM_VERSION;
        rtm->rtm_type = RTM_GET;
        rtm->rtm_addrs = RTA_DST;
        rtm->rtm_pid = pid = getpid();
        rtm->rtm_seq = SEQ;

        sin = (struct sockaddr_in *) (rtm + 1);
        sin->sin_family = AF_INET;
    Inet_pton(AF_INET, argv[1], &sin->sin_addr);

        Write(sockfd, rtm, rtm->rtm_msglen);

        do {
                n = Read(sockfd, rtm, BUFLEN);
        } while (rtm->rtm_type != RTM_GET || rtm->rtm_seq != SEQ ||
                        rtm->rtm_pid != pid);

        rtm = (struct rt_msghdr *) buf;
        sa = (struct sockaddr *) (rtm + 1);
        get_rtaddrs(rtm->rtm_addrs, sa, rti_info);
        if ( (sa = rti_info[RTAX_DST]) != NULL)
                printf("dest: %s\n", Sock_ntop_host(sa, sa->sa_len));   /* sa_len = 16 or 24 */

        if ( (sa = rti_info[RTAX_GATEWAY]) != NULL)
                printf("gateway: %s\n", Sock_ntop_host(sa, sa->sa_len));  /* sa_len = 16 or 24 */

        if ( (sa = rti_info[RTAX_NETMASK]) != NULL)
                printf("netmask: %s\n", Sock_masktop(sa, sa->sa_len));  /* sa_len = 0,5,6,7,8 */

        if ( (sa = rti_info[RTAX_GENMASK]) != NULL)
                printf("genmask: %s\n", Sock_masktop(sa, sa->sa_len));  /* sa_len = 0,5,6,7,8 */

        exit(0);
}
```

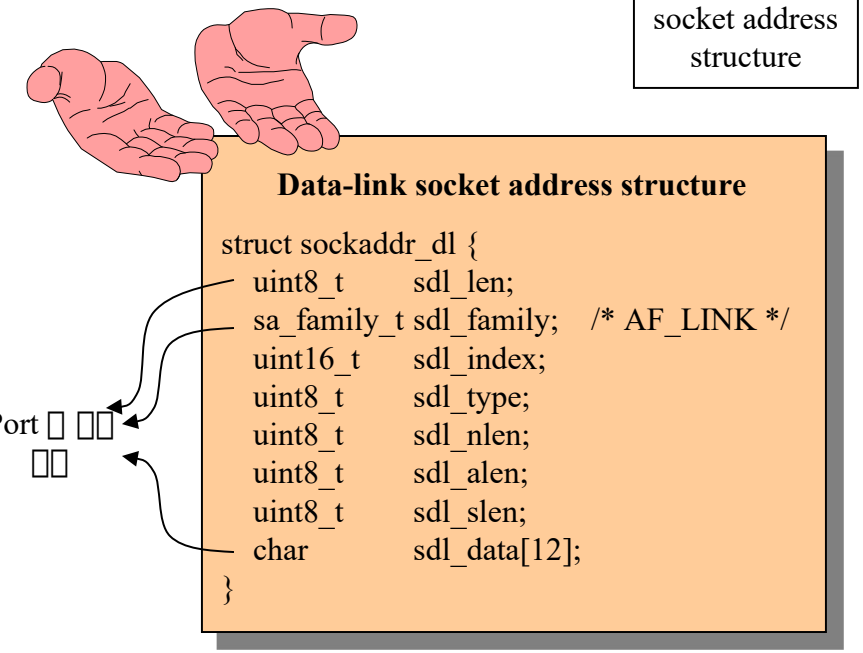
Page 454 □□

RMT_GET example (2)

Example >

bsdi # getrt 206.62.226.32
dest : 206.62.226.32
gateway : AF_LINK, index=2
netmask : 255.255.255.224



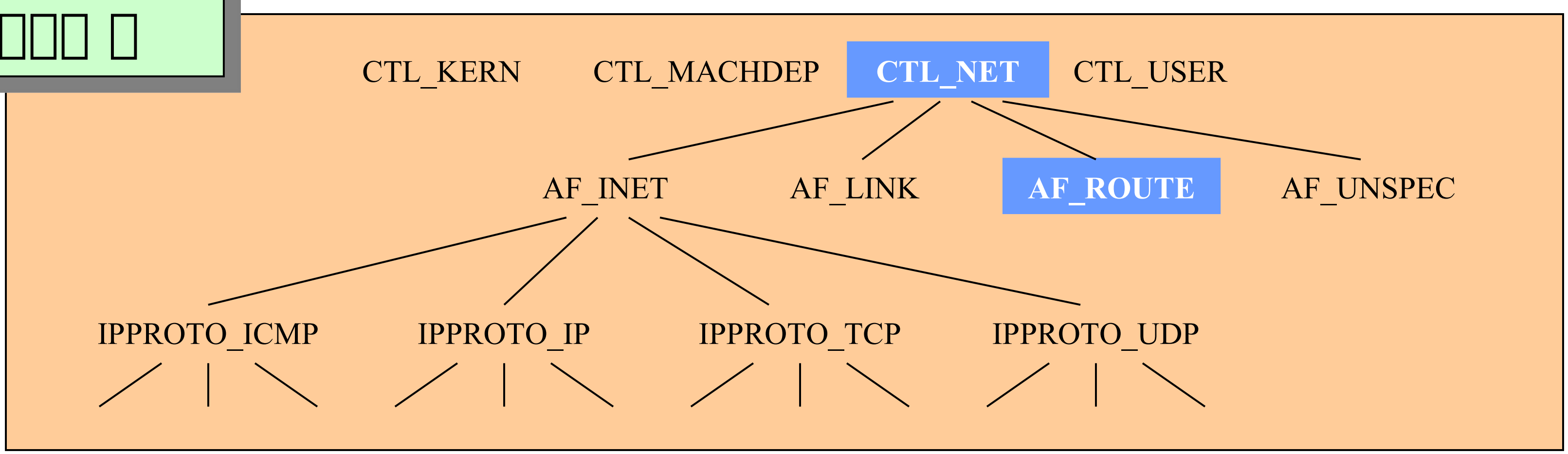


sysctl Operations (1)

- routing table □ interface list □□□ □□□□.
- □□ process □ □□ □□□□.
  ↔ cf) routing socket □□□ superuser □□ □□□□.

```
#include <sys/param.h>
#include <sys/sysctl.h>

int sysctl( int *name, u_int namelen, void *oldp, size_t *oldlenp,
                              void *newp, size_t newlen );

                                              return : 0 if OK, -1 on error
```

□□□ □

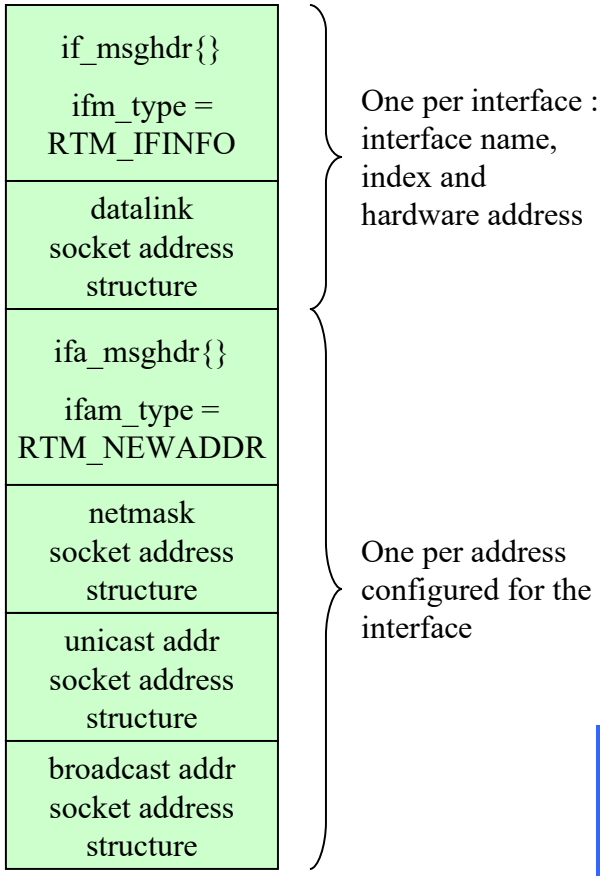


sysctl Operations (2)

name[]	Return IPv4 routing table	Return IPv4 ARP cache	Return IPv4 interface list
0	CTL_NET	CTL_NET	CTL_NET
1	AF_ROUTE	AF_ROUTE	AF_ROUTE
2	0	0	0
3	AF_INET	AF_INET	AF_INET
4	NET_RT_DUMP	NET_RT_FLAGS	NET_RT_IFLIST
5	0	RTF_LLINFO	0

```
mib[0] = CTL_NET;
mib[1] = AF_ROUTE;
mib[2] = 0;
mib[3] = family;
mib[4] = NET_RT_IFLIST;
mib[5] = flag    /* interface index or 0*/
sysctl( mib, 6, buff, lenp, NULL, 0 );
```

buffer returned by kernel



Contents

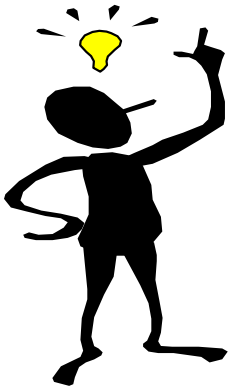
Chap.18 - Broadcast

- Broadcast support and address
- Broadcast datagram flow example
- Race Conditions
  - blocking and unblocking the signal
  - blocking and unblocking the signal with pselect
  - using sigsetjmp and siglongjmp
  - using IPC(Interprocess communication) from signal handler to function

We must study … ?

- Unicast 와 Broadcast 의 차이
- Broadcast 의 개념과 방법
- Race condition 의 원인과 그 해결책 (4 가지)

Broadcast support and address



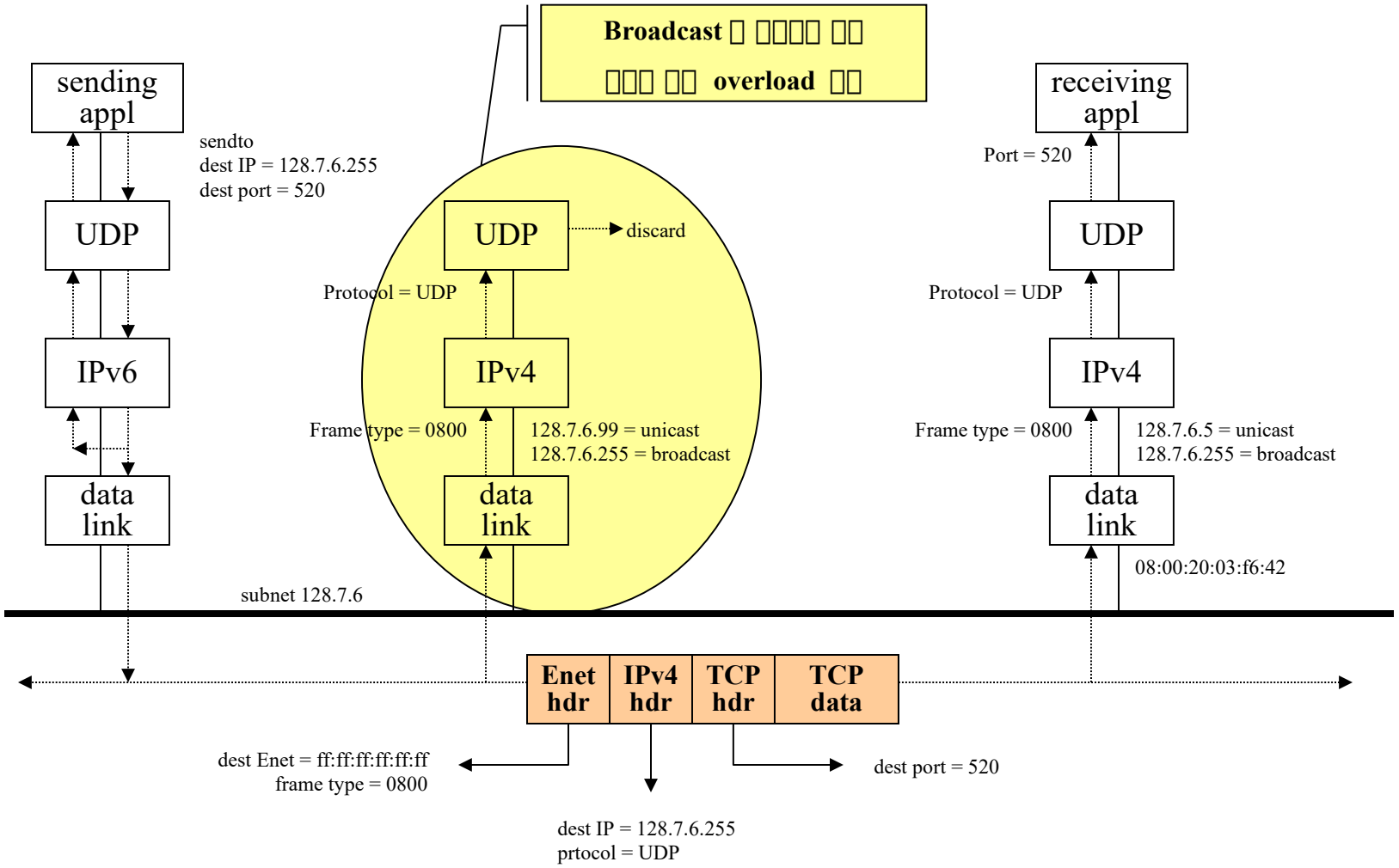
1. Multicast 지원은 IPv4 에서는 optional 하지만 IPv6 에서는 필수적이다 .
2. Broadcast 지원은 IPv6 에서는 제공되지 않는다 . IPv6 에서는 broadcast 를 multicast 로 다시 작성해 주어야 한다 .
3. Broadcast 와 multicast 는 UDP 를 요구하며 TCP 에서는 동작하지 않는다 .

All bits ‘1’

1. Subnet-directed broadcast address : {netid, subnetid, -1 }
2. All-subnets-directed broadcast address : { netid, -1, -1 }
3. Network-directed broadcast address : { netid, -1 }→ Subnet 이 없는 경우
4. Limited broadcast address : { -1, -1, -1 }

→ Router 내의 망
현재는 1 번으로 변환되어
사용되는 경우도 있다 .

Broadcast datagram flow example



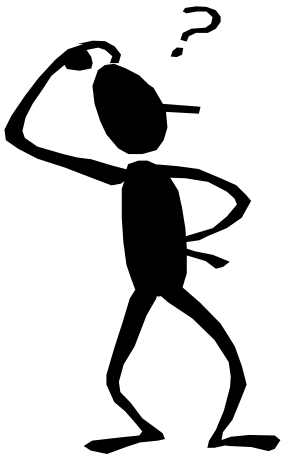
Example continue ...

