Socket programming in C

Socket programming

Goal: learn how to build client/server application that communicate using sockets

Socket API

- introduced in BSD4.1 UNIX, 1981
- explicitly created, used, released by apps
- client/server paradigm
- two types of transport service via socket API:
  - unreliable datagram
  - reliable, byte stream-oriented

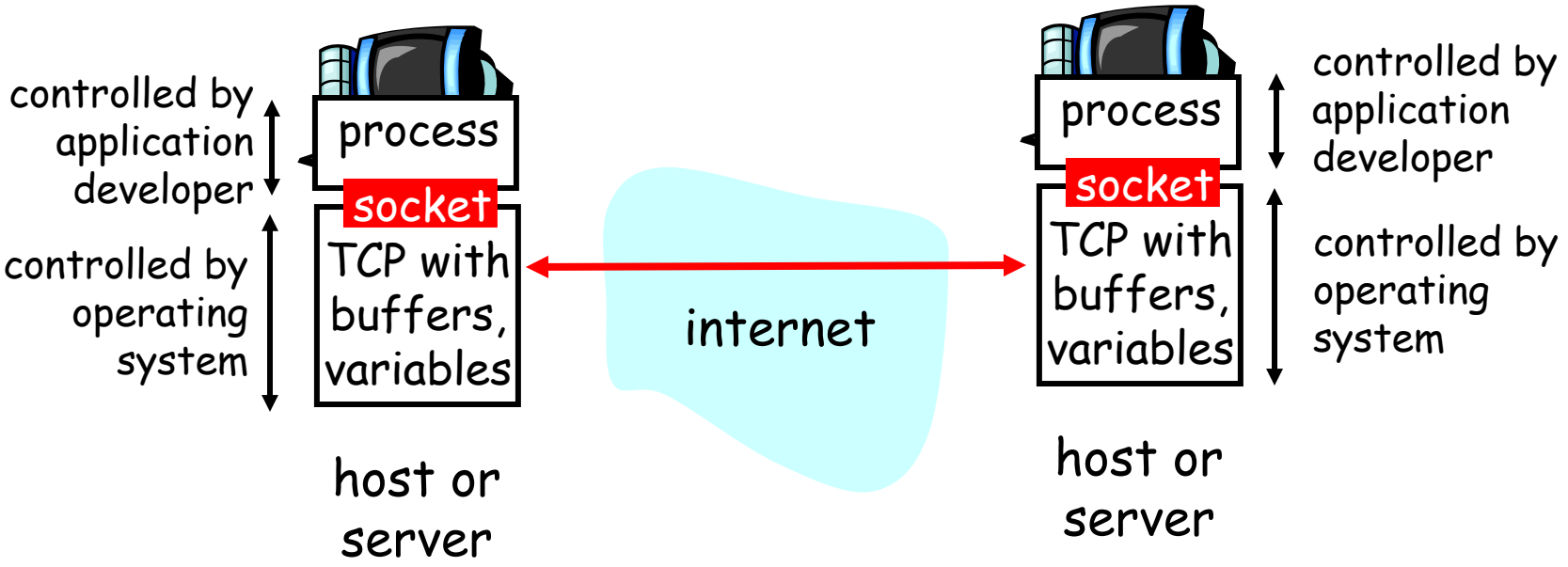
socket

a *host-local*, *application-created*, *OS-controlled* interface (a "door") into which application process can both send and receive messages to/from another application process

Socket-programming using TCP

Socket: a door between application process and end-end-transport protocol (UDP or TCP)

TCP service: reliable transfer of **bytes** from one process to another



Socket programming *with TCP*

Client must contact server

- server process must first be running
- server must have created socket (door) that welcomes client's contact

Client contacts server by:

- creating client-local TCP socket
- specifying IP address, port number of server process
- When client creates socket: client TCP establishes connection to server TCP

- When contacted by client, server TCP creates new socket for server process to communicate with client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients (more in Chap 3)

application viewpoint

TCP provides reliable, in-order transfer of bytes ("pipe") between client and server

