Introduction to Socket programming using C

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

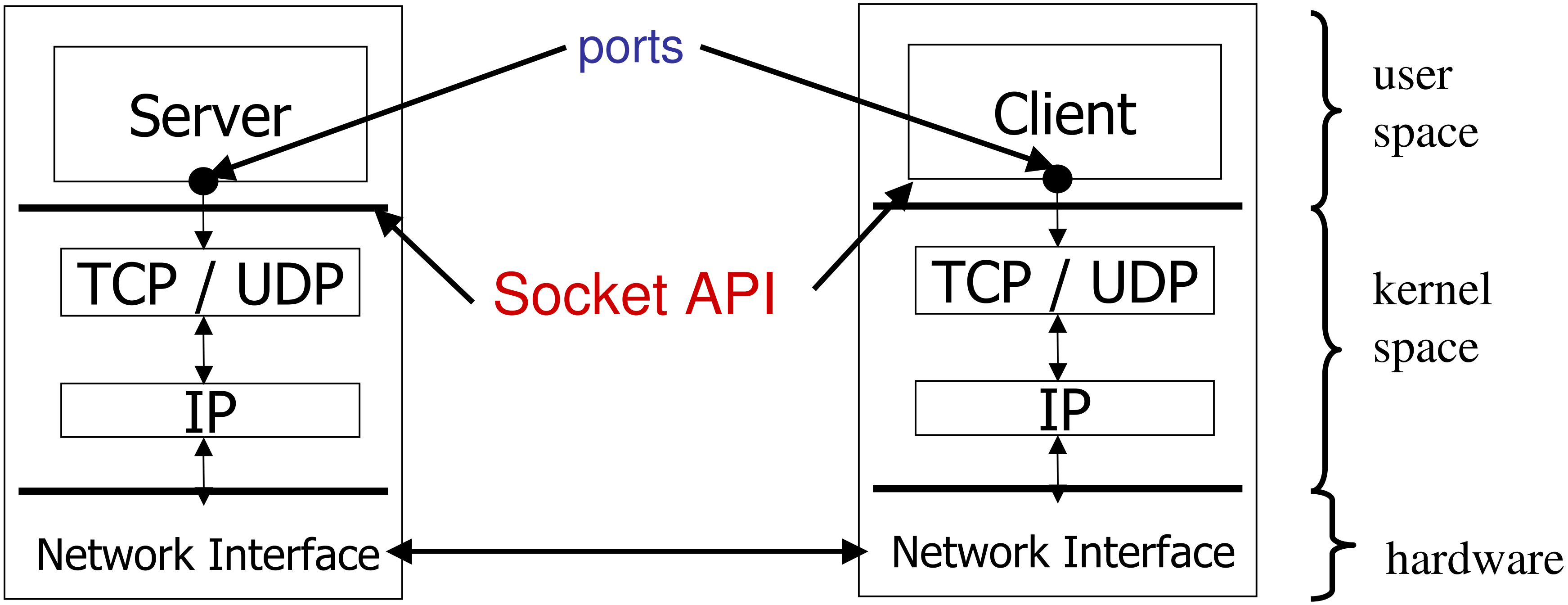
Vinay Narasimhamurthy
S0677790@sms.ed.ac.uk

CLIENT – SERVER MODEL

Sockets are used for interprocess communication.

Most of the interprocess communication follow a Client-Server Model, where client and server are two separate processes in itself.

Server and Client exchange messages over the network through a common Socket API



* **Port numbers**: **1,024 -- 65,535 (2^{16} - 1)**

Server Examples

- Web server (port 80)
- FTP server (20, 21)
- Telnet server (23)
- Mail server (25)

Client Examples

- Examples of client programs
  - Web browsers, `ftp`, `telnet`, `ssh`
- How does a client find the server?
  - The IP address in the server socket address identifies the host
  - The (well-known) port in the server socket address identifies the service, and thus implicitly identifies the server process that performs that service.
  - Examples of well know ports
    - Port 7: Echo server
    - Port 23: Telnet server
    - Port 25: Mail server
    - Port 80: Web server

What is an API ?

API expands as Application Programming Interface.

A set of routines that an application uses to request and carry out lower-level services performed by a computer's operating system.

What is a socket?

