CpSc 360: Distributed and Network Programming

UDP Sockets Programming

James Wang

Textbook Chapter 8, 14

Create a UDP socket

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

int sock;
sock = socket(PF_INET,
              SOCK_DGRAM,
              0);
if (sock<0) { /* ERROR */ }
```

2

Bind to well known address (typically done by server only)

```
int mysock;
struct sockaddr_in myaddr;

  mysock = socket(PF_INET,SOCK_DGRAM,0);
  myaddr.sin_family = AF_INET;
  myaddr.sin_port = htons( 1234 );
  myaddr.sin_addr.s_addr = htonl( INADDR_ANY );

  bind(mysock, &myaddr, sizeof(myaddr));
```

3

Send UDP Datagrams

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

- `sockfd` **is a UDP socket**
- `buff` **is the address of the data (`nbytes` long)**
- `flags` **(usually 0) are discussed in chapter 14.**
- `to` **is the address of a sockaddr containing the destination address.**
- **Return value is the number of bytes sent, or -1 on error.**

4

`sendto()`

- **You can send 0 bytes of data!**
- **Some possible errors :**
  - **`EBADF`, `ENOTSOCK`**: *bad socket descriptor*
  - **`EFAULT`**: *bad buffer address*
  - **`EMSGSIZE`**: *message too large*
  - **`ENOBUFS`**: *system buffers are full*
- **The return value of sendto() indicates how much data was accepted by the O.S. for sending as a datagram - not how much data made it to the destination.**
- **There is no error condition that indicates that the destination did not get the data!!!**

5

Receive UDP Datagrams

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

- `sockfd` **is a UDP socket**
- `buff` **is the address of a buffer (`nbytes` long)**
- `flags` **(usually 0) are discussed in chapter 14.**
- `from` **is the address of a sockaddr.**
- **Return value is the number of bytes received and put into buff, or -1 on error.**

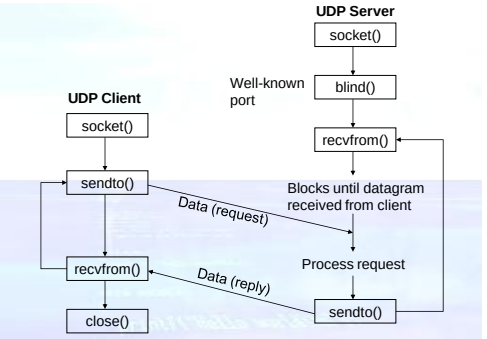
6

recvfrom()

- If buff is not large enough, any extra data is lost forever...
- You can receive 0 bytes of data!
- The `sockaddr` at `from` is filled in with the address of the sender.
- You should set `fromaddrlen` before calling.
- If `from` and `fromaddrlen` are NULL we don't find out who sent the data.
- Same errors as `sendto`, but also:
  - **`EINTR`**: *System call interrupted by signal.*
- Unless you do something special - `recvfrom` doesn't return until there is a datagram available.

7

UDP Client/Server



8

Typical UDP client code

- Create UDP socket.
- Create `sockaddr` with address of server.
- No call to `bind()` is necessary although it is not an error to do so!
- Call `sendto()`, sending request to the server.
- Possibly call `recvfrom()` (if we need a reply).

9

Simple UDP Client Code

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h> /* memset() */
#include <sys/time.h> /* select() */

#define REMOTE_SERVER_PORT 1500
#define MAX_MSG 100

int main(int argc, char *argv[]) {
  int sd, rc, i;
  struct sockaddr_in cliAddr, remoteServAddr;
  struct hostent *h;

  /* check command line args */
  if(argc<3) {
    printf("usage : %s <server> <data1> ... <dataN> \n", argv[0]);
    exit(1);
  }

  /* get server IP address (no check if input is IP address or DNS name */
  h = gethostbyname(argv[1]);
  if(h==NULL) {
    printf("%s: unknown host '%s' \n", argv[0], argv[1]);
    exit(1);
  }
```

10

Simple UDP Client Code (conti.)

```
printf("%s: sending data to '%s' (IP : %s) \n", argv[0], h->h_name,
                inet_ntoa(*(struct in_addr *)h->h_addr_list[0]));
remoteServAddr.sin_family = h->h_addrtype;
memcpy((char *) &remoteServAddr.sin_addr.s_addr, h->h_addr_list[0], h->h_length);
remoteServAddr.sin_port = htons(REMOTE_SERVER_PORT);

/* socket creation */
sd = socket(AF_INET,SOCK_DGRAM,0);
if(sd<0) {
  printf("%s: cannot open socket \n",argv[0]);
  exit(1);
}

/* send data */
for(i=2;i<argc;i++) {
  rc = sendto(sd, argv[i], strlen(argv[i])+1, 0,
              (struct sockaddr *) &remoteServAddr,  sizeof(remoteServAddr));
  if(rc<0) {
    printf("%s: cannot send data %d \n",argv[0],i-1);
    close(sd);
    exit(1);
  }
}
return 1;
}
```

11

Typical UDP Server code

- Create UDP *socket* and *bind* to well known address.
- Call `recvfrom()` to get a request, noting the address of the client.
- Process request and send reply back with `sendto()`.

