네트워크 프로토콜 및
리눅스 네트워크 프로그래밍

참고문헌

W. R. Stevens et al., “Unix Network Programming, Volume 1: The Sockets Networking API,” 3rd Edition, Addison Wesley, 2003.

B. A. Forouzan, “TCP/IP Protocol Suite,” 4th Edition, 2009.

C. Benvenuti, “Understanding Linux Network Internals,” O’Reilly, 2005.

1. TCP/IP 프로토콜 이해

목 차

1.1 OSI 모델

1.2 TCP/IP 프로토콜 및 주소

1.3 인터넷 프로토콜 (Internet Protocol)

- IP 패킷 라우팅
- ARP / RARP 프로토콜

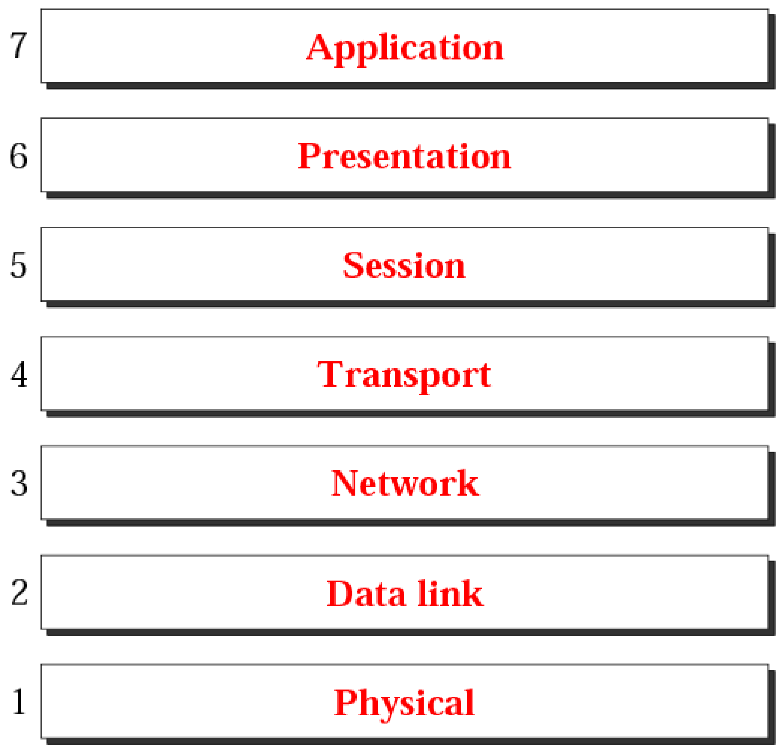
1.4 UDP 프로토콜 (User Datagram Protocol)

1.5 TCP 프로토콜 (Transmission Control Protocol)

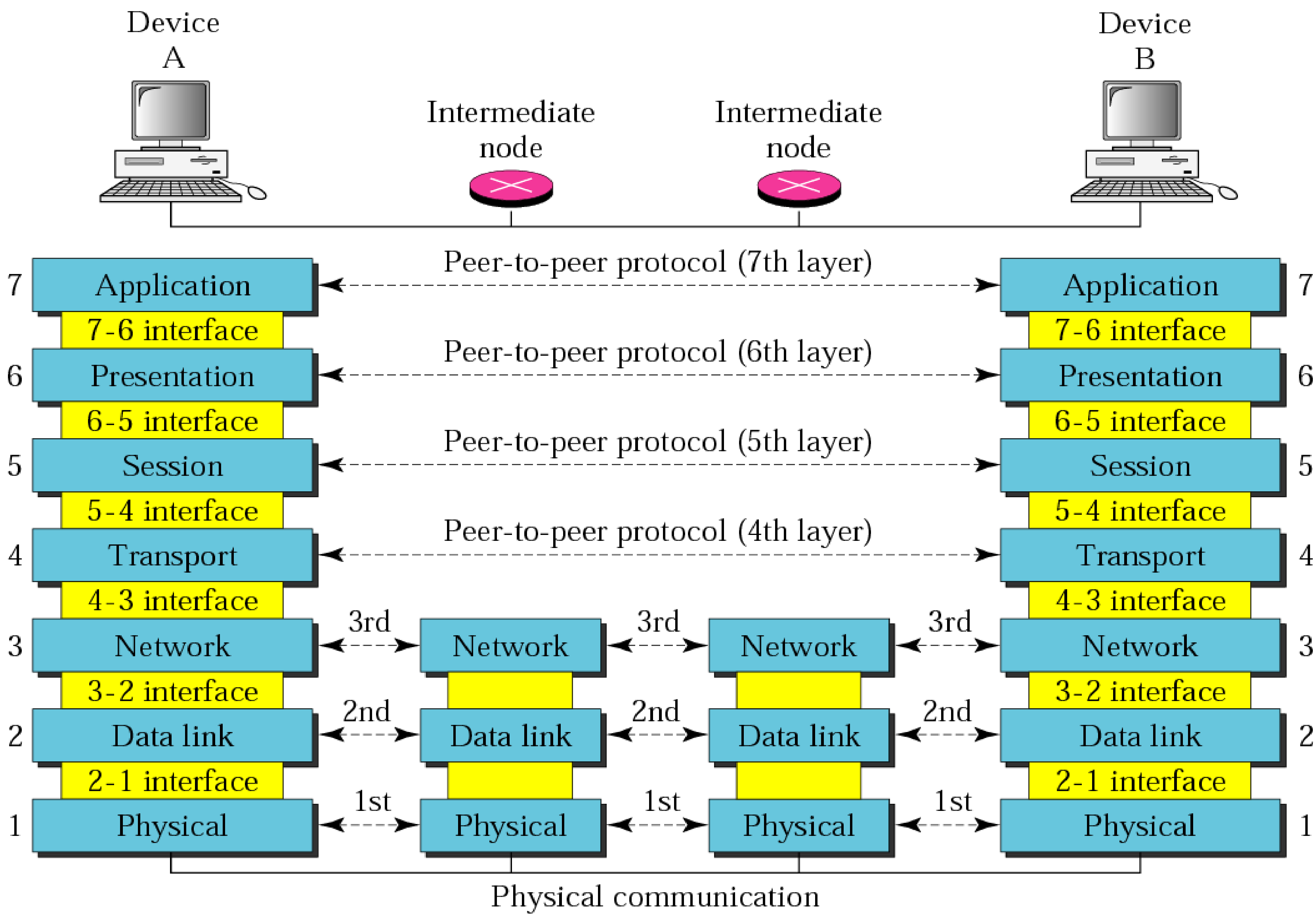
1.1 OSI 모델

International Standards Organization (ISO) 기관이 OSI 모델 제안

- 이종의 시스템들이 연결되어 있는 네트워크 환경에서 서로 원활히 통신하기 위해 고려되어야 할 사항들을 정리한 모델
- Open Systems Interconnection (OSI) 7 계층 모델

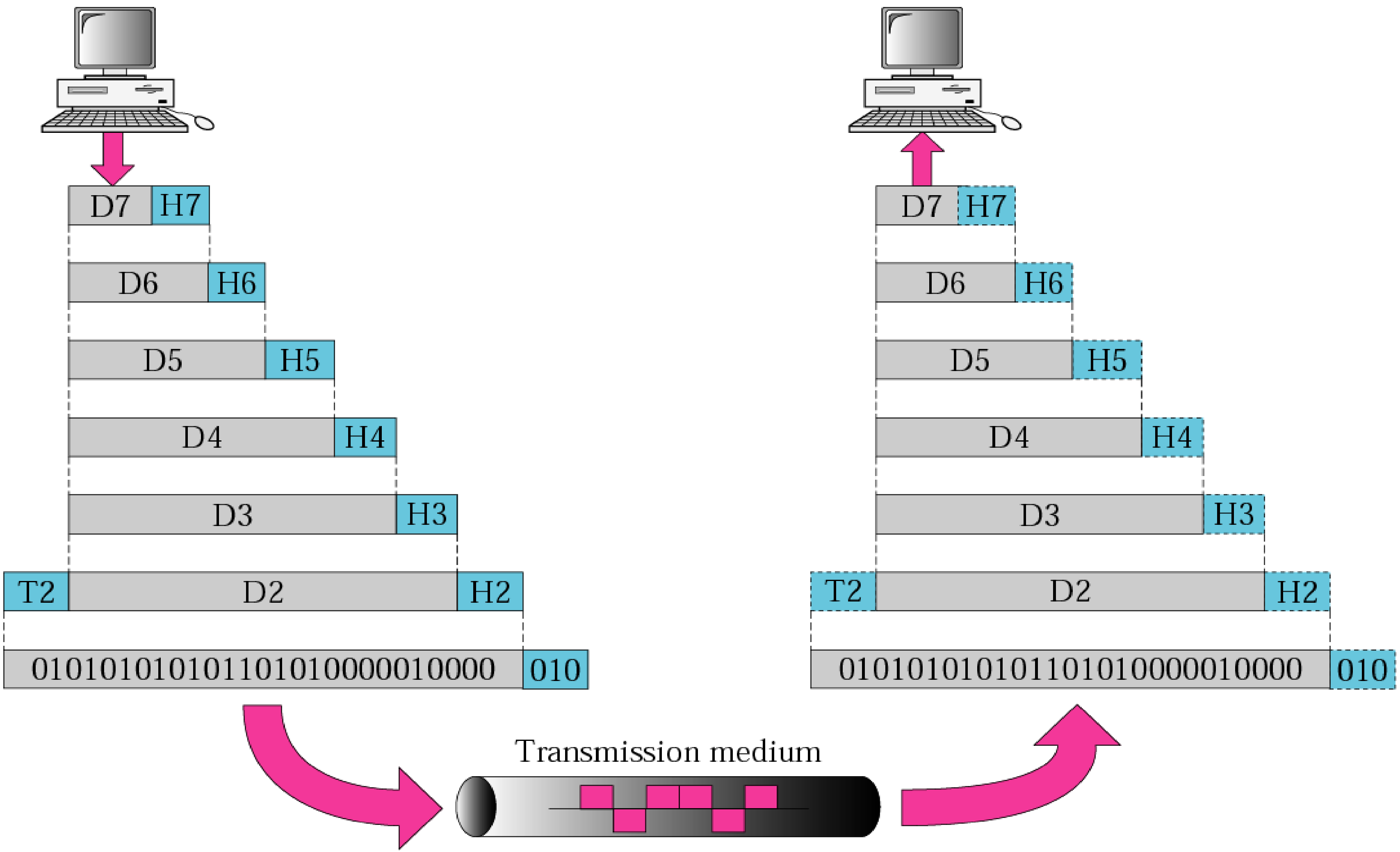


OSI 7 계층



Device A
Device B
Intermediate node
Intermediate node
Peer-to-peer protocol (7th layer)
Peer-to-peer protocol (6th layer)
Peer-to-peer protocol (5th layer)
Peer-to-peer protocol (4th layer)
7 Application
7-6 interface
6 Presentation
6-5 interface
5 Session
5-4 interface
4 Transport
4-3 interface
3 Network
3-2 interface
2 Data link
2-1 interface
1 Physical
3rd
2nd
1st
Network
Data link
Physical
Physical communication

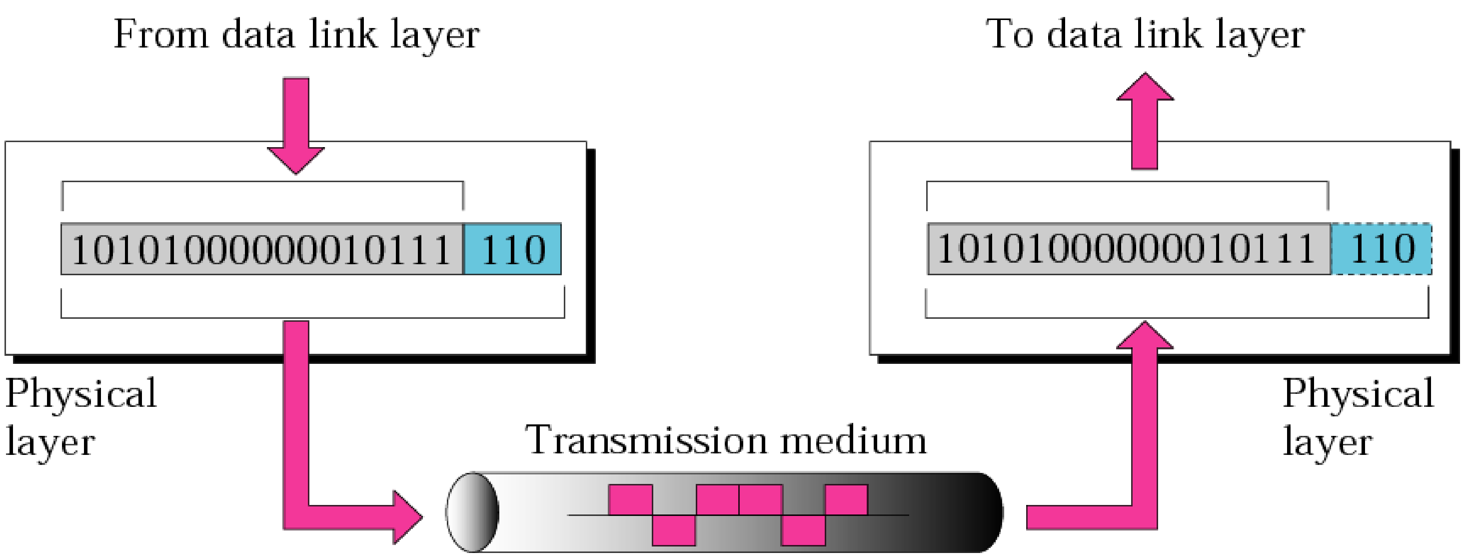
메시지 송수신을 위한 각 계층의 역할



물리 계층 (Physical Layer)

한 노드(또는 HOP)에서 다른 노드로 비트들의 전송 담당

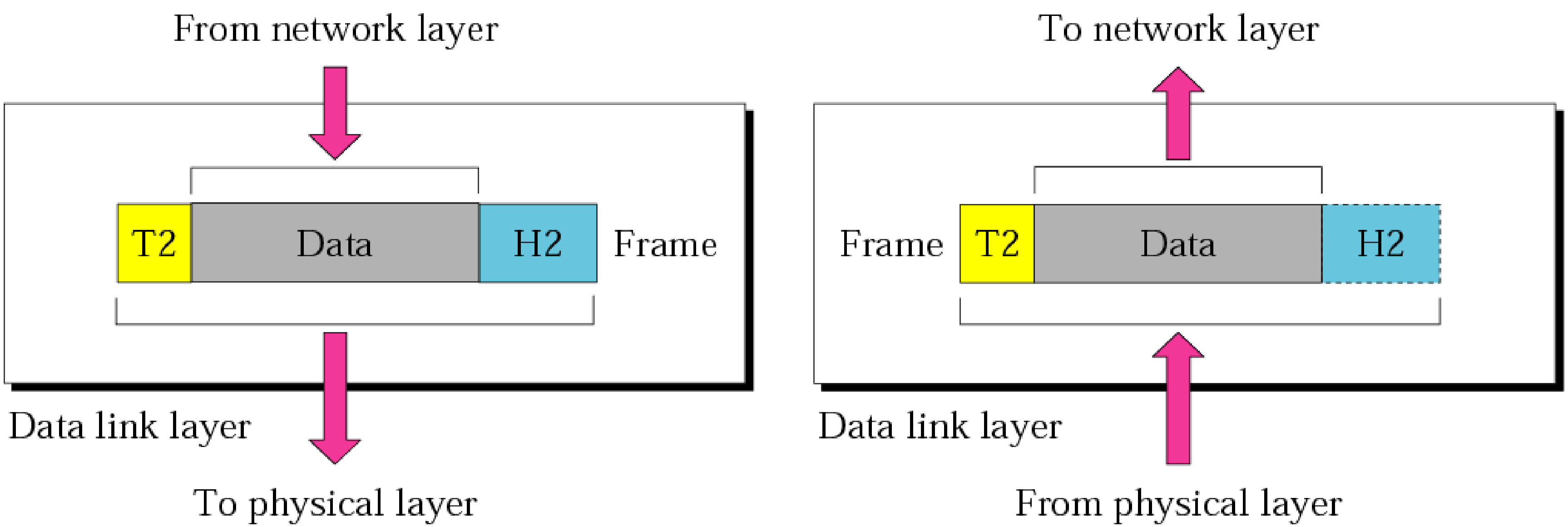
- 네트워크 장치와 통신 매체를 통해서 비트를 전송하기 위한 연결 설정과 해제
- 비트로 표현된 데이터와 전기적인 신호 간의 변환을 제어



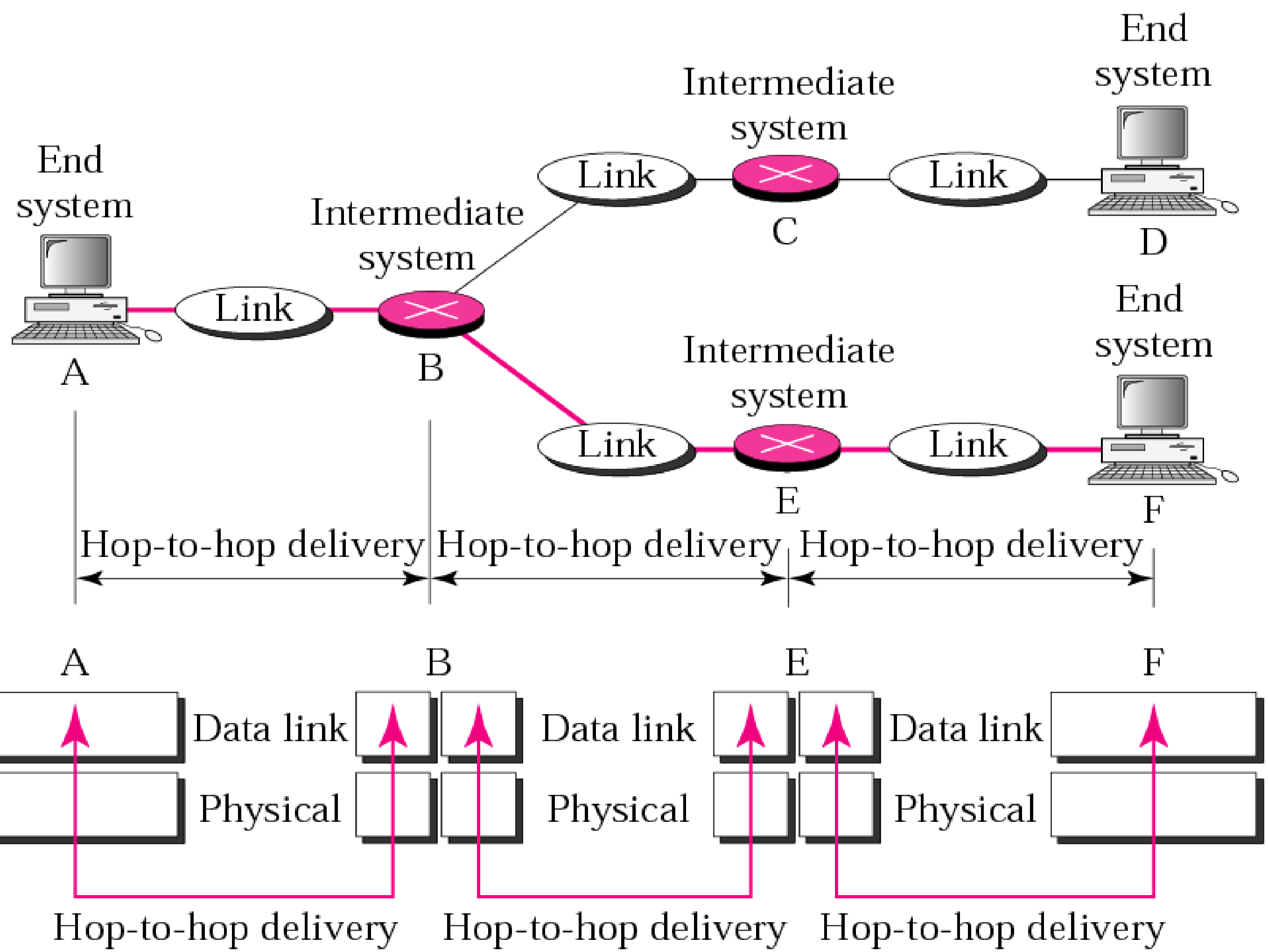
데이터 링크 계층 (Data Link Layer)

한 노드에서 다른 노드로 프레임(Frame) 전송 담당

- 노드간 일대일 통신을 가능하게 함
  - Hop-to-hop 전송
- 하위 물리 계층에서 발생 가능한 에러를 감지하고 복구



Hop-to-hop 전송

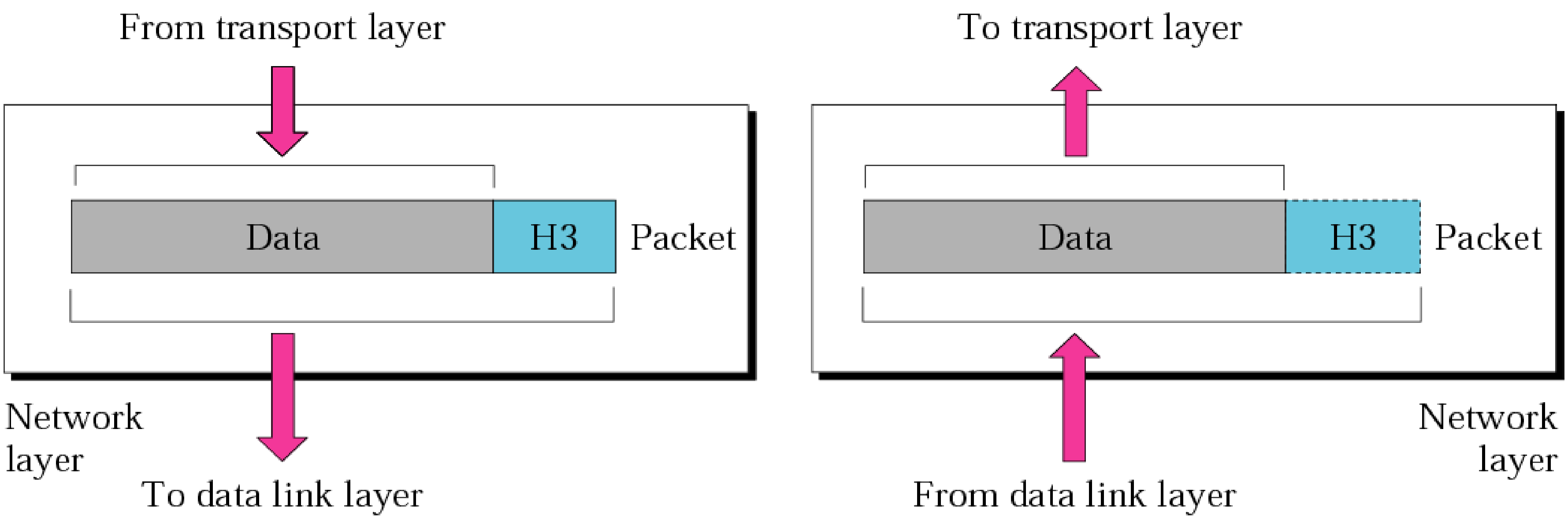


End system
A
Link
Intermediate system
B
Link
Intermediate system
C
Link
End system
D
Link
Intermediate system
E
Link
End system
F
Hop-to-hop delivery
Hop-to-hop delivery
Hop-to-hop delivery
A
B
E
F
Data link
Data link
Data link
Physical
Physical
Physical
Hop-to-hop delivery
Hop-to-hop delivery
Hop-to-hop delivery

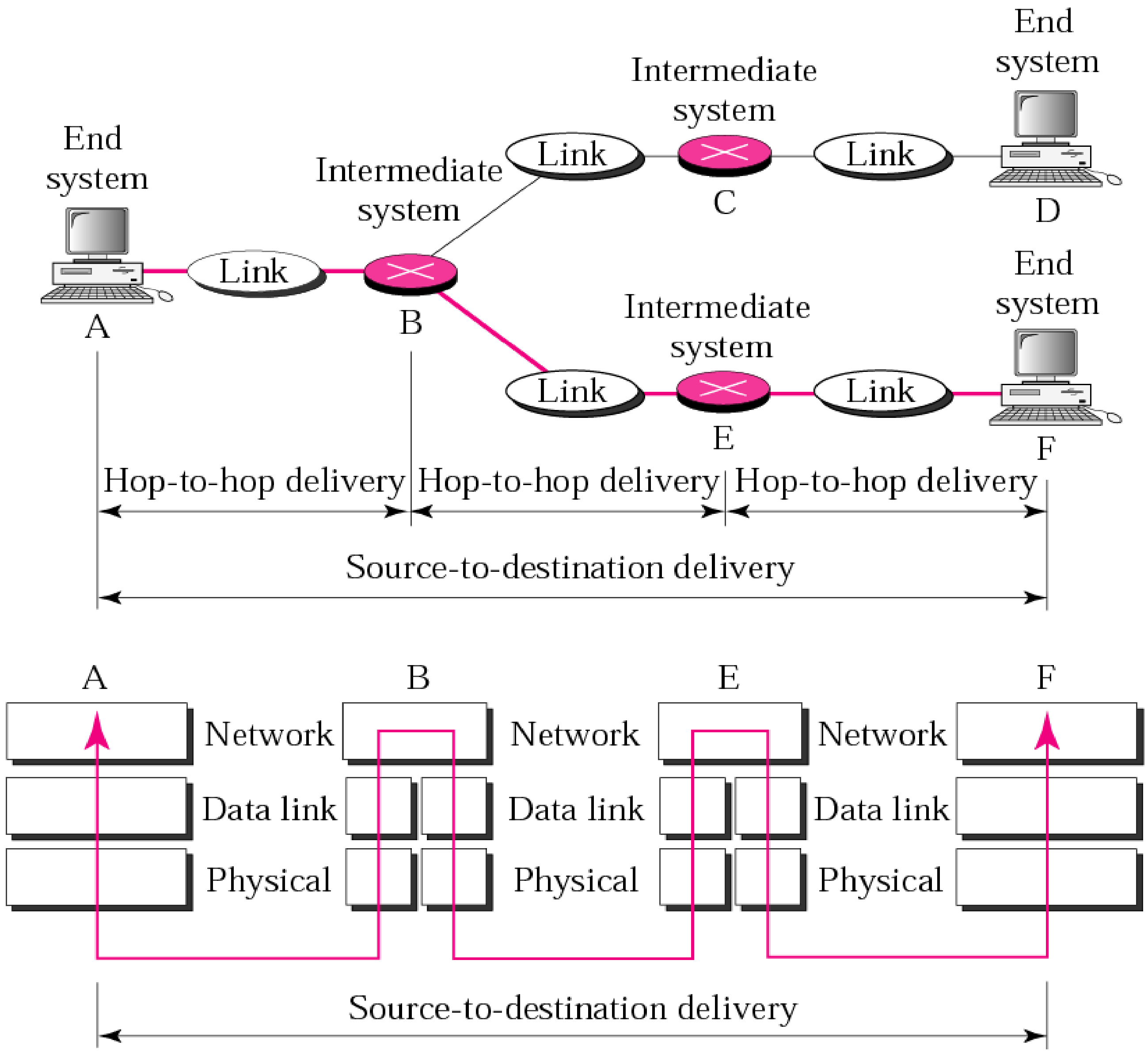
네트워크 계층 (Network Layer)

소스 호스트와 목적지 호스트간의 패킷 전송을 담당

- 멀리 떨어진 다른 네트워크에 존재하는 노드간 통신을 제공
  - Source-to-destination 전송

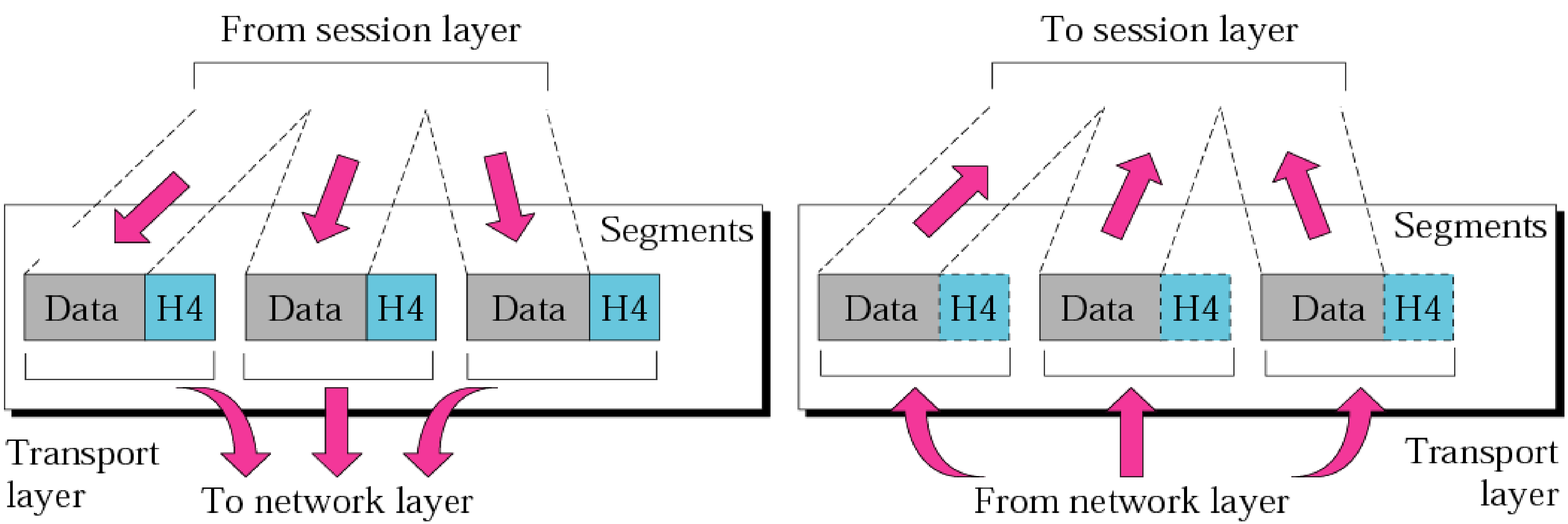


Source-to-destination 전송



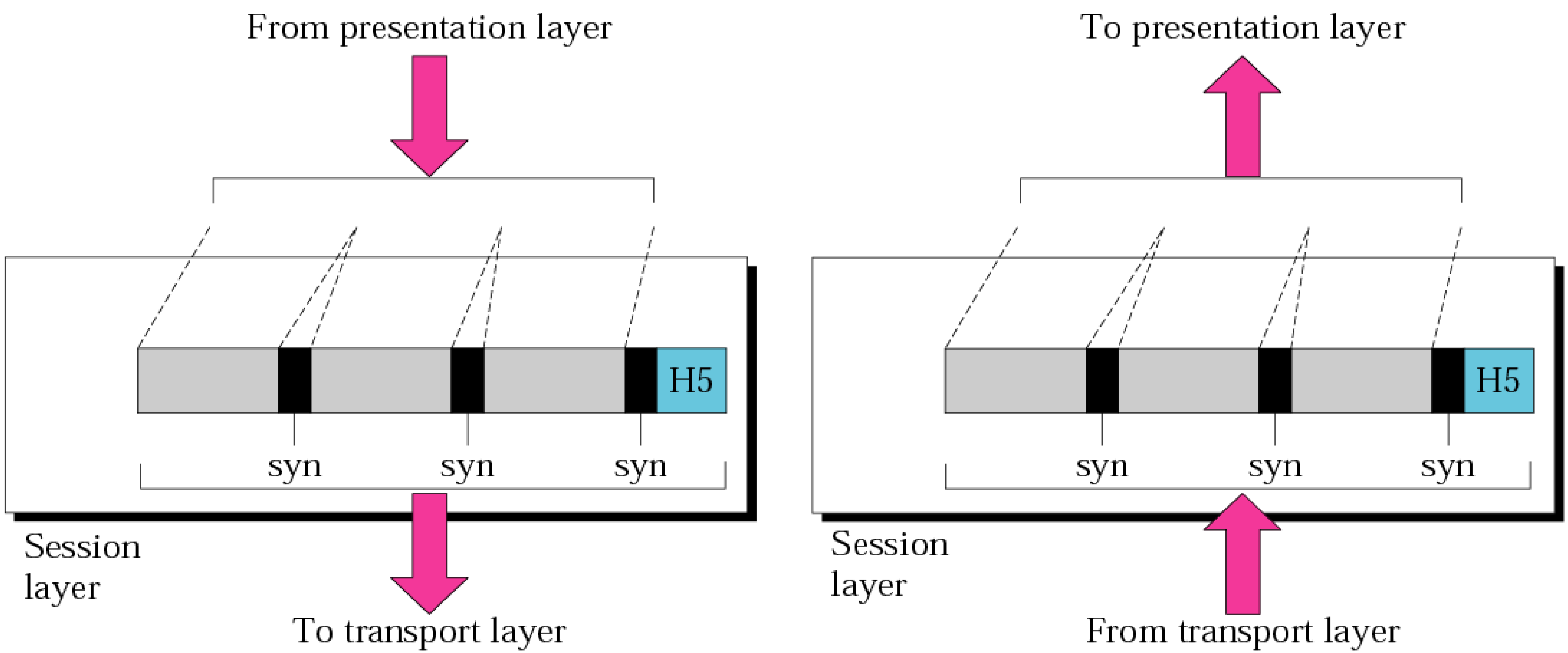
전송 계층 (Transport Layer)

한 프로세스와 자신 또는 다른 노드의 프로세스간 메시지 전송 담당



세션 계층 (Session Layer)

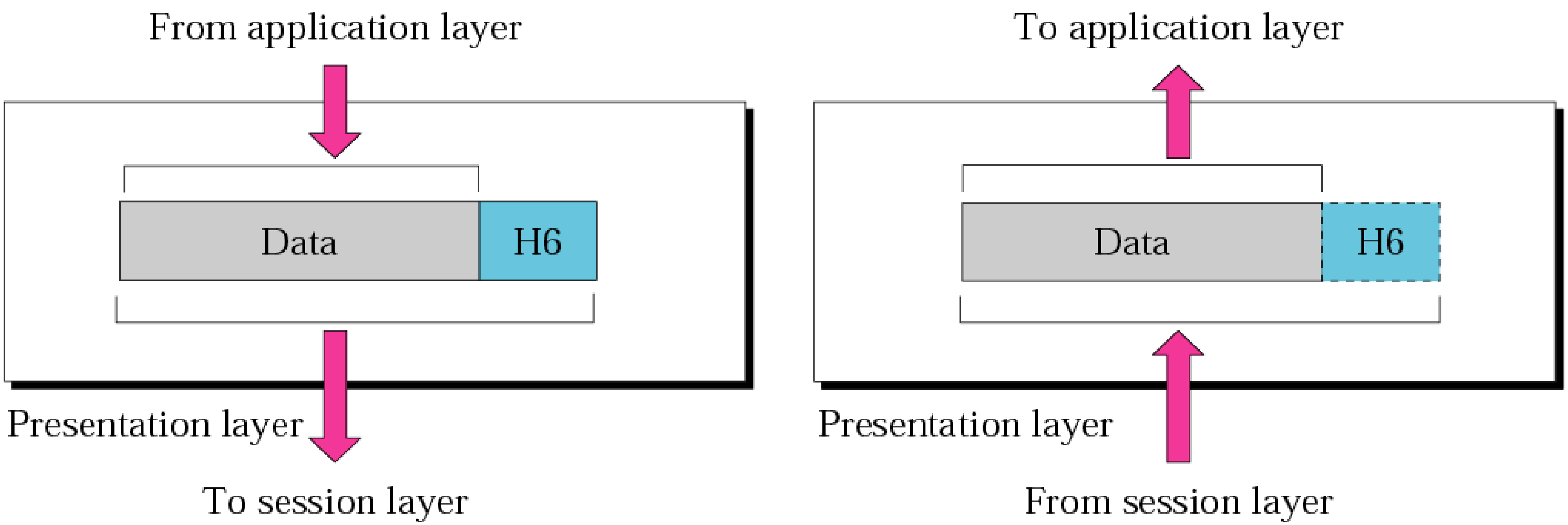
세션의 연결, 관리, 종료를 담당



표현 계층 (Presentation Layer)

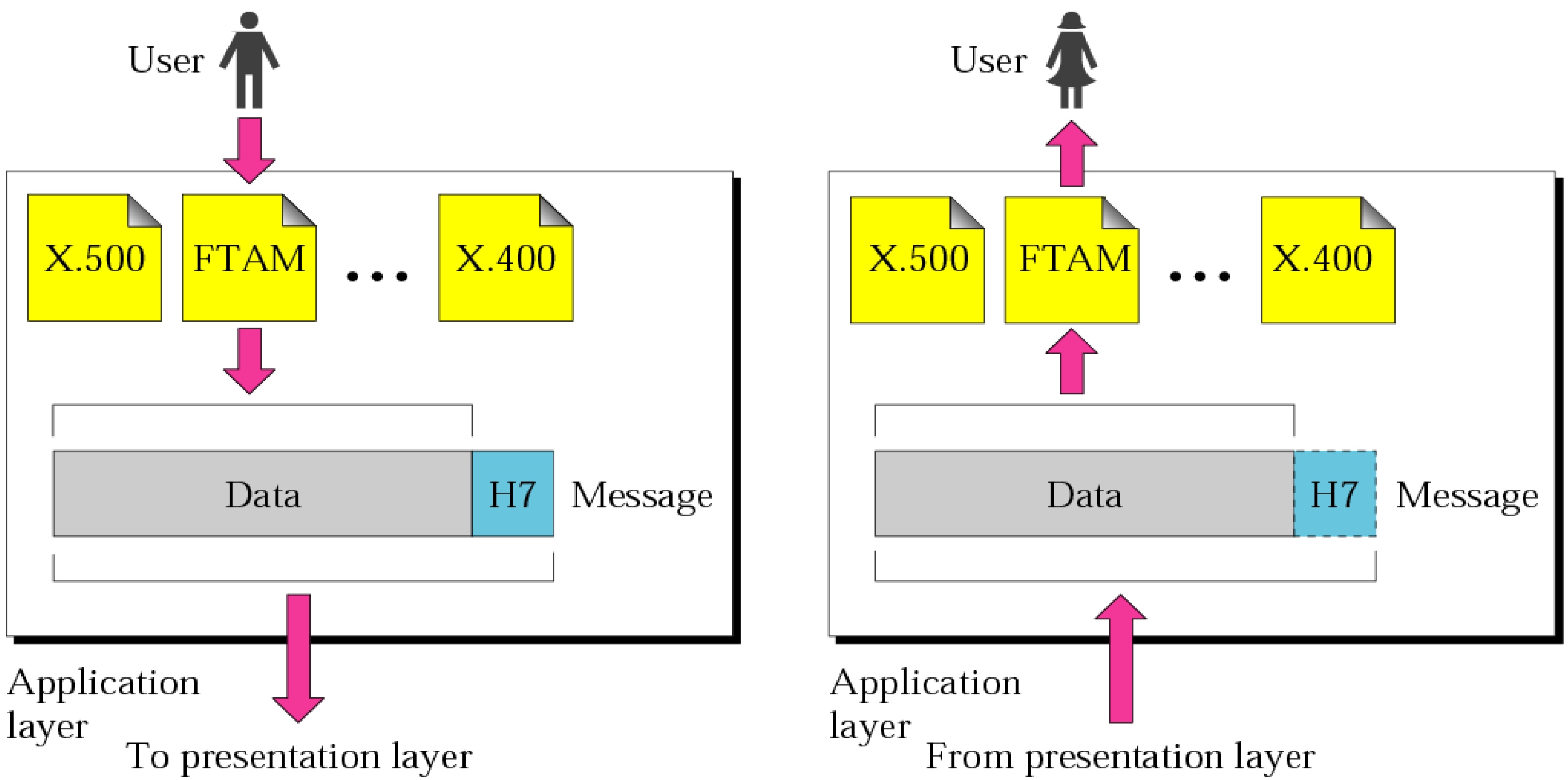
데이터의 암호화와 압축을 담당

- 송신부에서는 데이터의 암호화 및 압축
- 수신부에서는 데이터의 복화화 및 압축 해제

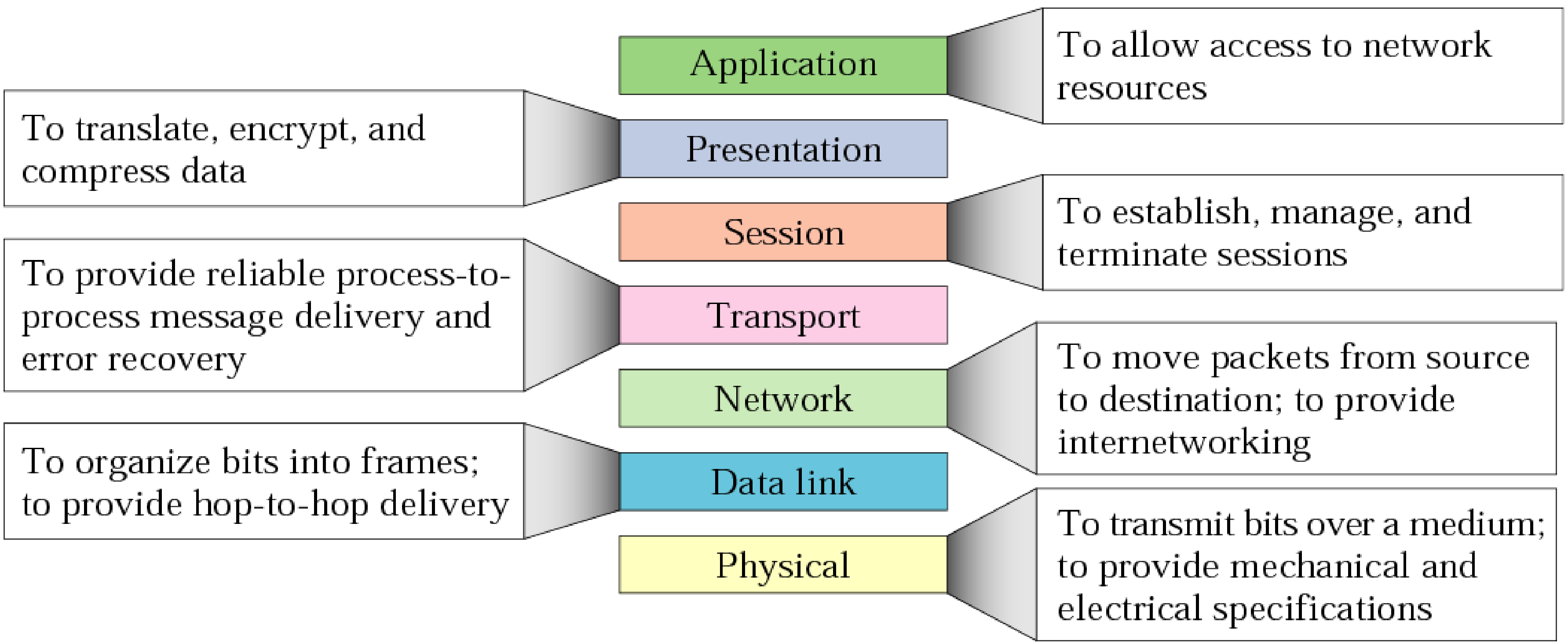


응용 계층 (Application Layer)

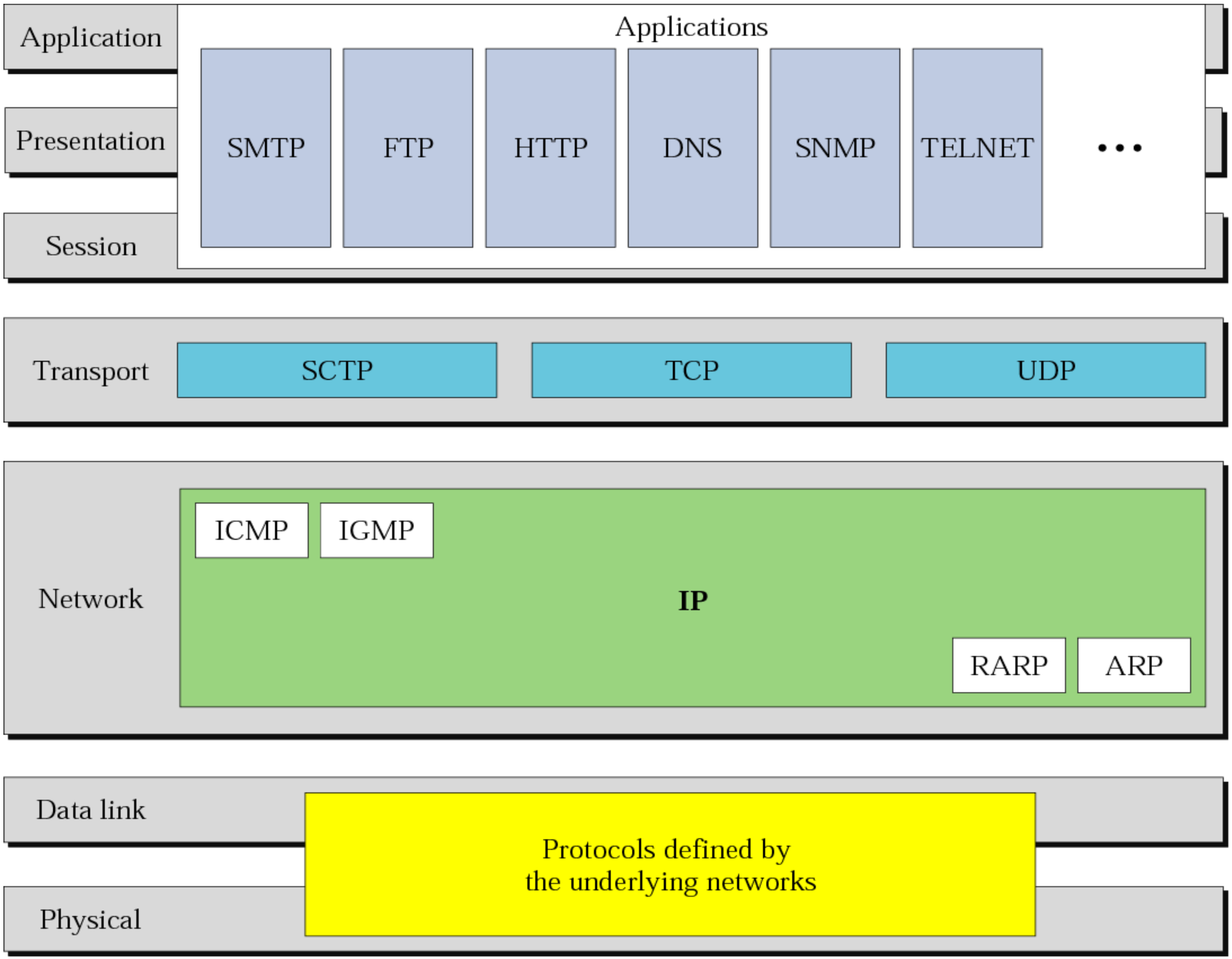
사용자로 하여금 네트워크 자원들에 접근하도록 허용



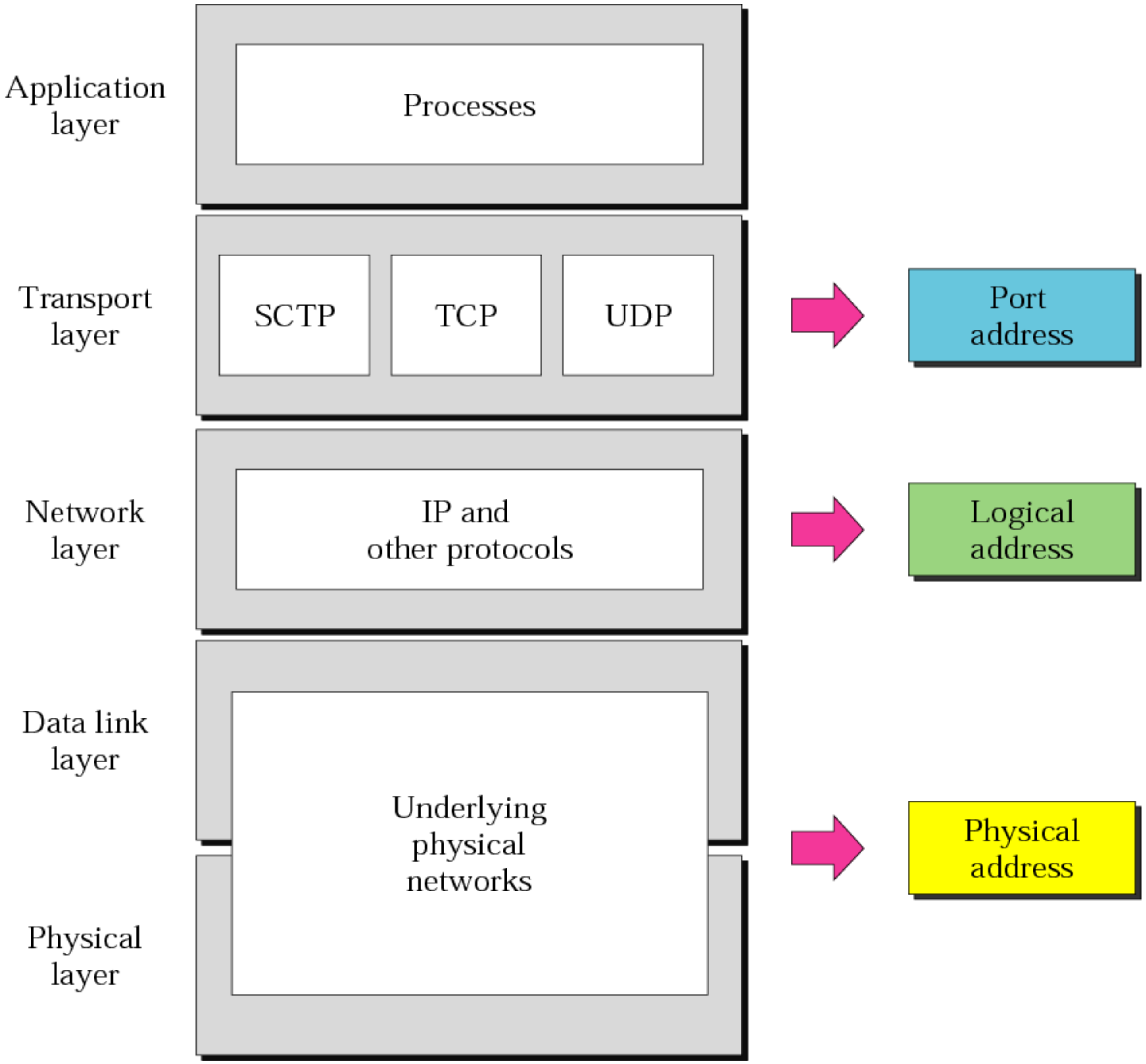
OSI 각 계층 역할 정리


Application
To allow access to network resources
Presentation
To translate, encrypt, and compress data
Session
To establish, manage, and terminate sessions
Transport
To provide reliable process-to-process message delivery and error recovery
Network
To move packets from source to destination; to provide internetworking
Data link
To organize bits into frames; to provide hop-to-hop delivery
Physical
To transmit bits over a medium; to provide mechanical and electrical specifications

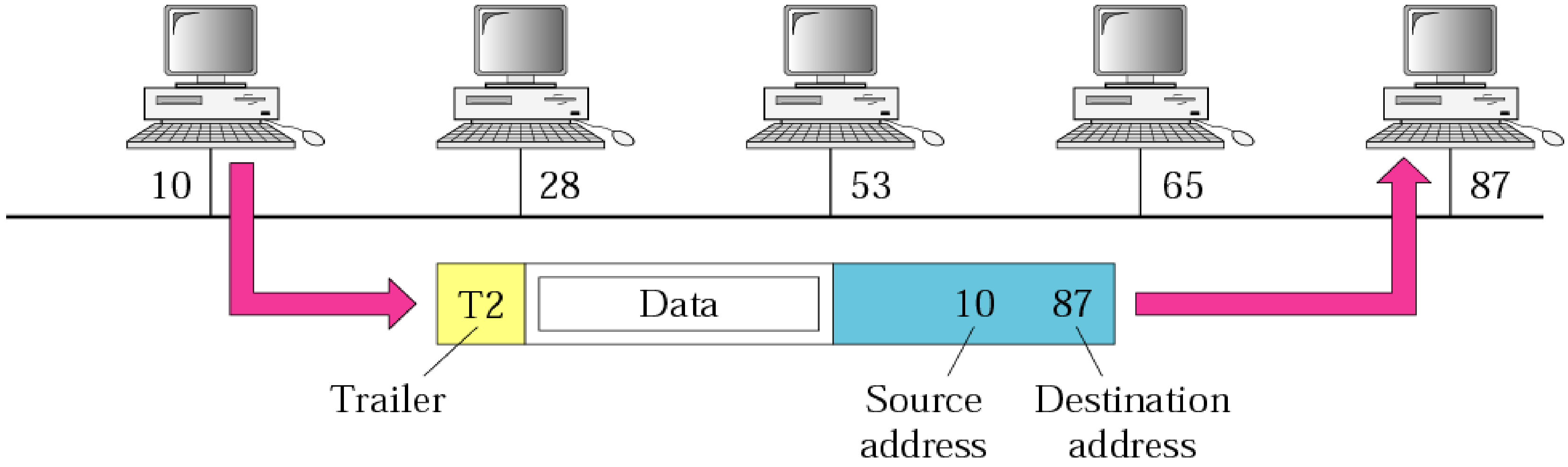
1.2 TCP/IP 프로토콜 및 주소



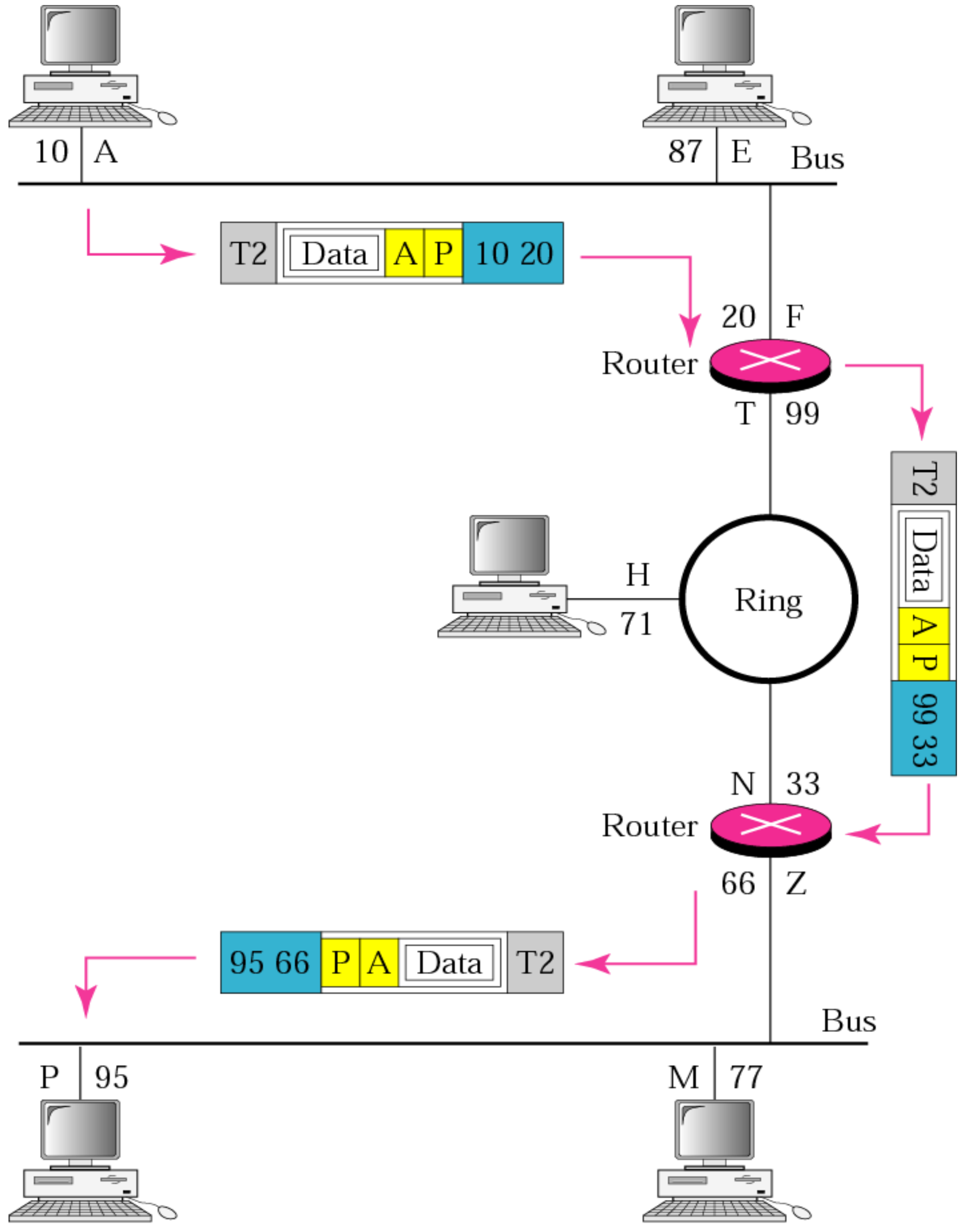
TCP/IP 프로토콜 및 주소 (계속)



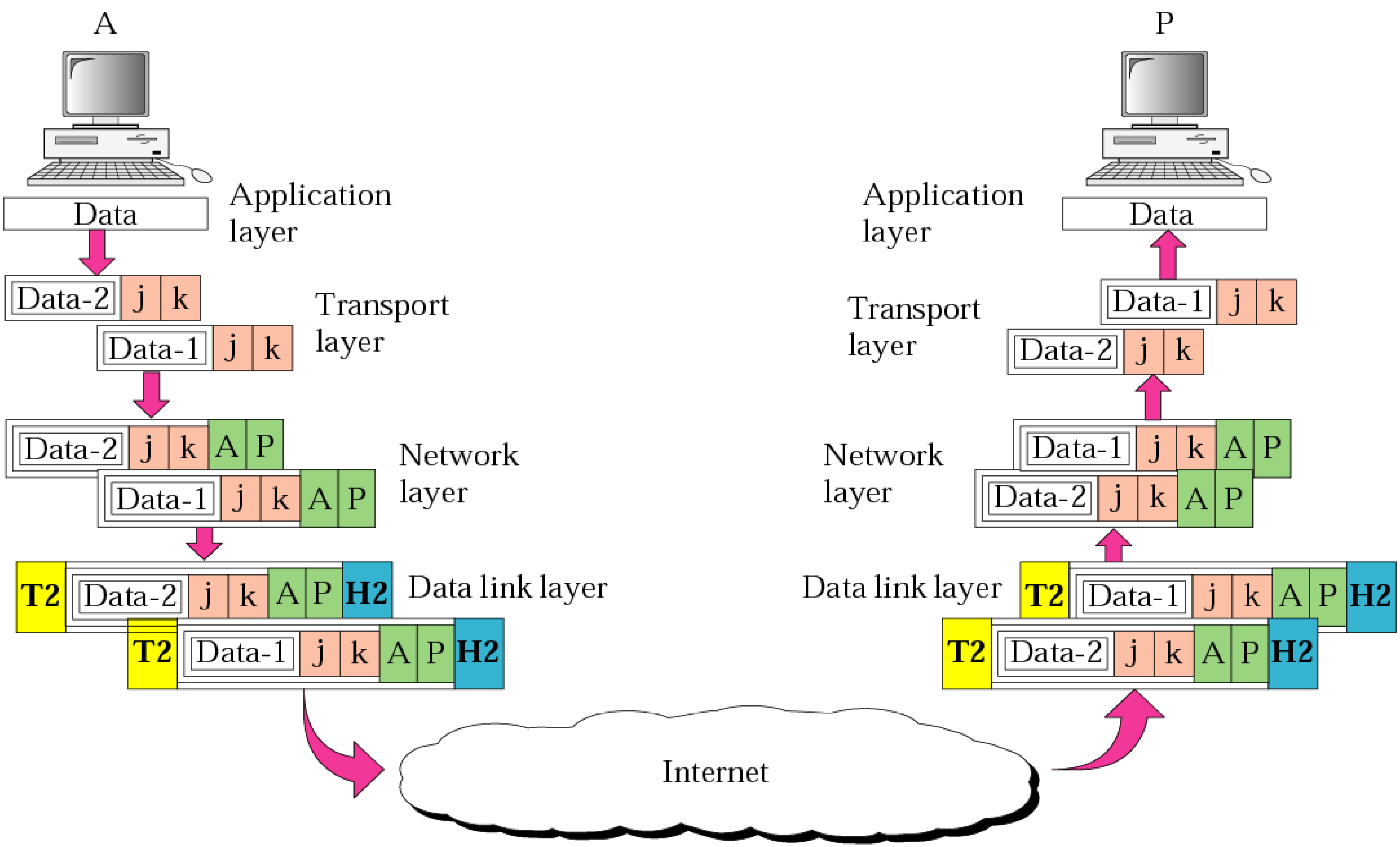
물리 주소를 사용한 Hop-to-hop 전송



IP 주소를 사용한 다른 네트워크의 노드 간 통신



포트 주소를 사용한 프로세스 간 통신



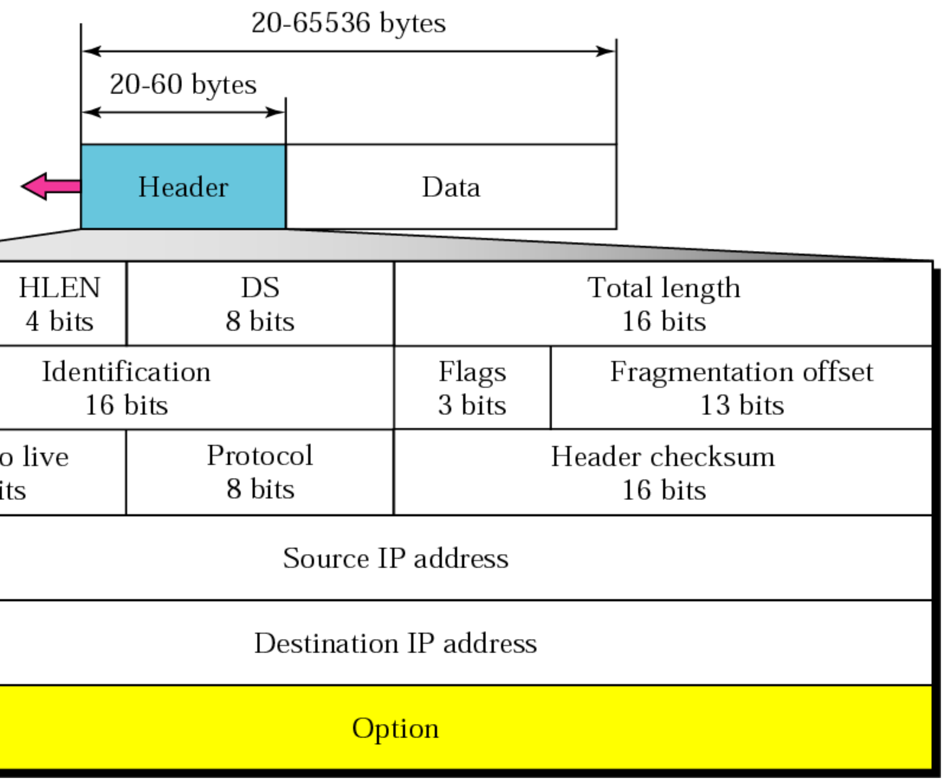
1.3 인터넷 프로토콜 (Internet Protocol)

데이터그램

- 인터넷 프로토콜에서의 패킷을 일컫는 또다른 표현
- 헤더와 데이터로 구성된 가변 길이의 패킷

IP 헤더

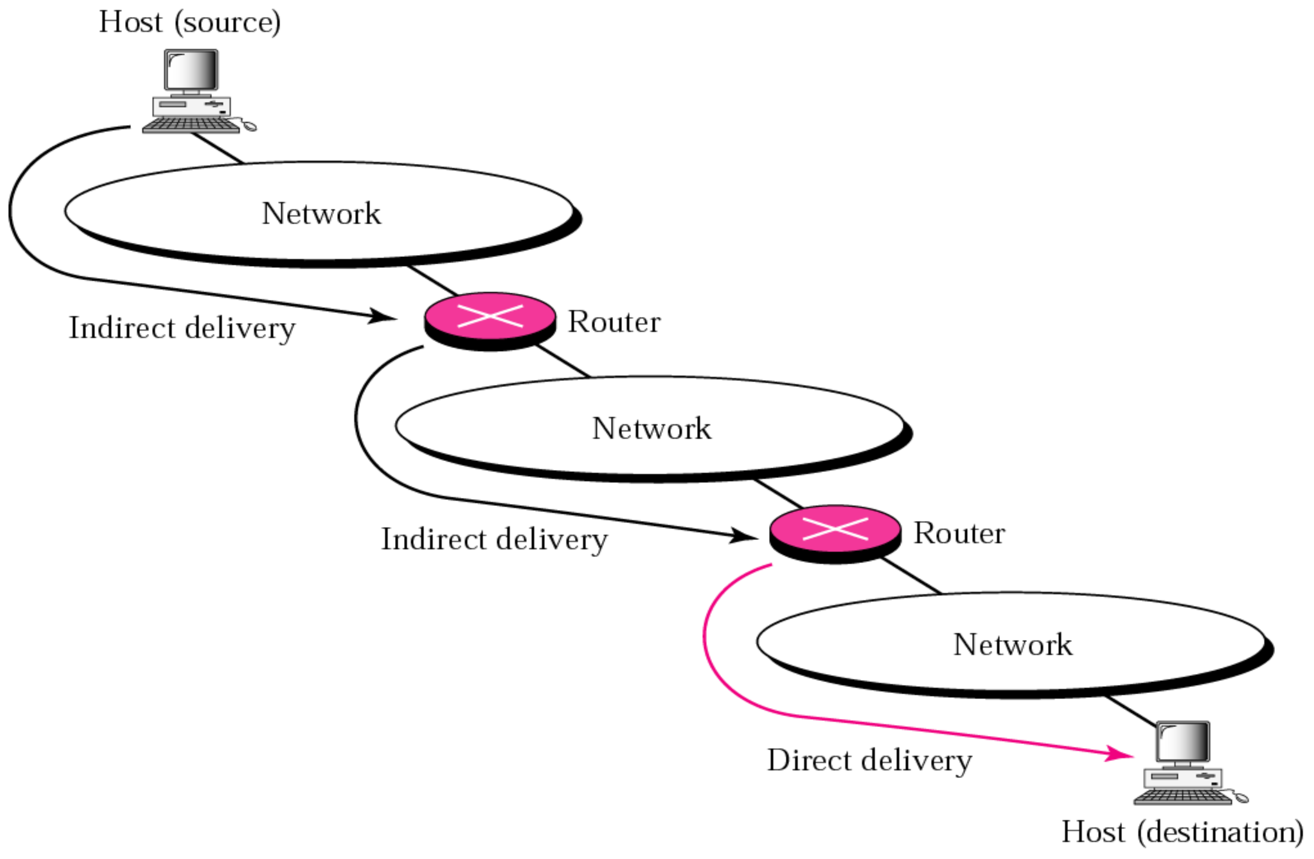
- 20 ~ 60 바이트 길이
- 패킷 라우팅과 전송을 위해서 필수적인 정보들 포함



