Computer Networks

Lab 0

Yvan Petillot

What You Will Do In This Lab.

The purpose of this lab is to help you become familiar with the UNIX/LINUX on the lab network. This means being able to do editing, compiling, etc. of simple programs. These programs will be written in C, so you may have some more learning/reviewing ahead of you.

You have one task before you:

- Using your favorite editor, type in (or paste) the program given later in this document. Compile it and run it and show that it produces communication between two instances of the program.
- You will know you are done when you have demonstrated to me that your program works.

Where To Get Documentation

There are many sources of information to help you with this lab. Here are some of those sources:

Learning C:

Learning GDB – how to debug:

Learning UNIX:

All of these skills can be acquired (I hope) from the documentation available on my webpage

If you don't like these documents, there are plenty of other ones out on the web. Go wild!

Where To Get Documentation

For information in more detail than is available off of my home page, see the following links:

GNU Debugger – remote copy is at:

http://www.gnu.org/manual/gdb-4.17/html_mono/gdb.html

GCC – Compiler: - remote copy is at:

http://gcc.gnu.org/onlinedocs/gcc-3.0.1/gcc.html

Detour – a gdb quickstart

Here's all you need to know to get started using gdb:

Start the debugger with "gdb program_name"
List the lines with "l"
Set a breakpoint with "b <line_number>"
Print the value of a variable with "p <variable_name>"
To run the first time, say "run <optional arguments>
To continue from a breakpoint, use "c"
To single step, use "s"
To stop the debugger, use "q"

Project 0:

Here's the code for this lab. We will be going through it so that you understand what it does.

Type it in using your favorite editor. In this example, the source file is named proj0.c

Get a port number from me. This way you won't all be colliding with each other.

To compile this code, say "gcc –g proj0.c –o proj0"

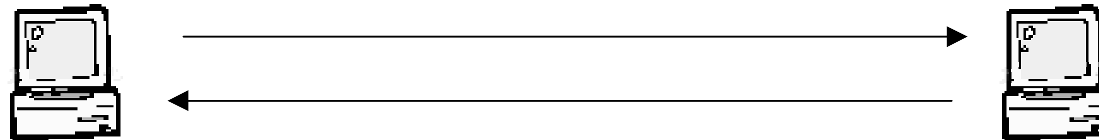
This will produce an output file that you can run.

As the code explains, there are several modes of execution.

proj0 –s&	**creates a process running the code as a server**
proj0 –c	**creates a process running the code as a client**

Computer Chat

- **How do we make computers talk?**



- **How are they interconnected?**

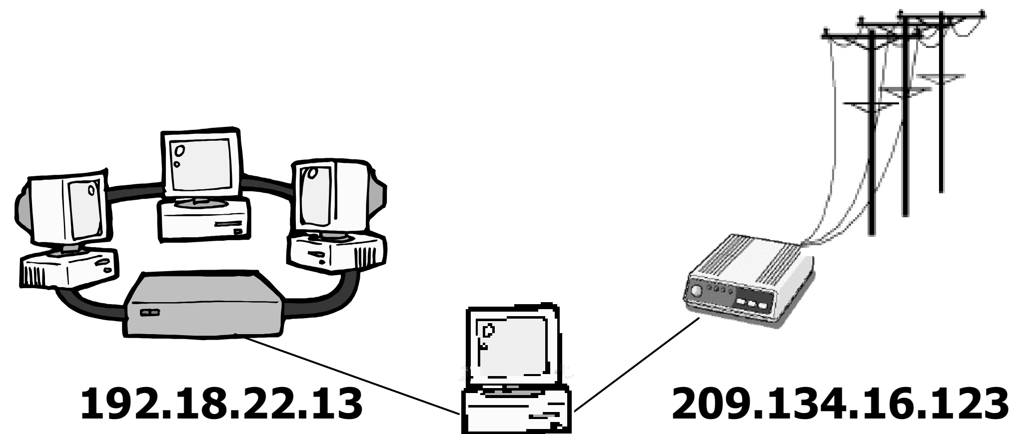
Internet Protocol (IP)

Internet Protocol (IP)

- **Datagram (packet) protocol**
- **Best-effort service**
  - Loss
  - Reordering
  - Duplication
  - Delay
- **Host-to-host delivery**

IP Address

- **32-bit identifier**
- **Dotted-quad: 134.111.10.43**
- **www.clarku.edu** -> **140.232.1.19**
- **Identifies a host interface (not a host)**