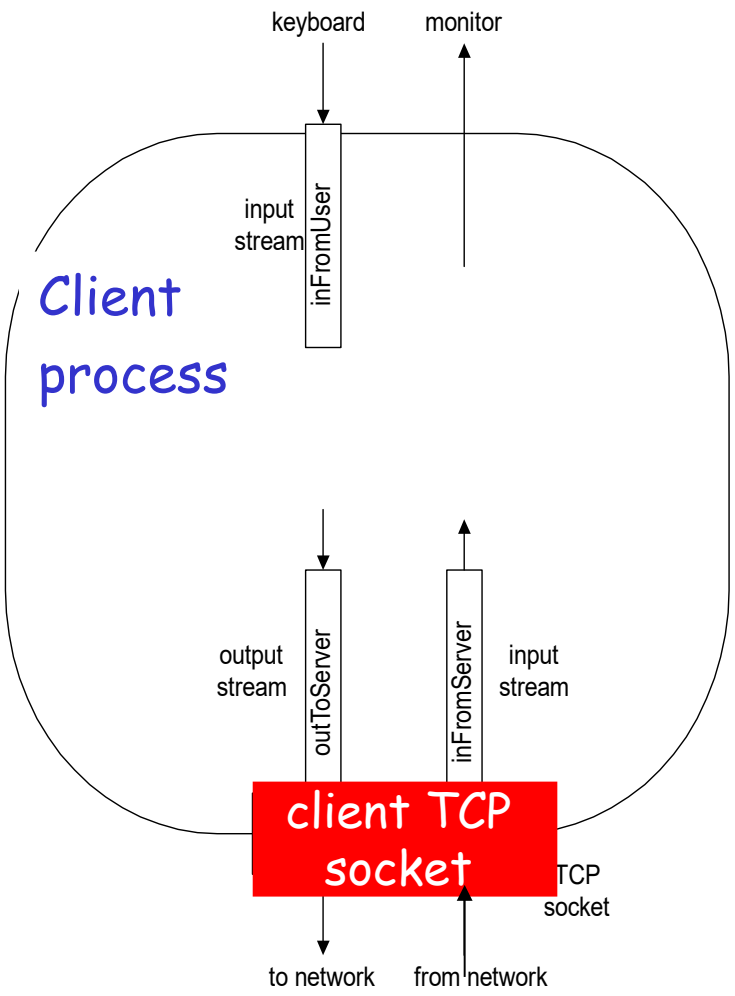
Stream jargon

- A stream is a sequence of characters that flow into or out of a process.
- An input stream is attached to some input source for the process, eg, keyboard or socket.
- An output stream is attached to an output source, eg, monitor or socket.