IP 패킷 라우팅 기법

패킷을 최종 목적지로 전송하기 위한 경로(Route) 탐색

- 호스트 또는 라우터가 라우팅 테이블을 가지고 있음
- Next-hop, Network-specific, Host-specific, Default 라우팅 방식



Next-hop 라우팅 방식

Routing table for host A

Destination	Route
Host B	R1, R2, Host B

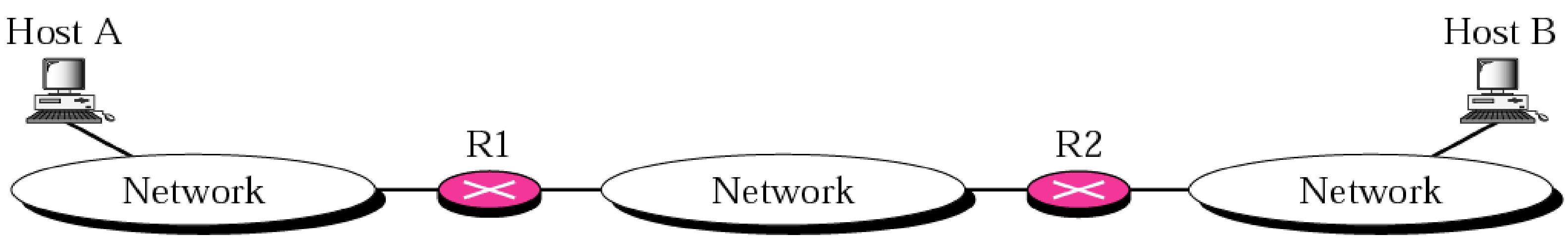
Routing table for R1

Destination	Route
Host B	R2, Host B

Routing table for R2

Destination	Route
Host B	Host B

a. Routing tables based on route



Routing table for host A

Destination	Next Hop
Host B	R1

Routing table for R1

Destination	Next Hop
Host B	R2

Routing table for R2

Destination	Next Hop
Host B	Ñ

b. Routing tables based on next hop

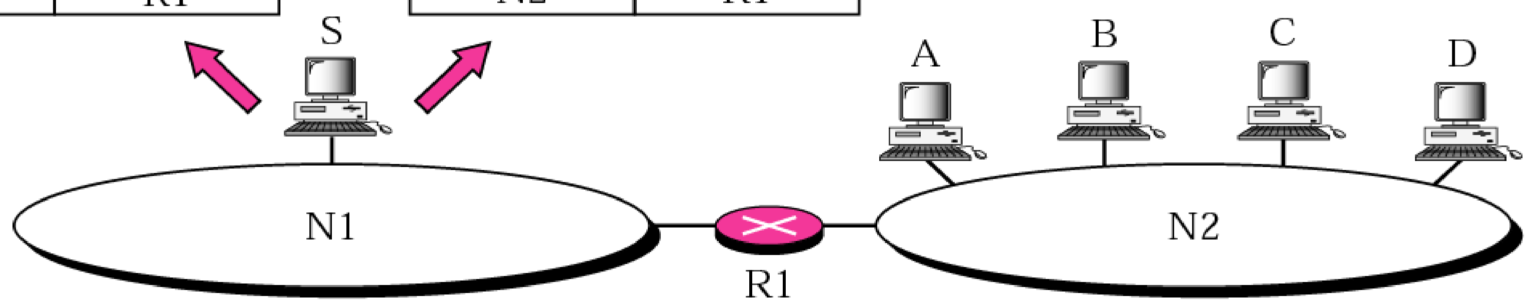
Network-specific 라우팅 방식

Routing table for host S based
on host-specific method

Destination	Next Hop
A	R1
B	R1
C	R1
D	R1

Routing table for host S based
on network-specific method

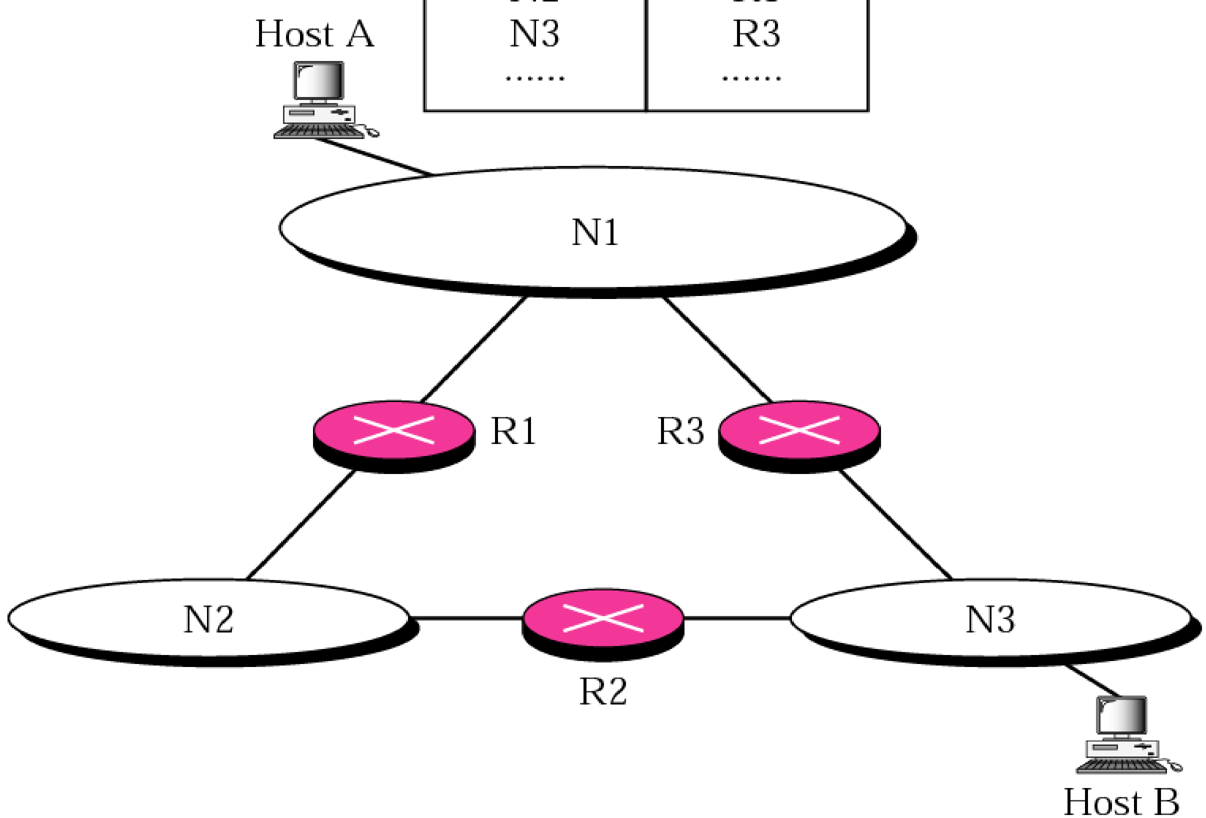
Destination	Next Hop
N2	R1



Host-specific 라우팅 방식

Routing table for host A

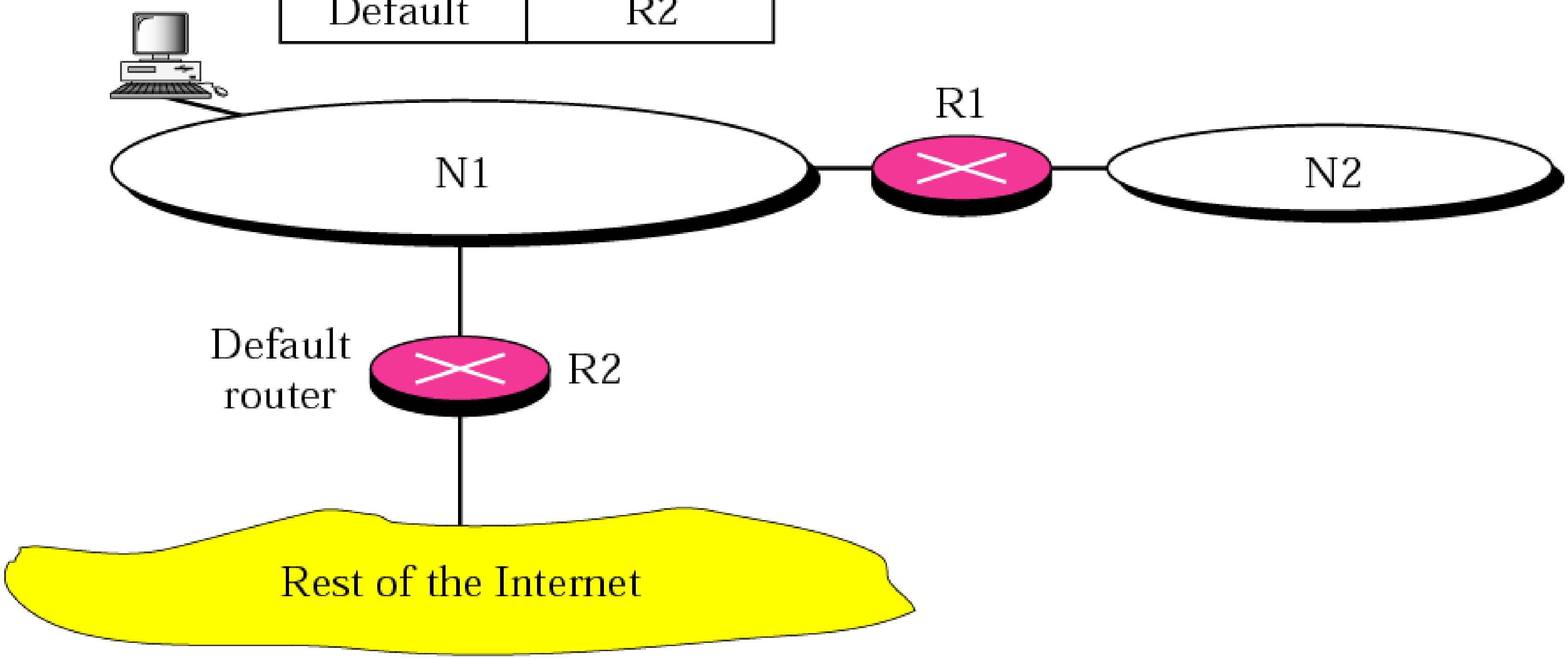
Destination	Next Hop
Host B	R3
N2	R1
N3	R3
......	......



Default 라우팅 방식

Routing table for host A

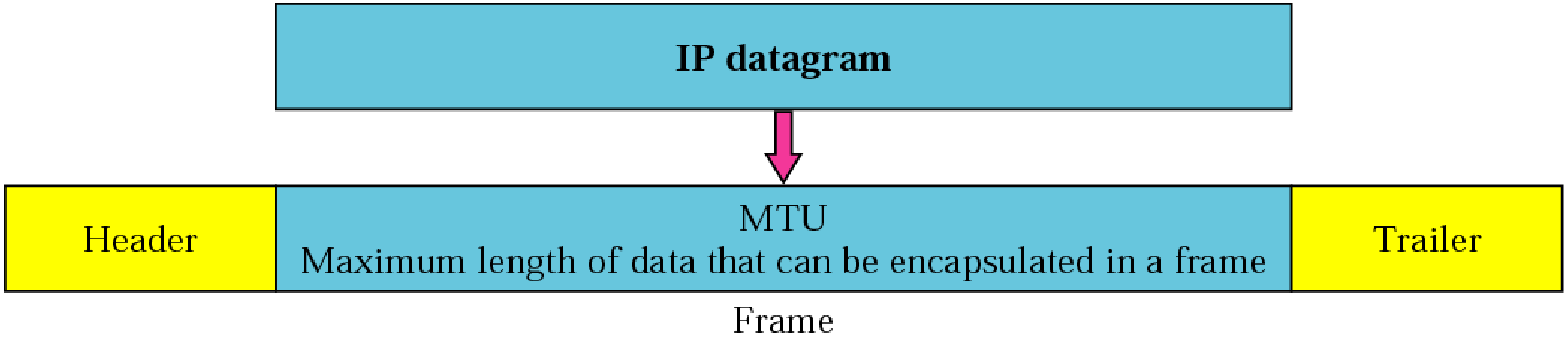
Destination	Next Hop
N2	R1
......	......
Default	R2



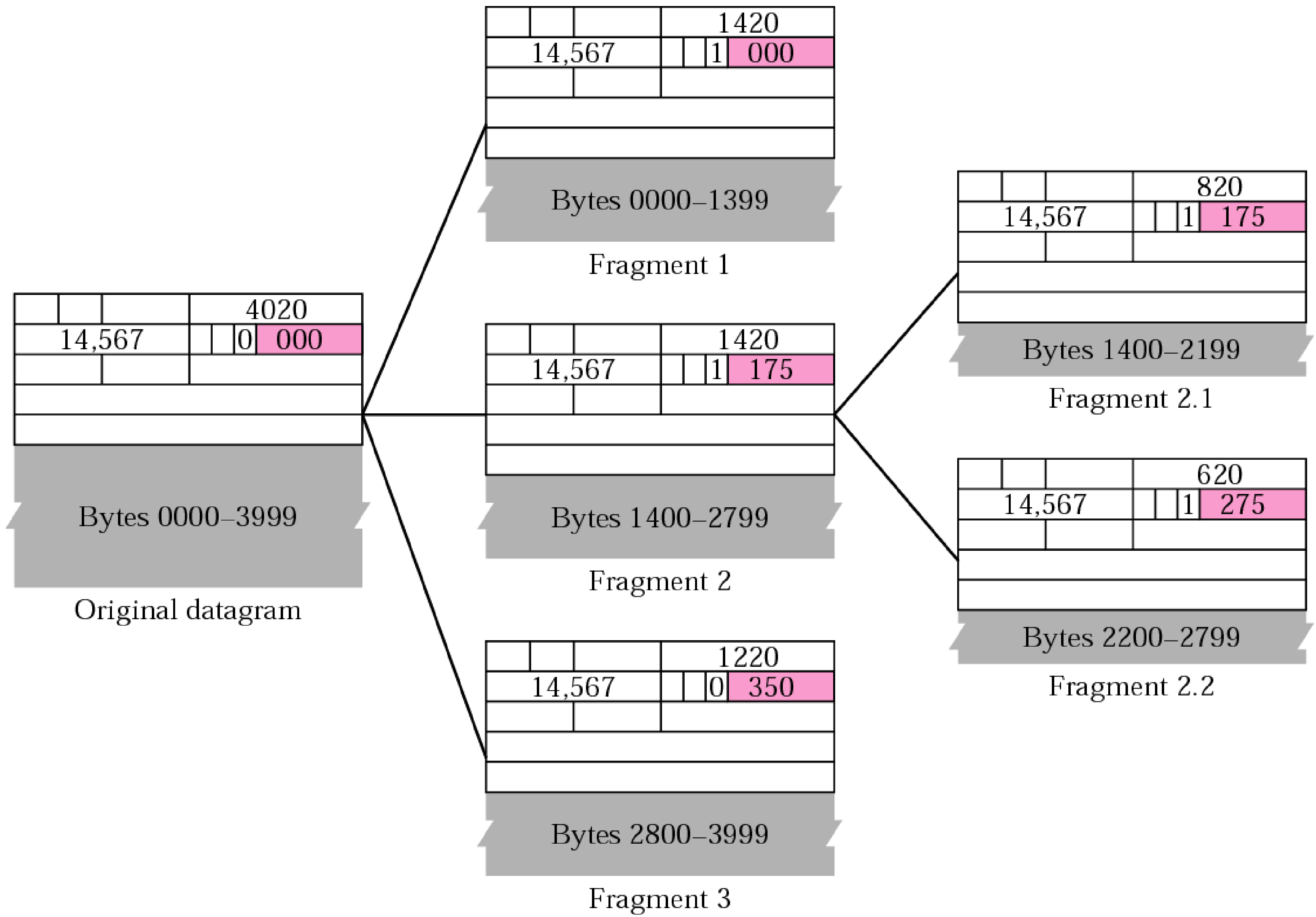
Fragmentation and defragmentation

데이터그램이 하위 물리 프로토콜 규정에 맞게 여러개의 패킷으로 나뉘어지고 다시 하나로 통합되는 작업

- 전송될 프레임의 형식과 크기(MTU)는 물리 네트워크 종류에 의해서 결정



Fragmentation 예제



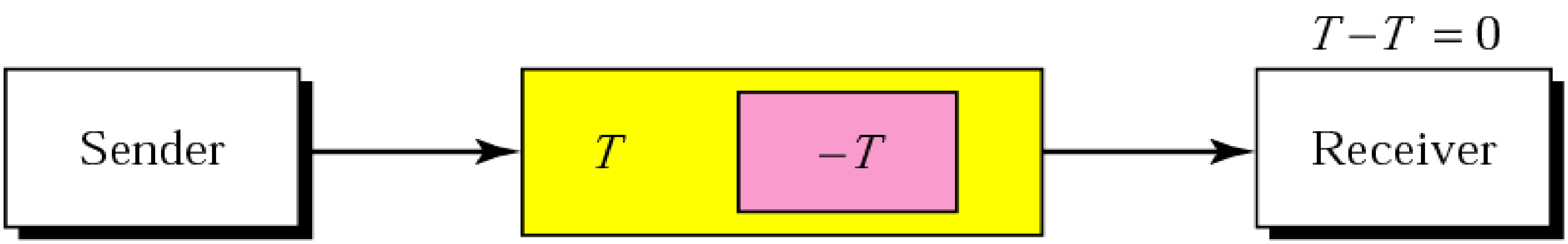
체크섬 (Checksum)

체크섬이란

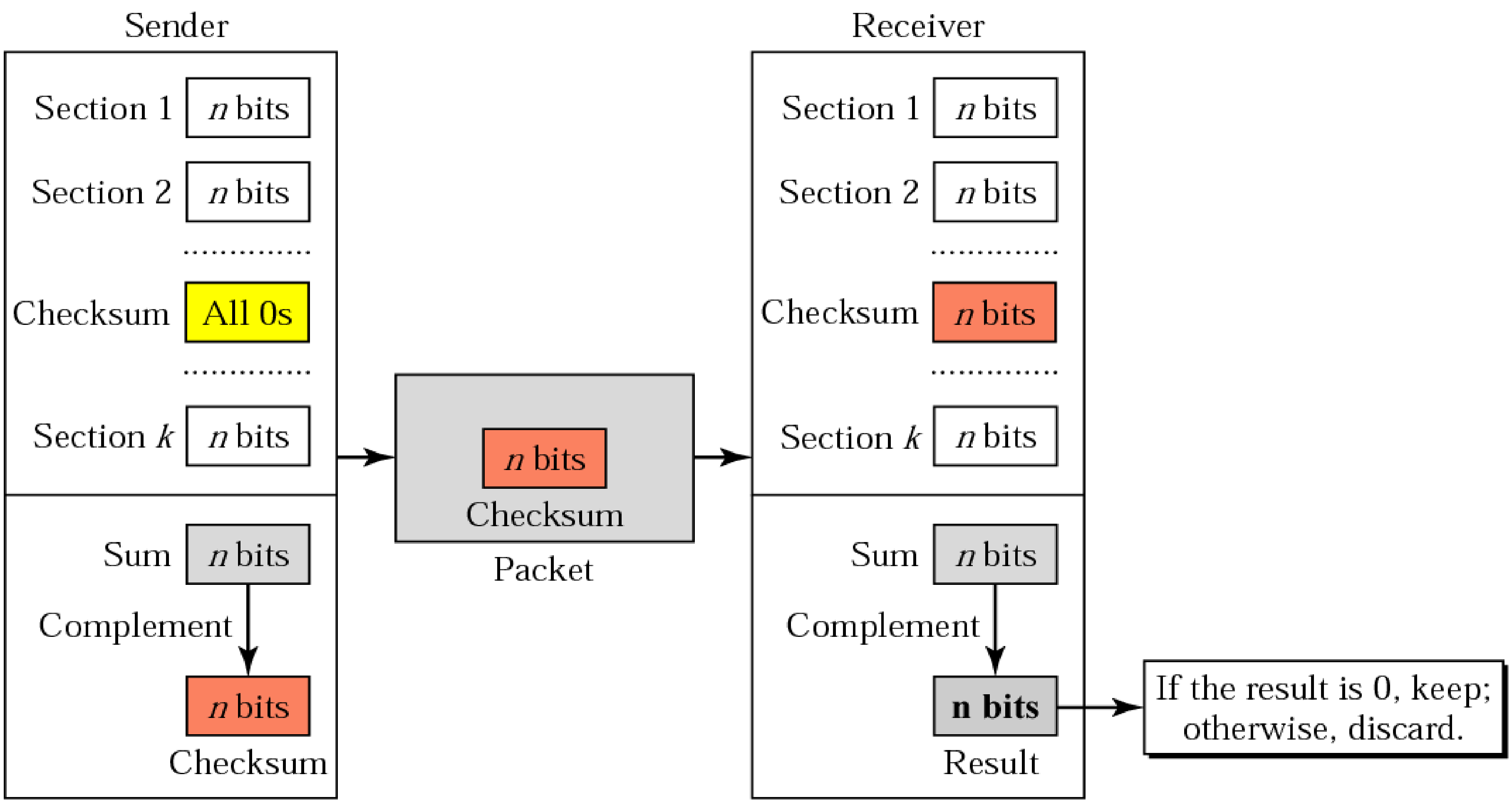
- 대부분의 TCP/IP 프로토콜에 의해서 사용되는 에러 검출 방식
- 패킷을 전송하는 동안 발생 가능한 데이터 Corruption 복구

송신측과 수신측에서 체크섬을 생성하고 확인하는 방식

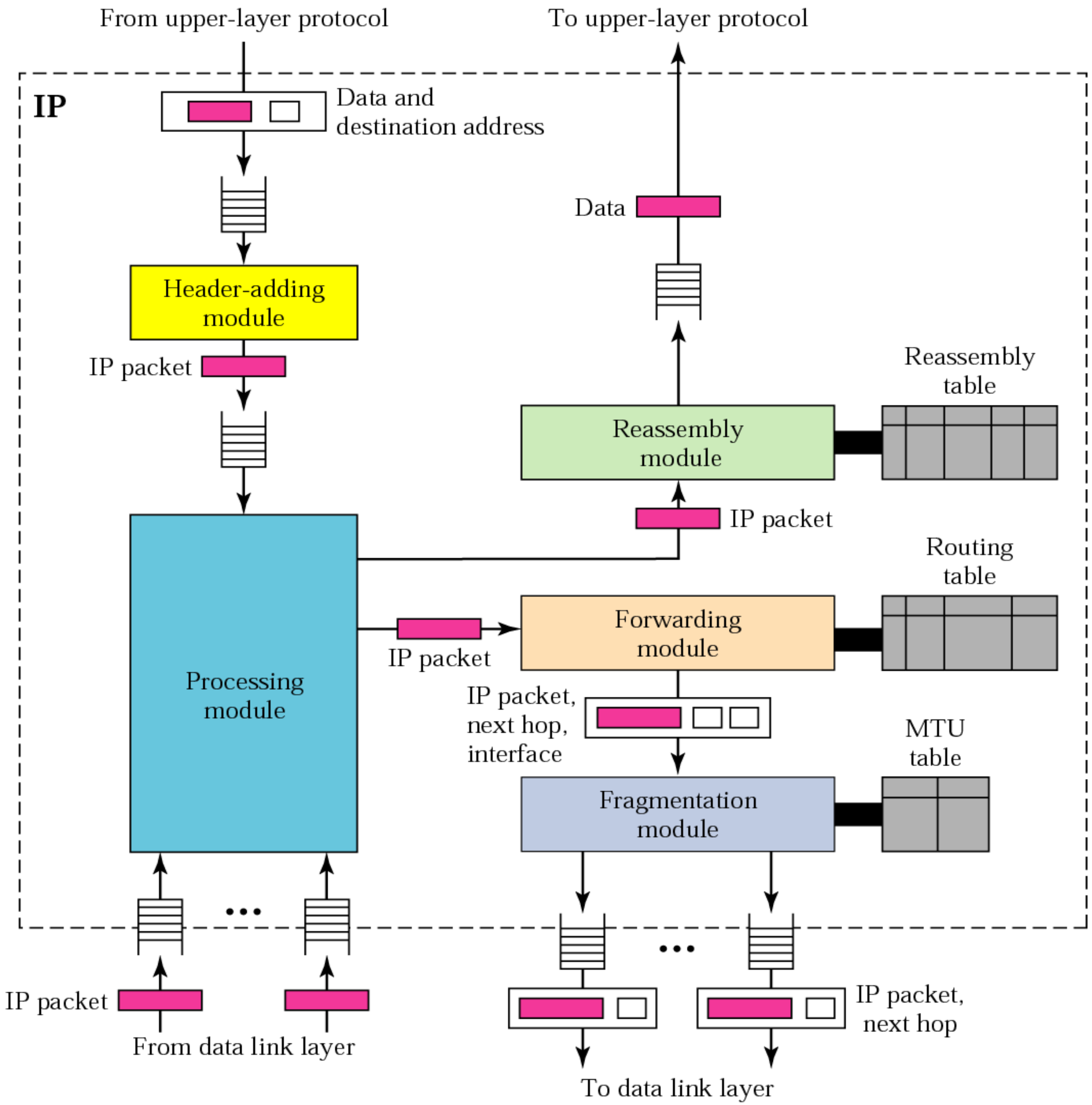
- 패킷을 n 비트 단위의 k개 섹션으로 나눔
- 모든 섹션의 데이터를 1의 보수 연산 방식으로 더함
- 이와 같은 방식으로 생성된 값에 대해서 ***1의 보수값을 최종 생성
  - 송신측: 생성된 값이 체크섬 값
  - 수신측: 해당 값이 0일 경우 온전한 데이터임을 확인



체크섬 개념 도식화

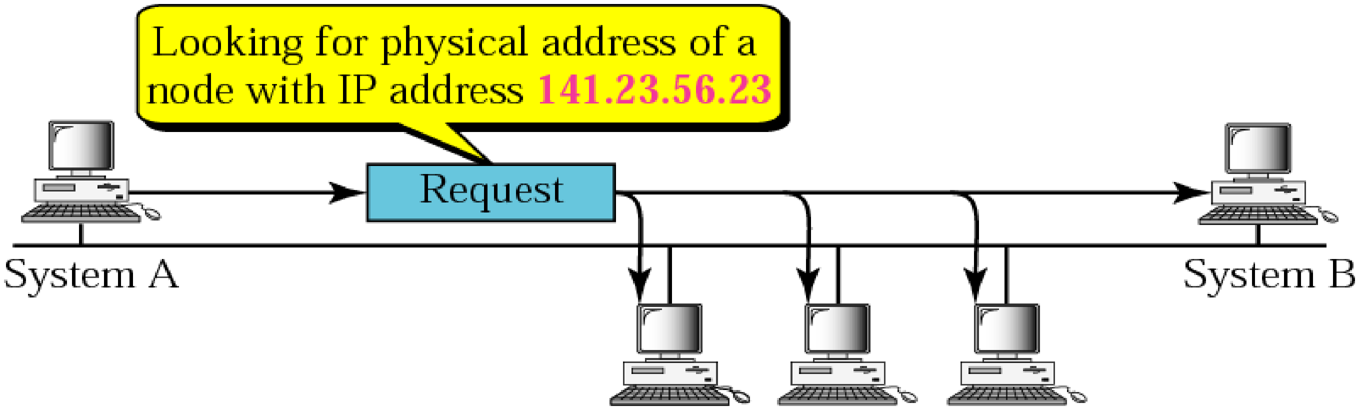

Sender
Section 1
n bits
Section 2
n bits
Checksum
All 0s
Section k
n bits
Sum
n bits
Complement
n bits
Checksum
n bits
Checksum
Packet
Receiver
Section 1
n bits
Section 2
n bits
Checksum
n bits
Section k
n bits
Sum
n bits
Complement
n bits
Result
If the result is 0, keep;
otherwise, discard.

IP 구성요소

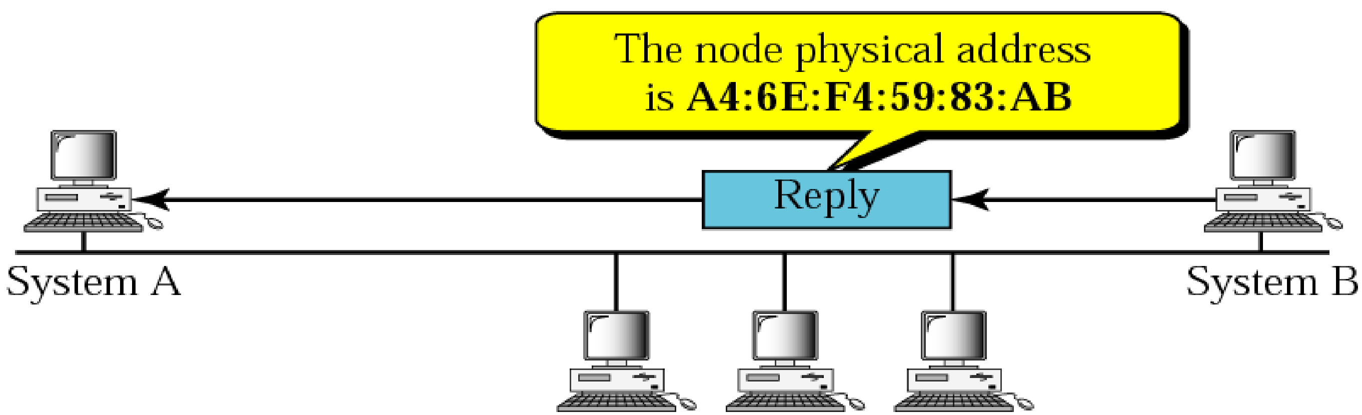

From upper-layer protocol
To upper-layer protocol
IP
Data and
destination address
Data
Header-adding
module
IP packet
Reassembly
table
Reassembly
module
IP packet
Routing
table
Forwarding
module
IP packet
Processing
module
IP packet,
next hop,
interface
MTU
table
Fragmentation
module
IP packet
From data link layer
IP packet,
next hop
To data link layer

ARP 프로토콜

IP 주소를 물리 주소로 변환하는 프로토콜



a. ARP request is broadcast

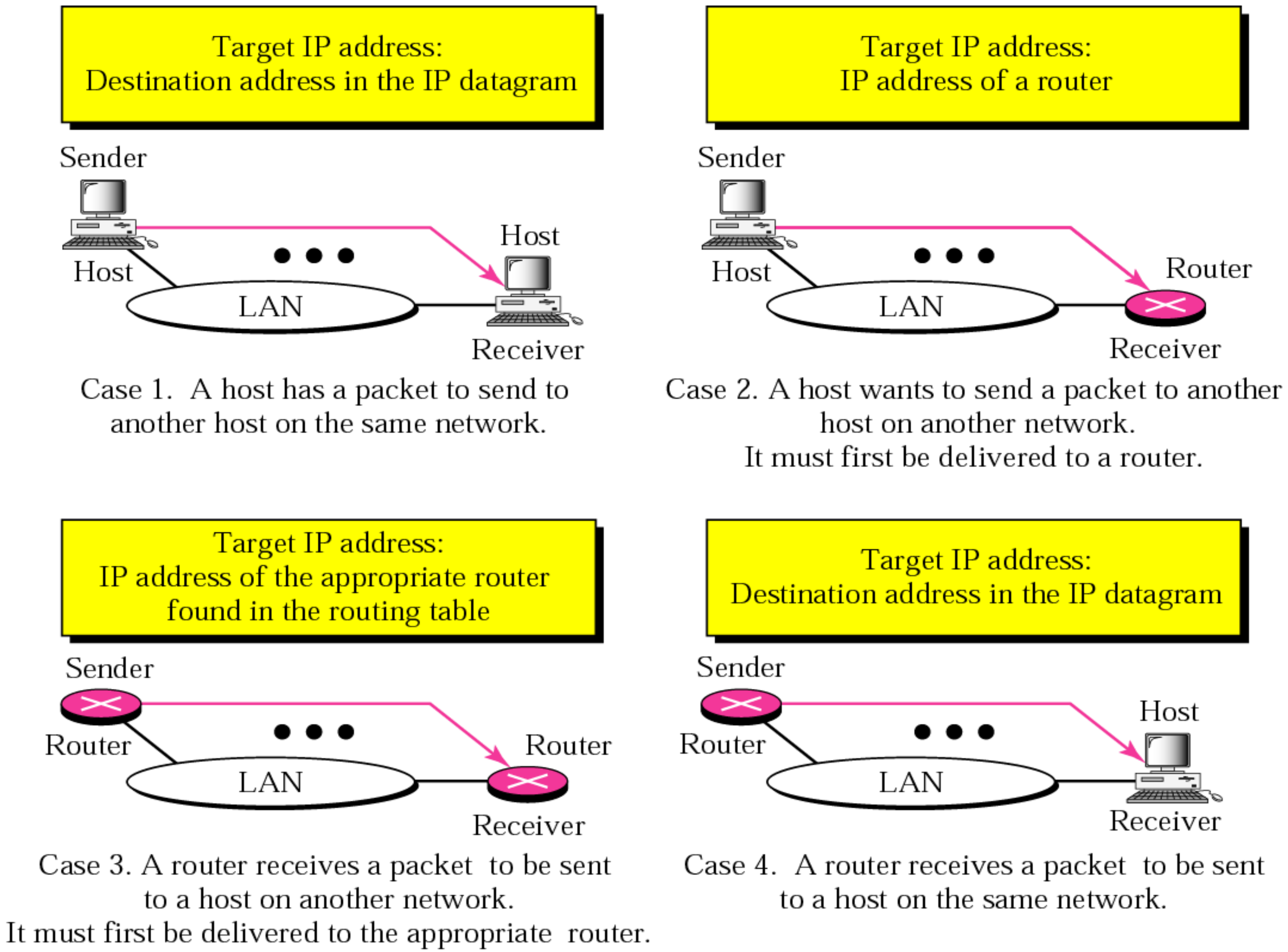


b. ARP reply is unicast

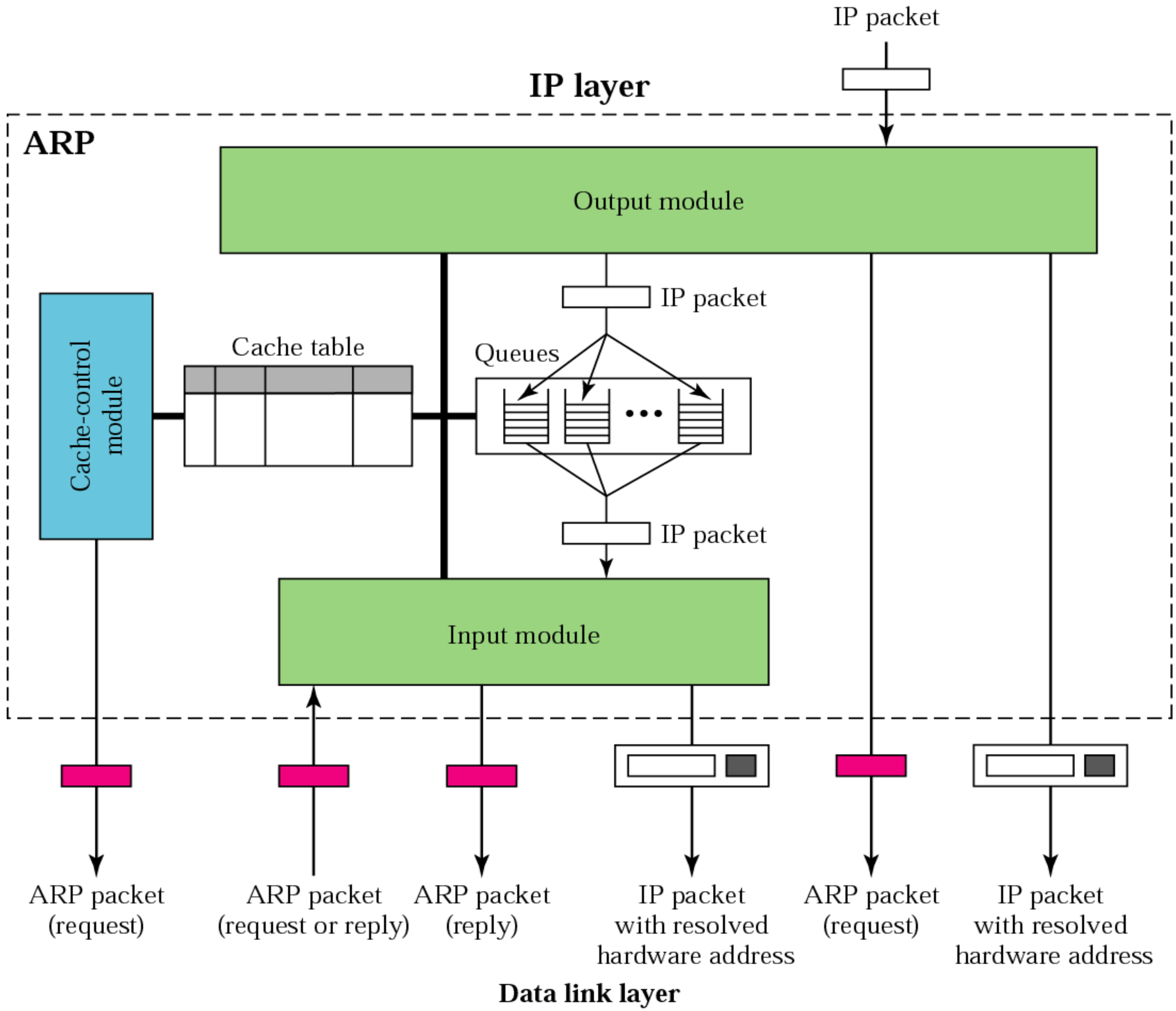
ARP 패킷 내부 구조

<table>
<tr><td colspan="2">Hardware Type</td><td>Protocol Type</td></tr>
<tr><td>Hardware
length</td><td>Protocol
length</td><td>Operation
Request 1, Reply 2</td></tr>
<tr><td colspan="3">Sender hardware address
(For example, 6 bytes for Ethernet)</td></tr>
<tr><td colspan="3">Sender protocol address
(For example, 4 bytes for IP)</td></tr>
<tr><td colspan="3">Target hardware address
(For example, 6 bytes for Ethernet)
(It is not filled in a request)</td></tr>
<tr><td colspan="3">Target protocol address
(For example, 4 bytes for IP)</td></tr>
</table>

ARP를 사용하는 네 가지 경우

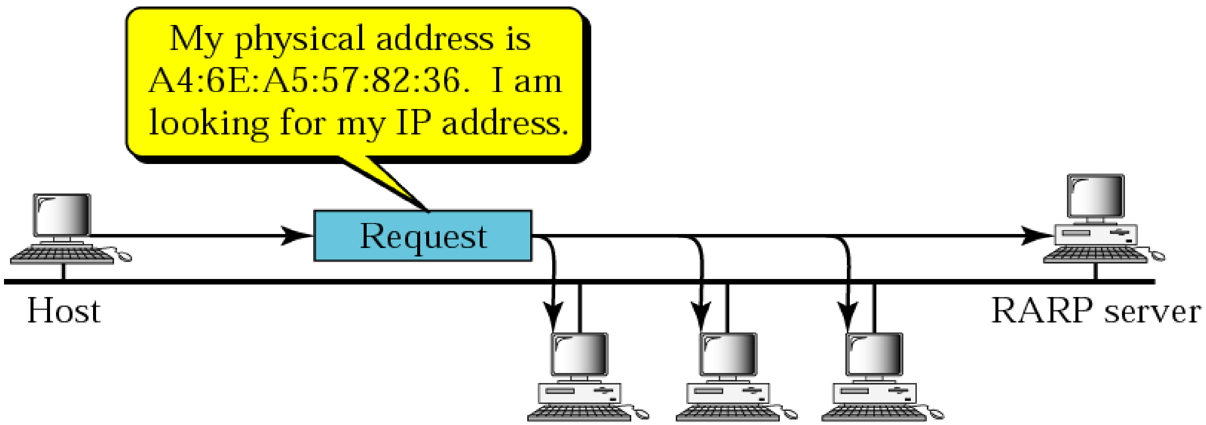


ARP 구성요소

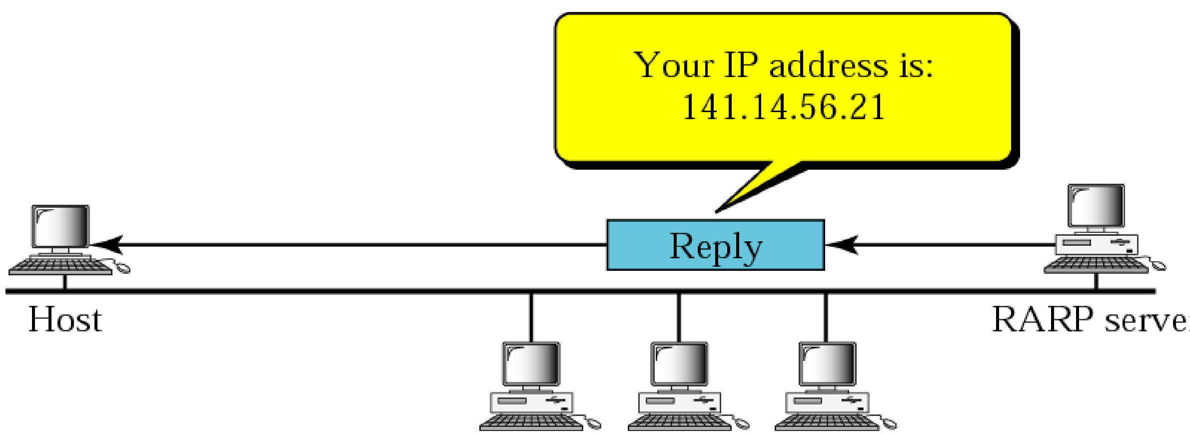

IP packet
IP layer
ARP
Output module
IP packet
Cache table
Queues
Cache-control module
IP packet
Input module
ARP packet (request)
ARP packet (request or reply)
ARP packet (reply)
IP packet with resolved hardware address
ARP packet (request)
IP packet with resolved hardware address
Data link layer

RARP (Reversed ARP) 프로토콜

특정 물리 주소를 가진 시스템에 대한 IP 주소를 확인하는 프로토콜



a. RARP request is broadcast



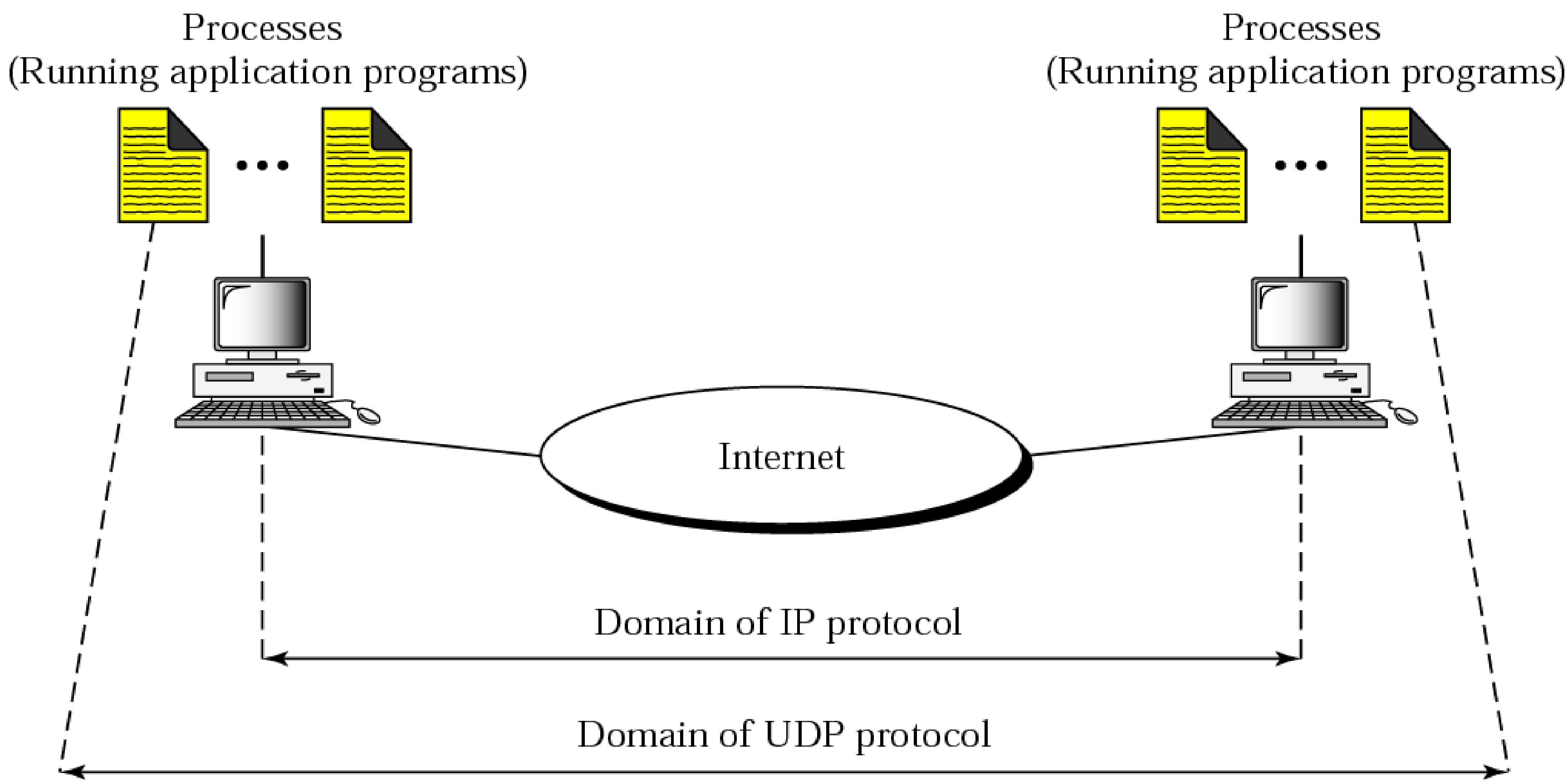
b. RARP reply is unicast

RARP 패킷 내부 구조

<table>
<tr><td colspan="2">Hardware type</td><td>Protocol type</td></tr>
<tr><td>Hardware
length</td><td>Protocol
length</td><td>Operation
Request 3, Reply 4</td></tr>
<tr><td colspan="3">Sender hardware address
(For example, 6 bytes for Ethernet)</td></tr>
<tr><td colspan="3">Sender protocol address
(For example, 4 bytes for IP)
(It is not filled for request)</td></tr>
<tr><td colspan="3">Target hardware address
(For example, 6 bytes for Ethernet)
(It is not filled for request)</td></tr>
<tr><td colspan="3">Target protocol address
(For example, 4 bytes for IP)
(It is not filled for request)</td></tr>
</table>

1.4 UDP 프로토콜 (User Datagram Protocol)

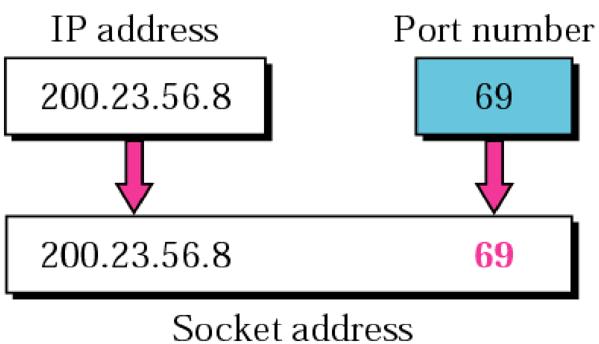
프로세스간 통신 지원 - Connectionless 통신 방식

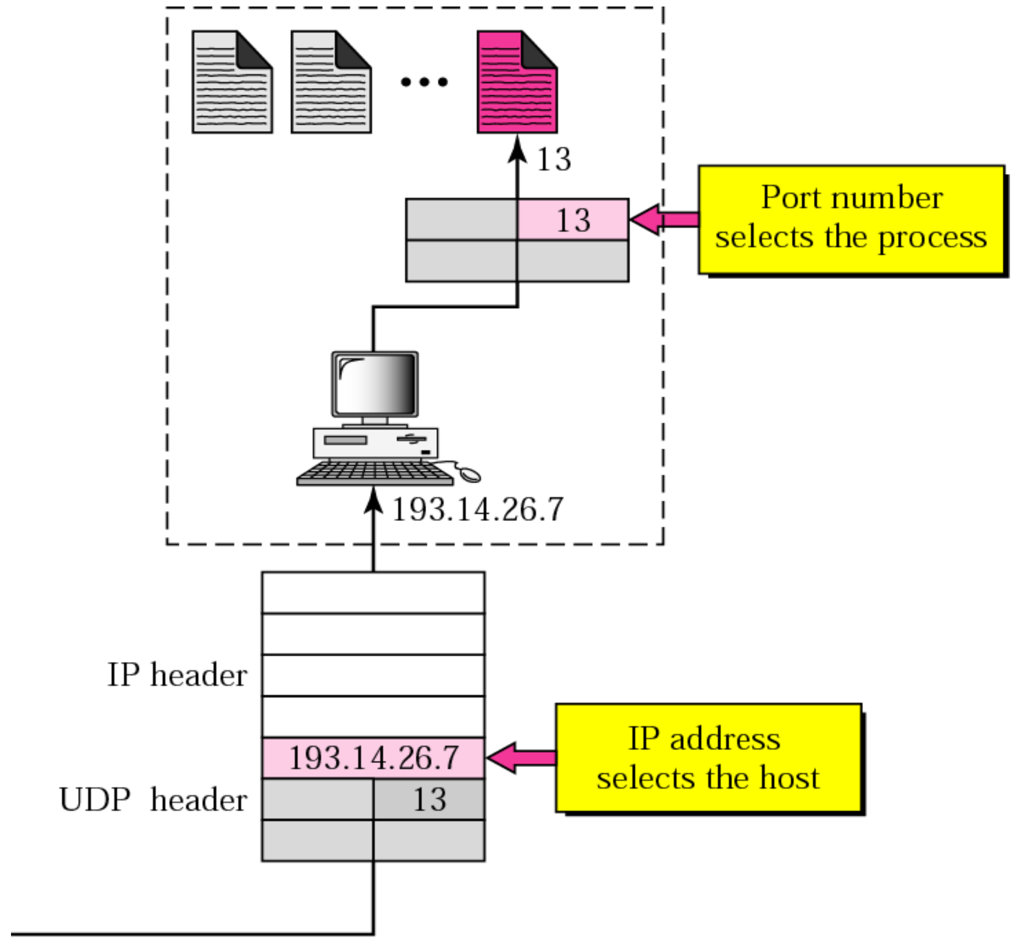


소켓 주소: IP 주소와 포트 번호

IP 주소는 호스트간 통신에서 사용

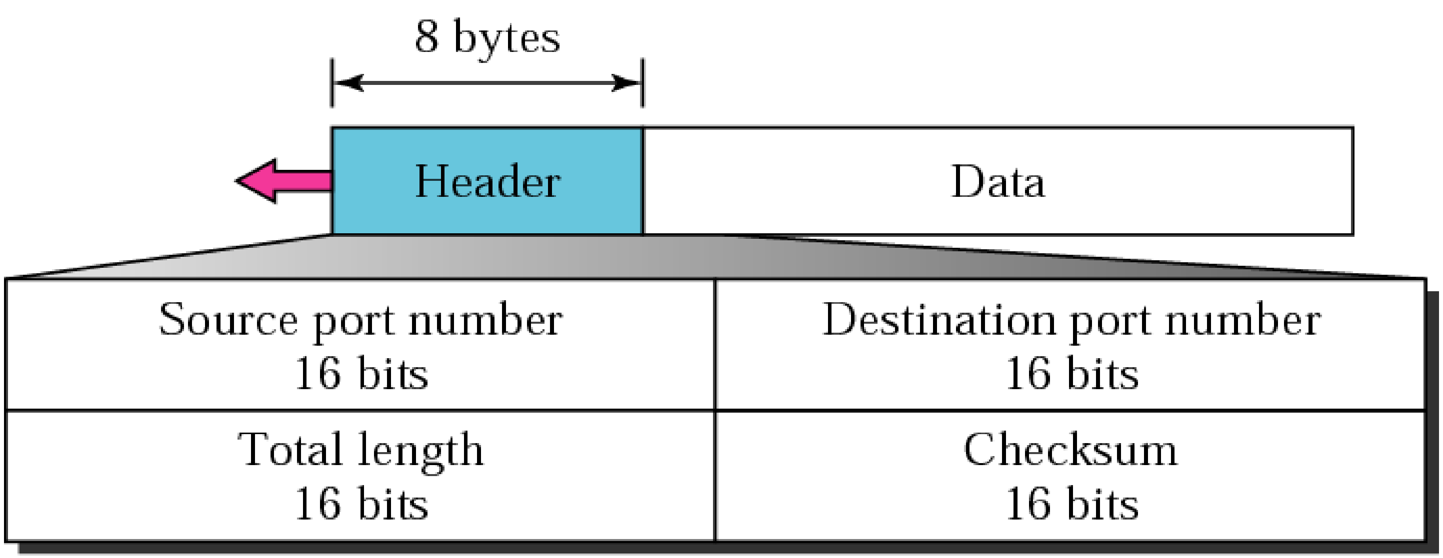
포트번호는 프로세스간 통신을 위해서 사용





UDP 데이터그램 형식

고정된 크기의 8 바이트 헤더와 데이터로 구성

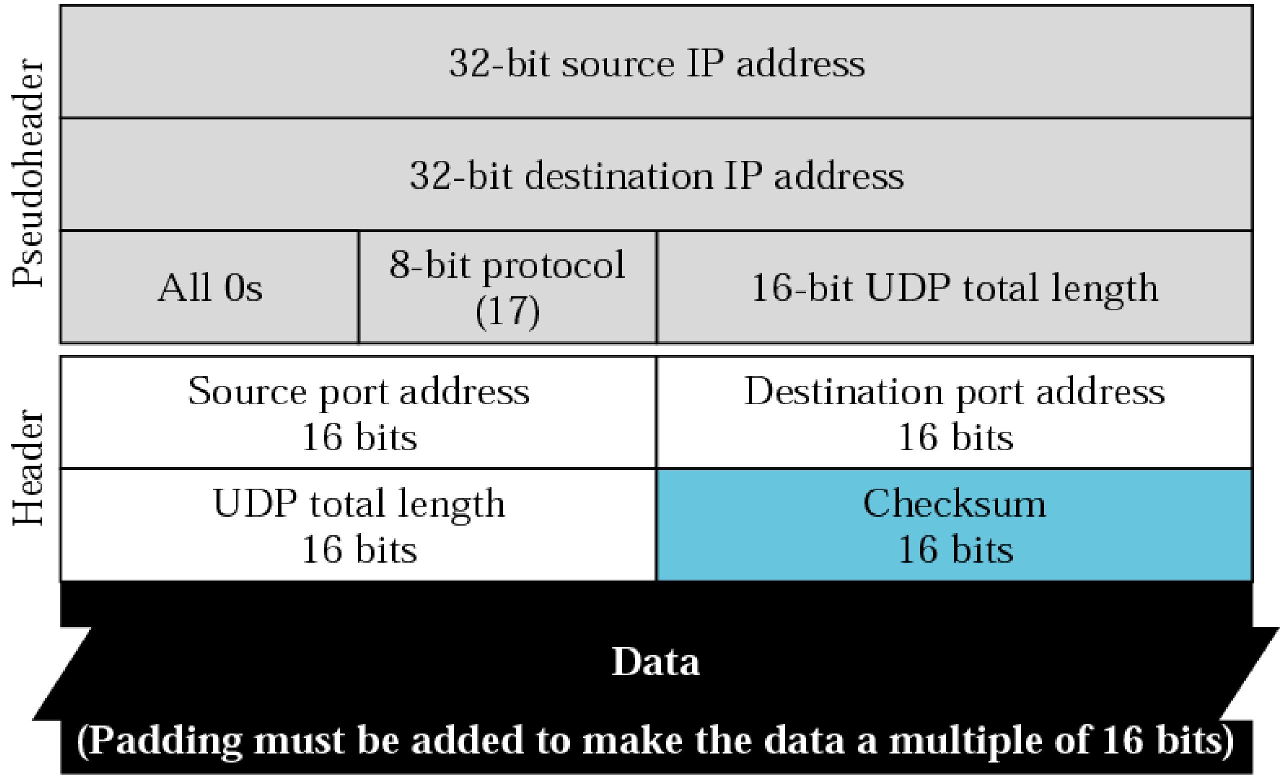


UDP 체크섬

IP 프로토콜에서와 마찬가지로 UDP 프로토콜에서도 체크섬 사용

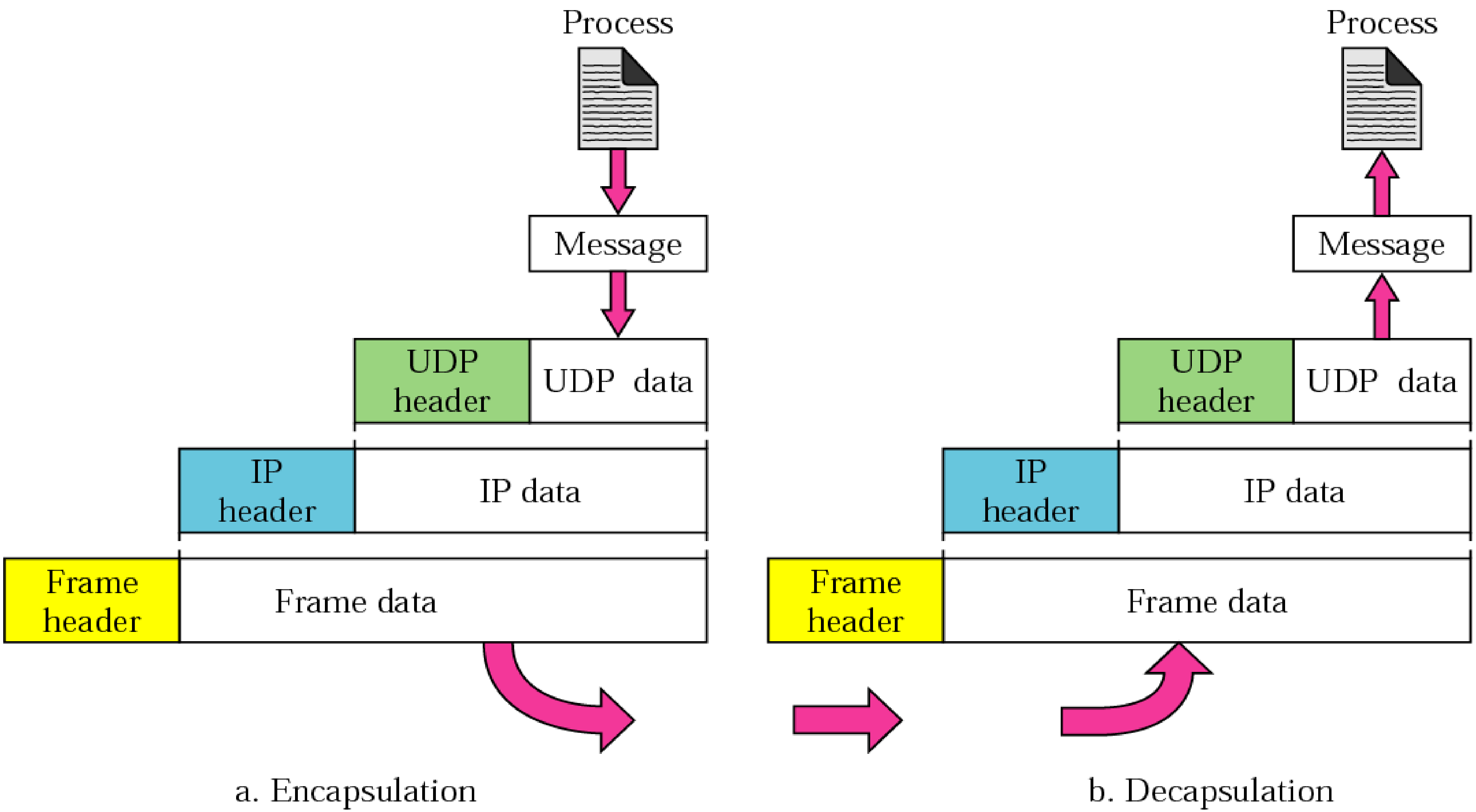
UDP 프로토콜에서 체크섬 계산에 사용되는 섹션

- Pseudoheader, UDP 헤더, 응용 계층으로부터 받은 데이터



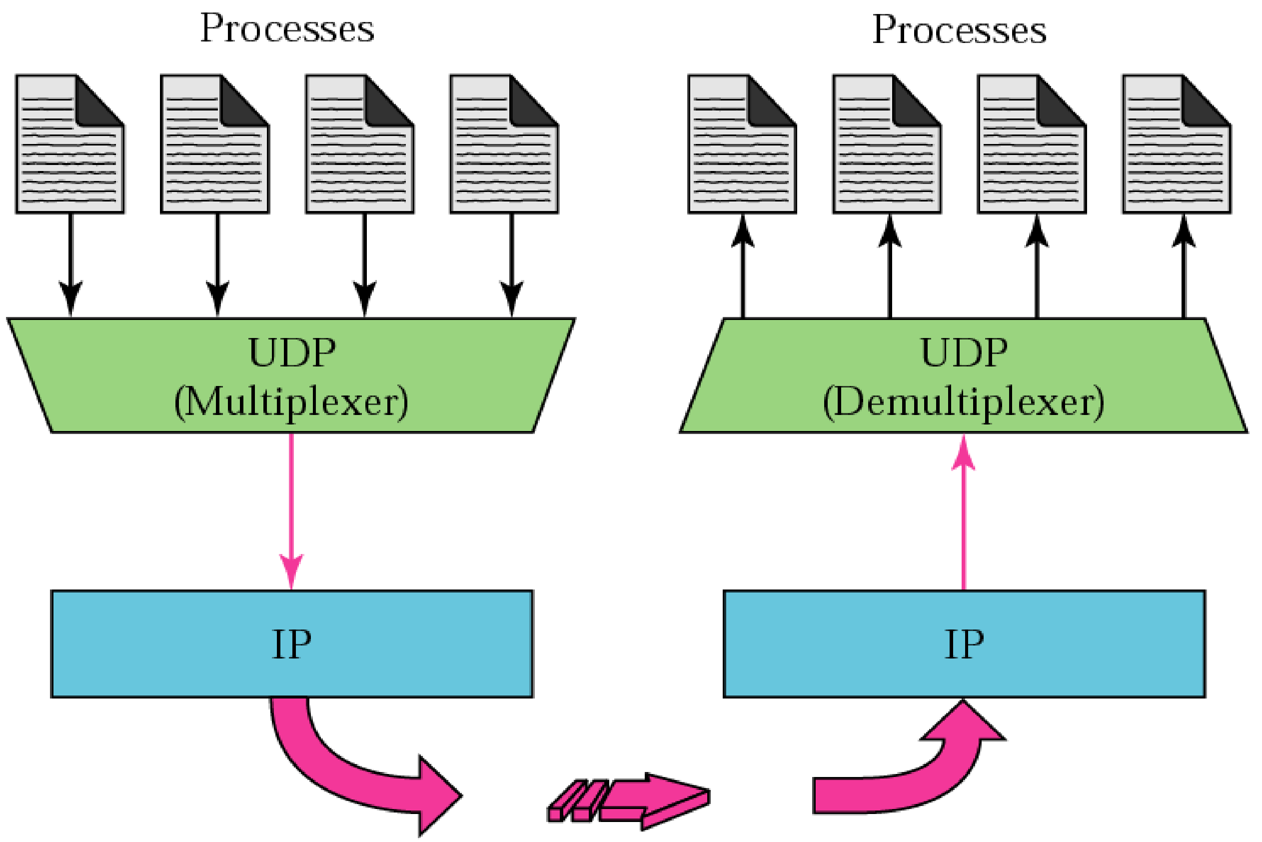
UDP 송수신 과정

Encapsulation and decapsulation

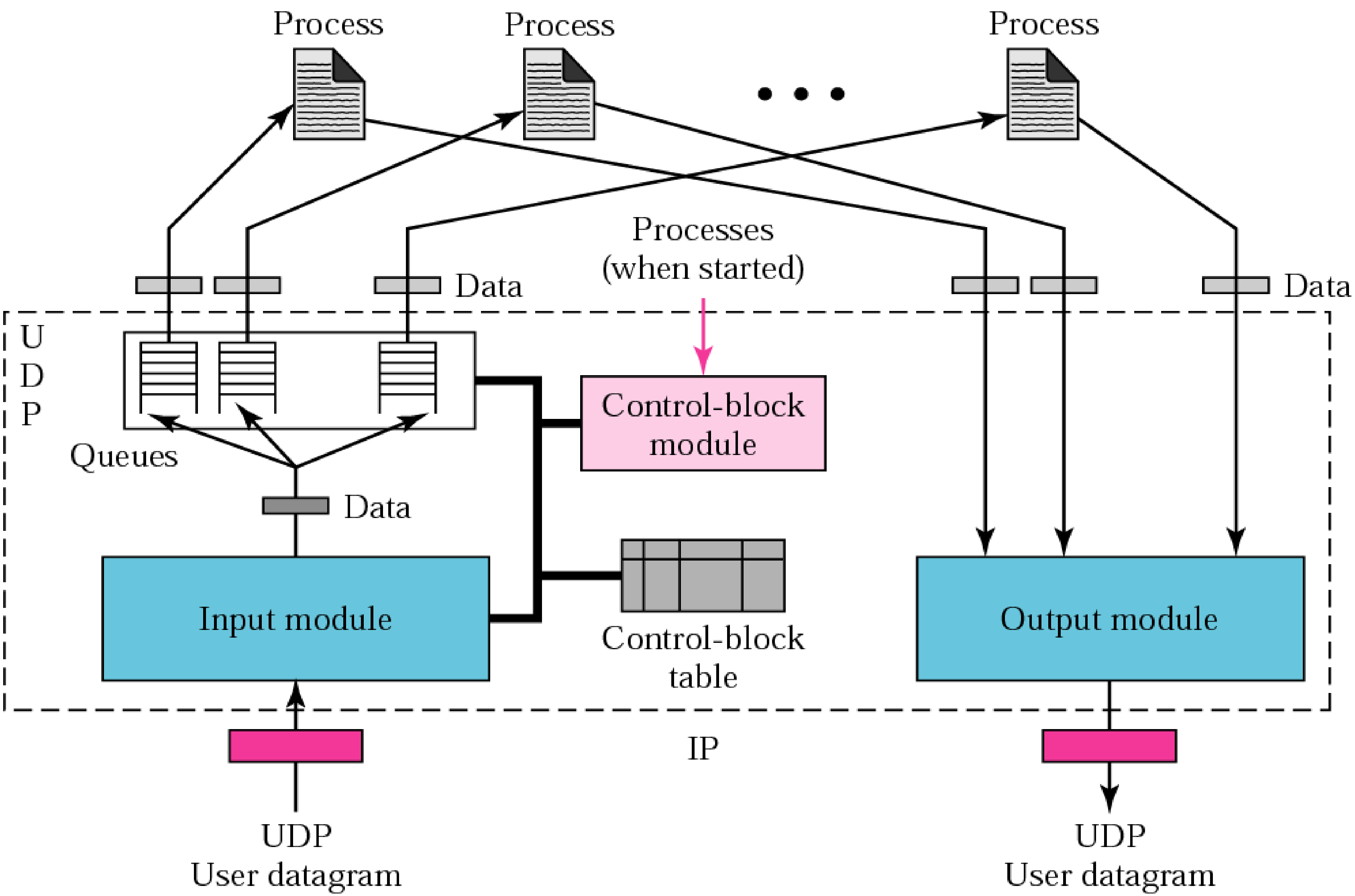


UDP 송수신 과정 (계속)