Transport Protocols

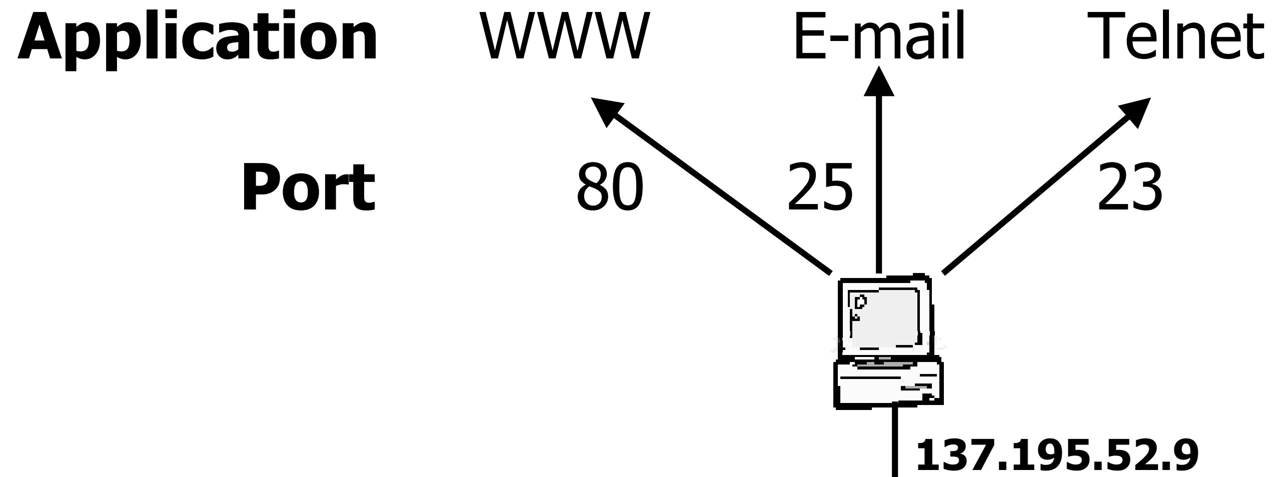
Best-effort not sufficient!

- **Add services on top of IP**
- **User Datagram Protocol (UDP)**
  - Data checksum
  - Best-effort
- **Transmission Control Protocol (TCP)**
  - Data checksum
  - Reliable byte-stream delivery
  - Flow and congestion control

Ports

Identifying the ultimate destination

- **IP addresses identify hosts**
- **Host has many applications**
- **Ports (16-bit identifier)**



