Network Programming: Ch.8: Elementary UDP Sockets

Li-Hsing Yen
NYCU
Ver. 1.0.0

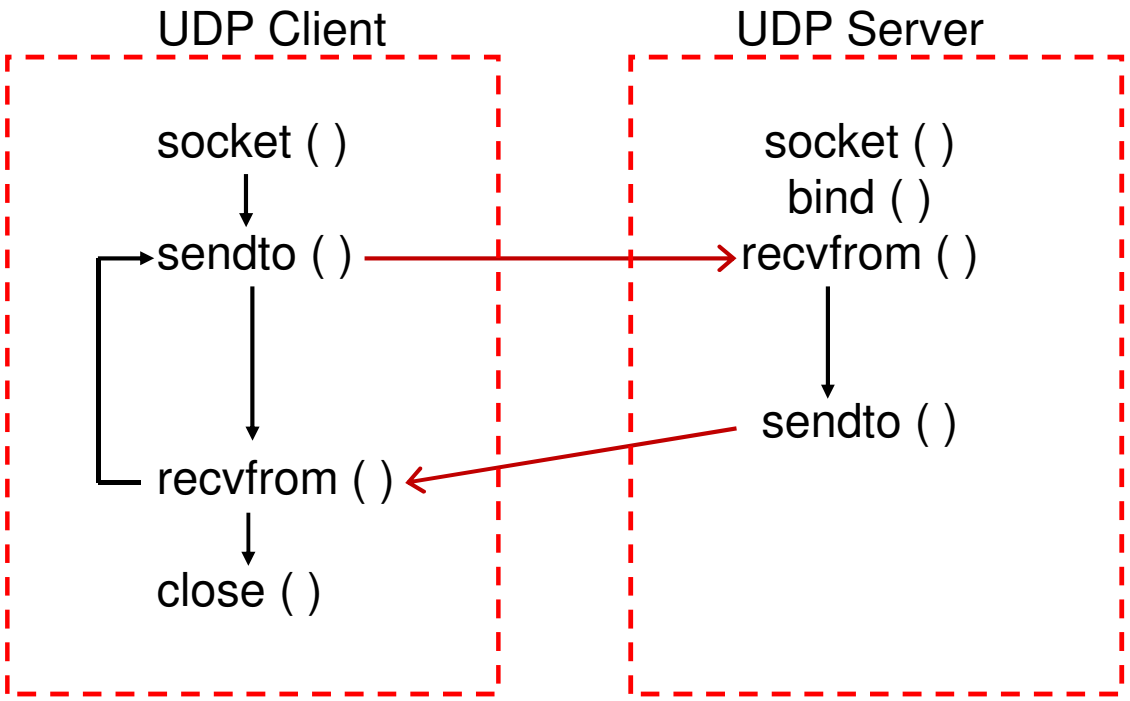
Elementary UDP Sockets
connectionless, unreliable, datagram

- `recvfrom` and `sendto` functions
- UDP echo server
- UDP echo client
- Verify received responses
- *connect* function with UDP
- Rewrite *dg_cli* function with *connect*
- Lack of flow control with UDP
- Determine outgoing interface with UDP
- TCP and UDP echo server using *select*

