Socket (2)

Suntae Hwang
Kookmin University

The connectionless paradigm

- connection-oriented socket과 마찬가지로 server와 client 모두 socket을 만든다.
- 하지만 connection-oriented와는 달리 양쪽 모두에서 binding한다.
- server는 recvfrom을 통해서 client의 address도 얻는다

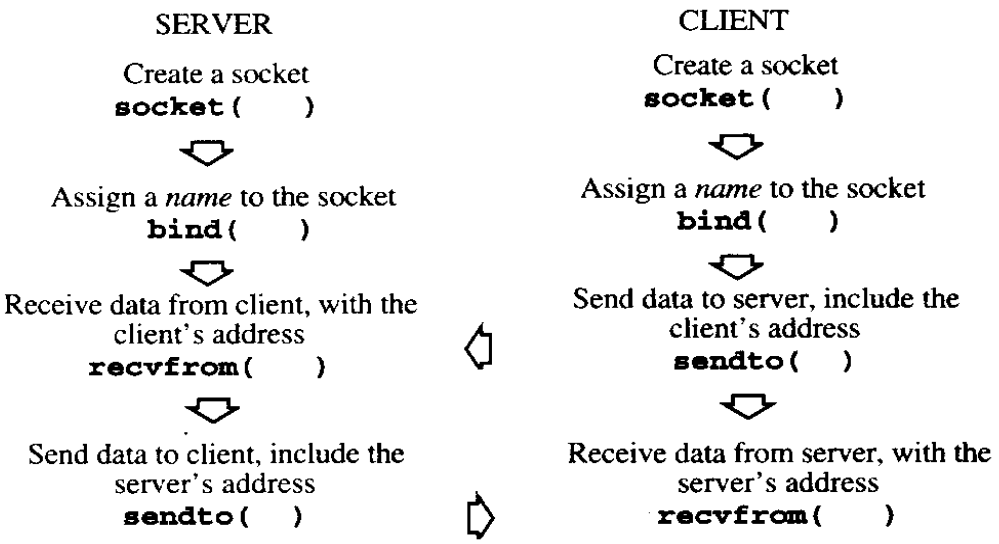


Fig. 10.14 A connectionless, client-server communication sequence.

recv, recvfrom, recvmsg (1/2)

NAME

recv, recvfrom, recvmsg - receive a message from a socket

SYNOPSIS

```
cc [ flag ... ] file ...  -lsocket -lnsl [ library ... ]
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/uio.h>
int recv(int s, char *buf, int len, int flags);
int recvfrom(int s, char *buf, int len, int flags,
    struct sockaddr *from, int *fromlen);
int recvmsg(int s, struct msghdr *msg, int flags);
```

MT-LEVEL

Safe

DESCRIPTION

recv(), recvfrom(), and recvmsg() are used to receive messages from another socket. recv() may be used only on a connected socket (see connect(3N)), while recvfrom() and recvmsg() may be used to receive data on a socket whether it is in a connected state or not. s is a socket created with socket(3N).

recv, recvfrom, recvmsg (2/2)

If from is not a NULL pointer, the source address of the message is filled in. fromlen is a value-result parameter, initialized to the size of the buffer associated with from, and modified on return to indicate the actual size of the address stored there. The length of the message is returned. If a message is too long to fit in the supplied buffer, excess bytes may be discarded depending on the type of socket the message is received from (see socket(3N)).
If no messages are available at the socket, the receive call waits for a message to arrive, unless the socket is nonblocking (see fcntl(2)) in which case -1 is returned with the external variable errno set to EWOULDBLOCK.
The select() call may be used to determine when more data arrives.

send, sendto, sendmsg (1/2)

NAME

send, sendto, sendmsg - send a message from a socket

SYNOPSIS

```
cc [ flag ... ] file ...  -lsocket -lnsl [ library ... ]
#include <sys/types.h>
#include <sys/socket.h>
int send(int s, const char *msg, int len, int flags);
int sendto(int s, const char *msg, int len, int flags,
    const struct sockaddr *to, int tolen);
int sendmsg(int s, const struct msghdr *msg, int flags);
```

MT-LEVEL

Safe

DESCRIPTION

send(), sendto(), and sendmsg() are used to transmit a message to another transport end-point. send() may be used only when the socket is in a connected state, while sendto() and sendmsg() may be used at any time. s is a socket created with socket(3N). The address of the target is given by to with tolen specifying its size. The length of the message is given by len.