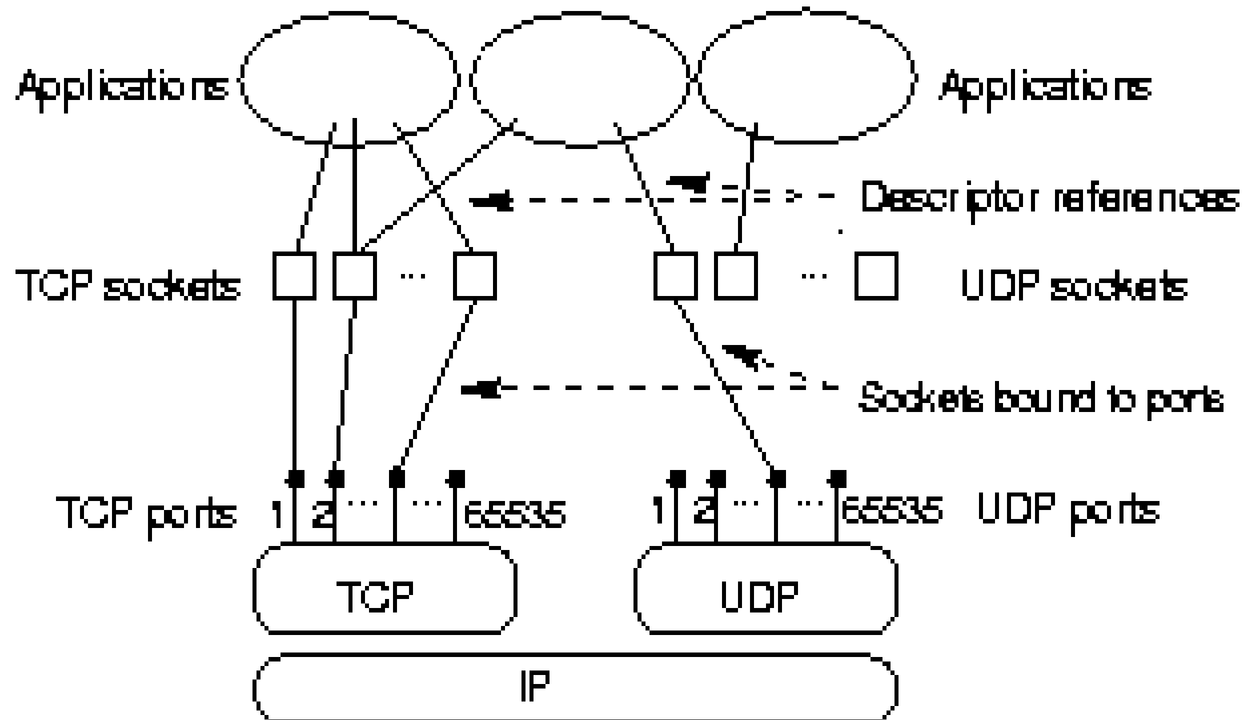
Socket

How does one speak TCP/IP?

- **Sockets provides interface to TCP/IP**
- **Generic interface for many protocols**

Sockets

- Identified by protocol and local/remote address/port
- Applications may refer to many sockets
- Sockets accessed by many applications



TCP/IP Sockets

- **mySock = socket(family, type, protocol);**
- **TCP/IP-specific sockets**

	Family	Type	Protocol
TCP	**PF_INET**	**SOCK_STREAM**	**IPPROTO_TCP**
UDP		**SOCK_DGRAM**	**IPPROTO_UDP**

- **Socket reference**
  - File (socket) descriptor in UNIX
  - Socket handle in WinSock

Specifying Addresses

Generic

- struct sockaddr

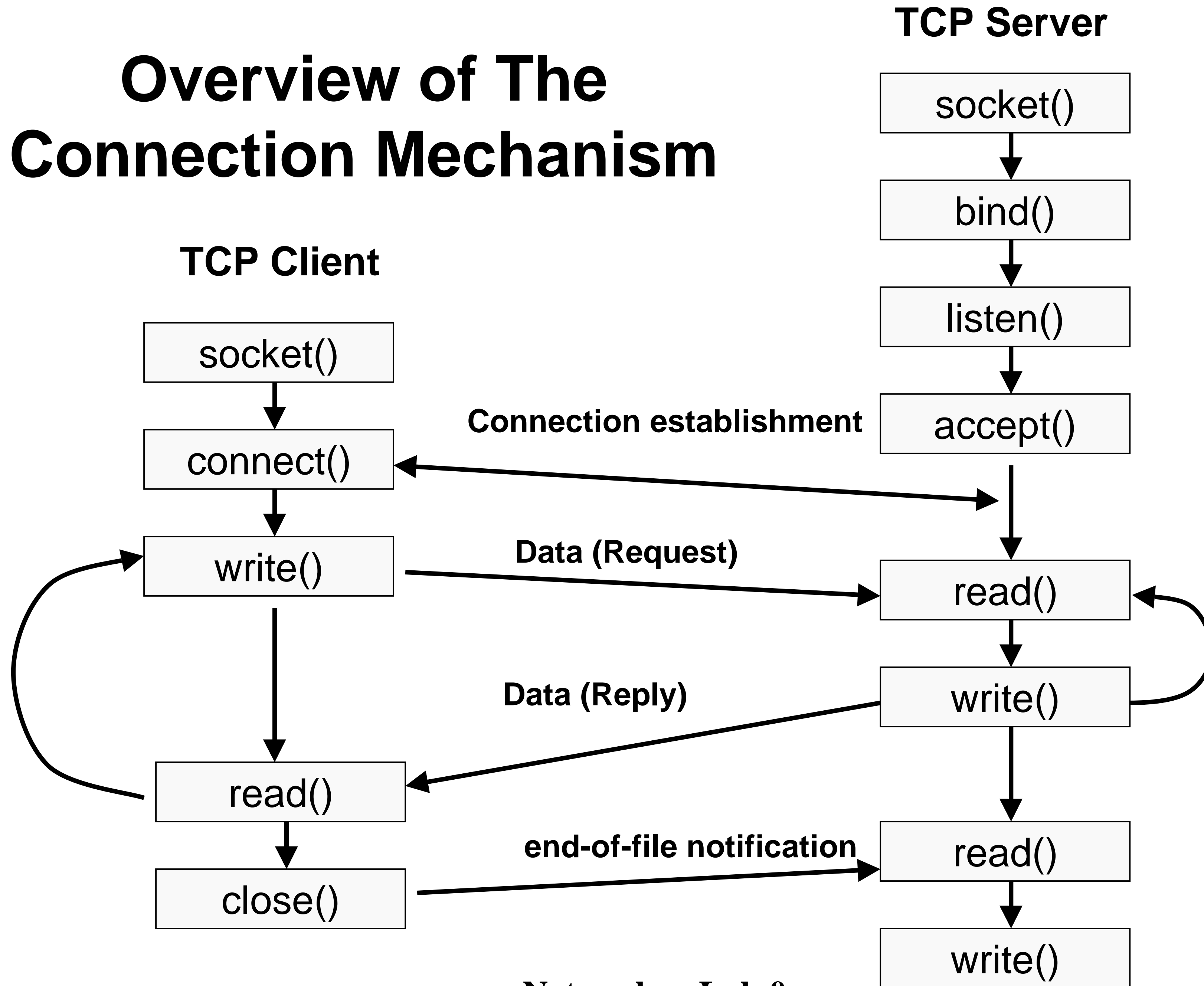
```
{
        unsigned short sa_family;     /* Address family (e.g., AF_INET) */
        char sa_data[14];             /* Protocol-specific address information */
};
```