Multiplexing and demultiplexing



UDP 설계

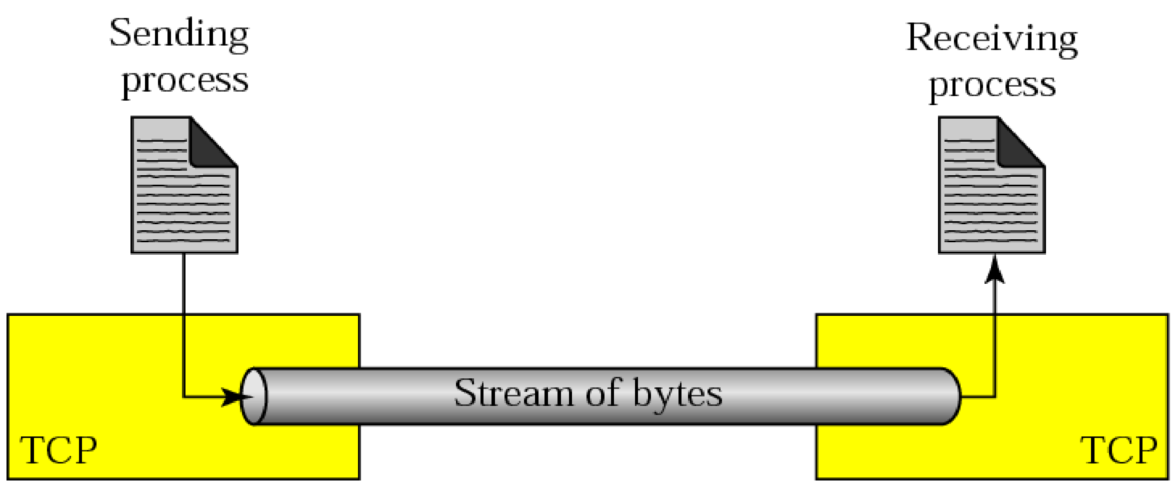


Process
Process
Process
Processes
(when started)
Data
Data
U
D
P
Queues
Data
Control-block
module
Input module
Control-block
table
Output module
IP
UDP
User datagram
UDP
User datagram

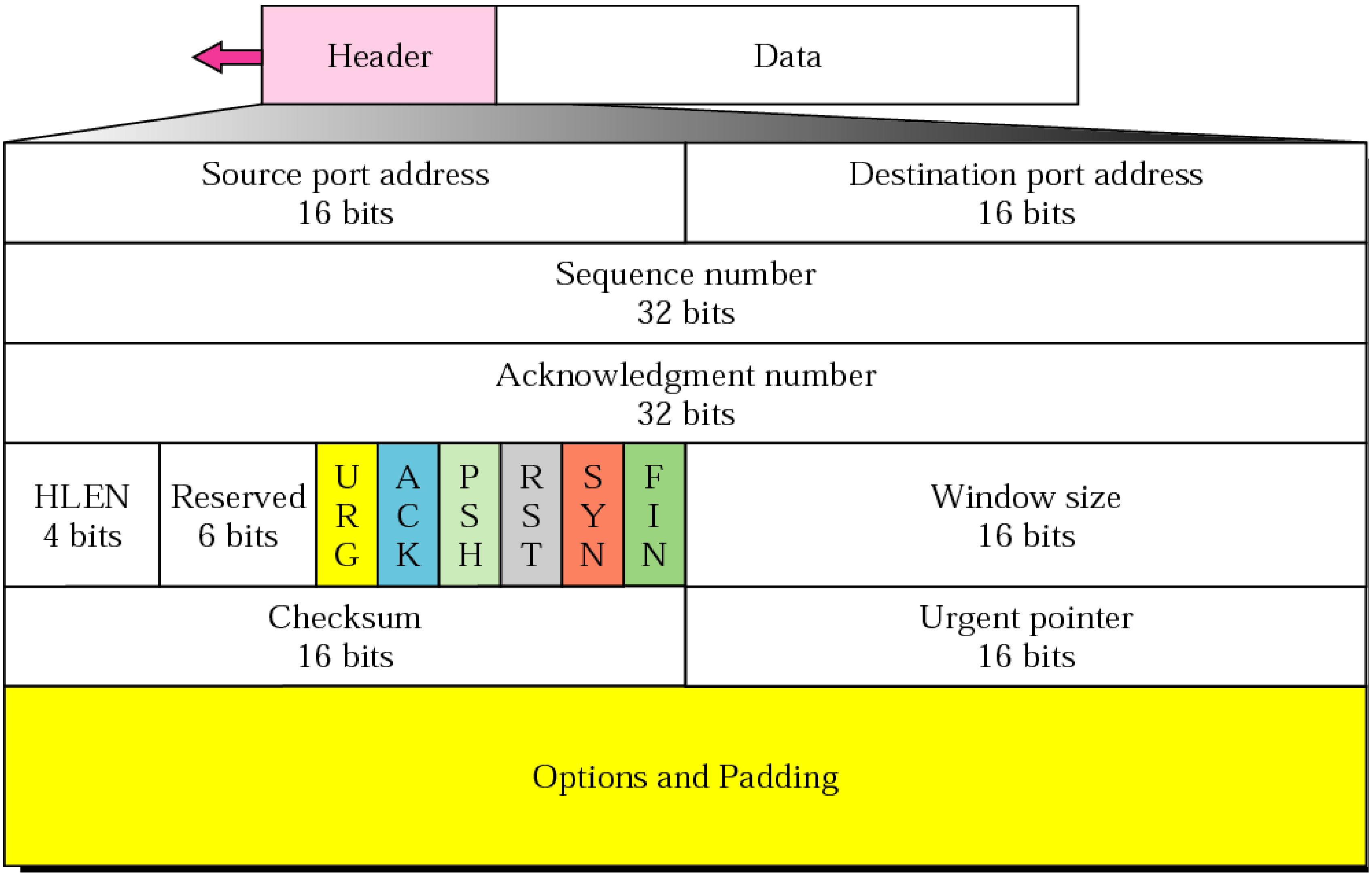
1.5 TCP 프로토콜 (Transmission Control Protocol)

특징

- 프로세스간 통신 지원
- 스트림 전송 서비스
- Full-duplex (전이중 쌍방향) 통신
- Connection-oriented 통신
  - Connection 설정, 데이터 송수신, Connection 해제
- 신뢰성 있는 통신

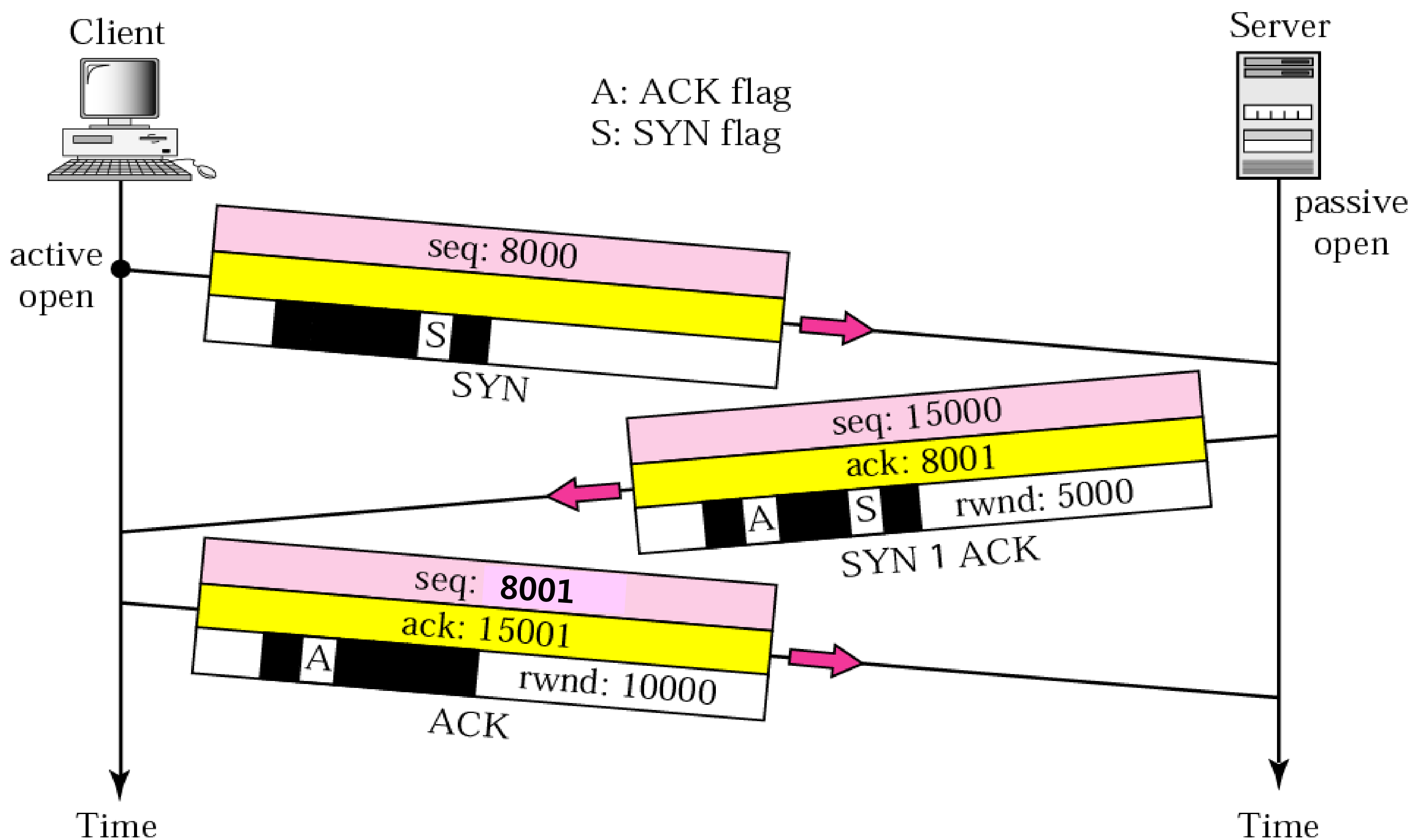


TCP 세그먼트 형식

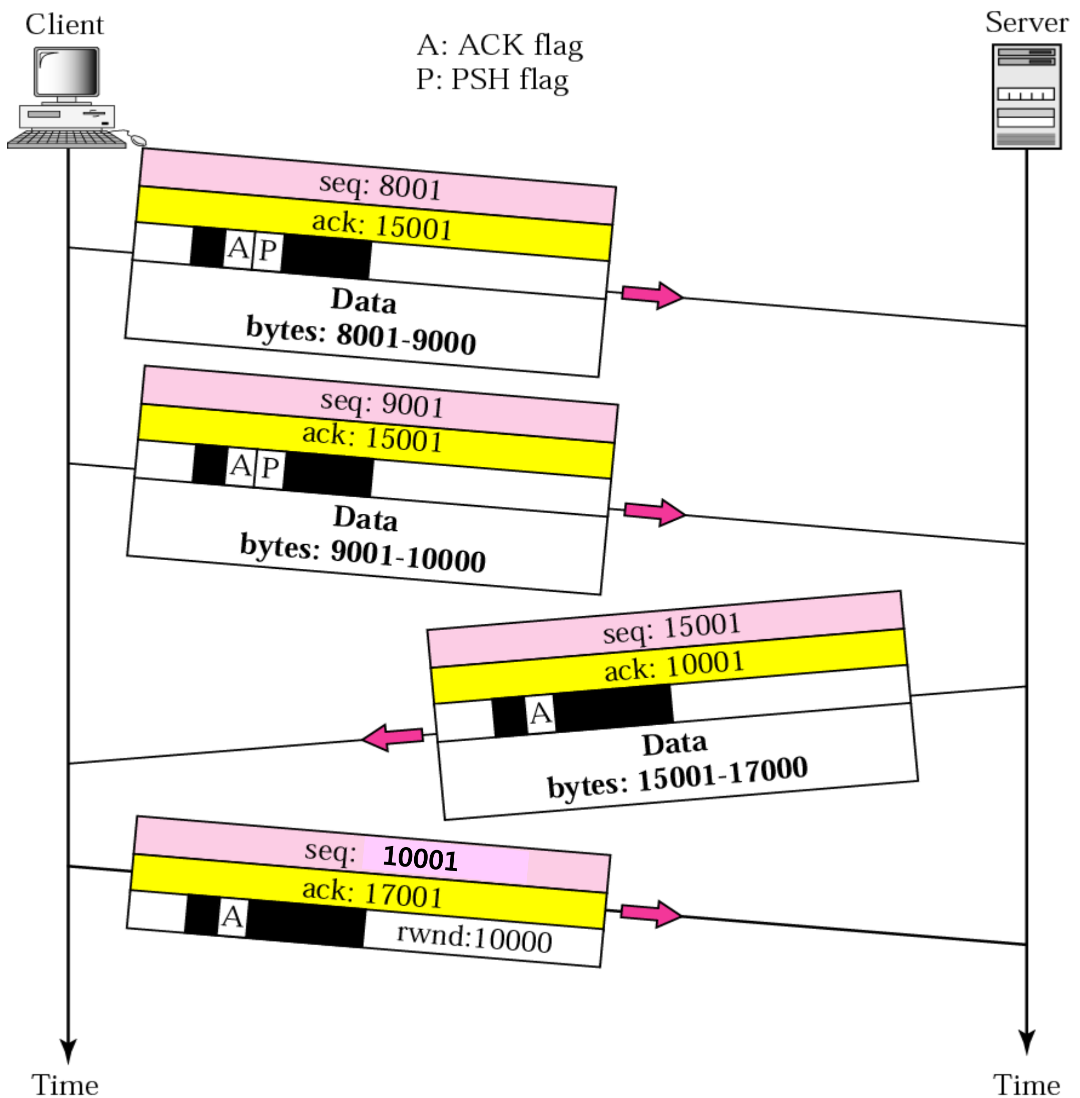

Header
Data
Source port address
16 bits
Destination port address
16 bits
Sequence number
32 bits
Acknowledgment number
32 bits
HLEN
4 bits
Reserved
6 bits
URG
ACK
PSH
RST
SYN
FIN
Window size
16 bits
Checksum
16 bits
Urgent pointer
16 bits
Options and Padding

Conection 설정

Three-way handshake

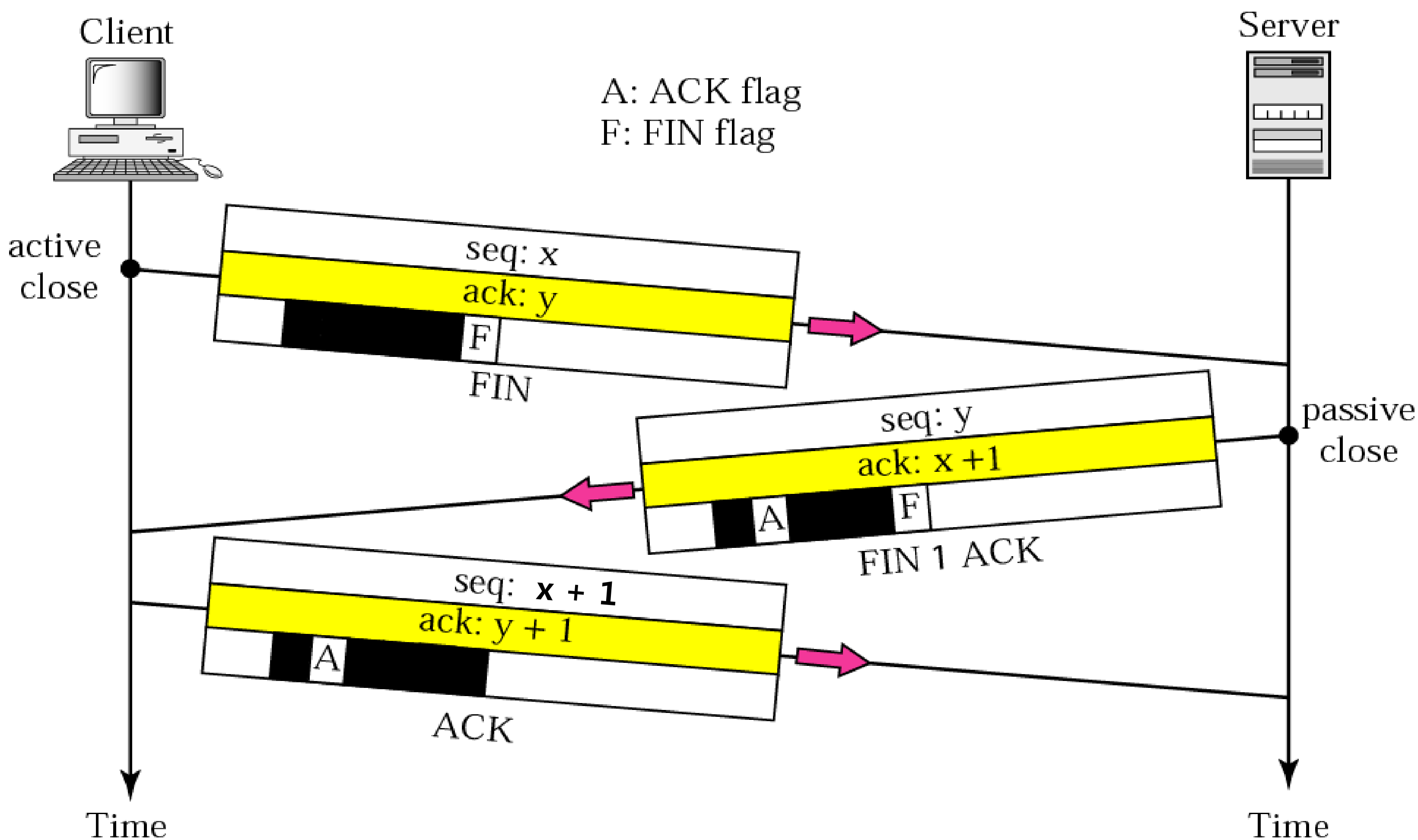


데이터 전송


Client
A: ACK flag
P: PSH flag
Server
seq: 8001
ack: 15001
A
P
Data
bytes: 8001-9000
seq: 9001
ack: 15001
A
P
Data
bytes: 9001-10000
seq: 15001
ack: 10001
A
Data
bytes: 15001-17000
seq: 10001
ack: 17001
A
rwnd:10000
Time
Time

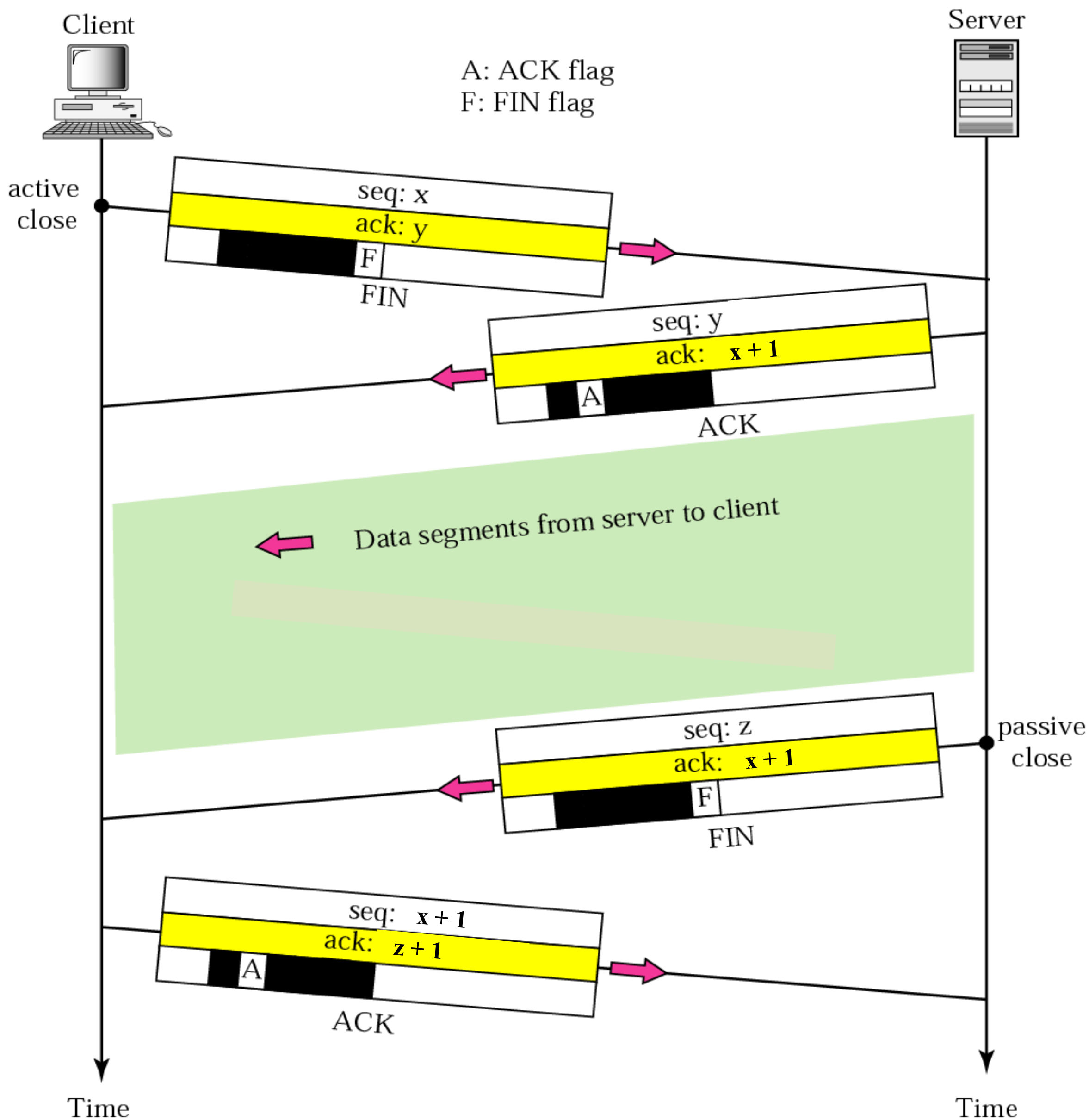
Connection 해제

Three-way handshake

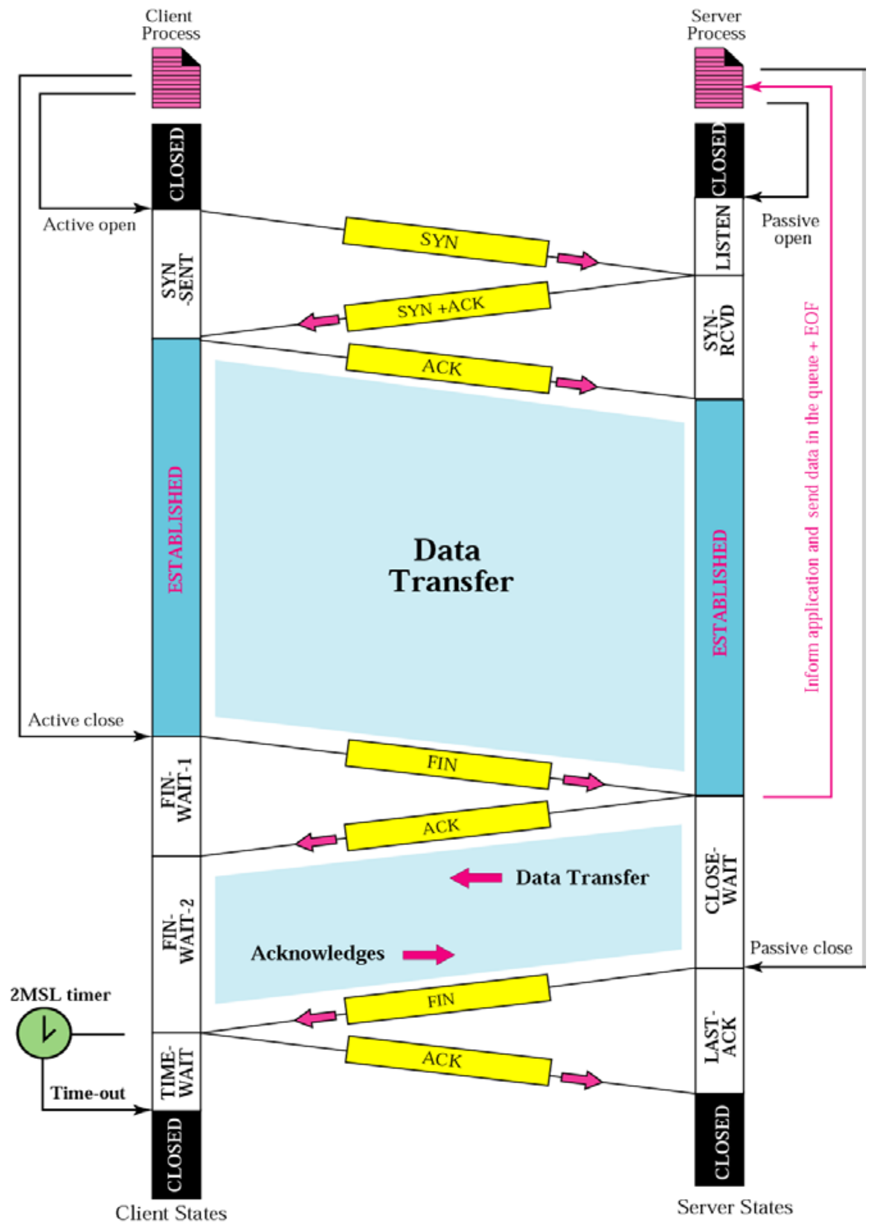


Connection 해제 (계속)

Half-close



TCP 통신 시나리오 정리



실습환경 구축

리눅스 설치

- 네트워크 환경 설정
  - 특히, 가상머신을 사용할 경우 외부 네트워크에서 접속가능한지 반드시 확인
- 모든 종류의 네트워크 보안 서비스 해제
  - 방화벽, SeLinux 등

예제 소스코드 다운로드

- UNIX Network Programming 예제 코드
  - http://www.unpbook.com/src.html

UNIX Network Programming 예제 코드 컴파일

소스코드 Daytime TCP 클라이언트 프로그램 컴파일 및 실행

- 03 – 05: 시스템 설정 확인 및 라이브러리 생성
- 07 – 08: 예제 소스코드 컴파일
- 10 – 11: INIT 서비스 중에서 Daytime 서버와 Echo 서버를 포함하도록 설정

```
01 # tar xvzf unpv13e.tar.gz
02 # cd unpv13e
03 # ./configure
04 # cd lib; make
05 # cd ../libfree; make

06 # cd ../intro
07 # make daytimetcpcli

08 # ntsysv
09 # service xinetd restart

10 # ./daytimetcpcli 127.0.0.1
11 20 JAN 2011 16:11:19 KST
```

2. 소켓 개요

목 차

2.1 Socket Address Structures

- IPv4 Socket Address Structure
- Generic Socket Address Structure
- New Generic Socket Address Structure
- Comparison of Socket Address Structures

2.2 Value-Resule Arguments

2.3 Byte Ordering Functions

- htons, htonl, ntohs, ntohl

2.4 Byte Manipulation Functions

- bzero, bcopy, bcmp
- memset, memcpy, memcmp

2.5 Address Transforming Functions

- inet_aton, inet_addr, inet_ntoa, inet_pton, inet_ntop

2.6 Socket I/O Functions

- readn, writen, readline

TCP 클라이언트 예제 코드:
TCP daytime client - [intro/daytimetcpcli.c]

```
#include  “unp.h”

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

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

    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(13);
```

TCP 클라이언트 예제 코드:
TCP daytime client - [intro/daytimetcpcli.c]

```
    if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr)
<= 0)
        err_quit("inet_pton error for %s", argv[1]);

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

    <이하 생략>
```

2.1 Socket Address Structures

소켓 주소 구조체 sockaddr_<unique suffix for each protocol>

- 대부분의 소켓 함수들은 인자값으로 소켓 주소 구조체에 대한 포인터값을 요구함

대표적인 소켓 주소 구조체

- 범용(Generic): sockaddr
- IPv4: sockaddr_in
- IPv6: sockaddr_in6
- Unix: sockaddr_un
- Datalink: sockaddr_dl
- New generic: sockaddr_storage

IPv4 Socket Address Structure

구조체 **sockaddr_in**은 **<netinet/in.h>**헤더파일에서 정의됨

```
typedef uint32_t in_addr_t;
struct in_addr
{
    in_addr_t s_addr;      /* 32-bit IPv4 address */
};

struct sockaddr_in
{
    uint8_t sin_len; /* length of structure (16 bytes) */
    sa_family sin_family;  /* AF_INET */
    in_port_t sin_port;    /* 16-bit TCP or UDP port */
                           /* network byte ordered */
    struct in_addr sin_addr;   /* 32-bit IPv4 address. */
                           /* network byte ordered */
    char sin_zero[8];             /* unused */
};
```

POSIX에 정의되어 있는 데이터타입

Datatype	Description	Header
int8_t	Signed 8-bit integer	<sys/types.h>
uint8_t	Unsigned 8-bit integer	<sys/types.h>
int16_t	Signed 16-bit integer	<sys/types.h>
uint16_t	Unsigned 16-bit integer	<sys/types.h>
int32_t	Signed 32-bit integer	<sys/types.h>
uint32_t	Unsigned 32-bit integer	<sys/types.h>
sa_family_t	Address family of socket address structure (nomally 8-bit)	<sys/socket.h>
socklen_t	Length of socket address structure, normally uint32_t	<sys/socket.h>
in_addr_t	IPv4 address, normally uint32_t	<netinet/in.h>
in_port_t	TCP or UDP port, normally uint16_t	<netinet/in.h>

Generic Socket Address Structure

소켓 함수들의 인자값으로써 다양한 종류의 소켓 주소 구조체에 대한 포인터 값들을 지시하는 방식 필요

- 현재의 ANSI C라면 간단하게 void *를 사용할 수 있음
- 1982년 당시의 해결책은 바로 범용 소켓 주소 구조체를 사용하는 것이었음

범용 소켓 주소 구조체 sockaddr은 <sys/socket.h>헤더에서 정의됨

```
struct sockaddr {
    uint8_t sa_len;

    /* address family: AF_xxx value */
    sa_family_t sa_family;

    /* protocol-specific address */
    char sa_data[14];
};
```

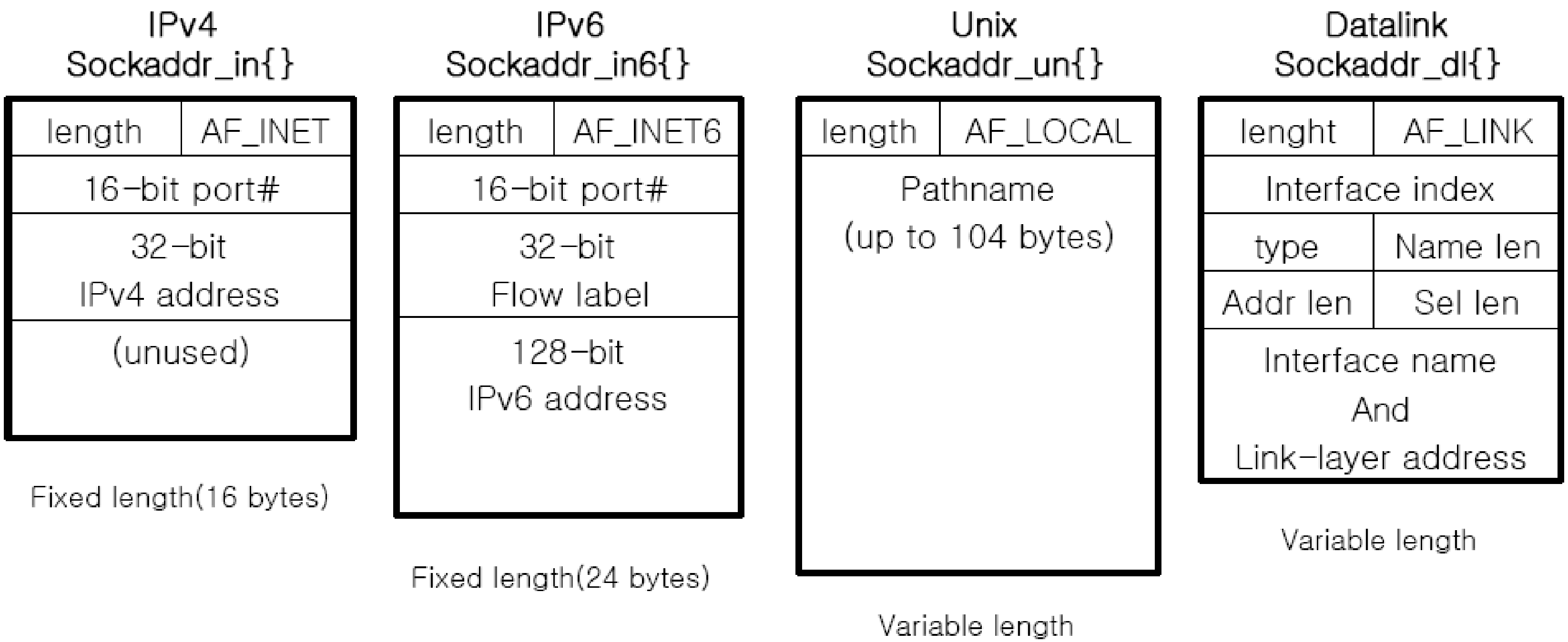
New Generic Socket Address Structure

구조체 sockaddr_storage는 <netinet/in.h>에서 정의됨

- 모든 소켓 주소 구조체들의 Alignment 요구사항들을 만족
- 모든 소켓 주소 구조체들을 포괄할 수 있는 충분한 용량 제공
  - 종전의 범용 소켓 주소 구조체의 크기는 16바이트인데, 크기가 28바이트인 IPv6 소켓 주소 구조체의 등장

```
struct sockaddr_storage {
   uint8_t sa_len;
   sa_family_t sa_family;
   /* address family: AF_xxx value */
   /* implementation-dependent elements to provide:
    * a) alignment sufficient to fulfill the alignment
    *    requirements of all socket adress types
    * b) enough storage to hold any type of socket addr
    */
};
```

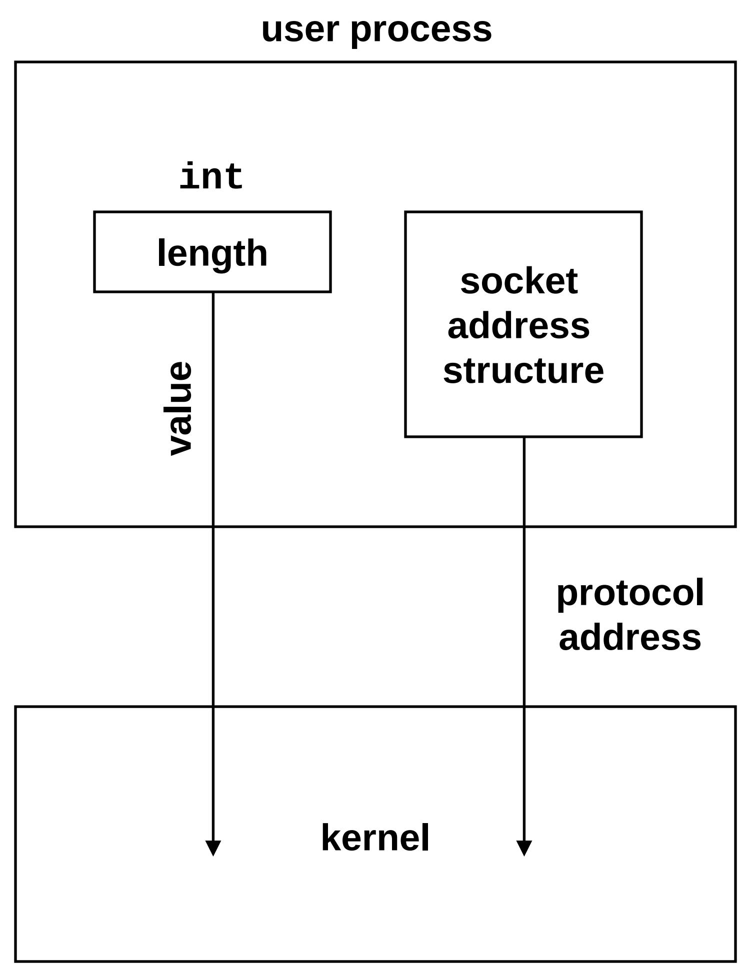
Comparison of Socket Address Structures



2.2 Value-Resule Arguments

함수 bind, connect, sendto는 소켓 주소 구조체에 대한 포인터를 인자 값으로 커널에 전달 (프로세스 → 커널)

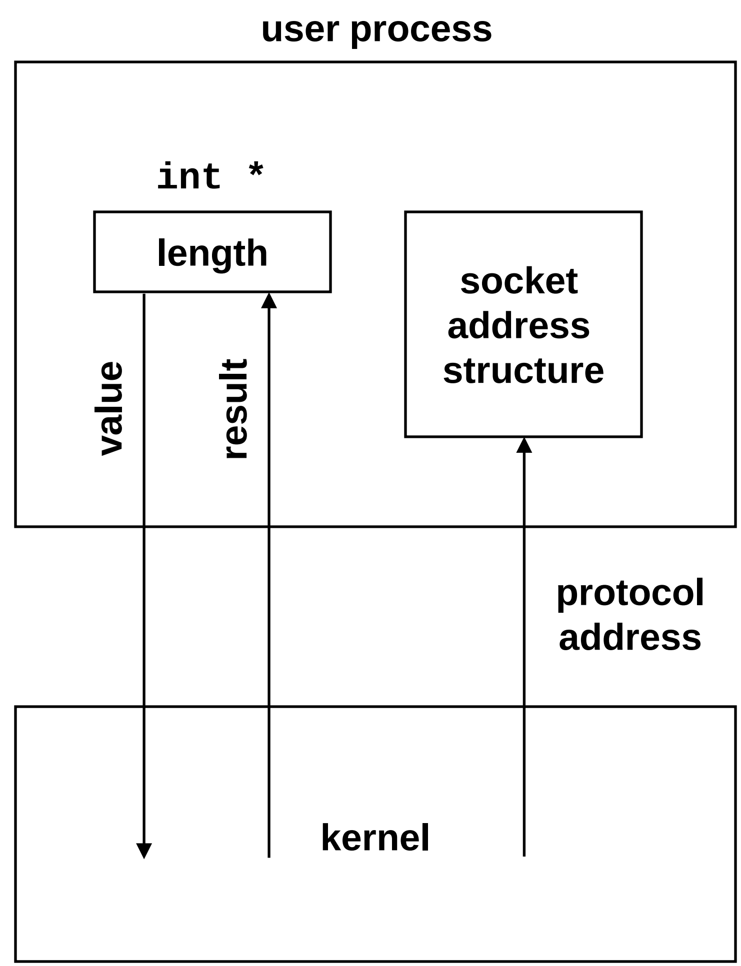
```
struct sockaddr_in serv;

/* fill in serv{} */
connect(sockfd, (SA *)
&serv, sizeof(serv));
```



Value-Result Arguments (계속)

함수 accept, recvfrom, getsockname, getpeername의 인자들은 함수의 결과값을 저장하는 용도로 사용됨 (커널 → 프로세스)

```
/* Unix domain */
struct sockaddr_un cli;
socklen_t len;

len = sizeof(cli);
getpeername(unixfd, (SA
*) &cli, &len);
// len may have changed
```

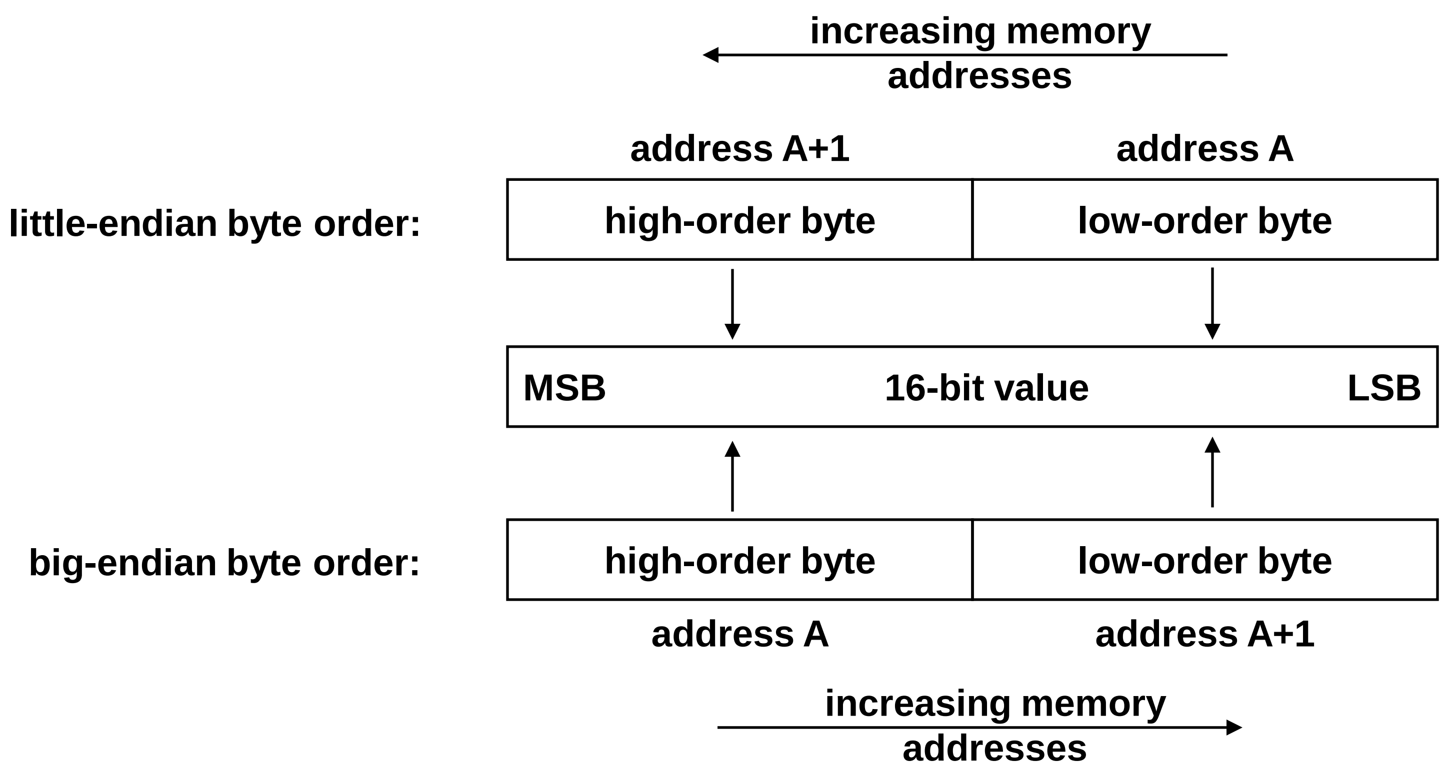


Byte Ordering: Big-endian vs. Little-endian

메모리에 다중 바이트로 구성된 값을 저장하는 방식

- Big-endian: 시작 주소에 높은 차수의 값 저장
- Little-endian: 시작 주소에 낮은 차수의 값 저장

16-bit integer에 대한 Little-endian과 Big-endian byte order 비교



호스트 시스템의 Byte ordering을 확인하는 예제 프로그램 - [intro/byteorder.c]

```
#include        "unp.h"

int
main(int argc, char **argv)
{
    union {
        short  s;
        char   c[sizeof(short)];
    } un;

    un.s = 0x0102;
    printf("%s: ", CPU_VENDOR_OS);
```

호스트 시스템의 Byte ordering을 확인하는 예제 프로그램 - [intro/byteorder.c]

```
    if (sizeof(short) == 2) {
            if (un.c[0] == 1 && un.c[1] == 2)
            printf("big-endian\n");
        else if (un.c[0] == 2 && un.c[1] == 1)
            printf("little-endian\n");
        else
                printf("unknown\n");
    } else
        printf("sizeof(short) = %d\n", sizeof(short));

    exit(0);
}
```

2.3 Byte Ordering Functions

시스템이 따라야 하는 Byte ordering 표준은 없음

- Host byte order: Little-endian 또는 Big-endian
  - x86 계열: Little-endian
  - sparc, powerpc 계열: Big-endian

네트워크 통신에서는 Big-endian을 사용

- Network byte order: Big-endian

2.3 Byte Ordering Functions

Host byte order와 Network byte order간 변환 함수 제공

```
#include <netinet/in.h>

uint16_t htons(uint16_t host16bitvalue);

uint32_t htonl(uint32_t host32bitvalue);

                                  Both return: value in network byte order

uint16_t ntohs(uint16_t net16bitvalue);

uint32_t ntohs(uint32_t net32bitvalue);

                                     Both return: value in host byte order
```

2.4 Byte Manipulation Functions

```
#include <strings.h>

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);
```

Returns: 0 if equal, nonzero if unequal

2.4 Byte Manipulation Functions

```
#include <string.h>

void *memset(void *dest, int c, size_t len);

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

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

Returns: 0 if equal, <0 or >0 if unequal

참고로, dest = src로 기억하면 쉬움

2.5 Address Transforming Functions - `inet_aton`, `inet_addr`, and `inet_ntoa`

ASCII 스트링과 Network byte ordered 바이너리 값 간의 변환

- 함수 이름에서 'a'는 ASCII, 'n'은 바이너리 값을 의미함

```
#include <arpa/inet.h>

int inet_aton(const char *strptr, sruct in_addr
*addraptr);
```

Returns: 1 if string was valid, 0 on error

```
in_addr_t inet_addr(const char *strptr);
```

에러값이 IP주소값 255.255.255.255을 의미함 → 사용X

Returns: 32-bit binary network byte ordered IPv4 address; INADDR_NONE if error

Not a pointer

```
char *inet_ntoa(struct in_addr inaddr);
```

Not reentrant

Returns: pointer to dotted-decimal string

inet_pton and inet_ntop Fuctions

스트링 값과 Network byte ordered 바이너리 값 간의 변환

- 함수 이름에서 'p'는 presentation, 'n'은 바이너리 값을 의미함

```
#include <arpa/inet.h>

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

Returns: 1 if OK, 0 if iput not a valid presentation format, -1 on error

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

Returns: pointer to result if OK, NULL on error

Can not be a NULL pointer

inet_pton and inet_ntop Fuctions

함수 사용 예

```
foo.sin_addr.s_addr = inet_addr(cp);
→ inet_pton(AF_INET, cp, &foo.sin_addr);

ptr = inet_ntoa(foo.sin_addr);
→ char str[16];
→ ptr = inet_ntop(AF_INET, &foo.sin_addr, str,
sizeof(str));
```

2.6 Socket I / O Functions - `readn`, `writen`, and `readline` Fuctions

스트림 소켓에 대해서 read와 write 함수를 호출할 경우 파일에 대한 접근과 다른 방식의 처리 필요

- 쓰기나 읽기를 요청한 바이트 수보다 더 적은 바이트를 쓰거나 읽어오더라도 에러 상황이 아닐 수 있음
- 커널에서 관리되는 소켓의 송/수신 버퍼 크기 제한 때문에 발생 가능한 문제
- 이 경우 단순히 read나 write 함수를 한번 더 호출하면 문제 해결

2.6 Socket I / O Functions - `readn`, `writen`, and `readline` Fuctions

자체적으로 정의한 라이브러리 함수들
(점선 테두리로 표시)

```
#include "unp.h"

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);
```

All return: number of bytes read or written, -1 on error

readn 함수:
Read n bytes from descriptor – [lib/readn.c]

```
#include  "unp.h"

ssize_t                          /* Read "n" bytes from a
descriptor. */
readn(int fd, void *vptr, size_t n)
{
    size_t nleft;
    ssize_t   nread;
    char   *ptr;

    ptr = vptr;
    nleft = n;
```

읽어들인 바이트 수가 요청된 바이트 수
보다 적으면 다시 read 함수 호출

readn 함수:
Read n bytes from descriptor – [lib/readn.c]

```
    while (nleft > 0) {
        if ( (nread = read(fd, ptr, nleft)) < 0) {
            if (errno == EINTR)
                nread = 0;    /* and call read() again */
            else
                return(-1);
        } else if (nread == 0)
            break;                /* EOF */

        nleft -= nread;
        ptr   += nread;
    }
    return(n - nleft);            /* return >= 0 */
}
```

시스템 콜 수행시 인터럽트가 발생하면
EINTR 에러를 반환할 수 있다

writen 함수:
Write n bytes to a descriptor – [lib/writen.c]

```
#include  "unp.h"

ssize_t                          /* Write "n" bytes to a
descriptor. */
writen(int fd, const void *vptr, size_t n)
{
    size_t    nleft;
    ssize_t       nwritten;
    const char*ptr;

    ptr = vptr;
    nleft = n;
```

writen 함수:
Write n bytes to a descriptor – [lib / writen.c]

쓰여진 바이트 수가 요청된 바이트 수보
다 적으면 다시 write 함수 호출

```
    while (nleft > 0) {
        if ( (nwritten = write(fd, ptr, nleft)) <= 0) {
            if (nwritten < 0 && errno == EINTR)
                nwritten = 0;  /* and call write() again */
            else
                return(-1);         /* error */
        }

        nleft -= nwritten;
        ptr   += nwritten;
    }
    return(n);
}
```

readline 함수 (one byte at a time): Read a text line from descriptor – [test/readline1.c]

소켓에서부터 한 줄 단위로 읽기를 원할 경우

- 표준 입출력(stdio)을 선호할 수 있으나, 이는 위험성을 내포함
  - Stdio는 내부적으로 버퍼링 기법을 사용하는데 해당 버퍼를 제어할 수 없음
  - 특히 select 같은 함수와 stdio 함수들을 같이 사용할 경우 버그 발생 가능성 높음

```
/* PAINFULLY SLOW VERSION -- example only */
ssize_t
readline(int fd, void *vptr, size_t maxlen)
{
    ssize_t   n, rc;
    char   c, *ptr;
    ptr = vptr;
```

readline 함수 (one byte at a time): Read a text line from descriptor – [test/readline1.c]

```c
    for (n = 1; n < maxlen; n++) {
again:
        if ( (rc = read(fd, &c, 1)) == 1) {
            *ptr++ = c;
            if (c == '\n')
                break;/* newline is stored, like fgets() */
        } else if (rc == 0) {
            *ptr = 0;
            return(n - 1);/* EOF, n - 1 bytes were read */
        } else {
            if (errno == EINTR)
                goto again;
            return(-1);   /* error, errno set by read() */
        }
    }
    *ptr = 0; /* null terminate like fgets() */
    return(n);
}
```

요청된 바이트 수많큼 루프를 돌면서
한 바이트씩 read 수행

readline 함수:
Better version – [lib/readline.c]

자신만의 버퍼를 가진 readline 함수 구현

- 버퍼를 제어할 수 있어서 위험성 감소
  - select 같은 함수들은 해당 버퍼의 존재를 알 수 없기 때문에 사용에 유의해야 함
  - 이미 읽어들여서 버퍼링된 내용에 대해서 select 함수가 계속해서 Block될 가능성이 있음

```
#include  "unp.h"

static intread_cnt;
static char   *read_ptr;
static char   read_buf[MAXLINE];
```

readline 함수:
Better version – [lib / readline.c]

```
static ssize_t
my_read(int fd, char *ptr)
{

    if (read_cnt <= 0) {
again:
        if ( (read_cnt = read(fd, read_buf, sizeof(read_buf))) < 0) {
            if (errno == EINTR)
                goto again;
            return(-1);
        } else if (read_cnt == 0)
            return(0);
        read_ptr = read_buf;
    }

    read_cnt--;
    *ptr = *read_ptr++;
    return(1);
}
```

버퍼를 할당하고 소켓으로부터 버퍼의
크기만큼 데이터를 읽어서 버퍼링함

매번 한 바이트씩 read 함수를 호출하는 대신
버퍼링해 둔 데이터에서 한 바이트씩 반환해줌

readline 함수: Better version – [lib/readline.c] (계속)

```
ssize_t
readline(int fd, void *vptr, size_t maxlen)
{
    ssize_t   n, rc;
    char   c, *ptr;

    ptr = vptr;
    for (n = 1; n < maxlen; n++) {
        if ( (rc = my_read(fd, &c)) == 1) {
            *ptr++ = c;
            if (c == '\n')
                break; /* newline is stored, like fgets()
*/
```

Read 함수 대신 my_read 함수를 호출해서 매번 한 바이트씩 읽어옴

readline 함수:
Better version – [lib/readline.c]   (계속)

```
        } else if (rc == 0) {
            *ptr = 0;
            return(n - 1);/* EOF, n - 1 bytes were read */
        } else
            return(-1);   /* error, errno set by read() */
    }

    *ptr = 0; /* null terminate like fgets() */
    return(n);
}

ssize_t
readlinebuf(void **vptrptr)
{
    if (read_cnt)
        *vptrptr = read_ptr;
    return(read_cnt);
}
```

내부 버퍼를 외부에서 접근 가능하도록
해당 버퍼에 대한 포인터 값을 반환함

TCP daytime client 분석 - [intro/daytimetcpcli.c]

```
#include  "unp.h"

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

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

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

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

TCP daytime client 분석 - [intro/daytimetcpcli.c]

```
    if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr) <= 0)
        err_quit("inet_pton error for %s", argv[1]);

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

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

    exit(0);
}
```

실습 과제

이전 슬라이드의 TCP daytime client 프로그램을 수정해서 단순한 Web client 프로그램 작성

- Line 15: 포트번호 80으로 접속하도록 수정
- Line 20 ~ 26: 웹 서버에 접속하여 “index.html” 파일을 내려 받아서 화면에 출력하도록 수정
  - writen을 사용해서 스트링 “GET / HTTP/1.0\r\n\r\n”를 서버에 전송
  - readn을 사용해서 서버로 부터 스트링을 수신해서 화면에 출력

3. 기본적인 TCP 소켓 API

목 차

3.1 Socket Functions

- socket, connect, bin, listen, accept, close

3.2 Get Socket Name Functions

- getsockname, getpeername

3.3 Name and Address Conversions

- Functions: gethostbyname, gethostbyaddr, getservbyname, getservbyport

3.4 Concurrent Servers

- Functions: fork, exec

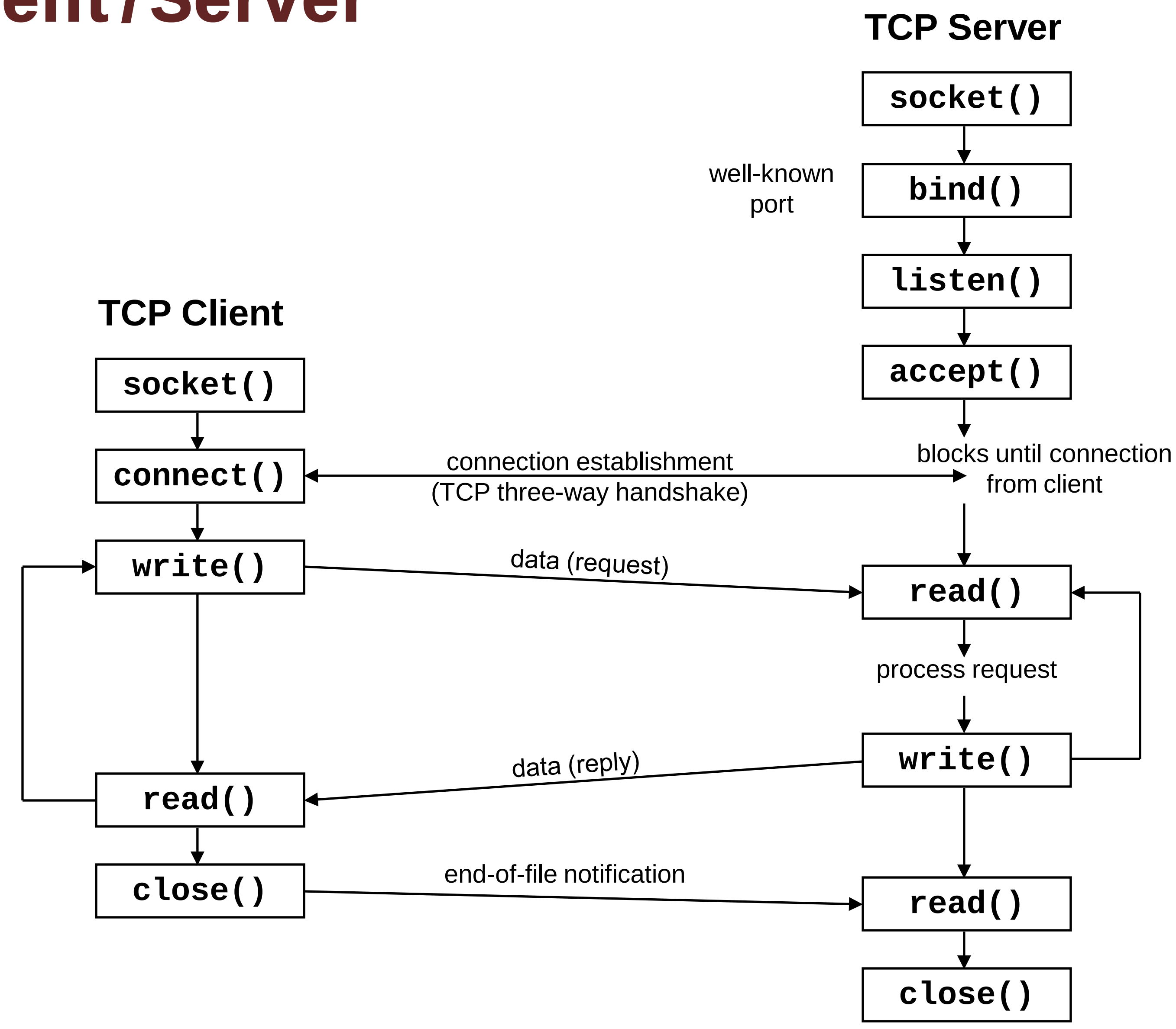
TCP 서버 예제 코드:
TCP daytime server - [intro/daytimetcpsrv.c]

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

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

    listenfd = Socket(AF_INET, SOCK_STREAM, 0);

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

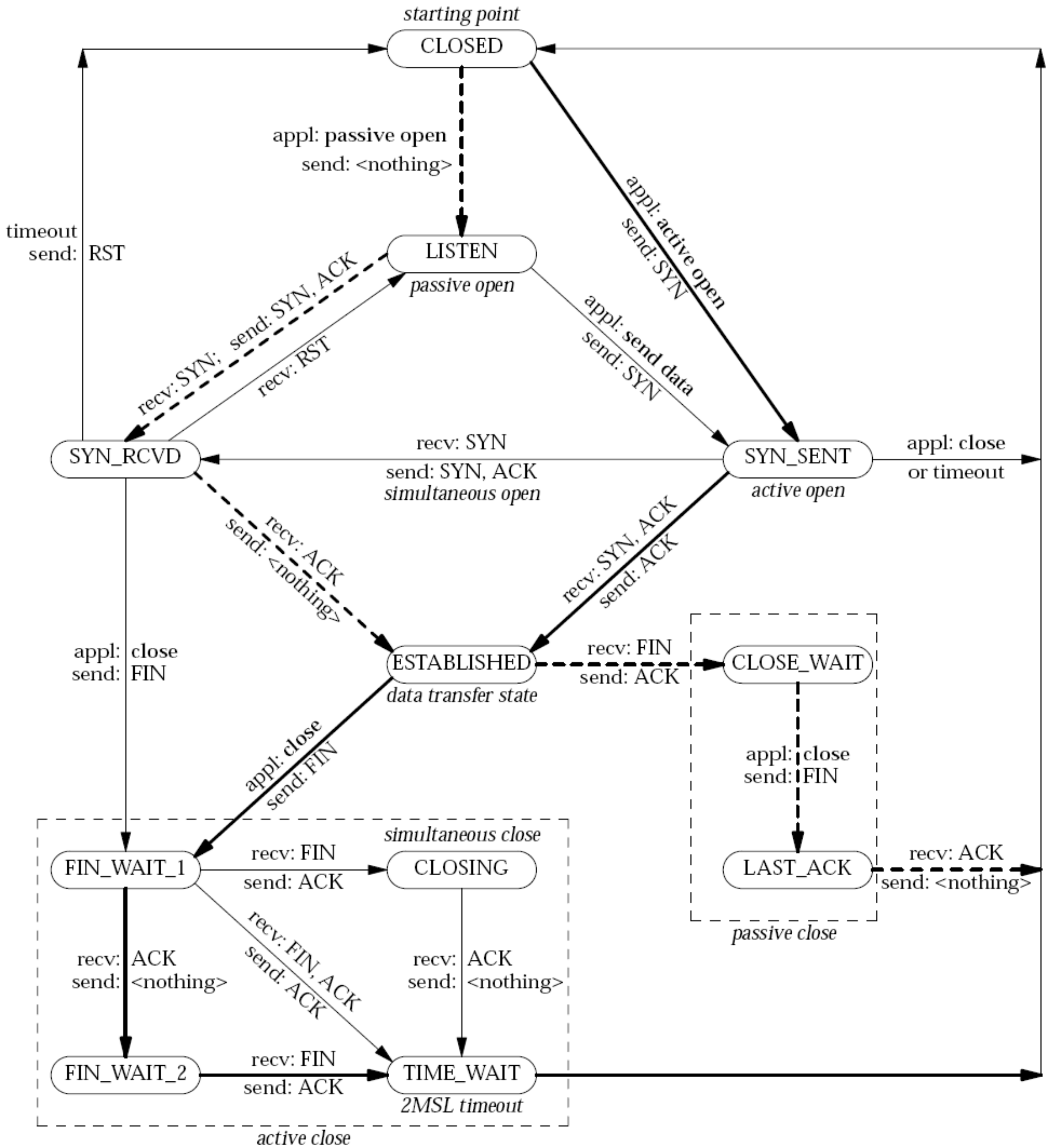
TCP 서버 예제 코드:
TCP daytime server - [intro/daytimetcpsrv.c]

```
    Bind(listenfd, (SA *) &servaddr, sizeof(servaddr));

    Listen(listenfd, LISTENQ);

    for ( ; ; ) {
        connfd = Accept(listenfd, (SA *) NULL, NULL);