stream

datagram

Socket Functions for UDP Client-Server



Function recvfrom

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

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

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

同 read

Ch. 14

socket address structure of the source

value-result argument

因為用UDP接收資料不需事先建立connection，所以收到的DUP資料可能來自於任何一部機器。如果要知道傳送此資料的傳送者位址資訊就需要後兩個參數。

Functions sendto

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

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

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

同 write

Ch. 14

socket address structure of the destination

value argument

因為用UDP傳送資料不需事先建立connection，所以每一筆傳送的DUP資料都要個別指定接收者的位址資訊。

UDP Echo Server: main Function

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));
}
```

UDP socket

next page

UDP Echo Server: dg_echo Function

lib/dg_echo.c

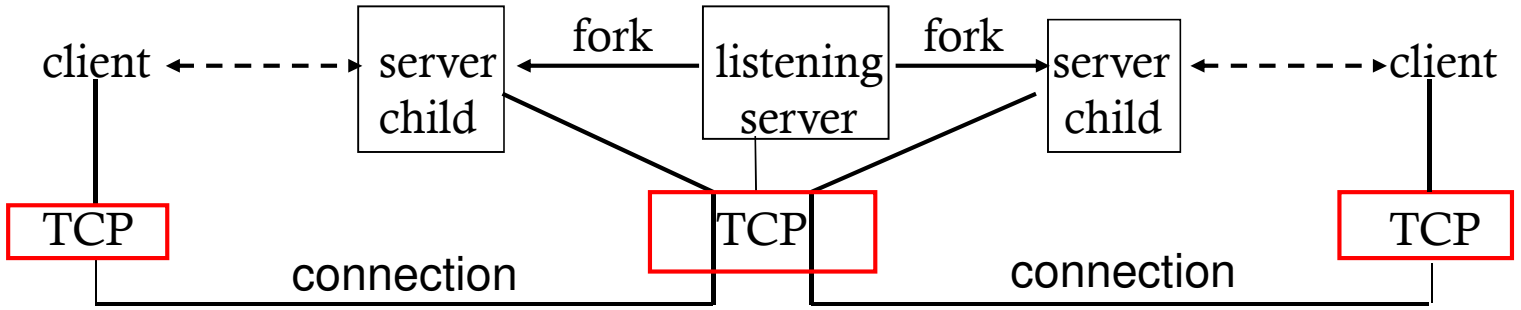
```
#include       "unp.h"
void
dg_echo(int sockfd, SA *pcliaddr, socklen_t clilen)
{
        int             n;
        socklen_t   len;
        char            mesg[MAXLINE];

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

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

所以再次呼叫時要重設

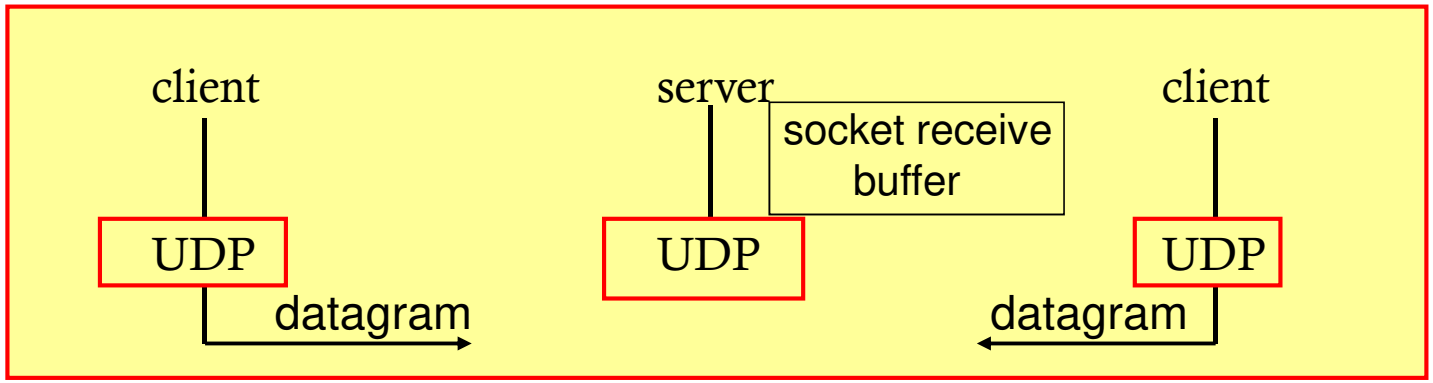
return後可能會改變值

用新值呼叫sento

Comparing TCP and UDP with Two Clients



TCP client-server with two clients



UDP client-server with two clients

Most TCP servers are concurrent and most UDP servers are iterative.

UDP Echo Client: main Function

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>");
        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);
}
```

UDP socket

UDP Echo Client: dg_cli Function

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);
       }
}
```

字串結尾的'\0'未送出

自行加0

source address is of no interest

應該是server回應的吧

Problems with UDP Sockets

1. Lost Datagrams
   - `recvfrom` blocks forever
2. Malicious (惡意的) Datagrams
   - Receive malicious datagrams
3. Server Not Running
   - `recvfrom` blocks forever
4. Lack of Flow Control

Problem 1: Lost Datagrams

- In case of lost datagrams (client request or server reply):
  - `recvfrom` in dg_cli blocks forever

```
Sendto(sockfd, sendline, strlen(sendline), 0, pservaddr, servlen);
n = Recvfrom(sockfd, recvline, MAXLINE, 0, NULL, NULL);
```

  - can place a timeout on `recvfrom`, but still don't know whether request or reply gets lost

Problem 2: Malicious (惡意的) Datagrams

- Malicious datagrams inter-mixed with server replies:
  - should ignore any received datagrams not from that server (要檢查source address)
  - by allocating another socket address structure to store address returned by `recvfrom` and comparing it with that specified in `sendto` (Figure 8.9)

