Network Programming

Dr. Thaier Hayajneh

Computer Engineering Department

UDP Sockets

Simple Daytime Client 1

- Specify Server IP Address and Port
  - Fill an *Internet socket address structure* with server's IP address and port
  - Set entire structure to zero first using bzero
  - Set address family to AF_INET
  - Set port number to 13 (well-known port for daytime server on host supporting this service)
  - Set IP address to value specified as command line argument (argv[1])
  - IP address and port number must be in specific format
  - htons → host to network short
  - inet_pton → *presentation to numeric*, converts ASCII dotted-decimal command line argument (128.82.4.66) to proper format

Simple Daytime Client 2

- Establish connection with server
  - Connect (sockfd, (SA *) &servaddr, sizeof(servaddr))
  - Establish a TCP connection with server specified by socket address structure pointed to by second argument
  - Specify length of socket address structure as third argument
  - SA is #defined to be struct sockaddr in unp.h
- Read and Display server reply
  - ➢ Server reply normally a 26-byte string of the form

    Mon May 26 20:58:40 2003\r\n
  - ➢ TCP a *byte-stream* protocol, always code the read in a loop and terminate loop when read returns 0 (other end closed connection) or value less than 0 (error)

Simple Daytime Client 3

- Terminate program
  - ➢ Exit terminates the program exit (0)
  - ➢ Unix closes all open descriptors when a process terminates
  - ➢ TCP socket closed
- Program protocol dependent on IPv4, will see later how to change to IPv6 and even make it protocol independent

Error Handling: Wrapper Functions

- Check every function call for error return
- In previous example, check for errors from socket, inet_pton, connect, read, and fputs
- When error occurs, call textbook functions err_quit and err_sys to print an error message and terminate the program
- Define wrapper functions in wrapsock.c
- Unix errno value
  - When an error occurs in a Unix function, global variable errno is set to a positive value indicating the type of error and the function normally returns -1
  - err_sys function looks at errno and prints corresponding error message (e.g., connection timed out)

Simple Daytime Server 1/2

- Create a TCP Socket
  - Identical to client code
- Bind server well-known port to socket
  - Fill an Internet socket address structure
  - Call Bind (wrapper function) → local protocol address bound to socket
  - Specify IP address as INADDR_ANY: accept client connection on any interface (if server has multiple interfaces)
- Convert socket to listening socket
  - Socket becomes a listening socket on which incoming connections from clients will be accepted by the kernel
  - LISTENQ (defined in unp.h) specifies the maximum number of client connections the kernel will queue for this listening descriptor

Simple Daytime Server 2/2

- Accept client connection, send reply
  - Server is put to sleep (blocks) in the call to accept
  - After connection accepted, the call returns and the return value is a new descriptor called the *connected descriptor*
  - New descriptor used for communication with the new client
- Terminate connection
  - Initiate a TCP connection termination sequence
- Some Comments
  - Server handles one client at a time
  - If multiple client connections arrive at about the same time, kernel queues them up, up to some limit, and returns them to accept one at a time (An example of an iterative server, other options?)