```
[54 feel:kjschaos ~/mywork/network/unpv12e/bcast ] udpcli03 150.150.55.127
hi
from 150.150.55.52: Mon Jan 10 18:01:23 2000
from 150.150.55.53: Mon Jan 10 18:04:38 2000
from 150.150.55.55: Mon Jan 10 18:05:48 2000
from 150.150.55.99: Mon Jan 10 17:50:41 2000
from 150.150.55.10:  6:02:48 2000-01-10
from 150.150.55.54: Mon Jan 10 18:06:23 2000

[55 feel:kjschaos ~/mywork/network/unpv12e/bcast ] udpcli03 255.255.255.255
hi
from 150.150.55.52: Mon Jan 10 18:01:42 2000
from 150.150.55.53: Mon Jan 10 18:04:57 2000
from 150.150.55.55: Mon Jan 10 18:06:07 2000
from 150.150.55.99: Mon Jan 10 17:51:00 2000
from 150.150.55.10:  6:03:07 2000-01-10
from 150.150.55.100: Mon Jan 10 06:20:44 2000
from 150.150.55.54: Mon Jan 10 18:06:42 2000

[56 feel:kjschaos ~/mywork/network/unpv12e/bcast ] udpcli03 150.150.55.52
hi
from 150.150.55.52: Mon Jan 10 18:02:01 2000
```

Race Conditions (1)

When happen ?

- 여러 개의 process 가 공유하는 data 를 참조할 때
- Signal 을 다룰 때



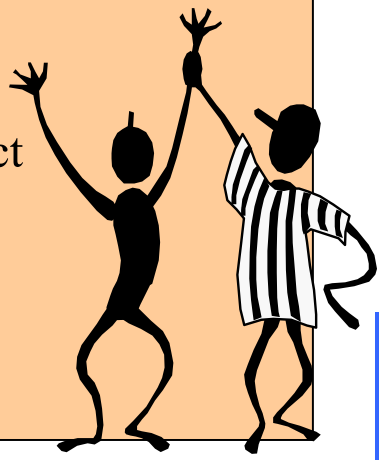
Why concern ?

? Broadcast 에 의한 응답이 여러 host 로부터 올 경우 signal 에 의한 원하지 않는 곳에서의 영원한 blocking 을 염두에 두고 처리해 주어야 한다 .

Race Conditions (2)

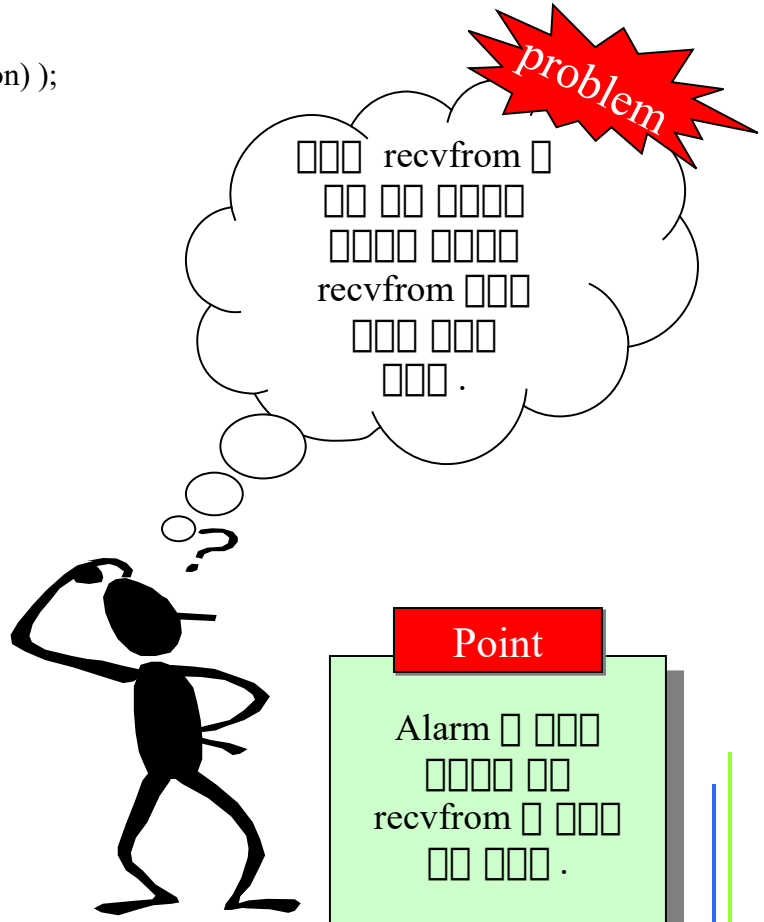
Solution

- 여러 개의 process 가 공유하는 data 를 참조할 때
  1) mutual exclusion variables(상호 배제 변수)
  2) condition variables
- Signal 을 다룰 때
  1) blocking and unblocking the signal
  2) blocking and unblocking the signal with pselect
  3) using sigsetjmp and siglongjmp
  4) using IPC(Interprocess communication)
     from signal handler to function



Race Conditions - Signal case (1)

```
void dg_cli( … )
{
   Setsockopt( sockfd, SOL_SOCKET, SO_BROADCAST, &on, sizeof(on) );
   ...
   Sigemptyset( &sigset_alrm );
   Sigaddset( &sigset_alrm, AIGALRM );

   Signal( SIGALRM, recvfrom_alarm );

   while ( … ) {
      Sendto( sockfd, … )

      alarm(5);
      for ( ; ; ) {
                Sigprocmask( SIG_UNBLOCK, &sigset_alrm, NULL );
                n = recvfrom( sockfd, recvline, … );
                Sigprocmask( SIG_BLOCK, &sigset_alrm, NULL );
                …
      }
   }
}

static vois recvfsrom_alarm( int signo )
{
   return;
}
```

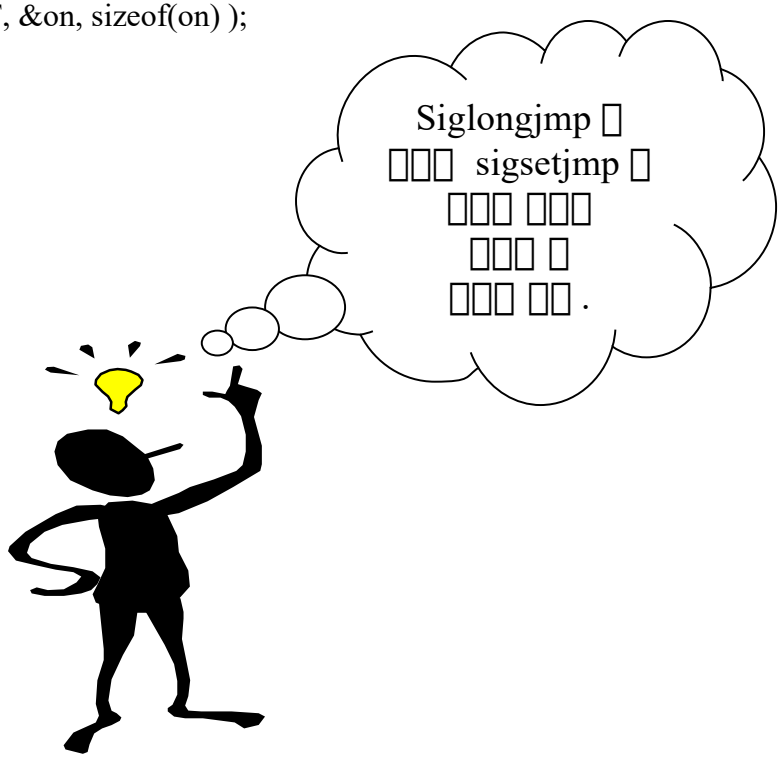


Race Conditions - Signal case (2)

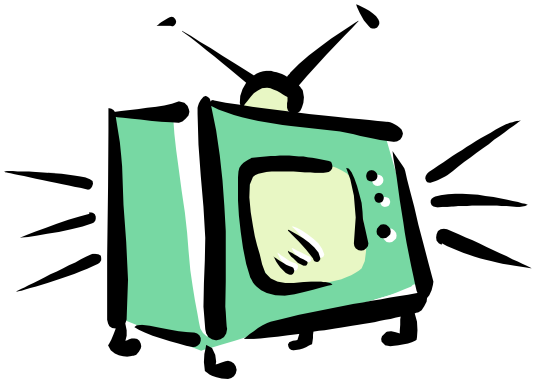

```
void dg_cli( … )
{
   Setsockopt( sockfd, SOL_SOCKET, SO_BROADCAST, &on, sizeof(on) );
   ...
   Sigemptyset( &sigset_alrm ); Sigemptyset( &sigset_alrm );
   Sigaddset( &sigset_alrm, AIGALRM );

   Signal( SIGALRM, recvfrom_alarm );

   while ( … ) {
      Sendto( sockfd, … )
      Sigprocmask( SIG_BLOCK, &sigset_alrm, NULL );

      alarm(5);
      for ( ; ; ) {
                  FD_SET( sockfd, &rset );
                  n = pselect( sockfd+1, &rset, NULL, NULL, NULL, &sigset_empty );
                  ...
      }
   }
}


int pselect( ... )
{
   sigprocmask( SIG_SETMASK, signask, &savemask );  /* caller's mask */
   n = select( … );
   sigprocmask( SIG_SETMASK, &savemask, NULL );  /* restore maks */
}
```

Race Conditions - Signal case (3)

```
void dg_cli( … )
{
   Setsockopt( sockfd, SOL_SOCKET, SO_BROADCAST, &on, sizeof(on) );

