Socket

Sunate Hwang
Kookmin University

Introduction

- One of the nice things about UNIX is that it uses a common interface for the access of files and devices that reside on a single host.

- user는 file descriptor가 어떤 device에 연결 되어 있는지 상관 없이 마치 file에서 읽고 쓰는 것 처럼 프로그램을 작성할 수 있다.

- 마찬 가지로 pipe등을 이용하면 각 related process들은 file을 다루는 것처럼 read와 write에 의해서 process간에 communication을 할 수 있다.

Introduction(cont'd)

- System V 계열의 message queue, semaphore, shared memory등은 앞의 read/write paradigm에서는 벗어나는 것이다. 이들은 각각 자신의 방법으로 ending/receiving을 해결한다. 이들이 unrelated process까지 포함하여 interprocess communication에 이용될 수 있지만 다소 제한적이다.

- RPC는 분산 환경에서 unrelated process간에 communication이 필요한 응용을 좀더 쉽게 작성할 수 있도록 고안되었다. 하지만 오히려 더 복잡하고 제한적인 면이 있다.

Introduction(cont'd)

- □It would seem that what is needed is an extension of the read/write paradigm with the inclusion of sufficient networking semantics to permit unrelated processes, on different hosts, to communicate d if they were reading and writing to local file.
- □This sort of intermediate level of interprocess communications would lie somewhere in between pipes, message queues, shared memory techniques and RPC applications.
- □이와 같은 type의 communication을 허용하는 interface가 있는데 가장 대표적인 것이 Berkeley socket과 AT&T의 TLI(Transport Level Interface)이다.
- □우리는 socket에 대해서만 고찰 하기로 한다.

Introduction(cont'd)

□ A socket is an abstract data structure that is used to create a channel (connection point) to send and receive information between unrelated processes. Once a channel is established, the connected processes can use generalized file system type access routines for communication.

□ For most part, when using a socket-based connection, the server process creates a socket, maps the socket to a local address, and waits (listens) for requests from clients.

Introduction(cont'd)

□ The client process creates its own socket and determines the location specifics(such as a host name and port number) of the server.

□ Depending upon the type of transport/connection used, the client process will begin to send and receive data, either with or without receiving a formal acknowledgment (acceptance) from the server process.

Socket communication domain

- □different from network domain

- □UNIX domain: In this domain, when sockets are created, they have actual file(path) names. These domain sockets can only be used with processes that reside on the same host. Sometimes used as the first step in the development of socket-based communications

- □Internet domain: allow unrelated processes on different hosts to communicate.

Protocol family

- Processes must also upon a set of rules and conventions for their communications. A set of such rules and conventions is called a protocol.

- ISO/OSI의 7 layer중 transport와 network layer를 protocol family로 묶는다.

- PF_UNIX(UNIX) or PF_INET(TCP/IP)

Socket types

□ Stream sockets:

- 전화에 비유할 수 있다
- reliable
- data is delivered in order, in the same sequence in which it was sent.
- There is no duplication of data, and some type of error checking and flow control is usually present
- allow bi-directional(full duplex) communication
- connection oriented. That is, a logical connection is created by the two processes using the socket.
- Information concerning the connection is established prior to the transmission of data and is maintained by each end of the connection during the communication.
- Data is transmitted as a stream of bytes.

Socket types(2)

□Datagram sockets

- 우편 엽서에 비유할 수 있다
- unreliable
- received data may not be out of order
- support bi-directional communication but are considered conectionless
- no logical connection
- Each datagram is sent, and processed independently. (may take different routes to the same destination)
- no flow control
- Datagram packets are normally small and fixed in size.

□raw sockets

□sequenced sockets

pr10.1.c(1/2)

```
~sthwang/lecture/OSII/socket/pr10.1.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/scket.h>
#define BUF_SZ 10
main(void) {
  int       sock[2],  /* The scoket pair */
            cpid, i;
  static char buf[BUF_SZ];    /* Temporary buffer for message */
 if (socketpair(PF_UNIX, SOCK_STREAM, 0, sock) < 0) {
      perror("Generation error");
      exit(1);
 }
 switch (cpid = (int)fork()) {
    case -1:
      perror("Bad fork");
      exit(2);
    case 0:           /* The child process */
      close(sock[1]);
      for (i=0; i<10; i+=2) {
        sleep(1);
        sprintf(buf, "c: %d\n", i);
        write(sock[0], buf, sizeof(buf));
        read(sock[0], buf, BUF_SZ);
        printf("c-> %s", buf);          /* Message from parent */
        }
        close(sock[0]);
        break;
```

Protocol family: same host communucation으로 제한 된다

Socket type: SOCK_STREAM (connection-oriented) 또는 SOCK_DGRAM (connectionless) 중에 하나이다

Protocol:
0은 system이 protocol을 정한다는 의미이다.
Internet domain communication에서 system은 connectionless socket에는 UDP를 connection-oriented socket에는 TCP를 선택할 것이다.

