Systems Software

Week 7: Sockets

Notes By: Jonathan McCarthy

Overview

- IPC Summary
- Past Learning: Socket Revision
- Introduction to Sockets
- Socket Documentation
- Server Setup
- Client Setup
- In-Class Example
- Summary

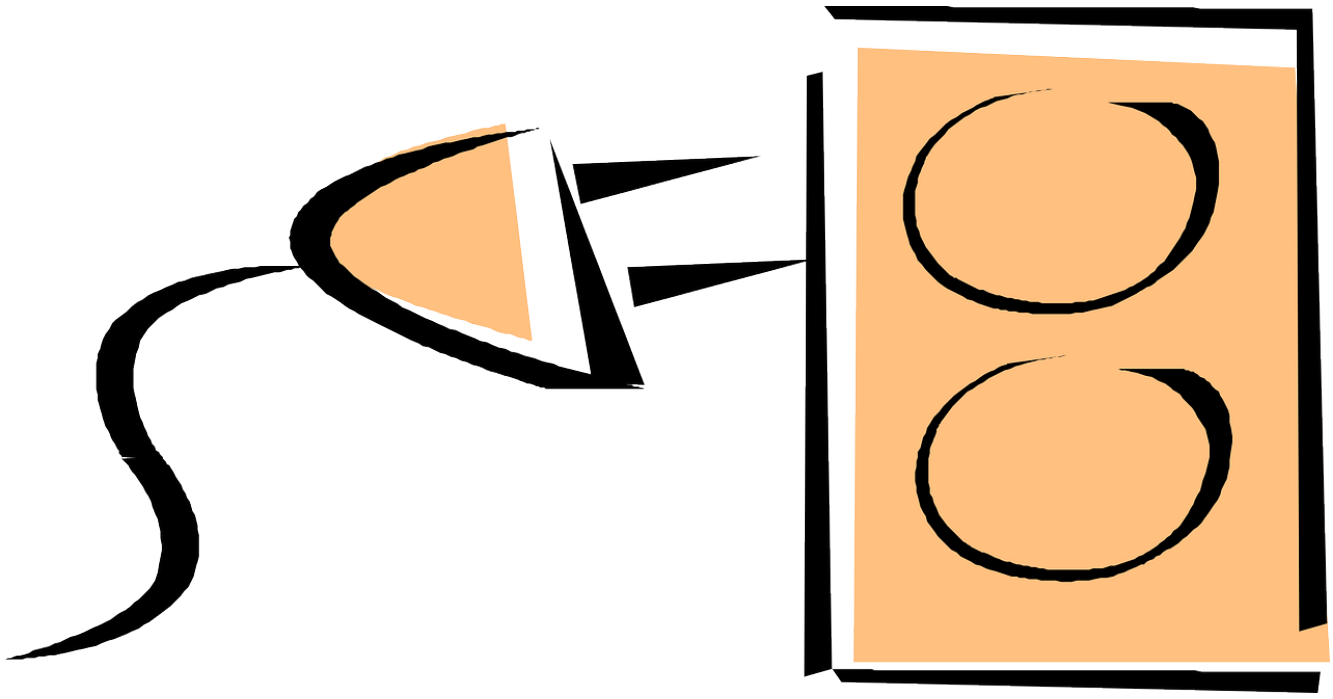
Revision – What we covered last week

- IPC – Communication between processes on the same machine.
  - Unnamed Pipes
  - Named Pipes
  - Message Queue

- What happened if the processes are not on the same machine??