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

        Close(connfd);
    }
}
```

Socket Functions for Elementary TCP Client/Server



TCP State Transition Diagram



3.1 Socket Functions

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

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

Returns: non-negative descriptor if OK, -1 on error

socket Function

서버/클라이언트간 통신에서 사용되는 소켓 구조체를 생성

반환값: socket descriptor (또는 sockfd)는 file descriptor처럼 취급

인자값: 사용하고자 하는 프로토콜 설정

- int family (AF_xxx 또는 PF_xxx)
  - AF_INET: IPv4 protocols
  - AF_INET6: IPv6 protocols
  - AF_LOCAL: Unix domain protocols
  - AF_ROUTE: Routing sockets
  - AF_KEY: Key socket
- int type (SOCK_xxx)
  - SOCK_STREAM: stream socket
  - SOCK_DGRAM: datagram socket
  - SOCK_SEQPACKET: sequenced packet socket
  - SOCK_RAW: raw socket

socket Function (계속)

인자값

- int protocol (IPPROTO_xxx)
  - IPPROTO_TCP: TCP transport protocol
  - IPPROTO_UDP: UDP transport protocol
  - IPPROTO_SCTP: SCTP transport protocol
- protocol에 '0'을 넘겨주면 family와 type에 근거하여 디폴트 값 선택

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

connect Function

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

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

Returns: 0 if OK, -1 on error

connect Function (계속)

클라이언트가 서버와 Connection 설정

인자값: 자신의 소켓 정보와 연결하고자 하는 서버 주소 정보 포함

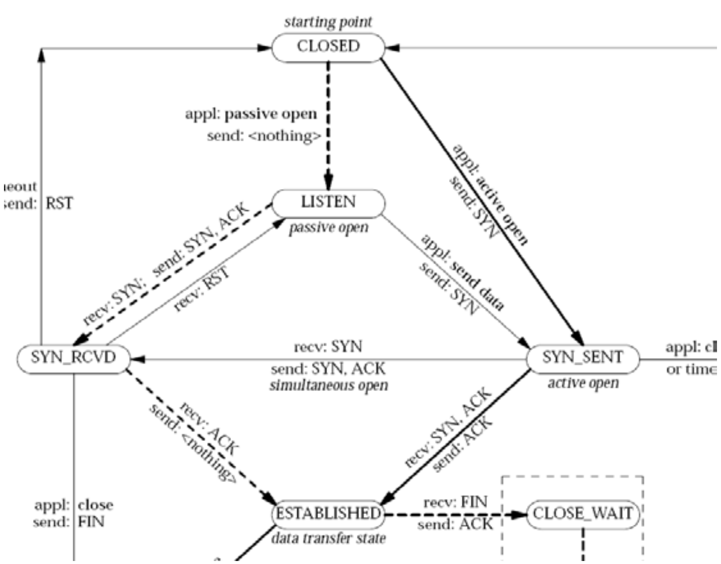
- int sockfd: 함수 socket의 반환값인 socket descriptor 값
- const struct sockaddr *servaddr: 서버의 IP주소와 포트번호 정보 포함
  - IPv4 소켓 주소 구조체 포인터를 넘겨주려면 Type casting을 해주어야 Warning을 피할 수 있다
- socklen_t addrlen: 소켓 주소 구조체의 실제 크기

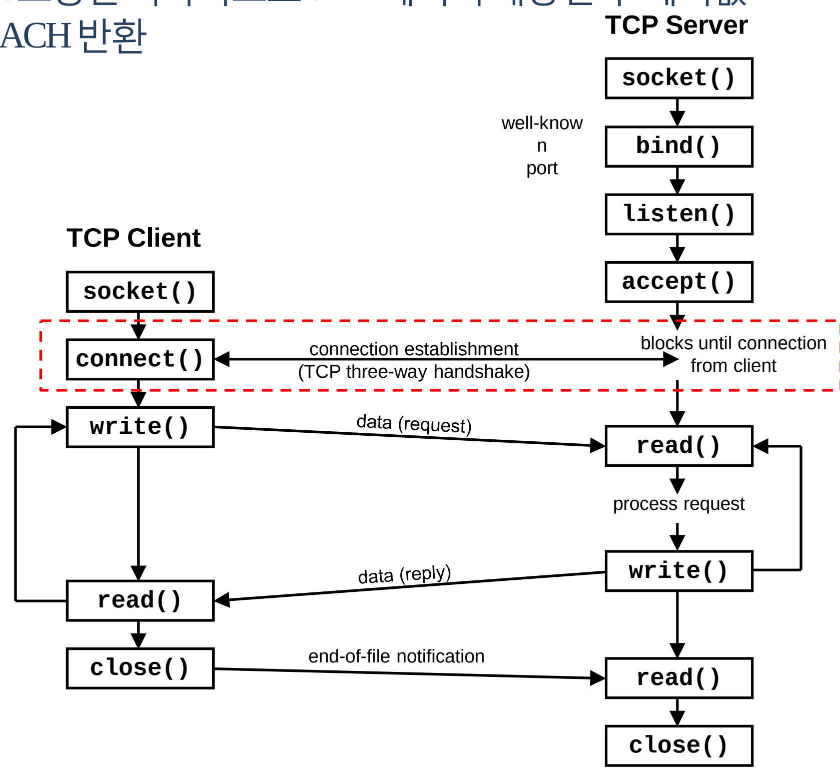
```
struct in_addr
{
    in_addr_t s_addr;
};

struct sockaddr_in
{
    uint8_t sin_len;
    sa_family sin_family;
    in_port_t sin_port;
    struct in_addr sin_addr;
    char sin_zero[8];
};
```

connect Function (계속)

connect 함수를 호출하면 TCP three-way handshake 과정을 수행

- 클라이언트는 여러가지 종류의 에러 상황에 놓일 수 있음
  - SYN 세그먼트에 대한 응답을 받지 못한 경우: 4.4BSD의 경우 총 75초를 기다린 후 에러값 반환
  - SYN에 대한 응답으로 RST를 받는 경우 (hard error): 서버 프로세스가 없는 경우로써 즉시 에러값 ECONNREFUSED 반환
  - 서버로 부터의 응답 대신 ICMP “destination unreachable” 메시지를 수신한 경우 (soft error): 일시적인 네트워크 장애 상황으로 판단하고 75초동안 지속적으로 SYN 메시지 재송신 후 에러값 EHOSTUNREACH 또는 ENETUNREACH 반환





bind Function

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

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

Returns: 0 if OK, -1 on error

bind Function

서버가 자신의 소켓에 IP주소와 포트번호 정보를 지정

- 클라이언트는 보통 bind를 수행하지 않음
  - 소켓이 연결될 때, 커널이 임의의 포트번호와 송신 IP 주소를 소켓에 지정
- bind 함수의 가장 일반적인 에러 반환값은 EADDRINUSE
  - 포트번호가 이미 사용중인 경우 발생

인자값: 자신의 주소 정보 포함

- int sockfd: 함수 socket의 반환값인 socket descriptor 값
- const struct sockaddr *myaddr
  - 서버의 포트번호는 보통 잘 알려진 값을 지정함으로써 클라이언트가 해당 포트로 접속할 수 있도록 허용
  - 만약 포트번호로 '0'을 지정하면 커널이 임의의 값을 지정
- socklen_t addrlen: 소켓 주소 구조체의 실제 크기

Wildcard 주소

소켓 주소 구조체의 IP 주소에 Wildcard 주소 사용 가능

- IPv4: INADDR_ANY (보통 0으로 정의됨)
- IPv6: in6addr_any 변수 (헤더파일 netlinet/in.h에 정의)

Wildcard 주소를 사용했을 경우, 임의로 배정된 주소값 확인 방법

- getsockname 함수 호출

Process specifies		**Result**
IP address	**port**	
Wildcard	0	Kernel chooses IP adress and port
Wildcard	non-zero	Kernel chooses IP address, process specifies port
Local IP address	0	Process specifies IP address, kernel chooses port
Local IP address	non-zero	Process specifies IP address and port

listen Function

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

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

Returns: 0 if OK, -1 on error

listen Function

TCP 서버가 호출하는 함수

- 연결되지 않은 소켓을 “passive socket”으로 지정
  - 커널로 하여금 listen을 수행한 소켓으로 들어오는 연결 요청을 수락하도록 지시

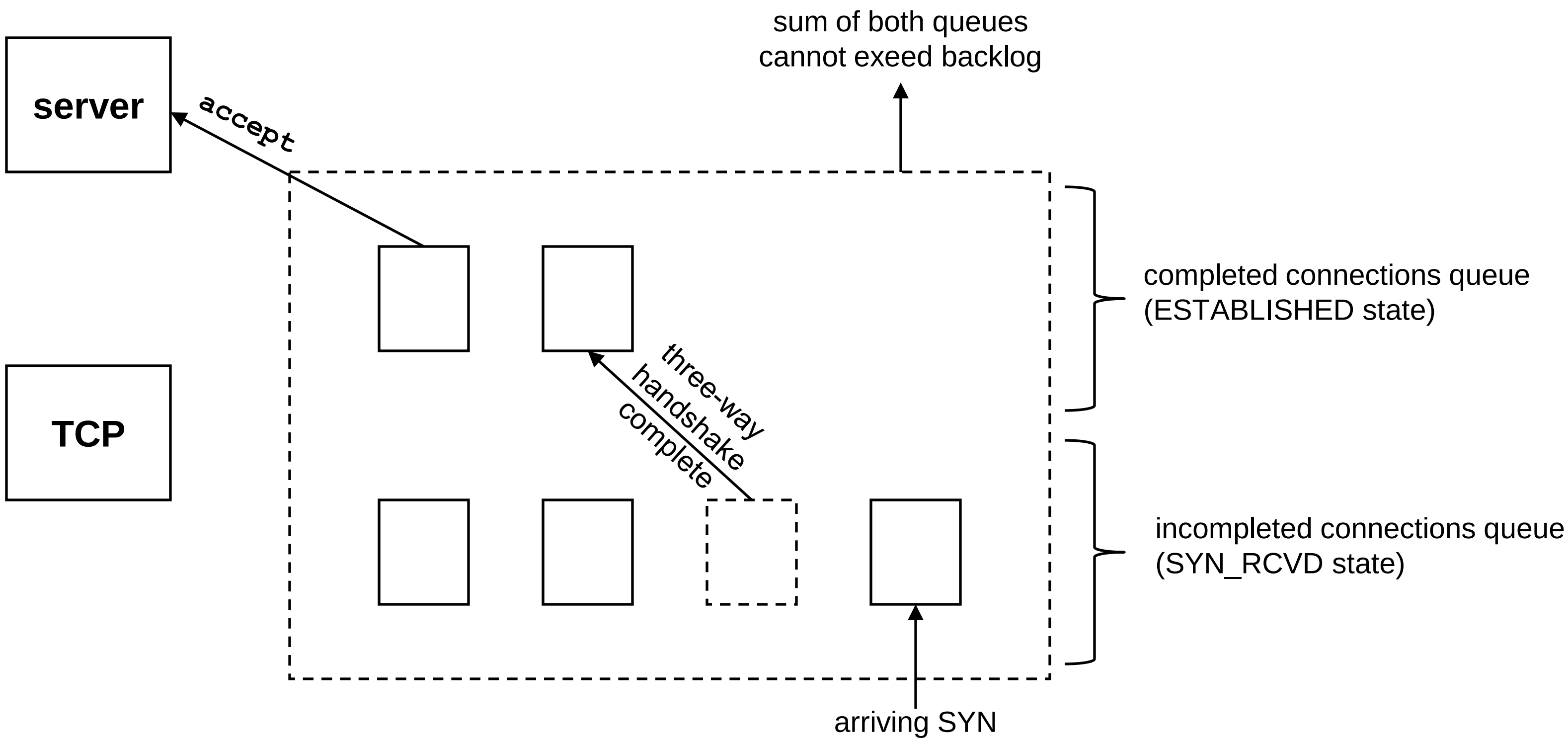
인자값

- int sockfd: 함수 socket의 반환값인 socket descriptor 값
- int backlog: 수용하고자 하는 Connection의 최대 개수

Two Queues for a Listening Socket

커널은 listen을 수행한 소켓에 대해서 두개의 Queue를 관리함

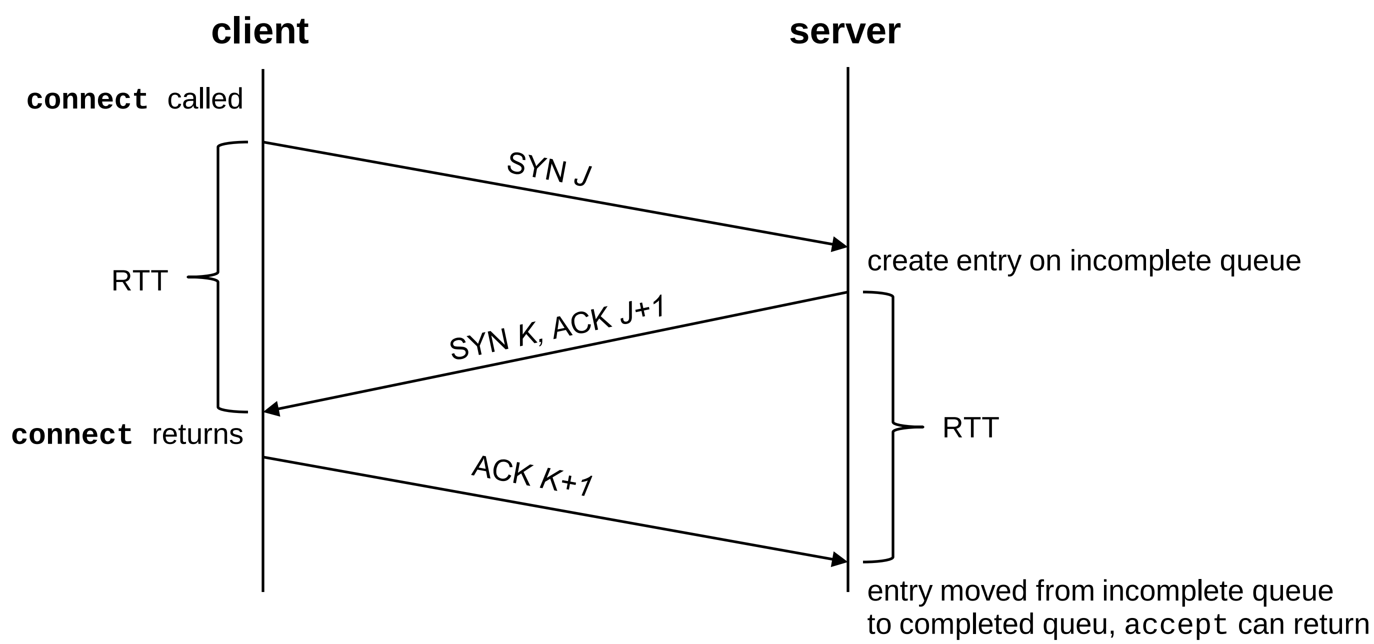
- Incomplete connection queue
  - SYN만 받은 SYN_RCVD 상태의 소켓들을 포함
- Complete connection queue
  - 3-way handshake를 완료한 ESTABLISHTED 상태의 소켓들을 포함



TCP three-way handshake and the two queues for a listening socket

서버의 listen 소켓이 SYN 패킷을 수신하면 커널은 새로운 소켓 생성

- listen 소켓은 실제로 클라이언트와 연결되지 않음
- 3-way handshake가 완료되면 서버가 accept를 수행하는 것과 상관없이 클라이언트와의 연결은 완료됨
- TCP 서버는 각 클라이언트에 대해서 하나의 소켓을 생성해서 연결함



accept Function

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

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

Returns: non-negative descriptor if OK, -1 on error

accept Function

TCP 서버가 listen 함수를 호출하고 난 다음 호출하는 함수

- Connection이 완료된 소켓 정보를 반환
  - 커널의 completed connection queue 맨 앞에 있는 새로운 소켓에 대한 socket descriptor를 반환

인자값 (혹은 반환값)

- int sockfd: 현재 listen하고 있는 passive socket에 대한 socket descriptor 값
- struct sockaddr *cliaddr
  - 인자값: NULL 또는 할당된 메모리 주소값
  - 반환값: NULL 또는 accept가 반환한 소켓과 연결된 클라이언트의 소켓 주소 구조체
- socklen_t *addrlen:
  - 인자값: NULL 또는 cliaddr의 크기
  - 반환값: NULL 또는 연결된 클라이언트의 소켓 주소 구조체 크기

close Function

```
#include <unistd.h>

int close(int sockfd);
```

Returns: 0 if OK, -1 on error

close Function

- 소켓을 닫고 TCP connection을 종료함
  - 더이상 종료된 소켓에 대해 read나 write를 수행할 수 없음
    - 만약 데이터가 송신 큐에 존재할 경우, 해당 데이터를 모두 송신한 직후에 TCP 종료 과정을 수행
  - 두개 이상의 프로세스가 하나의 소켓을 공유할 경우 TCP 종료 수행 안함
    - 각 Socket descripter에 대한 Reference count를 유지하여 해당 값이 0일 경우에만 TCP 종료 수행
    - Reference count에 관계없이 TCP 연결을 종료하려면 shutdown 함수 사용
- 인자값 (혹은 반환값)
  - int sockfd: 종료하고자 하는 소켓에 대한 socket descriptor 값

TCP 서버 예제 코드:
TCP daytime server - [intro/daytimetcpsrv.c]

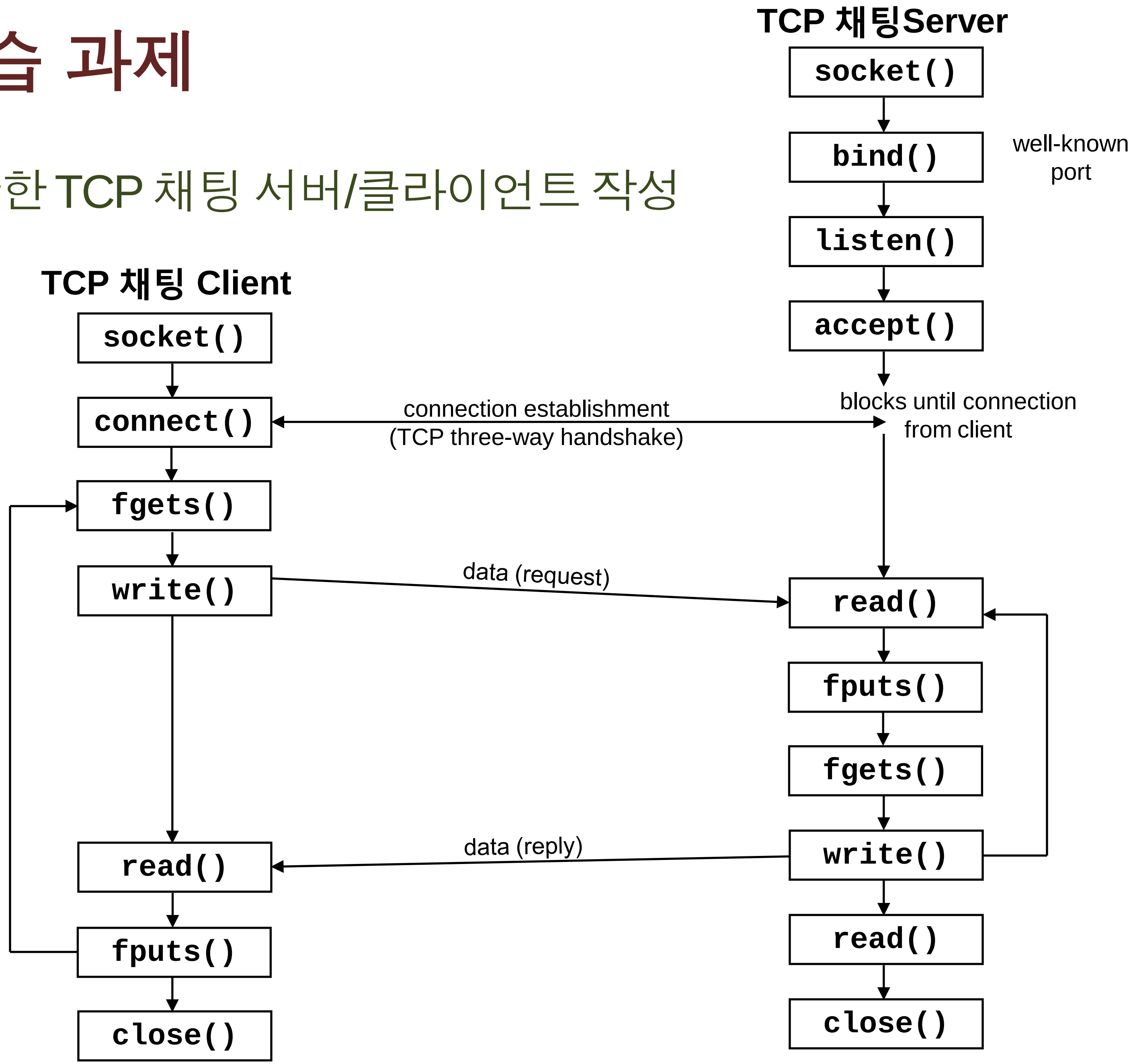
```
#include  "unp.h"
#include  <time.h>

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

    listenfd = Socket(AF_INET, SOCK_STREAM, 0);

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

TCP 서버 예제 코드:
TCP daytime server - [intro/daytimetcpsrv.c]

```
    Bind(listenfd, (SA *) &servaddr, sizeof(servaddr));

    Listen(listenfd, LISTENQ);

    for ( ; ; ) {
        connfd = Accept(listenfd, (SA *) NULL, NULL);

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

        Close(connfd);
    }
}
```

실습 과제

간단한 TCP 채팅 서버/클라이언트 작성



3.2 Get Socket Name Functions - getsockname and getpeername Functions

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

int getsockname(int sockfd, struct sockaddr *localaddr,
                socklen_t *addrlen);

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

Returns: 0 if OK, -1 on error

Get Socket Name Functions - getsockname and getpeername Functions

getsockname 함수

- 인자값으로 넘겨준 socket descriptor 자신의 소켓 주소 확인

getpeername 함수

- 함수인자값으로 넘겨준 socket decriptor와 연결되어 있는 상대방의 소켓 주소

인자값 (혹은 반환값)

- int sockfd: 주소값을 확인하고자 하는 소켓에 대한 socket descriptor 값
- struct sockaddr *addr: 소켓 주소값이 저장되어 반환
- socklen_t *addrlen: 반환된 소켓 주소값의 크기

3.3 Name and Address Conversions
- gethostbyname and gethostbyaddr Functions

```
#include <netdb.h>

struct hostent *gethostbyname(const char *hostname);

struct hostent *gethostbyaddr(const char *addr, socklen_t
                 len, int family);
```

Returns: non-null pointer if OK, NULL on error with h_error set

Name and Address Conversions
- gethostbyname and gethostbyaddr Functions

두 함수 모두 특정 호스트에 대한 struct hostent 정보를 얻기위해 호출

- gethostbyname은 도메인 네임을 인자값으로 호출
- gethostbyaddr은 IP 주소를 인자값으로 호출

```
struct hostent{
    char *h_name;    /* official (canonical) name of host */
    char **h_aliases;   /* pointer to array of pointers to
                          alias names */
    int    h_addrtype;   /* host address type: AF_INET */
    int h_length;          /* length of address: 4 */
    char **h_addr_list; /* ptr to array of ptrs with IPv4
                          addrs */
};
```

Call gethostbyname and print returned information - [names/hostent.c]

```
#include  "unp.h"

int
main(int argc, char **argv)
{
    char            *ptr, **pptr;
    char            str[INET_ADDRSTRLEN];
    struct hostent   *hptr;

    while (--argc > 0) {
        ptr = *++argv;
        if ( (hptr = gethostbyname(ptr)) == NULL) {
           err_msg("gethostbyname error for host: %s: %s",
                  ptr, hstrerror(h_errno));
           continue;
        }
        printf("official hostname: %s\n", hptr->h_name);
```

Call gethostbyname and print returned information - [names/hostent.c]

```
        for (pptr = hptr->h_aliases; *pptr != NULL; pptr++)
            printf("\talias: %s\n", *pptr);

        switch (hptr->h_addrtype) {
        case AF_INET:
            pptr = hptr->h_addr_list;
            for ( ; *pptr != NULL; pptr++)
                printf("\taddress: %s\n",
                    Inet_ntop(hptr->h_addrtype, *pptr, str,
                              sizeof(str)));
            break;

        default:
            err_ret("unknown address type");
            break;
        }
    }
    exit(0);
}
```

getservbyname and getservbyport Functions

```
#include <netdb.h>

struct servent *getservbyname(const char *servname, const
                char *protoname);

struct servent *getservbyport(int port, const char
                *protoname);
```

Returns: non-null pointer if OK, NULL on error

getservbyname and getservbyport Functions

두 함수 모두 시스템 내에서 동작하고 있는 특정 서비스에 대한 struct servent 정보를 얻기위해 호출

- getservbyname은 서비스 이름과 프로토콜 이름을 인자값으로 호출
- getservbyport은 포트 넘버와 프로토콜 이름을 인자값으로 호출

```
struct servent{
    char *s_name;     /* official service name */
    char **s_aliases;    /* alias list */
    int    s_port;    /* port number, network-byte order */
    char *s_proto;        /* protocol to use */
};
```

TCP daytime client that uses gethostbyname and getservbyname – [names/daytimetcpcli1.c]

```
#include  "unp.h"

int
main(int argc, char **argv)
{
    int                 sockfd, n;
    char                recvline[MAXLINE + 1];
    struct sockaddr_in  servaddr;
    struct in_addr      **pptr;
    struct in_addr      *inetaddrp[2];
    struct in_addr      inetaddr;
    struct hostent      *hp;
    struct servent      *sp;

    if (argc != 3)
        err_quit("usage: daytimetcpcli1 <hostname>
               <service>");
```

TCP daytime client that uses gethostbyname and getservbyname – [names/daytimetcpcli1.c]

```
    if ( (hp = gethostbyname(argv[1])) == NULL) {
        if (inet_aton(argv[1], &inetaddr) == 0) {
            err_quit("hostname error for %s: %s", argv[1],
                    hstrerror(h_errno));
        } else {
            inetaddrp[0] = &inetaddr;
            inetaddrp[1] = NULL;
            pptr = inetaddrp;
        }
    } else {
        pptr = (struct in_addr **) hp->h_addr_list;
    }

    if ( (sp = getservbyname(argv[2], "tcp")) == NULL)
        err_quit("getservbyname error for %s", argv[2]);
```

TCP daytime client that uses gethostbyname and getservbyname – [names/daytimetcpcli1.c]

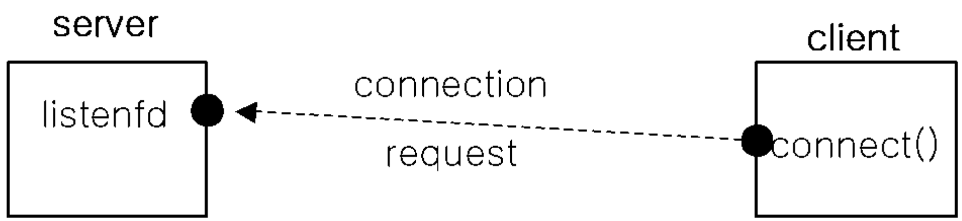
```
    for ( ; *pptr != NULL; pptr++) {
        sockfd = Socket(AF_INET, SOCK_STREAM, 0);

        bzero(&servaddr, sizeof(servaddr));
        servaddr.sin_family = AF_INET;
        servaddr.sin_port = sp->s_port;
        memcpy(&servaddr.sin_addr, *pptr, sizeof(struct
               in_addr));
        printf("trying %s\n“, Sock_ntop((SA *) &servaddr,
               sizeof(servaddr)));
```

TCP daytime client that uses gethostbyname and getservbyname – [names/daytimetcpcli1.c]

```
        if (connect(sockfd, (SA *) &servaddr,
                sizeof(servaddr)) == 0)
            break;    /* success */
        err_ret("connect error");
        close(sockfd);
    }
    if (*pptr == NULL)
        err_quit("unable to connect");

    while ( (n = Read(sockfd, recvline, MAXLINE)) > 0) {
        recvline[n] = 0; /* null terminate */
        Fputs(recvline, stdout);
    }
    exit(0);
}
```

3.4 Concurrent Servers

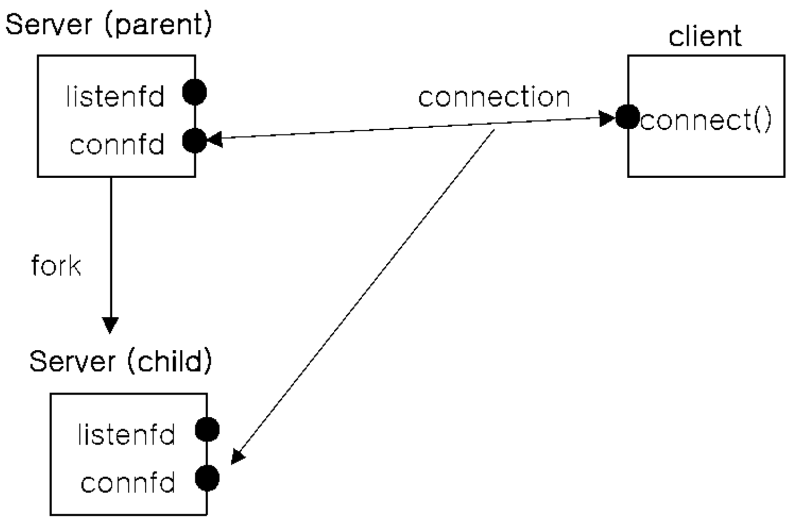
Before call to accept returns



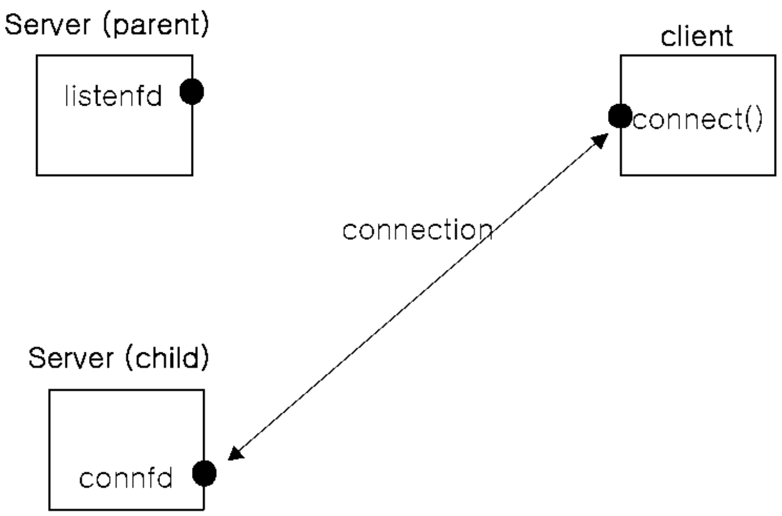
After return from accept

Concurrent Servers (계속)

After fork returns



After parent and child close appropriate sockets



fork and exec Functions

```
#include <unistd.h>

pid_t fork(void);
```

Returns: 0 in child, process ID of child in parent, -1 on error

fork and exec Functions

```
#include <unistd.h>

int execl(const char *pathname, const char *arg0, ... /*
              (char *) 0 */ );

int execv(const char *pathname, char *const argv[]);

int execle(const char *pathname, const char *arg0, ...
              /* (char *) 0, char *const envp[] */ );

int execve(const char *pathname, char *const argv[] , char
              *const envp[]);

int execlp(const char *filename, const char *arg0, ... /*
              (char *) 0 */ );

int execvp(const char *filename, char *const argv[]);
```

All six return: -1 on error, no return on sucess

Outline for typical concurrent server

```
pid_t  pid;
int listenfd, connfd;

listenfd = Socket( ... );

/* fill in sockaddr_in{} with serv’s well-known port */
Bind(listenfd, ... );
Listen(listenfd, LISTENQ);
```

Outline for typical concurrent server

```
for ( ; ; ) {
    connfd = Accept(listenfd, ... );/* probably blocks */

    if ( (pid = Fork()) == 0) {
        Close(listenfd); /* child closes listen socket */
        doit(connfd);    /* process the request */
        Close(connfd);       /* done with this client */
        exit(0);         /* child terminates */
    }

    Close(connfd);   /* parent closes connected socket */
}
```

실습 과제

DNS 호출 클라이언트

- 도메인 네임을 서버로 송신
- 서버로부터 응답받은 스트링을 화면에 출력

단순한 DNS 서버 (1)

- 클라이언트로부터 도메인 네임을 수신
- gethostbyname 함수를 사용해서 수신한 도메인 네임에 대한 호스트 정보 확인
- 대표 IP 주소를 문자열로 만들어서 클라이언트로 송신

단순한 DNS 서버 (2)

- fork 함수를 사용해서 다중 접속 지원하도록 수정

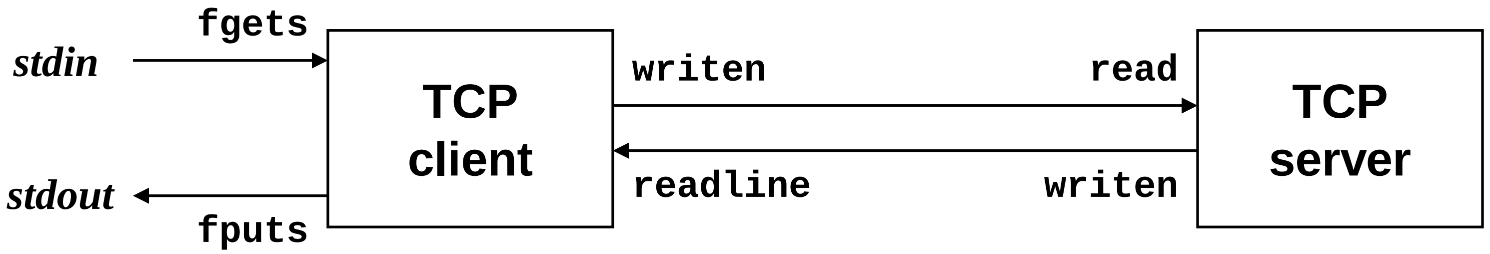
4. TCP 클라이언트/서버 예제

목 차

4.1 TCP Echo 클라이언트/서버 개요

TCP 클라이언트/서버 예제 프로그램 동작

- 클라이언트는 표준 입력으로 한줄을 읽어서 서버로 전송
- 서버는 클라이언트로부터 받은 메시지를 바로 클라이언트로 전송
- 클라이언트는 서버로부터 받은 Echo 메시지를 화면에 표준 출력



4.2 TCP Echo Server – [tcpcliserv / tcpserv01.c]

```
#include  "unp.h"

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

    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);
```

TCP Echo Server – [tcpcliserv / tcpserv01.c]

```
    Bind(listenfd, (SA *) &servaddr, sizeof(servaddr));

    Listen(listenfd, LISTENQ);

    for ( ; ; ) {
        clilen = sizeof(cliaddr);
        connfd = Accept(listenfd, (SA *) &cliaddr,
                   &clilen);

        if ( (childpid = Fork()) == 0) {/* child proc */
            Close(listenfd); /* close listening socket */
            str_echo(connfd);   /* process the request */
            exit(0);
        }
        Close(connfd);/* parent closes connected socket */
    }
}
```

str_echo function: echoes data on a socket – [lib/str_echo.c]

```
#include  "unp.h"

void
str_echo(int sockfd)
{
    ssize_t      n;
    char       buf[MAXLINE];

again:
    while ( (n = read(sockfd, buf, MAXLINE)) > 0)
        Writen(sockfd, buf, n);

    if (n < 0 && errno == EINTR)
        goto again;
    else if (n < 0)
        err_sys("str_echo: read error");
}
```

4.3 TCP Echo Client – [tcpcliserv/tcpcli01.c]

```
#include  "unp.h"

int
main(int argc, char **argv)
{
    int                 sockfd;
    struct sockaddr_in  servaddr;

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

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

TCP Echo Client – [tcpcliserv/tcpcli01.c]

```
    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, (SA *) &servaddr, sizeof(servaddr));

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

    exit(0);
}
```

str_cli function: client processing loop – [lib/str_cli.c]

```
#include  "unp.h"

void
str_cli(FILE *fp, int sockfd)
{
    char   sendline[MAXLINE], recvline[MAXLINE];

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

        Writen(sockfd, sendline, strlen(sendline));

        if (Readline(sockfd, recvline, MAXLINE) == 0)
            err_quit("str_cli: server terminated
                prematurely");

        Fputs(recvline, stdout);
    }
}
```

4.4 정상 동작 상황 - Normal Startup

정상 동작 시 에코 서버 / 클라이언트의 상태 변화

- 서버 / 클라이언트 실행 후 netstat과 ps를 통해서 각 프로세스 상태 관찰

```
# ./tcpserv01 &
[1] 30387

# netstat -a | grep LISTEN
tcp        0      0 *:9877                                          *:*
LISTEN

# ./tcpcli01 127.0.0.1
```

4.4 정상 동작 상황 - Normal Startup

```
# netstat –a

Active Internet connections (servers and established)

Proto Recv-Q Send-Q Local Address                          Foreign
Address                State

tcp        0      0 *:9877                                 *:*
LISTEN

tcp        0      0 localhost:52704
localhost:9877                 ESTABLISHED

tcp        0      0 localhost:9877
localhost:52704                ESTABLISHED
```

4.4 정상 동작 상황 - Normal Startup

```
# ps -A -o pid,ppid,tty,stat,args,wchan

 PID  PPID TT       STAT COMMAND                                     WCHAN

18058 18056 pts/0    Ss   bash                                         wait

30666 18058 pts/0    S    ./tcpserv01
inet_csk_accept

30670 18058 pts/0    S+   ./tcpcli01 127.0.0.1
n_tty_read

30671 30666 pts/0    S    ./tcpserv01
sk_wait_data
```

Normal Termination

클라이언트 정상 종료 시 에코 서버의 상태 변화

- 클라이언트를 종료하자마자 netstat과 ps를 통해서 각 프로세스 상태 관찰
  - 서버의 자식 프로세스가 좀비 프로세스로 전환
    - *SIGCHLD 시그널을 처리해야 함*

```
# ./tcpcli01 127.0.0.1
hello, world
hello, world
good bye
good bye
^D
```

Normal Termination

```
# netstat -a | grep 9877

tcp        0      0 *:9877                     *:*
LISTEN

tcp        0      0 localhost:47454
localhost:9877             TIME_WAIT

# ps -A -o pid,ppid,tty,stat,args,wchan

 PID  PPID TT       STAT COMMAND                    WCHAN

30666 18058 pts/0    S    ./tcpserv01
inet_csk_accept

30671 30666 pts/0    Z    [tcpserv01] <defunct>       exit
```

4.5 예외 상황 처리 - POSIX Signal Handling

시그널 또는 Soft interrupts

- 프로세스에게 어떤 이벤트가 발생함을 알려주는 것
  - 프로세스가 프로세스에게 시그널 발생
  - 커널이 프로세스에게 시그널 발생

시그널에 대한 처리 (disposition)

- 시그널 핸들러를 호출
- 무시
  - 단, SIGKILL과 SIGSTOP 시그널은 무시할 수 없음
- 기본 동작 수행
  - 각 시그널마다 기본적인 처리 방식을 가지고 있음

Signal Function that calls the POSIX sigaction Function – [lib/signal.c]

```
#include  "unp.h"

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

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

Signal Function that calls the POSIX sigaction Function – [lib/signal.c]

```
    if (signo == SIGALRM) {
#ifdef SA_INTERRUPT
        act.sa_flags |= SA_INTERRUPT; /* SunOS 4.x */
#endif
    } else {
#ifdef SA_RESTART
        act.sa_flags |= SA_RESTART;   /* SVR4, 44BSD */
#endif
    }
    if (sigaction(signo, &act, &oact) < 0)
        return(SIG_ERR);
    return(oact.sa_handler);
}
```

Handling SIGCHLD Signals

좀비 프로세스 처리

- 서버 프로그램에서 accept 함수 호출 전에 “Signal (SIGCHLD, sig_chld);” 추가

```
#include  "unp.h"

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

    pid = wait(&stat);
    printf("child %d terminated\n", pid);
    return;
}
```

Handling Interrupted System Calls

Slow system call

- 함수 read, write, accept와 같이 Block될 수 있는 시스템 콜
- 시그널에 의해서 인터럽트될 수 있음
  - 이 경우 단순히 재실행하면 해결됨

```
for ( ; ; ) {
    chilen = sizeof(cliaddr);
    if ( (connfd = accept(listenfd, (SA *) &cliaddr,
&clilen)) < 0) {
        if (errno == EINTR)
            continue;      /* back to for() */
        else
            err_sys(“accept error”);
```

wait and waitpid Functions

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

pid_t wait(int *statloc);

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

Both return: process ID if OK, 0 or -1 on error

wait and waitpid Functions

두 함수 모두 자식 프로세스의 종료를 기다림

- 자식 프로세스는 자신의 종료 상태를 부모 프로세스에게 반환
  - int *statloc에 자식 프로세스의 종료 상태가 기록
- wait 함수는 자식 프로세스 중 하나가 처음으로 종료될 때까지 기다림
- waitpid 함수는 특정 pid를 가진 자식 프로세스의 종료를 기다림
  - pid_t pid 값이 -1이면 임의의 자식 프로세스의 종료를 기다림
  - int option 값을 WNOHANG으로 지정하면 waitpid 함수는 Block되지 않음

Signal 함수를 호출하도록 개선된 TCP Echo 서버

```
#include  "unp.h"

int
main(int argc, char **argv)
{

    ...

    Listen(listenfd, LISTENQ);

    Signal(SIGCHLD, sig_chld);

```

Signal 함수를 호출하도록 개선된 TCP Echo 서버

```
    for ( ; ; ) {
        clilen = sizeof(cliaddr);
        if ( (connfd = accept(listenfd, (SA *) &cliaddr,
                             &clilen)) < 0) {
            if (errno == EINTR)
                continue;     /* back to for() */
            else
                err_sys("accept error");
        }

        if ( (childpid = Fork()) == 0) {/* child proc */
            Close(listenfd); /* close listening socket */
            str_echo(connfd);    /* process the request */
            exit(0);
        }
        Close(connfd);/* parent closes connected socket */
    }
}
```

TCP client that establishes five connections with server – [tcpcliserv / tcpcli04.c]

```
# #include"unp.h"

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

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

TCP client that establishes five connections with server – [tcpcliserv/tcpcli04.c]

```
    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);
}
```

sig_chld Function that calls waitpid Function - [tcpcliserv/sigchldwaitpid.c]

```
#include  "unp.h"

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

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

TCP Echo Server that Handles EINTR – [tcpcliserv/tcpserv04.c]

```
#include  "unp.h"

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

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

TCP Echo Server that Handles EINTR – [tcpcliserv/tcpserv04.c]

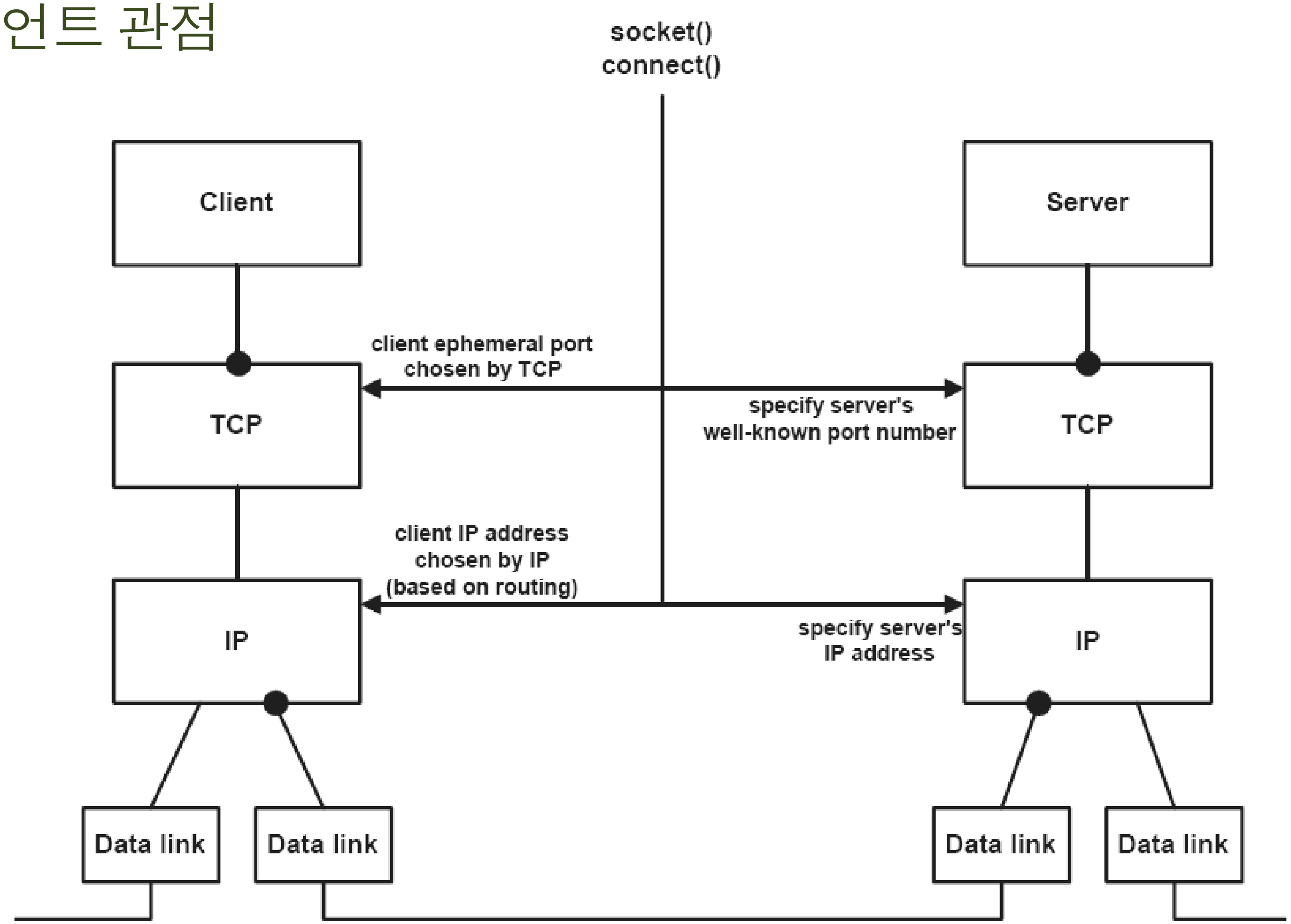
```
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family      = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port        = htons(SERV_PORT);

    Bind(listenfd, (SA *) &servaddr, sizeof(servaddr));

    Listen(listenfd, LISTENQ);

    Signal(SIGCHLD, sig_chld); /* must call waitpid() */
```

TCP Echo Server that Handles EINTR – [tcpcliserv/tcpserv04.c]

```
    for ( ; ; ) {
        clilen = sizeof(cliaddr);
        if ( (connfd = accept(listenfd, (SA *) &cliaddr,
                        &clilen)) < 0) {
            if (errno == EINTR)
                continue;     /* back to for() */
            else
                err_sys("accept error");
        }

        if ( (childpid = Fork()) == 0) {/* child proc */
            Close(listenfd); /* close listening socket */
            str_echo(connfd);   /* process the request */
            exit(0);
        }
        Close(connfd);/* parent closes connected socket */
    }
}
```

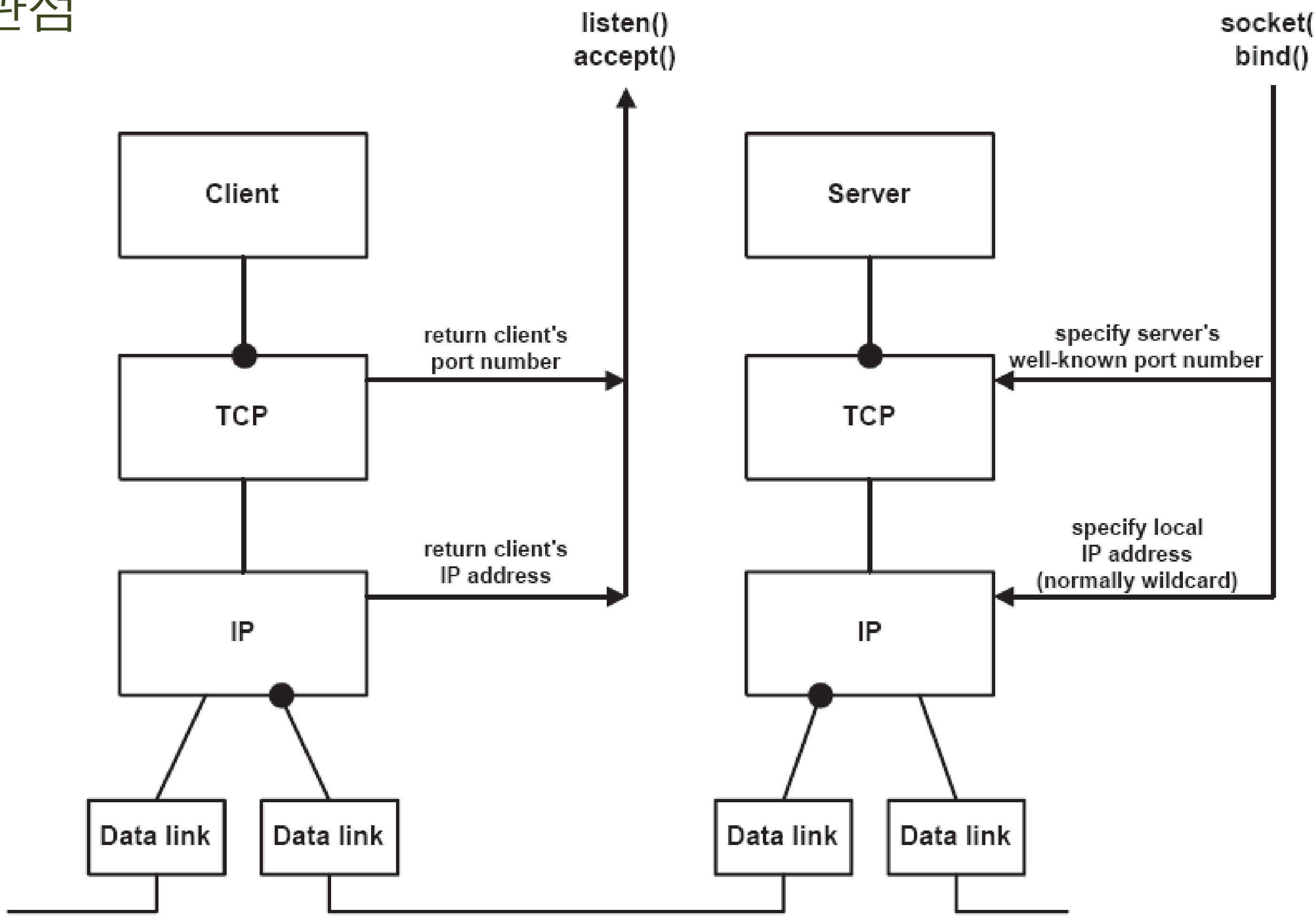
Summary of TCP Example

클라이언트 관점



Summary of TCP Example (계속)

서버 관점



4.6 Data Format 고려

Example: Passing text strings between client and server

- str_echo fuction that adds two numbers – [tcpcliserv/str_echo08.c]
  - 이 함수는 호스트 시스템의 Byte ordering에 상관없이 제대로 동작

```
#include  "unp.h"

void
str_echo(int sockfd)
{
    long       arg1, arg2;
    ssize_t       n;
    char       line[MAXLINE];
```

Data Format 고려

```
    for ( ; ; ) {
        if ( (n = Readline(sockfd, line, MAXLINE)) == 0)
            return;   /* connection closed by other end */

        if (sscanf(line, "%ld%ld", &arg1, &arg2) == 2)
            snprintf(line, sizeof(line), "%ld\n", arg1 +
                arg2);
        else
            snprintf(line, sizeof(line), "input error\n");

        n = strlen(line);
        Writen(sockfd, line, n);
    }
}
```

Example: Passing binary structures between client and server

str_cli function which sends two binary integers to server – [tcpcliserv/str_cli09.c]

```
struct args {
  long arg1;
  long arg2;
};

struct result {
  long sum;
};
```

Example: Passing binary structures between client and server

```
#include  "unp.h"
#include  "sum.h"

void
str_cli(FILE *fp, int sockfd)
{
    char          sendline[MAXLINE];
    struct args      args;
    struct result result;
```

Example: Passing binary structures between client and server

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

        if (sscanf(sendline, "%ld%ld", &args.arg1,
                   &args.arg2) != 2) {
            printf("invalid input: %s", sendline);
            continue;
        }
        Writen(sockfd, &args, sizeof(args));

        if (Readn(sockfd, &result, sizeof(result)) == 0)
            err_quit("str_cli: server terminated
                   prematurely");

        printf("%ld\n", result.sum);
    }
}
```

Example: Passing binary structures between client and server (계속)

```
#include  "unp.h"
#include  "sum.h"

void
str_echo(int sockfd)
{
    ssize_t          n;
    struct args      args;
    struct result result;

    for ( ; ; ) {
       if ( (n = Readn(sockfd, &args, sizeof(args))) == 0)
            return;   /* connection closed by other end */

        result.sum = args.arg1 + args.arg2;
        Writen(sockfd, &result, sizeof(result));
    }
}
```

Example: Passing binary structures between client and server (계속)

str_echo function that adds two binary integers – [tcpcliserv/str_echo09.c]

```
% ./tcpcli09 127.0.0.1
11 22
33
-11 -44
-55
```

```
% ./tcpcli09 [다른 Endian을 사용하는 호스트]
1 2
3
-22 -77
-16777314
```

실습 과제

양방향 채팅 서버/클라이언트 각각 작성

- 함수 fork() 사용

양방향 채팅 서버/클라이언트 통합 프로그램 작성

5. I/O 멀티플렉싱

목 차

5.1 I / O Models

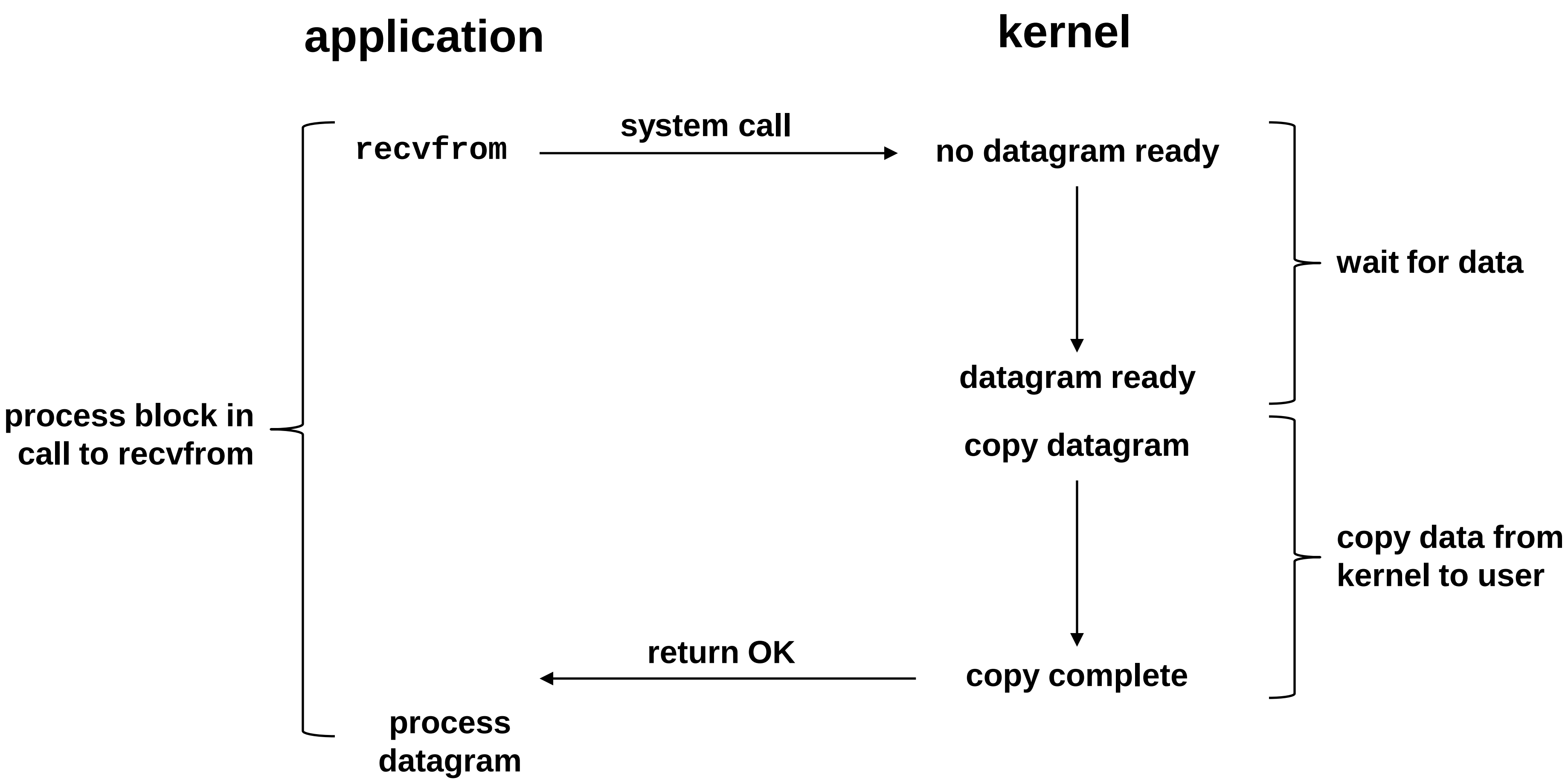
I/O multiplexing이 필요한 상황

- 클라이언트가 다수개의 descriptor들을 관리
  - 사용자 입력, 네트워크 소켓 입출력
- 다수개의 소켓을 동시에 관리
- 서버가 listen 소켓과 연결된 소켓을 관리
- 서버가 UDP와 TCP 소켓을 동시에 관리
- 서버가 다수개의 프로토콜을 지원하는 다수의 서비스를 관리
  - 예를 들면 inetd 데몬

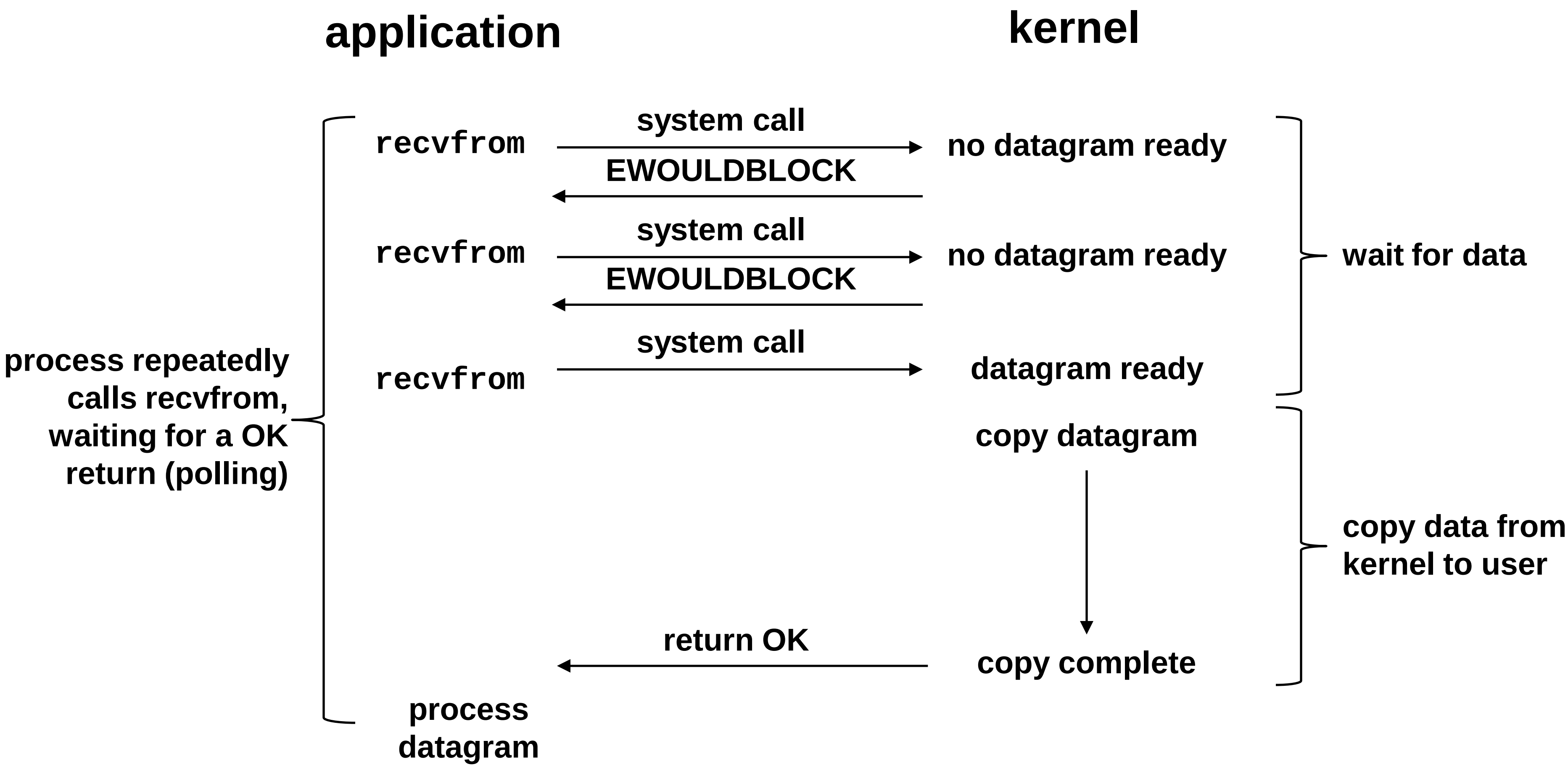
I/O 모델 분류

- Blocking I/O
- Nonblocking I/O
- I/O multiplexing (select and poll)
- signal driven I/O (SIGIO)
- asynchronous I/O (the POSIX aio_ function)