pr10.1.c(2/2)

```
    default:          /* The parent process */
      close(sock[0]);
      for (i=1; i<10; i+=2) {
        sleep(1);
        read(sock[1], buf, BUF_SZ);
        printf("p-> %s", buf);          /*
Message from child */
        sprintf(buf, "p: %d\n", i);
        write(sock[1], buf, sizeof(buf));
        }
      close(sock[1]);
    }
    return 0;
 }
```

```
$ gcc -o  pr10.1 pr10.1.c -lsocket
$ pr10.1
p-> c: 0
c-> p: 1
p-> c: 2
c-> p: 3
p-> c: 4
c-> p: 5
p-> c: 6
c-> p: 7
p-> c: 8
c-> p: 9
```

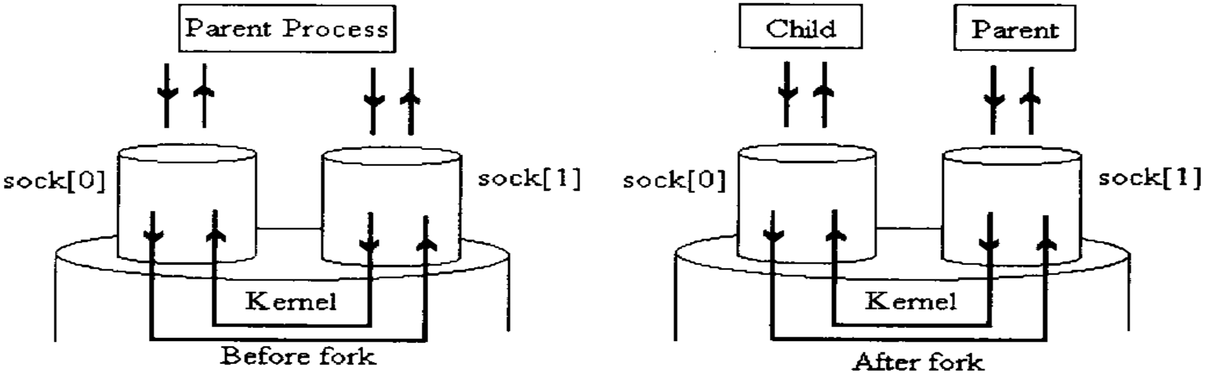

Fig. 10.5 The **socketpair** before and after the process forks.

Socketpair

NAME

socketpair - create a pair of connected sockets

SYNOPSIS

```
cc [ flag ... ] file ...  -lsocket -lnsl [ library ... ]
#include <sys/types.h>
#include <sys/socket.h>
int socketpair(int  domain,  int  type,  int  protocol,  int sv[2]);
```

DESCRIPTION

The socketpair() library call creates an unnamed pair of connected sockets in the specified address family d, of the specified type , and using the optionally specified protocol. The descriptors used in referencing the new sockets are returned in sv[0] and sv[1]. The two sockets are indistinguishable.

RETURN VALUES

socketpair() returns -1 on failure, and 0 on success.

The connection-oriented paradigm

socket network call return an integer that can be used to reference the socket descriptor. There is no name or address/port number pair associated with the socket

Bind network call is used to associate a name (in the UNIX domain) or address/port pair (in the internet domain) with a socket.
process가 server인 경우 socket은 반드시 bound되어야 하는데 이는 전화 번호, 주소등을 연상시킨다

Create a queue for incoming connection request

By default, the accept call will block, if there are no pending requests for connections. It will return a new socket descriptor that can be used for reading and writing

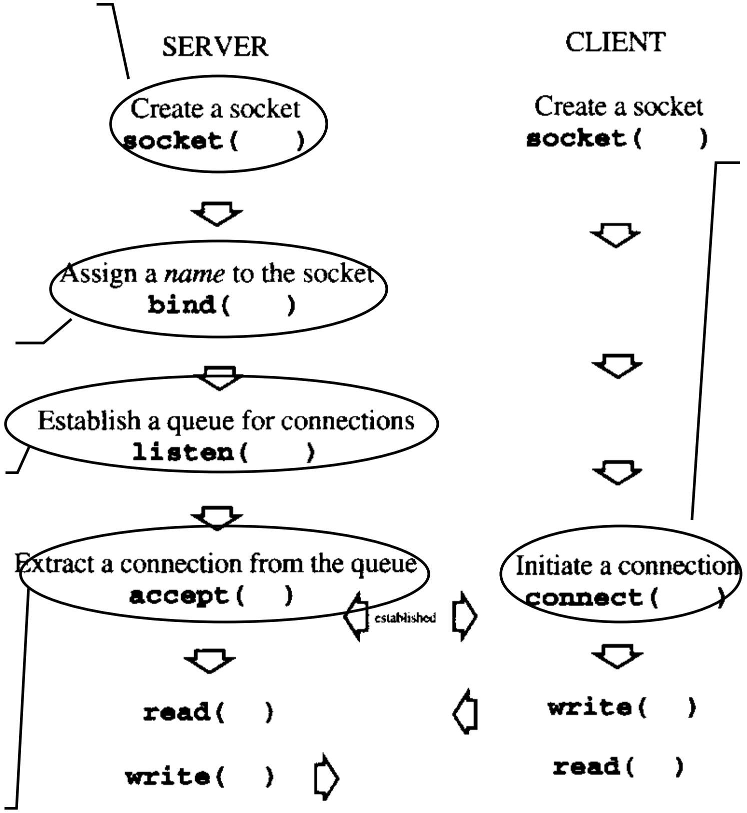

Fig. 10.6 A connection-oriented, client-server communication sequence.