   Signal( SIGALRM, recvfrom_alarm );

   while ( … ) {
      Sendto( sockfd, … )

      alarm(5);
      for ( ; ; ) {
                  if ( sigsetjmp( jmpbuf, 1 ) != 0 );
                     break;
                  n = Recvfrom( sockfd, recvline, … );
                  ...
      }
   }
}

static vois recvfsrom_alarm( int signo )
{
   siglongjmp( jmpbuf, 1 );
}
```



Unix Network Programming



Chapter 19.
Multicasting

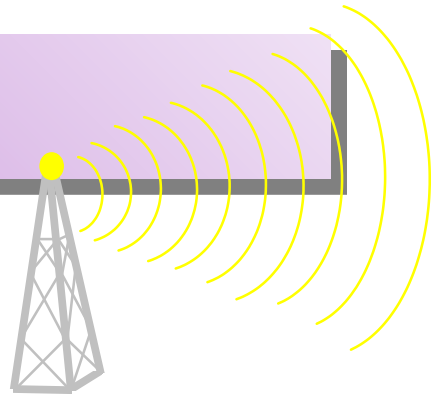
Contents

- **Introduction**
- ***Multicast Address***
- ***Multicasting vs Broadcasting on A LAN***
- **Multicast Socket Options**
- ***mcast_join()* and Related Functions**
- ***dg_cli()* Functions Using Multicasting**
- **Receiving MBone Session Announcements**
- **Sending and Receiving**
- **SNTP : Simple Network Time Protocol**
- **SNTP(Continued)**
- **Summary**

Introduction

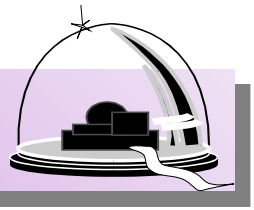
- **Broadcasting is normally limited to a LAN**
- **Multicasting can be used on a LAN or across a WAN.**
- **Five socket options**
  - 3 that affect the sending of UDP datagrams to a multicast address, and
  - 2 that affect the host's receptions of multicast datagrams

Multicast Address



- **IPv4 Class D Address**
  - range : 224.0.0.0 ~ 239.255.255.255
  - low order bit 28 bits : multicast group ID
  - 32-bit address : group address
  - IPv4 Ethernet multicast address : *01:00:5e*
- **IPv6 Multicast Address**
  - high-order byte of an IPv6 multicast address : *ff*
  - special IPv6 Multicast address
    - † ff02::1 is all-node group
    - † ff02::2 is the all-routers group

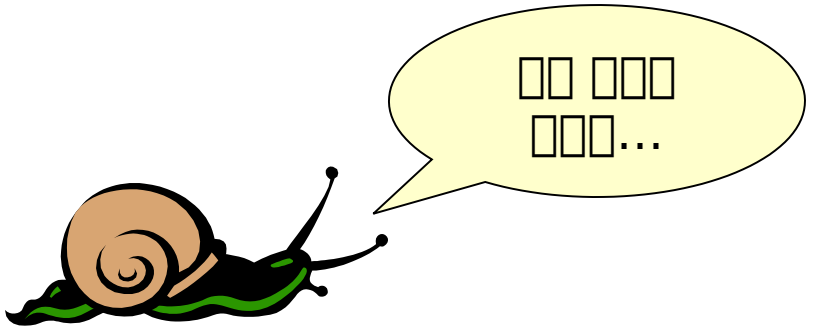
Multicast Socket Options



Command	Datatype	Descrption
IP_ADD_MEMBERSHIP	struct ip_mreq	join a multicast group
IP_DROP_MEMBERSHIP	struct ip_mreq	leave a multicast group
IP_MULTICAST_IF	struct in_addr	specify default interface for outgoing multicasts
IP_MULTICAST_TTL	u_char	specify TTL for outgoing multicasts
IP_MULTICAST_LOOP	u_char	enable or disable loopback of outgoing multicasts
IPV6_ADD_MEMBERSHIP	struct ipv6_mreq	join a multicast group
IPV6_DROP_MEMBERSHIP	struct ipv6_mreq	leave a multicast group
IPV6_MULTICAST_IF	u_int	specify default interface for outgoing multicasts
IPV6_MULTICAST_TTL	int	specify TTL for outgoing multicasts
IPV6_MULTICAST_LOOP	u_int	enable or disable loopback of outgoing multicasts

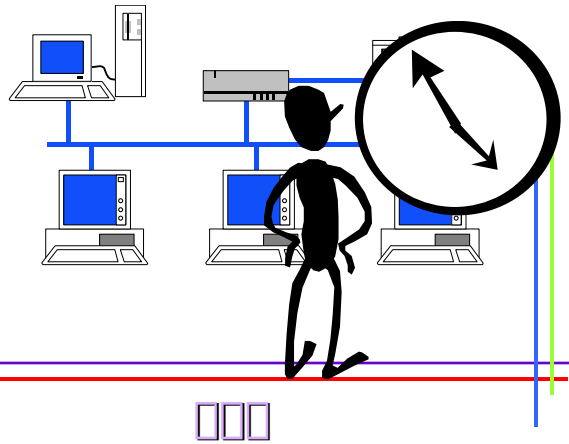
Receiving MBone Session Announcements

- **MBone**
  - multimedia conference
- **SAP**
  - Session Announcement Protocol
- **SDP**
  - Session Description Protocol

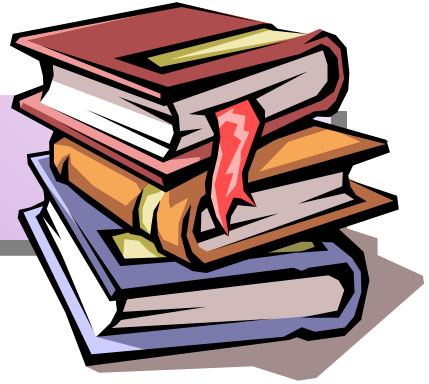


SNTP (Simple Network Protocol)

- **NTP(Network Time Protocol)**
  - sophisticated protocol for synchronizing clocks across a WAN or LAN, and
  - can often millisecond accuracy
- **SNTP**
  - common for a few hosts on a LAN to synchronize their clocks across the Internet to other NTP hosts, and
  - then redistribute this time on the LAN using either broadcasting or multicasting



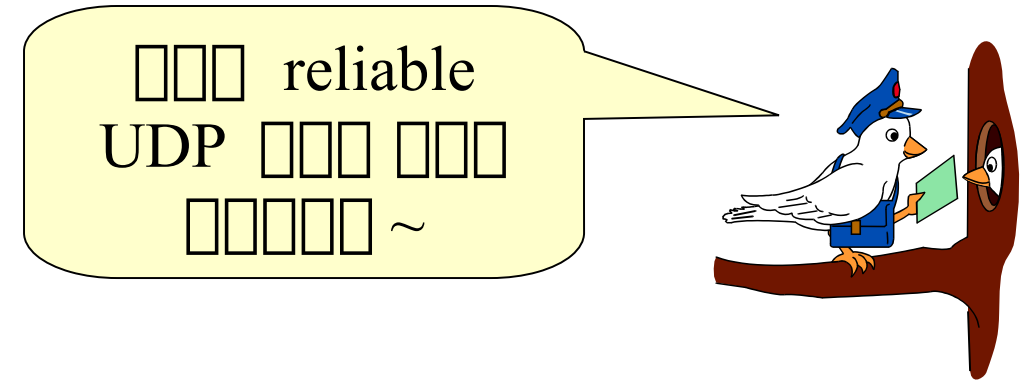
Summary



- **Multicast application starts by joining the multicast group assigned to the application**
  - tell the IP layer to join the group
- **Using hardware filtering reduces the load on all the other hosts that are not participating in the application**
- **5 Five socket option**
  - join a multicast group on an interface
  - leave a multicast group,
  - set the default interface for outgoing multicasts,
  - set the TTL or hop limit for outgoing multicasts,
  - enable or disable loopback of multicasts

for receiving

for sending

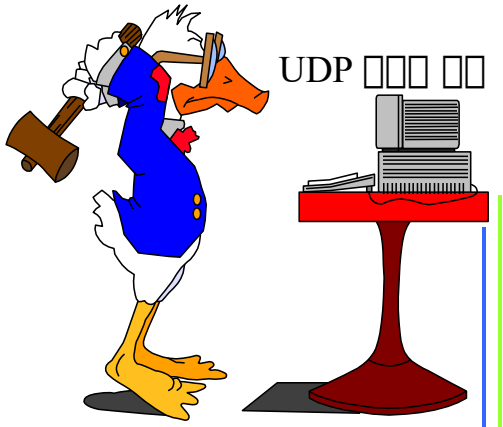


Unix Network Programming

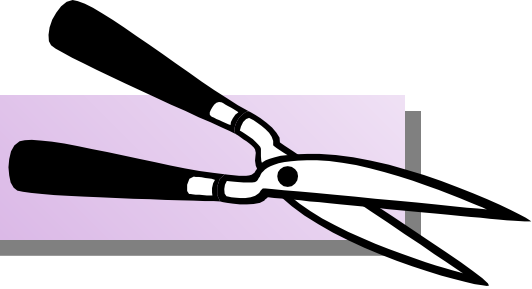
Chapter 20
Advanced UDP Sockets

Contents

- Introduction
- Receiving Flags, Destination IP address, and Interface Index
- **Datagram Truncation**
- **When to Use UDP Instead of TCP**
- *Adding Reliability to a UDP Application*
- Biding Interface Addresses
- **Concurrent UDP servers**
- IPV6 Packet Information
- **Summary**



Datagram Truncation



- **When a UDP datagram arrives that arrives that is larger than the application's buffer, *recvmsg* sets the *MSG_TRUNC* flag.**
- **3** □□ □□□ □□□□
  - Discard the exess bytes and return the MSG_TRUNC flag to the application. This requires that the application call recvmsg to receive the flag

BSD/OS

  - Discard the excess bytes but do not tell the application

solaris 2.5 →

  - Keep the excess bytes and return them in subsequent read operations on the socket

early ver. of SVR4 →

When to Use UDP Instead of TCP

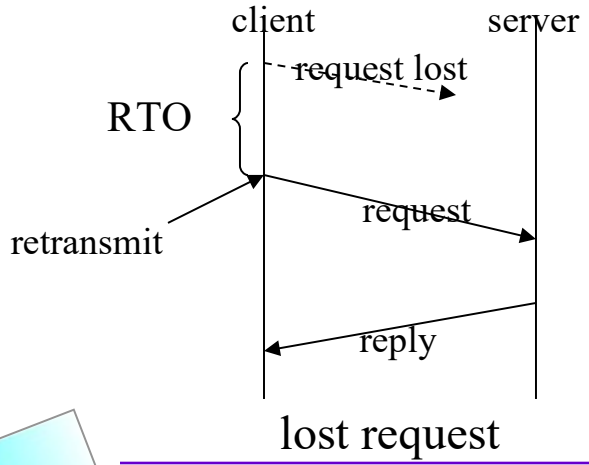
- **Advantages of UDP**
  - support broadcasting & multicasting
  - has no connection setup or teardown
    - † UDP : RTT+SPT (T/TCP □ UDP □ □□)
    - † TCP : 2 X RTT + SPT
- **only TCP Feature**
  - Positive acknowledgments, retransmission of lost packets, duplication detection, and sequencing of packets reordered by the network
  - Windowed flow control
  - Slow start and congestion avoidance

When to Use UDP

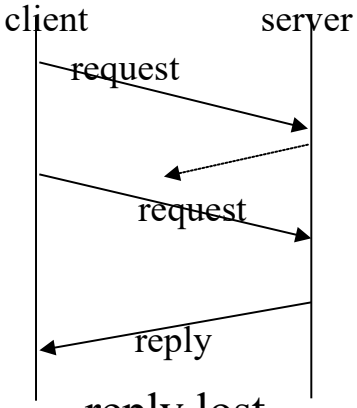
- **Recommendations**
  - UDP must be used for **broadcast** or **multicast** application
  - UDP can be used for ***simple request-reply applications*** but error detection must be built into the application
  - UDP should ***not*** be used for bulk data transfer

Adding Reliability to a UDP Application

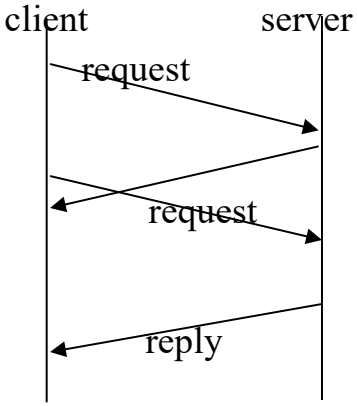
- **2 features to our client**
  - ***Timeout*** / ***Retransmission*** to handle datagrams
    - †
  - ***Sequence numbers*** to match reply with request so that the client can verify reply
    - † □□□) DNS, SNMP, TFPT, RPC □
- **Retransmission ambiguity Problem**



lost request



reply lost



RTO too short

Outline of RTT functions

Adding Reliability to a UDP Application

