# Socket Programming

- Socket programming with TCP
- Socket programming with UDP

# 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 applications
- client/server paradigm
- two types of transport service via socket API:
  - unreliable datagram (UDP)
  - reliable, byte stream-oriented (TCP)

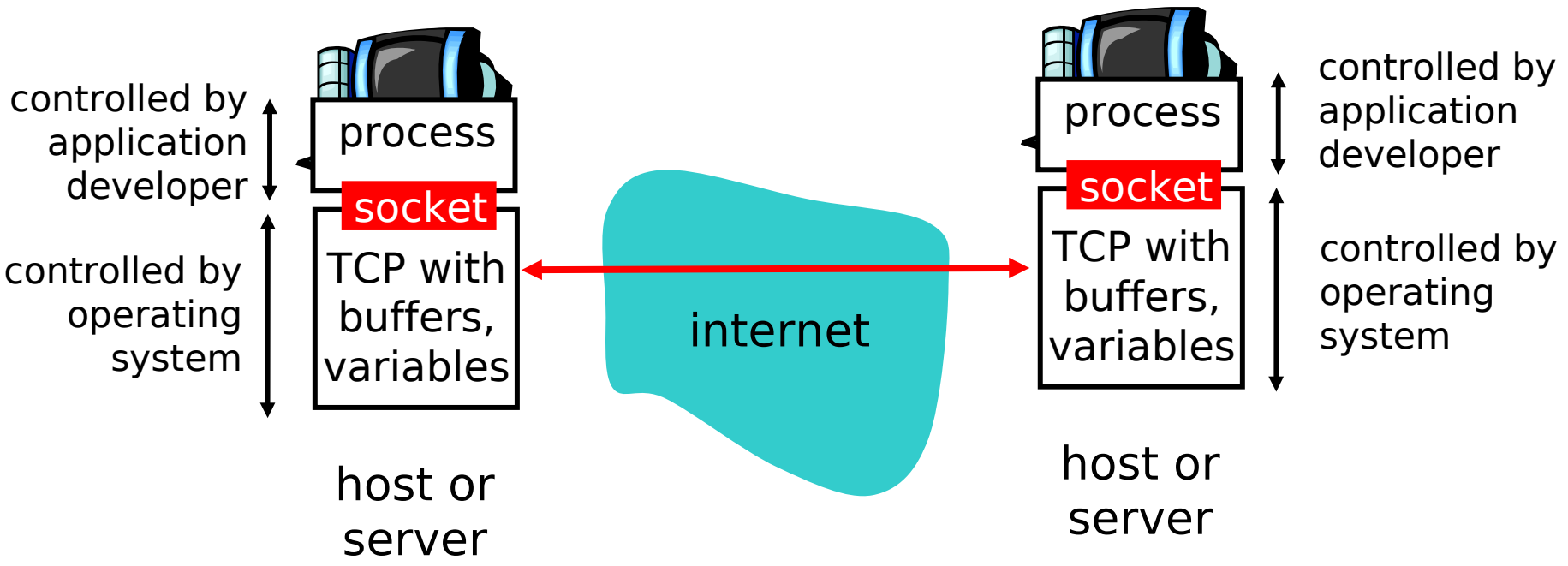
**socket**

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 (UCP 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:

