Elementary UDP Sockets © Dr. Ayman Abdel-Hamid, CS4254 Spring 2006

Outline

• Elementary UDP Sockets (Chapter 8)
  ➢ Information to write a complete UDP client and server

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 accept a connection from a client
• Server calls `recvfrom` function which waits until data arrives from some client

Socket functions for UDP client/server

UDP Server

- `socket()`
- `bind()`
- `recvfrom()`
- `sendto()`
- `close()`

UDP Client

- `socket()`
- `bind()`
- `recvfrom()`
- `sendto()`
- `close()`
recvfrom and sendto Function 1/2

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

- Both return the amount of user data in the datagram received
- Writing a datagram of length 0 is acceptable (return value from recvfrom?)
- Closing a UDP connection does not make sense?

UDP Echo server: main

```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(cliaddr));
  dg_echo(sockfd, (SA *) &cliaddr,sizeof(cliaddr));
}
```

UDP Echo server: dg_echo function

```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);
  }
}
```

TCP versus UDP server 1/2

Summary of TCP client-server with two clients.
TCP versus UDP server

**Socket receive buffer**

Summary of UDP client-server with two clients.

**UDP Echo Client: main**

```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 Echo Client: dg_cli function**

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

**Lost Datagrams**

- 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.
  - The only way to prevent this is to place a timeout on the recvfrom
  - Timeout not the entire solution
    - Do not know whether client datagram never made it to server
    - Or, Server reply never made it back
  - Will fix later

- Did not assign an ephemeral port to UDP Socket?
Verifying Received Response

// Need to return IP address and port of who sent back reply
// Source code in 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, pseraddr, servlen);
        len = servlen;
        n = Recvfrom(sockfd, recvline, MAXLINE, 0, preply_addr, &len);
        /* continued in next slide */
    }
}

Verifying Received Response 2/3

// Need to return IP address and port of who sent back reply
// Source code in udpcliserv/dgcliaddr.c

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

// Program can fail if server is multi-homed (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)

• Program can fail if server is multi-homed (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)

  ➢ Other 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
    ✓ Reply from this socket

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 (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
**connect function with UDP**

- This does not result in anything like a TCP connection: *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.

- With a connected UDP socket, **three things change**:
  1. We 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. We do not use `recvfrom` but `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.

---

**UDP Echo Client: dg_cli revisited**

```
#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);
    }

}  //ICMP error received after attempting to send the first datagram to server
```