```
static sigjmp_buf jmpbuf;
{
   ......
   form request
   signal(SIGALRM, sig_alrm);
   rtt_newpack();
sendagain:
   sendto();
   alarm(rtt_start());
   if (sigsetjmp(jmpbuf, 1) !=0) {
        if (rtt_timeout() ) give up
        goto sendagain;   }
   do {
      recvfrom();
   } while(wrong sequence#);
  alarm(0);
  rtt_stop();
  process reply();
   ......
}
void sig_alarm(int signo) {
   siglongjmp(jmpbuf, 1);
}
```

establish signal handler

initialize rexmt counter to 0

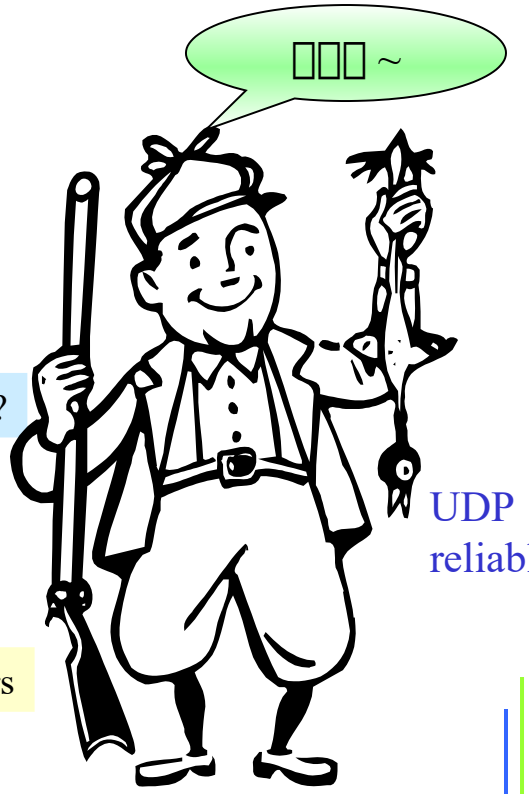
set alarm for RTO seconds

double RTO, retransmitted enough?

Retransmit

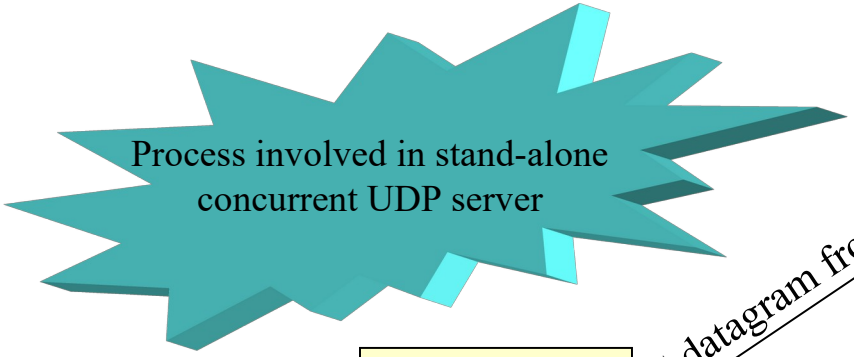
turn off alarm

calculate RTT and update estimators



UDP reliable

Concurrent UDP Server(1)



server
(parent)

port 69

creates socket, *binds*
recvfrom, blocks until
client reqeust, *fork*,
another *recvfrom*

1st datagram from client

client

fork

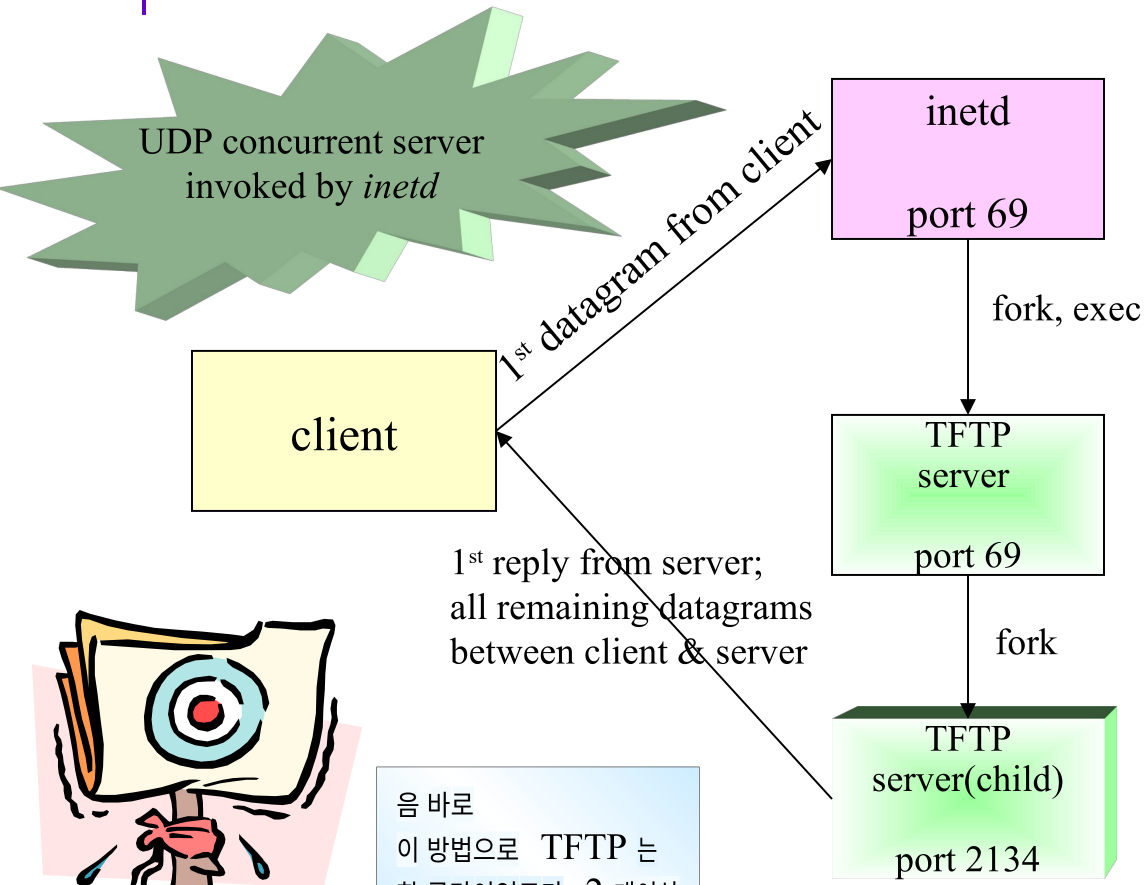
1st reply from server;
all remaining datagrams
between client & server

server
(child)

port 2134

create new socket, *bind*
ephemeral port(2134),
process clients request,
exchange additional
datagrams with client

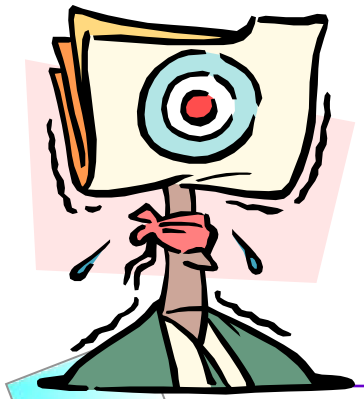
Concurrent UDP server(2)



create socket, *bind*s well-known port
when client request arrives, fork a child
and turn off selectability on UDP 69

recvfrom of client datagram,
fork a child, then *exit*

Create new socket, *bind* ephemeral
port, process client's request, exchange
additional datagrams with client on
new socket



음 바로
이 방법으로 TFTP 는
한 클라이언트가 2 개이상
물어보는 것을 해결하는군

Summary

- **IP_RECVDSTADDR / IP_RECVIF socket options can be enabled to return this information as ancillary data with each datagrams**
- **UDP**
  - broadcasting or multicasting
  - simple request-reply scenarios
  - Not used for bulk data transfer
- **Added reliability**
  - by detecting lost packets using a timeout and retransmission
  - RTT / timestamp

Race Conditions - Signal case (4)

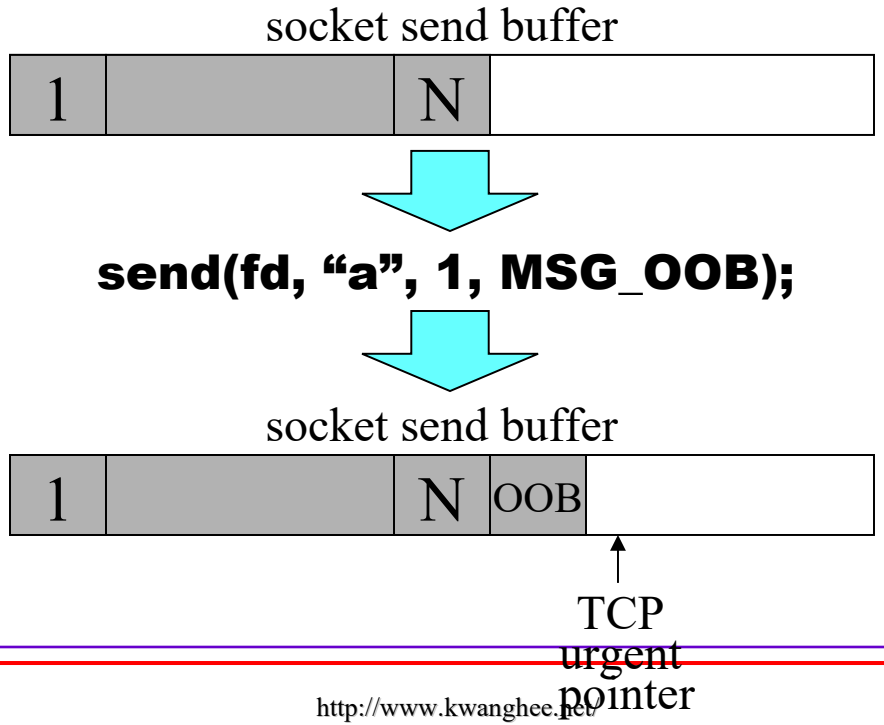
```
void dg_cli( … )
{
   Setsockopt( sockfd, SOL_SOCKET, SO_BROADCAST, &on,
sizeof(on) );

   Pipe( pipefd );

   Signal( SIGALRM, recvfrom_alarm );

   while ( … ) {
      Sendto( sockfd, … )

      alarm(5);
      for ( ; ; ) {
                  FD_SET( sockfd, &rset );
                  FD_SET( pipefd[0], &rset );
                  if ( ( n=select( … ) ) < 0 ) {
                     if ( errno == EINTR )
                        continue;
                     else
                        err_sys( "select error" );
                  }
                  if ( FD_ISSET( socfd, &rset ) ) {
                     n = Recvfrom( sockfd, .. );
                  }

…… Continue
```

```
Continue …..
                  if ( FD_ISSET( pipefd[0], &rset ) ) {
                     Read( pipefd[0], &n, 1 );  /* timer expired */
                     break;
                  }
      }
   }
}