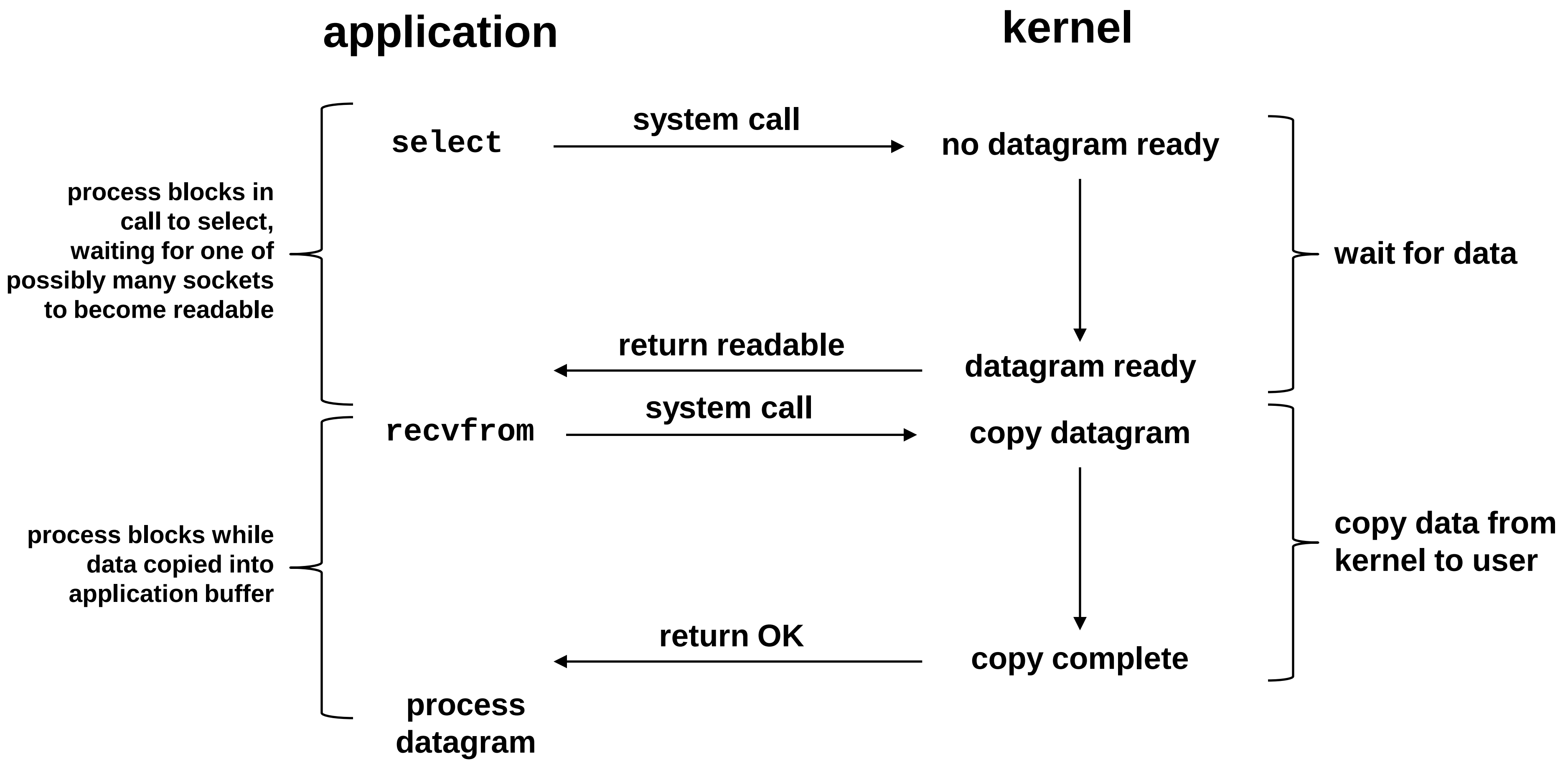
Blocking I/O Model



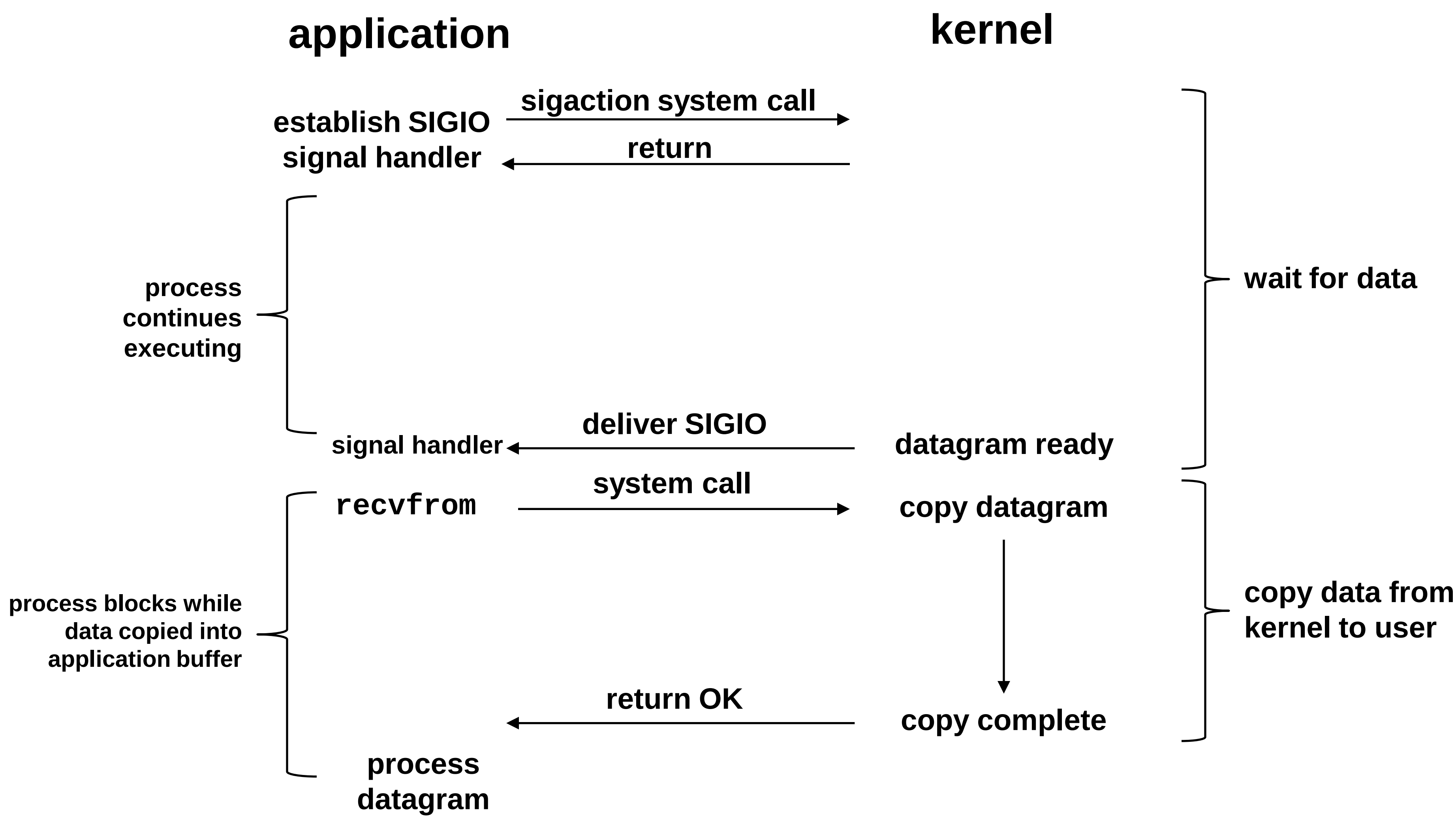
Nonblocking I/O Model



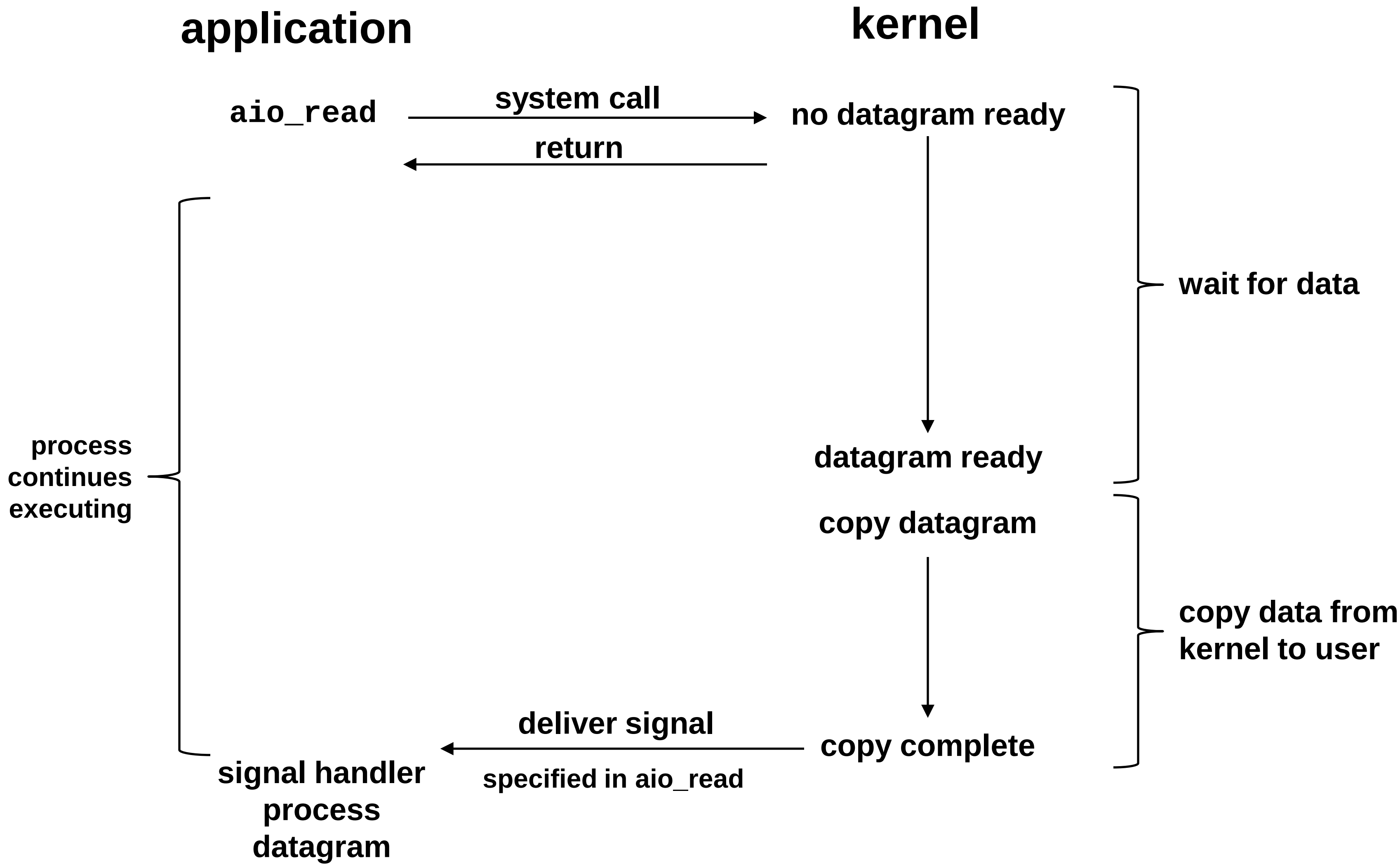
I/O Multiplexing Model



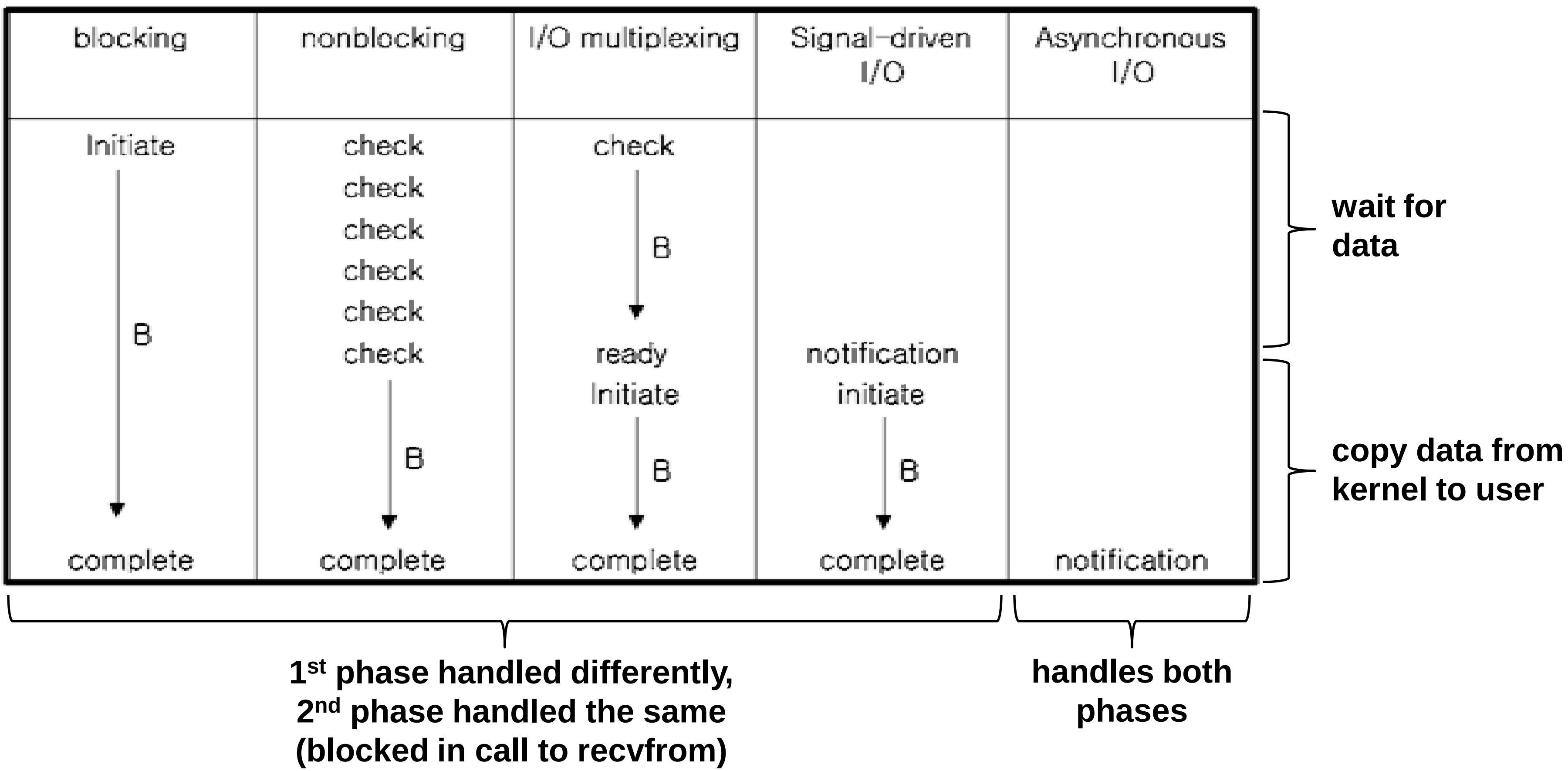
Signal-Driven I/O Model



Asynchronous I/O Model



Comparison of the I/O Models



5.2 select Function

```
#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

select Function

커널로 하여금 다수 개의 이벤트 발생에 대해서 대기하도록 지시

- 하나 혹은 다수 개의 이벤트가 발생할 경우 대기하고 있는 프로세스를 깨움
- 지정된 시간이 경과하면 프로세스를 깨움

const struct timeval *timeout 인자 값

- NULL: 영원히 대기
- 시간을 명시하면 해당 시간 동안 대기 후 타임 아웃됨

```
struct timeval {
    long        tv_sec;     /* seconds */
    long        tv_usec;    /* microseconds */
};
```

- 0: 대기 없음 (polling 방식)

select Function (계속)

인자 값으로 readset, writeset, exceptset descriptor sets를 요구함

- 각각 특정 descriptor에 대해서 read, write, exception이 발생하는지 여부를 모니터링하기 위해서 사용
  - 만약 특정 fd_set 구조체가 NULL이면 해당 조건에 대해서 관심 없음을 의미함
  - 세 개의 fd_set 구조체가 모두 NULL이면 더 조밀한 수준의 SLEEP 함수처럼 동작
  - 현재 select에서 감지할 수 있는 대표적인 Exception은 out-of-band 데이터 도착의 알림
- struct fd_set 자료구조는 보통 Integer 배열로 구현되어 있고 한 비트가 하나의 descriptor에 대한 상태를 표현하도록 구현
  - 운영체제 구현에 따라 다를 수 있음
- 손쉬운 사용을 위해서 struct fd_set 자료구조에 대해 4개의 매크로 지원

인자 값 maxfdp1

- 모니터링 될 descriptor들의 값 중에서 최대값 + 1

fd_set 구조체 운용 매크로

```
void FD_ZERO(fd_set *fdset);
        /* clear all bits in fdset */

void FD_SET(int fd, fd_set *fdset);
        /* turn on the bit for fd in fdset */

void FD_CLR(int fd, fd_set *fdset);
        /* turn off the bit for fd in fdset */

int FD_ISSET(int fd, fd_set *fdset);
        /* is the bit for fd on in fdset? */
```

fd_set 구조체 운용 매크로

FD_ISSET 매크로

- fdset 값은 value-result 인자임
  - 반환값으로써 변화가 감지된 descriptor에 해당하는 비트값이 1로 셋팅, 나머지 비트들을 모두 0
  - 가장 흔한 프로그래밍 에러는 FD_ISSET이 호출되면 fdset 값이 바뀐다는 것을 간과함으로써 발생

사용 예

```
fd_set set;

FD_ZERO(&rset);
/* initialize the set: all bits off */

FD_SET(1, &rset);   /* turn on bit for fd 1 */
FD_SET(4, &rset);   /* turn on bit for fd 4 */
FD_SET(5, &rset);   /* turn on bit for fd 5 */
```

Under What Conditions Is a Descriptor Ready?

소켓에 대한 reading이 감지된 경우

- 소켓 수신 버퍼에 수신된 데이터 양이 low-water mark 보다 크거나 같은 경우
  - 수신 버퍼의 low-water mark의 기본 값은 1이며 SO_RCVLOWAT 소켓 옵션을 사용해서 셋팅할 수 있음
- Connection의 read half가 종료된 경우 (read 함수가 EOF 반환)
- listen 소켓의 경우, 새로운 connection이 완료된 경우
- 소켓 에러가 대기 중인 경우

Under What Conditions Is a Descriptor Ready?

소켓에 대한 writing이 감지된 경우

- 소켓 송신 버퍼에 송신할 데이터 양이 low-water mark 보다 크거나 같으면서
  - 소켓이 연결되어 있거나 소켓이 연결을 필요하지 않는 경우
  - 송신 버퍼의 low-water mark의 기본 값은 2048이며 SO_SNDLOWAT 소켓 옵션을 사용해서 셋팅할 수 있음
- Connection의 write half가 종료된 경우
  - 이 경우 write를 수행하면 SIGPIPE 에러 발생
- Non-blocking connect를 사용하는 소켓에서 Connection이 완료되거나 실패한 경우
- 소켓 에러가 대기 중인 경우

소켓에 대한 exception이 감지된 경우

- Out-of-band 데이터를 수신한 경우

Conditions that cause a socket to be ready for select

condition	Readable?	Writable?	Exception?
Data to read	O		
Read-half of the connection claose	O		
New connection ready for listening socket	O		
Space available for writing		O	
Write-half of the connection closed		O	
Pending error	O	O	
TCP out of band data			O

5.3 str_cli Function (Revisted)

앞서 작성한 str_cli 함수의 문제점 – [lib/str_cli.c]

- 소켓에 대해서 어떤 이벤트가 발생하더라도 fgets 함수에서 블럭될 수 있음
- select 함수를 사용해서 표준 입력과 소켓에 대한 입력을 동시에 처리

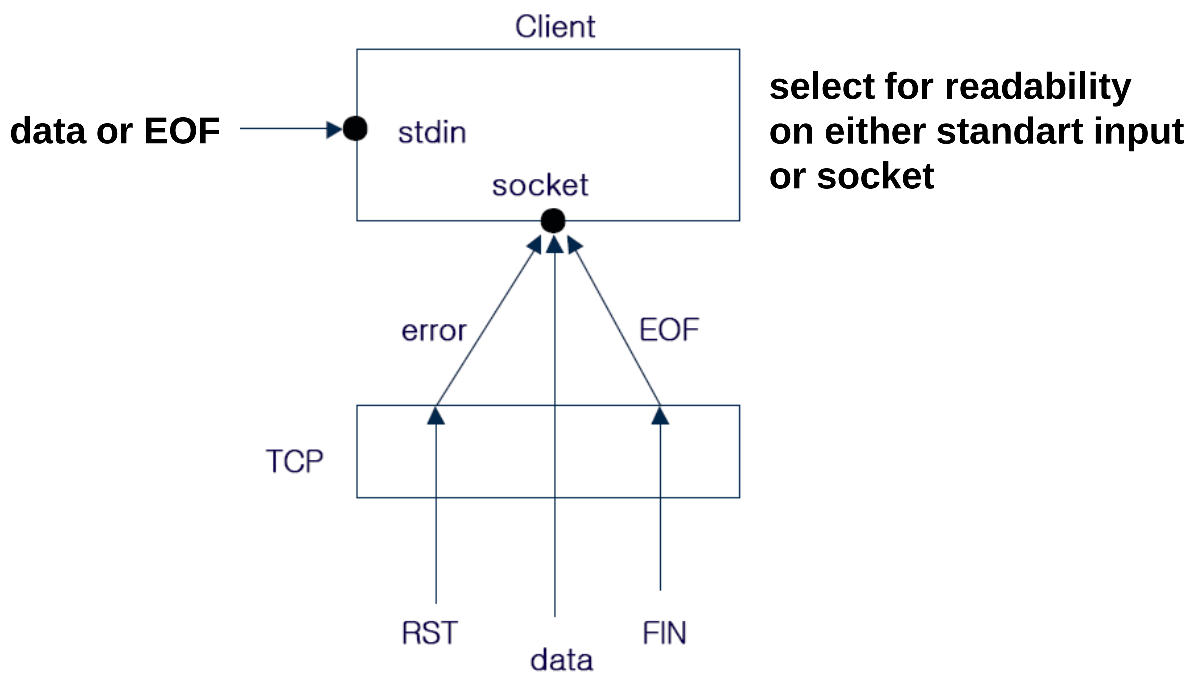
str_cli Function (Revisted)

```
#include  "unp.h"

void
str_cli(FILE *fp, int sockfd)
{
    char   sendline[MAXLINE], recvline[MAXLINE];

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

        Writen(sockfd, sendline, strlen(sendline));

        if (Readline(sockfd, recvline, MAXLINE) == 0)
            err_quit("str_cli: server terminated prematurely");

        Fputs(recvline, stdout);
    }
}
```

Conditions handled by select in str_cli

개선된 str_cli 함수에서 소켓 핸들링

- 일반 데이터 수신(read) 처리
- FIN 수신(EOF) 처리
- RST 수신(에러) 처리



str_cli Function using select Function – [select/strcliselect01.c]

```
#include  "unp.h"

void
str_cli(FILE *fp, int sockfd)
{
    int        maxfdp1;
    fd_set     rset;
    char       sendline[MAXLINE], recvline[MAXLINE];

    FD_ZERO(&rset);
    for ( ; ; ) {
        FD_SET(fileno(fp), &rset);
        FD_SET(sockfd, &rset);
        maxfdp1 = max(fileno(fp), sockfd) + 1;
        Select(maxfdp1, &rset, NULL, NULL, NULL);
```

str_cli Function using select Function – [select/strcliselect01.c]

```
        if (FD_ISSET(sockfd, &rset)) {
            /* socket is readable */
            if (Readline(sockfd, recvline, MAXLINE) == 0)
                err_quit("str_cli: server terminated
                    prematurely");
            Fputs(recvline, stdout);
        }

        if (FD_ISSET(fileno(fp), &rset)) {
            /* input is readable */
            if (Fgets(sendline, MAXLINE, fp) == NULL)
                return;       /* all done */
            Writen(sockfd, sendline, strlen(sendline));
        }
    }
}
```

5.4 Batch Input and Buffering

개선된 str_cli 함수는 여전히 문제점을 내포함

- Batch Input
  - 예) 리다이렉션을 통해서 파일의 내용을 표준 입력으로 받는 경우
- Buffering
  - 예) 표준 입출력에서 사용되는 내부 버퍼링

Batch Input 문제

- 표준 입력으로 EOF를 수신하더라도 소켓을 바로 종료해서는 않됨
  - 앞서 보낸 요청에 대한 응답을 아직 수신 완료하지 못했을 수 있음
  - 송신부는 종료 하더라도 수신부는 종료하면 서버의 응답을 손실할 가능성 있음
- shutdown 함수를 사용하면 자연스럽게 해결됨

5.4 Batch Input and Buffering

Buffering 문제

- fgets 같은 표준 입출력 함수와 select 함수를 같이 사용할 경우 발생
  - fgets 함수는 내부 버퍼링을 사용하고 한 줄 단위로 데이터를 읽어 들이므로 내부 버퍼에는 데이터가 존재할 수 있음
  - select는 read 시스템 콜의 관점에서 reading할 데이터가 있는 경우 매번 프로세스에게 알려줌
  - 만약 더이상 read 시스템 콜 관점에서 read할 데이터가 수신되지 않는다면 select는 계속 대기 상태로 블록됨
  - 이 경우 내부 버퍼에 존재하는 데이터는 처리되지 못함
- 표준 입출력 함수와 select 같은 함수를 같이 사용하지 않는 것이 바람직함

5.5 showdown Function

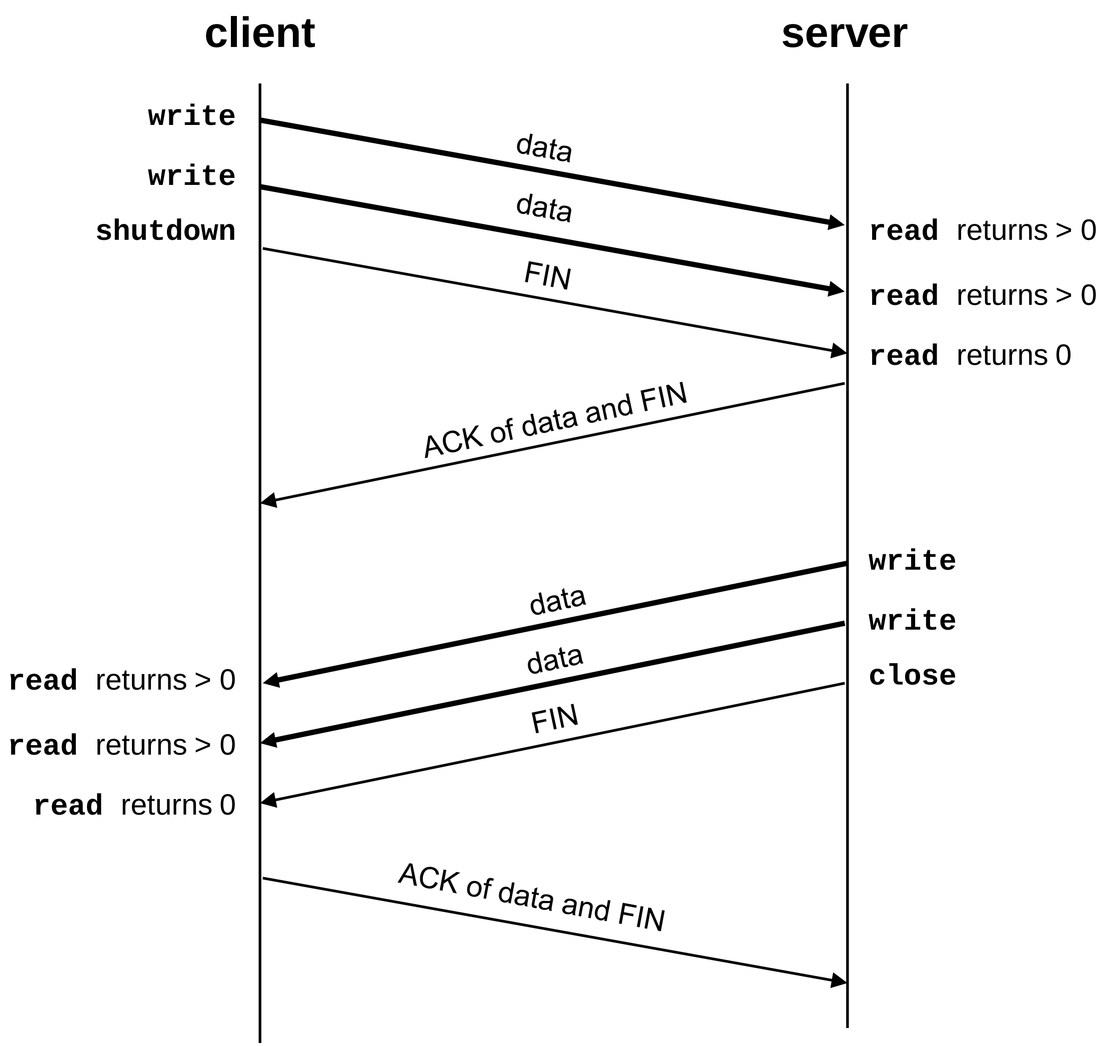
```
#include <sys/socket.h>

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

Returns: 0 if OK, -1 on error

showdown Function

- 네트워크 연결을 종료하는 함수
  - close 함수는 reference count가 0이어야지만 연결을 종료하는 반면, shutdown 함수는 그에 상관없이 연결을 종료
  - close 함수는 소켓의 송수신부 모두를 종료하는 반면, shutdown 함수는 송수신부를 선택적으로 종료할 수 있음
- howto 인자 값
  - SHUT_RD: 수신부 종료
  - SHUT_WR: 송신부 종료
  - SHUT_RDWR: 송수신부 종료

Calling shutdown to Close Half of a TCP Connection



5.6 str_cli Function (Revisted Again) – [select/strcliselect02.c]

```
#include  "unp.h"

void
str_cli(FILE *fp, int sockfd)
{
    int         maxfdp1, stdineof;
    fd_set      rset;
    char        buf[MAXLINE];
    int    n;

    stdineof = 0;
    FD_ZERO(&rset);
```

str_cli Function (Revisted Again) – [select/strcliselect02.c]

```
    for ( ; ; ) {
        if (stdineof == 0)
            FD_SET(fileno(fp), &rset);
        FD_SET(sockfd, &rset);
        maxfdp1 = max(fileno(fp), sockfd) + 1;
        Select(maxfdp1, &rset, NULL, NULL, NULL);

        if (FD_ISSET(sockfd, &rset)) {
            /* socket is readable */
            if ( (n = Read(sockfd, buf, MAXLINE)) == 0) {
                if (stdineof == 1)
                    return;        /* normal termination */
                else
                    err_quit("str_cli: server terminated
                                prematurely");
            }

            Write(fileno(stdout), buf, n);
        }
```

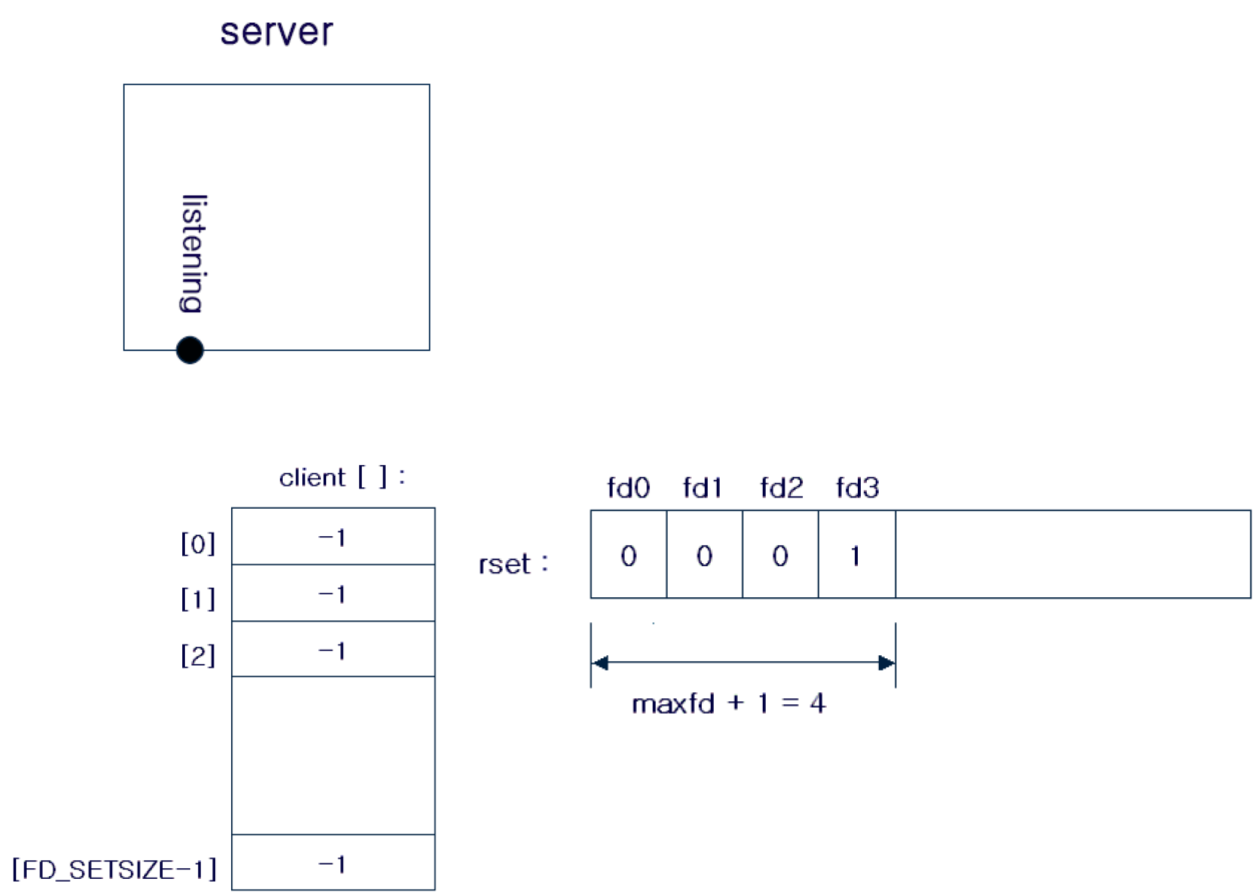
str_cli Function (Revisted Again) – [select/strcliselect02.c]

```
        if (FD_ISSET(fileno(fp), &rset)) {
            /* input is readable */
            if ((n = Read(fileno(fp), buf, MAXLINE))==0) {
                stdineof = 1;
                Shutdown(sockfd, SHUT_WR); /* send FIN */
                FD_CLR(fileno(fp), &rset);
                continue;
            }

            Writen(sockfd, buf, n);
        }
    }
}
```

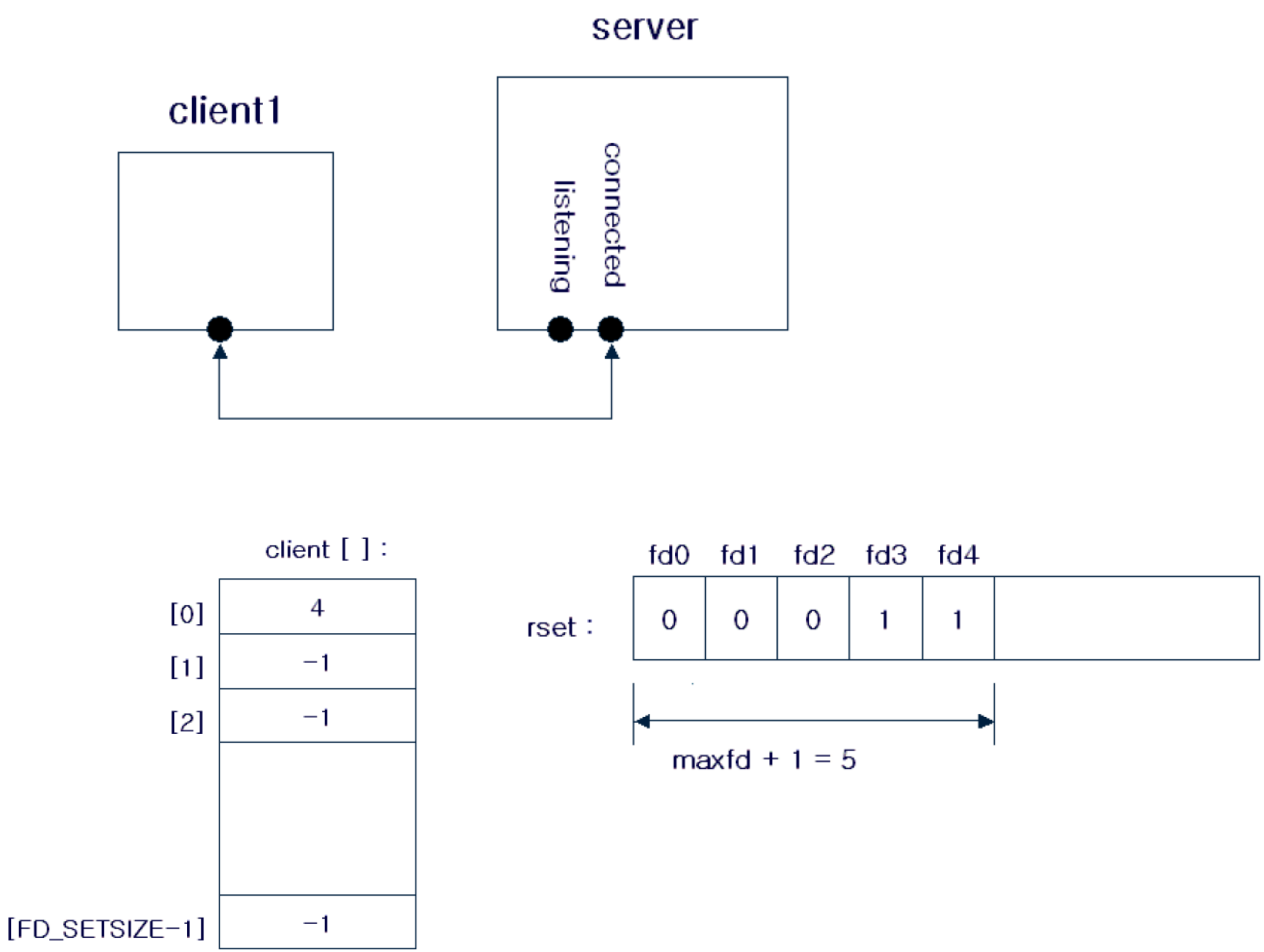
5.7 TCP Echo Server (Revisted)

select 함수를 사용해서 단일 프로세스가 다중 클라이언트를 처리함

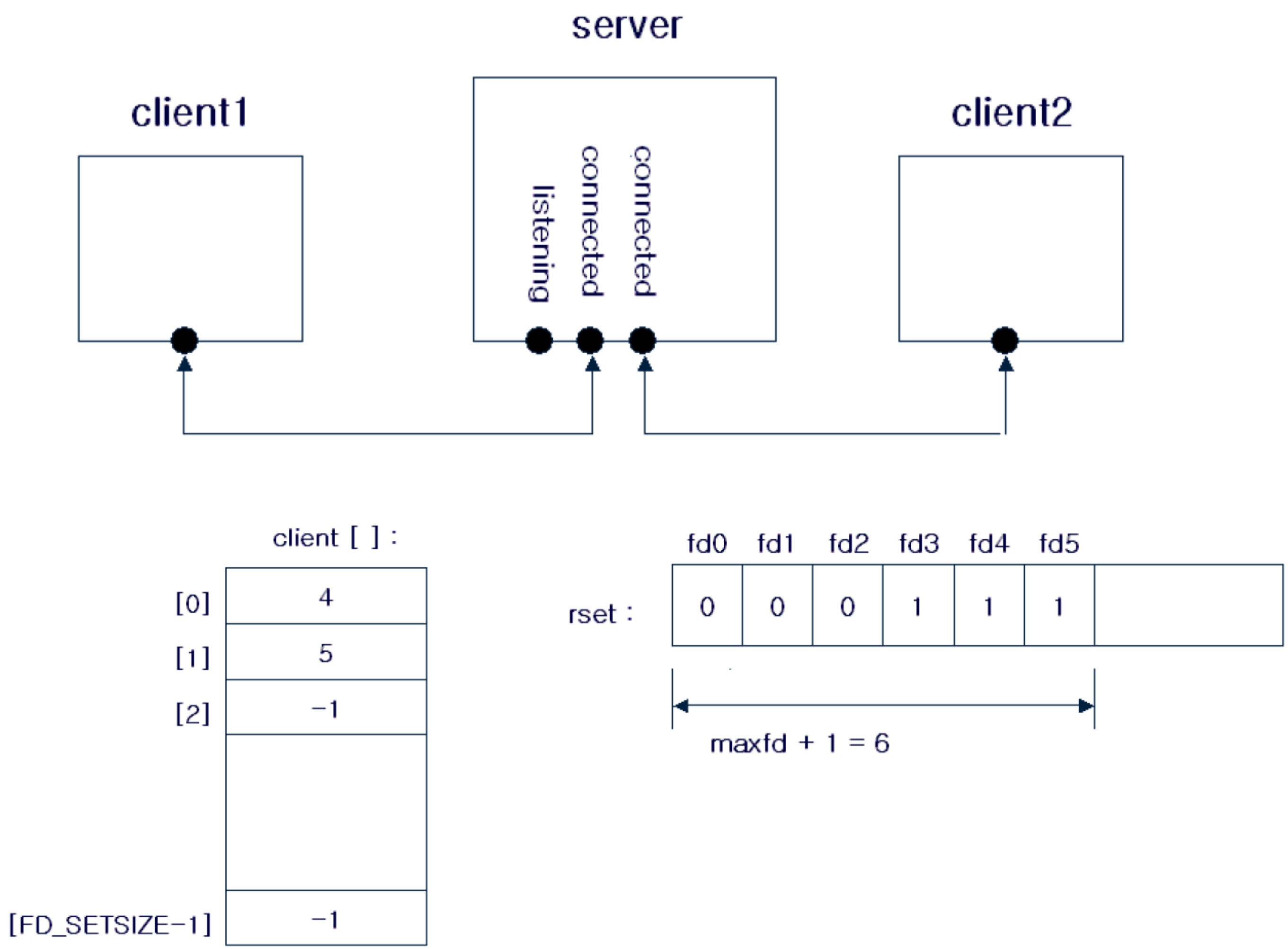
TCP 서버가 첫번째 클라이언트와 연결되기 전 상황



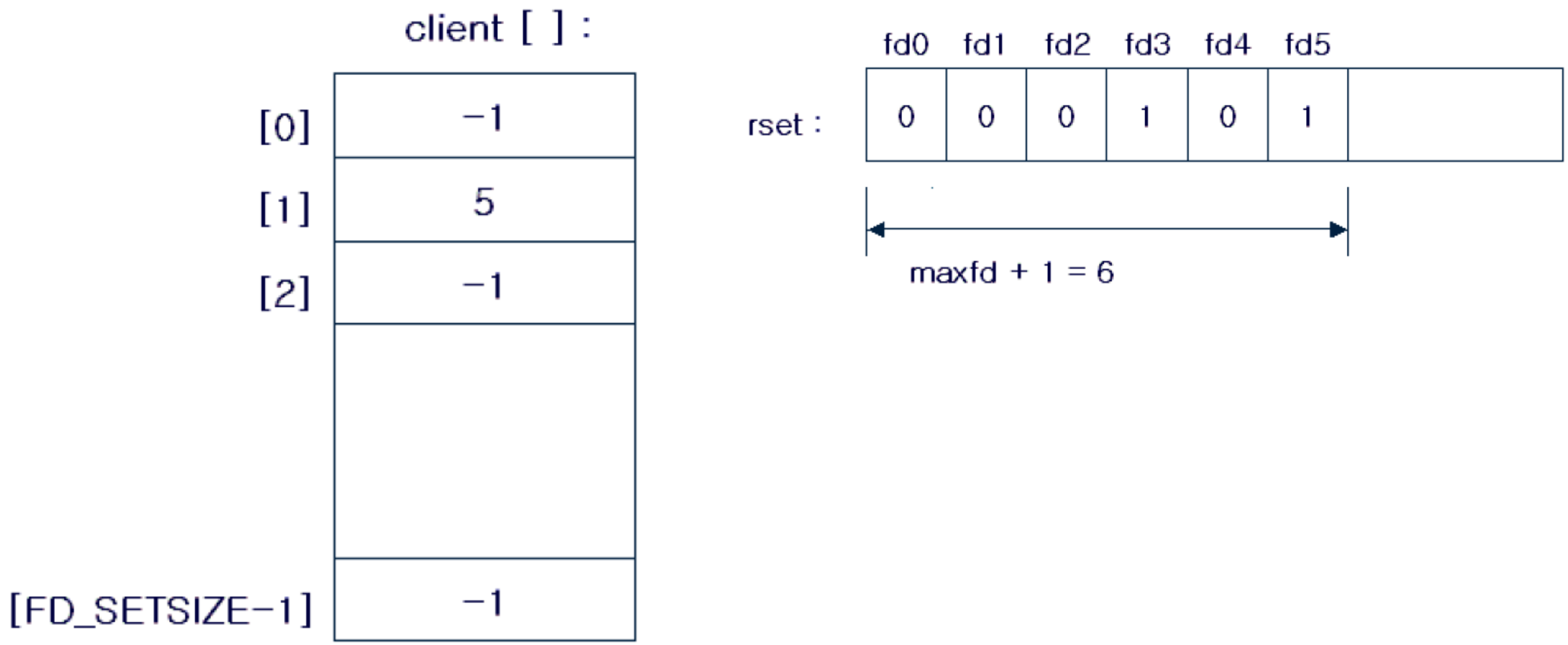
TCP 서버가 첫번째 클라이언트와 연결된 후



TCP 서버가 두번째 클라이언트와 연결된 후



TCP 서버가 첫번째 클라이언트와 연결 해제된 후



TCP Server using a Single Process and select: initialization – [tcpcliserv/tcpservselect01.c]

```
#include  "unp.h"

int
main(int argc, char **argv)
{
    int                 i, maxi, maxfd, listenfd, connfd,
                        sockfd;
    int                 nready, client[FD_SETSIZE];
    ssize_t                n;
    fd_set              rset, allset;
    char                buf[MAXLINE];
    socklen_t           clilen;
    struct sockaddr_in  cliaddr, servaddr;

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

TCP Server using a Single Process and select: initialization – [tcpcliserv/tcpservselect01.c]

```
    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);

    maxfd = listenfd;            /* initialize */
    maxi = -1;             /* index into client[] array */
    for (i = 0; i < FD_SETSIZE; i++)
        client[i] = -1;/* -1 indicates available entry */
    FD_ZERO(&allset);
    FD_SET(listenfd, &allset);
```

TCP Server using a Single Process and select loop – [tcpcliserv/tcpservselect01.c]

```
for ( ; ; ) {
        rset = allset;      /* structure assignment */
        nready = Select(maxfd+1, &rset, NULL, NULL, NULL);

        if (FD_ISSET(listenfd, &rset)) {
            /* new client connection */
            clilen = sizeof(cliaddr);
            connfd = Accept(listenfd, (SA *) &cliaddr,
                        &clilen);

            for (i = 0; i < FD_SETSIZE; i++)
                if (client[i] < 0) {
                    client[i] = connfd; /* save desc */
                    break;
                }
            if (i == FD_SETSIZE)
                err_quit("too many clients");
```

TCP Server using a Single Process and select loop – [tcpcliserv/tcpservselect01.c]

```
            FD_SET(connfd, &allset);
                /* add new descriptor to set */
            if (connfd > maxfd)
                maxfd = connfd;         /* for select */
            if (i > maxi)
                maxi = i;
                /* max index in client[] array */

            if (--nready <= 0)
                continue;
            /* no more readable descriptors */
        }
```

TCP Server using a Single Process and select loop – [tcpcliserv/tcpservselect01.c]

```
        for (i = 0; i <= maxi; i++) {
            /* check all clients for data */
            if ( (sockfd = client[i]) < 0)
                continue;
            if (FD_ISSET(sockfd, &rset)) {
                if ( (n = Read(sockfd, buf, MAXLINE))==0) {
                        /* connection closed by client */
                    Close(sockfd);
                    FD_CLR(sockfd, &allset);
                    client[i] = -1;
                } else
                    Writen(sockfd, buf, n);

                if (--nready <= 0)
                    break;    /* no more readable desc */
            }
        }
    }
}
```

5.8 poll Function

```
#include <poll.h>

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

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

poll Function

- select 함수와 비슷한 기능을 제공
  - polling 기법이 아님
- struct pollfd *fdarray 배열 인자 값
  - 관심있는 descriptor에 대해서 하나의 구조체를 사용

```
struct pollfd {
    int      fd;        /* descriptor to check */
    short    events;    /* events of interest on fd */
    short    revents;   /* events that occurred on fd */
};
```

- nfds 인자 값
  - 관심있는 descriptor 개수
- timeout 인자 값
  - millisecond 단위 타임 아웃 시간

5.9 TCP Echo Server (Revisted Again) – [tcpcliserv/tcpservpoll01.c]

```
#include  "unp.h"
#include  <limits.h>    /* for OPEN_MAX */

int
main(int argc, char **argv)
{
    int                 i, maxi, listenfd, connfd, sockfd;
    int                 nready;
    ssize_t                n;
    char                buf[MAXLINE];
    socklen_t           clilen;
    struct pollfd       client[OPEN_MAX];
    struct sockaddr_in  cliaddr, servaddr;

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

TCP Echo Server (Revisted Again) – [tcpcliserv/tcpservpoll01.c]

```
    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);

    client[0].fd = listenfd;
    client[0].events = POLLRDNORM;
    for (i = 1; i < OPEN_MAX; i++)
        client[i].fd = -1;
            /* -1 indicates available entry */
    maxi = 0;
            /* max index into client[] array */
```

TCP Echo Server (Revisted Again) – [tcpcliserv/tcpservpoll01.c]

```
for ( ; ; ) {
        nready = Poll(client, maxi+1, INFTIM);

        if (client[0].revents & POLLRDNORM) {
            /* new client connection */
            clilen = sizeof(cliaddr);
            connfd = Accept(listenfd, (SA *) &cliaddr,
                            &clilen);

            for (i = 1; i < OPEN_MAX; i++)
                if (client[i].fd < 0) {
                    client[i].fd = connfd;
                        /* save descriptor */
                    break;
                }
            if (i == OPEN_MAX)
                err_quit("too many clients");
```

TCP Echo Server (Revisted Again) – [tcpcliserv/tcpservpoll01.c]

```
            client[i].events = POLLRDNORM;
            if (i > maxi)
                maxi = i; /* max index in client[] array */

            if (--nready <= 0)
                continue; /* no more readable desc */
        }

        for (i = 1; i <= maxi; i++) {
            /* check all clients for data */
            if ( (sockfd = client[i].fd) < 0)
                continue;
            if(client[i].revents&(POLLRDNORM | POLLERR)) {
                if ((n = read(sockfd, buf, MAXLINE)) < 0) {
                    if (errno == ECONNRESET) {
                            /*connection reset by client */
                        Close(sockfd);
                        client[i].fd = -1;
```

TCP Echo Server (Revisted Again) – [tcpcliserv/tcpservpoll01.c]

```
                    } else
                        err_sys("read error");
                } else if (n == 0) {
                        /*connection closed by client */
                    Close(sockfd);
                    client[i].fd = -1;
                } else
                    Writen(sockfd, buf, n);

                if (--nready <= 0)
                    break;
                        /* no more readable descriptors */
            }
        }
    }
}
```

실습 과제

다중 접속 가능한 DNS 서버 작성

- 함수 select() 사용 (표준 입출력도 처리)
- 함수 poll() 사용

6. UDP 클라이언트/서버 통신

목 차

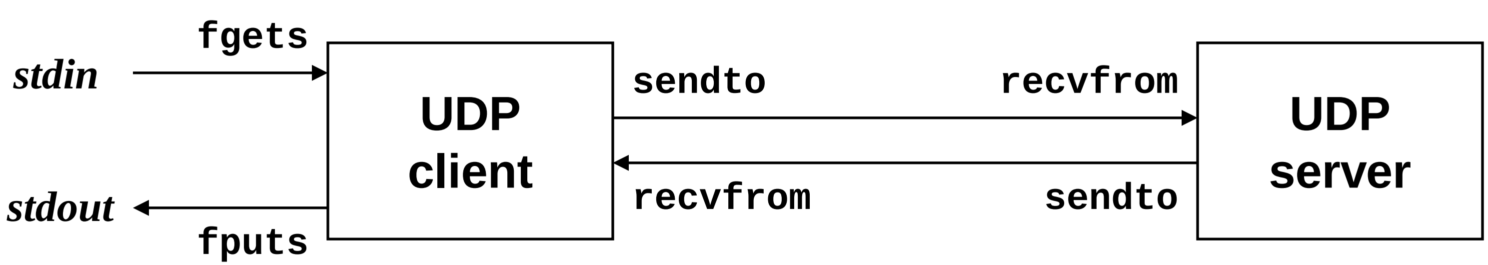
6.1 UDP Echo 클라이언트/서버 개요

UDP vs. TCP

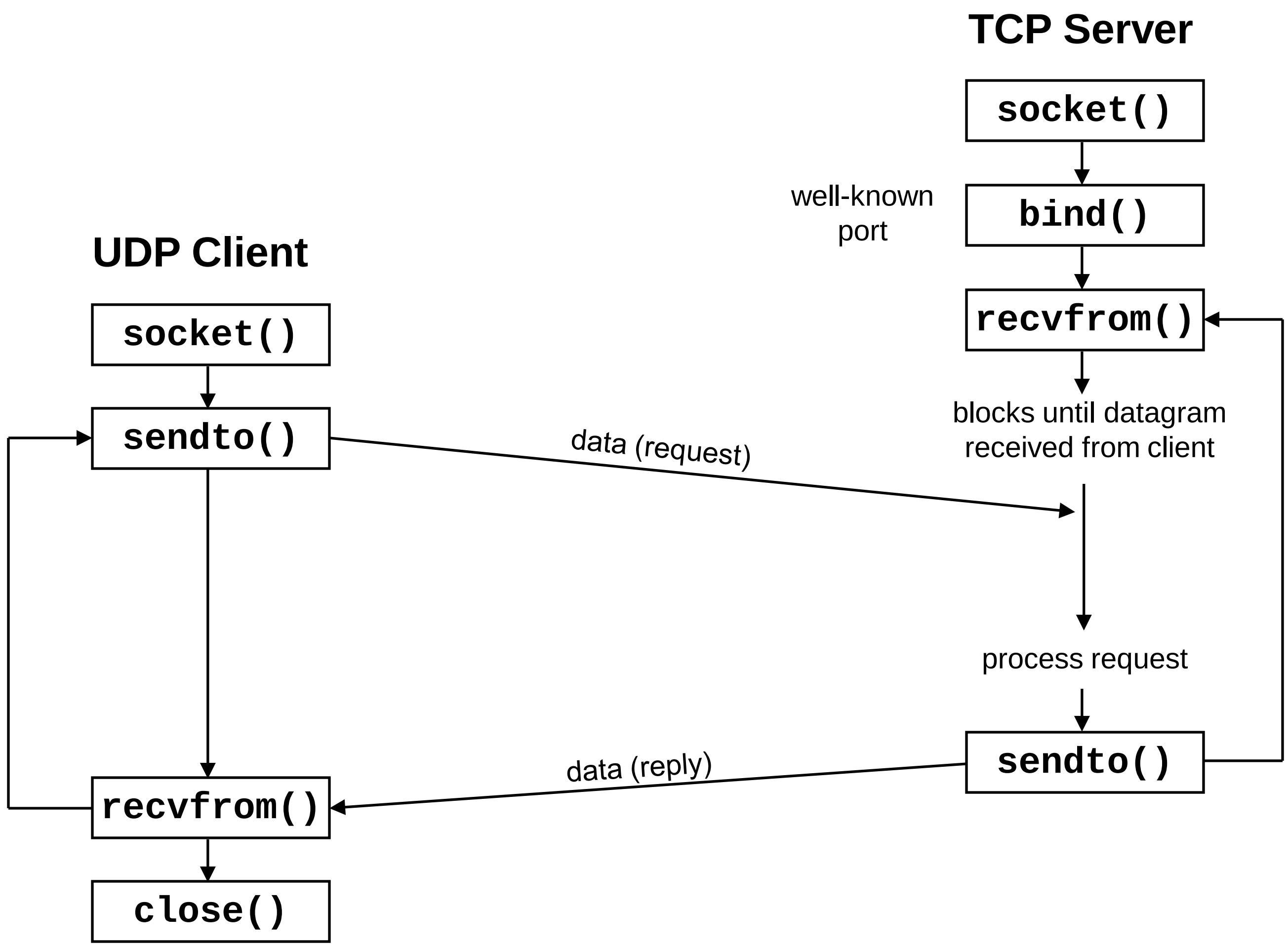
- UDP: connectionless, unreliable, datagram protocol
- TCP: connection-oriented, reliable byte stream

UDP 클라이언트/서버 예제 프로그램 동작

- 클라이언트는 표준 입력으로 한줄을 읽어서 서버로 전송
- 서버는 클라이언트로부터 받은 메시지를 바로 클라이언트로 전송
- 클라이언트는 서버로부터 받은 Echo 메시지를 화면에 표준 출력



Socket Functions for UDP Client / Server



6.2 recvfrom and sendto Functions

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

ssize_t recvfrom(int sockfd, void *buff, size_t nbytes, int
        flags, struct sockaddr *from, socklen_t *addrlen);

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

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

recvfrom and sendto Functions

connection이 설정되지 않는 UDP에서 데이터 전송 담당

인자 값

- 첫 세 인자는 read, write의 세 인자와 동일함
- int flag 인자는 보통 0으로 셋팅
  - 자세한 내용은 recv, send 함수에서 자세히 설명
- 소켓 주소 구조체 인자
  - recvfrom의 struct sockaddr *from은 패킷을 보낸 상대방 소켓 주소 정보를 포함
    - *socklen_t *addrlen 인자와 함께 Value-result 인자임에 주의*
  - sendto의 struct sockaddr *to는 패킷을 보낼 상대방의 소켓 주소 정보를 포함
- sendto 함수의 socklen_t addrlen 인자
  - 소켓 주소 구조체의 크기 명시

6.3 UDP Echo Server – [udpcliserv/udpserv01.c]

```
#include  "unp.h"

int
main(int argc, char **argv)
{
    int                 sockfd;
    struct sockaddr_in  servaddr, cliaddr;

    sockfd = Socket(AF_INET, SOCK_DGRAM, 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(sockfd, (SA *) &servaddr, sizeof(servaddr));

    dg_echo(sockfd, (SA *) &cliaddr, sizeof(cliaddr));
}
```

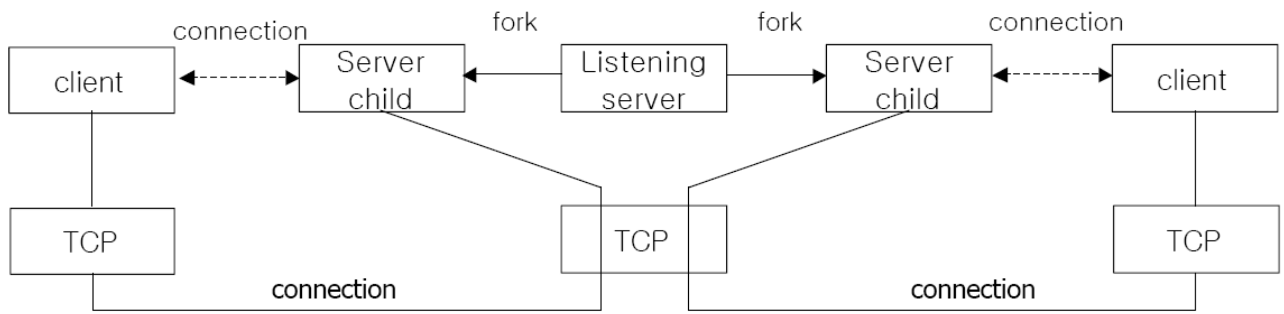
dg_echo Function – [lib / dg_echo.c]

```
#include   "unp.h"

void
dg_echo(int sockfd, SA *pcliaddr, socklen_t clilen)
{
    int         n;
    socklen_t len;
    char        mesg[MAXLINE];

    for ( ; ; ) {
        len = clilen;
        n = Recvfrom(sockfd, mesg, MAXLINE, 0, pcliaddr,
                    &len);

        Sendto(sockfd, mesg, n, 0, pcliaddr, len);
    }
}
```

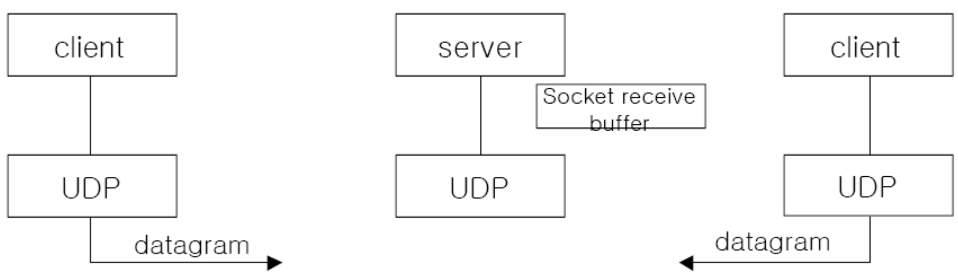
Connectionless 프로토콜이므로 TCP
처럼 EOF를 수신하지 않음,
하나의 서버 소켓이 다중 클라이언트를
처리함에 주목

Client / server with two client

TCP



UDP



6.4 TCP Echo Client – [udpcliserv/udpcli01.c]

```
#include  "unp.h"

int
main(int argc, char **argv)
{
    int                 sockfd;
    struct sockaddr_in  servaddr;

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

TCP Echo Client – [udpcliserv/udpcli01.c]

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

    sockfd = Socket(AF_INET, SOCK_DGRAM, 0);

    dg_cli(stdin, sockfd, (SA *) &servaddr,
           sizeof(servaddr));

    exit(0);
}
```

dg_cli Function: client processing loop – [lib/dg_cli.c]

```
#include  "unp.h"

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

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

dg_cli Function: client processing loop – [lib/dg_cli.c]

```
        Sendto(sockfd, sendline, strlen(sendline), 0,
            pservaddr, servlen);

        n = Recvfrom(sockfd, recvline, MAXLINE, 0, NULL,
            NULL);

        recvline[n] = 0; /* null terminate */
        Fputs(recvline, stdout);
    }
}
```

dg_cli that verifies returned socket address – [udpcliserv/dgcliaddr.c]

```
#include  "unp.h"

void
dg_cli(FILE *fp, int sockfd, const SA *pservaddr,
       socklen_t servlen)
{
    int            n;
    char       sendline[MAXLINE], recvline[MAXLINE + 1];
    socklen_t    len;
    struct sockaddr  *preply_addr;

    preply_addr = Malloc(servlen);

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

dg_cli that verifies returned socket address – [udpcliserv/dgcliaddr.c]

```
        Sendto(sockfd, sendline, strlen(sendline), 0,
pservaddr, servlen);

        len = servlen;
        n = Recvfrom(sockfd, recvline, MAXLINE, 0,
                     preply_addr, &len);
        if (len != servlen || memcmp(pservaddr,
                preply_addr, len) != 0) {
            printf("reply from %s (ignored)\n",
                   Sock_ntop(preply_addr, len));
            continue;
        }

        recvline[n] = 0; /* null terminate */
        Fputs(recvline, stdout);
    }
}
```

서버가 아닌 다른 노드로부터 UDP 데이터그램을 수신한 경우 해당 데이터그램을 무시함, (이 경우 IP 주소를 비교하기 때문에 Multihomed 서버일 경우 오류 발생)

connect Function with UDP

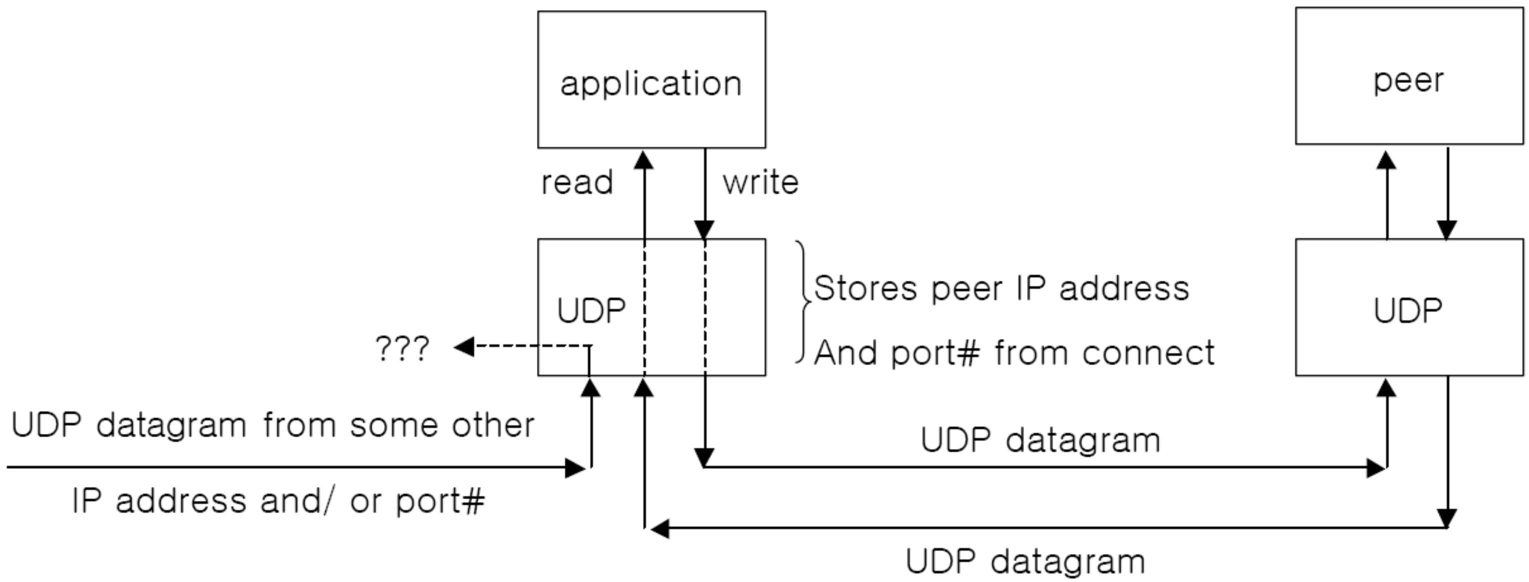
UDP 소켓에 대해서 connect 함수를 호출한 경우

- 데이터 전송을 위해서 read, write 함수 사용 가능
- Asynchronous 에러 문제 해결
  - Asynchronous error - 서버 시스템에 서버 프로세스가 동작하지 않을 때 클라이언트가 메시지를 전송하면 발생하는 에러 (ICMP "port unreachable"), 이 경우 클라이언트는 recvfrom 함수에서 영원히 대기함
  - 서버와 클라이언트간 연결 설정이 되므로 서버가 동작하지 않는 경우 바로 에러를 반환
    - » *UDP는 connect 함수를 호출하더라도 커널 내부적으로 three-way handshake를 수행하지는 않음*

connect Function with UDP

Connection UDP Socket

- 연결되지 않은 소켓 주소로부터 받은 데이터그램은 폐기됨



dg_cli Function that calls connect – [udpcliserv/dgcliconnect.c]

```c
#include  "unp.h"

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

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

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

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

        n = Read(sockfd, recvline, MAXLINE);

        recvline[n] = 0; /* null terminate */
        Fputs(recvline, stdout);
    }
}
```

6.5 TCP and UDP Echo Server Using select – [udpcliserv/udpservselect01.c]

```
#include  "unp.h"

int
main(int argc, char **argv)
{
    int                 listenfd, connfd, udpfd, nready,
                        maxfdp1;
    char                mesg[MAXLINE];
    pid_t               childpid;
    fd_set              rset;
    ssize_t                n;
    socklen_t           len;
    const int           on = 1;
    struct sockaddr_in  cliaddr, servaddr;
    void                sig_chld(int);
```

TCP and UDP Echo Server Using select – [udpcliserv/udpservselect01.c]

14-22: Creating listening TCP socket

```
14      /* 4create listening TCP socket */
15     listenfd = Socket(AF_INET, SOCK_STREAM, 0);

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

20     Setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, &on,
21               sizeof(on));
       Bind(listenfd, (SA *) &servaddr, sizeof(servaddr));
22
       Listen(listenfd, LISTENQ);
```

같은 소켓 주소를 TCP listen 소켓과 UDP 소켓에 중복해서 bind해야 하므로, listen 소켓에 할당된 포트번호를 재사용 가능하도록 설정

TCP and UDP Echo Server Using select – [udpcliserv/udpservselect01.c]

```
23       /* 4create UDP socket */
24    udpfd = Socket(AF_INET, SOCK_DGRAM, 0);

