Socket Programming

On Linux

Prayas Mohanty (Red Hat Certified Instructor)

Red Hat Certification ID: 100-005-594

Objective

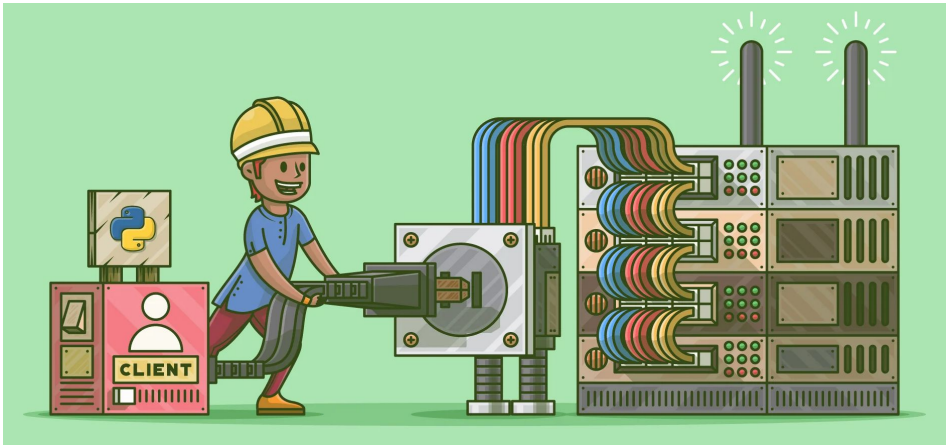
- What is Socket Programming
- Datagram Vs Stream Sockets
- How to Create a UDP Socket
- How to Create a TCP Socket
- How to Handle Concurrent Servers
- Coding Guidelines for Socket Programming.

Prerequisite of Participants

- Having Knowledge on Linux Platform
- Having familiarity with Linux Command line
- Basic Knowledge on vi Editor
- Proper Knowledge on C programming
- Basic Knowledge on File Handling in C

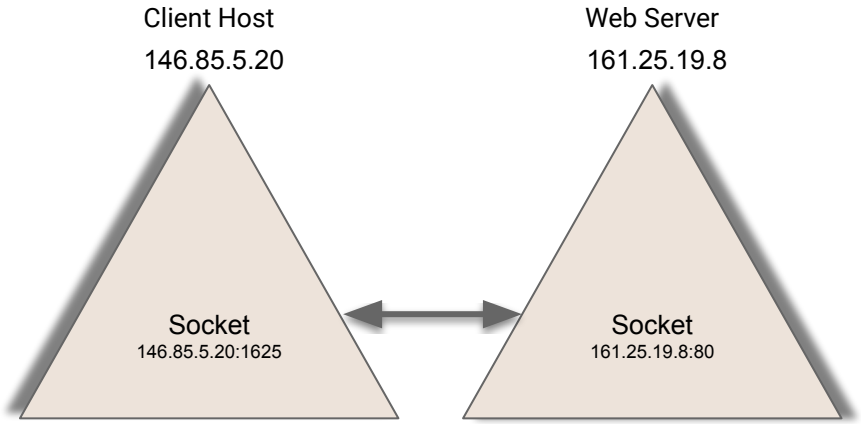
What is a Socket

- A socket is one endpoint of a two way communication link between two programs running on the network.
- The socket mechanism provides a means of inter-process communication (IPC) by establishing named contact points between which the communication take place.
- Like ‘Pipe’ is used to create pipes and sockets is created using ‘socket’ system call.
- The socket provides bidirectional FIFO Communication facility over the network.