IP Specific

- struct sockaddr_in

```
{
        unsigned short sin_family;    /* Internet protocol (AF_INET) */
        unsigned short sin_port;      /* Port (16-bits) */
        struct in_addr sin_addr;      /* Internet address (32-bits) */
        char sin_zero[8];             /* Not used */
};
struct in_addr
{
         unsigned long s_addr;        /* Internet address (32-bits) */
};
```

Overview of The Connection Mechanism



server.c – the code:

```
/*******************************************************************************

   server.c    Designed as a simple class example.  The program waits for
              a request.  It assumes that request is numerical.  It adds
              +1 to the input and sends it back.

   This program expects no arguments:
   server

   Yvan Petillot October 2002
*******************************************************************************/

#include       <stdlib.h>
#include       <sys/socket.h>
#include       <netinet/in.h>

#define         TRUE            1
#define         FALSE           0
#define         BUFFER_SIZE     20
#define         PORT            10000

void    SysError( char * );
```

These say to include more information from include files.

The compiler substitutes these values whenever it sees the define.

A prototype.

server.c – the code:

```
main ( int argc, char *argv[] )
    {
    long                input_value;
    int                 family = AF_INET;       /* The default for most cases */
    int                 type   = SOCK_STREAM;   /* Says it's a TCP connection */
    in_port_t           port=PORT;
    int                 result;
    struct sockaddr_in  server;
    int                 lserver = sizeof(server);
    int                 fdListen, fdConn, fd;
    char                console_buffer[BUFFER_SIZE];
    char                ip_input_buffer[BUFFER_SIZE];
    char                ip_output_buffer[BUFFER_SIZE];
```

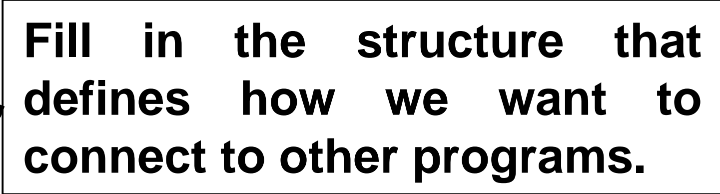
A C program always starts at main()

This section is declaring the variables.

server.c – the code:

```
if ((fd = socket (family, type, 0)) < 0)
        SysError ("Error on socket");

    server.sin_family        = family;
    server.sin_port          = port;              /* client & server see same 
port*/
    server.sin_addr.s_addr   = htonl(INADDR_ANY); /* the kernel assigns the IP 
addr*/
```

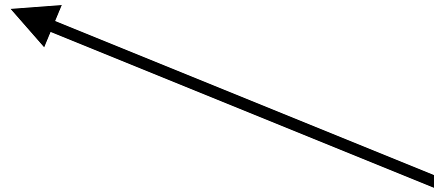


server.c – the code:

```
if (bind (fd, (struct sockaddr *)&server, sizeof(server) ) == -1)
    SysError ("Error on bind");

if (listen (fd, SOMAXCONN) == -1)   /* set up for listening */
    SysError ("Error on listen");

fdListen = fd;
```



Server program here. Then do the bind and listen.