Socket Programming

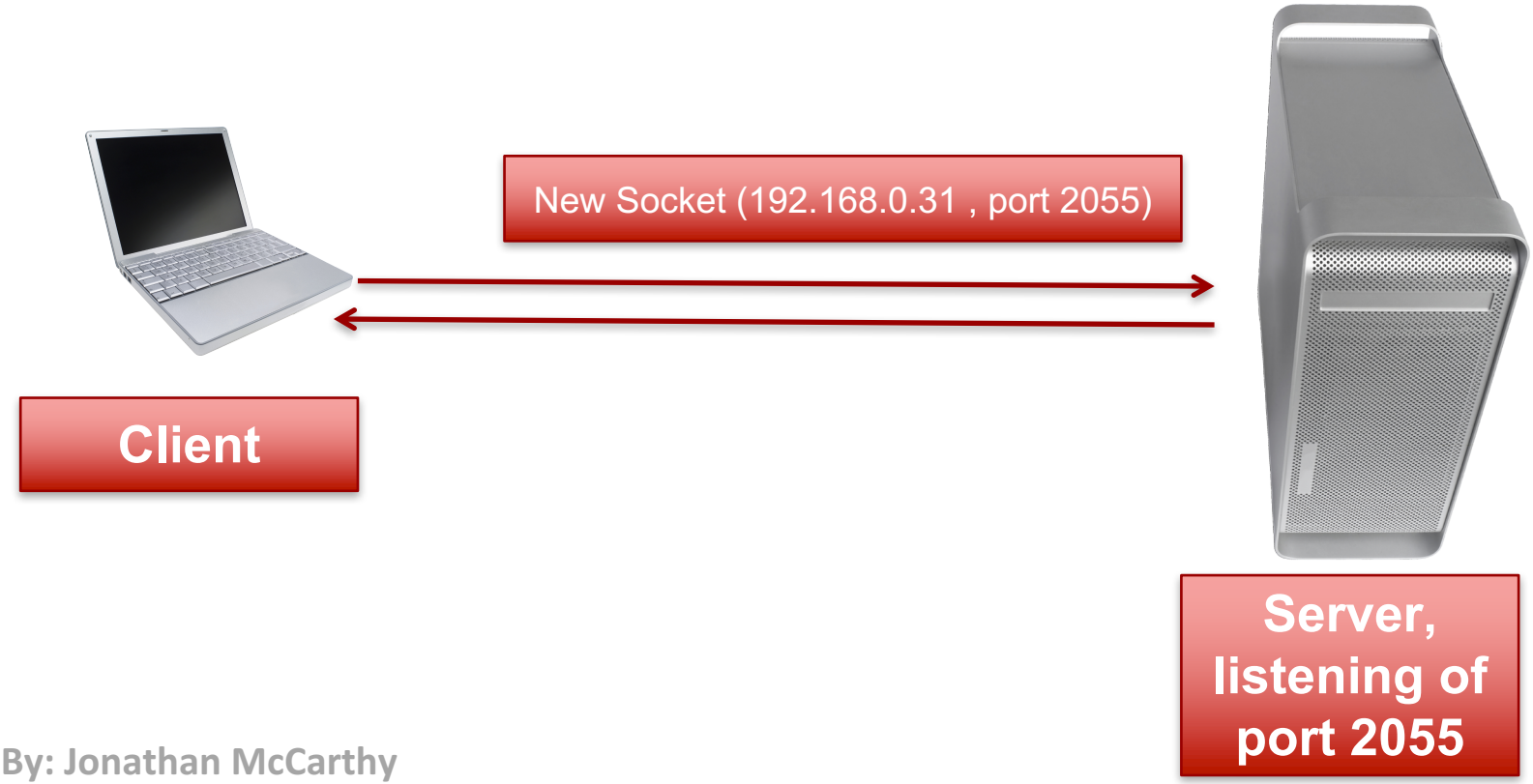
Introduction to Socket Programming using C in a Linux Environment



Notes By: Jonathan McCarthy

Revision:: Sockets

- Two machines on a network can communicate over a network using a Hostname/IP address and a given port address.