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

29    Bind(udpfd, (SA *) &servaddr, sizeof(servaddr));
```

23-29: Create UDP socket

getsockopt and setsockopt Function

```
#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

getsockopt and setsockopt Function

Open된 소켓의 옵션 값을 확인하고 설정하는 함수

인자 값

- int sockfd는 열려진 소켓 descriptor
- int level은 적용할 옵션이 포함된 계층을 명시
  - SOL_SOCKET, IPPROTO_IP 등
- int optname은 적용할 옵션 이름
  - SO_REUSEADDR등
- void *optval은 해당 옵션 값
- socklen_t optlen은 옵션 값이 저장된 메모리 크기

TCP and UDP Echo Server Using select – [udpcliserv/udpservselect01.c]

Establish signal handler for SIGCHLD

```
30     Signal(SIGCHLD, sig_chld); /* must call waitpid() */

31     FD_ZERO(&rset);
32     maxfdp1 = max(listenfd, udpfd) + 1;
33     for ( ; ; ) {
34         FD_SET(listenfd, &rset);
35         FD_SET(udpfd, &rset);
36         if ( (nready = select(maxfdp1, &rset, NULL, NULL,
                  NULL)) < 0) {
37             if (errno == EINTR)
38                 continue;    /* back to for() */
39             else
40                 err_sys("select error");
41         }
```

31-41: Prepare for select and call select

TCP and UDP Echo Server Using select – [udpcliserv/udpservselect01.c]

42-51: Handle new client connection

```
42      if (FD_ISSET(listenfd, &rset)) {
43          len = sizeof(cliaddr);
44          connfd = Accept(listenfd, (SA *) &cliaddr,
                             &len);

45          if ( (childpid = Fork()) == 0) {
                /* child process */
46              Close(listenfd);
                    /* close listening socket */
47              str_echo(connfd);
                    /* process the request */
48              exit(0);
49          }
50          Close(connfd);
                /* parent closes connected socket */
51      }
```

TCP and UDP Echo Server Using select – [udpcliserv/udpservselect01.c]

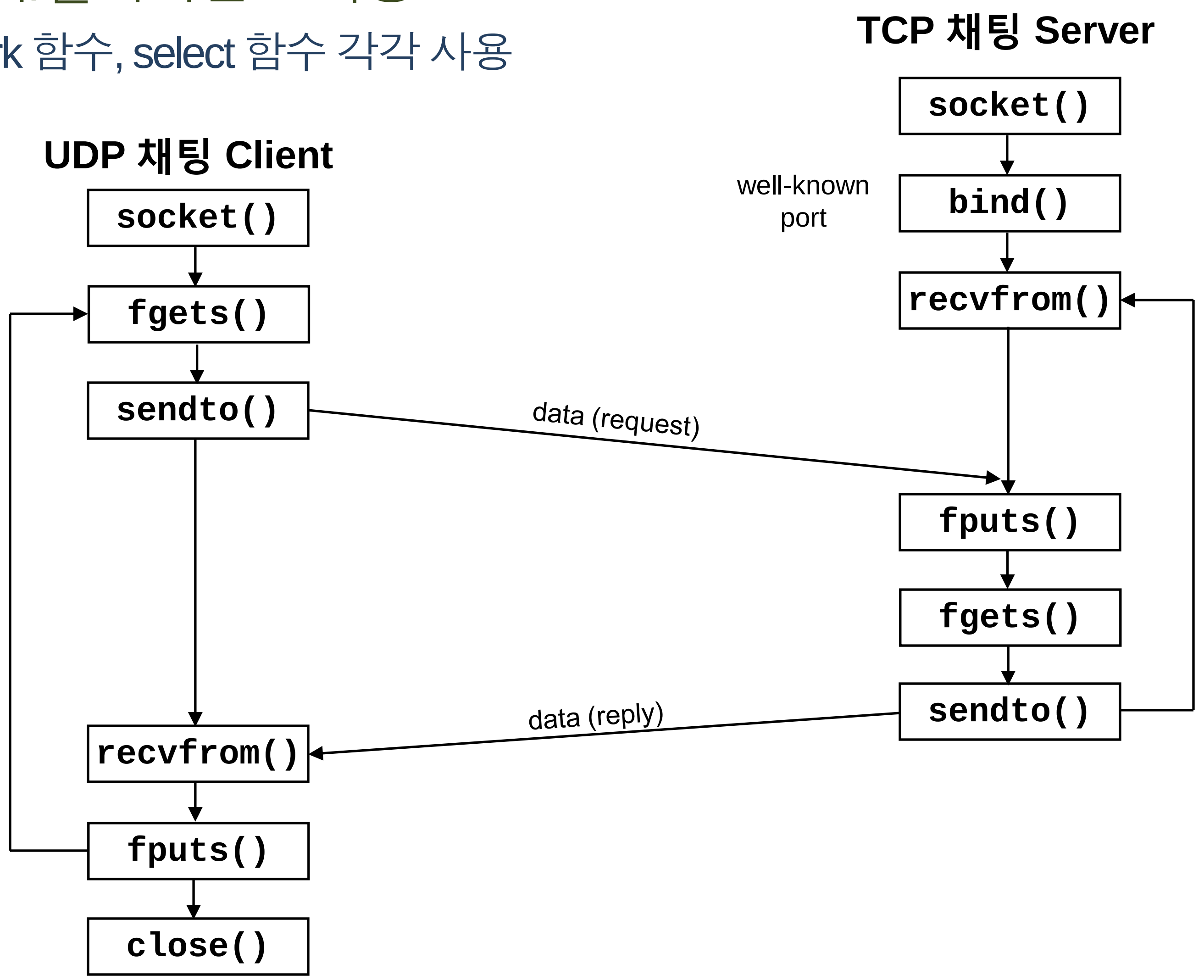
```
52          if (FD_ISSET(udpfd, &rset)) {
53              len = sizeof(cliaddr);
54              n = Recvfrom(udpfd, mesg, MAXLINE, 0, (SA *)
                        &cliaddr, &len);

55              Sendto(udpfd, mesg, n, 0, (SA *) &cliaddr,
                        len);
56          }
57      }
58  }
```

52-57: Handle arrival datagram

실습 과제

채팅 서버/클라이언트 작성

- fork 함수, select 함수 각각 사용



7. NONBLOCKING I/O

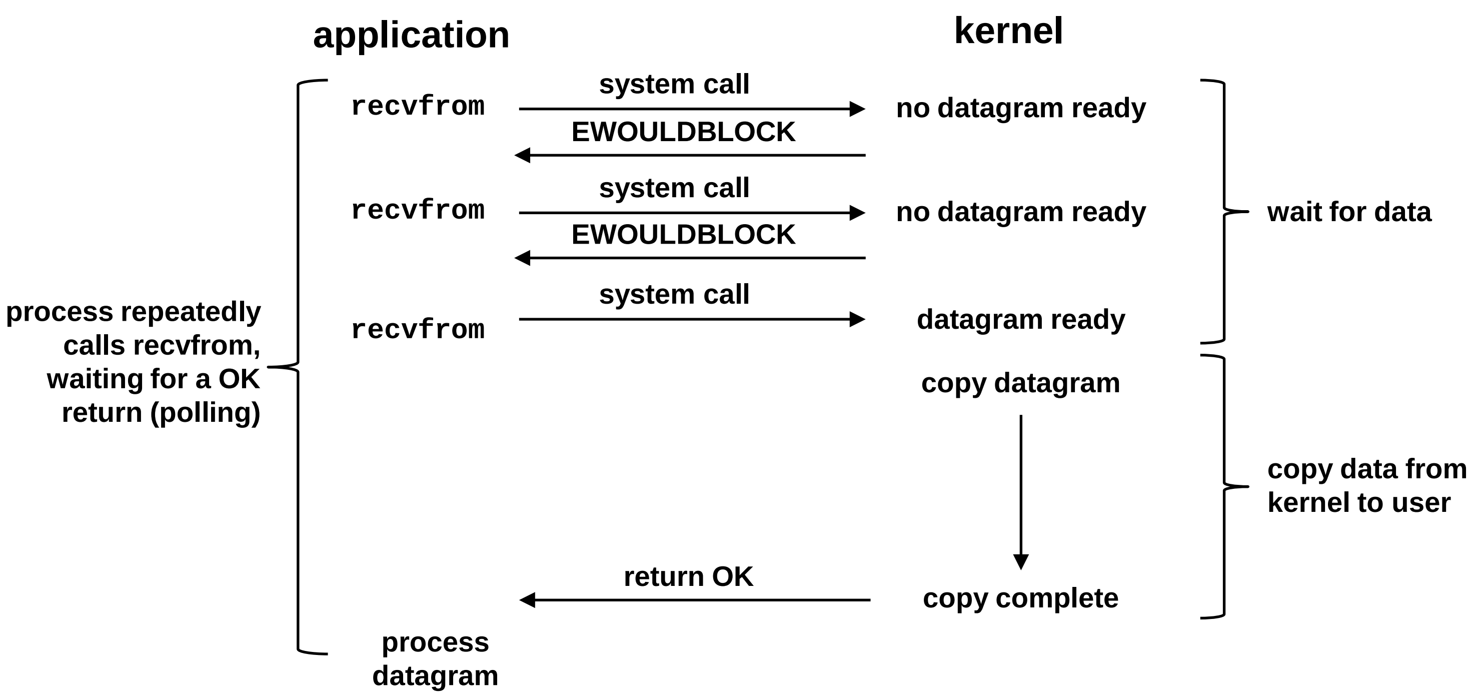
목 차

7.1 Introduction

기본적으로 소켓은 Blocking임

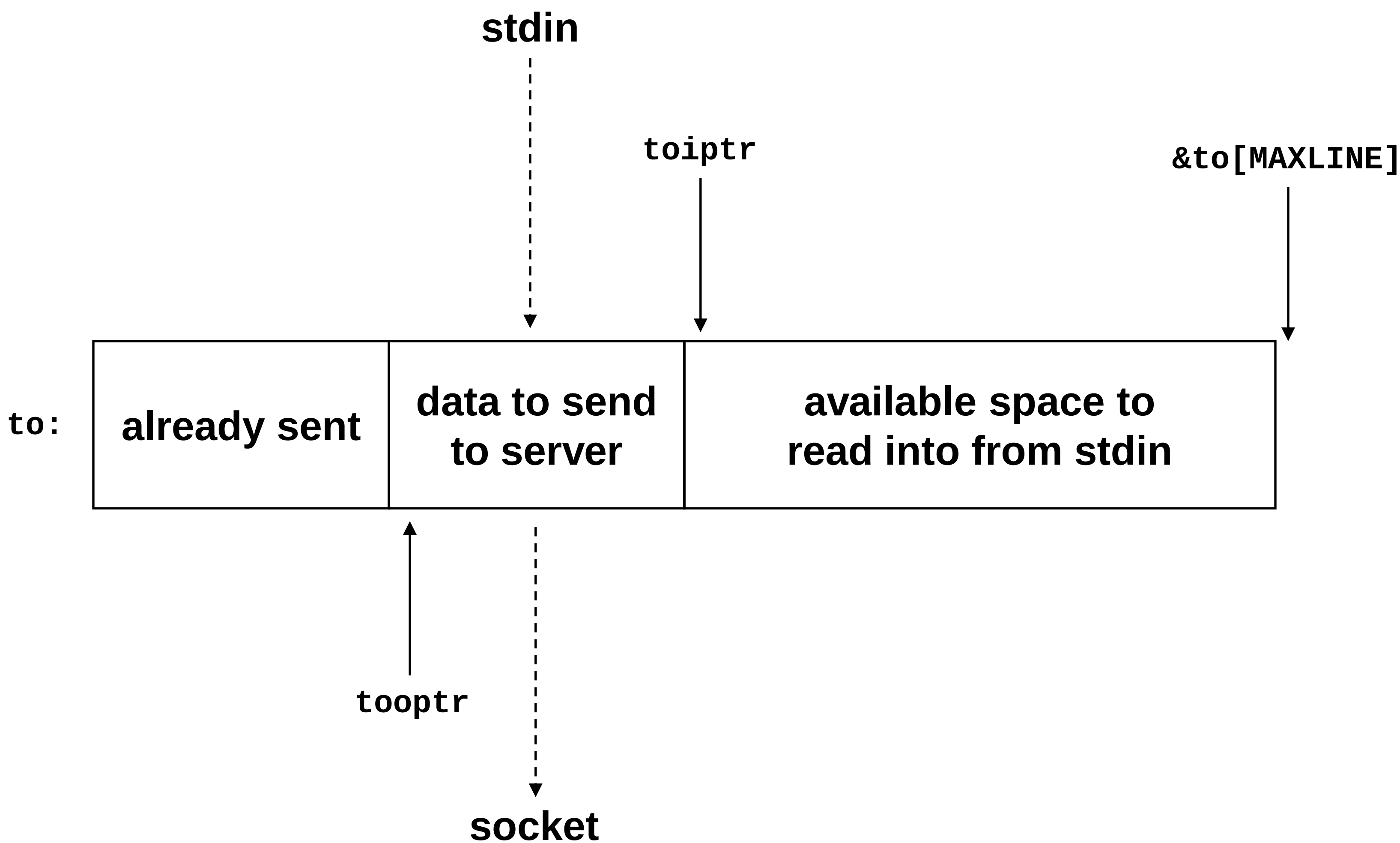
- Input operation, output operation
- Accepting incoming connections, initiating outgoing connections

Nonblocking I/O model



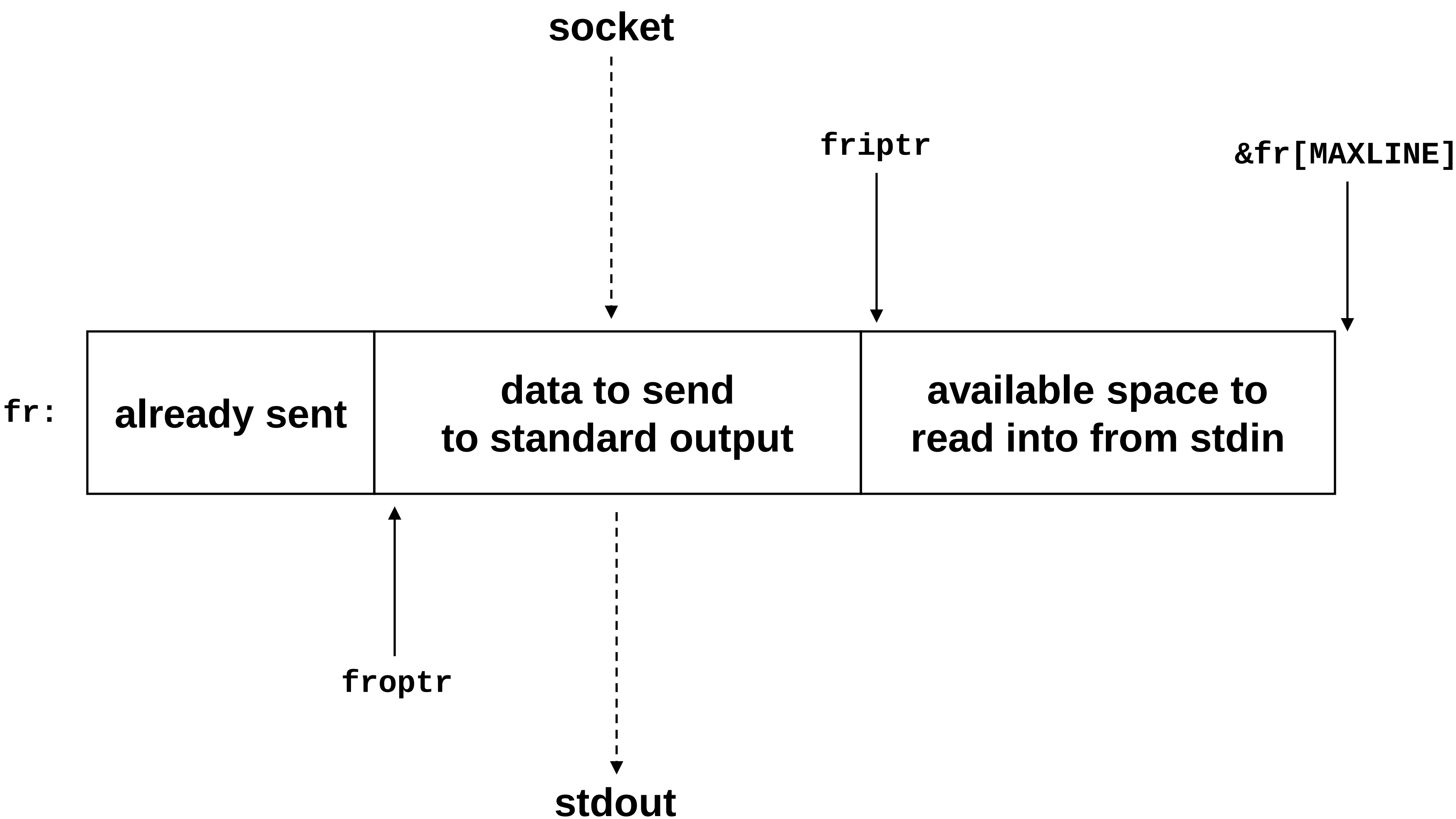
7.2 Nonblocking Reads and Writes: str_cli Fuction (Revisited)

Buffer containing data from standard input going to the socket



Nonblocking Reads and Writes: str_cli Fuction (Revisited)

Buffer containing data from the socket going to standard output



str_cli Function – [nonblock/strclinonb.c]

First part: initializes and calls select

- 10-15: set descriptors to nonblocking

```
01 #include  "unp.h"

02 void
03 str_cli(FILE *fp, int sockfd)
04 {
05     int     maxfdp1, val, stdineof;
06     ssize_t   n, nwritten;
07     fd_set rset, wset;
08     char       to[MAXLINE], fr[MAXLINE];
09     char       *toiptr, *tooptr, *friptr, *froptr;

10     val = Fcntl(sockfd, F_GETFL, 0);
11     Fcntl(sockfd, F_SETFL, val | O_NONBLOCK);
```

str_cli Function – [nonblock/strclinonb.c]

First part: initializes and calls select

- 16-19: initialize buffer pointers

```
12    val = Fcntl(STDIN_FILENO, F_GETFL, 0);
13    Fcntl(STDIN_FILENO, F_SETFL, val | O_NONBLOCK);

14    val = Fcntl(STDOUT_FILENO, F_GETFL, 0);
15    Fcntl(STDOUT_FILENO, F_SETFL, val | O_NONBLOCK);

16    toiptr = tooptr = to;/* initialize buffer pointers */
17    friptr = froptr = fr;
18    stdineof = 0;

19    maxfdp1 = max(max(STDIN_FILENO, STDOUT_FILENO),
              sockfd) + 1;
```

str_cli Function – [nonblock/strclinonb.c]

First part: initializes and calls select

- 20: main loop: prepare to call select
- 21-30: specify descriptors we are interested in

```
20      for ( ; ; ) {
21          FD_ZERO(&rset);
22          FD_ZERO(&wset);
23          if (stdineof == 0 && toiptr < &to[MAXLINE])
24              FD_SET(STDIN_FILENO, &rset);
                    /* read from stdin */
25          if (friptr < &fr[MAXLINE])
26              FD_SET(sockfd, &rset);
                    /* read from socket */
27          if (tooptr != toiptr)
28              FD_SET(sockfd, &wset);
                    /* data to write to socket */
29          if (froptr != friptr)
30              FD_SET(STDOUT_FILENO, &wset);
                    /* data to write to stdout */
```

str_cli Function – [nonblock/strclinonb.c]

First part: initializes and calls select

- 31: call select

Second part: reads from standard input or socket

- 32-33: read from standard input
- 34-35: handle nonblocking error

```
31        Select(maxfdp1, &rset, &wset, NULL, NULL);
32        if (FD_ISSET(STDIN_FILENO, &rset)) {
33            if ( (n = read(STDIN_FILENO, toiptr,
                             &to[MAXLINE] - toiptr)) < 0) {
34                if (errno != EWOULDBLOCK)
35                    err_sys("read error on stdin");
```

str_cli Function – [nonblock/strclinonb.c]

Second part: reads from standard input or socket

- 36-40: read returns EOF
- 41-45: read returns data

```
36          } else if (n == 0) {
37              fprintf(stderr, "%s: EOF on stdin\n",
                    gf_time());
38              stdineof = 1;    /* all done with stdin */
39              if (tooptr == toiptr)
40                  Shutdown(sockfd, SHUT_WR); /* FIN */

41          } else {
42              fprintf(stderr, "%s: read %d bytes from
43                  stdin\n", gf_time(), n);
44              toiptr += n;            /* # just read */
45              FD_SET(sockfd, &wset);
46          }
47      }
```

str_cli Function – [nonblock/strclinonb.c]

Second part: reads from standard input or socket

- 48-64: read from socket

```
48          if (FD_ISSET(sockfd, &rset)) {
49              if ( (n = read(sockfd, friptr, &fr[MAXLINE] -
                            friptr)) < 0) {
50                  if (errno != EWOULDBLOCK)
51                      err_sys("read error on socket");

52              } else if (n == 0) {
53                  fprintf(stderr, "%s: EOF on socket\n",
                            gf_time());
54                  if (stdineof)
55                      return;       /* normal termination */
56                  else
57                      err_quit("str_cli: server terminated
                            prematurely");
```

str_cli Function – [nonblock/strclinonb.c]

Second part: reads from standard input or socket

- 48-64: read from socket

```
58             } else {
59                 fprintf(stderr, "%s: read %d bytes from
                       socket\n", gf_time(), n);
60                 friptr += n;     /* # just read */
61                 FD_SET(STDOUT_FILENO, &wset);
62                     /* try and write below */
63             }
64         }
```

str_cli Function – [nonblock/strclinonb.c]

Third part: writes to standard ouput or socket

- 65-68: write to standard output
- 69-75: write OK

```
66         if (FD_ISSET(STDOUT_FILENO, &wset) && ( (n =
                  friptr - froptr) > 0)) {
67             if ( (nwritten = write(STDOUT_FILENO, froptr,
                      n)) < 0) {
68                 if (errno != EWOULDBLOCK)
69                     err_sys("write error to stdout");

70             } else {
```

str_cli Function – [nonblock/strclinonb.c]

Third part: writes to standard ouput or socket

- 65-68: write to standard output
- 69-75: write OK

```
71                  fprintf(stderr, "%s: wrote %d bytes to
                          stdout\n“, gf_time(), nwritten);
72                  froptr += nwritten; /* # just written */
73                  if (froptr == friptr)
74                      froptr = friptr = fr;
                          /* back to beginning of buffer */
75              }
76          }
```

str_cli Function – [nonblock/strclinonb.c]

Third part: writes to standard ouput or socket

- 77-91: write to socket

```
77          if (FD_ISSET(sockfd, &wset) && ( (n = toiptr -
                       tooptr) > 0)) {
78              if ( (nwritten = write(sockfd, tooptr, n)) < 0)
                 {
79                   if (errno != EWOULDBLOCK)
80                         err_sys("write error to socket");

81               } else {
82                   fprintf(stderr, "%s: wrote %d bytes to
83                         socket\n“, gf_time(), nwritten);
```

str_cli Function – [nonblock/strclinonb.c]

Third part: writes to standard ouput or socket

- 77-91: write to socket

```
84                  tooptr += nwritten; /* # just written */
85                  if (tooptr == toiptr) {
86                      toiptr = tooptr = to;
                            /* back to beginning of buffer */
87                      if (stdineof)
88                          Shutdown(sockfd, SHUT_WR);
                                /* send FIN */
89                  }
90              }
91          }
92      }
93  }
```

gf_time Function – [lib/gf_time.c]

Returns pointer to time string

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

char *
gf_time(void)
{
    struct timeval   tv;
    static char      str[30];
    char            *ptr;

    if (gettimeofday(&tv, NULL) < 0)
        err_sys("gettimeofday error");
```

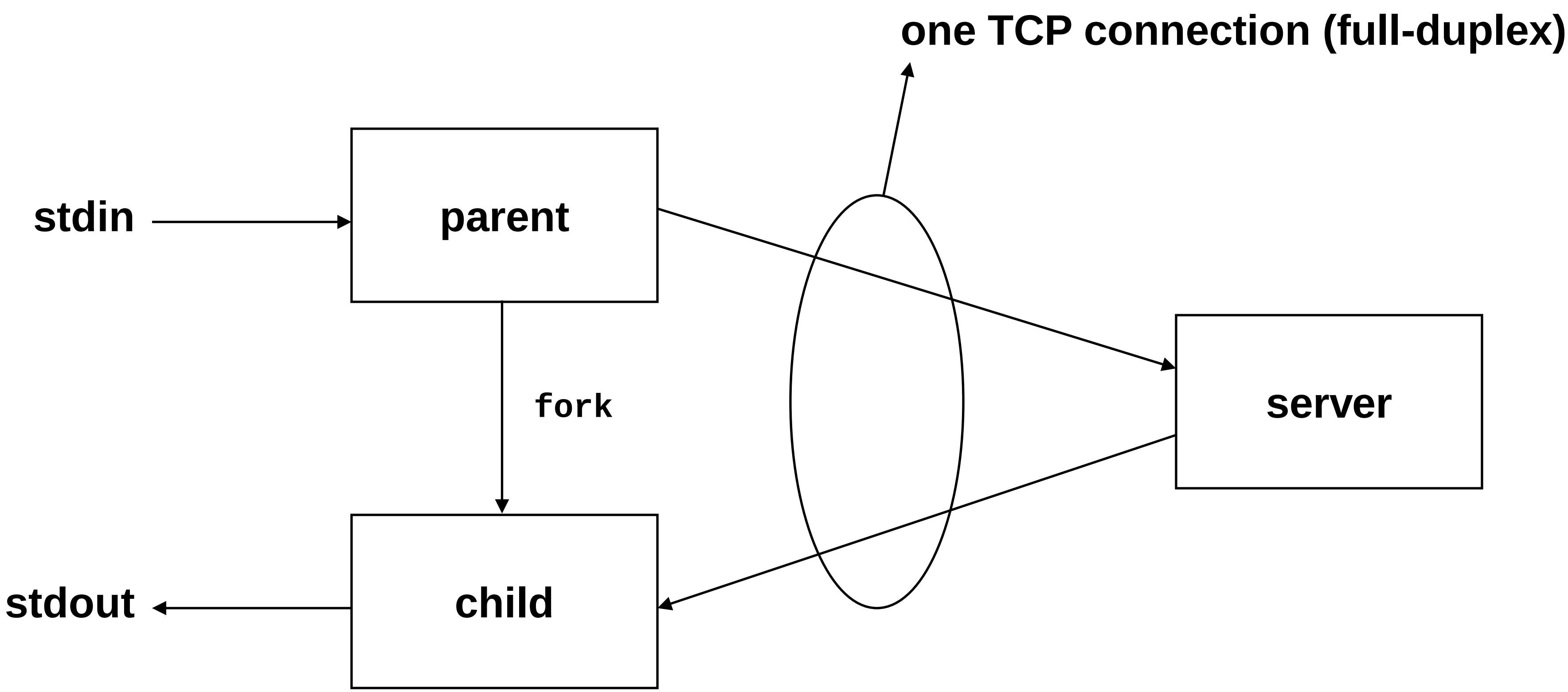
gf_time Function – [lib/gf_time.c]

Returns pointer to time string

```
    ptr = ctime(&t);
    strcpy(str, &ptr[11]);
        /* Fri Sep 13 00:00:00 1986\n\0 */
        /* 0123456789012345678901234 5  */
    snprintf(str+8, sizeof(str)-8, ".%06ld", tv.tv_usec);

    return(str);
}
```

A Simpler Version of str_cli

str_cli function using two processes



Version of str_cli Function that uses fork – [nonblock/strclifork.c]

```
#include  "unp.h"

void
str_cli(FILE *fp, int sockfd)
{
    pid_t  pid;
    char   sendline[MAXLINE], recvline[MAXLINE];

    if ( (pid = Fork()) == 0) {
            /* child: server -> stdout */
        while (Readline(sockfd, recvline, MAXLINE) > 0)
            Fputs(recvline, stdout);

        kill(getppid(), SIGTERM);
            /* in case parent still running */
        exit(0);
    }
```

Version of str_cli Function that uses fork – [nonblock/strclifork.c]

```
        /* parent: stdin -> server */
    while (Fgets(sendline, MAXLINE, fp) != NULL)
        Writen(sockfd, sendline, strlen(sendline));

    Shutdown(sockfd, SHUT_WR);
        /* EOF on stdin, send FIN */
    pause();
    return;
}
```

7.3 Nonblocking connect

Nonblocking connect

- TCP 소켓에 대해서 Nonblocking 셋팅
  - Connection 수행하면 바로 EINPROGRESS 에러 반환
  - Three-way handshake 수행
- Connection 완료는 select 함수를 사용해서 확인

Three uses for a nonblocking connect:

- TCP three-way handshake하는 동안 다른 일을 처리
- 동시에 다수의 Connection 설정
- Connection 완료를 select 함수에서 확인하므로 Timeout 설정 가능

Nonblocking connect

Other details we must handle:

- 같은 호스트에 서버가 동작할 경우, Connection이 즉시 완료됨
- 함수 select와 Nonblocking connect를 사용할 경우 규칙
  - Connection 설정이 완료된 경우에만 Writable함
  - Connection 설정이 실패할 경우, readable과 writable 가능

7.4 Nonblocking connect: Daytime client

TCP daytime client - [intro/daytimetcpcli.c]

- 18: replace connect with connect_nonb

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

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

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

Issue a Nonblocking connect – [lib/connect_nonb.c]

```
#include  "unp.h"

int
connect_nonb(int sockfd, const SA *saptr, socklen_t
        salen, int nsec)
{
    int          flags, n, error;
    socklen_t    len;
    fd_set       rset, wset;
    struct timeval   tval;
```

Issue a Nonblocking connect – [lib/connect_nonb.c]

9-14: set socket nonblocking

```
09      flags = Fcntl(sockfd, F_GETFL, 0);
10      Fcntl(sockfd, F_SETFL, flags | O_NONBLOCK);

11      error = 0;
12      if ( (n = connect(sockfd, saptr, salen)) < 0)
13          if (errno != EINPROGRESS)
14              return(-1);
```

Issue a Nonblocking connect – [lib/connect_nonb.c]

15: overlap processing with connection establishment

16-17: check for immediate completion

18-24: call select

25-28: handle timeouts

```
15      /* Do whatever we want while the connect is taking
           place. */

16      if (n == 0)
17          goto done;/* connect completed immediately */

18      FD_ZERO(&rset);
19      FD_SET(sockfd, &rset);
20      wset = rset;
21      tval.tv_sec = nsec;
22      tval.tv_usec = 0;
```

Issue a Nonblocking connect – [lib/connect_nonb.c]

15: overlap processing with connection establishment

16-17: check for immediate completion

18-24: call select

25-28: handle timeouts

```
23     if ( (n = Select(sockfd+1, &rset, &wset, NULL,
24                      nsec ? &tval : NULL)) == 0) {
25         close(sockfd);       /* timeout */
26         errno = ETIMEDOUT;
27         return(-1);
28     }
```

Issue a Nonblocking connect – [lib/connect_nonb.c]

29-34: check for readability or writability

36-42: turn off nonblocking and return

```
29      if (FD_ISSET(sockfd, &rset) || FD_ISSET(sockfd,
                &wset)) {
30          len = sizeof(error);
31          if (getsockopt(sockfd, SOL_SOCKET, SO_ERROR,
                    &error, &len) < 0)
32              return(-1);     /* Solaris pending error */
33      } else
34          err_quit("select error: sockfd not set");
```

Issue a Nonblocking connect – [lib/connect_nonb.c]

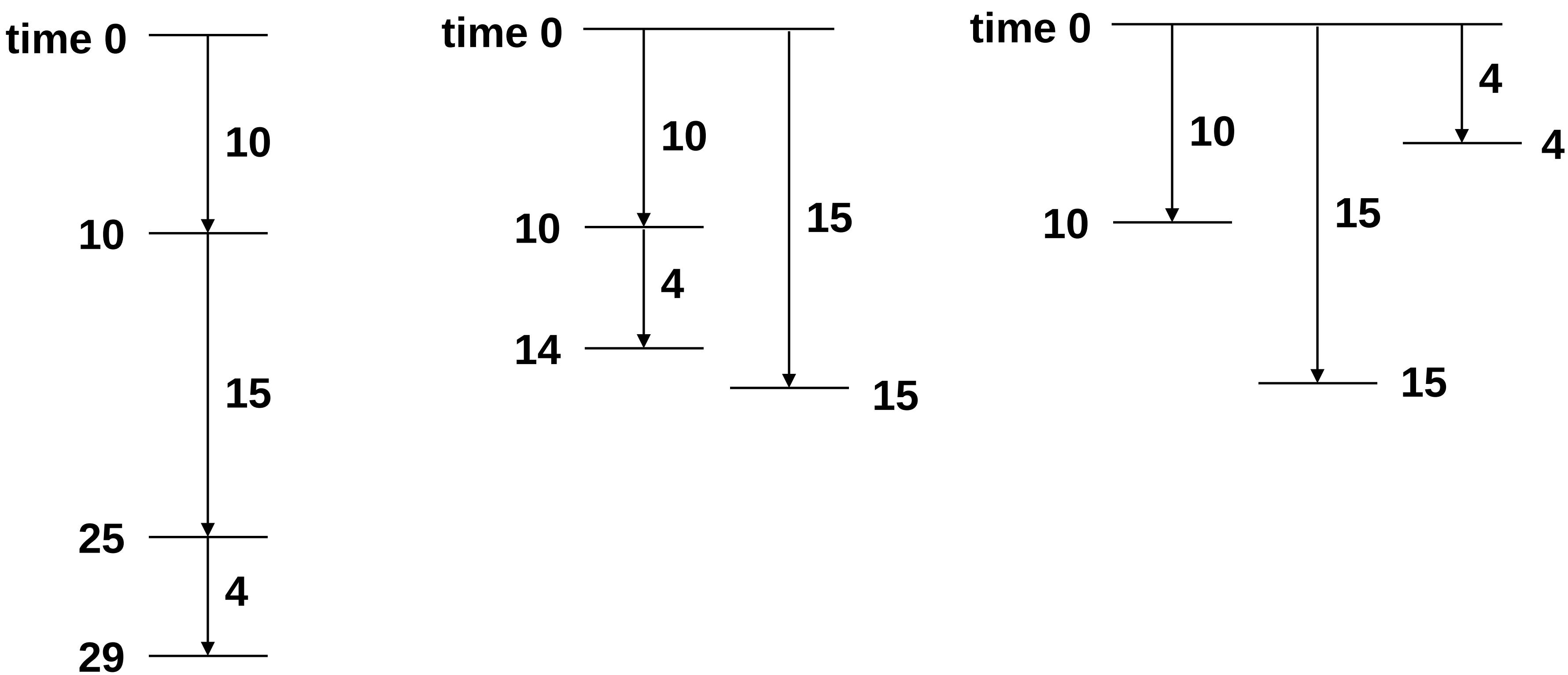
29-34: check for readability or writability

36-42: turn off nonblocking and return

```
35 done:
36     Fcntl(sockfd, F_SETFL, flags);
           /* restore file status flags */

37     if (error) {
38         close(sockfd);      /* just in case */
39         errno = error;
40         return(-1);
41     }
42     return(0);
43 }
```

7.5 Nonblocking connect: Web Client

Establishing multiple connections in parallel



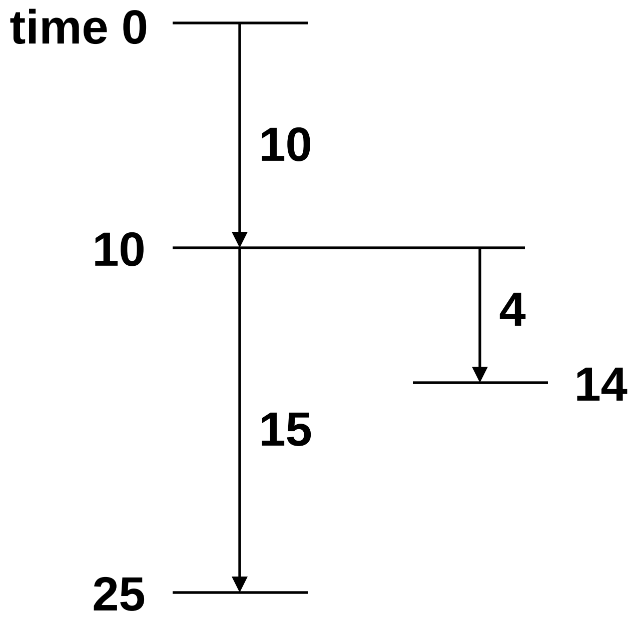
three connections done serially

three connections done in parallel; maximum of two connections at a time

three connections done in parallel; maximum of three connections at a time

Nonblocking connect: Web Client

Complete first connection, then multiple connections in parallel



web.h Header – [nonblock/web.h]

2-13: define file structure

```
01 #include  "unp.h"

02 #define   MAXFILES  20
03 #define   SERV      "80"/* port num or service name */

04 struct file {
05   char *f_name;     /* filename */
06   char *f_host;     /* hostname or IPv4/IPv6 address */
07   int    f_fd;             /* descriptor */
08   int   f_flags;        /* F_xxx below */
09 } file[MAXFILES];

10 #define   F_CONNECTING  1   /* connect() in progress */
11 #define   F_READING     2   /* connect() complete*/
12 #define   F_DONE        4   /* all done */

13 #define   GET_CMD       "GET %s HTTP/1.0\r\n\r\n"
```

web.h Header – [nonblock/web.h]

14-20: define globals and function prototypes

```
14              /* globals */
15 int    nconn, nfiles, nlefttoconn, nlefttoread, maxfd;
16 fd_set rset, wset;

17              /* function prototypes */
18 void    home_page(const char *, const char *);
19 void    start_connect(struct file *);
20 void    write_get_cmd(struct file *);
```

First part of simultaneous connect - [nonblock / web.c]

Globals and start of main

```
#include  "web.h"

int
main(int argc, char **argv)
{
    int     i, fd, n, maxnconn, flags, error;
    char    buf[MAXLINE];
    fd_set rs, ws;

    if (argc < 5)
        err_quit("usage: web <#conns> <hostname> <homepage> <file1> ...");
    maxnconn = atoi(argv[1]);
```

First part of simultaneous connect - [nonblock/web.c]

Globals and start of main

- 11-17: process command-line arguments

```
11     nfiles = min(argc - 4, MAXFILES);
12     for (i = 0; i < nfiles; i++) {
13         file[i].f_name = argv[i + 4];
14         file[i].f_host = argv[2];
15         file[i].f_flags = 0;
16     }
17     printf("nfiles = %d\n", nfiles);
```

First part of simultaneous connect - [nonblock / web.c]

Globals and start of main

- 18: read home page
- 19-23: initialize globals

```
18     home_page(argv[2], argv[3]);

19     FD_ZERO(&rset);
21     FD_ZERO(&wset);
21     maxfd = -1;
22     nlefttoread = nlefttoconn = nfiles;
23     nconn = 0;
```

Main Loop of main Function - [nonblock / web.c]

24-35: initiate another connection, if possible

36-38: select – wait for something to happen

```
24      while (nlefttoread > 0) {
25          while (nconn < maxnconn && nlefttoconn > 0) {
26                  /* find a file to read */
27              for (i = 0 ; i < nfiles; i++)
28                  if (file[i].f_flags == 0)
29                      break;
30              if (i == nfiles)
31                  err_quit("nlefttoconn = %d but nothing
                        found", nlefttoconn);
32              start_connect(&file[i]);
33              nconn++;
34              nlefttoconn--;
35          }
```

Main Loop of main Function - [nonblock/web.c]

24-35: initiate another connection, if possible

36-38: select – wait for something to happen

```
36          rs = rset;
37          ws = wset;
38          n = Select(maxfd+1, &rs, &ws, NULL, NULL);
```

Main Loop of main Function - [nonblock/web.c]

39-55: handle all ready descriptors

```
39         for (i = 0; i < nfiles; i++) {
40             flags = file[i].f_flags;
41             if (flags == 0 || flags & F_DONE)
42                 continue;
43             fd = file[i].f_fd;
44             if (flags & F_CONNECTING &&
45                 (FD_ISSET(fd, &rs) || FD_ISSET(fd, &ws))) {
46                 n = sizeof(error);
47                 if (getsockopt(fd, SOL_SOCKET, SO_ERROR,
                           &error, &n) < 0 ||
48                     error != 0) {
49                     err_ret("nonblocking connect failed
50                         for %s“, file[i].f_name);
51                 }
```

Main Loop of main Function - [nonblock / web.c]

39-55: handle all ready descriptors

```
52                  /* connection established */
53                  printf("connection established for %s\n",
                       file[i].f_name);
54                  FD_CLR(fd, &wset);
                       /* no more writeability    test */
55                  write_get_cmd(&file[i]);
                       /* write() the GET command */
```

Main Loop of main Function - [nonblock/web.c]

56-67: see if descriptor has data

```
56              } else if (flags & F_READING && FD_ISSET(fd,
                    &rs)) {
57                  if ( (n = Read(fd, buf, sizeof(buf))) == 0)
                    {
58                      printf("end-of-file on %s\n",
                                    file[i].f_name);
59                      Close(fd);
60                      file[i].f_flags = F_DONE;
                            /* clears F_READING */
```

Main Loop of main Function - [nonblock / web.c]

56-67: see if descriptor has data

```
61                      FD_CLR(fd, &rset);
62                      nconn--;
63                      nlefttoread--;
64                  } else {
65                      printf("read %d bytes from %s\n", n,
                                    file[i].f_name);
66                  }
67              }
68          }
69      }
70      exit(0);
71  }
```

home_page Function – [nonblock/home_page.c]

7: establish connection with server

8-17: send HTTP command to server, read reply

```
01 #include  "web.h"

02 void
03 home_page(const char *host, const char *fname)
04 {
05     int    fd, n;
06     char   line[MAXLINE];
07     fd = Tcp_connect(host, SERV);
           /* blocking connect() */
```

home_page Function – [nonblock/home_page.c]

7: establish connection with server

8-17: send HTTP command to server, read reply

```
08      n = snprintf(line, sizeof(line), GET_CMD, fname);
09      Writen(fd, line, n);
10      for ( ; ; ) {
11          if ( (n = Read(fd, line, MAXLINE)) == 0)
12              break;      /* server closed connection */

13          printf("read %d bytes of home page\n", n);
14          /* do whatever with data */
15      }
16      printf("end-of-file on home page\n");
17      Close(fd);
18  }
```

start_connect Function – [nonblock/start_connect.c]

Initiate nonblocking connect

- 7-13: create socket, set to nonblocking

```
01 #include  "web.h"

02 void
03 start_connect(struct file *fptr)
04 {
05     int           fd, flags, n;
06     struct addrinfo  *ai;

07     ai = Host_serv(fptr->f_host, SERV, 0, SOCK_STREAM);
```

start_connect Function – [nonblock/start_connect.c]

Initiate nonblocking connect

- 7-13: create socket, set to nonblocking

```
08      fd = Socket(ai->ai_family, ai->ai_socktype, ai->
                  ai_protocol);
09      fptr->f_fd = fd;
10      printf("start_connect for %s, fd %d\n", fptr->f_name,
                        fd);

11          /* Set socket nonblocking */
12      flags = Fcntl(fd, F_GETFL, 0);
13      Fcntl(fd, F_SETFL, flags | O_NONBLOCK);
```

start_connect Function – [nonblock/start_connect.c]

14-22: initiate nonblocking connect

23-24: handle connection complete

```
14         /* Initiate nonblocking connect to the server. */
15     if ( (n = connect(fd, ai->ai_addr, ai->ai_addrlen))
                   < 0) {
16         if (errno != EINPROGRESS)
17             err_sys("nonblocking connect error");
18         fptr->f_flags = F_CONNECTING;
19         FD_SET(fd, &rset);
               /* select for reading and writing */
20         FD_SET(fd, &wset);
21         if (fd > maxfd)
22             maxfd = fd;

23     } else if (n >= 0)      /* connect is already done */
24         write_get_cmd(fptr);/* write() the GET command */
25 }
```

write_get_cmd Function – [nonblock/write_get_cmd.c]

Send an HTTP GET command to the server

- 7-9: build command and send it
- 10-13: set flags

```
01 #include  "web.h"

02 void
03 write_get_cmd(struct file *fptr)
04 {
05     int     n;
06     char    line[MAXLINE];

07     n = snprintf(line, sizeof(line), GET_CMD, fptr->
               f_name);
08     Writen(fptr->f_fd, line, n);
09     printf("wrote %d bytes for %s\n", n, fptr->f_name);
```

write_get_cmd Function – [nonblock/write_get_cmd.c]

Send an HTTP GET command to the server

- 7-9: build command and send it
- 10-13: set flags

```
10     fptr->f_flags = F_READING; /* clears F_CONNECTING */

11     FD_SET(fptr->f_fd, &rset);
           /* will read server's reply */
12     if (fptr->f_fd > maxfd)
13         maxfd = fptr->f_fd;
14 }
```

host_serv Function – [lib/host_serv.c]

Returns: pointer to addrinfo structure if OK, NULL on error

```
#include  "unp.h"

struct addrinfo *
host_serv(const char *host, const char *serv, int family,
       int socktype)
{
    int             n;
    struct addrinfo  hints, *res;

    bzero(&hints, sizeof(struct addrinfo));
    hints.ai_flags = AI_CANONNAME;
        /* always return canonical name */
    hints.ai_family = family;
        /* AF_UNSPEC, AF_INET, AF_INET6, etc. */
    hints.ai_socktype = socktype;
        /* 0, SOCK_STREAM, SOCK_DGRAM, etc. */
```

host_serv Function – [lib/host_serv.c]

Returns: pointer to addrinfo structure if OK, NULL on error

```
    if ( (n = getaddrinfo(host, serv, &hints, &res)) != 0)
        return(NULL);

    return(res);
        /* return pointer to first on linked list */
}
```

getaddrinfo and freeaddrinfo Functions

```
#include <netdb.h>

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

Returns: 0 if OK, nonzero on error

```
void freeaddrinfo(struct addrinfo *ai);
```

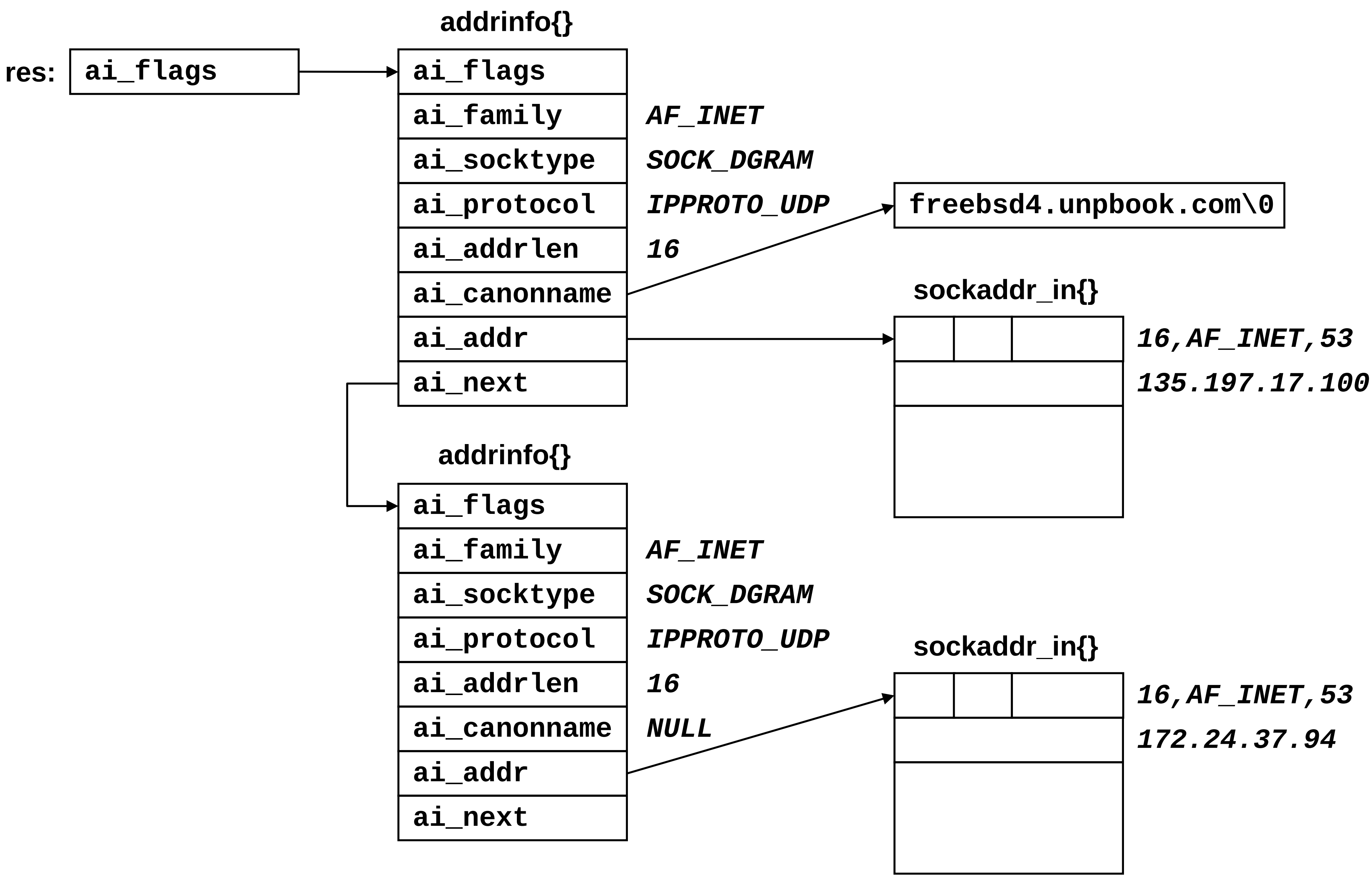
struct addrinfo

```
struct addrinfo {
   int         ai_flags;    /* AI_PASSIVE, AI_CONONNAME */
   int            ai_family;   /* AF_xxx */
   int            ai_socktype;    /* SOCK_xxx */
   int            ai_protocol;
      /* 0 or IPPROTO_xxx for IPv4 and IPv6 */
   socklen_t ai_addrlen;      /* length of ai_addr */
   char       *ai_canonname;
      /* ptr to canonical name for host */
   struct sockaddr  *ai_addr;
      /* ptr to socket address structur */
   struct addrinfo  *ai_next;
      /* prt to next struct in linked list */
};
```

struct addrinfo

```
struct addrinfo  hints, *res;

bzero(&hints, sizeof(hints));
hints.ai_flags = AI_CANONNAME;
hints.ai_family = AF_INET;

getaddrinfo(“freebsd4”, “domain”, &hints, &res);
```

Example of Information Returned by getaddrinfo



tcp_connect Function – [lib/tcp_connect.c]

Returns: connected socket descriptor if OK, no return on error

```
#include  "unp.h"

int
tcp_connect(const char *host, const char *serv)
{
    int              sockfd, n;
    struct addrinfo  hints, *res, *ressave;

    bzero(&hints, sizeof(struct addrinfo));
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;

   if ( (n = getaddrinfo(host, serv, &hints, &res)) != 0)
        err_quit("tcp_connect error for %s, %s: %s",
                 host, serv, gai_strerror(n));
    ressave = res;
```

tcp_connect Function – [lib/tcp_connect.c]

14-25: try each addrinfo structure until success or end of list

```
14      do {
15          sockfd = socket(res->ai_family, res->ai_socktype,
                    res->ai_protocol);
16          if (sockfd < 0)
17              continue; /* ignore this one */

18          if (connect(sockfd, res->ai_addr, res->ai_addrlen)
                    == 0)
19              break;    /* success */

20          Close(sockfd);   /* ignore this one */
21      } while ( (res = res->ai_next) != NULL);
```

tcp_connect Function – [lib/tcp_connect.c]

14-25: try each addrinfo structure until success or end of list

```
22     if (res == NULL) /* errno set from final connect() */
23         err_sys("tcp_connect error for %s, %s", host,
                serv);

24     freeaddrinfo(ressave);

25     return(sockfd);
26 }
```

7.6 Nonblocking accept

함수 accept를 select와 함께 사용

- 함수 accept에서 Blocking되는 것을 피할 수 있음

이 경우에 Timing 문제 발생 가능

- 클라이언트가 Connection을 설정하고 해제함
- 서버는 select에서 리턴된 후 accept를 수행하기 전의 상태가 됨
- 서버가 RST 메시지를 수신
- 설정 완료된 Connection이 Queue에서 제거되고 더 이상 Connection이 없음
- 서버가 accept를 호출하지만 설정 완료된 Connection이 없으므로 Block 됨

Timing 문제 해결 방안

- 함수 accept와 select를 같이 사용할 경우 항상 listening 소켓을 nonblocking으로 셋팅
- 다음의 에러 상황 무시
  - EWOULDBLOCK, ECONNABORTED, EPROTO, EINTR

Example of Timing Problem

Timing 문제 확인 예제

- 함수 accept를 select와 함께 사용하는 TCP 서버에서 select 반환 이후 accept를 호출하기 전에 sleep(5)를 삽입
- TCP 클라이언트가 Connection을 설정하고 RST를 서버에 송신

TCP echo client that creates connection and sends an RST

```
#include  "unp.h"

int
main(int argc, char **argv)
{
    int                 sockfd;
    struct linger    ling;
    struct sockaddr_in  servaddr;

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

    sockfd = 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);
```

TCP echo client that creates connection and sends an RST

16-19: set SO_LINGER socket option

```
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  }
```

실습 과제

Nonblocking I/O를 사용한 DNS 서버 작성

Nonblocking I/O를 사용한 DNS 클라이언트 작성

- Nonblocking connect 기법도 사용

8. THREADS

목 차

8.1 Introduction

프로세스 fork 사용시 문제점들

- fork is expensive
- IPC is required

한 프로세스 내의 모든 쓰레드가 공유하는 정보

- Process instructions
- Most data (global variable 포함)
- Open files (e.g., descriptors)
- Signal handlers and signal dispositions
- Current working directory
- User and group IDs

각 쓰레드가 따로 가지는 정보

- Thread ID
- Set of registers, including program counter and stack pointer
- Stack (for local variables and return address)
- errno
- Signal mask
- Priority

8.2 Basic Thread Functions: Creation and Termination

```
#include <pthread.h>

int pthread_create(pthread_t *tid, const pthread_attr_t
       *attr, void *(*func)(void *), void *arg);
```

Returns: 0 if OK, positive Exxx value on error

```
int pthread_join(pthread_t *tid, void **status);
```

Returns: 0 if OK, positive Exxx value on error

Basic Thread Functions: Creation and Termination

```
#include <pthread.h>

pthread_t pthread_self(void);
```

Returns: thread ID of calling thread

```
int pthread_detach(pthread_t tid);
```

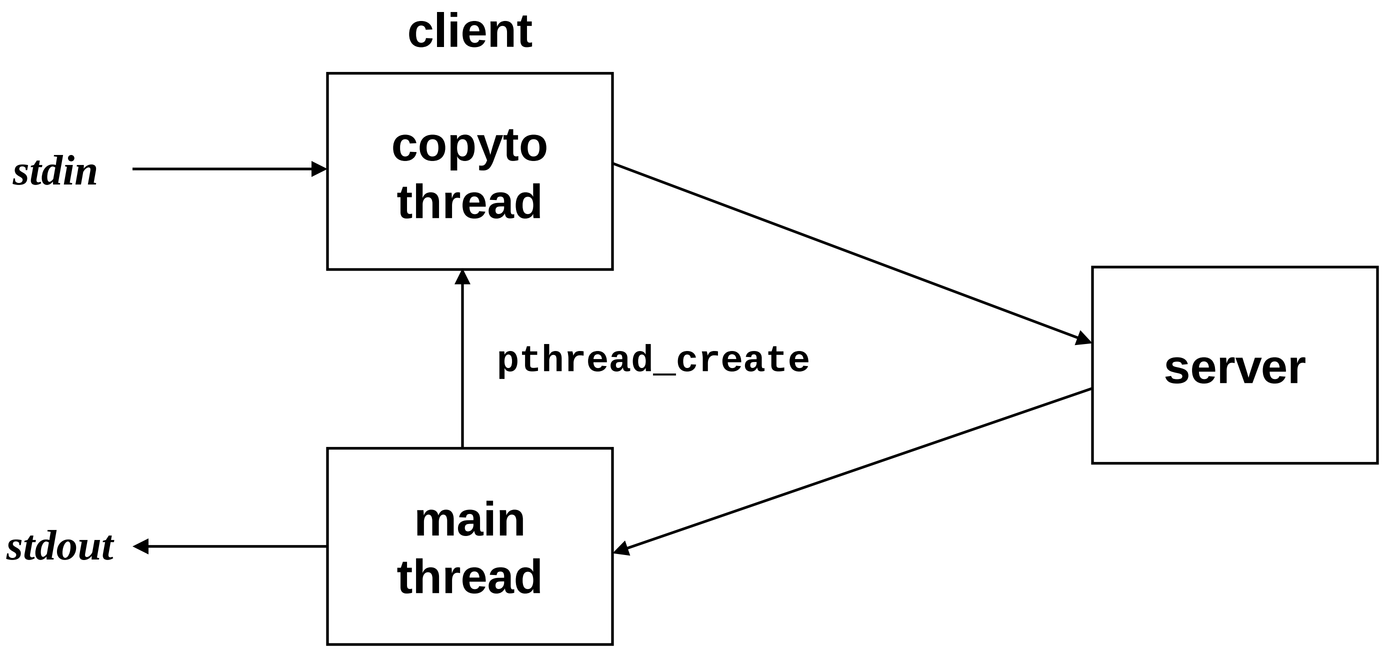
Returns: 0 if OK, positive Exxx value on error

```
void pthread_exit(void *status);
```

Does not return to caller

8.3 str_cli Function Using Threads

Recoding str_cli to use threads



str_cli Function Using Threads – [threads/strclithread.c]

1: unpthread.h header

10-11: save arguments in externals

```
01 #include  "unpthread.h"

02 void   *copyto(void *);

03 static intsockfd;/* global for both threads to access */
04 static FILE   *fp;

05 void
06 str_cli(FILE *fp_arg, int sockfd_arg)
07 {
08     char      recvline[MAXLINE];
09     pthread_t tid;

10     sockfd = sockfd_arg;/* copy arguments to externals */
11     fp = fp_arg;
```

str_cli Function Using Threads – [threads/strclithread.c]

12: create new thread

13-14: main thread loop: copy socket to standard output

15: terminate

```
12      Pthread_create(&tid, NULL, copyto, NULL);

13      while (Readline(sockfd, recvline, MAXLINE) > 0)
14          Fputs(recvline, stdout);
15  }
```

str_cli Function Using Threads – [threads/strclithread.c]

16-25: copy thread

```
16 void *
17 copyto(void *arg)
18 {
19     char   sendline[MAXLINE];

20     while (Fgets(sendline, MAXLINE, fp) != NULL)
21         Writen(sockfd, sendline, strlen(sendline));

22     Shutdown(sockfd, SHUT_WR); /* EOF stdin, send FIN */

23     return(NULL);
24         /* return when EOF on stdin */
25 }
```

8.4 TCP Echo Server Using Threads – [threads/tcpserv01.c]

```
#include  "unpthread.h"

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

int
main(int argc, char **argv)
{
    int            listenfd, connfd;
    pthread_t      tid;
    socklen_t      addrlen, len;
    struct sockaddr  *cliaddr;
```

TCP Echo Server Using Threads – [threads/tcpserv01.c]

```
    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);
```

TCP Echo Server Using Threads – [threads/tcpserv01.c]

17-21: create thread

23-30: thread function

```
17      for ( ; ; ) {
18          len = addrlen;
19          connfd = Accept(listenfd, cliaddr, &len);
20         Pthread_create(&tid, NULL, &doit, (void *) connfd);
21      }
22  }

23  static void *
24  doit(void *arg)
25  {
26      Pthread_detach(pthread_self());
27      str_echo((int) arg);/* same function as before */
28      Close((int) arg);   /* done with connected socket */
29      return(NULL);
30  }
```

tcp_listen Function – [lib / tcp_connect.c]

Returns: connected socket descriptor if OK, no return on error

- 8-15: call getaddrinfo

```
01 #include  "unp.h"

02 int
03 tcp_listen(const char *host, const char *serv, socklen_t
          *addrlenp)
04 {
05     int             listenfd, n;
06     const int       on = 1;
07     struct addrinfo  hints, *res, *ressave;
```

tcp_listen Function – [lib/tcp_connect.c]

Returns: connected socket descriptor if OK, no return on error

- 8-15: call getaddrinfo

```
08     bzero(&hints, sizeof(struct addrinfo));
09     hints.ai_flags = AI_PASSIVE;
10     hints.ai_family = AF_UNSPEC;
11     hints.ai_socktype = SOCK_STREAM;

12    if ( (n = getaddrinfo(host, serv, &hints, &res)) != 0)
13         err_quit("tcp_listen error for %s, %s: %s",
14                 host, serv, gai_strerror(n));
15     ressave = res;
```

tcp_listen Function – [lib / tcp_connect.c]

16-25: create socket and bind address

```
16     do {
17        listenfd = socket(res->ai_family, res->ai_socktype,
18               res->ai_protocol);
19         if (listenfd < 0)
20               continue;     /* error, try next one */

21        Setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, &on,
               sizeof(on));
22         if (bind(listenfd, res->ai_addr, res->ai_addrlen)
                  == 0)
23               break;        /* success */

24         Close(listenfd);
               /* bind error, close and try next one */
25     } while ( (res = res->ai_next) != NULL);
```

tcp_listen Function – [lib / tcp_connect.c]

26-27: check for failure

28: call listen

29-32: return size of socket address structure

```
26     if (res == NULL) /* errno from socket() or bind() */
27         err_sys("tcp_listen error for %s, %s", host, serv);

28     Listen(listenfd, LISTENQ);

29     if (addrlenp)
30         *addrlenp = res->ai_addrlen;
               /* return size of protocol addr */

31     freeaddrinfo(ressave);

32     return(listenfd);
33 }
```

Passing Arguments to New Threads

```
int
main(int argc, char **argv)
{
    int    listenfd, connfd;
    ...
    for ( ; ; ) {
        len = addrlen;
        connfd = accept(listenfd, cliaddr, &len);

        pthread_create(&tid, NULL, &doit, &connfd);
    }
}
```

Passing Arguments to New Threads

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

    connfd = *((int *) arg;
    pthread_detach(pthread_self());
    str_echo(connfd);        /* same function as before */
    close(connfd);      /* done with connected socket */
    return(NULL);
}
```

TCP Echo Server Using Threads – [threads/tcpserv02.c]

More Portable Argument Passing

```
#include  "unpthread.h"

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

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

TCP Echo Server Using Threads – [threads/tcpserv02.c]

More Portable Argument Passing

```
    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);
```

TCP Echo Server Using Threads – [threads/tcpserv02.c]

```
 for ( ; ; ) {
        len = addrlen;
        iptr = Malloc(sizeof(int));
        *iptr = Accept(listenfd, cliaddr, &len);
        Pthread_create(&tid, NULL, &doit, iptr);
   }
}
```

TCP Echo Server Using Threads – [threads/tcpserv02.c]

```
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);       /* done with connected socket */
    return(NULL);
}
```

8.5 Thread-Specific Data

Common problem is due to static variables

- Use thread-specific data
- Change the calling sequence so that the caller packages all the arguments into a structure.
- Restructure the interface to avoid any static variables

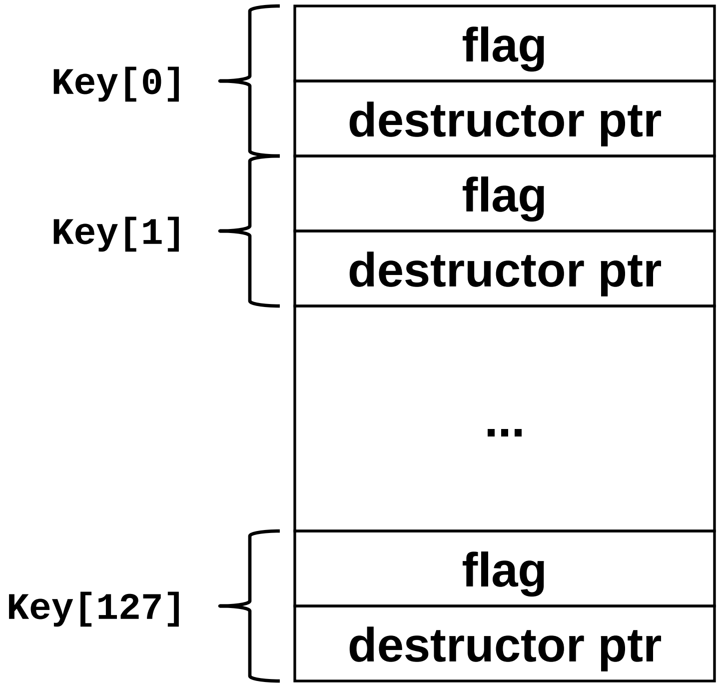
8.5 Thread-Specific Data

Data structure and function prototype for re-enterant version of readline

```
typedef struct {
   int        read_fd;      /* caller’s descriptor to
read from */
   char    *read_ptr;/* caller’s buffer to read into */
   size_t read_maxlen;/* caller’s max # bytes to read */
             /* next three are used internally by the
function */
   int        r1_cnt;       /* initialize to 0 */
   char    *r1_bufptr;   /* initialize to r1_buf */
   char    r1_buf[MAXLINE];
} Rline;

void   readline_rinit(int, void *, size_t, Rline *);
ssize_t   readline_r(Rline *);
ssize_t   Readline_r(Rline *);
```