connect는 struct sockaddr *name 이라는 매개 변수가 필요한데 이는connection-oriented setting에서는 communicate할 socket의 address 이고, connectionless에서는 datagram이 보내질 곳이다

The connection-oriented paradigm(cont’d)

```
in <sys/socket.h>
/*
 * Structure used by kernel to store most
 * addresses.
 */
struct sockaddr {
        u_short sa_family;              /* address family */
        char    sa_data[14];            /* up to 14 bytes of direct address */
};
in <sys/un.h>
/*
 * Definitions for UNIX IPC domain.
 */
struct  sockaddr_un {
        short   sun_family;             /* AF_UNIX */
        char    sun_path[108];          /* path name (gag) */
};
```

The connection-oriented paradigm(cont’d)

```
in <netinet/in.h>

/*
 * Internet address
 *       This definition contains obsolete fields for compatibility
 *       with SunOS 3.x and 4.2bsd.  The presence of subnets renders
 *       divisions into fixed fields misleading at best.  New code
 *       should use only the s_addr field.
 */
struct in_addr {
        union {
                struct { u_char s_b1, s_b2, s_b3, s_b4; } S_un_b;
                struct { u_short s_w1, s_w2; } S_un_w;
                u_long S_addr;
        } S_un;

 /*
 * Socket address, internet style.
 */
struct sockaddr_in {
        short   sin_family;
        u_short sin_port;
        struct  in_addr sin_addr;
        char    sin_zero[8];
};
```

Socket(1/2)

NAME

socket - create an endpoint for communication

SYNOPSIS

```
cc [ flag ... ] file ...  -lsocket -lnsl [ library ... ]
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
```

DESCRIPTION

socket() creates an endpoint for communication and returns a descriptor.

The domain parameter specifies a communications domain within which communication will take place; this selects the protocol family which should be used. The protocol family generally is the same as the address family for the addresses supplied in later operations on the socket. These families are defined in the include file <sys/socket.h>. There must be an entry in the netconfig(4) file for at least each protocol family and type required. If protocol has been specified, but no exact match for the tuplet family, type, protocol is found, then the first entry containing the specified family and type with zero for protocol will be used. The currently understood formats are:

PF_UNIX	UNIX system internal protocols
PF_INET	ARPA Internet protocols

Socket(2/2)

The socket has the indicated type, which specifies the communication semantics. Currently defined types are:

SOCK_STREAM
SOCK_DGRAM
SOCK_RAW
SOCK_SEQPACKET
SOCK_RDM

A SOCK_STREAM type provides sequenced, reliable, two-way connection-based byte streams. An out-of-band data transmission mechanism may be supported. A SOCK_DGRAM socket supports datagrams (connectionless, unreliable messages of a fixed (typically small) maximum length). . . . . . .

. . . . . .

RETURN VALUES

A -1 is returned if an error occurs. Otherwise the return value is a descriptor referencing the socket.

bind

NAME

bind - bind a name to a socket

SYNOPSIS

```
cc [ flag ... ] file ...  -lsocket -lnsl [ library ... ]
#include <sys/types.h>
#include <sys/socket.h>
int bind(int s, const struct sockaddr *name, int namelen);
```

DESCRIPTION

bind() assigns a name to an unnamed socket. When a socket is created with socket(3N), it exists in a name space (address family) but has no name assigned. bind() requests that the name pointed to by name be assigned to the socket.

RETURN VALUES

If the bind is successful, 0 is returned. A return value of - 1 indicates an error, which is further specified in the global errno.

Listen

NAME

listen - listen for connections on a socket

SYNOPSIS

```
cc [ flag ... ] file ...  -lsocket -lnsl [ library ... ]
#include <sys/types.h>
#include <sys/socket.h>
int listen(int s, int backlog);
```

DESCRIPTION

To accept connections, a socket is first created with socket(3N), a backlog for incoming connections is specified with listen()
and then the connections are accepted with accept(3N). The listen() call applies only to sockets of type SOCK_STREAM or SOCK_SEQPACKET.

The backlog parameter defines the maximum length the queue of pending connections may grow to If a connection request arrives with the queue full, the client will receive an error with an indication of ECONNREFUSED for AF_UNIX sockets. If the underlying protocol supports retransmission, the connection request may be ignored so that retries may succeed. For AF_INET sockets, the tcp will retry the connection. If the backlog is not cleared by the time the tcp times out, the connect will fail with ETIMEDOUT.

RETURN VALUES

A 0 return value indicates success; -1 indicates an error.

Accept(1/2)

NAME

accept - accept a connection on a socket

SYNOPSIS

```
cc [ flag ... ] file ...  -lsocket -lnsl [ library ... ]
#include <sys/types.h>
#include <sys/socket.h>
int accept(int s, struct sockaddr *addr, int *addrlen);
```

DESCRIPTION

The argument s is a socket that has been created with socket(3N) and bound to an address with bind(3N), and that is listening for connections after a call to listen(3N). accept() extracts the first connection on the queue of pending connections, creates a new socket with
the properties of s, and allocates a new file descriptor, ns, for the socket. If no pending connections are present on the queue and the socket
is not marked as non-blocking, accept() blocks the caller until a connection is present.   If the socket is marked as non-blocking and no pending connections are present on the queue, accept() returns an error as described below. accept() uses the netconfig(4) file to determine
the STREAMS device file name associated with s. This is the device on which the connect indication will be accepted. The accepted socket,
ns, is used to read and write data to and from the socket that connected to ns; it is not used to accept more connections.