12

Simple UDP Server Code

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h> /* close() */
#include <string.h> /* memset() */

#define LOCAL_SERVER_PORT 1500
#define MAX_MSG 100

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

  int sd, rc, n, cliLen;
  struct sockaddr_in cliAddr, servAddr;
  char msg[MAX_MSG];

  /* socket creation */
  sd=socket(AF_INET, SOCK_DGRAM, 0);
  if(sd<0) {
    printf("%s: cannot open socket \n",argv[0]);
    exit(1);
  }

  /* bind local server port */
  servAddr.sin_family = AF_INET;
  servAddr.sin_addr.s_addr = htonl(INADDR_ANY);
  servAddr.sin_port = htons(LOCAL_SERVER_PORT);
  rc = bind (sd, (struct sockaddr *) &servAddr,sizeof(servAddr));
  if(rc<0) {
    printf("%s: cannot bind port number %d \n",
                    argv[0], LOCAL_SERVER_PORT);
    exit(1);
  }

  printf("%s: waiting for data on port UDP %u\n",
                    argv[0],LOCAL_SERVER_PORT);

  /* server infinite loop */
  while(1) {

    /* init buffer */
    memset(msg,0x0,MAX_MSG);


    /* receive message */
    cliLen = sizeof(cliAddr);
    n = recvfrom(sd, msg, MAX_MSG, 0,
                    (struct sockaddr *) &cliAddr, &cliLen);

    if(n<0) {
      printf("%s: cannot receive data \n",argv[0]);
      continue;
    }

    /* print received message */
    printf("%s: from %s:UDP%u : %s \n",
                    argv[0],inet_ntoa(cliAddr.sin_addr),
                    ntohs(cliAddr.sin_port),msg);

  }/* end of server infinite loop */

return 0;

}
```


UDP Echo Server

```
int mysock;
struct sockaddr_in myaddr, cliaddr;
char msgbuf[MAXLEN];
socklen_t clilen;
int msglen;


mysock = socket(PF_INET,SOCK_DGRAM,0);
myaddr.sin_family = AF_INET;
myaddr.sin_port = htons( S_PORT );
myaddr.sin_addr.s_addr = htonl( INADDR_ANY );
bind(mysock, &myaddr, sizeof(myaddr));
while (1) {
        clilen=sizeof(cliaddr);
        msglen=recvfrom(mysock,msgbuf,
                MAXLEN,0,&cliaddr,&clilen);
        sendto(mysock,msgbuf,msglen,0,&cliaddr,clilen);
}
```

NEED TO CHECK FOR ERRORS!!!

Debugging

- **Debugging UDP can be difficult.**
- **Write routines to print out sockaddrs.**
- **Use trace, strace, ptrace, truss, etc.**
- **Include code that can handle unexpected situations.**

Timeout when calling `recvfrom()`

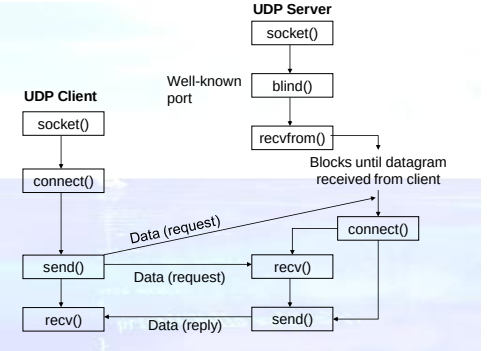
- **It might be nice to have each call to `recvfrom()` return after a specified period of time even if there is no incoming datagram.**
- **We can do this by using `SIGALRM` and wrapping each call to `recvfrom()` with a call to `alarm()`**

`recvfrom()` and `alarm()`

```
signal(SIGALRM, sig_alrm);
alarm(max_time_to_wait);
if (recvfrom(…)<0)
    if (errno==EINTR)
        /* timed out */
    else
        /* some other error */
else
    /* no error or time out
    - turn off alarm */
    alarm(0);
```

There are some other (better) ways to do this - check out section 14.2

UDP Client/Server Variant



Connected mode

- **A UDP socket can be used in a call to `connect()`.**
- **This simply tells the O.S. the address of the peer.**
- **No handshake is made to establish that the peer exists.**
- **No data of any kind is sent on the network as a result of calling `connect()` on a UDP socket.**
- **A default UDP socket is unconnected and becomes connected after calling `connect()`.**

19

Connected UDP

- **Once a UDP socket is *connected:***
  - can use `sendto()` with a null dest. address
  - can use `write()` and `send()`
  - can use `read()` and `recv()`
    - only datagrams from the peer will be returned.
  - do not need to use recvfrom() to learn the sender of a datagram, but `read()`, `recv()`, or `recvmsg()` instead.
  - asynchronous errors will be returned to the process. **(OS Specific, some won't do this!)**

20

Asynchronous Errors

- **What happens if a client sends data to a server that is not running?**
  - ICMP "port unreachable" error is generated by receiving host and sent to sending host.
  - The ICMP error may reach the sending host after sendto() has already returned!
  - The next call dealing with the socket could return the error.

21

UDP connect()

- **Connect() is typically used with UDP when communication is with a single peer only.**
- **Many UDP clients use connect().**
- **Some servers (TFTP).**
- **It is possible to disconnect and connect the same socket to a new peer.**

22