If the message is too long to pass atomically through the underlying protocol, then the error EMSGSIZE is returned, and the message is not transmitted.

A return value of -1 indicates locally detected errors only.

send, sendto, sendmsg (2/2)

It does not implicitly mean the message was not delivered.

If the socket does not have enough buffer space available to hold the message being sent, send() blocks, unless the socket has been placed in non-blocking I/O mode (see fcntl(2)). The select(3C) or poll(2) call may be used to determine when it is possible to send more data.

A UNIX domain datagram socket example

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

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

Server.c (1/2) - UNIX domain, connectionless

```
#include "local.h"
#include "cleanup.h"

main(void) {
  int   orig_sock,      /* Original socket descriptor in server */
        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   */

  if ((orig_sock = socket(AF_UNIX, SOCK_DGRAM, 0)) < 0) { /* SOCKET
*/
    perror("generate error");
    exit(1);
    }
  serv_adr.sun_family = AF_UNIX; /* Set tag appropriately   */
  strcpy(serv_adr.sun_path,SERVER_FILE);/* Assign name      */
  unlink(SERVER_FILE);                  /* Remove leftovers
```

Server.c (2/2) - UNIX domain, connectionless

```
serv_adr.sun_family = AF_UNIX;      /* Set tag appropriately */
strcpy(serv_adr.sun_path,SERVER_FILE);/* Assign name         */
unlink(SERVER_FILE);                /* Remove leftovers     */

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,SERVER_FILE);
  exit(2);
  }

for (i=1; i<=10; i++) {
  recvfrom(orig_sock, buf, sizeof(buf), 0, /* RECEIVE it  */
           (struct sockaddr *)&clnt_adr, &clnt_len);

  printf("s-> %s", buf);
}
clean_up(orig_sock,SERVER_FILE);
exit(0);
}
```

client의 address를 얻는다. 하지만 사용되는 않았다.

Client.c(1/2) - UNIX domain, connectionless

```
#include "local.h"
#include "cleanup.h"

main(void) {
  int   orig_sock,     /* Original socket descriptor in client */
        i;             /* Loop counter                         */
  static struct sockaddr_un
               clnt_adr,
               serv_adr;  /* UNIX address of the server process */
  static char  buf[10],   /* Buffer for messages                */
               client_file[15];

  serv_adr.sun_family = AF_UNIX;      /* Set tag appropriately */
  strcpy(serv_adr.sun_path,SERVER_FILE);  /* Assign name  */
if ((orig_sock = socket(AF_UNIX, SOCK_DGRAM, 0)) < 0) { /* SOCKET */
    perror("generate error");
    exit(1);
    }
  sprintf(client_file,"%07d_socket",getpid());
  clnt_adr.sun_family = AF_UNIX;
  strcpy(clnt_adr.sun_path, client_file);
```

Client.c(1/2) - UNIX domain, connectionless

```
if (bind(orig_sock, (struct sockaddr *)&clnt_adr,    /* BIND */
        sizeof(clnt_adr.sun_family)+strlen(clnt_adr.sun_path)) < 0)
{
    perror("bind error");
    exit(2);
    }
  for (i=1; i<=10; i++) {
    sleep(1);                    /* Slow down the client       */
    sprintf(buf, "c: %d\n", i);       /* Create message        */
    sendto(orig_sock, buf, sizeof(buf), 0,   /* Send it        */
        (struct sockaddr *)&serv_adr, sizeof(struct sockaddr));
    }
  clean_up(orig_sock,client_file);
  exit(0);
  }
```

실행결과(1/2)

```
sizzle:~/lecture/OSII/socket/datagram/UNIX$ server&
[1] 5059
sizzle:~/lecture/OSII/socket/datagram/UNIX$ client sizzle
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
[1]+  Done                    server
sizzle:~/lecture/OSII/socket/datagram/UNIX$
```