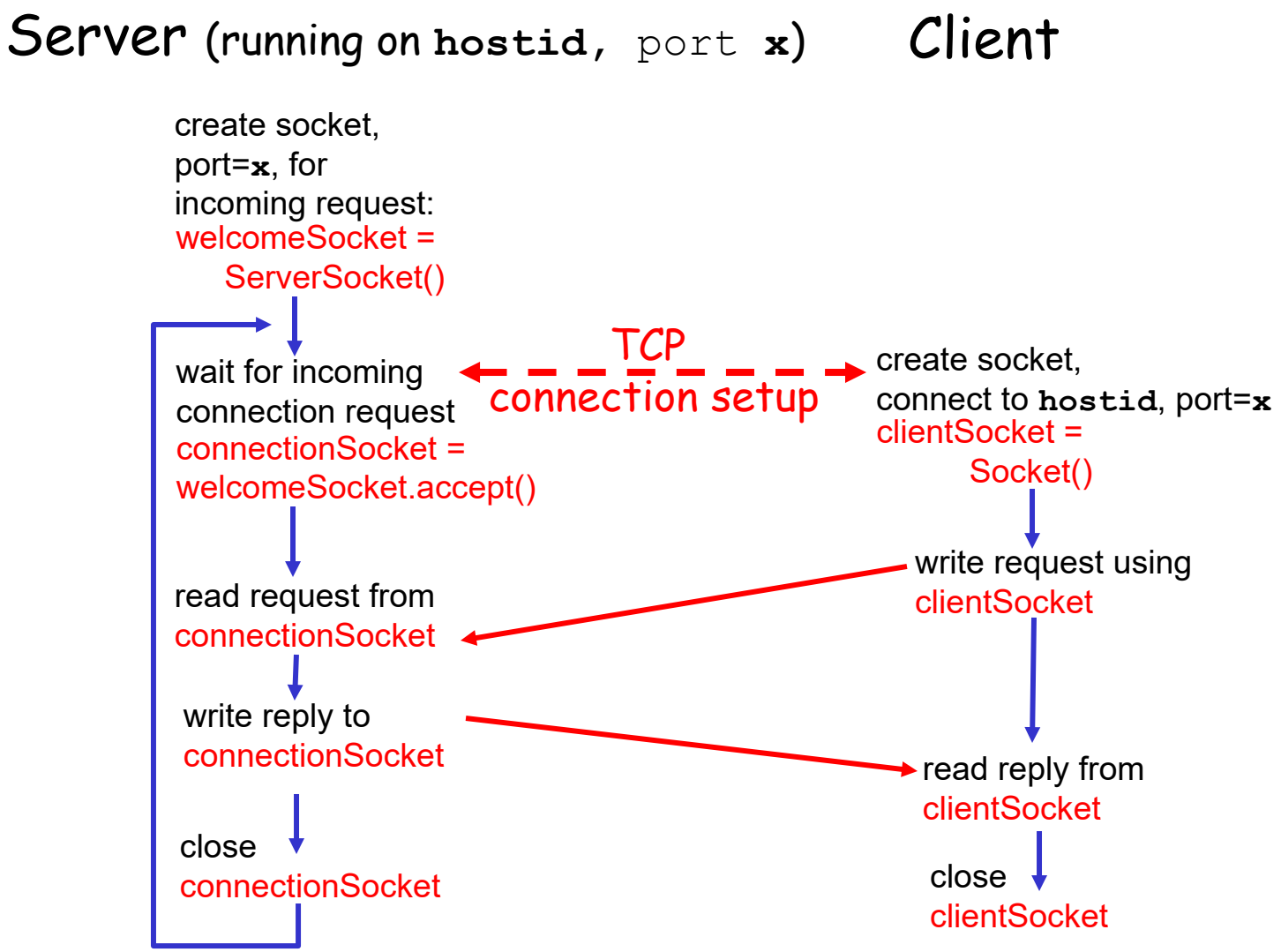
Socket programming with TCP

Example client-server app:

1) client reads line from standard input (**`inFromUser`** stream) , sends to server via socket (**`outToServer`** stream)
2) server reads line from socket
3) server converts line to uppercase, sends back to client
4) client reads, prints modified line from socket (**`inFromServer`** stream)



Client/server socket interaction: TCP



Example: C client (TCP)

```
/* client.c  */
void main(int argc, char *argv[])
{
struct sockaddr_in sad; /* structure to hold an IP address */
int clientSocket; /* socket descriptor */
struct hostent *ptrh; /* pointer to a host table entry */

char Sentence[128];
char modifiedSentence[128];

host = argv[1]; port = atoi(argv[2]);

clientSocket = socket(PF_INET, SOCK_STREAM, 0);
        memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr structure */
        sad.sin_family = AF_INET; /* set family to Internet */
        sad.sin_port = htons((u_short)port);
        ptrh = gethostbyname(host); /* Convert host name to IP address */
        memcpy(&sad.sin_addr, ptrh->h_addr, ptrh->h_length);
connect(clientSocket, (struct sockaddr *)&sad, sizeof(sad));
```

Create client socket, connect to server

Example: C client (TCP), cont.

Get input stream from user

```
gets(Sentence);
```

Send line to server

```
n=write(clientSocket, Sentence, strlen(Sentence)+1);
```

Read line from server

```
n=read(clientSocket, modifiedSentence,
sizeof(modifiedSentence));

printf("FROM SERVER: %s\n",modifiedSentence);
```

Close connection

```
close(clientSocket);

}
```

Example: C server (TCP)

```
/* server.c  */
void main(int argc, char *argv[])
{
struct sockaddr_in sad; /* structure to hold an IP address */
struct sockaddr_in cad;
int welcomeSocket, connectionSocket; /* socket descriptor */
struct hostent *ptrh; /* pointer to a host table entry */

char clientSentence[128];
char capitalizedSentence[128];

port = atoi(argv[1]);

welcomeSocket = socket(PF_INET, SOCK_STREAM, 0);
        memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr structure */
        sad.sin_family = AF_INET; /* set family to Internet */
        sad.sin_addr.s_addr = INADDR_ANY; /* set the local IP address */
        sad.sin_port = htons((u_short)port);/* set the port number */
bind(welcomeSocket, (struct sockaddr *)&sad, sizeof(sad));
```