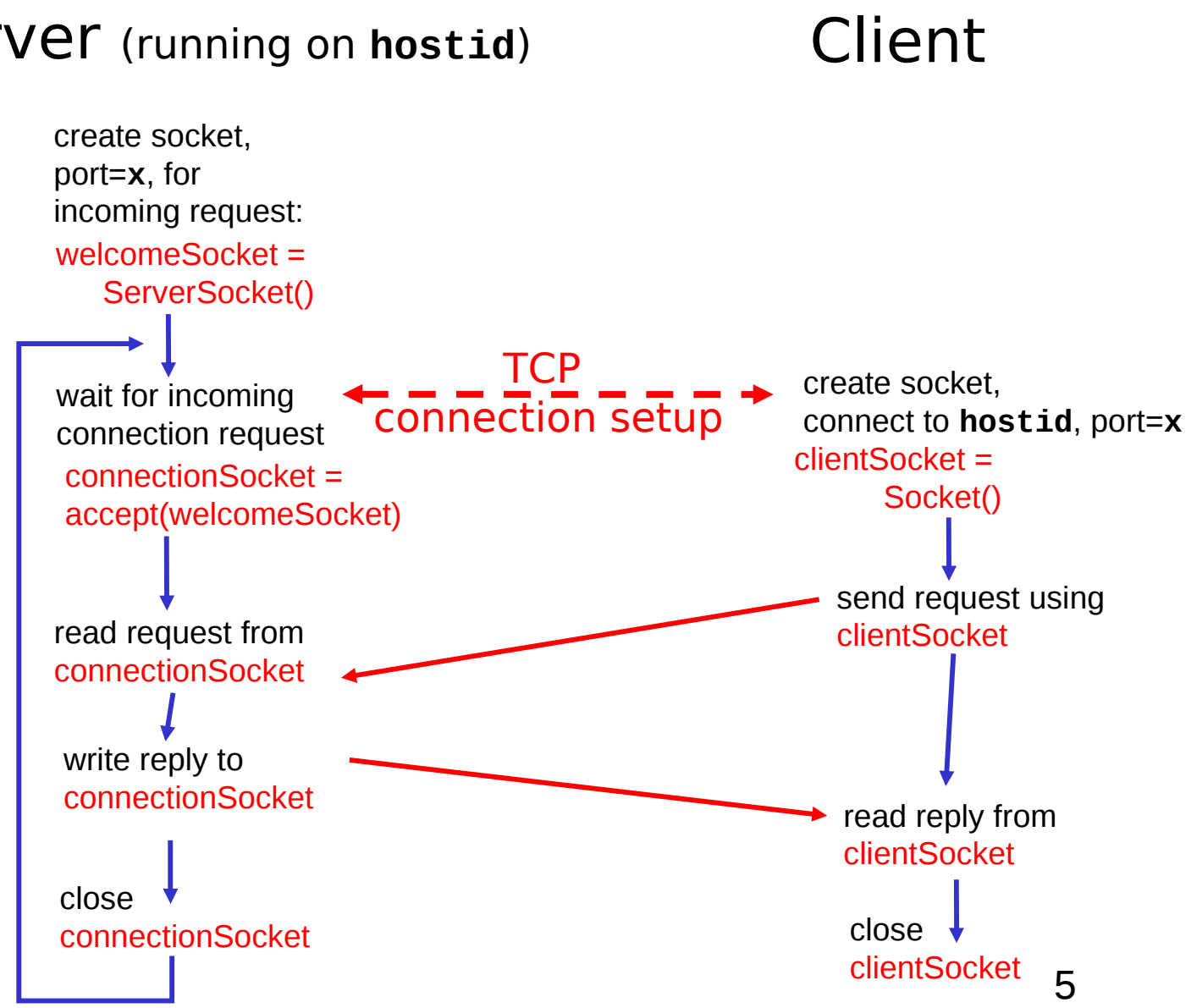
- creating client-local TCP socket
- specifying IP address, port number of server process
- When client creates socket: client TCP establishes connection to server TCP