실행결과(2/2)

```
cs:~/lecture/OSII/socket/datagram/UNIX$ server&
[1] 13811
cs:~/lecture/OSII/socket/datagram/UNIX$ client& client& ls -l *socket
[2] 13812
[3] 13813
p---------   1 sthwang  faculty        0 Nov  3 16:57 0013812_socket
p---------   1 sthwang  faculty        0 Nov  3 16:57 0013813_socket
p---------   1 sthwang  faculty        0 Nov  3 16:57 server_socket
cs:~/lecture/OSII/socket/datagram/UNIX$ s-> c: 1
s-> c: 1
s-> c: 2
s-> c: 2
s-> c: 3
s-> c: 3
s-> c: 4
s-> c: 4
s-> c: 5
s-> c: 5
[1]   Done                    server
[2]-  Done                    client
[3]+  Done                    client
cs:~/lecture/OSII/socket/datagram/UNIX$
```

An internet domain datagram socket example

```
"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>
static char    buf[BUFSIZ];
```

Server.c(1/3) - Internet domain, connectionless

```
#include "local.h"

main(void) {
  int   sock, n,
        server_len, client_len;
  struct sockaddr_in server,  /* Address structures  */
                     client;  /* create the SOCKET   */

  if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0) { /* SOCKET */
    perror("SERVER socket ");
    exit(1);
    }                                    /* Set svr adr info    */
  server.sin_family       = AF_INET;  /* Address family      */
  server.sin_addr.s_addr   = htonl(INADDR_ANY);
                                       /* use any adr         */
  server.sin_port         = htons(0);
                                        /* pick a free port   */
 if (bind(sock, (struct sockaddr *)&server,  /* BIND the socket
        */
        sizeof(server)) < 0) {
```

Direct the system to select a port

Server.c(2/3) - Internet domain, connectionless

```
    perror("SERVER bind");
    exit(2);
    }
  if (getsockname(sock, (struct sockaddr *)&server,
        &server_len) < 0) {
    perror("SERVER getsocketname ");
    exit(3);
    }
  printf("Server using port %d\n",ntohs(server.sin_port));
/*display port # */
  while(1) {
    client_len = sizeof(client);      /* estimate length     */
    memset(buf, 0, BUFSIZ);           /* clear the buffer    */
    if ((n = recvfrom(sock, buf, BUFSIZ, 0,
         /* the client message          */
              (struct sockaddr *)&client, &client_len)) < 0) {
        perror("SERVER recvfrom ");        close(sock);
        exit(4);
      }
```

Issued to determine which port the system selected

Server process는 information이 도달할 때 까지 blocked될 것이다.

Server.c(3/3) - Internet domain, connectionless

```
    write(fileno(stdout), buf, n); /* show msg to server    */
    memset(buf, 0, BUFSIZ);          /* clear the buffer    */
    if (fgets(buf, BUFSIZ, stdin) != NULL) { /* server's msg
                */
      if ((sendto(sock, buf, strlen(buf), 0, /* server's msg
                */
                (struct sockaddr *)&client, client_len)) < 0) {
        perror("SERVER sendto ");
        close(sock);
        exit(5);
    }
  }
 }
}
```

Client.c(1/3)-Internet domain, connectionless socket

```
 #include "local.h"

main(int argc, char *argv[]) {
  int           sock, n,
                server_len;
  static struct sockaddr_in   /* Address structures  */
                server, client;
  struct hostent       *host;  /* For host info       */

  if (argc < 3) {              /* need server & port  */
    fprintf(stderr,"usage: %s server port_#\n", argv[0]);
    exit(1);
  }
  if (!(host=gethostbyname(argv[1]))) {/* get server info   */
    perror("CLIENT gethostbyname ");
    exit(2);
  }
```