Create welcoming socket at port
&
Bind a local address

Example: C server (TCP), cont

```
/* Specify the maximum number of clients that can be queued */
listen(welcomeSocket, 10)

while(1) {

    connectionSocket=accept(welcomeSocket, (struct sockaddr *)&cad, &alen);

    n=read(connectionSocket, clientSentence, sizeof(clientSentence));

    /* capitalize Sentence and store the result in capitalizedSentence*/


    n=write(connectionSocket, capitalizedSentence, strlen(capitalizedSentence)+1);

    close(connectionSocket);
    }
}
```

Wait, on welcoming socket for contact by a client

Write out the result to socket

End of while loop, loop back and wait for another client connection

Socket programming *with UDP*

UDP: no "connection" between client and server

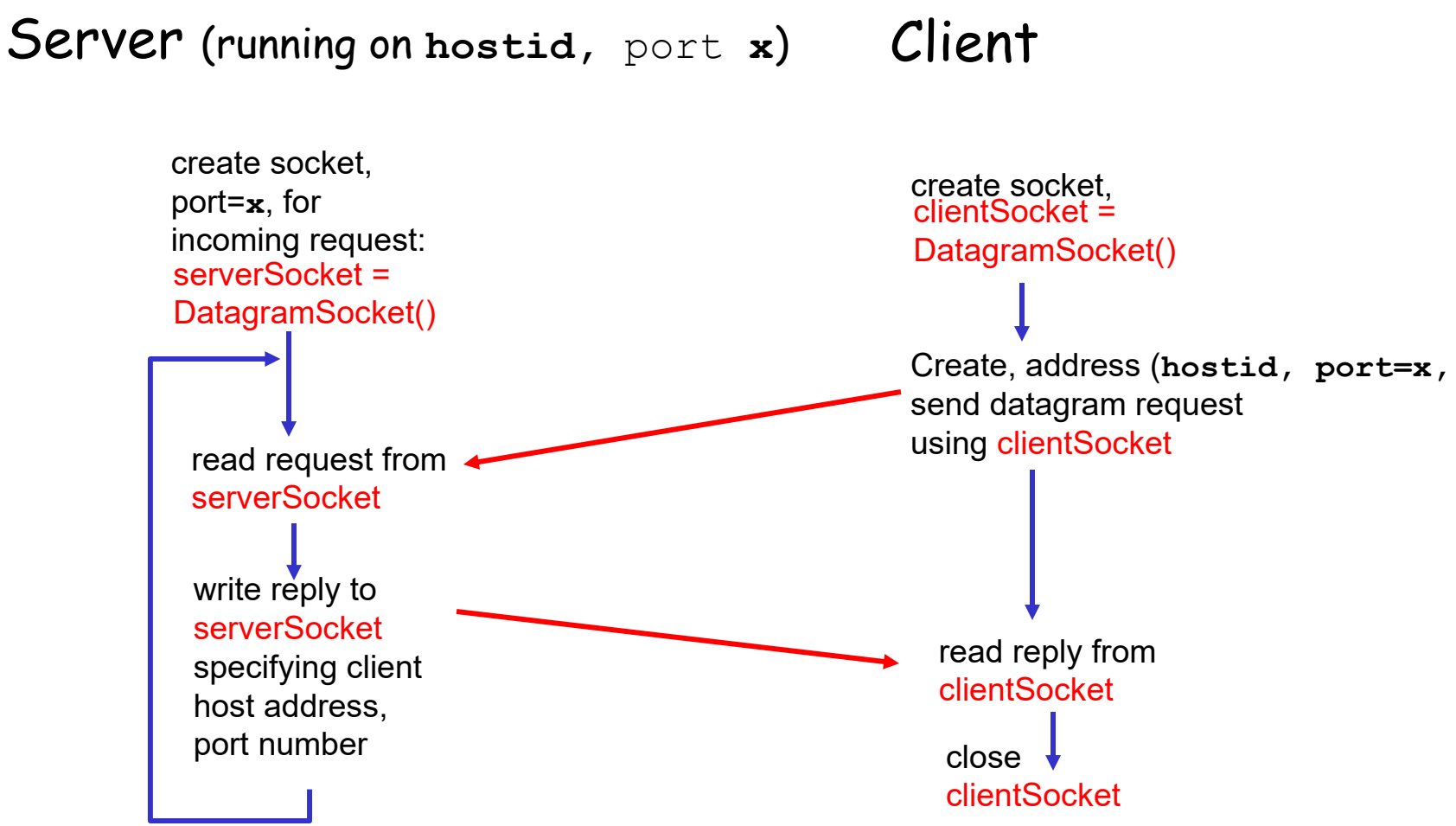
- no handshaking
- sender explicitly attaches IP address and port of destination to each packet
- server must extract IP address, port of sender from received packet