- 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*

# Client/server socket interaction: TCP



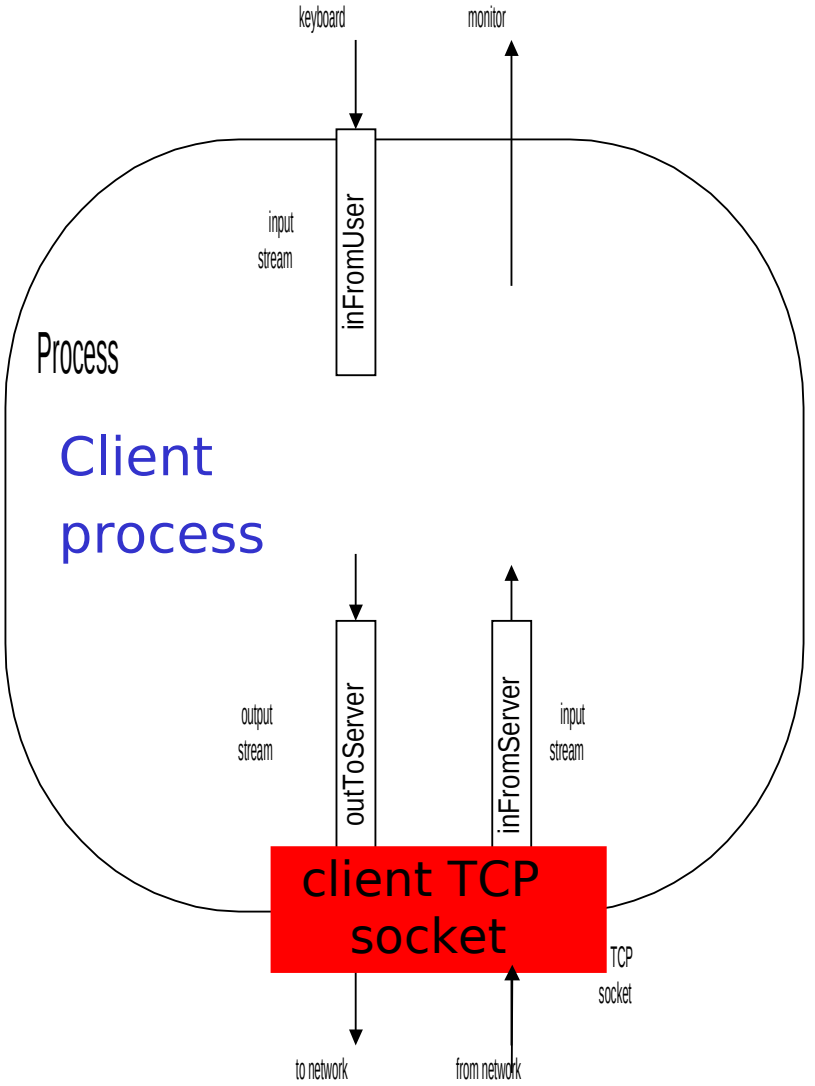
# Stream jargon

- 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.

e.g., keyboard or socket (stdin)

- An output stream is attached to an output source.

e.g., monitor or socket (stdout)



# Socket programming with TCP

Example client-server app:

1) client reads line from standard input (**`stdin`** stream) , sends to server via socket
2) server reads line from socket
3) server counts the number of characters sent, and respondes echoing the message with the count number
4) client reads the response and prints it.

# Simple server pseudo-code...

```
socket
bind
listen
while (1) { /*server available for new connections*/
      accept            /*creating newsocket) */
       while (until_done) {   /* process the clients request*/

            receive
            send
      }
      closesocket(newsocket)
}
```

# Simple client pseudo-code...

```
socket
connect
/* process the clients request*/
while (1 || until_done) {
    send
    receive
}
closesocket(socket)
```

# Programming:
# the header files in Windows

```
#include <winsock.h>
#define WSVERS MAKEWORD(2,0)
WSADATA wsadata;

if (WSAStartup(WSVERS,&wsadata) != 0) {
      printf("WSAStartup failed\n");
      WSACleanup();
      exit(1);
}
```

# Programming:
# the header files in Unix

```
#include <sys/types.h>
#include <sys/socket.h>
//in some cases
#include  <netinet/in.h>
```

# Programming: Data Structures

- sockaddr_in

```
struct sockaddr_in {
   short          sin_family;
   u_short        sin_port;
   struct in_addr sin_addr;
   char           sin_zero[8];
};
```

  - **sockaddr_in** contains 4 fields.
  - assume that **sin_family** specify TCP/IP (AF_INET)
  - **sin_port** specifies a ***Port number***
  - **sin_addr** a 32-bit ***IP address***
  - **sin_zero** Humm... that should be set to zeros!
    - Pretty common in Networking...related to past or future use.

# Programming: Data Structures (cont.)

Old way:

```
struct in_addr {
  union {
  struct { u_char s_b1,s_b2,s_b3,s_b4; }S_un_b;
  struct { u_short s_w1,s_w2; } S_un_w;
  u_long s_addr;
 } s_un;
```

“New” way:

```
  struct in_addr {
      uint32_t s_addr;// a single 32 bit address
  }
```

# IP address and port number

- Both are human unfriendly formats, how to we convert from and to the appropriate formats?
- Short answer:
  - Use given conversion functions…
  - `htons()` to convert port numbers
    - "Host to Network Short"
    - Also `htonl(), ntohs(), ntohl()`
  - `inet_addr()` to convert IP addresses
    - Which format?? From X.Y.Z.T to binary
    - "presentation to network"
    - Also `inet_aton(), inet_pton(), inet_ntop()`