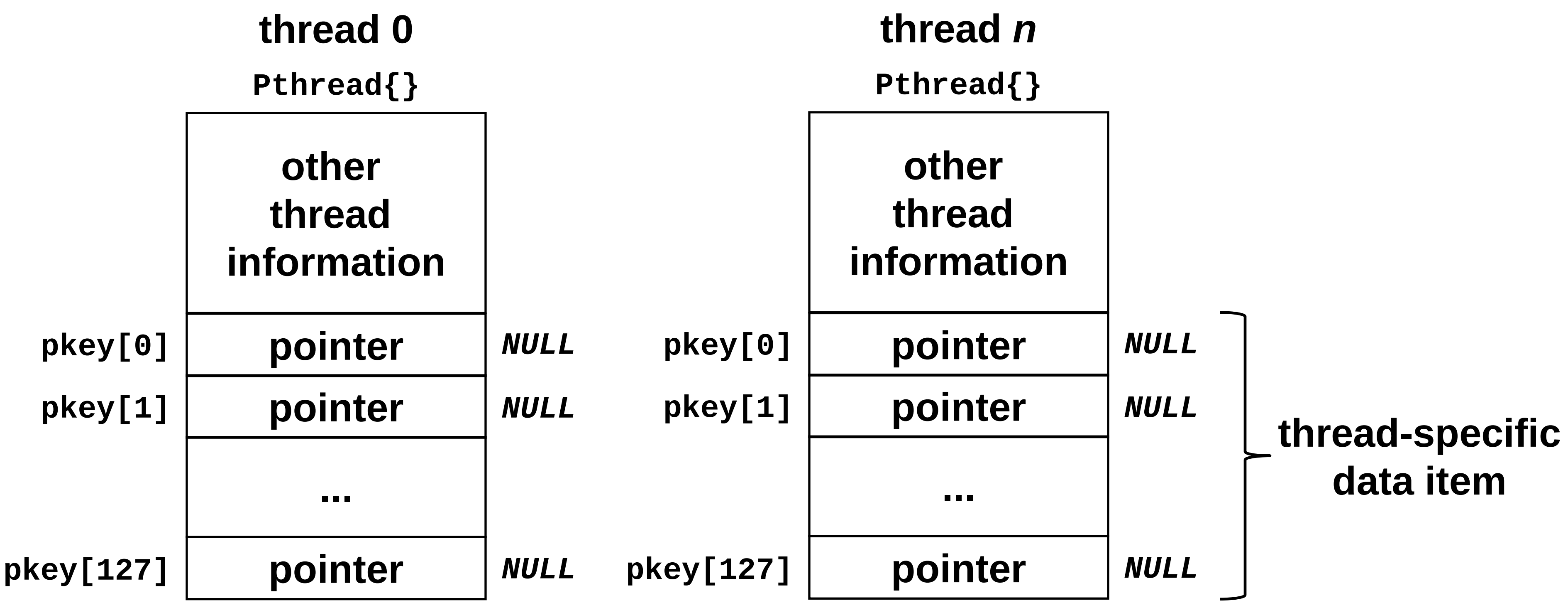
Possible Implementation of Thread-Specific Data

Process 당 하나만 존재하는 정보



Information Maintained by the System about Each Thread

쓰레드 당 하나씩 존재하는 정보



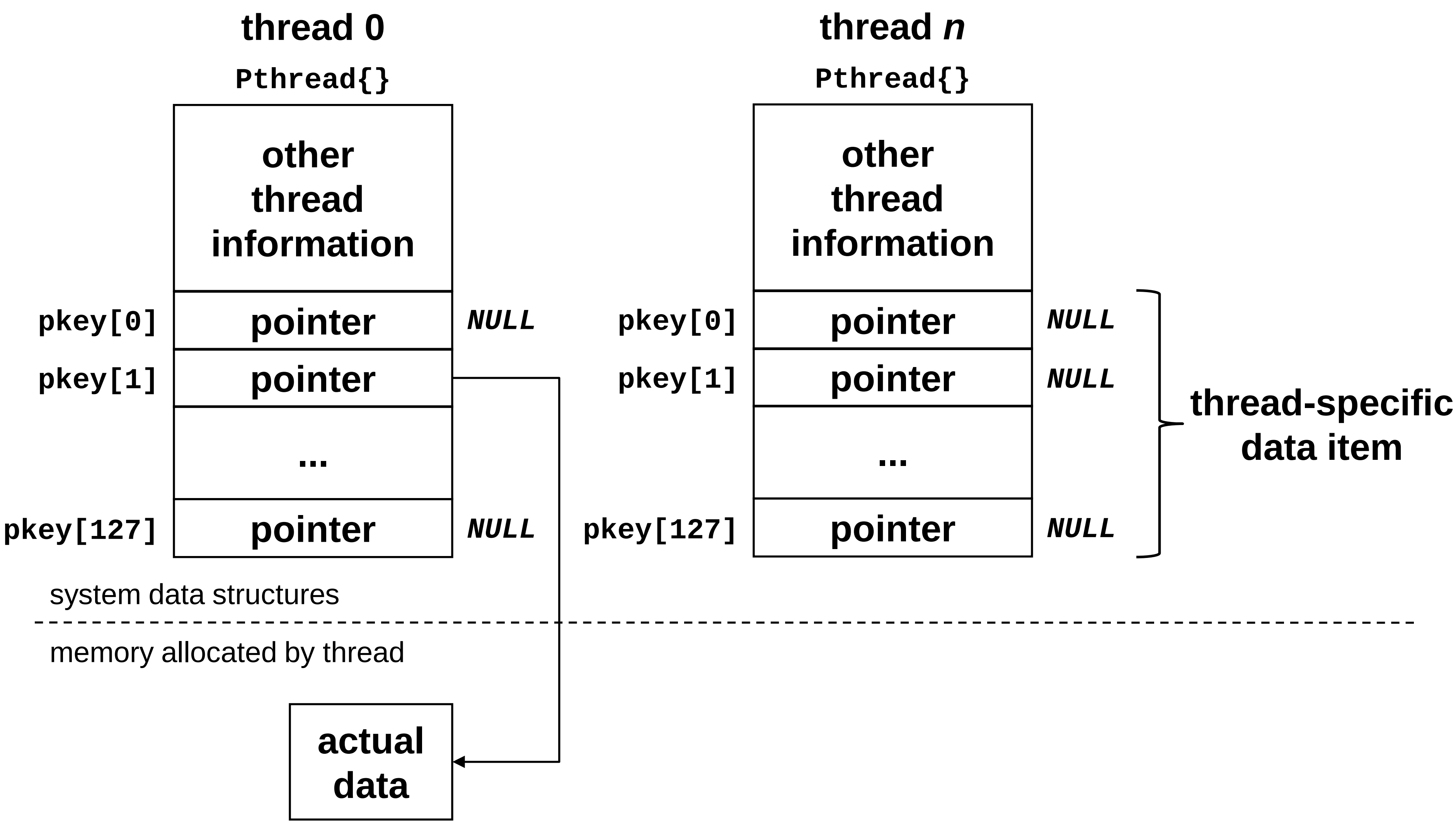
Example for Thread-Specific Data

Associating malloced region with thread-specific data pointer

- 프로세스가 시작되고 다수의 쓰레드 생성
- 0번 쓰레드가 readline 함수를 수행하면서 pthread_key_create 호출
  - First unused Key 구조체를 찾음 – Key[1]
- pthread_once 함수 호출
  - 같은 Key 값에 대해서 pthread_key_create를 맨 처음 호출한 쓰레드만 해당 함수 수행
- pthread_getspecific를 호출함으로써 pkey[1] 값을 확인
  - NULL이라면 실제 데이터를 위한 메모리 공간 할당
  - pthread_setspecific 함수를 통해서 할당된 메모리에 대한 포인터 값을 셋팅

Example for Thread-Specific Data (계속)

Associating malloced region with thread-specific data pointer



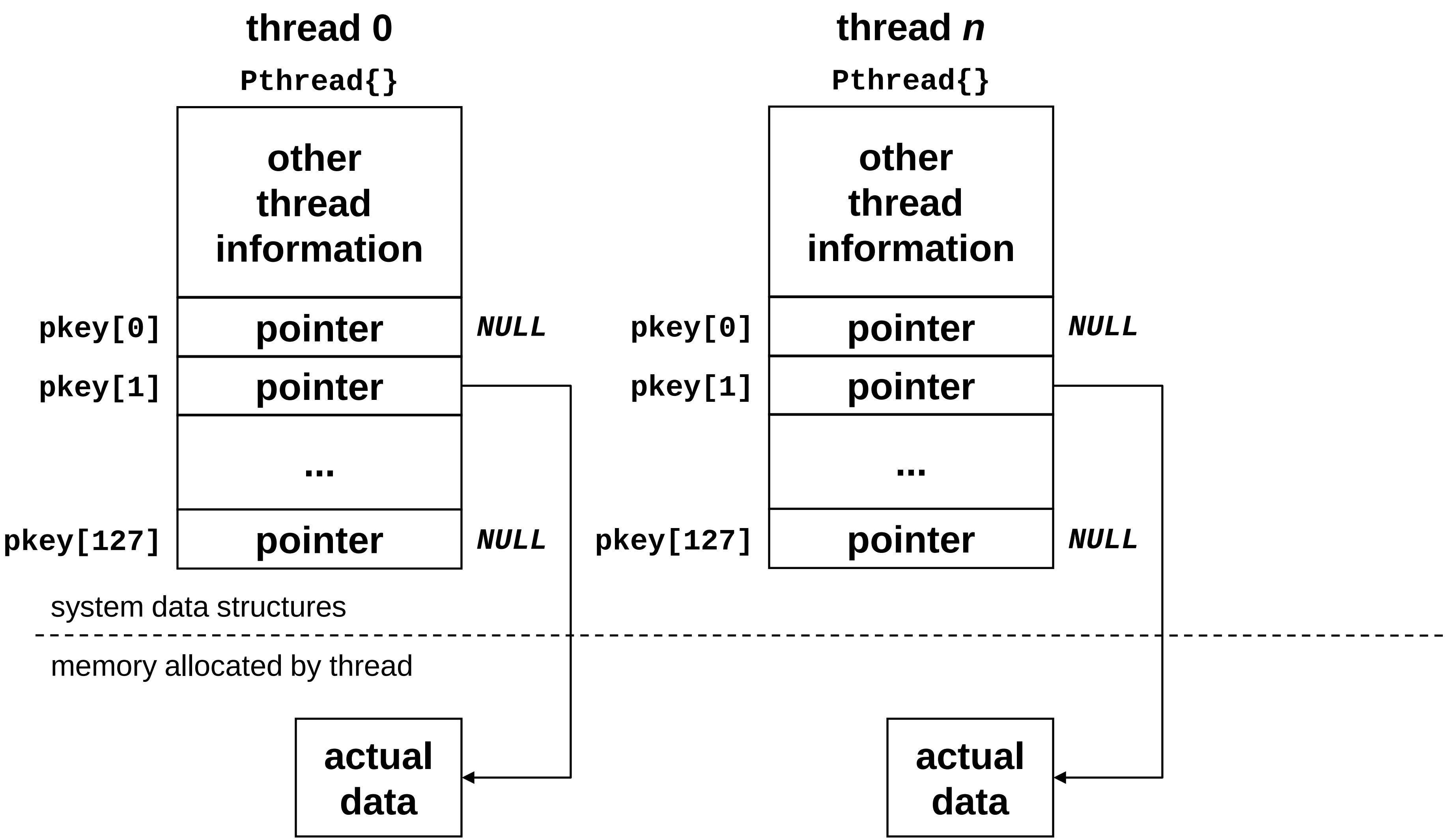
Example for Thread-Specific Data (계속)

Data structures after thread n initializes its thread-specific data

- n번 쓰레드가 readline 함수를 수행하면서 pthread_key_create 호출
  - First unused Key 구조체를 찾음 – Key[1]
- pthread_getspecific를 호출함으로써 pkey[1] 값을 확인
  - NULL이라면 실제 데이터를 위한 메모리 공간 할당
  - pthread_setspecific 함수를 통해서 할당된 메모리에 대한 포인터 값을 셋팅

Example for Thread-Specific Data (계속)

Data structures after thread n initializes its thread-specific data



Functions for Thread-Specific Data

```
#include <pthread.h>

int pthread_once(pthread_once_t *onceptr, void
        (*init)(void));

int pthread_key_create(pthread_key_t *keyptr, void
        (*destructor)(void *value));
```

Both return: 0 if OK, positive Exxx value on error

```
void pthread_getspecific(pthread_key_t key);
```

Returns: pointer to thread-specific data (possibly a null pointer)

```
int pthread_setspecific(pthread_key_t key, const void
        *value);
```

Returns: 0 if OK, positive Exxx value on error

Thread-Safe readline Function - [threads/readline.c]

More Portable Argument Passing

- 4-8: destructor, 9-13: one-time function, 14-18: Rline structure

```
01 #include  "unpthread.h"

02 static pthread_key_trl_key;
03 static pthread_once_t   rl_once = PTHREAD_ONCE_INIT;

04 static void
05 readline_destructor(void *ptr)
06 {
07     free(ptr);
08 }
```

Thread-Safe readline Function - [threads/readline.c]

More Portable Argument Passing

- 4-8: destructor, 9-13: one-time function, 14-18: Rline structure

```
09 static void
10 readline_once(void)
11 {
12     Pthread_key_create(&rl_key, readline_destructor);
13 }

14 typedef struct {
15   int   rl_cnt;          /* initialize to 0 */
16   char *rl_bufptr;          /* initialize to rl_buf */
17   char  rl_buf[MAXLINE];
18 } Rline;
```

Thread-Safe readline Function - [threads/readline.c]

19-35: my_read function

```
19 static ssize_t
20 my_read(Rline *tsd, int fd, char *ptr)
21 {
22     if (tsd->rl_cnt <= 0) {
23 again:
24         if ( (tsd->rl_cnt = read(fd, tsd->rl_buf, MAXLINE))
                < 0) {
25             if (errno == EINTR)
26                 goto again;
27             return(-1);
```

Thread-Safe readline Function - [threads/readline.c]

19-35: my_read function

```
28          } else if (tsd->rl_cnt == 0)
29              return(0);
30          tsd->rl_bufptr = tsd->rl_buf;
31      }

32      tsd->rl_cnt--;
33      *ptr = *tsd->rl_bufptr++;
34      return(1);
35  }
```

Thread-Safe readline Function - [threads/readline.c]

42: allocate thread-specific data

43-46: fetch thread-specific data pointer

```
36 ssize_t
37 readline(int fd, void *vptr, size_t maxlen)
38 {
39     size_t    n, rc;
40     char   c, *ptr;
41     Rline  *tsd;

42     Pthread_once(&rl_once, readline_once);
43     if ( (tsd = pthread_getspecific(rl_key)) == NULL) {
44         tsd = Calloc(1, sizeof(Rline));   /* init to 0 */
45         Pthread_setspecific(rl_key, tsd);
46     }
```

Thread-Safe readline Function - [threads/readline.c]

```
    ptr = vptr;
    for (n = 1; n < maxlen; n++) {
        if ( (rc = my_read(tsd, fd, &c)) == 1) {
            *ptr++ = c;
            if (c == '\n')
                break;
        } else if (rc == 0) {
            *ptr = 0;
            return(n - 1);   /* EOF, n - 1 bytes read */
        } else
            return(-1);   /* error, errno set by read() */
    }

    *ptr = 0;
    return(n);
}
```

8.6 Web Client and Simultaneous Connections (Continued) – [threads/web01.c]

```
#include  "unpthread.h"
#include  <thread.h>    /* Solaris threads */

#define   MAXFILES  20
#define   SERV      "80"   /* port number or service name */

struct file {
   char   *f_name;        /* filename */
   char   *f_host;        /* hostname or IP address */
   int    f_fd;           /* descriptor */
   int f_flags;        /* F_xxx below */
   pthread_t  f_tid;      /* thread ID */
} file[MAXFILES];
```

8.6 Web Client and Simultaneous Connections (Continued) – [threads/web01.c]

```
#define   F_CONNECTING 1   /* connect() in progress */
#define   F_READING    2   /* connect() complete; now
reading */
#define   F_DONE       4   /* all done */

#define   GET_CMD      "GET %s HTTP/1.0\r\n\r\n"

int    nconn, nfiles, nlefttoconn, nlefttoread;

void   *do_get_read(void *);
void   home_page(const char *, const char *);
void   write_get_cmd(struct file *);
```

Web Client and Simultaneous Connections (Continued) – [threads/web01.c]

36: home_page function is unchaged

```
int
main(int argc, char **argv)
{
    int        i, n, maxnconn;
    pthread_t tid;
    struct file  *fptr;

    if (argc < 5)
        err_quit("usage: web <#conns> <IPaddr> <homepage> file1 ...");
    maxnconn = atoi(argv[1]);
```

Web Client and Simultaneous Connections (Continued) – [threads/web01.c]

36: home_page function is unchaged

```
29     nfiles = min(argc - 4, MAXFILES);
30     for (i = 0; i < nfiles; i++) {
31         file[i].f_name = argv[i + 4];
32         file[i].f_host = argv[2];
33         file[i].f_flags = 0;
34     }
35     printf("nfiles = %d\n", nfiles);

36     home_page(argv[2], argv[3]);

37     nlefttoread = nlefttoconn = nfiles;
38     nconn = 0;
```

Web Client and Simultaneous Connections (Continued) – [threads/web01.c]

40-52: if possible, create another thread

```
39     while (nlefttoread > 0) {
40         while (nconn < maxnconn && nlefttoconn > 0) {
41                 /* 4find a file to read */
42             for (i = 0 ; i < nfiles; i++)
43                 if (file[i].f_flags == 0)
44                     break;
45             if (i == nfiles)
46                 err_quit("nlefttoconn = %d but nothing
found", nlefttoconn);

47             file[i].f_flags = F_CONNECTING;
48             Pthread_create(&tid, NULL, &do_get_read,
&file[i]);
49             file[i].f_tid = tid;
50             nconn++;
51             nlefttoconn--;
52         }
```

Web Client and Simultaneous Connections (Continued) – [threads/web01.c]

53-54: wait for any thread to terminate

```
53 if ( (n = thr_join(0, &tid, (void **) &fptr)) != 0)
54           errno = n, err_sys("thr_join error");

55       nconn--;
56       nlefttoread--;
57       printf("thread id %d for %s done\n", tid,
                   fptr->f_name);
58    }

59    exit(0);
60 }
```

Web Client and Simultaneous Connections (Continued) – [threads/web01.c]

68-71: create TCP socket, establish connection

```
61 void *
62 do_get_read(void *vptr)
63 {
64     int                 fd, n;
65     char                line[MAXLINE];
66     struct file             *fptr;

67     fptr = (struct file *) vptr;

68     fd = Tcp_connect(fptr->f_host, SERV);
69     fptr->f_fd = fd;
70     printf("do_get_read for %s, fd %d, thread %d\n",
71            fptr->f_name, fd, fptr->f_tid);
```

Web Client and Simultaneous Connections (Continued) – [threads/web01.c]

72: write request to server

73-82: read server’s reply

```
72      write_get_cmd(fptr);/* write() the GET command */

73          /* Read server's reply */
74      for ( ; ; ) {
75          if ( (n = Read(fd, line, MAXLINE)) == 0)
76              break;    /* server closed connection */

77        printf("read %d bytes from %s\n", n, fptr->f_name);
78      }
79      printf("end-of-file on %s\n", fptr->f_name);
80      Close(fd);
81      fptr->f_flags = F_DONE;    /* clears F_READING */

82      return(fptr);    /* terminate thread */
83  }
```

8.7 Mutexes: Mutual Exclusion

두 쓰레드가 전역변수를 공유할 경우 문제 발생 가능한 시나리오

- 쓰레드 A가 동작중이고 전역변수 nconn 값(3)을 레지스터로 로딩
- 쓰레드 A에서 B로 스위칭
- 쓰레드 B가 nconn--; 를 수행하고 해당 값(2)를 저장
- 쓰레드 B에서 A로 스위칭
- 쓰레드 A가 nconn--; 를 수행하고 해당 값(2)를 저장

위의 경우 nconn의 값은 1이어야 함

- 이러한 문제는 드물게 발생하지만 치명적임, 디버깅 난해함

Mutex를 사용하면 문제 해결 가능함

- Mutex: 쓰레드는 mutex를 가지고 있을 때만 특정 변수 접근 가능

Mutex Example – [threads/example01.c]

Two threads that increment a global variable incorrectly

```
#include  "unpthread.h"

#define   NLOOP 5000

int           counter;     /* incremented by threads */

void   *doit(void *);

int
main(int argc, char **argv)
{
    pthread_t tidA, tidB;
```

Mutex Example – [threads/example01.c]

Two threads that increment a global variable incorrectly

```
    Pthread_create(&tidA, NULL, &doit, NULL);
    Pthread_create(&tidB, NULL, &doit, NULL);

       /* wait for both threads to terminate */
    Pthread_join(tidA, NULL);
    Pthread_join(tidB, NULL);

    exit(0);
}
```

Mutex Example – [threads/example01.c]

```
void *
doit(void *vptr)
{
    int    i, val;

    /*
     * Each thread fetches, prints, and increments the
       counter NLOOP times.
     * The value of the counter should increase
       monotonically.
     */

    for (i = 0; i < NLOOP; i++) {
        val = counter;
        printf("%d: %d\n", pthread_self(), val + 1);
        counter = val + 1;
    }

    return(NULL);
}
```

Output from program “example01.c”

4: 1	
4: 2	
4: 3	
4: 4	
	continues as thread 4 executes
4: 517	
4: 518	
5: 518	thread 5 now executes
5: 519	
5: 520	
	continues as thread 5 executes
5: 926	
5: 927	
4: 519	thread 4 now executes; stored value is wrong
4: 520	

Mutex Lock and Unlock Functions

```
#include <pthread.h>

int pthread_mutex_lock(pthread_mutex_t *mptr);

int pthread_mutex_unlock(pthread_mutex_t *mptr);
```

Returns: 0 if OK, positive Exxx value on error

Corrected Version of Mutex Example – [threads/example02.c]

Using a mutex to protect the shared variable

```
#include  "unpthread.h"

#define   NLOOP 5000

int            counter;     /* incremented by threads */
pthread_mutex_t  counter_mutex =
PTHREAD_MUTEX_INITIALIZER;

void   *doit(void *);
```

Corrected Version of Mutex Example – [threads/example02.c]

Using a mutex to protect the shared variable

```
int
main(int argc, char **argv)
{
    pthread_t tidA, tidB;

    Pthread_create(&tidA, NULL, &doit, NULL);
    Pthread_create(&tidB, NULL, &doit, NULL);

        /* wait for both threads to terminate */
    Pthread_join(tidA, NULL);
    Pthread_join(tidB, NULL);

    exit(0);
}
```

Corrected Version of Mutex Example – [threads/example02.c]

```
void *
doit(void *vptr)
{
    int    i, val;

    /*
     * Each thread fetches, prints, and increments the
counter NLOOP times.
     * The value of the counter should increase
monotonically.
     */
```

Corrected Version of Mutex Example – [threads/example02.c]

```
    for (i = 0; i < NLOOP; i++) {
        Pthread_mutex_lock(&counter_mutex);

        val = counter;
        printf("%d: %d\n", pthread_self(), val + 1);
        counter = val + 1;

        Pthread_mutex_unlock(&counter_mutex);
    }

    return(NULL);
}
```

8.8 Condition Variable

특정 조건이 만족될 때가지 Sleep 상태로 유지되도록 허용

```
#include <pthread.h>

int pthread_cond_wait(pthread_cond_t *cptr,
      pthread_mutex_t *mptr);

int pthread_cond_signal(pthread_cond_t *cptr);

int pthread_cond_broadcast(pthread_cond_t *cptr);

int pthread_cond_timedwait(pthread_cond_t *cptr,
      pthread_mutex_t *mptr, const struct timespec
      *abstime);
```

Returns: 0 if OK, positive Exxx value on error

8.9 Web Client and Simultaneous Connections (Continued) – [threads/web03.c]

Main processing loop of main function

- 44-56: if possible, create another thread (not changed)
- 57-60: wait for thread to terminate

```
43 while (nlefttoread > 0) {
44         while (nconn < maxnconn && nlefttoconn > 0) {
45                 /* find a file to read */
46             for (i = 0 ; i < nfiles; i++)
47                 if (file[i].f_flags == 0)
48                     break;
49             if (i == nfiles)
50                 err_quit("nlefttoconn = %d nothing found",
   nlefttoconn);
```

8.9 Web Client and Simultaneous Connections (Continued) – [threads/web03.c]

Main processing loop of main function

- 44-56: if possible, create another thread (not changed)
- 57-60: wait for thread to terminate

```
51              file[i].f_flags = F_CONNECTING;
52              Pthread_create(&tid, NULL, &do_get_read,
&file[i]);
53              file[i].f_tid = tid;
54              nconn++;
55              nlefttoconn--;
56          }

57              /* Wait for thread to terminate */
58          Pthread_mutex_lock(&ndone_mutex);
59          while (ndone == 0)
60              Pthread_cond_wait(&ndone_cond, &ndone_mutex);
```

Web Client and Simultaneous Connections (Continued) – [threads/web03.c]

61-73: handle terminated thread

```
61          for (i = 0; i < nfiles; i++) {
62              if (file[i].f_flags & F_DONE) {
63                  Pthread_join(file[i].f_tid, (void **)
&fptr);

64                  if (&file[i] != fptr)
65                      err_quit("file[i] != fptr");
66                  fptr->f_flags = F_JOINED;
                        /* clears F_DONE */
```

Web Client and Simultaneous Connections (Continued) – [threads/web03.c]

61-73: handle terminated thread

```
67                 ndone--;
68                 nconn--;
69                 nlefttoread--;
70                 printf("thread %d for %s done\n", fptr->f_tid, fptr->f_name);
71             }
72         }
73         Pthread_mutex_unlock(&ndone_mutex);
74     }

75     exit(0);
76 }
```

실습 과제

pthread를 사용한 DNS 서버 작성

9. CLIENT/SERVER DESIGN ALTERNATIVES

목 차

9.1 Introduction

Preforking

- Creating a pool of child processes when the server starts

Prethreading

- Creating a pool of available threads when the server starts

Timing comparisons

Various servers discussed in this chapter

Row	Server description	Process control CPU time, seconds (difference from baseline)
0	Iterate server (baseline measurement; no process control)	0.0
1	Concurrent server, one fork per client request	20.90
2	Prefork with each child calling accept	1.80
3	Prefork with file locking to protect accept	2.0
4	Prefork with thread mutex locking to protect accept	1.75
5	Prefork with parent passing socket descriptor to child	2.58
6	Concurrent server, create one thread per client request	0.99
7	Prethreaded with mutex locking to protect accept	1.93
8	Prethreaded with main thread calling accept	2.05

9.2 TCP Client Alternatives

Basic TCP client

Client using select

Client using nonblocking I/O

Two processes

- Client beyond single-process, single-thread design

Two threads

9.3 TCP Test Client – [server/client.c]

TCP client program for testing the various servers

```
#include  "unp.h"

#define   MAXN   16384     /* max # bytes to request from server */

int
main(int argc, char **argv)
{
    int    i, j, fd, nchildren, nloops, nbytes;
    pid_t  pid;
    ssize_t   n;
    char   request[MAXLINE], reply[MAXN];
```

9.3 TCP Test Client – [server/client.c]

TCP client program for testing the various servers

```
    if (argc != 6)
        err_quit("usage: client <hostname or IPaddr>
<port> <#children>” "<#loops/child> <#bytes/request>");

    nchildren = atoi(argv[3]);
    nloops = atoi(argv[4]);
    nbytes = atoi(argv[5]);
    snprintf(request, sizeof(request), "%d\n", nbytes);
/* newline at end */
```

TCP Test Client – [server/client.c]

```
    for (i = 0; i < nchildren; i++) {
        if ( (pid = Fork()) == 0) {       /* child */
            for (j = 0; j < nloops; j++) {
                fd = Tcp_connect(argv[1], argv[2]);

                Write(fd, request, strlen(request));

                if ( (n = Readn(fd, reply, nbytes)) != 
nbytes)
                    err_quit("server returned %d bytes", n);

                Close(fd);
                    /* TIME_WAIT on client, not server */
            }
```

TCP Test Client – [server/client.c]

```
            printf("child %d done\n", i);
            exit(0);
        }
        /* parent loops around to fork() again */
    }

    while (wait(NULL) > 0)
            /* now parent waits for all children */
        ;
    if (errno != ECHILD)
        err_sys("wait error");

    exit(0);
}
```

9.4 TCP Iterative Server – [server/serv00.c]

16: register signal handler

```
01 #include  "unp.h"

02 int
03 main(int argc, char **argv)
04 {
05     int                 listenfd, connfd;
06     void                sig_int(int), web_child(int);
07     socklen_t           clilen, addrlen;
08     struct sockaddr        *cliaddr;
```

9.4 TCP Iterative Server – [server/serv00.c]

16: register signal handler

```
09     if (argc == 2)
10         listenfd = Tcp_listen(NULL, argv[1], &addrlen);
11     else if (argc == 3)
12         listenfd = Tcp_listen(argv[1], argv[2], &addrlen);
13     else
14         err_quit("usage: serv00 [ <host> ] <port#>");
15     cliaddr = Malloc(addrlen);

16     Signal(SIGINT, sig_int);
```

TCP Iterative Server – [server/serv00.c]

20: web_child function

25: signal handler for SIGINT

28: pr_cpu_time function - prints total CPU time

```
17 for ( ; ; ) {
18         clilen = addrlen;
19         connfd = Accept(listenfd, cliaddr, &clilen);

20         web_child(connfd);      /* process the request */

21         Close(connfd);/* parent closes connected socket */
22     }
23 }
```

TCP Iterative Server – [server/serv00.c]

20: web_child function

25: signal handler for SIGINT

28: pr_cpu_time function - prints total CPU time

```
24 void
25 sig_int(int signo)
26 {
27     void   pr_cpu_time(void);

28     pr_cpu_time();
29     exit(0);
30 }
```

web_child Function – [server / web_child.c]

Handle each client’s request

```
#include  "unp.h"

#define   MAXN   16384     /* max # bytes client can request */

void
web_child(int sockfd)
{
    int        ntowrite;
    ssize_t        nread;
    char       line[MAXLINE], result[MAXN];

```

web_child Function – [server/web_child.c]

Handle each client's request

```
    for ( ; ; ) {
        if ( (nread = Readline(sockfd, line, MAXLINE))== 0)
            return;   /* connection closed by other end */

    /* line from client specifies #bytes to write back */
        ntowrite = atol(line);
        if ((ntowrite <= 0) || (ntowrite > MAXN))
            err_quit("client request for %d bytes",
            ntowrite);

        Writen(sockfd, result, ntowrite);
    }
}
```

9.5 TCP Concurrent Server, One Child per Client – [server/serv01.c]

```
#include  "unp.h"

int
main(int argc, char **argv)
{
    int                 listenfd, connfd;
    pid_t               childpid;
    void                sig_chld(int), sig_int(int),
                        web_child(int);
    socklen_t           clilen, addrlen;
    struct sockaddr        *cliaddr;
```

9.5 TCP Concurrent Server, One Child per Client – [server/serv01.c]

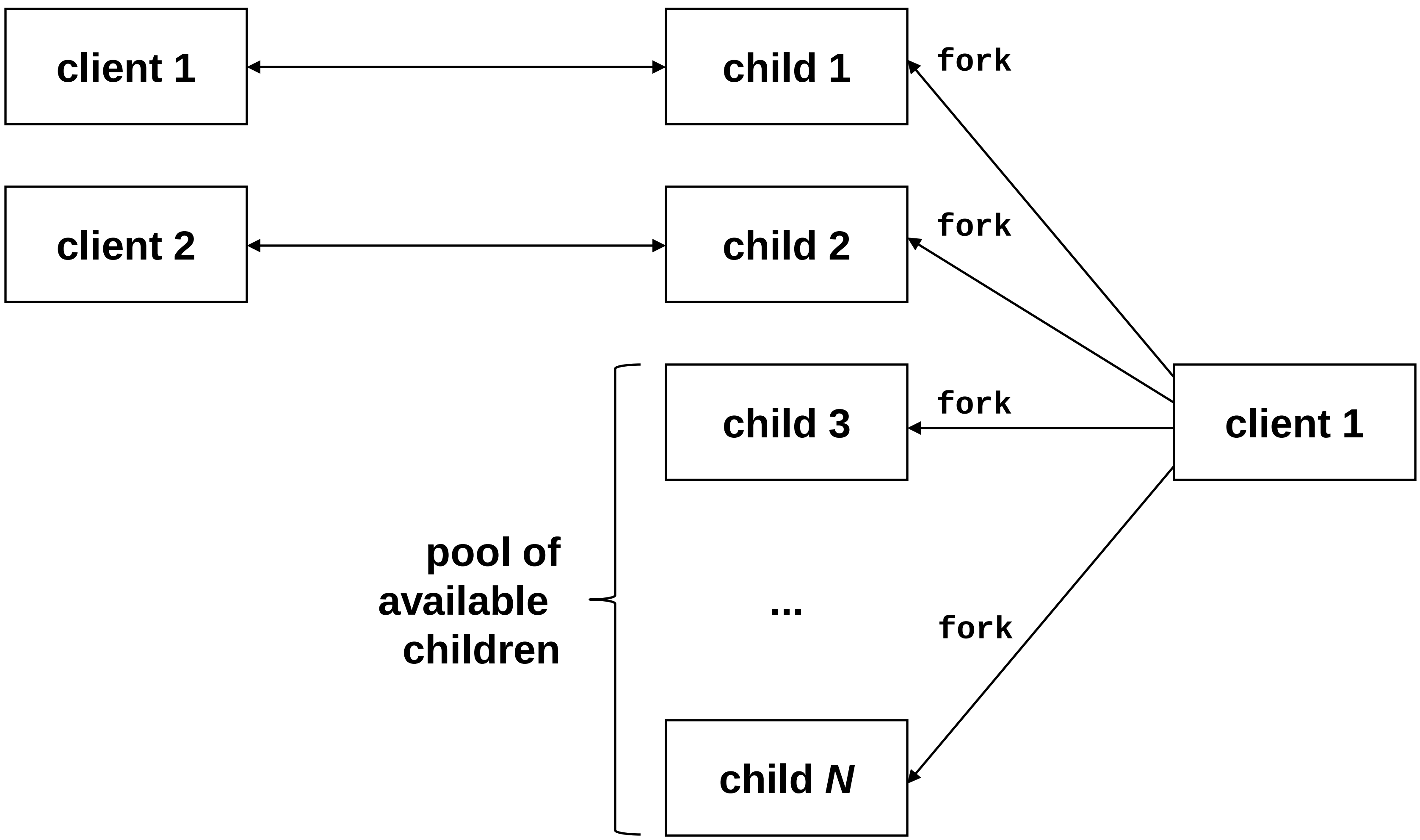
```
    if (argc == 2)
        listenfd = Tcp_listen(NULL, argv[1], &addrlen);
    else if (argc == 3)
        listenfd = Tcp_listen(argv[1], argv[2], &addrlen);
    else
        err_quit("usage: serv01 [ <host> ] <port#>");
    cliaddr = Malloc(addrlen);

    Signal(SIGCHLD, sig_chld);
    Signal(SIGINT, sig_int);
```

TCP Concurrent Server, One Child per Client – [server/serv01.c]

```
    for ( ; ; ) {
        clilen = addrlen;
        if ( (connfd = accept(listenfd, cliaddr, &clilen))
                < 0) {
            if (errno == EINTR)
                continue;    /* back to for() */
            else
                err_sys("accept error");
        }

        if ( (childpid = Fork()) == 0) {
            /* child process */
            Close(listenfd); /* close listening socket */
            web_child(connfd);  /* process request */
            exit(0);
        }
        Close(connfd);/* parent closes connected socket */
    }
}
```

9.6 TCP Preforked Server, No Locking Around accept

Preforking of children by server



TCP Preforked Server – [server/serv02.c]

```
#include  "unp.h"

static int    nchildren;
static pid_t  *pids;

int
main(int argc, char **argv)
{
    int        listenfd, i;
    socklen_t addrlen;
    void       sig_int(int);
    pid_t      child_make(int, int, int);
```

TCP Preforked Server – [server/serv02.c]

```
    if (argc == 3)
        listenfd = Tcp_listen(NULL, argv[1], &addrlen);
    else if (argc == 4)
        listenfd = Tcp_listen(argv[1], argv[2], &addrlen);
    else
        err_quit("usage: serv02 [ <host> ] <port#>
<#children>");
    nchildren = atoi(argv[argc-1]);
    pids = Calloc(nchildren, sizeof(pid_t));
```

TCP Preforked Server – [server/serv02.c]

```
    for (i = 0; i < nchildren; i++)
        pids[i] = child_make(i, listenfd, addrlen);
            /* parent returns */

    Signal(SIGINT, sig_int);

    for ( ; ; )
        pause();  /* everything done by children */
}
```

TCP Preforked Server – [server/serv02.c]

```
void
sig_int(int signo)
{
    int     i;
    void    pr_cpu_time(void);

        /* terminate all children */
    for (i = 0; i < nchildren; i++)
        kill(pids[i], SIGTERM);
    while (wait(NULL) > 0)  /* wait for all children */
        ;
    if (errno != ECHILD)
        err_sys("wait error");

    pr_cpu_time();
    exit(0);
}
```

child_make Function – [server/child02.c]

Creates each child

```
#include  "unp.h"

pid_t
child_make(int i, int listenfd, int addrlen)
{
    pid_t  pid;
    void        child_main(int, int, int);

    if ( (pid = Fork()) > 0)
        return(pid);     /* parent */

    child_main(i, listenfd, addrlen); /* never returns */
}
```

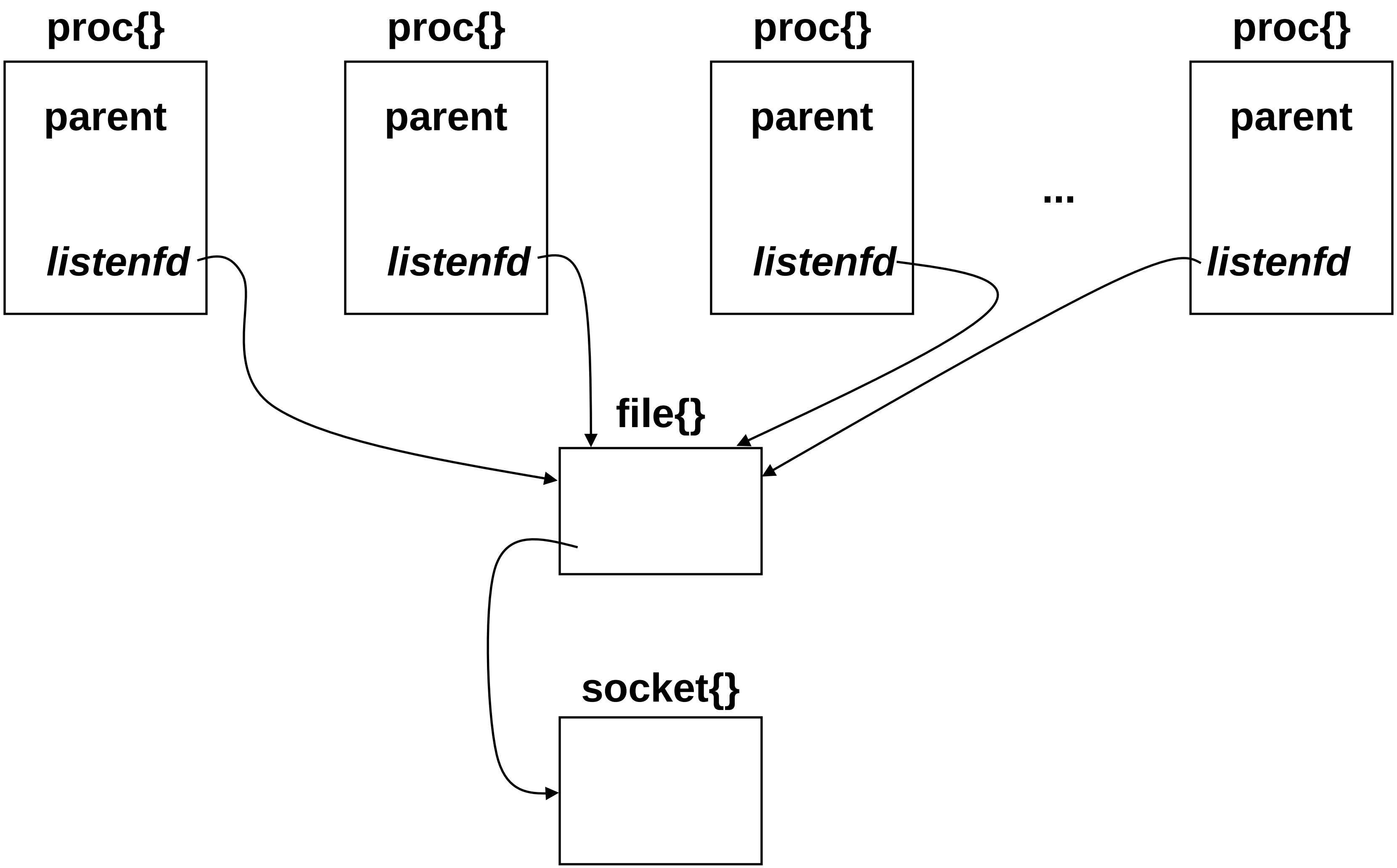
child_main Function – [server/child02.c]

Infinite loop executed by each child

```
void
child_main(int i, int listenfd, int addrlen)
{
    int          connfd;
    void            web_child(int);
    socklen_t    clilen;
    struct sockaddr  *cliaddr;

    cliaddr = Malloc(addrlen);

    printf("child %ld starting\n", (long) getpid());
    for ( ; ; ) {
        clilen = addrlen;
        connfd = Accept(listenfd, cliaddr, &clilen);

        web_child(connfd);     /* process the request */
        Close(connfd);
    }
}
```

4.4BSD Implementation

Arrangement of proc, file, and socket sturcture



meter Function – [server/meter.c]

Allocate an array in shared memory

- Distribution of connections to the children

```
#include  "unp.h"
#include  <sys/mman.h>

long *
meter(int nchildren)
{
    int    fd;
    long   *ptr;

#ifdef MAP_ANON
    ptr = Mmap(0, nchildren*sizeof(long), PROT_READ |
PROT_WRITE, MAP_ANON | MAP_SHARED, -1, 0);
#else
    fd = Open("/dev/zero", O_RDWR, 0);
```

meter Function – [server/meter.c]

Allocate an array in shared memory

- Distribution of connections to the children

```
    ptr = Mmap(0, nchildren*sizeof(long), PROT_READ |
PROT_WRITE, MAP_SHARED, fd, 0);
    Close(fd);
#endif

    return(ptr);
}
```

select Collisions

```
  void
  child_main(int i, int listenfd, int addrlen)
  {
      ...

      printf("child %ld starting\n", (long) getpid());
+     FD_ZERO(&rset);
      for ( ; ; ) {
+         FD_SET(listenfd, &rset);
+         Select(listenfd+1, &rset, NULL, NULL, NULL);
+         if (FD_ISSET(listenfd, &rset) == 0)
+             err_quit("listenfd readable");
+
          clilen = addrlen;
          connfd = Accept(listenfd, cliaddr, &clilen);

          web_child(connfd);       /* process the request */
          Close(connfd);
      }
  }
```

9.7 TCP Preforked Server, File Locking Around accept

Server main function – [server/serv03.c]

```
  int
  main(int argc, char **argv)
  {
      ...

+     my_lock_init("/tmp/lock.XXXXXX");
              /* lock file for all children */
      for (i = 0; i < nchildren; i++)
          pids[i] = child_make(i, listenfd, addrlen); /*
  parent returns */

      ...
  }
```

TCP Preforked Server, File Locking Around accept

child_main function – [server/child03.c]

```
    void
    child_main(int i, int listenfd, int addrlen)
    {

        ...

        for ( ; ; ) {
            clilen = addrlen;
+           my_lock_wait();
            connfd = Accept(listenfd, cliaddr, &clilen);
+           my_lock_release();

            web_child(connfd);      /* process the request */
            Close(connfd);
        }
    }
```

my_lock_init Function – [server/lock_fcntl.c]

```
#include  "unp.h"

static struct flock lock_it, unlock_it;
static int       lock_fd = -1;
   /* fcntl() will fail if my_lock_init() not called */

void
my_lock_init(char *pathname)
{
    char  lock_file[1024];

       /* must copy caller's string, in case it's a
constant */
    strncpy(lock_file, pathname, sizeof(lock_file));
    lock_fd = Mkstemp(lock_file);
```

my_lock_init Function – [server/lock_fcntl.c]

```
    Unlink(lock_file);  /* but lock_fd remains open */

    lock_it.l_type = F_WRLCK;
    lock_it.l_whence = SEEK_SET;
    lock_it.l_start = 0;
    lock_it.l_len = 0;

    unlock_it.l_type = F_UNLCK;
    unlock_it.l_whence = SEEK_SET;
    unlock_it.l_start = 0;
    unlock_it.l_len = 0;
}
```

my_lock_wait and my_lock_release Functions using fcntl – [server/lock_fcntl.c]

```
void
my_lock_wait()
{
    int       rc;

    while ( (rc = fcntl(lock_fd, F_SETLKW, &lock_it)) < 0) {
        if (errno == EINTR)
            continue;
        else
            err_sys("fcntl error for my_lock_wait");
    }
}

void
my_lock_release()
{
    if (fcntl(lock_fd, F_SETLKW, &unlock_it) < 0)
        err_sys("fcntl error for my_lock_release");
}
```

9.8 TCP Preforked Server, Thread Locking Around accept – [server/lock_pthread.c]

my_lock_init function using pthread locking between processes

```
#include  "unpthread.h"
#include  <sys/mman.h>

static pthread_mutex_t *mptr;
    /* actual mutex will be in shared memory */

void
my_lock_init(char *pathname)
{
    int    fd;
    pthread_mutexattr_t mattr;
```

9.8 TCP Preforked Server, Thread Locking Around accept – [server/lock_pthread.c]

my_lock_init function using pthread locking between processes

```
    fd = Open("/dev/zero", O_RDWR, 0);

    mptr = Mmap(0, sizeof(pthread_mutex_t), PROT_READ |
        PROT_WRITE, MAP_SHARED, fd, 0);
    Close(fd);

    Pthread_mutexattr_init(&mattr);
    Pthread_mutexattr_setpshared(&mattr,
        PTHREAD_PROCESS_SHARED);
    Pthread_mutex_init(mptr, &mattr);
}
```

TCP Preforked Server, Thread Locking Around accept – [server/lock_pthread.c]

my_lock_wait and my_lock_release functions using pthread locking

```
void
my_lock_wait()
{
    Pthread_mutex_lock(mptr);
}

void
my_lock_release()
{
    Pthread_mutex_unlock(mptr);
}
```

9.9 TCP Preforked Server, Descriptor Passing

Child structure

```
typedef struct {
  pid_t       child_pid;    /* process ID */
  int     child_pipefd;
              /* parent's stream pipe to/from child */
  int     child_status;/* 0 = ready */
  long    child_count;     /* # connections handled */
} Child;

Child  *cptr; /* array of Child structures; calloc'ed */
```

child_make Function - [server/child05.c]

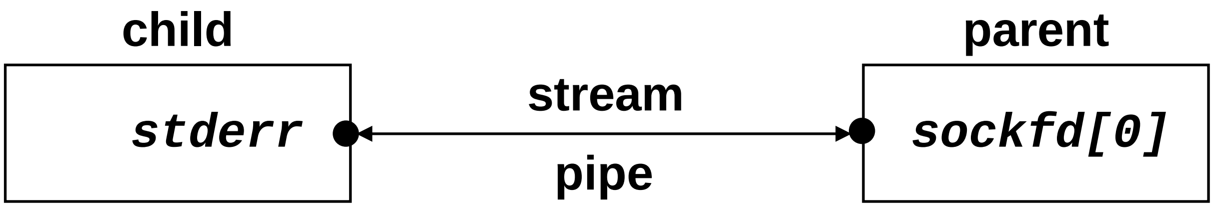
```
#include  "unp.h"
#include  "child.h"

pid_t
child_make(int i, int listenfd, int addrlen)
{
    int    sockfd[2];
    pid_t  pid;
    void   child_main(int, int, int);

    Socketpair(AF_LOCAL, SOCK_STREAM, 0, sockfd);

    if ( (pid = Fork()) > 0) {
        Close(sockfd[1]);
        cptr[i].child_pid = pid;
        cptr[i].child_pipefd = sockfd[0];
        cptr[i].child_status = 0;
        return(pid);     /* parent */
    }
```

child_make Function - [server/child05.c]

```
    Dup2(sockfd[1], STDERR_FILENO);
        /* child's stream pipe to parent */
    Close(sockfd[0]);
    Close(sockfd[1]);
    Close(listenfd);
        /* child does not need this open */
    child_main(i, listenfd, addrlen); /* never returns */
}
```

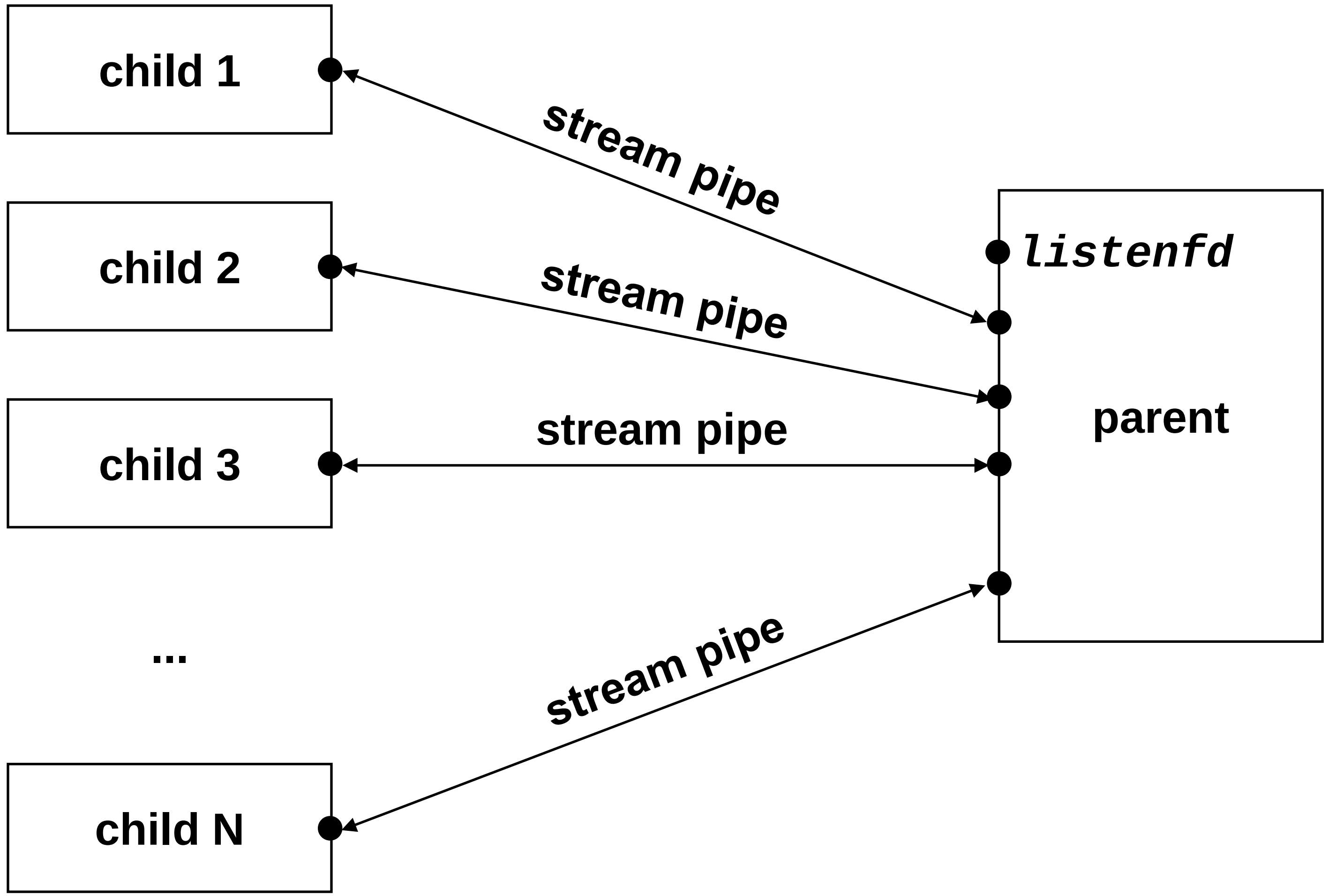
child_make Function

Stream pipe after parent and child both close one end



child_make Function

Stream pipes after all children have been created



child_main Function - [server/child05.c]

32-33: wait for descriptor from parent

38: tell parent we are ready

```
23 void
24 child_main(int i, int listenfd, int addrlen)
25 {
26     char           c;
27     int            connfd;
28     ssize_t           n;
29     void           web_child(int);

30     printf("child %ld starting\n", (long) getpid());
31     for ( ; ; ) {
32         if ( (n = Read_fd(STDERR_FILENO, &c, 1, &connfd))
== 0)
33             err_quit("read_fd returned 0");
```

child_main Function - [server/child05.c]

32-33: wait for descriptor from parent

38: tell parent we are ready

```
34         if (connfd < 0)
35             err_quit("no descriptor from read_fd");

36         web_child(connfd);            /* process request */
37         Close(connfd);

38         Write(STDERR_FILENO, "", 1);
               /* tell parent we're ready again */
39     }
40 }
```

main Function that uses descriptor passing – [server/serv05.c]

```
#include  "unp.h"
#include  "child.h"

static int    nchildren;

int
main(int argc, char **argv)
{
    int    listenfd, i, navail, maxfd, nsel, connfd, rc;
    void       sig_int(int);
    pid_t      child_make(int, int, int);
    ssize_t       n;
    fd_set    rset, masterset;
    socklen_t addrlen, clilen;
    struct sockaddr  *cliaddr;
```

main Function that uses descriptor passing – [server/serv05.c]

```
    if (argc == 3)
        listenfd = Tcp_listen(NULL, argv[1], &addrlen);
    else if (argc == 4)
        listenfd = Tcp_listen(argv[1], argv[2], &addrlen);
    else
        err_quit("usage: serv05 [ <host> ] <port#>
<#children>");
```

main Function that uses descriptor passing – [server/serv05.c]

36-37: turn off listening socket if no available children

```
20 FD_ZERO(&masterset);
21     FD_SET(listenfd, &masterset);
22     maxfd = listenfd;
23     cliaddr = Malloc(addrlen);

24     nchildren = atoi(argv[argc-1]);
25     navail = nchildren;
26     cptr = Calloc(nchildren, sizeof(Child));

27         /* prefork all the children */
28     for (i = 0; i < nchildren; i++) {
29         child_make(i, listenfd, addrlen);
               /* parent returns */
30         FD_SET(cptr[i].child_pipefd, &masterset);
31         maxfd = max(maxfd, cptr[i].child_pipefd);
32     }
```

main Function that uses descriptor passing – [server/serv05.c]

36-37: turn off listening socket if no available children

```
33    Signal(SIGINT, sig_int);

34    for ( ; ; ) {
35        rset = masterset;
36        if (navail <= 0)
37            FD_CLR(listenfd, &rset);

38        nsel = Select(maxfd + 1, &rset, NULL, NULL, NULL);
```

main Function that uses descriptor passing – [server/serv05.c]

39-55: accept new connection

```
39              /* check for new connections */
40          if (FD_ISSET(listenfd, &rset)) {
41              clilen = addrlen;
42              connfd = Accept(listenfd, cliaddr, &clilen);

43              for (i = 0; i < nchildren; i++)
44                  if (cptr[i].child_status == 0)
45                      break;              /* available */

46              if (i == nchildren)
47                  err_quit("no available children");
48              cptr[i].child_status = 1;
                    /* mark child as busy */
```

main Function that uses descriptor passing – [server/serv05.c]

39-55: accept new connection

```
49            cptr[i].child_count++;
50            navail--;

51            n = Write_fd(cptr[i].child_pipefd, "", 1, 
connfd);
52            Close(connfd);
53            if (--nsel == 0)
54                continue;
                      /* all done with select() results */
55        }
```

main Function that uses descriptor passing – [server/serv05.c]

56-66: handle any newly available children

```
56     /* find any newly-available children */
57         for (i = 0; i < nchildren; i++) {
58             if (FD_ISSET(cptr[i].child_pipefd, &rset)) {
59                 if ( (n = Read(cptr[i].child_pipefd, &rc, 1)) == 0)
60                     err_quit("child %d terminated unexpectedly", i);
61                 cptr[i].child_status = 0;
62                 navail++;
63                 if (--nsel == 0)
64                     break;    /* all done with select() results */
65             }
66         }
67     }
68 }
```

TCP Concurrent Server, One Thread per Client

main function for TCP threaded server – [server/serv06.c]

```
#include  "unpthread.h"

int
main(int argc, char **argv)
{
    int             listenfd, connfd;
    void            sig_int(int);
    void            *doit(void *);
    pthread_t       tid;
    socklen_t       clilen, addrlen;
    struct sockaddr  *cliaddr;
```

9.10 TCP Concurrent Server, One Thread per Client

main function for TCP threaded server – [server/serv06.c]

```
    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: serv06 [ <host> ] <port#>");
    cliaddr = Malloc(addrlen);

    Signal(SIGINT, sig_int);
```

main function for TCP threaded server – [server/serv06.c]

19-23: main thread loop

25-33: per-thread function

```
19 for ( ; ; ) {
20         clilen = addrlen;
21         connfd = Accept(listenfd, cliaddr, &clilen);

22        Pthread_create(&tid, NULL, &doit, (void *) connfd);
23     }
24 }
```

main function for TCP threaded server – [server/serv06.c]

19-23: main thread loop

25-33: per-thread function

```
25 void *
26 doit(void *arg)
27 {
28     void   web_child(int);

29     Pthread_detach(pthread_self());
30     web_child((int) arg);
31     Close((int) arg);
32     return(NULL);
33 }
```

9.11 TCP Prethreaded Server, per-Thread accept

pthread07.h header – [server/pthread07.h]

```
typedef struct {
  pthread_t      thread_tid;      /* thread ID */
  long       thread_count; /* # connections handled */
} Thread;
Thread *tptr;
   /* array of Thread structures; calloc'ed */

int          listenfd, nthreads;
socklen_t    addrlen;
pthread_mutex_t  mlock;
```

main function for prethreaded TCP server-[server/serv07.c]

```
#include  "unpthread.h"
#include  "pthread07.h"

pthread_mutex_t  mlock = PTHREAD_MUTEX_INITIALIZER;

int
main(int argc, char **argv)
{
   int    i;
   void   sig_int(int), thread_make(int);

   if (argc == 3)
      listenfd = Tcp_listen(NULL, argv[1], &addrlen);
   else if (argc == 4)
      listenfd = Tcp_listen(argv[1], argv[2], &addrlen);
   else
      err_quit("usage: serv07 [ <host> ] <port#>
<#threads>");
```

main function for prethreaded TCP server-[server/serv07.c]

```
    nthreads = atoi(argv[argc-1]);
    tptr = Calloc(nthreads, sizeof(Thread));

    for (i = 0; i < nthreads; i++)
        thread_make(i);  /* only main thread returns */

    Signal(SIGINT, sig_int);

    for ( ; ; )
        pause();  /* everything done by threads */
}
```

thread_make and thread_main Function – [server/pthread07.c]

thread_make function

```
#include  "unpthread.h"
#include  "pthread07.h"

void
thread_make(int i)
{
    void   *thread_main(void *);

    Pthread_create(&tptr[i].thread_tid, NULL,
&thread_main, (void *) i);
    return;       /* main thread returns */
}
```

thread_make and thread_main Function – [server/pthread07.c]

thread_main function

```
void *
thread_main(void *arg)
{
    int             connfd;
    void            web_child(int);
    socklen_t       clilen;
    struct sockaddr  *cliaddr;

    cliaddr = Malloc(addrlen);
```

thread_make and thread_main Function – [server/pthread07.c]

thread_main function

```
    printf("thread %d starting\n", (int) arg);
    for ( ; ; ) {
        clilen = addrlen;
        Pthread_mutex_lock(&mlock);
        connfd = Accept(listenfd, cliaddr, &clilen);
        Pthread_mutex_unlock(&mlock);
        tptr[(int) arg].thread_count++;

        web_child(connfd);      /* process request */
        Close(connfd);
    }
}
```

9.12 TCP Prethreaded Server, Main Thread accept

pthread08.h header – [server/pthread08.h]

- 6-9: define shared array to hold connected sockets

```
01 typedef struct {
02   pthread_t       thread_tid;      /* thread ID */
03   long         thread_count; /* # connections handled */
04 } Thread;
05 Thread *tptr;
         /* array of Thread structures; calloc'ed */

06 #define   MAXNCLI   32
07 int              clifd[MAXNCLI], iget, iput;
08 pthread_mutex_t     clifd_mutex;
09 pthread_cond_t      clifd_cond;
```

main Function for Prethreaded Server – [server/serv08.c]

```
#include  "unpthread.h"
#include  "pthread08.h"

static int        nthreads;
pthread_mutex_t      clifd_mutex =
PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t       clifd_cond =
PTHREAD_COND_INITIALIZER;

int
main(int argc, char **argv)
{
    int        i, listenfd, connfd;
    void       sig_int(int), thread_make(int);
    socklen_t addrlen, clilen;
    struct sockaddr  *cliaddr;
```

main Function for Prethreaded Server – [server/serv08.c]

```
    if (argc == 3)
        listenfd = Tcp_listen(NULL, argv[1], &addrlen);
    else if (argc == 4)
        listenfd = Tcp_listen(argv[1], argv[2], &addrlen);
    else
        err_quit("usage: serv08 [ <host> ] <port#> <#threads>");
    cliaddr = Malloc(addrlen);

    nthreads = atoi(argv[argc-1]);
    tptr = Calloc(nthreads, sizeof(Thread));
    iget = iput = 0;
```

main Function for Prethreaded Server – [server/serv08.c]

23-25: create pool of threads

27-38: wait for each client connection

```
23        /* create all the threads */
24     for (i = 0; i < nthreads; i++)
25         thread_make(i);  /* only main thread returns */

26     Signal(SIGINT, sig_int);

27     for ( ; ; ) {
28         clilen = addrlen;
29         connfd = Accept(listenfd, cliaddr, &clilen);
```

main Function for Prethreaded Server – [server/serv08.c]

23-25: create pool of threads

27-38: wait for each client connection

```
30          Pthread_mutex_lock(&clifd_mutex);
31          clifd[iput] = connfd;
32          if (++iput == MAXNCLI)
33              iput = 0;
34          if (iput == iget)
35              err_quit("iput = iget = %d", iput);
36          Pthread_cond_signal(&clifd_cond);
37          Pthread_mutex_unlock(&clifd_mutex);
38      }
39  }
```

thread_make and thread_main Function f– [server/pthread08.c]

thread_make function

```
#include  "unpthread.h"
#include  "pthread08.h"

void
thread_make(int i)
{
    void   *thread_main(void *);

    Pthread_create(&tptr[i].thread_tid, NULL, &thread_main, (void *) i);
    return;      /* main thread returns */
}
```

thread_make and thread_main Function f– [server/pthread08.c]

thread_main function

- 17-26: wait for client desciptor to service

```
10 void *
11 thread_main(void *arg)
12 {
13     int    connfd;
14     void   web_child(int);

15     printf("thread %d starting\n", (int) arg);
16     for ( ; ; ) {
17         Pthread_mutex_lock(&clifd_mutex);
18         while (iget == iput)
19             Pthread_cond_wait(&clifd_cond, &clifd_mutex);
20         connfd = clifd[iget];
               /* connected socket to service */
```

thread_make and thread_main Function f– [server/pthread08.c]

thread_main function

- 17-26: wait for client desciptor to service

```
21         if (++iget == MAXNCLI)
22             iget = 0;
23         Pthread_mutex_unlock(&clifd_mutex);
24         tptr[(int) arg].thread_count++;

25         web_child(connfd);      /* process request */
26         Close(connfd);
27     }
28 }
```

실습 과제

Prefork를 사용한 DNS 서버 작성

Prethread를 사용한 DNS 서버 작성

10일차

10. RAW SOCKETS AND DATALINK ACCESS

목 차

10.1 Introduction to Raw Socket

Three features

- Read and write ICMPv4, IGMPv4, and ICMPv6 packets
- Read and write IPv4 datagrams with an IPv4 protocol field
- Build IPv4 header using the IP_HDRINCL socket option