UDP: transmitted data may be received out of order, or lost

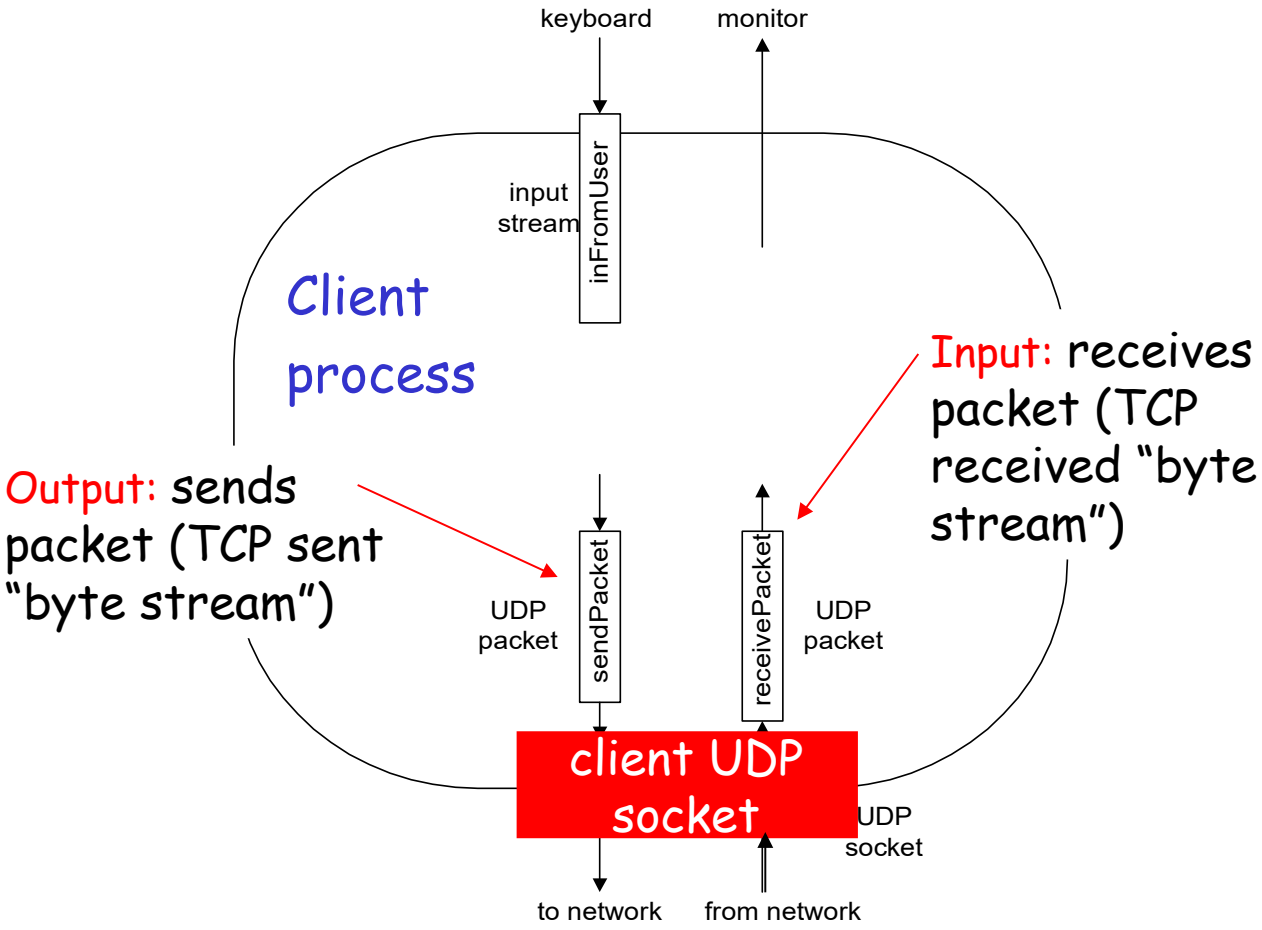
application viewpoint

UDP provides unreliable transfer of groups of bytes ("datagrams") between client and server

Client/server socket interaction: UDP



Example: Java client (UDP)



Example: C client (UDP)

```
/* client.c  */
void main(int argc, char *argv[])
{
struct sockaddr_in sad; /* structure to hold an IP address */
int clientSocket; /* socket descriptor */
struct hostent *ptrh; /* pointer to a host table entry */

char Sentence[128];
char modifiedSentence[128];

host = argv[1]; port = atoi(argv[2]);

clientSocket = socket(PF_INET, SOCK_DGRAM, 0);

/* determine the server's address */
        memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr structure */
        sad.sin_family = AF_INET; /* set family to Internet */
        sad.sin_port = htons((u_short)port);
        ptrh = gethostbyname(host); /* Convert host name to IP address */
        memcpy(&sad.sin_addr, ptrh->h_addr, ptrh->h_length);
```

Create client socket,
NO connection to server

Example: C client (UDP), cont.

Get input stream from user

```
gets(Sentence);
```

Send line to server

```
addr_len =sizeof(struct sockaddr);
n=sendto(clientSocket, Sentence, strlen(Sentence)+1,
                (struct sockaddr *) &sad, addr_len);
```

Read line from server

```
n=recvfrom(clientSocket, modifiedSentence, sizeof(modifiedSentence)
                (struct sockaddr *) &sad, &addr_len);
```

```
printf("FROM SERVER: %s\n",modifiedSentence);
```

Close connection

```
close(clientSocket);
}
```

Example: C server (UDP)

```
/* server.c  */
void main(int argc, char *argv[])
{
struct sockaddr_in sad; /* structure to hold an IP address */