- An interface between application and network which is used for communication between processes
- Once configured the application can
  - pass data to the socket for network transmission
  - receive data from the socket (transmitted through the network by some other host)
- To the kernel, a socket is an endpoint of communication.
- To an application, a socket is a file descriptor that lets the application read/write from/to the network.
- Clients and servers communicate with each by reading from and writing to socket descriptors.
    - Remember: All Unix I/O devices, including networks, are modeled as files.

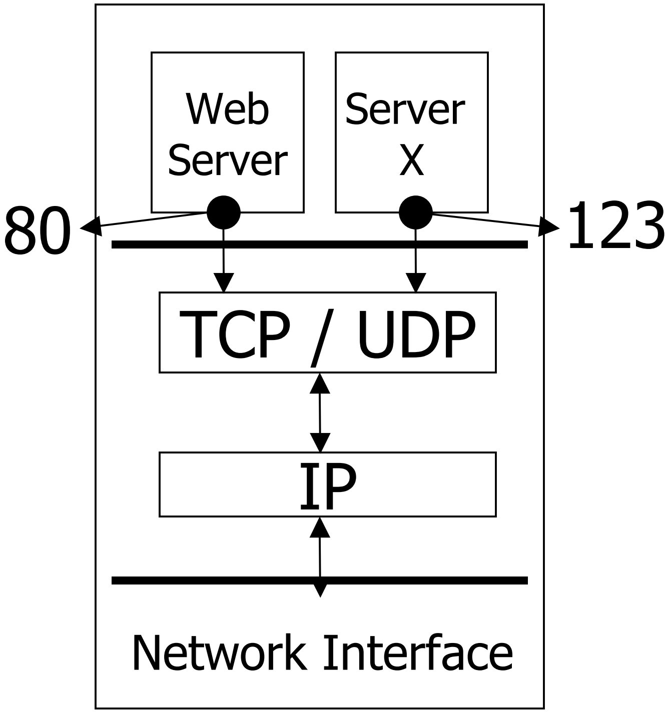
Two essential types of sockets

- SOCK_STREAM
  - TCP
  - connection-oriented
  - reliable delivery
  - in-order guaranteed
  - bidirectional

- SOCK_DGRAM
  - UDP
  - no notion of "connection" – app indicates dest. for each packet
  - unreliable delivery
  - no order guarantees
  - can send or receive

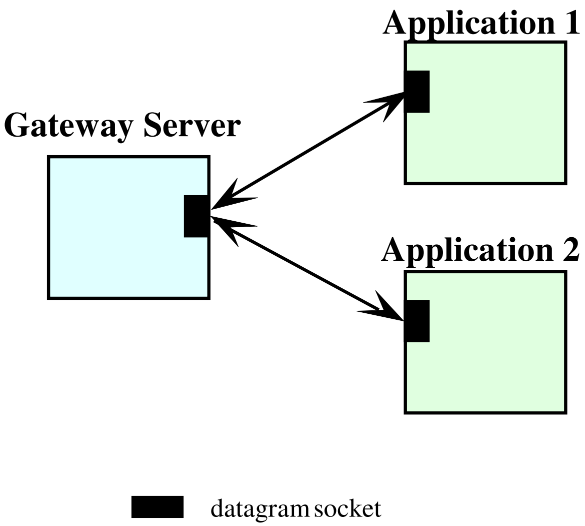
What is a Port? A Port Number?

- Port numbers are used to identify services on a host
- Port numbers can be
  - well-known (port 0-1023)
  - dynamic or private (port 1024-65535)
- Servers/daemons usually use well-known ports
  - any client can identify the server/service
  - HTTP = 80, FTP = 21, Telnet = 23, ...
  - /etc/service defines well-known ports
- Clients usually use dynamic ports
  - assigned by the kernel at run time



Connectionless sockets

With connectionless sockets, it is possible for multiple processes to simultaneously send datagrams to the same socket established by a receiving process.



Creating a Socket

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

- `family` specifies the protocol family (**`AF_INET`** for Internet, **`PF_INET`** for TCP/IP).
- `type` specifies the type of service (**`SOCK_STREAM`**, **`SOCK_DGRAM`**).
- `protocol` specifies the specific protocol (usually 0, which means *the default*).

