

Server-Side Sockets

- Creating the Listening Socket
- Setting the Socket Options (not required)
- Binding to a Port
- Listening
- Accepting Connections
- Compiling Socket Applications

Step 1: Creating the Listening Socket

The server requires 2 sockets: one for listening for incoming connections, and one to handle the client connections. Only the listener socket needs to be created initially:

```
typedef int socket_t;    // Defined for clarity

int main(int argc, char *argv[])
{
    socket_t listener;

    Error(listener = socket(PF_INET, SOCK_STREAM, 0));

    /* Some more code */
}
```

Step 1.5: Setting Socket Options

By default, a particular socket can only be opened once every 1-2 minutes. I suppose this is a security feature, but it can be very annoying when testing a server application. Here's how to correct the issue:

```
int TRUE = 1; // I know. This is annoying, but
              // it's the only way to pass TRUE
              // setsockopt

/* Socket Initializer Code (previous slide) */

Error(setsockopt(listener,
                 SOL_SOCKET,
                 SO_REUSEADDR,
                 &TRUE, sizeof(TRUE)));

/* More Code */
```

Step 2: Binding to a Port

To bind to a port, we have to let the computer know:

- 1) That any computer is allowed to connect to the server
- 2) The port number that we want to connect to

```
#define PORT 12345

struct sockaddr_in listaddr;

/* Previous Code */

memset(&listaddr, 0, sizeof(listaddr)); // Clear listaddr
listaddr.sin_family      = PF_INET;           // Required for
                                           // polymorphism
listaddr.sin_addr.s_addr = htonl(INADDR_ANY); // Accept from any
                                           // computer
listaddr.sin_port        = htons(PORT);       // Use PORT

Error(bind(listener,
           (struct sock_addr*) &listaddr,
           sizeof(listaddr)));
```

What are htonl and htons?

htonl and htons are functions that convert the bit representation used by your computer to the standardized bit representation used by the internet. These are used to ensure that computers using Little Endian format can correctly communicate with computers using Big Endian format.

htonl : Host to Network (long integer)

htons : Host to Network (short integer)

ntohl : Network to Host (long integer)

ntohs : Network to Host (short integer)

Step 3: Creating the Listening Queue

This is the easy part. To create a queue, you just need to specify the socket and the size of the queue. The size signifies a “waiting list” for connections that need to be passed to the client connection socket. If too many clients are waiting for a connection, then some of them will be denied access.

```
#define QUEUE_SIZE 5

/* Previous code */

Error(listen(listener, QUEUE_SIZE));

/* More code */
```

Step 4: Accepting Connections

Accepting a single connection is pretty straightforward. Simply specify the listening socket you want to use and provide a `sockaddr_in` structure to collect information about the client connection.

```
socket_t          conn;          // The client connection socket
struct sockaddr_in conn_addr;    // Will contain client connection info
socklen_t         len;          // The length of conn_addr;

/* Previous Code */

len = sizeof(conn_addr);        // accept needs the size of conn_addr
Error(conn = accept(listener,
                      (struct sockaddr_in*) &conn_addr,
                      &len));

/* More Code */
```

Creating a Persistent Server

Creating a persistent server is a little more complicated, but still not too bad:

```
socket_t          conn;          // The client connection socket
struct sockaddr_in conn_addr;    // Will contain client connection info
socklen_t         len;          // The length of conn_addr;

while(1)
{
    len = sizeof(conn_addr);
    Error(conn = accept(listener,
                            (struct sockaddr_in*) &connaddr,
                            &len));
    if( Error(fork()) == 0) // Child process
    {
        close(listener);
        break;
    }
    else
        close(conn);
}
```

Compiling Socket Applications

Sockets are not a part of the standard Unix C library. You will need to explicitly link the Berkley Socket Library to your program:

```
gcc -o server -lsocket server.c
```