struct sockaddr_in cad;
int serverSocket; /* socket descriptor */
struct hostent *ptrh; /* pointer to a host table entry */

char clientSentence[128];
char capitalizedSentence[128];

port = atoi(argv[1]);

serverSocket = socket(PF_INET, SOCK_DGRAM, 0);
        memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr structure */
        sad.sin_family = AF_INET; /* set family to Internet */
        sad.sin_addr.s_addr = INADDR_ANY; /* set the local IP address */
        sad.sin_port = htons((u_short)port);/* set the port number */
bind(serverSocket, (struct sockaddr *)&sad, sizeof(sad));
```

Create welcoming socket at port
&
Bind a local address

Example: C server (UDP), cont

```
while(1) {

    n=recvfrom(serverSocket, clientSentence, sizeof(clientSentence), 0
                    (struct sockaddr *) &cad, &addr_len );

    /* capitalize Sentence and store the result in capitalizedSentence*/

    n=sendto(serverSocket, capitalizedSentence, strlen(capitalizedSentence)+1,0
                (struct sockaddr *) &cad, &addr_len);

    }
}
```

Receive messages from clients

Write out the result to socket

End of while loop,
loop back and wait for
another client connection

Other functions

- getpeername( )
- gethostbyname( )
- gethostbyaddr( )

- getsockopt( )
- setsockopt ( )
- signal(SIGINT,sigf);

```
if ( (pid=fork()) == 0) {
   /* CHILD PROC */
   close(welcomeSocket);
   /* give service */
   exit(0);
}
/* PARENT PROC */
close(connectionSocket);
```

Waiting something from socket and stdin

```
FD_ZERO(&rset);
FD_SET(welcomeSocket, &rset);
FD_SET(fileno(stdin), &rset);
maxfd =max(welcomeSocket,fileno(stdin)) + 1;
select(maxfd, &rset, NULL,  NULL, NULL);
if (FD_ISSET(fileno(stdin), &rset)){
   /* read something from stdin */
}
```