static vois recvfsrom_alarm( int signo )
{
   Write( pipefd[1], "", 1 );  return;
}
```

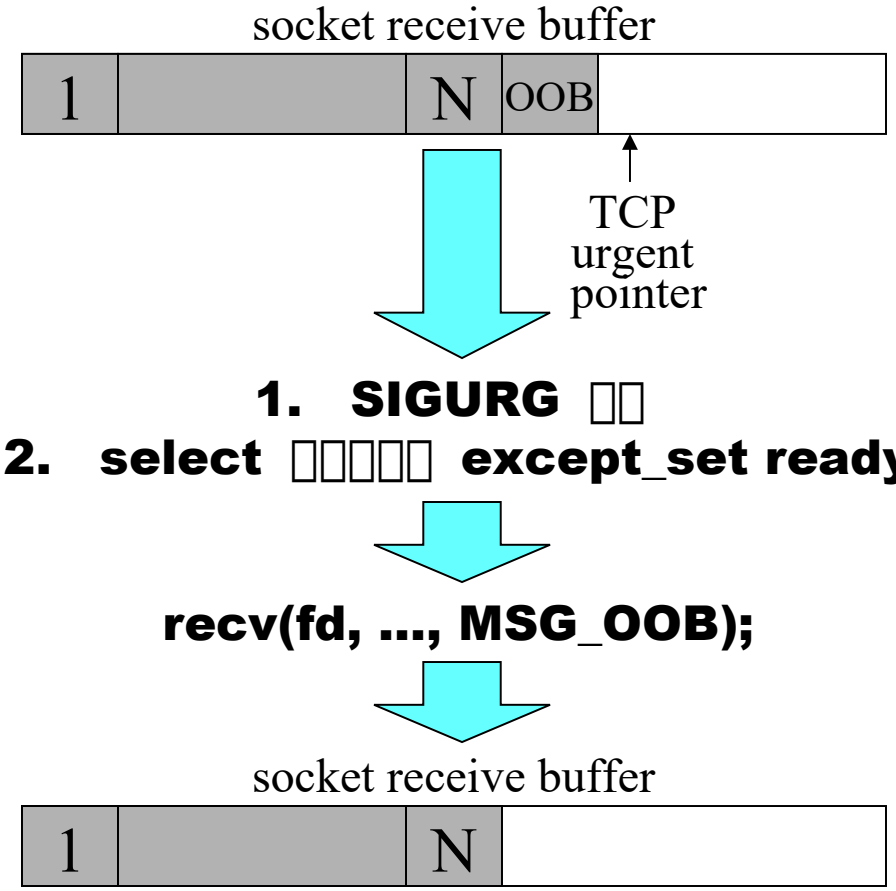
Chap. 21 *Out-of-Band Data*

Out-of-Band Data □... ? (1)

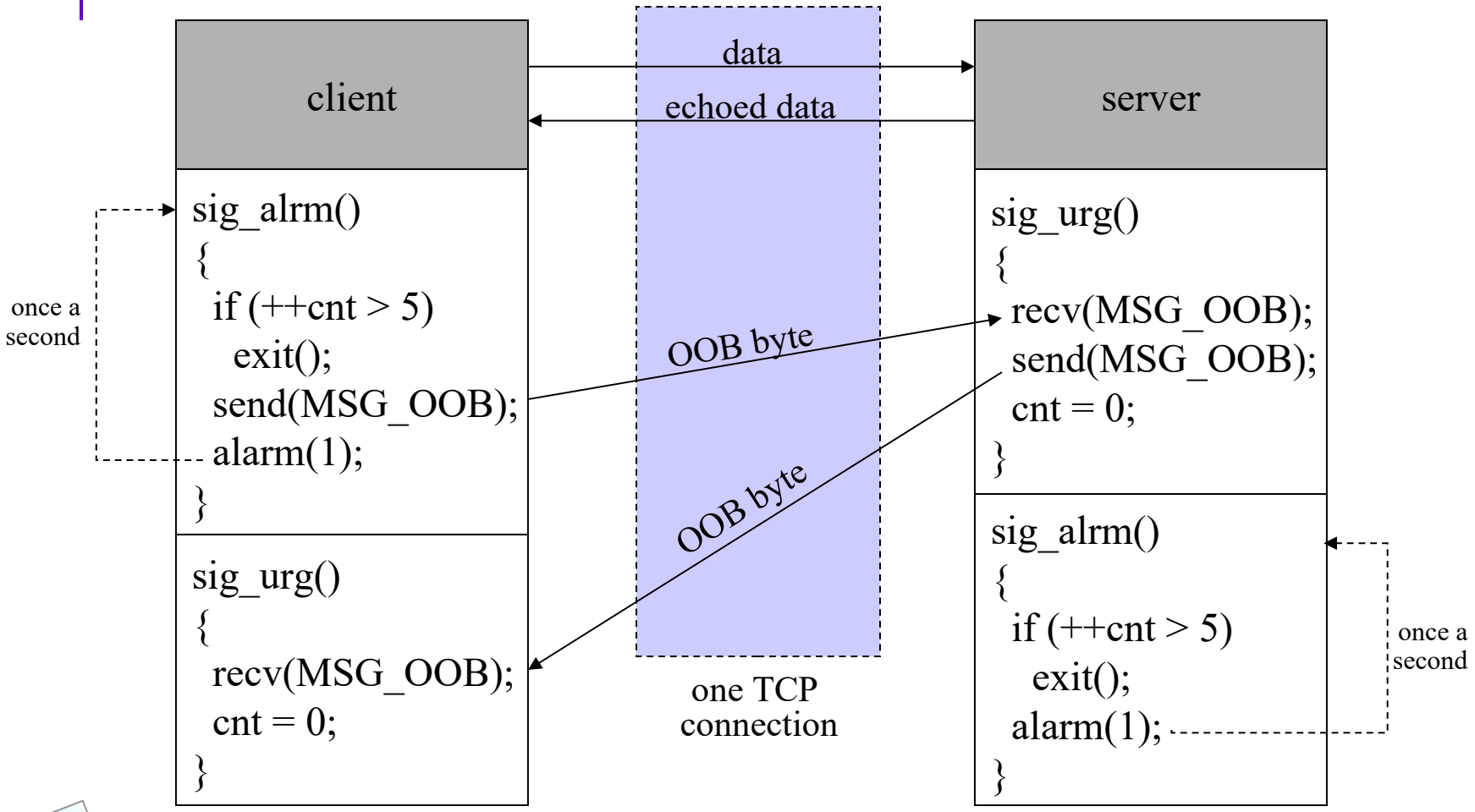
- **Out-of-band data □ normal(“inband”) data □□ □□ priority □ □□**
- **TCP □ urgent mode □ □□...**



Out-of-Band Data □... ? (2)



Example: Client-Server Heartbeat Functions



Chap. 22 *Signal-Driven I/O*

Signal-Driven I/O

- **3 Steps to use signal-driven I/O with a socket (SIGIO)**
  1. A signal handler must be established for the SIGIO signal
     ※ signal(SIGIO, sigio_handler);
  2. The socket owner must be set
     ※ fcntl(fd, F_SETOWN, getpid());
  3. Signal-driven I/O must be enabled for the socket
     ※ fcntl(fd, O_ASYNC, on);

When to drive SIGIO

- **SIGIO with UDP Sockets**
  - a datagram arrives for the socket
  - an asynchronous error occurs on the socket
- **SIGIO with TCP Sockets**
  - a connection request has completed on a listening socket
  - a disconnect request has been initiated
  - a disconnect request has completed
  - half of a connection has been shut down
  - data has arrived on a socket
  - data has been sent from a socket (i.e., the output buffer has free space)
  - an asynchronous error occurred

Chap. 23 *Threads*

What is Threads?

- **Comparison to fork…**
  - fork is expensive (e.g., memory is copied, all descriptors are duplicated, and so on…)
  - IPC is required to pass info. between the parent and child
- **Threads**
  - all threads within a process share…
    - † process instructions
    - † most data
    - † open files (e.g., descriptors)
    - † signal handlers and signal dispositions
    - † current working dir
    - † user ID and group ID
  - each thread has its own…
    - † thread ID
    - † set of registers, including PC and SP
    - † stack
    - † errno
    - † signal mask
    - † priority

Basic Thread Functions

- **int** pthread_create**(pthread_t *tid, const pthread_attr_t *attr, void * (*func) (void *), void *arg);**
- **int** pthread_join**(pthread_t tid, void **status);**
- **pthread_t** pthread_self**(void);**
- **int** pthread_detach**(pthread_t tid);**
- **void** pthread_exit**(void *status)**

TCP Echo Server Using Threads

```
#include    "unpthread.h"

static void *doit(void *);      /* each thread executes
        this function */

int
main(int argc, char **argv)
{
    int             listenfd, *iptr;
    socklen_t       addrlen, len;
    struct sockaddr *cliaddr;

    if (argc == 2)
        listenfd = Tcp_listen(NULL, argv[1], &addrlen);
    else if (argc == 3)
        listenfd = Tcp_listen(argv[1], argv[2], &addrlen);
    else
        err_quit("usage: tcpserv01 [ <host> ] <service
        or port>");

    cliaddr = Malloc(addrlen);

    for ( ; ; ) {
        len = addrlen;
        iptr = Malloc(sizeof(int));
       *iptr = Accept(listenfd, cliaddr, &len);

        Pthread_create(NULL, NULL, &doit, iptr);
    }
}
```

```
static void *
doit(void *arg)
{
    int     connfd;

    connfd = *((int *) arg);
    free(arg);

    Pthread_detach(pthread_self());
    str_echo(connfd);       /* same function as before */
    Close(connfd);          /* we are done with connected socket */
    return(NULL);
}
```

More Else about Threads

- **Thread-Specific Data**
  - int pthread_once(pthread_once_t *onceptr, void (*init)(void));
  - int pthread_key_create(pthread_key_t *keyptr, void (*destructor)(void *value)
  - void *pthread_getspecific(pthread_key_t key);
  - int pthread_setspecific(pthread_key_t key, const void *value);
- **Mutexes: Mutual Exclusion**
  - int pthread_mutex_lock(pthread_mutex_t *mptr);
  - int pthread_mutex_unlock(pthread_mutex_t *mptr);
- **Condition Variables**
  - int pthread_cond_wait(pthread_cond_t *cptr, pthread_mutex_t *mptr);
  - int pthread_cond_signal(pthread_cond_t *cptr);

Chap.24 - IP Options

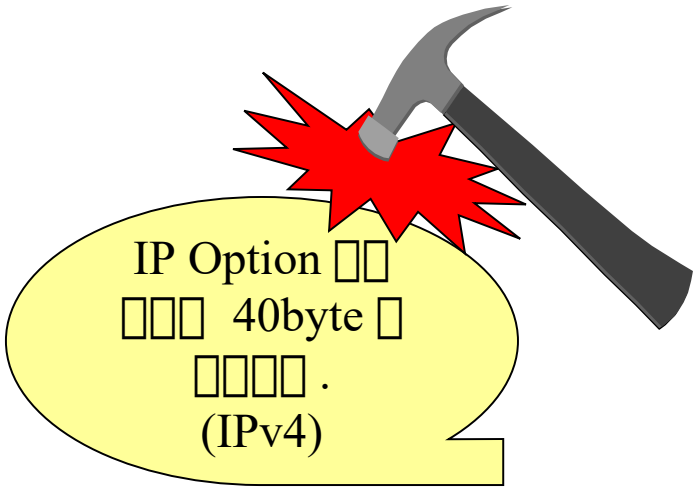
- IP Options □ □□
- IPv4 Source Route Options
- IPv6 Extension Headers
- IPv6 Routing Header

We must study … ?

- IP option 의 종류
- program 예를 가지고 사용법을 알아본다 .

IP Option □ □□

1. NOP : no-operation, padding 1byte
2. EOL : end-of-list, indicate option end
3. LSRR : loose source and record route
4. SSRR : strict source and record route
5. Time stamp
6. Record route
7. Basic security
8. Extended security
9. Stream identifier
10. Router alert



getsockopt □ setsockopt □□□□

level = IPPROTO_IP, optname = IP_OPTIONS □

□□ □□ □□ □□ □□□□ .

IPv4 source route option

☞ 송신단에 의해 명시되는 IP 주소록들의 목록

☞ 수신단이 새 목록을 받을 수 있게 하고, 송신단으로 되돌아 가는 반대 경로를 따라서 거꾸로 갈 수 있게 한다.

↳ SSRR

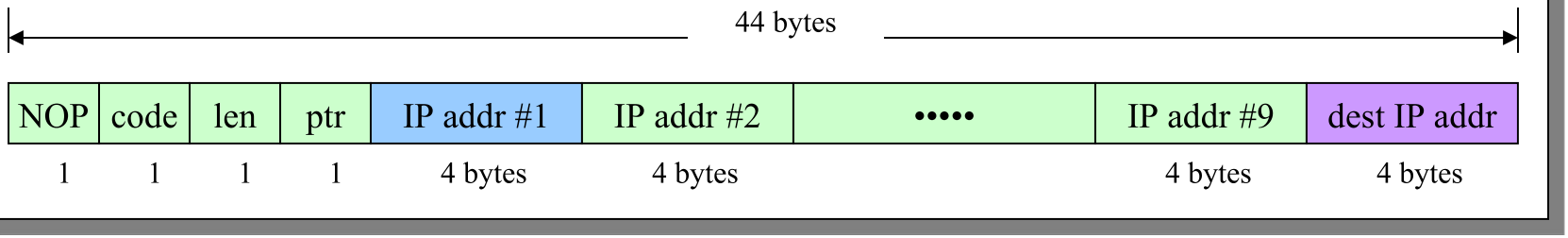
datagram 은 반드시 기록된 각각의 node 들을 통과해야 하고, 오직 기록된 node 만을 지나갈 수 있다.

↳ LSRR

datagram 은 기록된 각각의 node 들을 통과해야 하지만 기록되지 않은 다른 node 들을 통과할 수도 있다.

※ Kernel 에 의해서 수신된 source route 순서의 반대로 IP address 가 정렬되어 application 으로 넘어온다.

Send : to kernel by setsockopt

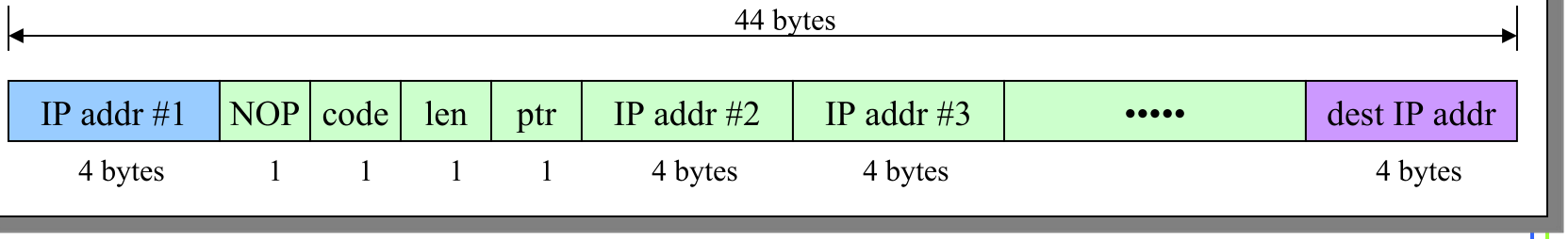


code : LSRR - 0x83, SSRR - 0x89

len : code ~ dest IP addr 까지의 byte 수

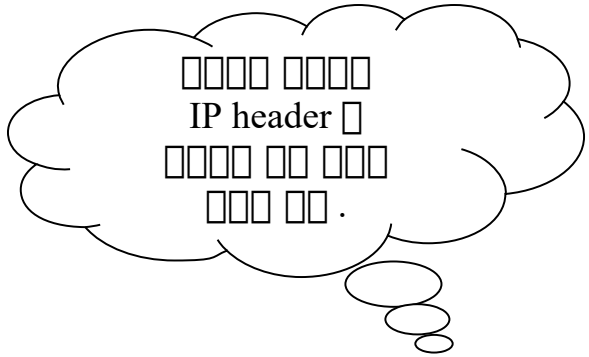
ptr : 경로에서 수행될 다음 번 IP 주소의 offset

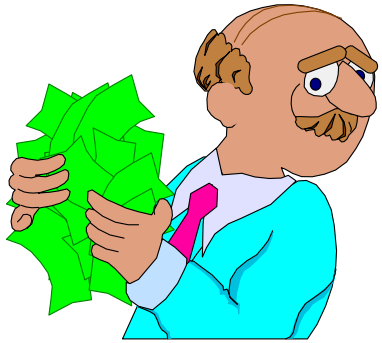
Receive : to application by getsockopt



Example