`socket()`

- The **`socket()`** system call returns a socket descriptor (small integer) or -1 on error.

- **`socket()`** allocates resources needed for a communication endpoint - but it does not deal with endpoint addressing.

Generic socket addresses

```
struct sockaddr {
   uint8_t      sa_len;
   sa_family_t  sa_family;
   char         sa_data[14];
};
```

- **`sa_family`** specifies the address type.
- **`sa_data`** specifies the address value.

struct sockaddr_in (IPv4)

```
struct sockaddr_in {
    uint8_t           sin_len;
    sa_family_t       sin_family;
    in_port_t         sin_port;
    struct in_addr    sin_addr;
    char              sin_zero[8];
  };
```

A special kind of sockaddr structure

struct in_addr

```
struct in_addr {
   in_addr_t        s_addr;
};
```

`in_addr` just provides a name for the ‘C’ type associated with IP addresses.

Network Byte Order

- All values stored in a **`sockaddr_in`** must be in network byte order.
  - **`sin_port`** a TCP/IP port number.
  - **`sin_addr`** an IP address.

Byte Ordering

- Big Endian
  - Sun Solaris, PowerPC, ...
- Little Endian
  - i386, alpha, ...
- Network byte order = Big Endian

128	2	194	95
95	194	2	128
`c[0]`	`c[1]`	`c[2]`	`c[3]`

Assigning an address to a socket

- The **`bind()`** system call is used to assign an address to an existing socket.

```
int bind( int sockfd,
          const struct sockaddr *myaddr,
          int addrlen);
```

- **`bind`** returns 0 if successful or -1 on error.

bind()

- calling **bind()** assigns the address specified by the **sockaddr** structure to the socket descriptor.
- You can give **bind()** a **sockaddr_in** structure:

```
bind( mysock,
      (struct sockaddr*) &myaddr,
      sizeof(myaddr) );
```

Uses for **`bind()`**

- There are a number of uses for **`bind()`**:
  - Server would like to bind to a well known address (port number).
  - Client can bind to a specific port.
  - Client can ask the O.S. to assign *any available* port number.

What is my IP address ?

- How can you find out what your IP address is so you can tell **`bind()`** ?
- There is no realistic way for you to know the right IP address to give bind() - what if the computer has multiple network interfaces?

- specify the IP address as: **`INADDR_ANY,`** this tells the OS to take care of things.

Other socket system calls

- Connection-oriented (TCP)
  - `connect()`
  - `listen()`
  - `accept()`
  - `read()`
  - `write()`
  - `close()`