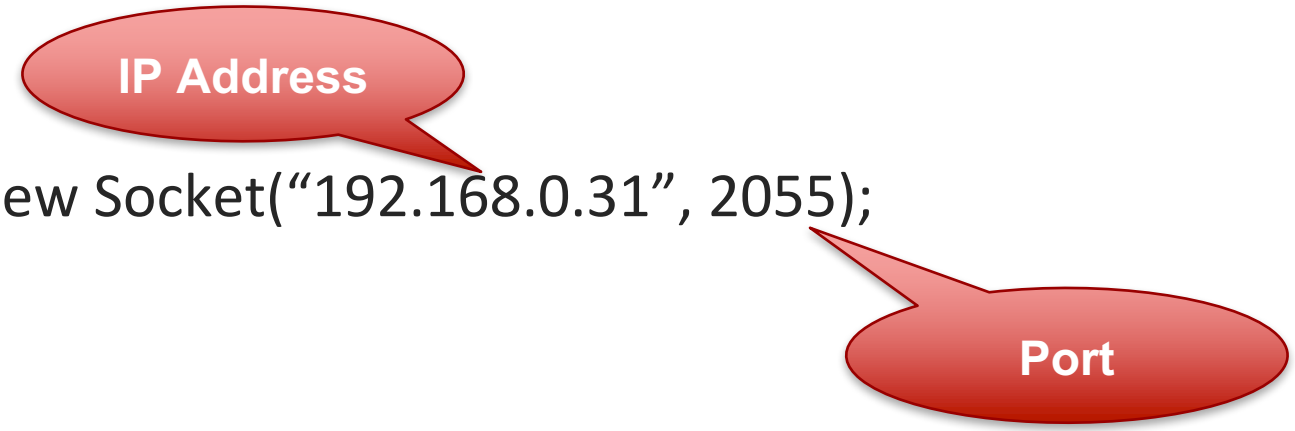


Revision:: Sockets

- Sockets will provide for communication between a Client program and a server.
- What services can the server offer?:
  - Processing (business logic)
  - Database functionality
  - Etc…
- To ensure reliable communication, TCP will be used to establish a connection between the client and the server.

Revision:: What is a Socket

- A socket is an end-point that provides for two way communication between a client and a server.
- The program on the server will assigned a port. This will allow a client to connect to the socket on the server using the server IP address and the assigned port number.
- Syntax:
- Socket s = new Socket("192.168.0.31", 2055);



Revision:: Client Server Socket Communication

- With a Socket created, both the client and the server will communicate with the help of the socket.
- Streams will be used both on the client and the server to facilitate two way communication.

Revision:: Sockets - Sequence

- **Step 1:** Server creates a Socket and starts listening for a client to connect.
- **Step 2:** Client makes a socket connection to the server.
- **Step 3:** Sever decides whether to allow the connection (true or false)
- **Step 4:** Client sets Input and Output streams for the socket. This will be used to send/receive data.
- **Step 5:** Client usually closes the socket connection.

Sockets in Linux

- The socket network Interface in Linux can be used to facilitate communication between different processes, no matter where they reside.
- Socket communication draws on the strength of network communication protocols.
- The examples we cover in this session are using TCP/IP. (It is possible to use other network protocols for socket communication, but TCP/IP is the most widely used standard)

Socket Descriptors

- A socket is an abstraction of a communication endpoint.
- In the Linux environment a socket descriptor is used to access sockets.
- Socket descriptors are implemented as file descriptors in the UNIX System.
- Most of the functions used for reading and writing work with the socket descriptor.

Socket Function

jmccarthy@debianJMC2017: /dev/mqueue

File Edit View Search Terminal Help

```
SOCKET(2)                  Linux Programmer's Manual                 SOCKET(2)

NAME
       socket - create an endpoint for communication

SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

       int socket(int domain, int type, int protocol);

DESCRIPTION
       socket()  creates  an endpoint for communication and returns a descrip-
       tor.

       The domain argument specifies a communication domain; this selects  the
       protocol  family  which will be used for communication.  These families
       are  defined  in  <sys/socket.h>.   The  currently  understood  formats
       include:
```

Socket Function

- #include <sys/socket.h>
- int socket(int domain, int type, int protocol);
- When a socket is successfully created it will return the file (socket) descriptor.
- The file descriptor will facilitate the communication between the socket endpoints.

Domain Argument

- When using the socket function to create a file descriptor the domain argument must be provided to determines the nature of the communication.

The domain argument specifies a communication domain; this selects the protocol family which will be used for communication. These families are defined in <sys/socket.h>. The currently understood formats include:

Name	Purpose	Man page
AF_UNIX, AF_LOCAL	Local communication	**unix**(7)
AF_INET	IPv4 Internet protocols	**ip**(7)
AF_INET6	IPv6 Internet protocols	**ipv6**(7)
AF_IPX	IPX - Novell protocols	
AF_NETLINK	Kernel user interface device	**netlink**(7)
AF_X25	ITU-T X.25 / ISO-8208 protocol	**x25**(7)
AF_AX25	Amateur radio AX.25 protocol	
AF_ATMPVC	Access to raw ATM PVCs	
AF_APPLETALK	AppleTalk	**ddp**(7)
AF_PACKET	Low level packet interface	**packet**(7)

Type

The socket has the indicated type, which specifies the communication semantics. Currently defined types are:

SOCK_STREAM Provides sequenced, reliable, two-way, connection-based byte streams. An out-of-band data transmission mechanism may be supported.

SOCK_DGRAM Supports datagrams (connectionless, unreliable messages of a fixed maximum length).

SOCK_SEQPACKET Provides a sequenced, reliable, two-way connection-based data transmission path for datagrams of fixed maximum length; a consumer is required to read an entire packet with each input system call.

