Unix Network Programming

Chapter 8

Elementary UDP Sockets

Typical scenario between UDP client/server

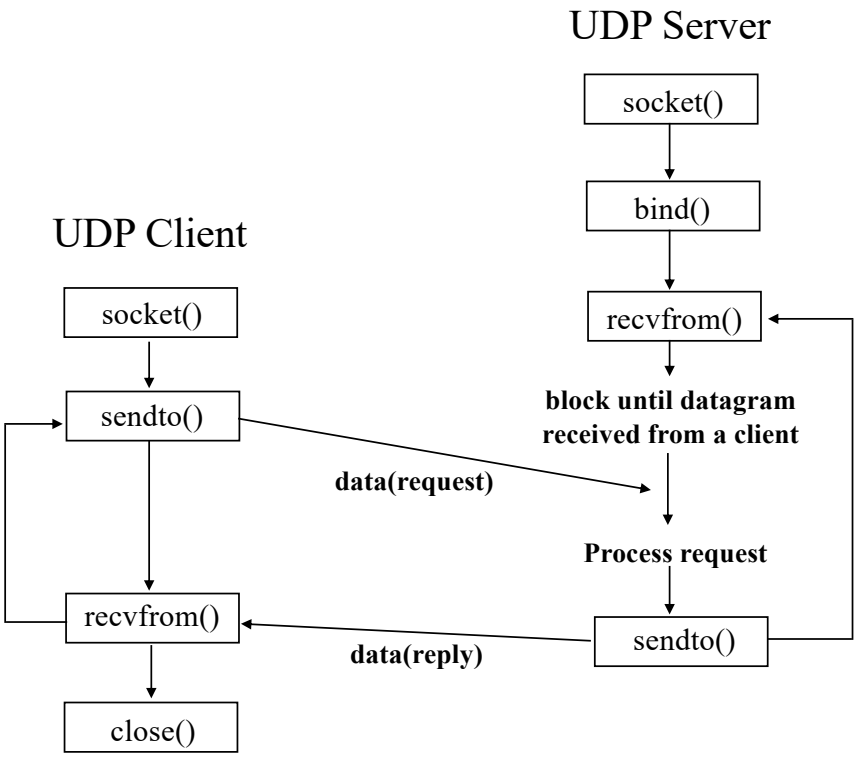
Typical UDP client

- Client does not establish a connection with the server
- Client sends a datagram to the server using **sendto()** function

Typical UDP server

- Does not not accept a connection from a client
- Server calls **recvfrom()** function which waits until data arrives from some client

Socket functions for UDP client/server



recvfrom() and sendto()

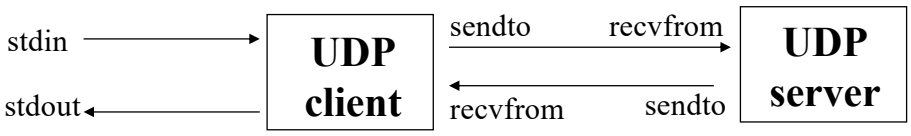
```
#include<sys/socket.h>

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);
```

- Both return the amount of user data in the datagram received or sent if no error; -1 on error
- Reading a datagram of length 0 is acceptable, i.e., returned by recvfrom()

Simple UDP echo client/server example



Simple echo client-server using UDP.

UDP echo server: main()

```
// Fig. 8.3, 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_fammily = 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));
}
```

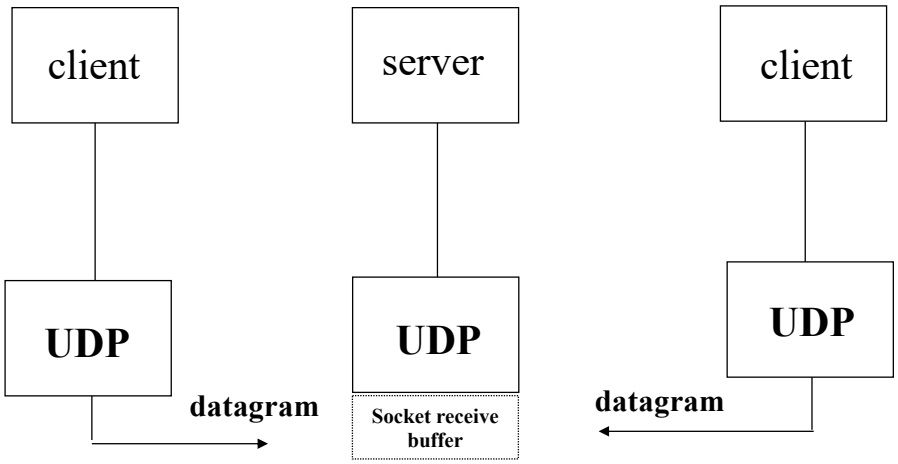
UDP echo server: dg_echo()

```
// Fig. 8.4, 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 ( ; ; ) {  // iterative UDP server
       len=clilen;
       // read from queuing in UDP layer
       n=Recvfrom(sockfd, mseg, MAXLINE, 0, pcliaddr, &len);

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

UDP client/server with two clients



Summary of UDP client-server with two clients.

Shared receive buffer for all datagrams received

UDP echo client: dg_cli()

```
// Fig 8.8, 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)  {
        Sendto(sockfd, sendline, strlen(sendline), 0, pservaddr, servlen);

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

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

Was an ephemeral port assigned to UDP Socket? No bind()?