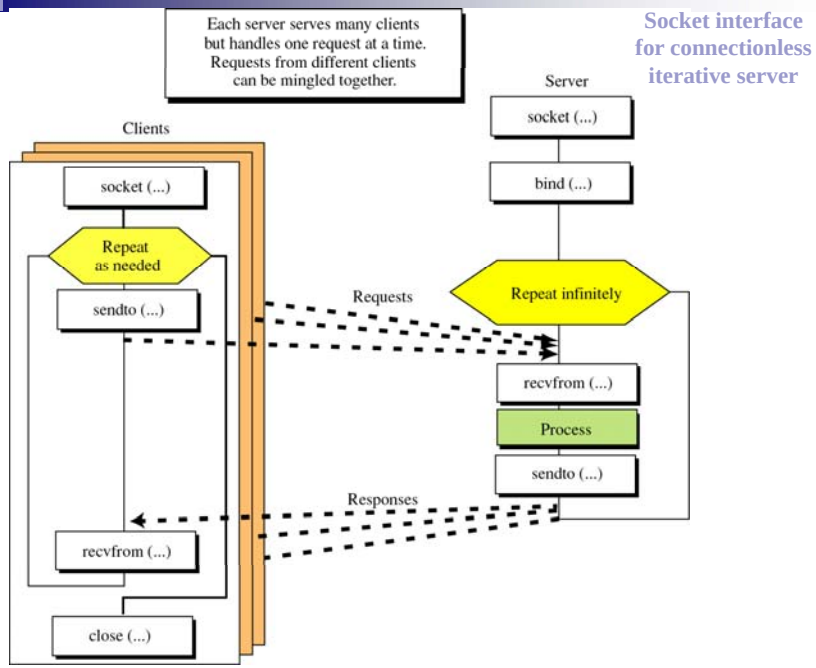
CONNECTIONLESS ITERATIVE SERVER

Connectionless Iterative UDP

- **The server serves one request at a time**
- **A server gets the request received in a datagram from UDP, processes the request, and gives the response to UDP to send to the client.**
- **The server pays no attention to the other datagrams**

Connectionless Iterative UDP(2)

- **The datagrams could be form one client or from many clients are stored in a queue waiting for service**
- **They are processed one by one in order of arrival**
- **The server uses one single port for this purpose, the well-known port**
- **All datagrams arriving at this port waits in line to be served.**



Server functions

- **Opening a socket**
  - **The server issues the socket call to ask the OS to create a socket**
  - **The socket call creates a new socket structure**
  - **The application program makes this call and passes three pieces of information:**
    - **Family**
    - **Type**
    - **Protocol**
  - **The OS creates a socket and enters the received information.**
  - **The OS returns an integer to define the socket uniquely (socket descriptor) which is used to refer to the socket in the following calls.**

Server functions (2)

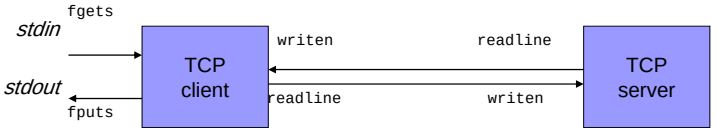
- **Binding**
  - The server issues the bind call to ask the OS to enter information in the socket structure created in the previous step (local socket address)
- **Steps to be repeated**
  - Receiving: The server issues the recvfrom call to tread from the incoming queue a datagram sent by a client
  - Sending: after processing the datagram the server issues the sendto call to send datagram that contains the result to the outgoing queue
  - The sendto call provides the remote socket address (client IP address and the client port number) for each datagram to be sent to the client (obtained by the recvfrom system call)

Client functions

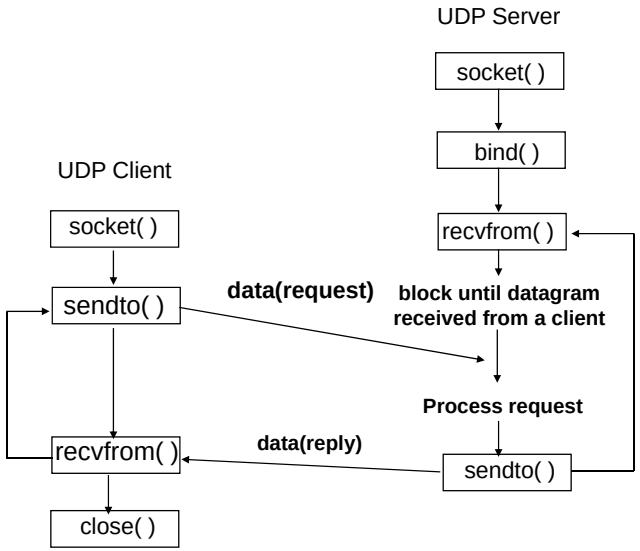
- **Opening a socket**
  - The client issues the socket call to ask the OS to create a socket
  - The client does not have to do binding because the local socket address can be provided by the OS.
  - The OS enters the local IP address and the ephemeral port number in the local socket address field of the created socket
- **Steps to be repeated**
  - Sending: after receiving the socket descriptor from the operating system the client issues the sendto calls to send its request to the server
  - Receiving: The client issues the recvfrom call to obtain the response of its request from the OS
- **Closing:**
  - when the client has no more requests it issues a close call to destroy the socket