SOCK_RAW Provides raw network protocol access.

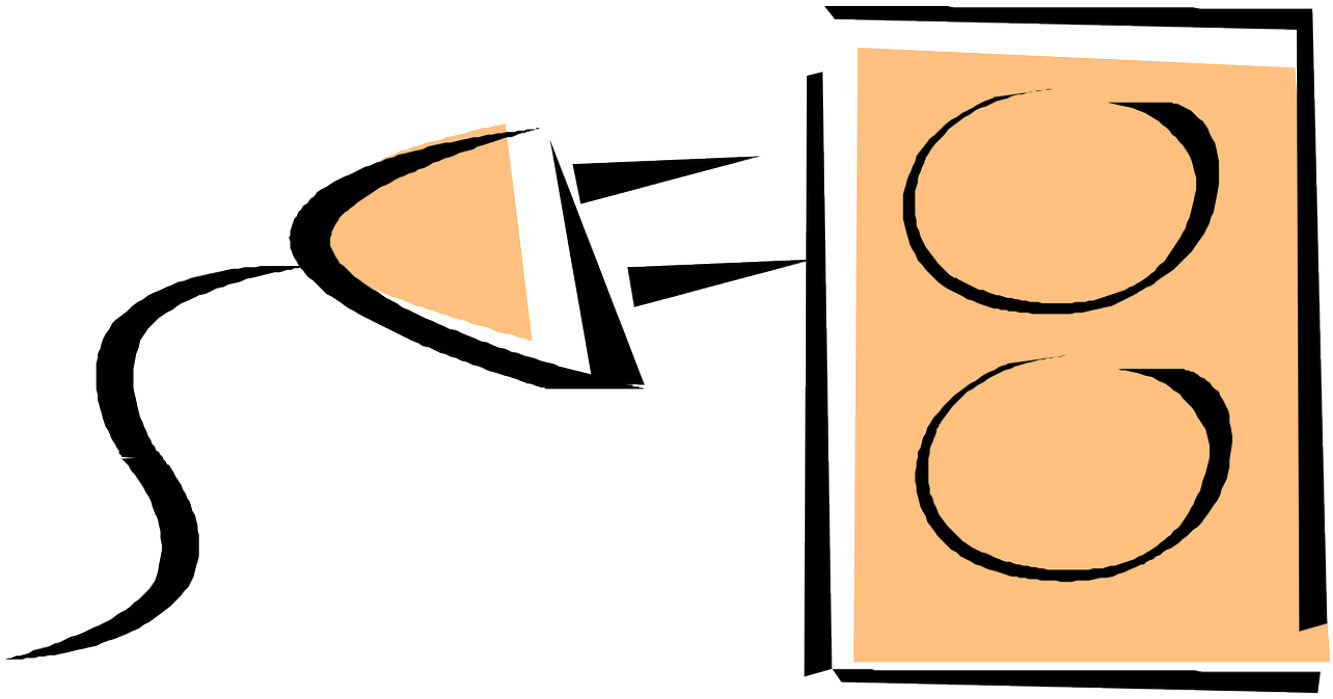
SOCK_RDM Provides a reliable datagram layer that does not guarantee ordering.

SOCK_PACKET Obsolete and should not be used in new programs; see **packet**(7).

Lifeline of a Socket Connection

- Sockets of type **SOCK_STREAM** are **full-duplex** byte streams.
- The connection must be set correctly before any data may be sent or received.
- The **connect** function is used to connect to an existing socket.
- When connected data can be send and received using **read** and **write**.
- When the processes are finished communicating, **close** function is called.

In Class Example



Sequence

- Server
  1. Set Includes
  2. Init Variables
  3. Create the Socket
  4. Init the Socket
  5. Bind Init to Socket
  6. Listen for Connections
  7. Accept Connection
  8. Read Data from Client
  9. Cleanup and end program
- Client
  1. Set Includes
  2. Init Variables
  3. Create the Socket
  4. Set Socket Variables
  5. Connect to Socket Server
  6. Communicate with Server
  7. Cleanup and end program

Server.c

Open

*server.c

~/Documents/Apps/week7/socket1

```
#include<stdio.h>       // for IO
#include<string.h>      //for strlen
#include<sys/socket.h> // for socket
#include<arpa/inet.h>  //for inet_addr
#include<unistd.h>      //for write

int main(int argc , char *argv[])
{
    |
}
```

