Computer Networks (CS425)

Instructor: Dr. Dheeraj Sanghi

Prev | Next | Index

Unix Socket Programming

Client Server Architecture

In the client server architecture, a machine(refered as client) makes a request to connect to another machine (called as server) for providing some service. The services running on the server run on known ports(application identifiers) and the client needs to know the address of the server machine and this port in order to connect to the server. On the other hand, the server does not need to know about the address or the port of the client at the time of connection initiation. The first packet which the client sends as a request to the server contains these informations about the client which are further used by the server to send any information. Client acts as the active device which makes the first move to establish the connection whereas the server passively waits for such requests from some client.

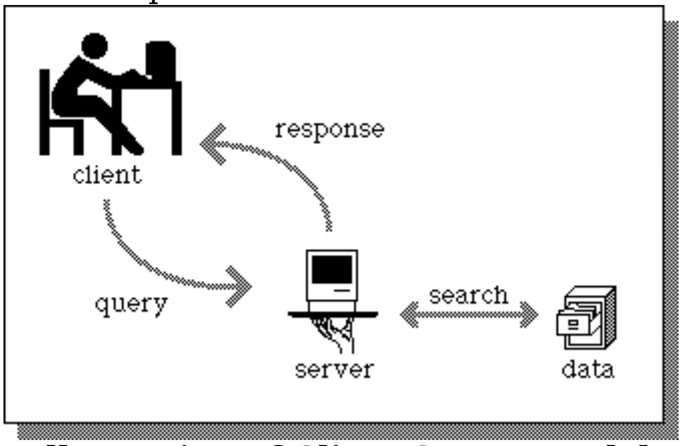

Illustration of Client Server Model

What is a Socket ?

In unix, whenever there is a need for inter process communication within the same machine, we use mechanism like signals or pipes(named or unnamed). Similarly, when we desire a communication between two applications possibly running on different machines, we need **sockets**. Sockets are treated as another entry in the unix open file table. So all the system calls which can be used for any IO in unix can be used on socket. The server and client applications use various system calls to conenct which use the basic construct called **socket**. A socket is one end of the communication channel between two applications running on different machines.

Steps followed by client to establish the connection:

1. Create a socket
2. Connect the socket to the address of the server
3. Send/Receive data
4. Close the socket

Steps followed by server to establish the connection:

1. Create a socket
2. Bind the socket to the port number known to all clients
3. Listen for the connection request
4. Accept connection request
5. Send/Receive data

Basic data structures used in Socket programming

Socket Descriptor: A simple file descriptor in Unix.

int

Socket Address: This construct holds the information for socket address

```
struct sockaddrs {
    unsigned short    sa_family;    // address family, AF_xxx or PF_xxx
    char              sa_data[14];  // 14 bytes of protocol address
};
```

AF stands for Address Family and PF stands for Protocol Family. In most modern implementations only the AF is being used. The various kinds of AF are as follows:

Name	Purpose
AF_UNIX, AF_LOCAL	Local communication
AF_INET	IPv4 Internet protocols
AF_INET6	IPv6 Internet protocols
AF_IPX	IPX - Novell protocols
AF_NETLINK	Kernel user interface device
AF_X25	ITU-T X.25 / ISO-8208 protocol
AF_AX25	Amateur radio AX.25 protocol
AF_ATMPVC	Access to raw ATM PVCs
AF_APPLETALK	Appletalk
AF_PACKET	Low level packet interface

In all the sample programs given below, we will be using AF_INET.

struct sockaddr_in: This construct holds the information about the address family, port number, Internet address,and the size of the struct sockaddr.

```
struct sockaddr_in {
    short int          sin_family;  // Address family
    unsigned short int sin_port;    // Port number
    struct in_addr     sin_addr;    // Internet address
    unsigned char      sin_zero[8]; // Same size as struct sockaddr
};
```

Some systems (like x8086) are Little Endian i-e. least signficant byte is stored in the higher address, whereas in Big endian systems most significant byte is stored in the higher address. Consider a situation where a Little Endian system wants to communicate with a Big Endian one, if there is no standard for data representation then the data sent by one machine is misinterpreted by the other. So standard has been defined for the data representation in the network (called Network Byte Order) which is the Big Endian. The system calls that help us to convert a short/long from Host Byte order to Network Byte Order and viceversa are

- htons() -- "Host to Network Short"
- htonl() -- "Host to Network Long"
- ntohs() -- "Network to Host Short"
- ntohl() -- "Network to Host Long"

IP addresses

Assuming that we are dealing with IPv4 addresses, the address is a 32bit integer. Remembering a 32 bit number is not convenient for humans. So, the address is written as a set of four integers seperated by dots, where each integer is a representation of 8 bits. The representation is like a.b.c.d, where a is the representation of the most significant byte. The system call which converts this representation into Network Byte Order is:

```
int inet_aton(const char *cp, struct in_addr *inp);
```

inet_aton() converts the Internet host address *cp* from the standard numbers-and-dots notation into binary data and stores it in the structure that *inp* points to. inet_aton returns nonzero if the address is valid, zero if not. For example, if we want to initialize the sockaddr_in construct by the IP address and desired port number, it is done as follows:

```
        struct sockaddr_in sockaddr;
        sockaddr.sin_family = AF_INET;
        sockaddr.sin_port = htons(21);
        inet_aton("172.26.117.168", &(sockaddr.sin_addr));
        memset(&(sockaddr.sin_zero), '\0', 8);
```

Socket System Call

A socket is created using the system call:

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

This system call returns a Socket Descriptor (like file descriptor) which is an integer value. Details about the Arguments:

1. **Domain:** It specifies the communication domain. It takes one of the predefined values described under the protocol family and address family above in this lecture.
2. **Type:** It specifies the semantics of communication , or the type of service that is desired . It takes the following values:
   - SOCK_STREAM : Stream Socket

    - SOCK_DGRAM : Datagram Socket
    - SOCK_RAW : Raw Socket
    - SOCK_SEQPACKET : Sequenced Packet Socket
    - SOCK_RDM : Reliably Delivered Message Packet
3. **Protocol:** This parameter identifies the protocol the socket is supposed to use . Some values are as follows:
    - IPPROTO_TCP : For TCP (SOCK_STREAM)
    - IPPROTO_UDP : For UDP (SOCK_DRAM)

   Since we have only one protocol for each kind of socket, it does not matter if we do not define any protocol at all. So for simplicity, we can put "0" (zero) in the protocol field.

Bind System Call

The system call bind associates an address to a socket descriptor created by socket.

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

The second parameter *myaddr* specifies a pointer to a predefined address of the socket.Its structure is a general address structure so that the bind system call can be used by both Unix domain and Internet domain sockets.

Other System Calls and their Functions

LISTEN : Annoumce willingness to accept connections ; give queue size.
ACCEPT : Block the caller until a commwction attempt arrives.
CONNECT : Actively attempt to establish a connection.
SEND : Send some data over the connection.
RECIEVE : Recieve sme data from the connection.
CLOSE : Release the connection.

---

back to top
Prev | Next | Index