```
> tcpcli01 -g sejong -g cupid feel
hi
hi

# tcpserv01
received IP options, len = 16
received LSRR: 150.150.55.52  150.150.55.53  150.150.55.55

> tcpcli01 -g cupid -g danwon -g sejong feel
hi
hi

# tcpserv01
received IP options, len = 20
received LSRR: 150.150.55.52  150.150.55.55  150.150.55.54  150.150.55.53
child 21180 terminated
```

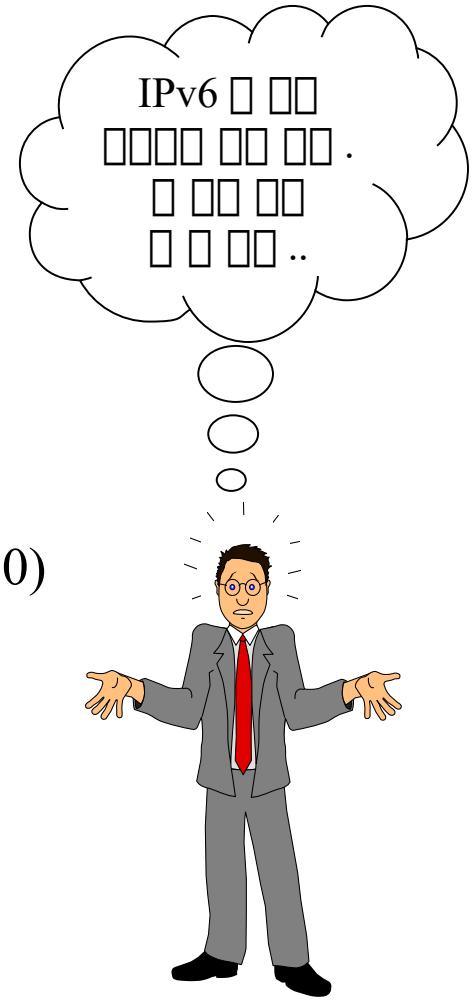




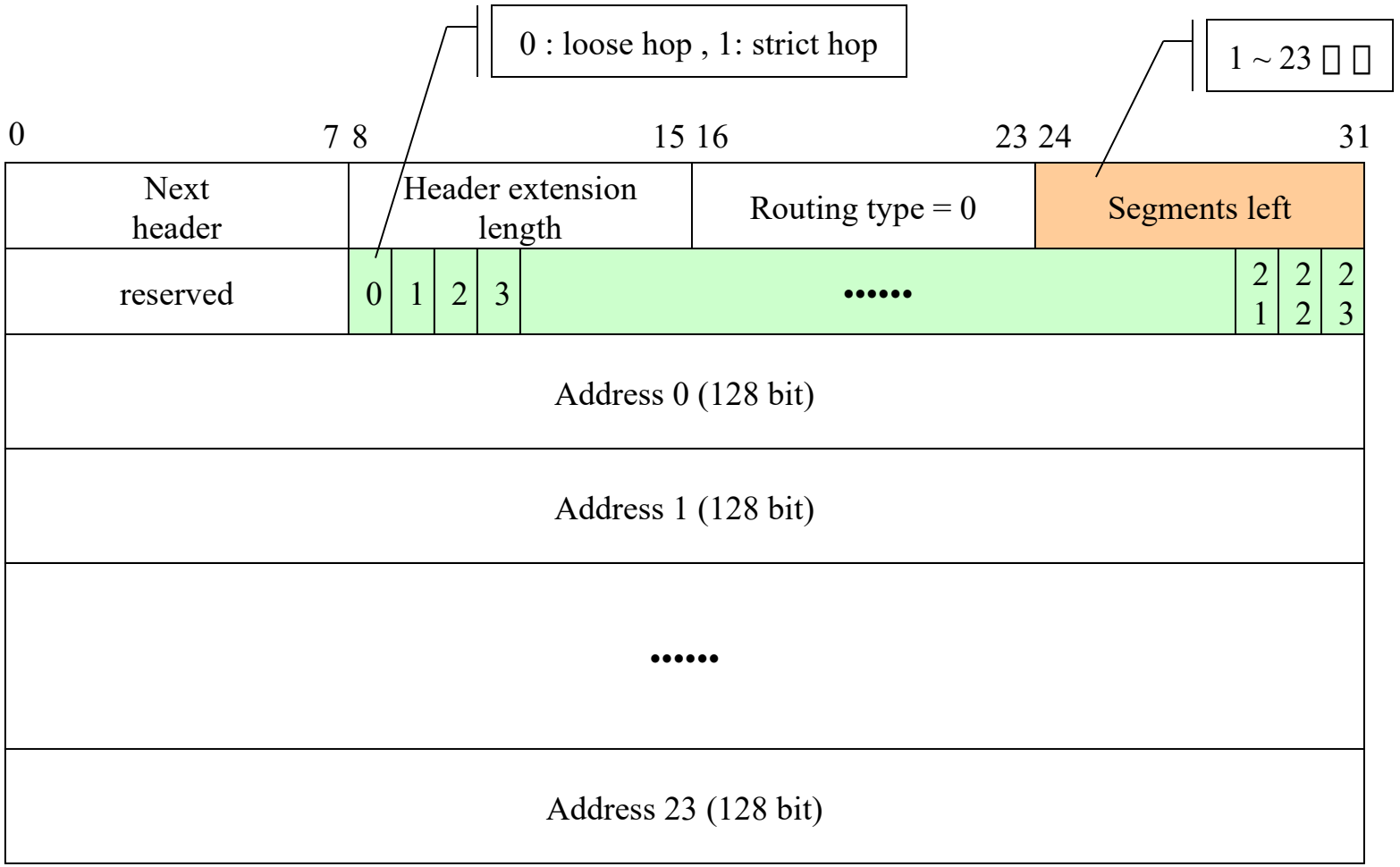
IPv6 extension headers

1. Hop-by-hop option(0) : currently non-defined
2. Destination option(60) : currently non-defined
3. Routing header(43) : source routing option
4. Fragmentation header(44)
5. Authentication header(51)
6. ESP(Encapsulating security payload) header(50)

※ ( ) : IP next header field value. if value is 59, no next header



IPv6 routing header



Contents

Chap.25 - Raw Socket

- Raw socket's features
- Raw socket creation, output, input
- Ping program
- Traceroute program
- ICMP message daemon

We must study … ?

- Raw socket □ □□
- Raw socket □ □□□ programming

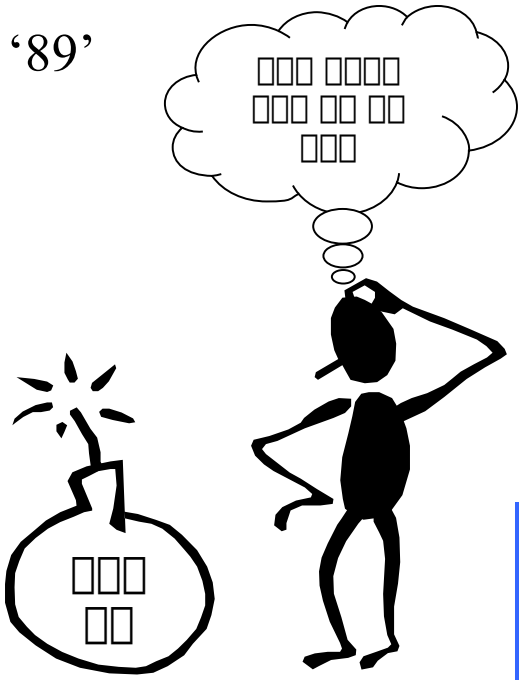
Raw socket's features

1. ICMPv4, IGMPv4, ICMPv6 packet 을 읽고 쓸 수 있게 한다 .

   Ex) ping program

2. Kernel 에 의해 처리되지 않는 IPv4 protocol field 를 가진 IPv4 datagram 을 읽고 쓸 수 있다 .

   Ex) OSPF routing protocol - protocol field '89'
   cf) ICMP-1, IGMP-4, TCP-6, UDP-17



Raw socket creation, output, input

Creation

오직 관리자만이 raw socket 을 생성할 수 있다 .

sockfd = socket(AF_INET, SOCK_RAW, protocol)

- **IP_HDRINCL socket option** 설정

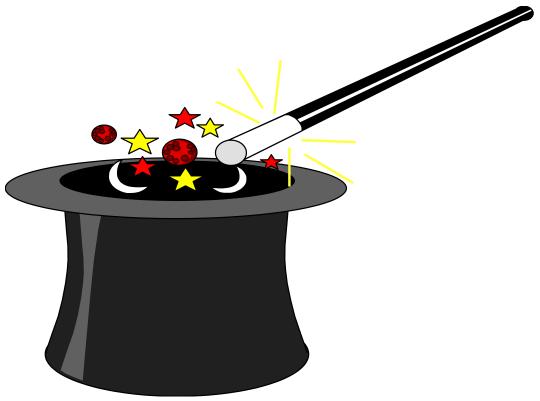
```
const int on = 1;