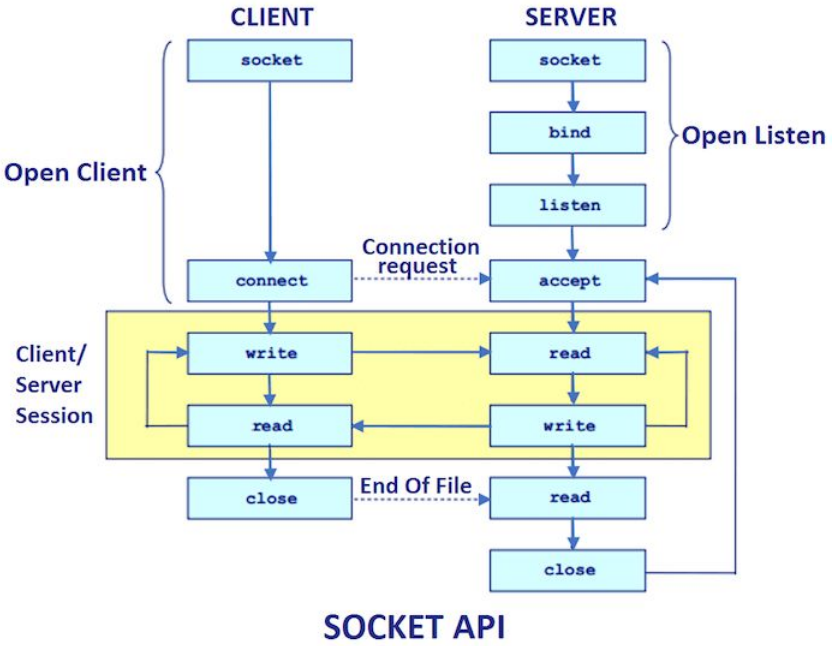
Why Socket Programming

- Socket programming shows how to use socket APIs to establish communication links between remote and local processes.
- The processes that use a socket can reside on the same system or different systems on different networks.
    - Sockets are useful for both stand-alone and network applications.
- Socket programming using C is extensively used for creating servers like web servers, mail servers, ftp servers and other servers.



Socket Programming Concepts

- Socket are generally employed in client server applications.
- The server creates a socket, attaches it to a network port addresses then waits for the client to contact it.
- The client creates a socket and then attempts to connect to the server socket.
- When the connection is established, transfer of data takes place.
- Client close connection after the required data get transferred.
- Server keep waiting for any new connection to get established.



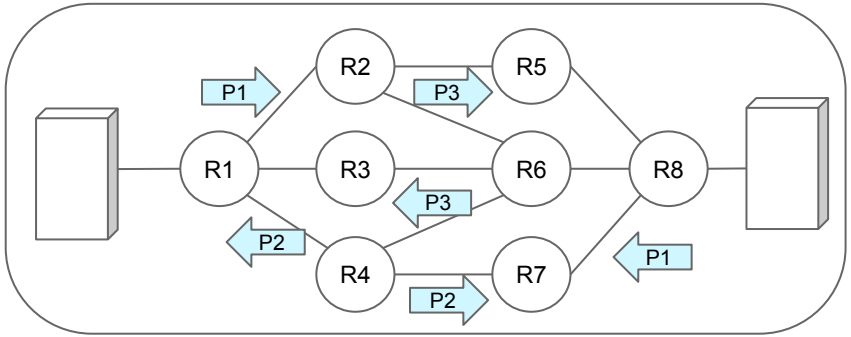
Types of Sockets (Datagram Vs Stream)

- Datagram Socket :
  - This is a type of network which has connectionless point for sending and receiving packets.
  - It is similar to mailbox.
  - The letters (data) posted into the box are collected and delivered (transmitted) to a letterbox (receiving socket).

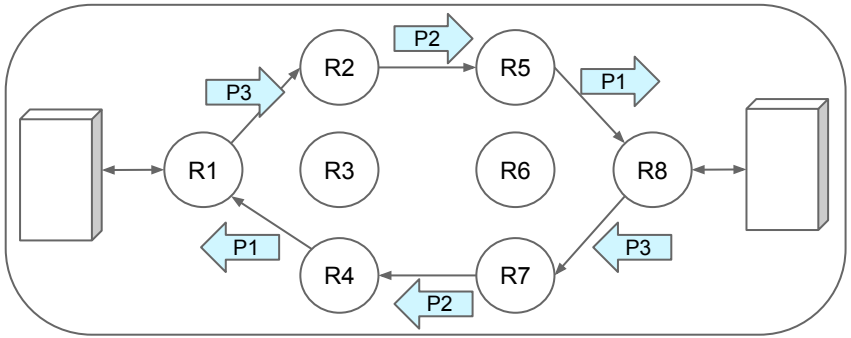
- Stream Socket
  - In Computer operating system, a stream socket is type of interprocess communications socket or network socket which provides a connection-oriented, sequenced, and unique flow of data without record boundaries with well defined mechanisms for creating and destroying connections and for detecting errors.
  - It is similar to phone.
  - A connection is established between the phones (two ends) and a conversation (transfer of data) takes place.

Connection-Oriented vs Connectionless Service

- Connection-oriented service involves the creation and termination of the connection for sending the data between two or more devices.
- In contrast, connectionless service does not require establishing any connection and termination process for transferring the data over a network.
- Transmission Control Protocol (TCP) is an example of a connection-oriented service, where as the UDP is an example of connectionless Service.