Accept(2/2)

The original socket (s) remains open for accepting further connections.

The argument addr is a result parameter that is filled in with the address of the connecting entity as it is known to the communications
layer. The exact format of the addr parameter is determined by the domain in which the communication occurs.

addrlen is a value-result parameter. Initially, it contains the amount of space pointed to by addr; on return it contains the length in bytes
of the address returned.

accept() is used with connection-based socket types, currently with SOCK_STREAM.

It is possible to select(3C) or poll(2) a socket for the purpose of an accept() by selecting or polling it for a read. However, this will
only indicate when a connect indication is pending; it is still necessary to call accept().

RETURN VALUES

accept() returns -1 on error. If it succeeds, it returns a non-negative integer that is a descriptor for the accepted socket.

Connect(1/2)

NAME

connect - initiate a connection on a socket

SYNOPSIS

```
cc [ flag ... ] file ...  -lsocket -lnsl [ library ... ]

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

int connect(int s, struct sockaddr *name, int namelen);
```

MT-LEVEL

Safe

DESCRIPTION

The parameter s is a socket. If it is of type SOCK_DGRAM, connect() specifies the peer with which the socket is to be associated; this address is the address to which datagrams are to be sent if a receiver is not explicitly designated; it is the only address from which datagrams are to be received. If the socket s is of type SOCK_STREAM, connect() attempts to make a connection to another socket.

Connect(2/2)

The other socket is specified by name. name is an address in the communication space of the socket. Each communication space interprets the name parameter in its own way.
If s is not bound, then it will be bound to an address selected by the underlying transport provider.
Generally, stream sockets may successfully connect() only once; datagram sockets may use connect() multiple times to change their association.
Datagram sockets may dissolve the association by connecting to a null address.

RETURN VALUES
If the connection or binding succeeds, 0 is returned. Otherwise, -1 is returned and sets errno to indicate the error.

Server.c(1/3)

```
"server.c"
/*
 * Program 10.2: Server - UNIX domain, connection-oriented
 */
#include "local.h"
main(void) {
  int    orig_sock,       /* Original socket descriptor in server */
         new_sock,        /* New socket descriptor from connect      */
         clnt_len,        /* Length of client address                */
         i;               /* Loop counter                            */
  static struct sockaddr_un
                 clnt_adr,        /* UNIX addresses of client-server */
                 serv_adr;
  static char    buf[10]; /* Buffer for messages               */
  void clean_up(int, char *);     /* Close socket and remove routine */

  if ((orig_sock = socket(AF_UNIX, SOCK_STREAM, 0)) < 0) { /* SOCKET */
    perror("generate error");
    exit(1);
    }
  serv_adr.sun_family = AF_UNIX;  /* Set tag appropriately  */
  strcpy(serv_adr.sun_path,NAME); /* Assign name (108 char max)      */
  unlink(NAME);                           /* Remove old copy if present
         */
```

Server.c(2/3)

```
if (bind(orig_sock, (struct sockaddr *)&serv_adr, /* BIND */
       sizeof(serv_adr.sun_family)+strlen(serv_adr.sun_path)) < 0) {
  perror("bind error");
  clean_up(orig_sock,NAME);
  exit(2);
  }
listen(orig_sock, 1);                                      /* LISTEN */
clnt_len = sizeof(clnt_adr);
if ((new_sock = accept(orig_sock, (struct sockaddr *)&clnt_adr,
                                 &clnt_len)) < 0) {        /* ACCEPT */
  perror("accept error");
  clean_up(orig_sock,NAME);
  exit(3);
  }
for (i=1; i<=10; i++) {
  sleep(1);
  read(new_sock, buf, sizeof(buf));
  printf("s-> %s", buf);
  }
close(new_sock);
clean_up(orig_sock,NAME);
exit(0);
```

Server.c(2/3)

```
void clean_up(int sd, char *the_file) {
  close(sd);              /* close it */
  unlink(the_file);       /* rm it */
  }
```

```
“local.h”
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>       /* as we are using UNIX protocol */
#define NAME "my_sock"
```

Client.c (1/2)

```
“client.c”
/*
 * Program 10.3: Client - UNIX domain, connection-oriented
 */
#include "local.h"
main(void) {
  int             orig_sock,       /* Original socket descriptor in client
*/
                  i;               /* Loop counter                    */
  static struct sockaddr_un
                  serv_adr;        /* UNIX address of the server process */
  static char     buf[10]; /* Buffer for messages                  */

  if ((orig_sock = socket(AF_UNIX, SOCK_STREAM, 0)) < 0) { /* SOCKET */
    perror("generate error");
    exit(1);
    }
  serv_adr.sun_family = AF_UNIX;   /* Set tag appropriately */
  strcpy(serv_adr.sun_path,NAME);  /* Assign name             */
  if (connect(orig_sock, (struct sockaddr *)&serv_adr,      /* CONNECT */
          sizeof(serv_adr.sun_family)+strlen(serv_adr.sun_path)) < 0) {
    perror("connect error");
    exit(1);
    }
```