# Programming: socket commands

- Creating and closing sockets

```
socket(int protocol_family, int transport, int
  zero);
```

- Protocol_family: **AF_INET** (PF_INET) for TCP/IP.
  - AF=address family, PF=protocol family
- Transport:
  - SOCK_STREAM for **TCP**
  - SOCK_DGRAM for **UDP** (more later...)
- zero is set to ... **0**

Example:

```
s = socket(PF_INET, SOCK_STREAM, 0);

closesocket(SOCKET s);
```

# socket commands (cont)...

- and making connections to a remote computer

connect(SOCKET s, struct sockaddr *destaddr, int addrlen);
Example:

```
connect(s, (struct sockaddr *)&sin, sizeof(sin))
```

- To **accept** the connection (server only):

SOCKET accept(SOCKET s, struct sockaddr *addr, int *addrlen);
Example:

```
ns=accept(s,(struct sockaddr *)(&remoteaddr),&addrlen);
```

# socket commands: Send

- Send data through a socket:

```
send(SOCKET s, char *msg, int msglen, int flags);
```

s = socket (inside the socket descriptor: *port* and *IP address*...)
msg = a pointer to a buffer (could be a string)
msglen = the length of the buffer
flags = 0 (forget about them for this exercise...)

Example:

```
send(s, sbuffer, strlen(sbuffer),0);
```

# socket commands: Receive

- Receive data:

int recv(SOCKET s, char *msg, int msglen, int flags);

s = socket
msg = pointer to a buffer
msglen = length of the buffer
flags = 0

Example:

```
recv(s, &rbuffer[n], 1, 0);
```

# socket commands: Bind

- Binding a port to a process (server only):

int bind(SOCKET s, struct sockaddr *localaddr, int addrlen);

A *server* has to *bind* port|process, so *clients* can access a known port for a service

localaddr = pointer to a local address stucture
addrlen = length of localaddr

Example:

```
bind(s,(struct sockaddr *)
  (&localaddr),sizeof(localaddr));
```

# socket commands:Listen

- Listen (server only):

int listen(SOCKET s, int queuelen);

A *server* listens using socket s, queues requests if there is more then one.

The queue limit vary (for windows up to 5).

Example:

```
listen(s,4);
```

# Examples in C

- Read the codes carefully
- Download, compile and run the examples
- Pay attention to details
  - The code is NOT error proof, e.g.:
    - What happens if type more than BUFFSIZE chars?
    - fgets() may misbehave (e.g., add a '\n')
    - Change / experiment
- More details when FTP assignment is explained

# Example: C client (TCP)

Initialisation

Remote IP address

Remote Port numb.

```
#include ...
int main(int argc, char *argv[]) {
//*************************************************************
// Initialization
//*************************************************************
   struct sockaddr_in remoteaddr;
   memset(&sin, 0, sizeof(remoteaddr));             //clean up
   int s;          //SOCKET s
   char send_buffer[80],receive_buffer[80];
   int n,bytes;
//*************************************************************
//   Dealing with user's arguments
//*************************************************************
   if (argc == 1) {
      printf("USAGE: client IP-address [port]\n");
      exit(1);
   }
//*************************************************************
//   Dealing with user's arguments
//*************************************************************
   if (argc != 3) {
      printf("USAGE: client IP-address port\n");
      exit(1);
   }
   else {
      remoteaddr.sin_addr.s_addr = inet_addr(argv[1]);//IP address
      remoteaddr.sin_port = htons((u_short)atoi(argv[2]));//Port number
   }
```

# Example: C client (TCP), cont.

Create socket 's'