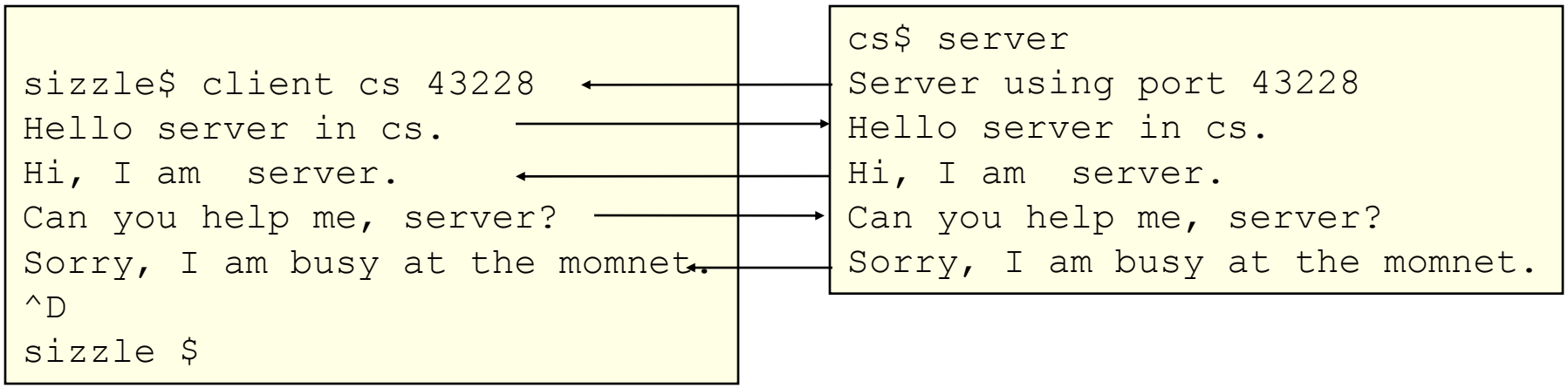
Client.c(2/3)-Internet domain, connectionless socket

```
server.sin_family = AF_INET;          /* address family      */
memcpy(&server.sin_addr, host->h_addr, host->h_length);
server.sin_port = htons(atoi(argv[2]));/* @ passed port #   */

if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
                                        /* create a SOCKET */
   perror("CLIENT socket ");
   exit(3);
}
client.sin_family = AF_INET;          /* address family      */
client.sin_addr.s_addr = htonl(INADDR_ANY); /* use any adr */
client.sin_port       = htons(0);     /* pick free port      */
 if (bind(sock, (struct sockaddr *)&client,
       sizeof(client)) < 0) {
   perror("CLIENT bind ");
   exit(4);
 }
 while (fgets(buf, BUFSIZ, stdin) != NULL) {/* get clnt's msg*/
   server_len = sizeof(server);       /* guess at length     */
   if (sendto(sock, buf, strlen(buf), 0,/* send msg to srvr */
       (struct sockaddr *)&server, server_len) < 0) {
```

Client.c(3/3)-Internet domain, connectionless socket

```
    perror("CLIENT sento ");
    close(sock);
    exit(5);
    }

  memset(buf, 0, BUFSIZ);           /* clear the buffer    */
  if ((n=recvfrom(sock, buf, BUFSIZ, 0,* server's message */
      (struct sockaddr *)&server, &server_len))<0) {
    perror("CLIENT recvfrom ");
    close(sock);
    exit(6);
    }
  write(fileno(stdout),buf,n);      /* show msg to clnt    */
  memset(buf,0,BUFSIZ);             /* clear the buffer    */
  }
close(sock);
exit(0);
}
```

실행결과 - Internet domain, connectionless socket



Getsockname

NAME
    getsockname - get socket name

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

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

    int getsockname(int s, struct sockaddr *name, int *namelen);

MT-LEVEL
    Safe

DESCRIPTION
    getsockname() returns the current name for socket s. The namelen parameter should be initialized to indicate the amount of space pointed to by name. On return it contains the actual size in bytes of the name returned.

Non-blocking

- It is clear from the last example that when processes communicate, they need a way to coordinate their activities other than blocking (waiting) for the recipient process to respond.

- One approach would be to change the socket from its default of blocking to non-blocking.