Client.c (2/2)

```
  for (i=1; i<=10; i++) {
    sprintf(buf, "c: %d\n", i);
    write(orig_sock, buf, sizeof(buf));
    }
  close(orig_sock);
  exit(0);
  }
```

```
sizzle:~/lecture/OSII/socket/stream/UNIX $make
sizzle:~/lecture/OSII/socket/stream/UNIX$ server&
•836
sizzle:~/lecture/OSII/socket/stream/UNIX$ ls -l my_sock
p---------   1 sthwang  faculty        0 Oct 27 22:30 my_sock
sizzle:~/lecture/OSII/socket/stream/UNIX$ client
sizzle:~/lecture/OSII/socket/stream/UNIX$ s-> c: 1
s-> c: 2
s-> c: 3
s-> c: 4
s-> c: 5
s-> c: 6
s-> c: 7
s-> c: 8
s-> c: 9
s-> c: 10
•+  Done                    server
sizzle:~/lecture/OSII/socket/stream/UNIX$
```

```
"Makefile"
CC=gcc
all: server client
server: server.c local.h
        $(CC) -o server
server.c -lsocket
client: client.c local.h
        $(CC) -o client
client.c -lsocket
clean:
        rm -f *.o server
client
```

Internet domain stream socket example

□ In the internet domain, processes must have address and port information to communicate.

□The gethostbyname network call will return information about a specific host when passed its name.

pr10.4.c(1/3)

```
in <netdb.h>
    struct hostent {
        char   *h_name;         /* canonical name of host */
        char   **h_aliases;     /* alias list */
        int    h_addrtype;      /* host address type */
        int    h_length;        /* length of address */
        char   **h_addr_list;   /* list of addresses */
    };
```

```
/*
 * Program 10.4: Checking host entries
 */
#include <stdio.h>
#include <sys/types.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
```

pr10.4.c(2/3)

```
main(void) {
  struct hostent *host;
  static char who[10];
  printf("Enter host name to look up: ");
  scanf("%10s",who);
  host = gethostbyname(who);
  if (host!=(struct hostent *)NULL) {
    printf("Here is what I found about %s :\n",who);
    printf("Official name : %s\n", host->h_name);
    printf("Aliases       : ");
    while (*host->h_aliases) {
        printf("%s  ",*host->h_aliases);       ++host->h_aliases;
      }
    printf("\nAddress type  : %i\n", host->h_addrtype);
    printf("Address length: %i\n",host->h_length);
    printf("Address list  : ");
    while (*host->h_addr_list) {
      struct in_addr in;
      memcpy(&in.s_addr, *host->h_addr_list, sizeof(in.s_addr));
      printf("[%s] = %s  ",*host->h_addr_list,inet_ntoa(in));
      ++host->h_addr_list;
      }
    printf("\n");
    }
  }
```

Translate the character encoded network address referenced by the `h_addr_list` member into the more standard dotted notation. Refer the manual page...

pr10.4.c(3/3)

```
sizzle:~/lecture/OSII/socket$gcc -o pr10.4 pr10.4.c -lnsl
sizzle:~/lecture/OSII/socket$ pr10.4
Enter host name to look up: cs.kwangwoon
sizzle:~/lecture/OSII/socket$ pr10.4
Enter host name to look up: cs00
Here is what I found about cs00 :
Official name : cs00
Aliases       :
Address type   : 2
Address length: 4
Address list   : [?'h] = 210.123.39.104
sizzle:~/lecture/OSII/socket$ pr10.4
Enter host name to look up: cs
Here is what I found about cs :
Official name : cs Aliases       : loghost
Address type   : 2
Address length: 4
Address list   : [?'-] = 210.123.39.31
```

In an internet domain setting, we can expect these value to be 2 (the value of AF_INET) and 4 (the number of bytes needed to store an integer value).

Internet domain stream socket example

- □ In addition to knowing the server’s 32-bit internet address, the client must also be able to make reference to a particular service at a given port on the server.

- □ There are some TCP- and UDP-based well-known ports which have standard services associated with them.

Internet domain stream socket example

- □ In /etc/services

```
#
# Network services, Internet style
#
tcpmux      1/tcp
echo        7/tcp
echo        7/udp
discard     9/tcp       sink null
discard     9/udp       sink null
systat      11/tcp      users
daytime     13/tcp
daytime     13/udp
netstat     15/tcp
chargen     19/tcp      ttytst source
chargen     19/udp      ttytst source
ftp-data    20/tcp
ftp         21/tcp
telnet      23/tcp
smtp        25/tcp      mail
time        37/tcp      timserver
time        37/udp      timserver
name        42/udp      nameserver
whois       43/tcp      nicname     # usually to sri-nic
domain      53/udp
domain      53/tcp
hostnames   101/tcp     hostname    # usually to sri-nic
sunrpc      111/udp     rpcbind
sunrpc      111/tcp     rpcbind
```

Internet domain stream socket example

□Port number <1024 :