Connect to remote IP/Port

Prompt user to type a line

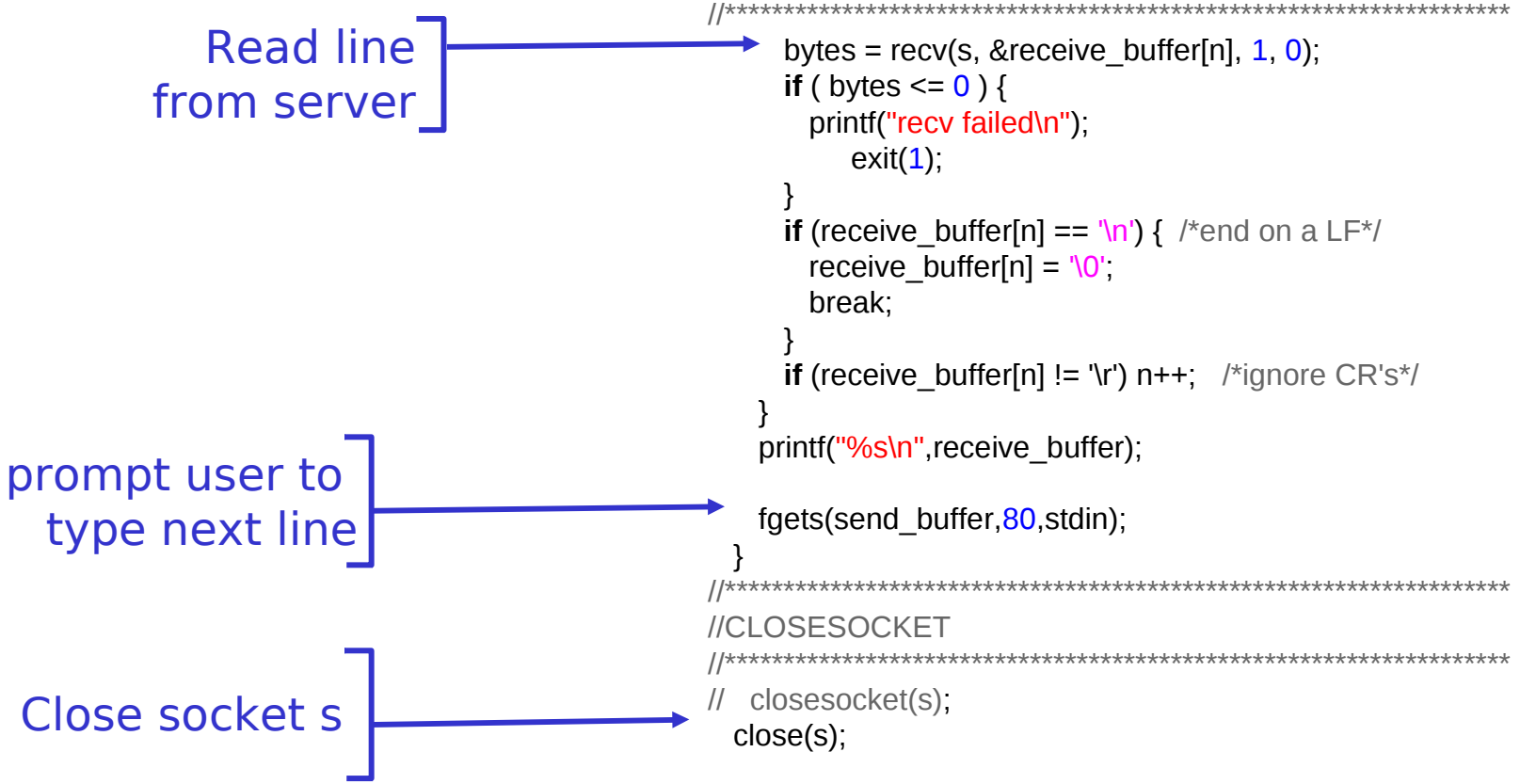
Send line to server

```
//*****************************************************************
//CREATE CLIENT'S SOCKET
//*****************************************************************
  s = socket(AF_INET, SOCK_STREAM, 0);
  if (s < 0) {
     printf("socket failed\n");
         exit(1);
  }
//*****************************************************************
//CONNECT
//*****************************************************************
  if (connect(s, (struct sockaddr *)&remoteaddr, sizeof(remoteaddr)) != 0) {
     printf("connect failed\n");
         exit(1);
  }
//*****************************************************************
//Get input while user don't type "."
//*****************************************************************
    //gets(send_buffer);//NEVER use 'gets', you may be vulnerable to attacks using buffer overflows
    fgets(send_buffer,80,stdin);

    while (strncmp(send_buffer,".",1) != 0) {

//*****************************************************************
//SEND
//*****************************************************************
      bytes = send(s, send_buffer, strlen(send_buffer),0);
      if (bytes < 0) {
        printf("send failed\n");
         exit(1);
      }
```