if ( setsockopt( sockfd, IPPROTO_IP, IP_HDRINCL, &on, sizeof(on) ) < 0 )
        error
```

- **Bind**

local address 를 설정한다 .

- **Connect**

외부 주소를 설정한다 .



Output

- 정상적인 출력은 sendto 나 sendmsg 중 하나를 호출하고 destination IP 주소를 지정하여 이루어진다.

  Ex] sendto(sockfd, sendbuf, len, 0, ***pr->sasend***, pr->salen);

- IP_HDRINCL 가 설정돼 있지 않으면 kernel 이 IP header 를 만들고 process 로부터 data 에 주소를 덧 붙이기 때문에 data 의 시작 주소는 IP header 뒤의 처음 byte 를 명시한다.

- IP_HDRINC 가 설정 돼 있으면 쓰려고 하는 kernel 을 위한 시작 주소는 IP header 의 처음 byte 를 명시한다.

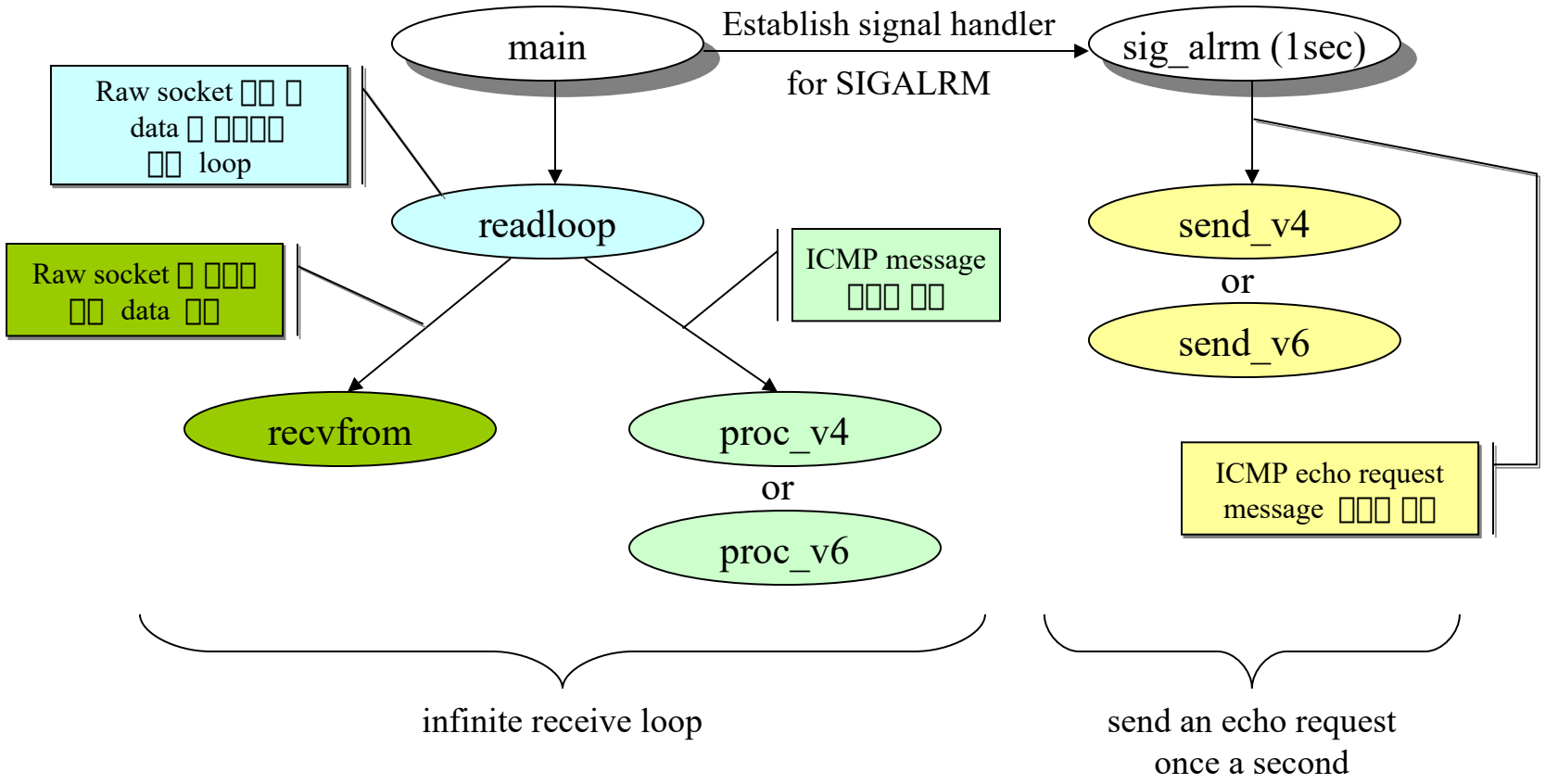
- kernel 은 출력 interface MTU 를 초과하는 순수 packet 을 작은 조각으로 만든다.

input

- 수신된 UDP packets 과 TCP packets 은 절대로 raw socket 으로 전달되지 않는다 . Process 가 원한다면 data link 계층에서 읽혀져야 한다 .
- kernel 이 ICMP message 를 처리한 후에 대부분의 ***ICMP packet*** 들은 raw socket 으로 전달된다 . Echo, timestamp, 주소 선별 요청은 전적으로 kernel 에 의해서 처리된다 .
- kernel 이 IGMP message 를 처리한 후에 모든 ***IGMP packet*** 들은 raw socket 으로 전달된다 .
- kernel 이 이해하지 못하는 protocol field 를 가진 모든 IP datagram 은 raw socket 으로 전달된다 .
- datagram 이 fragment 되어서 도착되면 모든 조각들이 도착해서 재조합될 때까지 아무것도 raw socket 으로 전달되지 않는다 .

Ping program

Function diagram



Traceroute program

☞ Host 에서 destination host 로의 datagram flow 경로를 trace 한다 .

① TTL(또는 hop count) 을 1 로 선정한 UDP datagram 을 destination 으로 보낸다 .

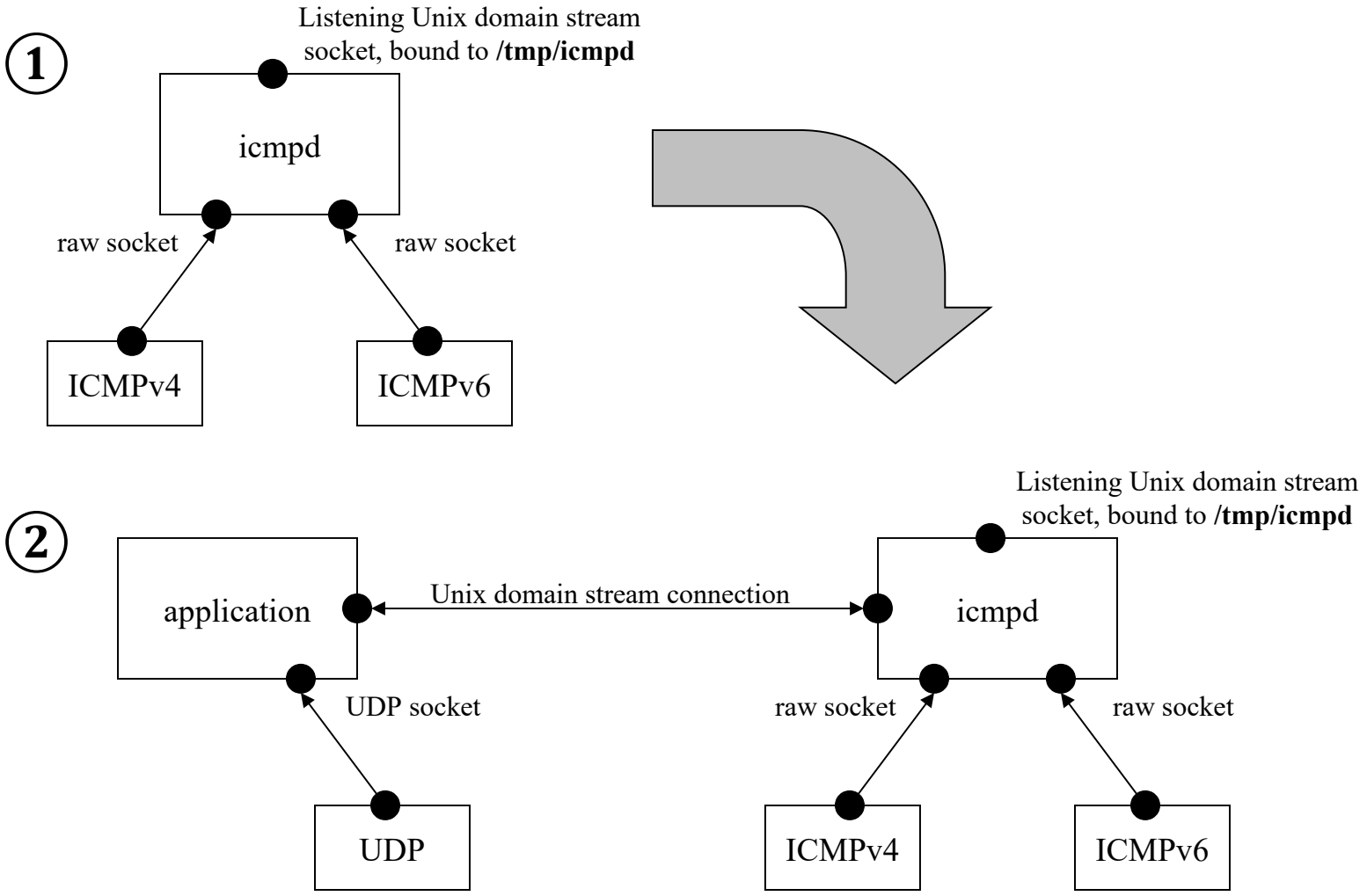
② First hop router 로부터 ICMP ‘time exceeded in transit’ 수신

③ TTL 을 하나씩 증가 시키면서 UDP datagram 을 destination 으로 보낸다 .

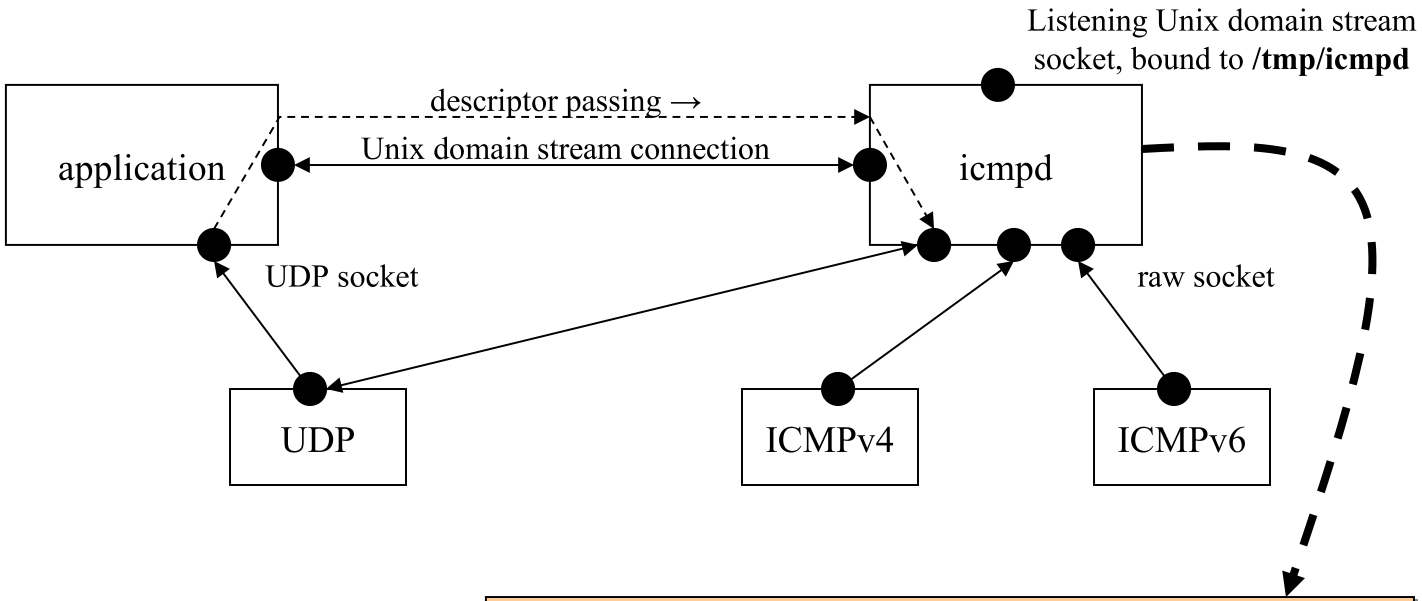
④ ③ 을 반복하면 계속 중간 hop host 로부터 ICMP ‘time exceeded in transit’ 가 수신된다 .

⑤ Destination host 에 도착을 하면 ICMP ‘port unreachable’ 이 수신된다 .