- reserved for processes with an effective ID of root

□Port >=1024 :

- may be used by any system user

□An application can issue the getservbyname network call to obtain information about a particular service/port.

pr10.5.c(1/3)

```
/*
 * Program 10.5: Checking service -- port entries for a host
 */
#include <stdio.h>
#include <netdb.h>
#include <netinet/in.h>
main(void) {
  struct servent *serv;
  static char protocol[10], service[10];
  printf("Enter service to look up : ");
  scanf("%9s", service);
  printf("Enter protocol to look up: ");
  scanf("%9s", protocol);
  serv = getservbyname(service,protocol);
  if (serv!=(struct servent *)NULL) {
    printf("Here is what I found \n");
    printf("Official name  : %s\n", serv->s_name);
    printf("Aliases        : ");
    while (*serv->s_aliases) {
      printf("%s  ", *serv->s_aliases);
      ++serv->s_aliases;
      }
```

pr10.5.c(2/3)

```
 printf("\nPort number     : %i\n", htons(serv->s_port));
    printf("Protocol Family: %s\n\n", serv->s_proto);
    }
  else
    printf("Service %s for protocol %s not found\n",
service,protocol);
  }
```

```
sizzle:~/lecture/OSII/socket$gcc -o pr10.5 pr10.5.c -lsocket
sizzle:~/lecture/OSII/socket$ pr10.5
Enter service to look up : mail
Enter protocol to look up: tcp
Here is what I found
Official name   : smtp
Aliases         : mail
Port number     : 25
Protocol Family: tcp

sizzle:~/lecture/OSII/socket$ pr10.5
Enter service to look up : rpcbind
Enter protocol to look up: udp
Here is what I found
Official name   : sunrpc
Aliases         : rpcbind
Port number     : 111
Protocol Family: udp
```

pr10.5.c(3/3)

```
"Makefile"

CC=gcc

all: server client

server: server.c local.h
        $(CC) -o server server.c -lsocket

client: client.c local.h
        $(CC) -o client client.c -lsocket -lnsl

clean:
        rm -f *.o server client
```

```
/* local.h */
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define PORT 6996
static char       buf[BUFSIZ];
```

The value for the port should be one that is currently not in use and is greater than or equal to 1024.
An alternate approach would be to add an entry for the port in the `/etc/services` file. The port information then be obtained dynamically with the `getservbyname` network call.

Pro10.6.c(1/3)

```
/*
 * Program 10.6: Server - Internet domain, connection-oriented
 */
#include "local.h”
main(void) {
  int    orig_sock,         /* Original socket descriptor in server */
         new_sock,          /* New socket descriptor from connect      */
         clnt_len;          /* Length of client address                */
  struct sockaddr_in
                 clnt_adr,         /* Internet addresses of client & server
*/
                 serv_adr;
  int    len, i;            /* Misc counters, etc.                     */
  static char    buf[10]; /* Buffer for messages                 */

  if ((orig_sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) { /* SOCKET */
Ensure byte ordering
    perror("generate error");     exit(1);
    }
  memset(&serv_adr, 0, sizeof(serv_adr));  /* Clear structure         */
  serv_adr.sin_family         = AF_INET;            /* Set address type
        */
  serv_adr.sin_addr.s_addr = htonl(INADDR_ANY);     /* Any interface */
  serv_adr.sin_port         = htons(PORT);  /* Use our fake port*/
```

Ensure byte ordering

Pro10.6.c(2/3)

```
if (bind(orig_sock, (struct sockaddr *)&serv_adr,   /* BIND */
         sizeof(serv_adr)) < 0) {
    perror("bind error");
   close(orig_sock);
    exit(2);
    }
  if (listen(orig_sock, 5)<0) {                                  /* LISTEN */
    perror("listen error");
    exit(3);
    }
  do {
    clnt_len = sizeof(clnt_adr);
    if ((new_sock = accept(orig_sock, (struct sockaddr
*)&clnt_adr,&clnt_len)) < 0) {     /* ACCEPT */
```

Maximum size of the queue

Pro10.6.c(3/3)

```
    perror("accept error");
    close(orig_sock);
    exit(4);
    }
  if (fork()==0) {                                /* In CHILD process
       */
    while((len=read(new_sock,buf,BUFSIZ))>0) {
      for (i=0; i<len; ++i)                       /* Change the case
       */
        buf[i] = toupper(buf[i]);
      write(new_sock, buf, len);         /* write it back */
      if (buf[0]=='.') break;                     /* are we done yet?
       */
      }
    close(new_sock);                              /* In CHILD process
       */
    exit(0);
    }
  else close(new_sock);                  /* In PARENT process      */
  } while(1);                                     /* FOREVER       */
}
```