Problem of Previous Method in Multi-homed Server

- For multi-homed server, verifying address may not work (server reply may go through another outgoing interface)
- solution 1: client verifies server domain name instead (Ch. 11)
- solution 2: multi-homed UDP server creates one socket for each interface (IP addr), bind IP addresses to sockets, use `select` across all sockets

Problem 3: Server Not Running

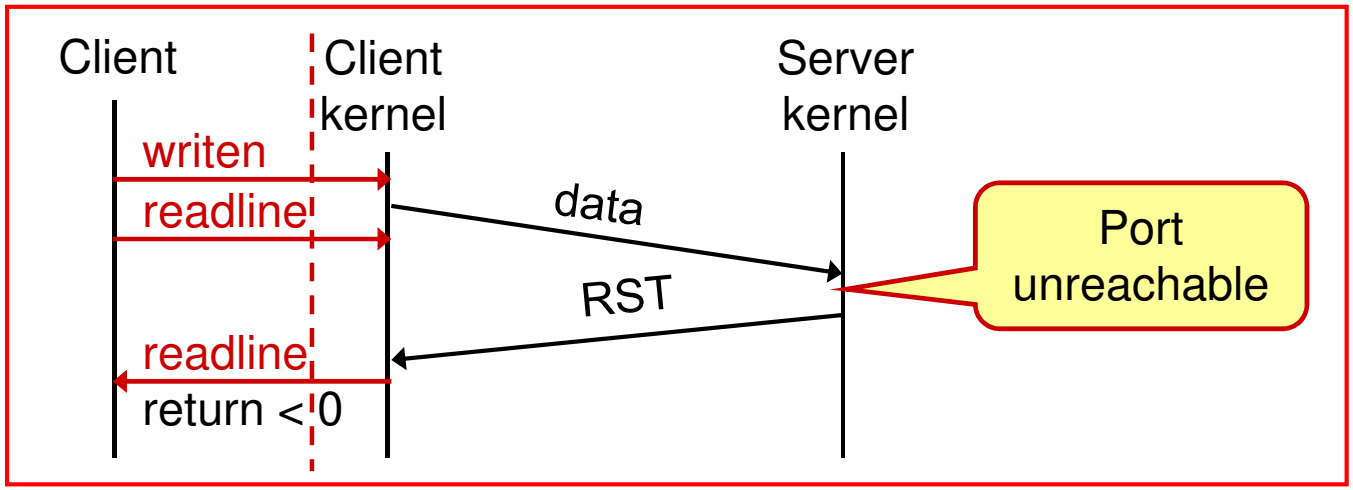
- ICMP port unreachable error (asynchronous error)

非同步錯誤的意思是在sendto成功return後錯誤才發生的

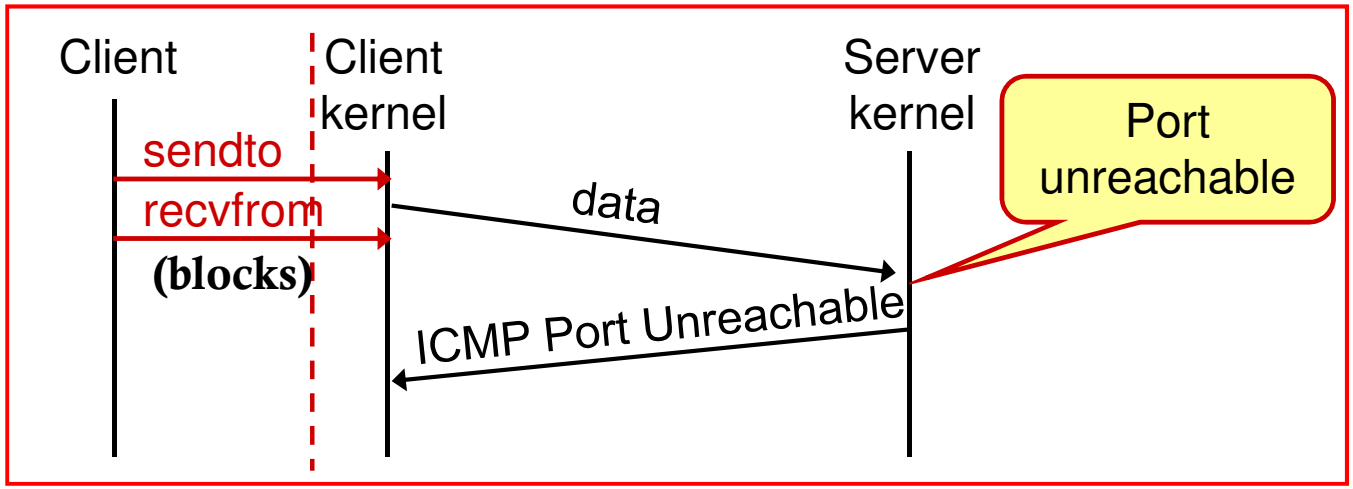
- asynchronous errors are not returned for UDP sockets, unlike TCP socket (reason: considering a client sending 3 datagrams to 3 servers, `recvfrom` has no way to tell which destination of the datagram causes the error)
- `recvfrom` blocks forever
- solution: call `connect` on a UDP socket