ICMP Message daemon

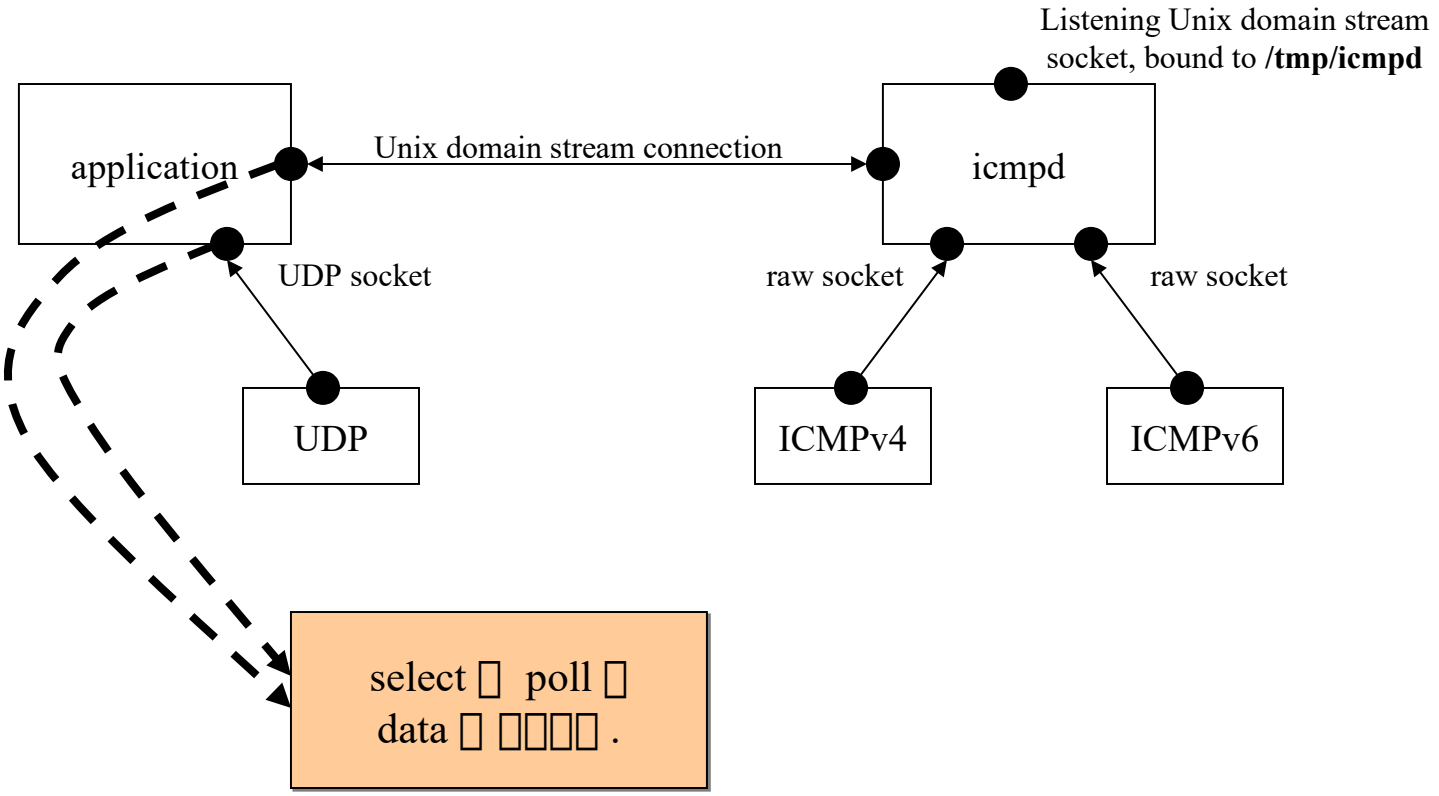


③



getsockname 을 호출하여 UDP socket 의
port number 를 알아낸다 .
그렇게 함으로써 icmp daemon 이 UDP socket 에
관련된 icmp message 를 처리한다 .

④

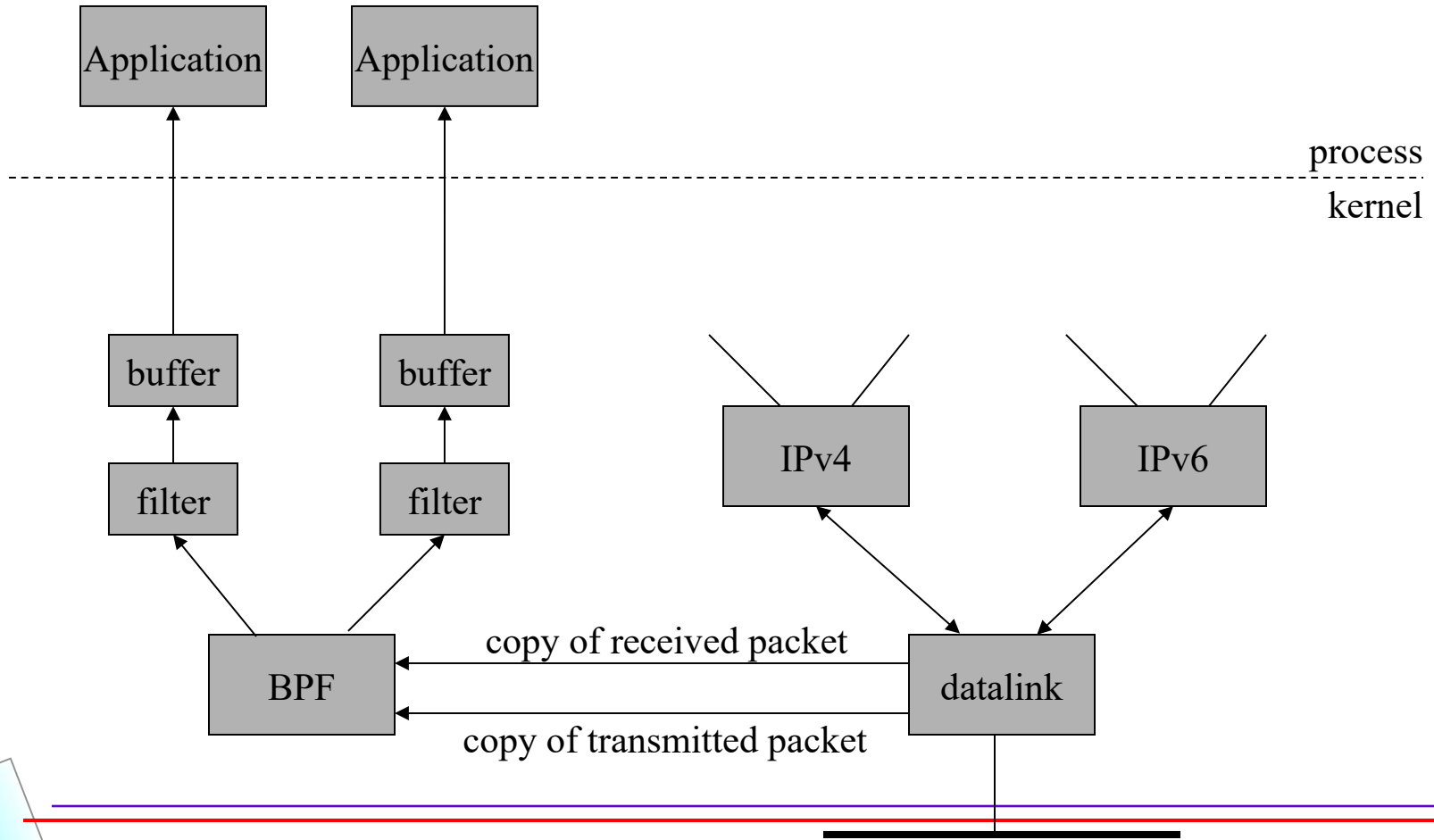

Listening Unix domain stream
socket, bound to /tmp/icmpd
application
Unix domain stream connection
icmpd
UDP socket
raw socket
raw socket
UDP
ICMPv4
ICMPv6
select □ poll □
data □ □□□□ .

Chap. 26 *Datalink Access*

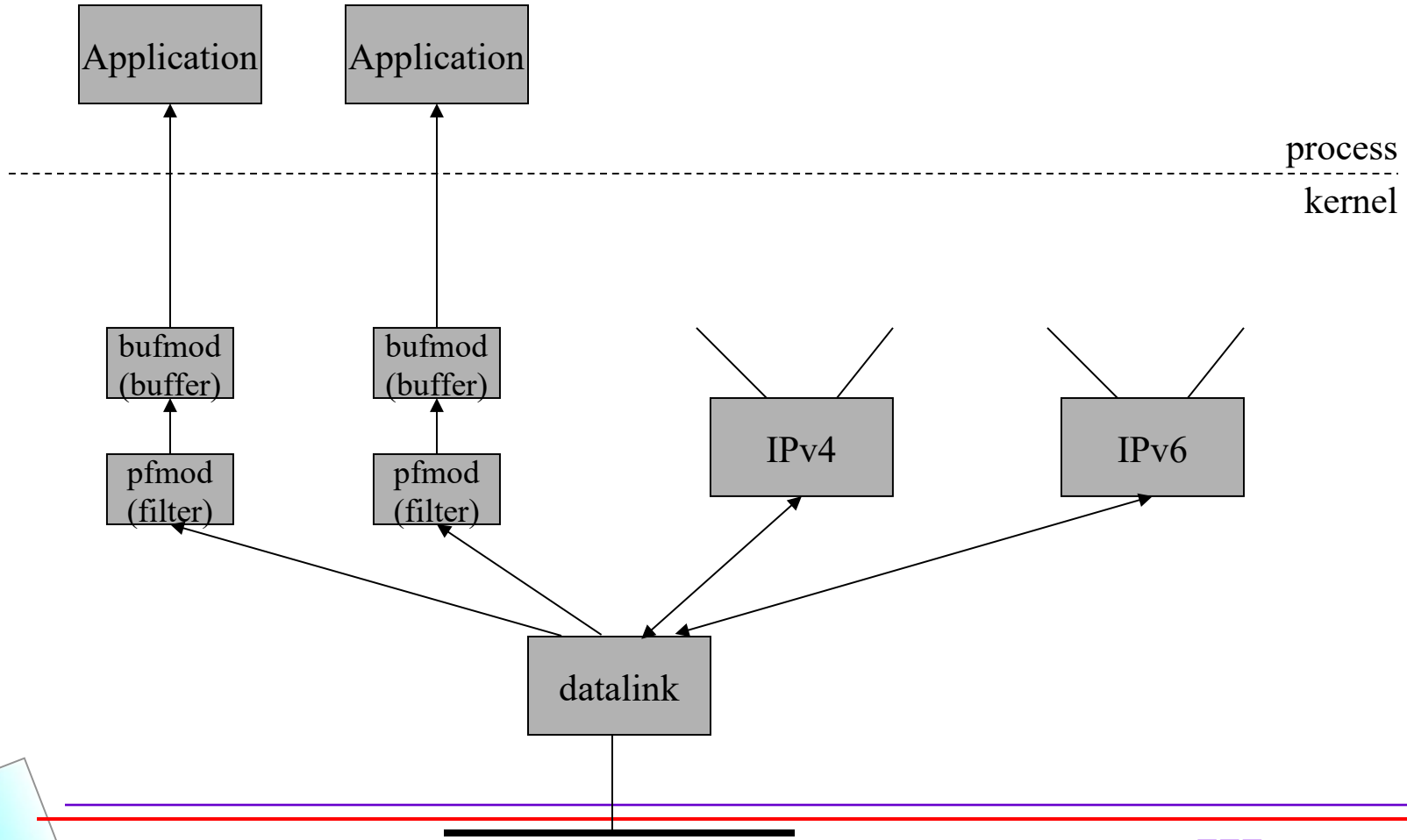
Introduction

- **Datalink access provides…**
  - The ability to watch the packets received by the datalink layer
  - The ability to run certain programs as normal applications instead of as part of the kernel(e.g., RARP server)
- **Common methods to access the datalink layer**
  - BSD Packet Filter(BPF)
  - SVR4 Data Link Provider Interface (DLPI)
  - Linux SOCK_PACKET interface
  - **libpcap, the publicly available packet capture library**

Packet Capture Using BPF

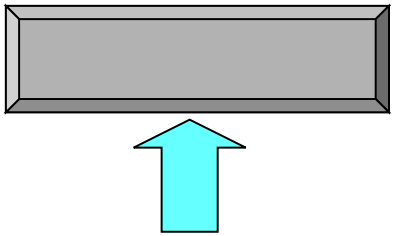


Packet Capture Using DLPI



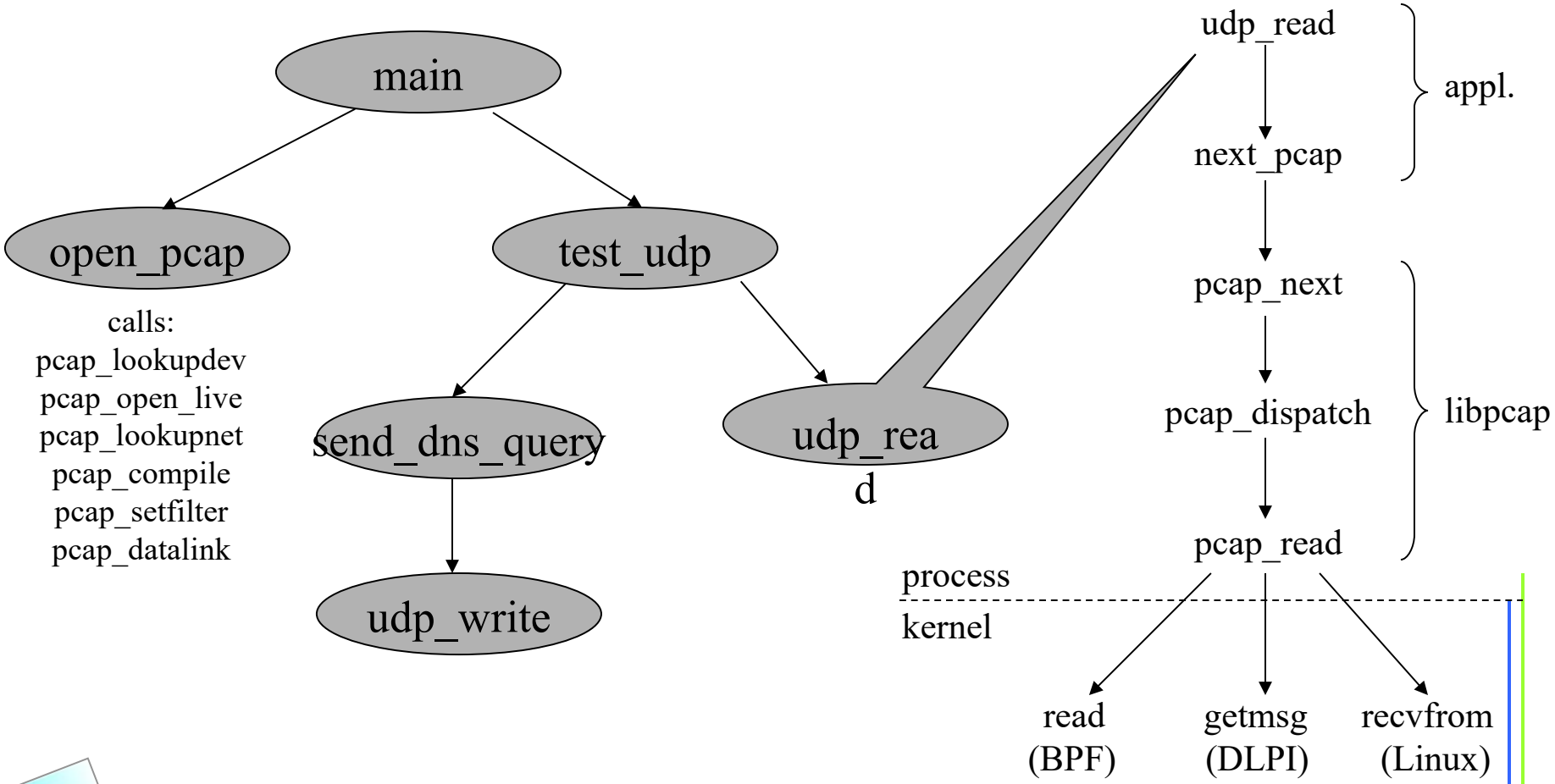
Linux: SOCK_PACKET

- **fd = socket(AF_INET,** SOCK_PACKET**, htons(ETH_P_ALL));**



ETH_P_ALL
ETH_P_IP
ETH_P_ARP
ETH_P_IPV6

libpcap Example: Examining the UDP Checksum Field



Chap. 27 *Client-server Design Alternatives*

Alternative TCP Server Models

- **Preforking server**
  - No Locking around accept (BSD only)
  - File Locking around accept
  - Thread Locking around accept
  - Descriptor Passing

- Prethreading server
  - per-Thread accept
  - Main Thread accept