Lost datagrams

- Loss can be in either reverse and forward direction
  - If the client datagram arrives at the server but the server's reply is lost, the client will block forever in its call to recvfrom()
  - Similarly, if client datagram is lost, client will block forever waiting for a server reply
- The only way to prevent this is to place a timeout on the recvfrom()

Socket Timeout Snippet

```
{
    int sockfd;
    struct timeval tv;
    tv.tv_sec = 3;
    tv.tv_usec = 0;
    setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO, &tv, sizeof(tv));
    // …
    for ( ; ; ) {
        n = recvfrom(sockfd, recvline, MAXLINE, 0, preply_addr, &len);
        if (n < 0) {
            if (errno == EINTR) {
                break;     // waited long enough
            } else if (errno == EWOULDBLOCK) {
                 // timed out, do bookkeeping
            } else {
                    printf("recvfrom error\n");
            }
        } // endif n
    }
}
```

Verifying received response 1/2

```
// Need to return IP address and port of who sent back reply
// Fig. 8.9; udpcliserv/dgcliaddr.c
#include “unp.h”
void dg_cli(FILE *fp, int sock, const SA *pseraddr, socklen_t servlen)
{
    int n;
    char sendline[MAXLINE], recvline[MAXLINE];
    socklen_t len;
    struct sockaddr  *preply_addr;
    preply_addr = Malloc(servlen);

    while (Fgets(sendline, MAXLINE, fp) ! = NULL) {
            Sendto(sockfd,sendline, strlen(sendline), 0, pservaddr, servlen);
            len = servlen;
            n = Recvfrom(sockfdm, recvline, MAXLINE, 0, preply_addr,&len);

            /* continued  */
```

Verifying received response 2/2

```
        /* … continued from previous slide */
        //Need to return IP address and port of who sent back reply
        //source code in udpcliserv/dgcliaddr.c

        if (len != servlen || memcmp(pservaddr, preply_addr, len) != 0) {
            printf("reply from %s  (ignore)\n", Sock_ntop(preply_addr, len));
            continue;
        }
        recvline[n] = 0;    /* NULL terminate */
        Fputs(recvline, stdout);
    }  /* while */
} /* end dg_cli */
```

- Program works if the server is on a host with a single IP address
- Program can fail if the server is multi-homed (Why?)

Multihomed Servers

- If server is multi-homed
  If the server has not bound an IP address to its socket, the kernel chooses the source address for the IP datagram outgoing from the server.

- Can this be solved otherwise?
  Verify respondent's host name by looking up its name from DNS (will see later how to do that).

- Another solution
  - Create a socket for every IP address configured on host
  - Bind IP address to socket
  - Wait for any of these sockets to become readable

Server not running

- Client blocks forever in the call to recvfrom()
- ICMP error "port unreachable" is an asynchronous error
- Error caused by sendto but sendto returns successfully
  - i.e., it only means there was room for resulting IP datagram on interface output queue
- ICMP error returned later → asynchronous error

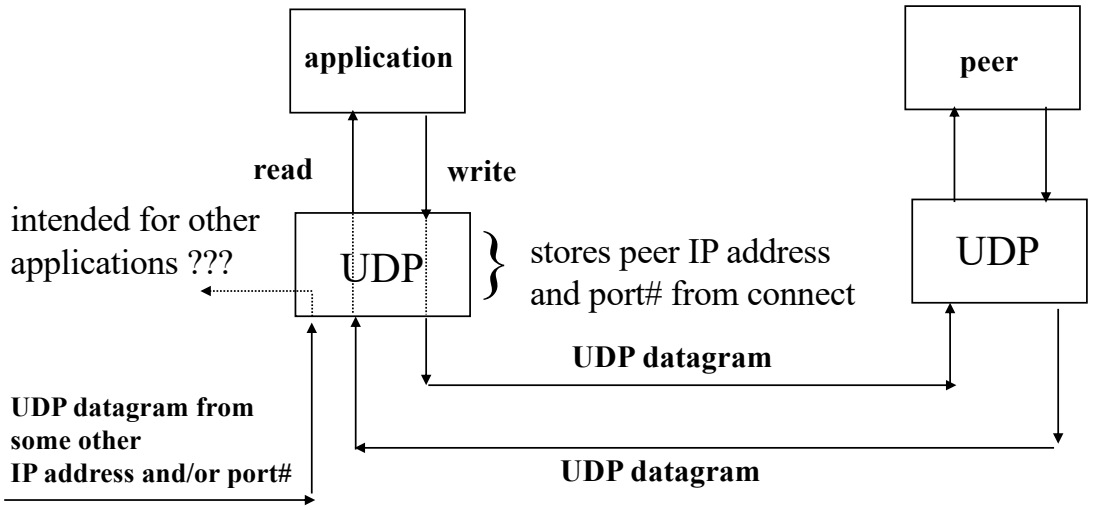
The basic rule is that asynchronous errors are not returned for UDP sockets unless the socket has been connected.

But … Linux returns most ICMP "destination unreachable" errors even for unconnected sockets, as long as the SO_BSDCOMPAT socket option is not enabled.

connect() with UDP

- UDP connect() is nothing like a TCP connect()
  - There is no three-way handshake
  - Instead, the kernel records the IP address and port number of the peer and returns immediately to calling process
  - Communicate with exactly one peer
- With a connected UDP socket, three things change:
  1. Can no longer specify the destination IP address and port for an output operation. That is, we do not use sendto() but use write() or send() instead. (Can use sendto() but fifth argument is null, and sixth is zero.)
  2. Do not use recvfrom() but use read() or recv() instead. Only datagrams returned by the kernel for an input operation on a connected UDP socket are those arriving from protocol address specified in connect().
  3. Asynchronous errors are returned to the process for a connected UDP socket.

connect() with UDP



UDP connect() session

UDP echo client: dg_cli revisited

```
// Fig. 8.17, udpcliserv/dgcliconnect.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);  // read error: Connection refused
        recvline[n] = 0;    /* null terminate */
        Fputs(recvline, stdout);
    }
} // ICMP error received after attempting to send the first datagram to a down server
```