UDP CLIENT-SERVER PROGRAMS

Simple echo client and server



1. The Client reads a line of text from its standard input and writes the line to the server.
2. The server reads the line from its network input and echoes the line back to the client.
3. The client reads the echoed line and prints it on its standard output.



Socket functions for UDP client-server

recvfrom and sendto Functions

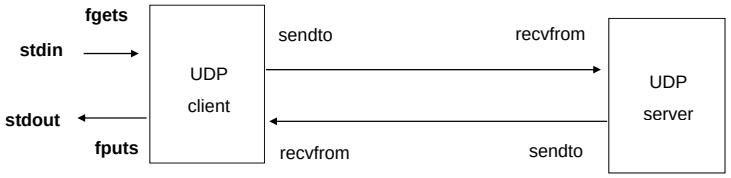
```
#include<sys/socket.h>

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

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

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

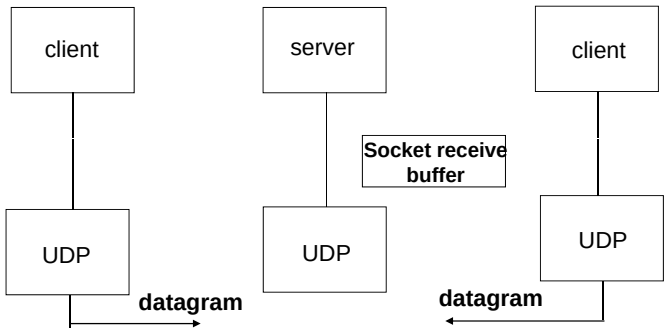
UDP simple echo client-server



Simple echo client-server using UDP

UDP echo server Function

```
#include “unp.h”
int main(int argc, char **argv)
{
    int sockfd, n;
    struck sockaddr_in servaddr, cliaddr;
    socklen_t len;
   char mesg[MAXLINE];
    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));
for( ; ; ) {
        len=sizeof(cliaddr);
        n=Recvfrom(sockfd, mseg, MAXLINE, 0, cliaddr, &len);
        sendto(sockfd, mesg, n, 0, pcliaddr, len);
    }
}
```



Summary of UDP client-server with two clients.

UDP echo client Function

```
#include “unp.h”
int main(int argc, char **argv)
{
    int sockfd, n;
    struct sockaddr_in servaddr;
    char sendline[MAXLINE], recvline[MAXLINE+1];

   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);
   char sendline[MAXLINE], recvline[MAXLINE+1];
     while(Fgets(sendline, MAXLINE, fp) != NULL)  {
          sendto(sockfd, sendline, strlen(sendline), 0, pservaddr, sizeof(servaddr);
          n = Recvfrom(sockfd, recvline, MAXLINE, 0, NULL, NULL);
          recvline[n] = 0;    /* null terminate */
          Fputs(recvline,stdout);
     }
   exit(0);
}
```

Course Project Phase(1)

- **Write a simple file storage UDP client-server program**
- **The client will send a request to the server asking for a file:**
  - **If the file is available then the server will inform the client and display a list of all available files.**
  - **If the file is not available then the client will send the file to the server.**
  - **The server will store all the files in a common directory and display all available files to the client upon request**
- **Due date: Thursday October 20, 2011**