# Example: C client (TCP), cont.




```
   n = 0;
   while (1) {
//*********************************************************************
//RECEIVE
//*********************************************************************
      bytes = recv(s, &receive_buffer[n], 1, 0);
      if ( bytes <= 0 ) {
         printf("recv failed\n");
              exit(1);
      }
      if (receive_buffer[n] == '\n') {  /*end on a LF*/
         receive_buffer[n] = '\0';
         break;
      }
      if (receive_buffer[n] != '\r') n++;   /*ignore CR's*/
    }
    printf("%s\n",receive_buffer);

    fgets(send_buffer,80,stdin);
  }
//*********************************************************************
//CLOSESOCKET
//*********************************************************************
//   closesocket(s);
   close(s);

   return 0;
}
```

# Example: C server (TCP)

```
main(int argc, char *argv[]) {
//*********************************************************************
// INITIALIZATION
//*********************************************************************
   struct sockaddr_in localaddr,remoteaddr;
   int s,ns;
   char send_buffer[80],receive_buffer[80];
```

```
   memset(&localaddr,0,sizeof(localaddr));    //clean up the structure
   memset(&localaddr,0,sizeof(remoteaddr));    //clean up the structure
//*********************************************************************
//SOCKET
//*********************************************************************
```

Create welcoming socket

```
   s = socket(PF_INET, SOCK_STREAM, 0);
   if (s <0) {
      printf("socket failed\n");
   }
```

Default port 1234

```
   localaddr.sin_family = AF_INET;
   if (argc == 2) localaddr.sin_port = htons((u_short)atoi(argv[1]));
   else localaddr.sin_port = htons(1234);   //default listening port
   localaddr.sin_addr.s_addr = INADDR_ANY;   //server address should be local
```

# Example: C server (TCP)

Bind socket to port

Listen on socket s

Wait: welcoming socket for contact by client

```
//*****************************************************************
//BIND
//*****************************************************************
  if (bind(s,(struct sockaddr *)(&localaddr),sizeof(localaddr)) != 0) {
    printf("Bind failed!\n");
    exit(1);
  }
//*****************************************************************
//LISTEN
//*****************************************************************
  listen(s,5);
//*****************************************************************
//INFINITE LOOP
//*****************************************************************
  while (1) {
    addrlen = sizeof(remoteaddr);
//*****************************************************************
//NEW SOCKET newsocket = accept
//*****************************************************************
    ns = accept(s,(struct sockaddr *)(&remoteaddr),&addrlen);
    if (ns <0 ) break;
    inet_ntop(AF_INET,&(remoteaddr.sin_addr), remoteIP, INET_ADDRSTRLEN);
    printf("connected to IP %s at port %d \n",remoteIP,ntohs(localaddr.sin_port));
```

# Example: C server (TCP), cont

```
while (1) {
        n = 0;
        while (1) {
//**********************************************************************
//RECEIVE
//**********************************************************************
            bytes = recv(ns, &receive_buffer[n], 1, 0);     //receive byte by byte...
//**********************************************************************
//PROCESS REQUEST
//**********************************************************************
            if ( bytes <= 0 ) break;
            if (receive_buffer[n] == '\n') {            /*end on a LF*/
              receive_buffer[n] = '\0';
              break;
            }
            if (receive_buffer[n] != '\r') n++;       /*ignore CRs*/
          }
          if ((bytes < 0) || (bytes == 0)) break;
          sprintf(send_buffer, "The client typed '%s' -  %d bytes\r\n", receive_buffer, n);
//**********************************************************************
//SEND
//**********************************************************************
          bytes = send(ns, send_buffer, strlen(send_buffer), 0);
          if (bytes < 0) break;
       }
//**********************************************************************
//CLOSE SOCKET
//**********************************************************************
      close(ns);
      printf("disconnected from %s\n",remoteIP);
   }
   close(s);
   return 0;
}
```

Read line from socket

Send line to socket

End of while loop, close the connec. wait for another client

# Running the sample codes

- Download Client.c and Serv1.c for the corresponding OS
- Compile both programs separately
- You should get two *.exe
- Run from command line:

```
server2012.exe 1234
```

In a separate window, run:

```
client2012.exe 127.0.0.1 1234
```

NOTES:

- Choose any port number you like, > 1024 (why?)
- Find your IP address using:
  - `ipconfig` (Windows)
  - `ifconfig` (Linux)
- The IP address for the same machine is **`127.0.0.1`**