Asynchronous Error

In TCP



In UDP



`connect` Function with UDP

- `connect` on UDP socket: no 3-way handshake, kernel only records the connected dest. address (並未真的建立連線)
- For a connected UDP socket (compared to unconnected UDP socket):
  - no longer specify dest IP addr/port for output, use `write`/`send` instead of `sendto`
  - use `read`/`recv` instead of `recvfrom`
  - asynchronous errors are returned to the process for a connected UDP socket (this is what we want)

`connect` Function with UDP (cont.)

- connected UDP socket no longer receives UDP datagrams from other address/port
- So `connect` is used only if the client/server uses the UDP socket to communicate with exactly one peer
- 因此通常用於UDP client 而非server

Calling `connect` Multiple Times

- A process with a connected UDP socket can call `connect` again for that socket
  - to specify a new IP address/port to communicate
  - to unconnect the socket (sin_family or sin6_family = AF_UNSPEC)
- 比較：TCP socket只可被connect一次

Performance Issue

- Call `sendto` for two datagrams on an unconnected UDP socket (temporary connecting):
  - steps in kernel: connect/output/unconnect, connect/output/unconnect (too much overhead)
  - solution: `connect` before `write` multiple datagrams (only one connecting)

dg_cli Function That Calls connect

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

先connect

呼叫write

呼叫read

Problem 4: Lack of Flow Control

- considering `dg_cli` in a client `sendto` 2000 1400-byte datagrams to the server
- client may overrun the server (e.g. 96% loss rate: mostly lost due to receive buffer overflow, some due to network congestion)
- use netstat -s to check the loss
- solution: use SO_RCVBUF option to enlarge buffer; use request-reply model instead of bulk transfer

`connect` to Determine Outgoing Interface with UDP

- Application has no way to know the outgoing interface of an unconnected UDP socket
- Side effect of `connect` on UDP socket:
  - kernel chooses the local IP address by searching routing table
  - process calls `getsockname` to obtain the local IP address and port

Combined TCP and UDP Echo Server Using `select`

- A single server using `select` to multiplex a TCP socket and a UDP socket:
  - create listening TCP socket
  - create UDP socket
  - establish signal handler for `SIGCHLD`
  - prepare a descriptor set for `select`
  - call `select`
  - handle new client connection (TCP)
  - handle arrival of datagram (UDP)

TCP and UDP Echo Server Using Select

udpcliserv/udpservselect.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);

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

TCP socket

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

Setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));
Bind(listenfd, (SA *) &servaddr, sizeof(servaddr));

Listen(listenfd, LISTENQ);

        /* create UDP socket */
udpfd = 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(udpfd, (SA *) &servaddr, sizeof(servaddr));
```

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

FD_ZERO(&rset);
maxfdp1 = max(listenfd, udpfd) + 1;
for ( ; ; ) {
        FD_SET(listenfd, &rset);
        FD_SET(udpfd, &rset);
        if ( (nready = select(maxfdp1, &rset, NULL, NULL, NULL)) < 0) {
                if (errno == EINTR)
                        continue;               /* back to for() */
                else
                        err_sys("select error");
        }
        if (FD_ISSET(listenfd, &rset)) {
                len = sizeof(cliaddr);
                connfd = Accept(listenfd, (SA *) &cliaddr, &len);
```

kill zombie 的 signal handler

準備descriptor set

被 signal handler 中斷過

listening socket is ready for read

呼叫accept不會block

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

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

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

UDP socket is ready for read

Summary

- Lots of features in TCP are lost with UDP:
  - detecting lost packets, retransmitting, verifying responses as being from correct peer, flow control, etc.
- Some reliability can be added
- UDP sockets may generate asynchronous errors reported only to connected sockets
- Use a request-reply model