Server.c(1/4) -Internet domain, connectionless - non-blocking

```
#include "local.h"
#include <sys/filio.h>
#include <errno.h>
extern errno;

main(void) {
  int   sock, n,
        server_len, client_len;
  int   errcount=0, flag=1;
  struct sockaddr_in server,  /* Address structures  */
                     client;  /* create the SOCKET   */

  if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0) { /* SOCKET */
    perror("SERVER socket ");
    exit(1);
  }                                /* Set svr adr info   */
  if (ioctl(sock, FIONBIO, &flag) < 0) {
    perror("Server ioctl ");
    exit(2);
  }
```

Server.c(2/3) -Internet domain, connectionless - non-blocking

```
  server.sin_family       = AF_INET;  /* Address family      */
  server.sin_addr.s_addr   = htonl(INADDR_ANY);/* use any adr*/
  server.sin_port         = htons(0); /* pick a free port    */

  if (bind(sock, (struct sockaddr *)&server, /* BIND the socket
       */
       sizeof(server)) < 0) {
    perror("SERVER bind");
    exit(3);
  }
  if (getsockname(sock, (struct sockaddr *)&server,
       &server_len) < 0) {
    perror("SERVER getsocketname ");
    exit(4);
  }
printf("Server using port %d\n",ntohs(server.sin_port));
                                        /*display port # */
 while(1) {
    client_len = sizeof(client);      /* estimate length     */
    memset(buf, 0, BUFSIZ);           /* clear the buffer    */
```

Server.c(3/3) -Internet domain, connectionless - non-blocking

```
if ((n = recvfrom(sock, buf, BUFSIZ, 0,/* the clnt message */
              (struct sockaddr *)&client, &client_len)) < 0) {
    if (errcount++>3 || errno!=EWOULDBLOCK) {
      perror("SERVER recvfrom ");
      close(sock);
       exit(5);
    }
       sleep(errcount*2);
       continue;
    }
  errcount=0;
  write(fileno(stdout), buf, n);    /* show msg to server  */
  memset(buf, 0, BUFSIZ);           /* clear the buffer    */
  if (fgets(buf, BUFSIZ, stdin) != NULL) { /* server's msg*/
    if ((sendto(sock, buf, strlen(buf), 0, /* server's msg*/
              (struct sockaddr *)&client, client_len)) < 0) {
      perror("SERVER sendto ");
      close(sock);
      exit(6);
     }
  }
 }
}
```

socket 이 non-blocking 일 때 recvfrom call은 아무런 message가 없으면 즉각 return한다.

Using select library call(1/4)

NAME
    select - synchronous I/O multiplexing

SYNOPSIS
    #include <sys/time.h>
    #include <sys/types.h>

    int select(int nfds, fd_set *readfds, fd_set *writefds,
        fd_set *exceptfds, struct timeval *timeout);

    void FD_SET(int fd, fd_set &fdset);

    void FD_CLR(int fd, fd_set &fdset);

    int FD_ISSET(int fd, fd_set &fdset);

    void FD_ZERO(fd_set &fdset);

MT-LEVEL
    MT-Safe

Using select library call(2/4)

DESCRIPTION

select() examines the I/O file descriptor sets whose addresses are passed in readfds, writefds, and exceptfds to see if any of their file descriptors are ready for reading, are ready for writing, or have an exceptional condition pending, respectively. Out-of-band data is the only exceptional condition. nfds is the number of bits to be checked in each bit mask that represents a file descriptor; the file descriptors from 0 to nfds -1 in the file descriptor sets are examined. On return, select() replaces the given file descriptor sets with subsets consisting of those file descriptors that are ready for the requested operation. The return value from the call to select() is the number of ready file descriptors.

The file descriptor sets are stored as bit fields in arrays of integers. The following macros are provided for manipulating such file descriptor sets: FD_ZERO() initializes a file descriptor set fdset to the null set. FD_SET() includes a particular file descriptor fd in fdset. FD_CLR() removes fd from fdset. FD_ISSET() is nonzero if fd is a member of fdset, zero otherwise. The behavior of these macros is undefined if a file descriptor value is less than zero or greater than or equal to FD_SETSIZE. FD_SETSIZE is a constant defined in <sys/select.h>.

Using select library call(3/4)

If timeout is not a NULL pointer, it specifies a maximum interval to wait for the selection to complete. If timeout is a NULL pointer, the select() blocks indefinitely. To effect a poll, the timeout argument should be a non-NULL pointer, pointing to a zero-valued timeval structure.

Any of readfds, writefds, and exceptfds may be given as NULL pointers if no file descriptors are of interest.