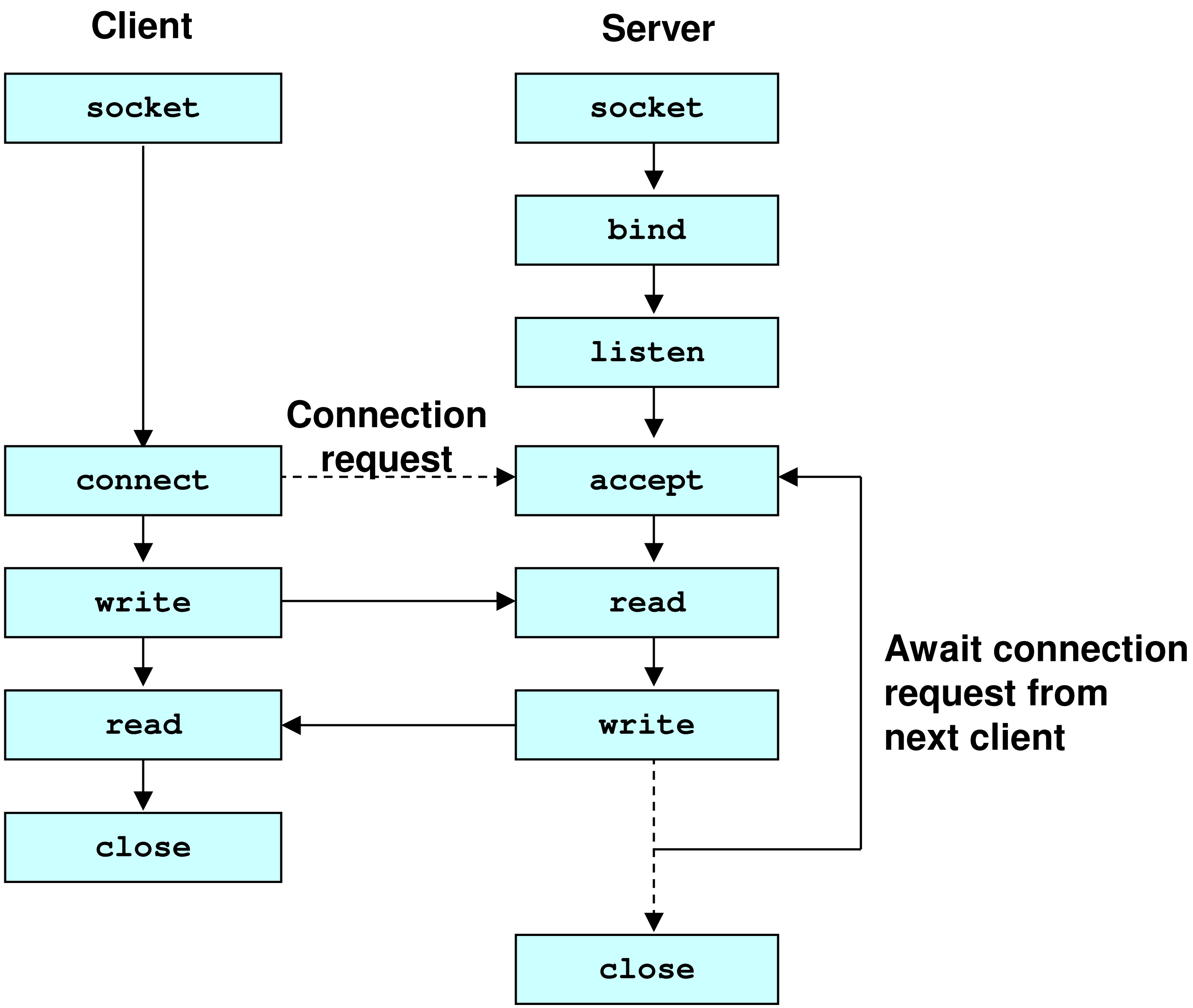
- Connectionless (UDP)
  - `connect()*`
  - `send()`
  - `recv()`
  - `sendto()`
  - `recvfrom()`