server.c – the code:

```
        while( TRUE )
            {
            if ((fdConn = accept (fdListen, (struct sockaddr *)&server,
&lserver )) <0)
                SysError ("Error on accept");

            bzero( ip_input_buffer, sizeof( ip_input_buffer ));

            while ( recv( fdConn, ip_input_buffer, BUFFER_SIZE - 2, 0 ) > 0 )
                {
                input_value = atoi( ip_input_buffer );
                input_value = input_value + 1;

                bzero( ip_output_buffer, sizeof( ip_output_buffer ));
                sprintf( ip_output_buffer, "%d", input_value );

                if ( send( fdConn, ip_output_buffer,
                          strlen(ip_output_buffer) +1, 0) <= 0 )
                    SysError( "Error on send" );

            }                           /* End of while recv is successful */
            close (fdConn);
        }                               /* End of while TRUE    */
    }                                   /* End of server case   */
```

Repeat forever

recv from client

Calculate the new value

Send back to the client

recv will keep on working until the client closes the connection. The recv will then take an error in that case.

client.c – the code:

```
main ( int argc, char *argv[] )
    {
    long                  input_value;
    int                   family = AF_INET;       /* The default for most cases */
    int                   type   = SOCK_STREAM;   /* Says it's a TCP connection */
    in_port_t             port=PORT;
    int                   result;
    struct sockaddr_in client;
    struct sockaddr_in server;
    int                   lclient = sizeof(client);
    int                   fdListen, fdConn, fd;
    char                  console_buffer[BUFFER_SIZE];
    char                  ip_input_buffer[BUFFER_SIZE];
    char                  ip_output_buffer[BUFFER_SIZE];
    struct hostent        *host;
```

A C program always starts at main()

This section is declaring the variables.

client.c – the code:

```
 /* Checks first we have the correct number of arguments */
  if ( argc < 2 ) {
    printf( "The program expects arguments\n" );
    printf( "tcp_client <hostname>\n" );
    exit(0);
  }
```

Check Number of arguments of program

```
 if ((host = gethostbyname(argv[1])) == (struct hostent *)NULL)
    {
      SysError("Error on gethostbyname");
      return(-1);
    }
```

Find Host IP address based on Name using DNS server

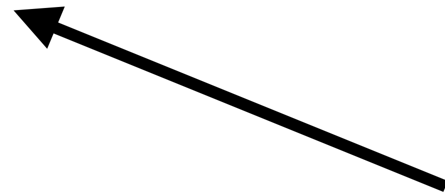
```
if ((fd = socket (family, type, 0)) < 0)
        SysError ("Error on socket");

    client.sin_family        = family;
    client.sin_port          = port;             /* client & server see same
port*/
    client.sin_addr.s_addr   = htonl(INADDR_ANY); /* the kernel assigns the IP
addr*/
```

Fill in the structure that defines how we want to connect to other programs.

Client.c – the code:

```
 /* Fills server socket structure with correct fields */
  server.sin_family        = family;
  server.sin_port          = port;              /* client & server see same
port*/
  memcpy((char *)&server.sin_addr, (char *)host->h_addr, host->h_length);
```

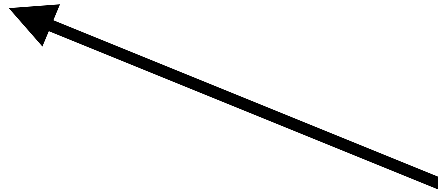


Initialise server characteristics.

client.c – the code:

This is for a client.

```
if (connect(fd, (struct sockaddr *)&server, sizeof(server)  ) )
            SysError ("Error on connect");
```



So the first thing a client does is a connect to the server.

client.c – the code:

```
while( TRUE )
    {
    printf( "> " );
    scanf( "%s", console_buffer );
    if ( atoi( console_buffer ) == -1 )
        {
        printf( "We Have Successfully Finished.\n" );
        exit(0);
    }
    bzero( ip_output_buffer, sizeof( ip_output_buffer ));
    strcpy( ip_output_buffer, console_buffer );

    if ( send( fd, ip_output_buffer,
          strlen(ip_output_buffer) + 1, 0 ) <= 0 )
        SysError( "Error on send" );

    bzero( ip_input_buffer, sizeof(ip_input_buffer) );

    if ( recv( fd, ip_input_buffer,
                sizeof(ip_input_buffer) - 2, 0 ) <= 0 )
        SysError( "Error on recv" );
    printf( "%s\n", ip_input_buffer );
}                                   /* End of while TRUE    */
}                                   /* End of client case   */
}                                   /* End of main          */
```

Loop here forever.

Get data from console

send data to server

recv data from server

End of main

server.c / client.c – the code:

```
void    SysError( char *string )
    {
    printf( "Error found:  String given is --> %s\n", string );
    exit(0);
}
```