Peeking과 Out of Band Message에서 바뀔 부분

Pro10.7.c(1/4)

```c
/*
 * Program 10.7: Client - Internet domain, connection-oriented
 */
#include "local.h"

main(int argc, char *argv[]) {
  int              orig_sock,        /* Original socket descriptor in client*/
                   len;              /* Length of server address         */
  struct sockaddr_in  serv_adr;      /* Internet address of the server process*/
  struct hostent *host;   /* The host (server)*/
    if (argc != 2) {                 /* Expect name of host on cmd line */
    fprintf(stderr,"usage: %s server\n", argv[0]);
    exit(1);
    }
  host = gethostbyname(argv[1]);     /* Get the host info      */
  if (host==(struct hostent *)NULL) {
    perror("gethostbyname ");
    exit(2);
    }
  memset(&serv_adr, 0, sizeof(serv_adr));  /* Clear the structure    */
  serv_adr.sin_family = AF_INET;           /* Set address type       */
  memcpy(&serv_adr.sin_addr, host->h_addr, host->h_length); /* Adr   */
  serv_adr.sin_port = htons(PORT);         /* Use our fake port      */
```

Pro10.7.c(2/4)

```
 if ((orig_sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) { /* SOCKET */
    perror("generate error");
    exit(3);
    }

/* CONNECT */
  if (connect(orig_sock, (struct sockaddr *)&serv_adr,      /* CONNECT */
         sizeof(serv_adr)) < 0) {
    perror("connect error");
    exit(4);
    }
  do {
    write(fileno(stdout),">",3);                     /* Prompt user    */
    if ((len=read(fileno(stdin),buf,BUFSIZ)) > 0) { /* Get input      */
      write(orig_sock,buf,len);                      /* Write to sck   */
      if ((len=read(orig_sock,buf,len)) > 0)         /* If returned    */
        write(fileno(stdout),buf,len);               /* display it.    */
      }
    } while (buf[0]!='.');
close(orig_sock);   exit(0);
  }
```

Peeking과 Out of Band Message에서 바뀔 부분

Pro10.7.c(3/4)

```
cs-cmn1:~$ cd lecture/OSII/socket/stream/internet
cs-cmn1:~/lecture/OSII/socket/stream/internet$ server&
[1] 7088
cs-cmn1:~/lecture/OSII/socket/stream/internet$ ps
   PID TTY      TIME CMD
  7086 pts/0    0:00 bash
  7088 pts/0    0:00 server
cs-cmn1:~/lecture/OSII/socket/stream/internet$ server
bind error: Address already in use
cs-cmn1:~/lecture/OSII/socket/stream/internet$ telnet cs
Trying 210.123.39.31...
Connected to cs.
Escape character is '^]'.


UNIX(r) System V Release 4.0 (cs)

login: sthwang
Password:
Last login: Tue Oct 28 13:46:33 from sizzle
cs:~$ cd lecture/OSII/socket/stream/internet
cs:~/lecture/OSII/socket/stream/internet$ client
usage: client server
cs:~/lecture/OSII/socket/stream/internet$ client cs-cmn1
>hello
HELLO >How is this?
HOW IS THIS?
>^Z
```

server를 background에서 실행한다.

Client를 잠시 suspend하고

Pro10.7.c(4/4)

```
[1]+  Stopped                 client cs-cmn1
cs:~/lecture/OSII/socket/stream/internet$ rsh cs-cmn1 ps -e
 7088 pts/0    0:00 server
 7092 pts/0    0:00server
 7095 ?        0:00 in.rshd
 7096 ?        0:00 ps
cs:~/lecture/OSII/socket/stream/internet$ fg
client cs-cmn1
hi
HI
>.
.
cs:~/lecture/OSII/socket/stream/internet$ ps
   PID TT       S  TIME COMMAND
  5242 pts/1    S  0:00 -bash
```

현재 command line에서의 RPC는 cs-cmn1에서만 허용하고 있다.

server가 2개가 되었다

client를 종료한다

byteorder, htonl, htons, ntohl, ntohs

NAME

byteorder, htonl, htons, ntohl, ntohs - convert values between host and network byte order

SYNOPSIS

```
#include <sys/types.h>
#include <netinet/in.h>
ulong htonl(u_long hostlong);
u_short htons(u_short hostshort);
u_long ntohl(u_long netlong);
u_short ntohs(u_short netshort);
```

MT-LEVEL

Safe

DESCRIPTION

These routines convert 16 and 32 bit quantities between network byte order and host byte order. On some architectures

these routines are defined as NULL macros in the include file <netinet/in.h>.

On other architectures, if their host byte order is different from network byte order, these routines are functional.

These routines are most often used in conjunction with Internet addresses and ports as returned by gethostent() and getservent(). (See gethostbyname(3N) and getservbyname(3N) respectively.)

SEE ALSO

gethostbyname(3N), getservbyname(3N)

Peeking a Data

□ recv, recvfrom, rcvmsg call은 received data를 소모하지 않고 볼 수 있다.