`* –optional but sometimes recommended`

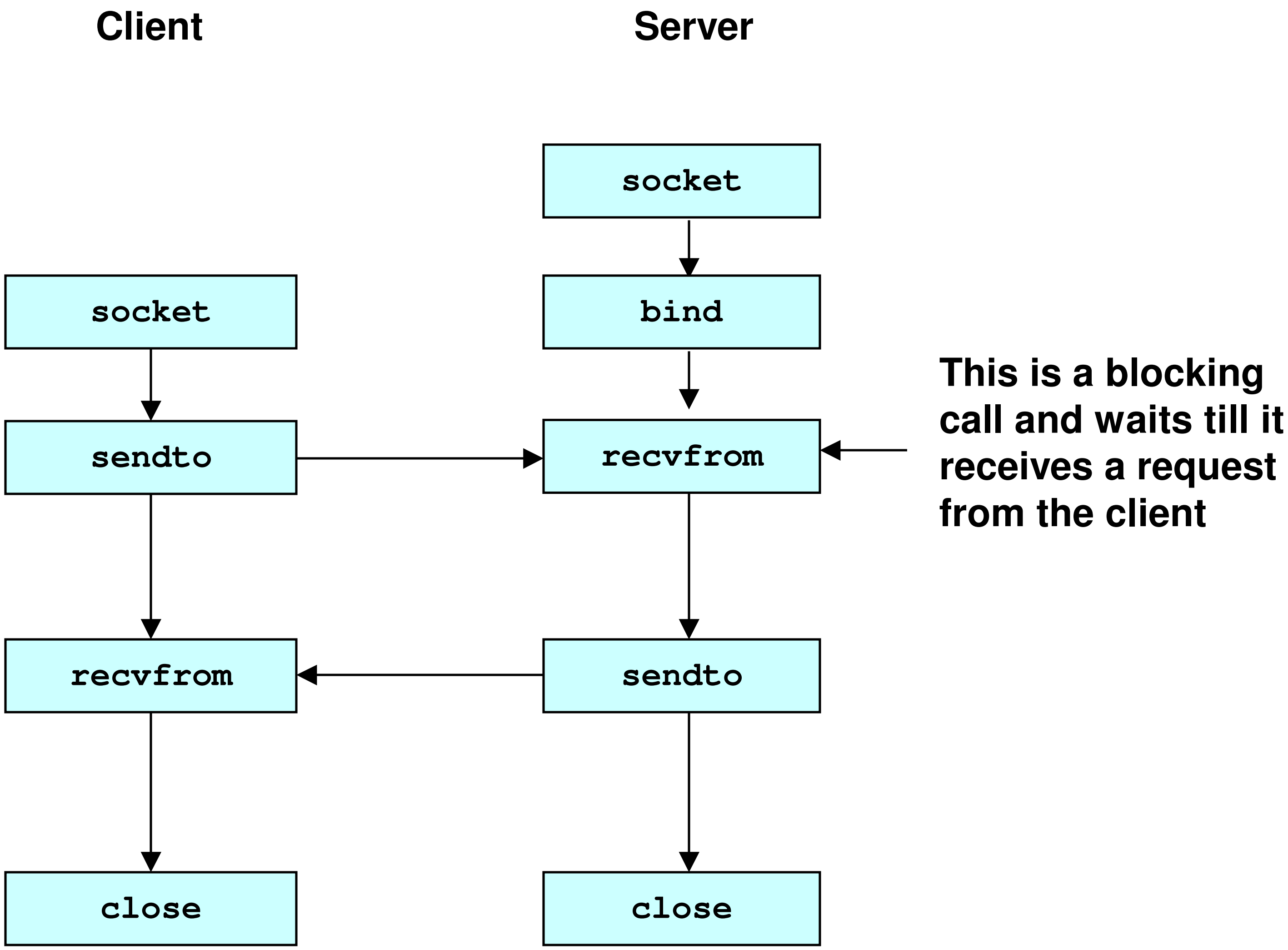
Methods :

- socket()
  - Creates a new socket and returns its descriptor
- bind()
  - Associates a socket with a port and address
- connect()
  - Establish queue for connection requests
- listen()
  - Accepts a connection request
- accept()
  - Initiates a connection to a remote host
- recv()
  - Receive data from a socket descriptor
- send()
  - Sends data to a socket descriptor

Socket programming *with TCP*



Socket programming *with UDP*



This is a blocking call and waits till it receives a request from the client

Example: C client (UDP)

```
/* UDP client in the internet domain */
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>

void error(char *);
int main(int argc, char *argv[])
{
   int sock, length, n;
   //socket structures
   struct sockaddr_in server, client;
   //hostent datastructure
   struct hostent *hp;
   char buffer[256];

   if (argc != 3) { printf("Usage: server port\n");
                    exit(1);
   }
```

```
//specifies that it is a datagram socket
//and the socket belongs to the INTERNET family
sock= socket(AF_INET, SOCK_DGRAM, 0);
if (sock < 0) error("socket");
//We initialize the individual fields of the sockaddr_in structure
//to fill sin_family which takes AF_INET as the value
server.sin_family = AF_INET;
//returns the hostname in the form of a hostent structure
hp = gethostbyname(argv[1]);
if (hp==0) error("Unknown host");

//The below function can also be replaced with memcopy
//but please never use strcpy() it wont work!
bcopy((char *)hp->h_addr,
    (char *)&server.sin_addr,
     hp->h_length);
//We initialize the individual fields of the sockaddr_in structure
//to fill sin_port which takes the port number as the value
//which was given as a command line parameter, remember to convert
//this value into host to network byte order, it is very important!
server.sin_port = htons(atoi(argv[2]));
length=sizeof(struct sockaddr_in);
printf("Please enter the message: ");
//This initializes the buffer with 0, we can also use memset as a replacement function
```

```
    bzero(buffer,256);
    //reads the value from the keyboard, stdin = keyboard
    fgets(buffer,255,stdin);
    //sends the buffer, to the server, the fourth parameter is by default zero.
    n=sendto(sock,buffer,
             strlen(buffer),0,&server,length);
    if (n < 0) error("Sendto");
    //receives the packet from the server which is stored in the buffer
    n = recvfrom(sock,buffer,256,0,&client, &length);
    if (n < 0) error("recvfrom");
    write(1,"Got an ack: ",12);
    write(1,buffer,n);
    //closes the socket descriptor
    close(sock);
}

void error(char *msg)
{
     perror(msg);
     exit(0);
}
```