RETURN VALUES

select() returns the number of ready file descriptors contained in the file descriptor sets or - 1 if an error occurred. If the time limit expires, then select() returns 0.

ERRORS

The call fails if:

EBADF One of the I/O file descriptor sets specified an invalid I/O file descriptor.

EINTR A signal was delivered before any of the selected events occurred, or the time limit expired.

Using select library call(4/4)

EINVAL A component of the pointed-to time limit is outside the acceptable range: t_sec must be between 0 and 10^8, inclusive. t_usec must be greater than or equal to 0, and less than 10^6.

SEE ALSO

poll(2), read(2), write(2)

NOTES

The default value for FD_SETSIZE (currently 1024) is larger than the default limit on the number of open files. In order to accommodate programs that may use a larger number of open files with select(), it is possible to increase this size within a program by providing a larger definition of FD_SETSIZE before the inclusion of <sys/types.h>.

The file descriptor sets are always modified on return, even if the call returns as the result of a timeout.

Server.c(1/4) - Internet domain, connectionless - using select

```
#include "local.h"
#include <sys/time.h>

main(void) {
  int   sock, n,
        server_len, client_len;
  int   n_ready, need_rsp;
  struct sockaddr_in server,  /* Address structures  */
                     client;  /* create the SOCKET   */
  fd_set                read_fd;
  struct timeval        time_2_wait;

  if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0) { /* SOCKET */
    perror("SERVER socket ");
    exit(1);
    }                                   /* Set svr adr info    */
  server.sin_family       = AF_INET;          /* Address family*/
  server.sin_addr.s_addr   = htonl(INADDR_ANY);/* use any adr*/
  server.sin_port         = htons(0); /* pick a free port    */
```

Server.c(2/4) - Internet domain, connectionless - using select

```
if (bind(sock, (struct sockaddr *)&server,   /* BIND the socket
        */
        sizeof(server)) < 0) {
    perror("SERVER bind");
    exit(2);
    }
  if (getsockname(sock, (struct sockaddr *)&server,
        &server_len) < 0) {
    perror("SERVER getsocketname ");
    exit(3);
    }
  printf("Server using port %d\n",ntohs(server.sin_port));
                                        /*display port # */
  time_2_wait.tv_sec=5;
```

Server.c(3/4) - Internet domain, connectionless - using select

```
while(1) {
   FD_ZERO(&read_fd);                      /* zero all bits       */
   FD_SET(sock,&read_fd);                  /* the one to read     */
   if ((n_ready=select(sock+1,&read_fd, (fd_set *)NULL,
                       (fd_set *)NULL, &time_2_wait)) < 0) {
    perror("SERVER read socket select ");
    continue;
    }
   if (FD_ISSET(sock,&read_fd)) {
    client_len = sizeof(client);           /* estimate length     */
    memset(buf, 0, BUFSIZ);                /* clear the buffer    */
    if ((n = recvfrom(sock, buf, BUFSIZ, 0,
                                           /* the clnt message    */
                (struct sockaddr *)&client, &client_len)) < 0) {
      perror("SERVER recvfrom ");
      close(sock);
      exit(4);
    }
    write(fileno(stdout), buf, n);         /* show msg to server  */
    memset(buf, 0, BUFSIZ);                /* clear the buffer    */
    need_rsp=1;
   }
```

Server.c(4/4) - Internet domain, connectionless - using select

```
if (need_rsp) {
   FD_ZERO(&read_fd);                    /* zero all bits       */
   FD_SET(fileno(stdin),&read_fd);       /* the one to read     */
   if ((n_ready=select(fileno(stdin)+1,&read_fd,
       (fd_set *)NULL, (fd_set *)NULL, &time_2_wait)) < 0) {
    perror("SERVER read stdin select ");
    continue;
    }
    if (FD_ISSET(fileno(stdin),&read_fd)) {
     if (fgets(buf, BUFSIZ, stdin) != NULL) {/* server's msg*/
      if ((sendto(sock, buf, strlen(buf), 0, /* server's msg*/
                (struct sockaddr *)&client, client_len)) < 0) {
        perror("SERVER sendto ");
        close(sock);
        exit(5);
       }
      need_rsp=0;
     }
    }
   }
  }
}
```