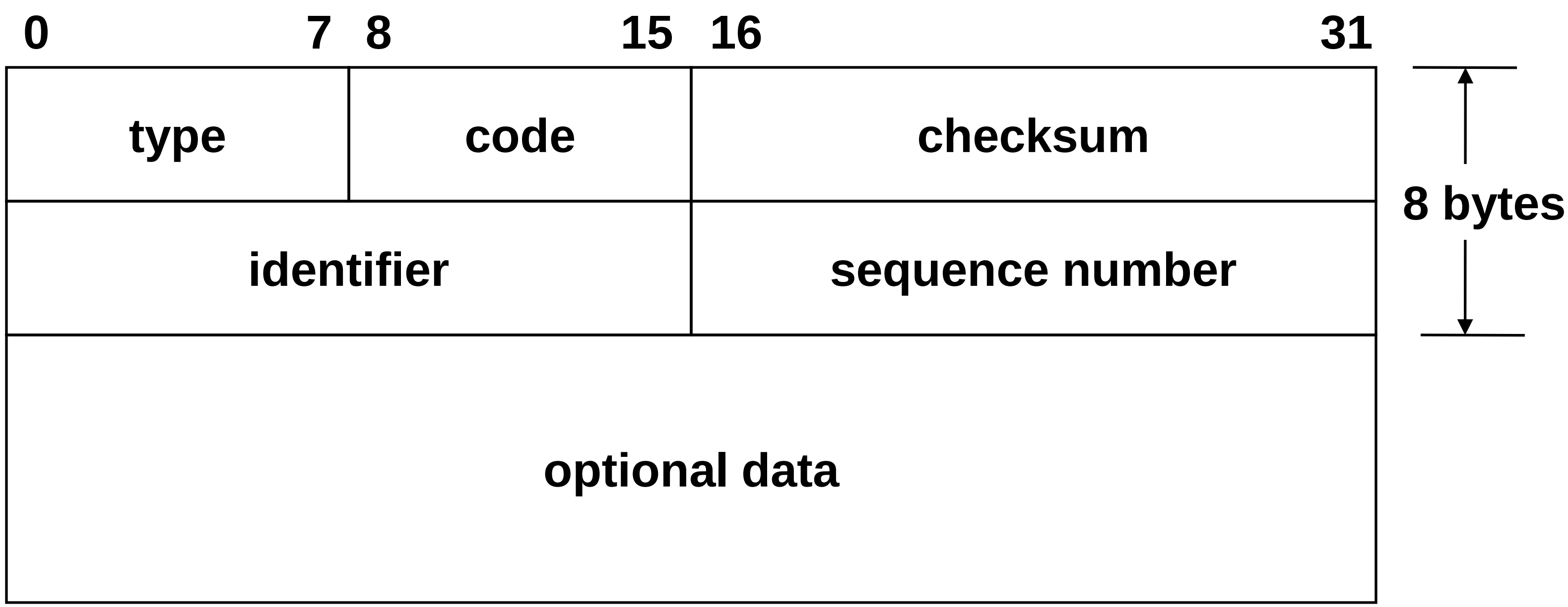
10.2 Raw Socket Creation

Steps involved in creating a raw socket

- socket
  - socket(AF_INET, SOCK_RAW, protocol)
  - Protocol field는 Nonzero 값으로 셋팅
- setsockopt
  - setsockopt(sockfd, IPPROTO_IP, IP_HDRINCL, &on, sizeof(on));
- bind
- connect

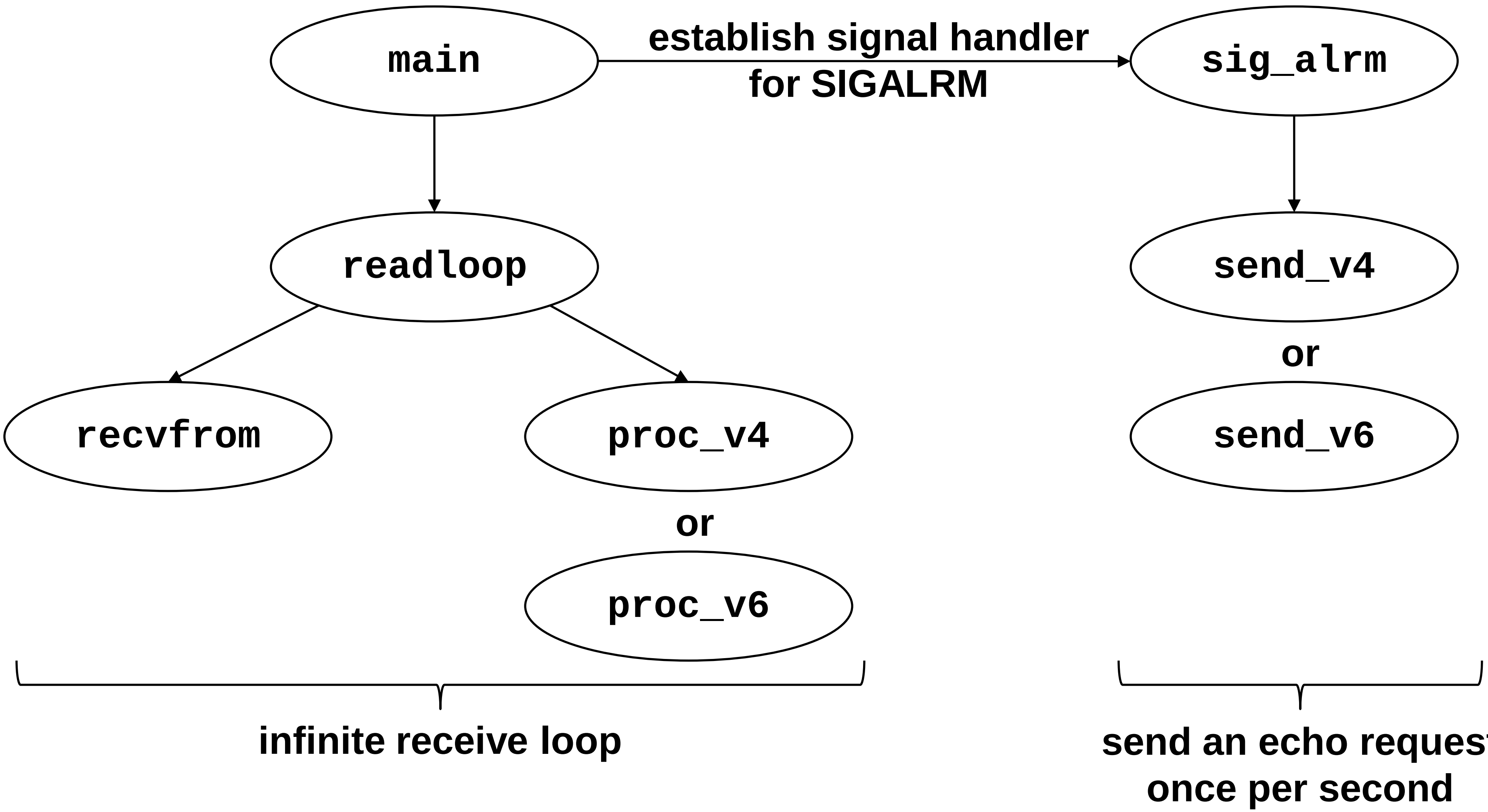
10.3 ping Program

Fromat of ICMP echo request and echo reply messages



ping Program (계속)

Overview of the functions in the ping program



ping.h Header – [ping / ping.h]

1-22: include IPv4 and ICMPv4 headers

```
01 #include  "unp.h"
02 #include  <netinet/in_systm.h>
03 #include  <netinet/ip.h>
04 #include  <netinet/ip_icmp.h>

05 #define   BUFSIZE      1500

06           /* globals */
07 char    sendbuf[BUFSIZE];

08 int     datalen;    /* # bytes of data following ICMP header */
09 char   *host;
10 int     nsent;          /* add 1 for each sendto() */
11 pid_t   pid;        /* our PID */
12 int     sockfd;
13 int     verbose;
```

ping.h Header – [ping / ping.h]

1-22: include IPv4 and ICMPv4 headers

```
14            /* function prototypes */
15 void    init_v6(void);
16 void    proc_v4(char *, ssize_t, struct msghdr *, struct timeval *);
17 void    proc_v6(char *, ssize_t, struct msghdr *, struct timeval *);
18 void    send_v4(void);
19 void    send_v6(void);
20 void    readloop(void);
21 void    sig_alrm(int);
22 void    tv_sub(struct timeval *, struct timeval *);
```

ping.h Header – [ping/ping.h]

23-31: define proto structure

32-35: include IPv6 and ICMPv6 headers

```
23 struct proto {
24   void  (*fproc)(char *, ssize_t, struct msghdr *, struct
timeval *);
25   void  (*fsend)(void);
26   void  (*finit)(void);
27   struct sockaddr  *sasend;
        /* sockaddr{} for send, from getaddrinfo */
28   struct sockaddr  *sarecv;
        /* sockaddr{} for receiving */
29   socklen_t      salen;    /* length of sockaddr{}s */
30   int        icmpproto; /* IPPROTO_xxx value for ICMP */
31 } *pr;
```

ping.h Header – [ping/ping.h]

23-31: define proto structure

32-35: include IPv6 and ICMPv6 headers

```
32 #ifdef IPV6

33 #include  <netinet/ip6.h>
34 #include  <netinet/icmp6.h>

35 #endif
```

main Function – [ping / main.c]

2-7: define proto structures for IPv4 and IPv6

8: length of optional data

```
1 #include  "ping.h"

2 struct proto  proto_v4 = { proc_v4, send_v4, NULL, NULL,
3 NULL, 0, IPPROTO_ICMP };

4 #ifdef IPV6
5 struct proto  proto_v6 = { proc_v6, send_v6, init_v6,
6 NULL, NULL, 0, IPPROTO_ICMPV6 };
7 #endif
```

main Function – [ping / main.c]

2-7: define proto structures for IPv4 and IPv6

8: length of optional data

```
 8  int datalen = 56;
        /* data that goes with ICMP echo request */

 9  int
10  main(int argc, char **argv)
11  {
12      int             c;
13      struct addrinfo  *ai;
14      char *h;
```

main Function – [ping / main.c]

15-24: handle command-line options

```
15      opterr = 0;
            /* don't want getopt() writing to stderr */
16      while ( (c = getopt(argc, argv, "v")) != -1) {
17          switch (c) {
18          case 'v':
19              verbose++;
20              break;

21          case '?':
22              err_quit("unrecognized option: %c", c);
23          }
24      }
```

main Function – [ping / main.c]

15-24: handle command-line options

```
15     opterr = 0
16     while ( (c = getopt(argc, argv, "v")) != -1) {
17         switch (c) {
18         case 'v':
19             verbose++;
20             break;

21         case '?':
22             err_quit("unrecognized option: %c", c);
23         }
24     }
```

main Function – [ping / main.c]

15-24: handle command-line options

```
25    if (optind != argc-1)
26        err_quit("usage: ping [ -v ] <hostname>");
27    host = argv[optind];

28    pid = getpid() & 0xffff;/* ICMP ID field 16 bits */
29    Signal(SIGALRM, sig_alrm);

30    ai = Host_serv(host, NULL, 0, 0);
```

main Function – [ping / main.c]

31-48: process hostname argument

```
31      h = Sock_ntop_host(ai->ai_addr, ai->ai_addrlen);
32      printf("PING %s (%s): %d data bytes\n",
33          ai->ai_canonname ? ai->ai_canonname : h, h,
        datalen);

34          /* initialize according to protocol */
35      if (ai->ai_family == AF_INET) {
36          pr = &proto_v4;
37  #ifdef IPV6
38      } else if (ai->ai_family == AF_INET6) {
39          pr = &proto_v6;
40         if (IN6_IS_ADDR_V4MAPPED(&(((struct sockaddr_in6 *)
41                                   ai->ai_addr)->sin6_addr)))
42              err_quit("cannot ping IPv4-mapped IPv6
                       address");
43  #endif
```

main Function – [ping / main.c]

31-48: process hostname argument

```
44     } else
45         err_quit("unknown address family %d", ai->
               ai_family);

46     pr->sasend = ai->ai_addr;
47     pr->sarecv = Calloc(1, ai->ai_addrlen);
48     pr->salen = ai->ai_addrlen;

49     readloop();

50     exit(0);
51 }
```

readloop Function – [ping/readloop.c]

12-13: create socket

14-15: perform protocol-specific initialization

```
1  #include  "ping.h"

2  void
3  readloop(void)
4  {
5      int             size;
6      char            recvbuf[BUFSIZE];
7      char            controlbuf[BUFSIZE];
8      struct msghdr   msg;
9      struct iovec    iov;
10     ssize_t            n;
11     struct timeval     tval;
```

readloop Function – [ping/readloop.c]

12-13: create socket

14-15: perform protocol-specific initialization

```
12    sockfd = Socket(pr->sasend->sa_family, SOCK_RAW, pr->
             icmpproto);
13    setuid(getuid());
         /* don't need special permissions any more */
14    if (pr->finit)
15       (*pr->finit)();
```

readloop Function – [ping/readloop.c]

16-17: set socket receive buffer size

18: send first packet

19-24: set up msghdr for recvmsg

```
16      size = 60 * 1024;       /* OK if setsockopt fails */
17      setsockopt(sockfd, SOL_SOCKET, SO_RCVBUF, &size,
                sizeof(size));

18      sig_alrm(SIGALRM);  /* send first packet */

19      iov.iov_base = recvbuf;
20      iov.iov_len = sizeof(recvbuf);
21      msg.msg_name = pr->sarecv;
22      msg.msg_iov = &iov;
23      msg.msg_iovlen = 1;
24      msg.msg_control = controlbuf;
```

readloop Function – [ping/readloop.c]

25-37: infinite loop reading all ICMP messages

```
25      for ( ; ; ) {
26          msg.msg_namelen = pr->salen;
27          msg.msg_controllen = sizeof(controlbuf);
28          n = recvmsg(sockfd, &msg, 0);
29          if (n < 0) {
30              if (errno == EINTR)
31                  continue;
32              else
33                  err_sys("recvmsg error");
34          }

35          Gettimeofday(&tval, NULL);
36          (*pr->fproc)(recvbuf, n, &msg, &tval);
37      }
38  }
```

recvmsg and sendmsg Functions

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

ssize_t recvmsg(int sockfd, struct msghdr *msg, int flags);

ssize_t sendmsg(int sockfd, struct msghdr *msg, int flags);
```

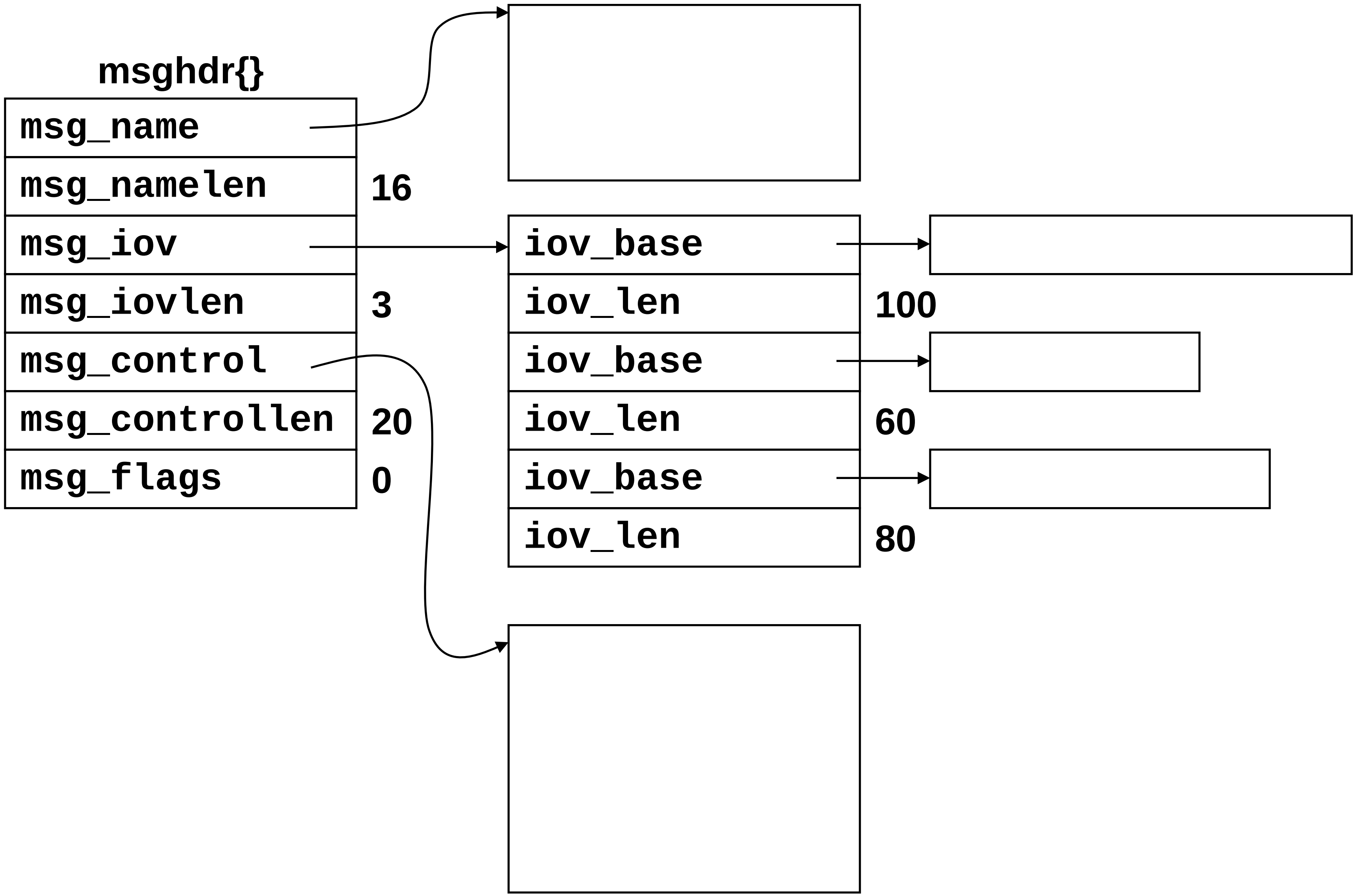
Both return: number of bytes read or written if OK, -1 on error

recvmsg and sendmsg Functions

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

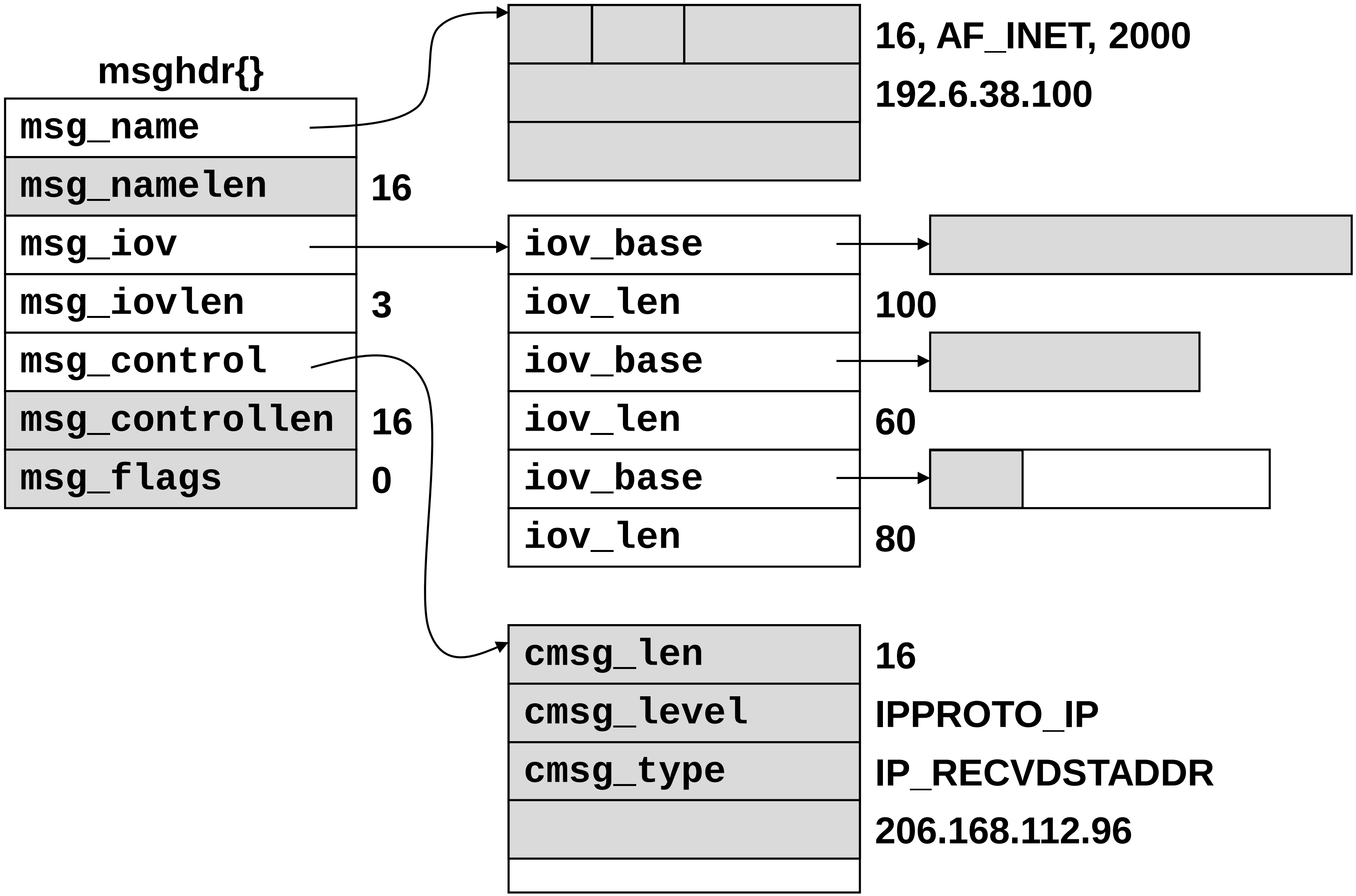
Data Structures for recvmsg

Data structures when recvmsg is called for a UDP socket



Data Structures for recvmsg (계속)

Update of data structures when recvmsg returns



proc_v4 Function: processes ICMPv4 message – [ping/proc_v4.c]

10-16: get pointer to ICMP header

```
1 #include  "ping.h"

2 void
3 proc_v4(char *ptr, ssize_t len, struct msghdr *msg,
  struct timeval *tvrecv)
4 {
5     int             hlen1, icmplen;
6     double          rtt;
7     struct ip       *ip;
8     struct icmp        *icmp;
9     struct timeval    *tvsend;
```

proc_v4 Function: processes ICMPv4 message – [ping/proc_v4.c]

10-16: get pointer to ICMP header

```
10      ip = (struct ip *) ptr;       /* start of IP header */
11      hlen1 = ip->ip_hl << 2;       /* length of IP header */
12      if (ip->ip_p != IPPROTO_ICMP)
13          return;                   /* not ICMP */

14      icmp = (struct icmp *) (ptr + hlen1);
            /* start of ICMP header */
15      if ( (icmplen = len - hlen1) < 8)
16          return;                   /* malformed packet */
```

proc_v4 Function: processes ICMPv4 message – [ping/proc_v4.c]

17-21: check for ICMP echo reply

28-32: print all received ICMP messages if vervose option specified

```
17      if (icmp->icmp_type == ICMP_ECHOREPLY) {
18          if (icmp->icmp_id != pid)
19              return;
                    /* not a response to our ECHO_REQUEST */
20          if (icmplen < 16)
21              return;             /* not enough data to use */

22          tvsend = (struct timeval *) icmp->icmp_data;
23          tv_sub(tvrecv, tvsend);
24          rtt = tvrecv->tv_sec * 1000.0 + tvrecv->tv_usec /
                    1000.0;
```

proc_v4 Function: processes ICMPv4 message – [ping / proc_v4.c]

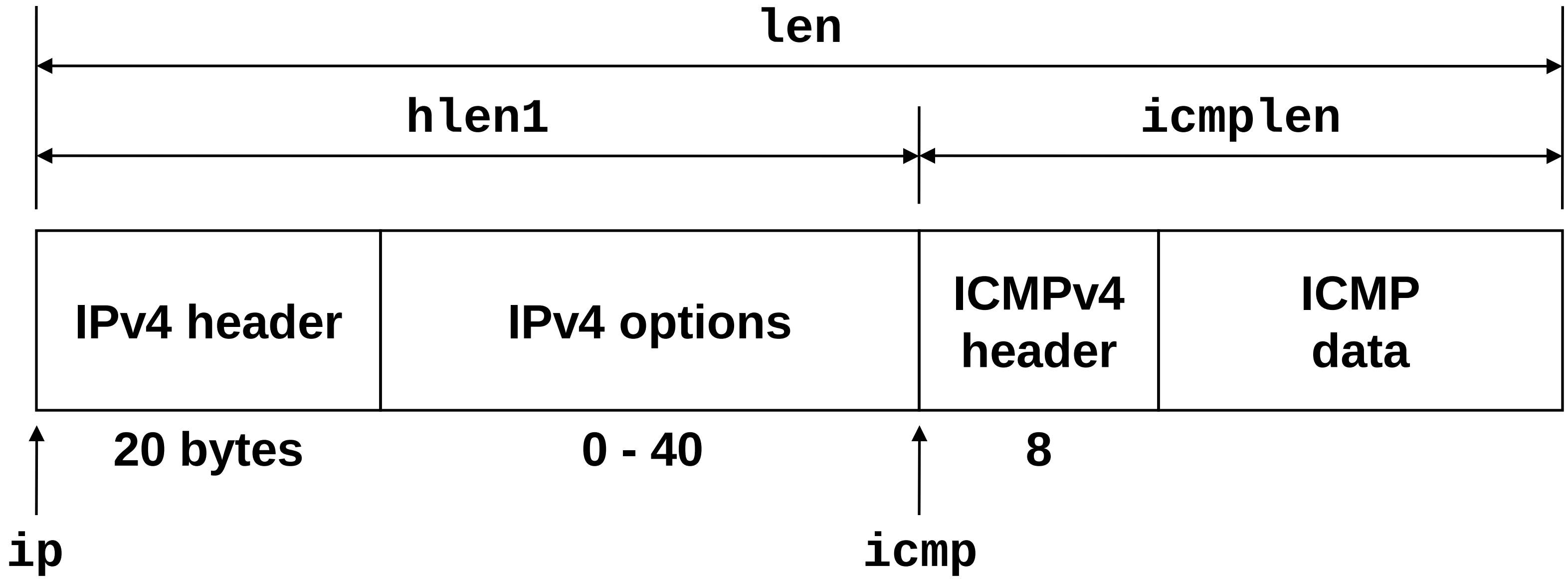
17-21: check for ICMP echo reply

28-32: print all received ICMP messages if vervose option specified

```
25         printf("%d bytes from %s: seq=%u, ttl=%d, rtt=%.3f
26 ms\n“, icmplen, Sock_ntop_host(pr->sarecv, pr->salen),
27 icmp->icmp_seq, ip->ip_ttl, rtt);

28     } else if (verbose) {
29         printf("  %d bytes from %s: type = %d, code
30 = %d\n“, icmplen, Sock_ntop_host(pr->sarecv, pr->salen),
31 icmp->icmp_type, icmp->icmp_code);
32     }
33 }
```

ICMP reply message

Headers, pointers, and lengths in processing ICMPv4 reply



tv_sub Function – [lib/tv_sub.c]

Subtracts two timeval structures

```
#include  "unp.h"

void
tv_sub(struct timeval *out, struct timeval *in)
{
    if ( (out->tv_usec -= in->tv_usec) < 0) {
            /* out -= in */
        --out->tv_sec;
        out->tv_usec += 1000000;
    }
    out->tv_sec -= in->tv_sec;
}
```

sig_alrm Function – [ping/sig_alrm.c]

SIGALRM signal handler

```
#include  "ping.h"

void
sig_alrm(int signo)
{
    (*pr->fsend)();

    alarm(1);
    return;
}
```

send_v4 Function – [ping/send_v4.c]

Builds an ICMPv4 echo request message and sends it

- 7-13: build ICMPv4 message
- 14-16: calculate ICMP checksum
- 17: send datagram

```
1  #include  "ping.h"

2  void
3  send_v4(void)
4  {
5      int         len;
6      struct icmp   *icmp;

7      icmp = (struct icmp *) sendbuf;
8      icmp->icmp_type = ICMP_ECHO;
9      icmp->icmp_code = 0;
10     icmp->icmp_id = pid;
11     icmp->icmp_seq = nsent++;
```

send_v4 Function – [ping/send_v4.c]

Builds an ICMPv4 echo request message and sends it

- 7-13: build ICMPv4 message
- 14-16: calculate ICMP checksum
- 17: send datagram

```
12      memset(icmp->icmp_data, 0xa5, datalen);
            /* fill with pattern */
13      Gettimeofday((struct timeval *) icmp->icmp_data,
            NULL);

14      len = 8 + datalen;/* checksum ICMP header and data */
15      icmp->icmp_cksum = 0;
16      icmp->icmp_cksum = in_cksum((u_short *) icmp, len);

17      Sendto(sockfd, sendbuf, len, 0, pr->sasend, pr->
            salen);
18  }
```

in_cksum Function – [libfree/in_cksum.c]

Calculate the Internet checksum

- Internet checksum algorithm

```
#include "unp.h"

uint16_t
in_cksum(uint16_t *addr, int len)
{
    int             nleft = len;
    uint32_t        sum = 0;
    uint16_t        *w = addr;
    uint16_t        answer = 0;

    while (nleft > 1)  {
        sum += *w++;
        nleft -= 2;
    }
```

in_cksum Function – [libfree/in_cksum.c]

Calculate the Internet checksum

- Internet checksum algorithm

```
        /* mop up an odd byte, if necessary */
    if (nleft == 1) {
       *(unsigned char *)(&answer) = *(unsigned char *)w ;
        sum += answer;
    }

        /* add back carry outs from top 16 bits to low 16
bits */
    sum = (sum >> 16) + (sum & 0xffff);
        /* add hi 16 to low 16 */
    sum += (sum >> 16);        /* add carry */
    answer = ~sum;             /* truncate to 16 bits */
    return(answer);
}
```

10.4 Introduction to Datalink Access

Providing access to the datalink layer

- Watching the packets received by the datalink layer
- Running certain programs as normal applications instead of as part of the kernel

Three common methods to access the datalink layer

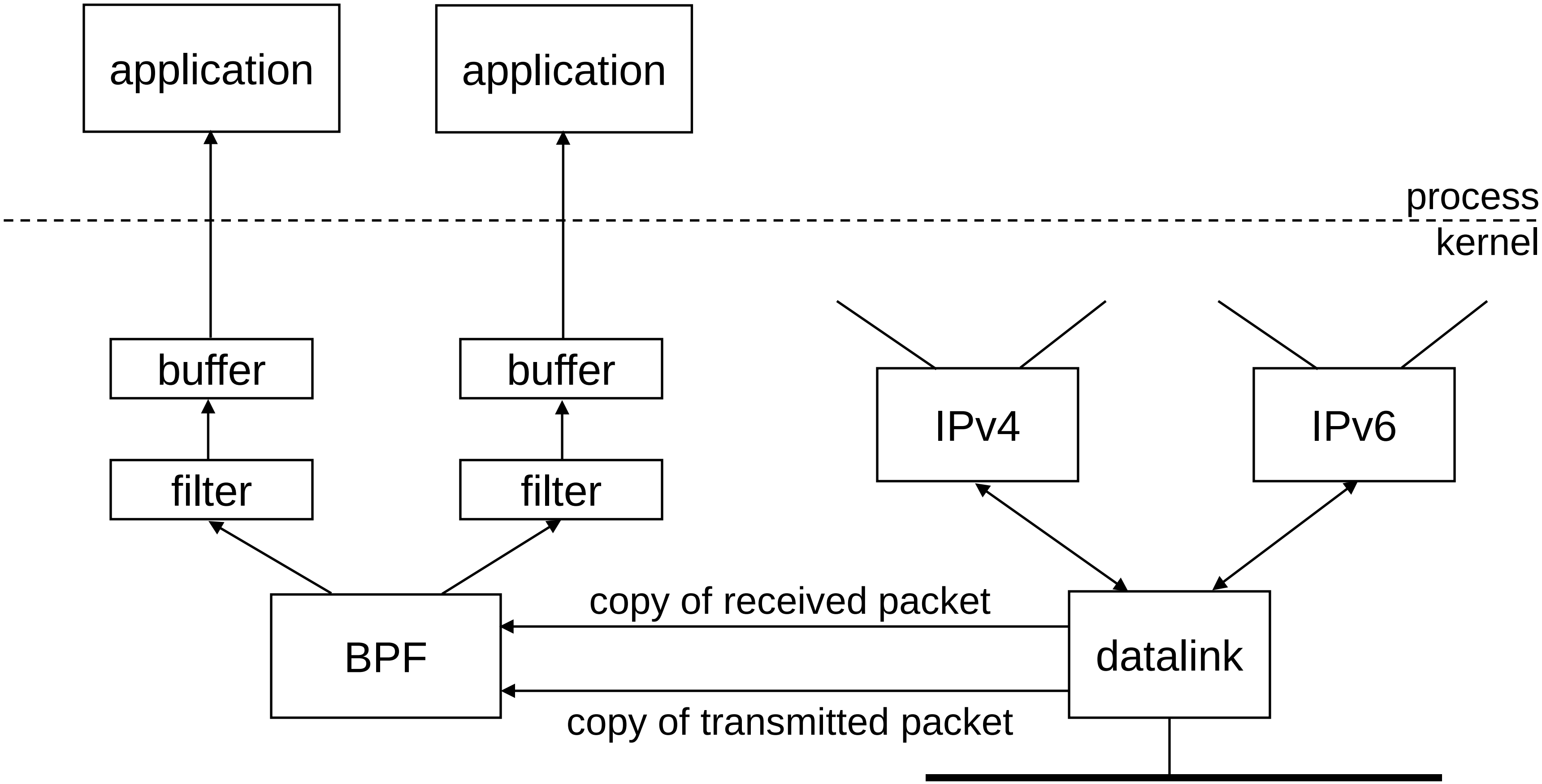
- BSD Packet Filter (BPF)
- SVR4 Datalink Provider Interface (DLPI)
- Linux SOCK_PACKET Interface

Publicly available packet capture library

- libpcap

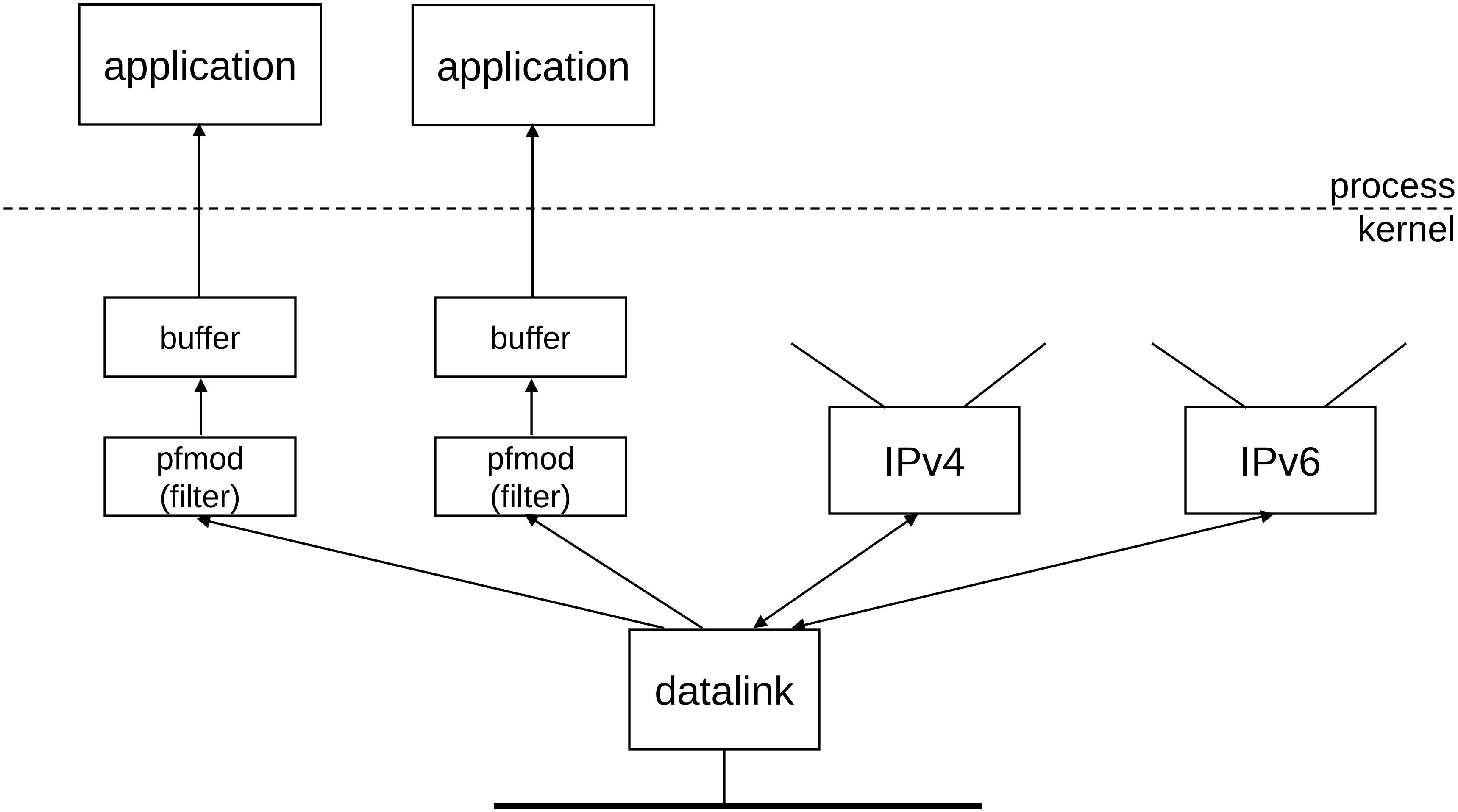
10.5 BSD Packet Filter (BPF)

Packet capture using BPF



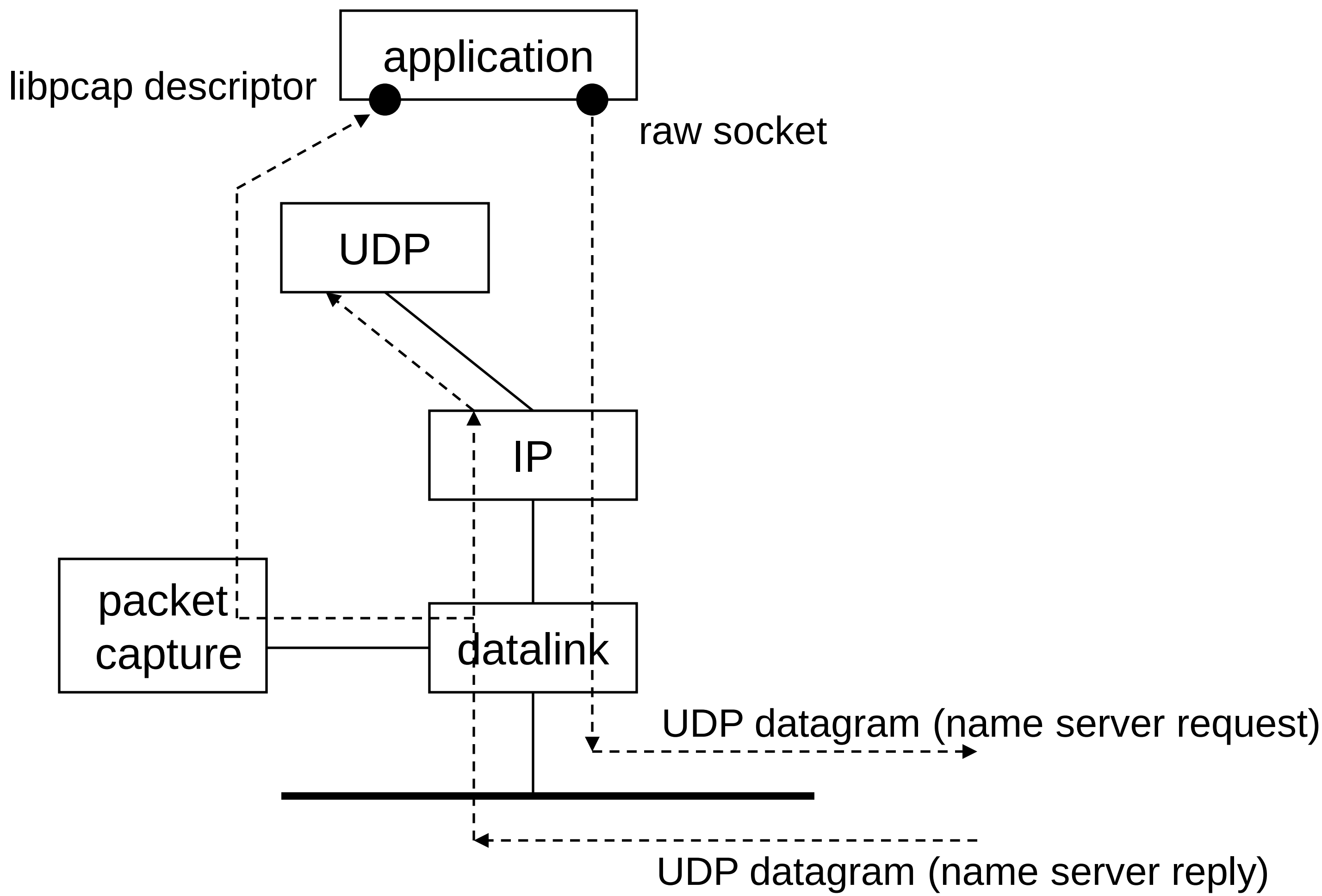
10.6 Datalink Provider Interface (DLPI)

Packet capture using DLPI, pfmod, and bufmod



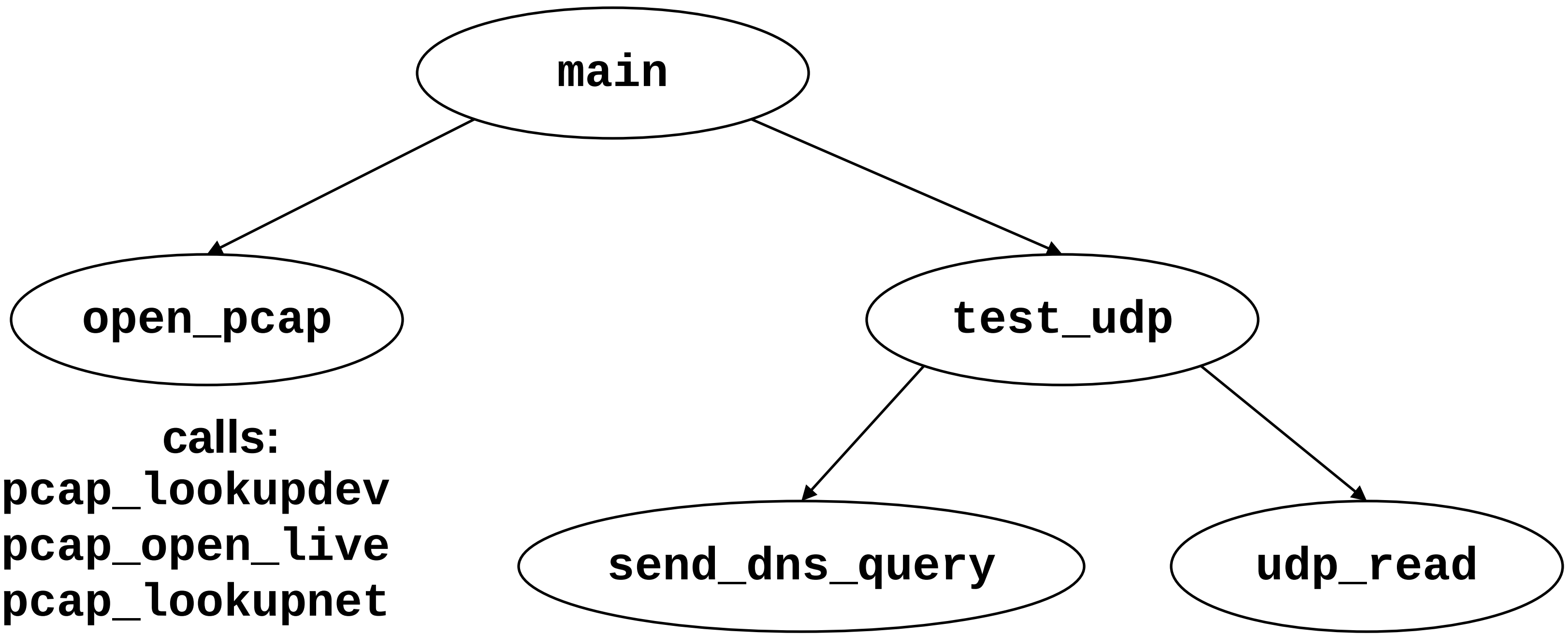
10.7 Examining the UDP Checksum Field

Application to check if a name server has UDP checksums enabled



udpcksum Program

Summary of fuctions for the udpcksum program



calls:

```
pcap_lookupdev
pcap_open_live
pcap_lookupnet
pcap_compile
pcap_setfilter
pcap_datalink
```

udpchsum.h Header – [udpcksum / udpcksum.h]

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

#include  <netinet/in_systm.h>/* required for ip.h */
#include  <netinet/in.h>
#include  <netinet/ip.h>
#include  <netinet/ip_var.h>
#include  <netinet/udp.h>
#include  <netinet/udp_var.h>
#include  <net/if.h>
#include  <netinet/if_ether.h>

#define   TTL_OUT      64             /* outgoing TTL */
```

udpchsum.h Header – [udpcksum/udpcksum.h]

```
                /* declare global variables */
extern struct sockaddr  *dest, *local;
extern socklen_t    destlen, locallen;
extern int    datalink;
extern char    *device;
extern pcap_t  *pd;
extern int    rawfd;
extern int    snaplen;
extern int    verbose;
extern int    zerosum;

                /* function prototypes */
void          cleanup(int);
char         *next_pcap(int *);
void          open_output(void);
void          open_pcap(void);
void          send_dns_query(void);
void          test_udp(void);
void          udp_write(char *, int);
struct udpiphdr *udp_read(void);
```

main Function: Definition – [udpcksum / main.c]

```
#include  "udpcksum.h"

          /* DefinE global variables */
struct sockaddr  *dest, *local;
struct sockaddr_in locallookup;
socklen_t    destlen, locallen;

int    datalink;
   /* from pcap_datalink(), in <net/bpf.h> */
char   *device;     /* pcap device */
pcap_t *pd;         /* packet capture struct pointer */
int    rawfd;       /* raw socket to write on */
int    snaplen = 200;   /* amount of data to capture */
int    verbose;
int    zerosum;  /* send UDP query with no checksum */
```

main Function: Definition – [udpcksum / main.c]

```
static void  usage(const char *);

int
main(int argc, char *argv[])
{
    int             c, lopt=0;
    char            *ptr, localname[1024], *localport;
    struct addrinfo  *aip;
```

main Function – [udpcksum / main.c]

Process command-line aruments

```
    opterr = 0;
        /* don't want getopt() writing to stderr */
    while ( (c = getopt(argc, argv, "0i:l:v")) != -1) {
        switch (c) {

        case '0':
            zerosum = 1;
            break;

        case 'i':
            device = optarg;        /* pcap device */
            break;
```

main Function – [udpcksum / main.c]

Process command-line arguments

```
    case 'l':
        /* local IP address and port #: a.b.c.d.p */
        if ( (ptr = strrchr(optarg, '.')) == NULL)
            usage("invalid -l option");

        *ptr++ = 0;   /* null replaces final period */
        localport = ptr;
            /* service name or port number */
        strncpy(localname, optarg, sizeof(localname));
        lopt = 1;
        break;
```

main Function – [udpcksum / main.c]

37-43: Process command-line arguments

46-49: process destination name and port

```
37         case 'v':
38             verbose = 1;
39             break;

40         case '?':
41             usage("unrecognized option");
42         }
43     }

44     if (optind != argc-2)
45         usage("missing <host> and/or <serv>");

46         /* 4convert destination name and service */
47    aip = Host_serv(argv[optind], argv[optind+1], AF_INET,
SOCK_DGRAM);
48     dest = aip->ai_addr;    /* don't freeaddrinfo() */
49     destlen = aip->ai_addrlen;
```

main Function – [udpcksum / main.c]

50-74: process local name and port

```
50     /*
51      * Need local IP address for source IP address for
          UDP datagrams.
52      * Can't specify 0 and let IP choose, as we need to
          know it for
53      * the pseudoheader to calculate the UDP checksum.
54      * If -l option supplied, then use those values;
          otherwise,
55      * connect a UDP socket to the destination to
          determine the right
56      * source address.
57      */
58     if (lopt) {
59             /* convert local name and service */
60         aip = Host_serv(localname, localport, AF_INET,
                     SOCK_DGRAM);
61         local = aip->ai_addr;  /* don't freeaddrinfo() */
62         locallen = aip->ai_addrlen;
```

main Function – [udpcksum / main.c]

50-74: process local name and port

```
63     } else {
64         int s;
65         s = Socket(AF_INET, SOCK_DGRAM, 0);
66         Connect(s, dest, destlen);
67     /* kernel chooses correct local address for dest */
68         locallen = sizeof(locallookup);
69          local = (struct sockaddr *)&locallookup;
70         Getsockname(s, local, &locallen);
71         if (locallookup.sin_addr.s_addr ==
                  htonl(INADDR_ANY))
72             err_quit("Can't determine local address - use
-l\n");
73         close(s);
74     }
```

main Function – [udpcksum / main.c]

75-76: create raw socket and open packet capture device

77-80: change permissions and establish signal handlers

81-82: perform test and cleanup

```
75     open_output();
           /* open output, either raw socket or libnet */

76     open_pcap();     /* open packet capture device */

77     setuid(getuid());
           /* don't need superuser privileges anymore */

78     Signal(SIGTERM, cleanup);
79     Signal(SIGINT, cleanup);
80     Signal(SIGHUP, cleanup);
81     test_udp();
82     cleanup(0);
83 }
```

open_pcap Function – [udpcksum / pcap.c]

Open and initialize packet capture device

- 10-14: choose packet capture device
- 15-17: open device

```
1  #include  "udpcksum.h"

2  #define   CMD    "udp and src host %s and src port %d"

3  void
4  open_pcap(void)
5  {
6      uint32_t           localnet, netmask;
7      char               cmd[MAXLINE],
                          errbuf[PCAP_ERRBUF_SIZE],
                          str1[INET_ADDRSTRLEN],
8                         str2[INET_ADDRSTRLEN];
9      struct bpf_program  fcode;
```

open_pcap Function – [udpcksum / pcap.c]

Open and initialize packet capture device

- 10-14: choose packet capture device
- 15-17: open device

```
10      if (device == NULL) {
11          if ( (device = pcap_lookupdev(errbuf)) == NULL)
12              err_quit("pcap_lookup: %s", errbuf);
13      }
14      printf("device = %s\n", device);

15          /* hardcode: promisc=0, to_ms=500 */
16      if ( (pd = pcap_open_live(device, snaplen, 0, 500,
errbuf)) == NULL)
17          err_quit("pcap_open_live: %s", errbuf);
```

open_pcap Function – [udpcksum / pcap.c]

18-23: obtain network address and subnet mask

24-30: compile packet filter

```
18     if (pcap_lookupnet(device, &localnet, &netmask, errbuf) < 0)
19         err_quit("pcap_lookupnet: %s", errbuf);
20     if (verbose)
21         printf("localnet = %s, netmask = %s\n",
22 Inet_ntop(AF_INET, &localnet, str1, sizeof(str1)),
23 Inet_ntop(AF_INET, &netmask, str2, sizeof(str2)));

24     snprintf(cmd, sizeof(cmd), CMD,
25              Sock_ntop_host(dest, destlen),
26              ntohs(sock_get_port(dest, destlen)));
27     if (verbose)
28         printf("cmd = %s\n", cmd);
29     if (pcap_compile(pd, &fcode, cmd, 0, netmask) < 0)
30         err_quit("pcap_compile: %s", pcap_geterr(pd));
```

open_pcap Function – [udpcksum / pcap.c]

31-32: load filter program

33-36: determine datalink type

```
31      if (pcap_setfilter(pd, &fcode) < 0)
32          err_quit("pcap_setfilter: %s", pcap_geterr(pd));

33      if ( (datalink = pcap_datalink(pd)) < 0)
34          err_quit("pcap_datalink: %s", pcap_geterr(pd));
35      if (verbose)
36          printf("datalink = %d\n", datalink);
37  }
```

test_udp Function – [udpcksum / udpcksum.c]

sig_alrm function: handles SIGALRM signal

```
#include  "udpcksum.h"
#include  <setjmp.h>

static sigjmp_buf   jmpbuf;
static int       canjump;

void
sig_alrm(int signo)
{
    if (canjump == 0)
        return;
    siglongjmp(jmpbuf, 1);
}
```

test_udp Function – [udpcksum / udpcksum.c]

Send queries and read responses

- 15: volatile varialbes
- 17-18: establish signal handler and jump buffer
- 19-23: handle siglongjmp

```
12 void
13 test_udp(void)
14 {
15     volatile int  nsent = 0, timeout = 3;
16     struct udpiphdr  *ui;

17     Signal(SIGALRM, sig_alrm);
```

test_udp Function – [udpcksum / udpcksum.c]

Send queries and read responses

- 15: volatile varialbes
- 17-18: establish signal handler and jump buffer
- 19-23: handle siglongjmp

```
18     if (sigsetjmp(jmpbuf, 1)) {
19         if (nsent >= 3)
20             err_quit("no response");
21         printf("timeout\n");
22         timeout *= 2; /* exponential backoff: 3, 6, 12 */
23     }
24     canjump = 1;  /* siglongjmp is now OK */
```

test_udp Function – [udpcksum/udpcksum.c]

25-30: send DNS query and read reply

31-36: examine received UDP checksum

```
25      send_dns_query();
26      nsent++;

27      alarm(timeout);
28      ui = udp_read();
29      canjump = 0;
30      alarm(0);

31      if (ui->ui_sum == 0)
32          printf("UDP checksums off\n");
33      else
34          printf("UDP checksums on\n");
35      if (verbose)
36          printf("received UDP checksum = %x\n", ntohs(ui-
37  >ui_sum));
    }
```

send_dns_query Function – [udpcksum / senddnsquery-raw.c]

11-12: allocate buffer and initialize pointer

```
1  #include  "udpcksum.h"

2  /*
3   * Build a DNS A query for "a.root-servers.net" and
4  write it tothe raw socket.
5   */

6  void
7  send_dns_query(void)
8  {
9      size_t    nbytes;
10     char      *buf, *ptr;

11     buf = Malloc(sizeof(struct udpiphdr) + 100);
12     ptr = buf + sizeof(struct udpiphdr);    /* leave room
   for IP/UDP headers */
```

send_dns_query Function – [udpcksum/senddnsquery-raw.c]

13-24: build DNS query

```
13      *((uint16_t *) ptr) = htons(1234);
            /* identification */
14      ptr += 2;
15      *((uint16_t *) ptr) = htons(0x0100);
            /* flags: recursion desired */
16      ptr += 2;
17      *((uint16_t *) ptr) = htons(1);
            /* # questions */
18      ptr += 2;
19      *((uint16_t *) ptr) = 0;   /* # answer RRs */
20      ptr += 2;
21      *((uint16_t *) ptr) = 0;   /* # authority RRs */
22      ptr += 2;
23      *((uint16_t *) ptr) = 0;   /* # additional RRs */
24      ptr += 2;
```

send_dns_query Function – [udpcksum / senddnsquery-raw.c]

31-32: write UDP datagram

```
25      memcpy(ptr, "\001a\014root-servers\003net\000", 20);
26      ptr += 20;
27      *((uint16_t *) ptr) = htons(1);/* query type = A */
28      ptr += 2;
29      *((uint16_t *) ptr) = htons(1);
            /* query class = 1 (IP addr) */
30      ptr += 2;

31      nbytes = (ptr - buf) - sizeof(struct udpiphdr);
32      udp_write(buf, nbytes);
33      if (verbose)
34          printf("sent: %d bytes of data\n", nbytes);
35  }
```

open_output Function – [udpcksum / udpwrite.c]

2: declare raw socket descriptor

7-13: create raw socket and enable IP_HDRINCL

```
1  #include  "udpcksum.h"

2  int    rawfd;        /* raw socket to write on */

3  void
4  open_output(void)
5  {
6      int on=1;
7      /*
8       * Need a raw socket to write our own IP datagrams to.
9  Process must have superuser privileges to create this
10 socket. Also must set IP_HDRINCL so we can write our own
11 IP headers. */
```

open_output Function – [udpcksum/udpwrite.c]

2: declare raw socket descriptor

7-13: create raw socket and enable IP_HDRINCL

```
12     rawfd = Socket(dest->sa_family, SOCK_RAW, 0);
13     Setsockopt(rawfd, IPPROTO_IP, IP_HDRINCL, &on, sizeof(on));
14 }
```

udp_write Function – [udpcksum/udpwrite.c]

Build UDP and IP headers and write IP datagram to raw socket

- 24-26: initialize packet header pointers
- 27: zero header
- 28-31: update lengths

```
19 void
20 udp_write(char *buf, int userlen)
21 {
22     struct udpiphdr     *ui;
23     struct ip        *ip;

24         /* fill in and checksum UDP header */
25     ip = (struct ip *) buf;
26     ui = (struct udpiphdr *) buf;
27     bzero(ui, sizeof(*ui));
28             /* add 8 to userlen for pseudoheader length */
29     ui->ui_len = htons((uint16_t) (sizeof(struct udphdr)
30 + userlen)); /* then add 28 for IP datagram length */
31     userlen += sizeof(struct udpiphdr);
```

udp_write Function – [udpcksum / udpwrite.c]

32-45: fill in UDP header and calculate UDP checksum

```
32      ui->ui_pr = IPPROTO_UDP;
33      ui->ui_src.s_addr = ((struct sockaddr_in *) local)->
                             sin_addr.s_addr;
34      ui->ui_dst.s_addr = ((struct sockaddr_in *) dest)->
                             sin_addr.s_addr;
35      ui->ui_sport = ((struct sockaddr_in *) local)->
                          sin_port;
36      ui->ui_dport = ((struct sockaddr_in *) dest)->
                          sin_port;
37      ui->ui_ulen = ui->ui_len;
```

udp_write Function – [udpcksum/udpwrite.c]

32-45: fill in UDP header and calculate UDP checksum

```
38     if (zerosum == 0) {
39 #if 1  /* change to if 0 for Solaris 2.x, x < 6 */
40         if ( (ui->ui_sum = in_cksum((u_int16_t *) ui, userlen)) == 0)
41             ui->ui_sum = 0xffff;
42 #else
43         ui->ui_sum = ui->ui_len;
44 #endif
45     }
```

udp_write Function – [udpcksum/udpwrite.c]

46-59: fill in IP header

```
46         /* fill in rest of IP header; */
47         /* ip_output() calcuates & stores IP header checksum */
48     ip->ip_v = IPVERSION;
49     ip->ip_hl = sizeof(struct ip) >> 2;
50     ip->ip_tos = 0;
51 #if defined(linux) || defined(__OpenBSD__)
52     ip->ip_len = htons(userlen);/* network byte order */
53 #else
54     ip->ip_len = userlen;          /* host byte order */
55 #endif
56     ip->ip_id = 0;   /* let IP set this */
57     ip->ip_off = 0;  /* frag offset, MF and DF flags */
58     ip->ip_ttl = TTL_OUT;

59     Sendto(rawfd, buf, userlen, 0, dest, destlen);
60 }
```

udp_read Function – [udpcksum / udpread.c]

```
#include  "udpcksum.h"

struct udpiphdr  *udp_check(char *, int);

/*
 * Read from the network until a UDP datagram is read
that matches the arguments.
 */

struct udpiphdr *
udp_read(void)
{
    int                 len;
    char                *ptr;
    struct ether_header *eptr;
```

udp_read Function – [udpcksum / udpread.c]

```
    for ( ; ; ) {
        ptr = next_pcap(&len);

        switch (datalink) {
        case DLT_NULL:   /* loopback header = 4 bytes */
            return(udp_check(ptr+4, len-4));

        case DLT_EN10MB:
            eptr = (struct ether_header *) ptr;
            if (ntohs(eptr->ether_type) != ETHERTYPE_IP)
        err_quit("Ethernet type %x not IP", ntohs(eptr->ether_type));
            return(udp_check(ptr+14, len-14));
```

udp_read Function – [udpcksum / udpread.c]

```
        case DLT_SLIP:   /* SLIP header = 24 bytes */
            return(udp_check(ptr+24, len-24));

        case DLT_PPP: /* PPP header = 24 bytes */
            return(udp_check(ptr+24, len-24));

        default:
            err_quit("unsupported datalink (%d)",
datalink);
        }
    }
}
```

next_pcap Function – [udpcksum / pcap.c]

Return next packet

```
char *
next_pcap(int *len)
{
    char              *ptr;
    struct pcap_pkthdr  hdr;

        /* keep looping until packet ready */
    while ( (ptr = (char *) pcap_next(pd, &hdr)) == NULL)
        ;

    *len = hdr.caplen;  /* captured length */
    return(ptr);
}
```

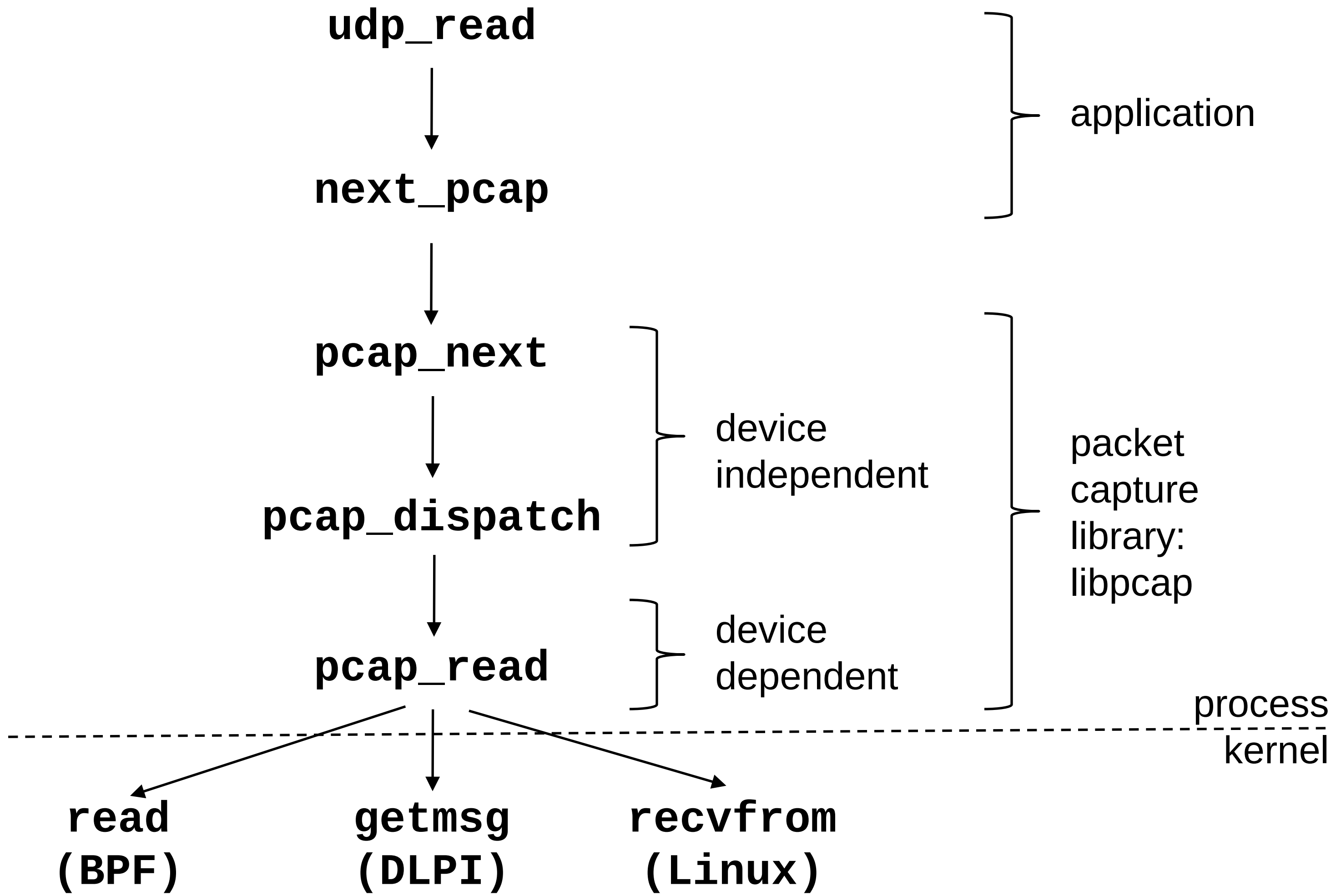
next_pcap Function
– [udpcksum / pcap.c]

Return next packet

```
struct pcap_pkthdr {
    struct timeval   ts;
/* timestamp */
    bpf_u_int32      caplen;
/* length of portion captured */
    bpf_u_int32      len;
/* length of this packet (off wire) */
};
```

Arrangement of Function Calls

Read from packet capture library



cleanup Function – [udpcksum / cleancup.c]

```
#include  "udpcksum.h"

void
cleanup(int signo)
{
    struct pcap_stat stat;

    putc('\n', stdout);

    if (verbose) {
        if (pcap_stats(pd, &stat) < 0)
            err_quit("pcap_stats: %s\n", pcap_geterr(pd));
        printf("%d packets received by filter\n", stat.ps_recv);
        printf("%d packets dropped by kernel\n", stat.ps_drop);
    }

    exit(0);
}
```

pcap_check Function – [udpcksum / udpread.c]

```
struct udpiphdr *
udp_check(char *ptr, int len)
{
    int             hlen;
    struct ip       *ip;
    struct udpiphdr     *ui;

    if (len < sizeof(struct ip) + sizeof(struct udphdr))
        err_quit("len = %d", len);

        /* minimal verification of IP header */
    ip = (struct ip *) ptr;
    if (ip->ip_v != IPVERSION)
        err_quit("ip_v = %d", ip->ip_v);
    hlen = ip->ip_hl << 2;
    if (hlen < sizeof(struct ip))
        err_quit("ip_hl = %d", ip->ip_hl);
    if (len < hlen + sizeof(struct udphdr))
        err_quit("len = %d, hlen = %d", len, hlen);
```

pcap_check Function – [udpcksum / udpread.c]

```
    if ( (ip->ip_sum = in_cksum((uint16_t *) ip, hlen)) != 0)
        err_quit("ip checksum error");

    if (ip->ip_p == IPPROTO_UDP) {
        ui = (struct udpiphdr *) ip;
        return(ui);
    } else
        err_quit("not a UDP packet");
}
```

실습 과제

네트워크 프로그램 디버깅 툴 작성

- 자신만의 디버깅 툴 설계 및 구현