Example: C server (UDP)

```
/* Creates a datagram server.  The port number is passed as an argument.  This
   server runs forever */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>

void error(char *msg)
{
    perror(msg);
    exit(0);
}

int main(int argc, char *argv[])
{
   int sock, length, clientlen, n;
   ///Declare the sockaddr_in structures for the client and the server
   struct sockaddr_in server;
   struct sockaddr_in client;
   char buf[1024];
```

```
if (argc < 2) {
   fprintf(stderr, "ERROR, no port provided\n");
   exit(0);
}
///The socket call which returns a file descriptor
sock=socket(AF_INET, SOCK_DGRAM, 0);
if (sock < 0) error("Opening socket");
length = sizeof(server);
///Initializes the server socket structure to zero, as a replacement we can also
///use memset
bzero(&server,length);
///We initialize the values for all the individual fields of the server socket
///structure remember to make use of the INADDR_ANY to assign the
///sin_addr.s_addr field and please convert the port number obtained from the
///command line to network byte order
server.sin_family=AF_INET;
server.sin_addr.s_addr=INADDR_ANY;
server.sin_port=htons(atoi(argv[1]));
///bind system call
if (bind(sock,(struct sockaddr *)&server,length)<0)
    error("binding");
clientlen = sizeof(struct sockaddr_in);
```

```
while (1) {
    ///ready to receive a packet from the client, the fourth parameter is by
    ///default zero
    n = recvfrom(sock,buf,1024,0,(struct sockaddr *)&client,&clientlen);
    if (n < 0) error("recvfrom");
    //writes output to the screen
    write(1,"Received a datagram: ",21);
    write(1,buf,n); //writes output to the screen
    ///sends a packet to the client acknowledging it
    n = sendto(sock,"Got your message\n",17,
               0,(struct sockaddr *)&client,clientlen);
    if (n  < 0) error("sendto");
  }
  ///closes the file descriptor
  close(sock);
}
```

How to use Compile/Make ?

```
CC      =       gcc

all: udpserver udpclient

udpclient: udpclient.c
        $(CC) -o udpclient udpclient.c –lnsl -<other compiler options>

udpserver: udpserver.c
        $(CC) -o udpserver udpserver.c –lnsl -<other compiler options>

clean:
        rm udpserver udpclient
```

Usage ➔ make –f file_name <all> / clean

Suggestions

- Make sure to #include the header files that define used functions
- Check man-pages and course web-site for additional info
- Sometimes, a “rough” exit from a program (e.g., ctrl-c) does not properly free up a port
- Eventually (after a few minutes), the port will be freed
- To reduce the likelihood of this problem, include the following code:

```
#include <signal.h>
void cleanExit(){exit(0);}
```

  - in socket code:

```
signal(SIGTERM, cleanExit);
signal(SIGINT, cleanExit);
```

And, please keep backing up your files periodically!

Resources

LINUX WORKSTATIONS ARE AVAILABLE AT THE UNIVERSITY COMPUTING LABS IN AT OR JCMB

For More Information

- Unix Man Pages
- Douglas Comer, "Computer Networks and Internets (4/e)", Pearson Education, 2004
- W. Richard Stevens, “Unix Network Programming: Networking APIs: Sockets and XTI”, Volume 1, Second Edition, Prentice Hall, 1998.
  - THE network programming bible.

For More Information

The C Programming Language by Brian Kernighan and Dennis Ritchie,
<http://cm.bell-labs.com/cm/cs/cbook/> .

C for Java programmers
http://www.cs.cornell.edu/courses/cs414/2001SP/tutorials/cforjava.htm

For C programming FAQ s check
<http://www.eskimo.com/~scs/C-faq/top.html>

Web site which lists the differences between Java and C
<http://www.comp.lancs.ac.uk/computing/users/ss/java2c/diffs.html>

Some of these pointers to C are from Prof. Nigel Topham.