Init Server Variables

```
int s; // socket descriptor
int cs; // Client Socket
int connSize; // Size of struct
int READSIZE;  // Size of sockaddr_in for client connection

struct sockaddr_in server , client;
char message[500];
```

Create the Socket

```
//Create socket
s = socket(AF_INET , SOCK_STREAM , 0);
if (s == -1)
{
    printf("Could not create socket");
} else {
    printf("Socket Successfully Created!!");
}
```

Init the Socket

```
// set sockaddr_in variables
server.sin_port = htons( 8081 ); // Set the prot for communication
server.sin_family = AF_INET; // Use IPV4 protocol
server.sin_addr.s_addr = INADDR_ANY;
// When INADDR_ANY is specified in the bind call, the  socket will  be bound
   to all local interfaces.
```

Bind the configuration to the socket

```
//Bind
if( bind(s,(struct sockaddr *)&server , sizeof(server)) < 0)
{
    perror("Bind issue!!");
    return 1;
} else {
    printf("Bind Complete!!");
}
```

Server set to listen for connection

```
//Listen for a conection
listen(s,3);
//Accept and incoming connection
printf("Waiting for incoming connection from Client>>");
connSize = sizeof(struct sockaddr_in);
```

Accept a Connection

```
//accept connection from an incoming client
cs = accept(s, (struct sockaddr *)&client, (socklen_t*)&connSize);
if (cs < 0)
{
    perror("Can't establish connection");
    return 1;
} else {
    printf("Connection from client accepted!!");
}
```

Read data from Client

```
while(1) {
    memset(message, 0, 500);
    //READSIZE = read(cs,message,500);
    READSIZE = recv(cs , message , 2000 , 0);
    printf("Client said: %s\n", message);
    //puts(message);
    write(cs , "What ??" , strlen("What ??"));
}
```

Clean up on client disconnect

```
if(READSIZE == 0)
{
    puts("Client disconnected");
    fflush(stdout);
}
else if(READSIZE == -1)
{
    perror("read error");
}

return 0;
```

Create the Client: client.c

Open

client.c

~/Documents/Apps/week7/socket1

```
#include<stdio.h>
#include<string.h>    //for strlen
#include<sys/socket.h>
#include<arpa/inet.h> //for inet_addr
#include<unistd.h>    //for write

int main(int argc , char *argv[])
{
```

Init the Client Variables

```
int SID;
struct sockaddr_in server;
char clientMessage[500];
char serverMessage[500];
```

Create the socket

```
//Create socket
SID = socket(AF_INET , SOCK_STREAM , 0);
if (SID == -1)
{
    printf("Error creating socket");
} {
    printf("socket created");
}
```

Set Socket Variables

```
// set sockaddr_in variables
server.sin_port = htons( 8081 ); // Port to connect on
server.sin_addr.s_addr = inet_addr("127.0.0.1"); // Server IP
server.sin_family = AF_INET; // IPV4 protocol
```

Connect to Socket Server

```
//Connect to server
if (connect(SID , (struct sockaddr *)&server , sizeof(server)) < 0)
{
    printf("connect failed. Error");
    return 1;
}

printf("Connected to server ok!!\n");
```

Communicate with Server

```
while(1)
{
    printf("\nEnter message : ");
    scanf("%s" , clientMessage);

    //Send some data
    if( send(SID , clientMessage , strlen(clientMessage) , 0) < 0)
    {
        printf("Send failed");
        return 1;
    }

    //Receive a reply from the server
    if( recv(SID , serverMessage , 500 , 0) < 0)
    {
        printf("IO error");
        break;
    }

    printf("\nServer sent: ");
    printf(serverMessage);
}

close(SID);
return 0;
```

Run the Programs

jmccarthy@debianJMC2017: /c

File Edit View Search Terminal Help

```
$pwd
/home/jmccarthy/Documents/Apps/week7/socket1
$gcc -o server server.c
$./server
```

jmccarthy@debianJMC2017: ~/Documents

File Edit View Search Terminal Help

```
$pwd
/home/jmccarthy/Documents/Apps/week7/socket1
$gcc -o client client.c
$./client
socket createdConnected to server ok!!

Enter message :
```

Summary

- Socket Programming can be used to facilitate IPC.
- The main benefits is that the processes don’t have to be on the same machine.
- Networks and Internet Protocols can be used to facilitate communication.
- Sockets offers full duple communications.

Assignment and General Questions