□The data will still be available for the next receive-type call

Server.c

```
/*
 * Program 10.14: Server - Internet domain, connection-oriented -
MSG_PEEK
 */
        /* same as Program 10.6 */

    if (fork()==0) {                                    /* In CHILD process
        */
         while((len=recv(new_sock,buf,BUFSIZ, MSG_PEEK))>0) {
             write(fileno(stdout), "Peeked and found: ",20);
             write(fileno(stdout),buf,len);                  /* show
peeked message    */
             if (!strncmp(buf,".done",len-1)) break;
                len = recv(new_sock,buf,BUFSIZ,0);  /* retrieves same
msg       */
                write(fileno(stdout),"Re-read buffer : ",20);
                write(fileno(stdout),buf,len);
         }
         write(fileno(stdout),"Leaving child process\n",23);
         close(new_sock);                               /* In CHILD process
        */
         exit(0);
    }
    else close(new_sock);                      /* In PARENT process      */
    } while(1);                                        /* FOREVER        */
}
```

receive하지만 data를 소모하지는 않는다

같은 message를 한번 더 읽을 수 있다

Client.c

```
Modified client  : ~sthwang/lecture/OSII/socket/stream/msg_peek/client.c
/*
 * Program 10.15: Client - Internet domain, connection-oriented - MSG_PEEK
 */
         /* same as Program 10.7 */

do {
    write(fileno(stdout),">",3);                    /* Prompt user   */
    if ((len=read(fileno(stdin),buf,BUFSIZ)) > 0) { /* Get input     */
      write(fileno(stdout),"Sending ",9);
      write(fileno(stdout),buf,len);
      send(orig_sock,buf,len,0);
      }
    } while (strncmp(buf,".done",len-1));
  close(orig_sock);
  exit(0);
  }
```

실행결과

```
server
cs:~$ cd
lecture/OSII/socket/stream/msg
_peek/
cs:~/lecture/OSII/socket/strea
m/msg_peek$ server
Peeked and found: hello
Re-read buffer : hello
Peeked and found: this
Re-read buffer : this
Peeked and found: is
Re-read buffer : is
Peeked and found: peeking
Re-read buffer : peeking
Peeked and found: .done
Leaving child process
^C
cs:~/lecture/OSII/socket/strea
m/msg_peek$
```

```
client
sizzle:~$ cd
lecture/OSII/socket/stream/msg_pe
ek/
sizzle:~/lecture/OSII/socket/stre
am/msg_peek$ client cs
>hello
Sending hello
>this
Sending this
>is
Sending is
>peeking
Sending peeking
>.done
Sending .done
sizzle:~/lecture/OSII/socket/stre
am/msg_peek$
```

Out of Band Message

- 경우에 따라서는 급한 메세지를 보내고 싶을 때가 있다.

- 이런 경우에 MSG_OOB flag를 사용하여 급한 메세지를 보낸다.

- 현재는 stream-based socket에서만 out of band messaging을 지원한다.

Server.c(1/2)

```
Modified server : ~sthwang/lecture/OSII/socket/stream/msg_oob/server.c
/*
 * Program 10.16: Server - Internet domain, connection-oriented -
 */
         /* same as Program 10.6 */

if (fork()==0) {                           /* In CHILD process       */
      int urg=0, mark=0; /* flag reception of OOB message, and       *
                           /* note its localtion in the stream ...   */
      do {
        sleep(3);
        if ((len=recv(new_sock, buf, BUFSIZ, MSG_OOB)) > 0) {
          write(fileno(stdout), "URGENT msg pending\n", 19);
          urg=1;
        }
        if (urg) ioctl(new_sock, SIOCATMARK, &mark);
        if (mark) {
           write(fileno(stdout),"<-- the URGENT msg\n",20);
           mark=urg=0;
          }
        if ((len=recv(new_sock,buf,BUFSIZ,0)) > 0) {
          if (!strncmp(buf,".done",len-1)) break;
             write(fileno(stdout),buf,len);
          }
        }
```

The `ioctl` call will assign the variable `mark` a positive value if the next I/O call will process data that is beyond the urgent data; otherwise, it will assign `mark` a 0 value.

If the server is beyond the processing of the urgent message data the string
"← the URGENT msg" is appended to the data currently displayed, and `mark` and `urg` variables are cleared by resetting them to 0.

Server.c(2/2)

```
while(1);
      write(fileno(stdout),"Leaving child process\n",23);
      close(new_sock);                             /* In CHILD process
         */
      exit(0);
      }
    else close(new_sock);                  /* In PARENT process      */
    } while(1);                                    /* FOREVER        */
  }
```

Client.c

```
Modified client : ~sthwang/lecture/OSII/socket/stream/msg_oob/client.c
/*
 * Program 10.17: Client - Internet domain, connection-oriented -
MSG_OOB
 */
         /* same as Program 10.7 */
do {
    write(fileno(stdout),">",3);                          /* Prompt user    */
    if ((len=read(fileno(stdin),buf,BUFSIZ)) > 0) { /* Get input      */
      if (buf[0]=='!') {
        write(fileno(stdout),"URGENT msg sent\n",16);
        send(orig_sock,buf,len,MSG_OOB);
        }
   else send(orig_sock,buf,len,0);
   }
    } while (strncmp(buf,".done",len-1));
  close(orig_sock);
  exit(0);
  }
```

Urgent message

Normal message

실행결과

```
server
cs:~/lecture/OSII/socket/stream/msg_oob$
server
a
b
URGENT msg pending
c
d
e
!help<-- the URGENT msg
f
g
Leaving child process
^C
cs:~/lecture/OSII/socket/stream/msg_oob$
```

```
client
sizzle:~/lecture/OSII/socket/s
tream/msg_oob$ client cs
>a
>b
>c
>d
>e
>!help
URGENT msg sent
>f
>g
>.done
sizzle:~/lecture/OSII/socket/s
tream/msg_oob$
```