Connectionless Socket



Connection Oriented Socket

UDP Client Server Application

```c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <arpa/inet.h>

int main() {
    struct sockaddr in     servaddr;
    char buffer[1024];
    char *hello = "Hello from client";
    int n, len,sockfd,l2=sizeof(servaddr);
    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin family = AF INET;
    servaddr.sin port = htons(8080);
    servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");

    sockfd = socket(AF INET, SOCK DGRAM, 0);
    sendto(sockfd, hello, strlen(hello),MSG_CONFIRM,(struct sockaddr *)&servaddr,l2);
    printf("Hello message sent.\n");
    n = recvfrom(sockfd,buffer,1024,MSG_WAITALL,(struct sockaddr *)&servaddr,&len);
    buffer[n] = '\0';
    printf("Server : %s\n", buffer);
    close(sockfd);
}
```

```c
int main() {
    struct sockaddr in servaddr, cliaddr;
    char *hello = "Hello from server";
    char buffer[1024];
    int sockfd,n,len=sizeof(cliaddr),l2=sizeof(servaddr);

    memset(&servaddr, 0, sizeof(servaddr));
    memset(&cliaddr, 0, sizeof(cliaddr));
    servaddr.sin family    = AF INET; // IPv4
    servaddr.sin addr.s addr = INADDR_ANY;
    servaddr.sin_port = htons(8080);

    sockfd = socket(AF INET, SOCK DGRAM, 0);
    bind(sockfd, (const struct sockaddr *)&servaddr,l2);
    n=recvfrom(sockfd,buffer,1024,MSG_WAITALL,( struct sockaddr *)&cliaddr,&len);
    buffer[n] = '\0';
    printf("Client : %s\n", buffer);
    sendto(sockfd,hello,strlen(hello),MSG_CONFIRM,(struct sockaddr *)&cliaddr,len);
    printf("Hello message sent.\n");
}
```

TCP Client Server Applications

```c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <arpa/inet.h>
int main() {
    struct sockaddr in servaddr;
    char buffer[1024];
    char *hello = "Hello from client";
    int sockfd,l2=sizeof(servaddr);
    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin family = AF INET;
    servaddr.sin port = htons(8080);
    servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");

    sockfd = socket(AF INET, SOCK STREAM, 0);
    connect(sockfd,(struct sockaddr *)&servaddr,l2);
    write(sockfd, hello, 1024);
    printf("Hello message sent.\n");
    memset(buffer, 0, sizeof(buffer));
    read(sockfd,buffer,1024);
    printf("Server : %s\n", buffer);
    close(sockfd);
}
```

```c
int main() {
    struct sockaddr in servaddr, cliaddr;
    char hello[] = "Hello from server";
    char buffer[1024];
    int sockfd,connfd,len=sizeof(cliaddr),l2=sizeof(servaddr);

    memset(&servaddr, 0, sizeof(servaddr));
    memset(&cliaddr, 0, sizeof(cliaddr));
    servaddr.sin family    = AF INET; // IPv4
    servaddr.sin addr.s addr = INADDR_ANY;
    servaddr.sin_port = htons(8080);

    sockfd = socket(AF INET, SOCK STREAM, 0);
    bind(sockfd, (struct sockaddr *)&servaddr,l2);
    listen(sockfd,5);
    connfd=accept(sockfd,(struct sockaddr *)&cliaddr, &len);

    memset(buffer, 0, sizeof(buffer));
    read(connfd,buffer,1024);
    printf("Client : %s\n", buffer);
    write(connfd,hello,1024);
    printf("Hello message sent.\n");
    close(sockfd);
}
```

Concurrent Server

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <arpa/inet.h>
int main() {
    struct sockaddr in servaddr;
    char buffer[1024];
    char *hello = "Hello from client";
    int sockfd,l2=sizeof(servaddr);
    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin family = AF INET;
    servaddr.sin port = htons(8080);
    servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");

    sockfd = socket(AF INET, SOCK STREAM, 0);
    connect(sockfd,(struct sockaddr *)&servaddr,l2);
    write(sockfd, hello, 1024);
    printf("Hello message sent.\n");
    memset(buffer, 0, sizeof(buffer));
    read(sockfd,buffer,1024);
    printf("Server : %s\n", buffer);
    close(sockfd);
}
```

```
int main() {
    struct sockaddr in servaddr, cliaddr;
    char hello[] = "Hello from server",buffer[1024];
    pid t pid;
    int sockfd,connfd,len=sizeof(cliaddr),l2=sizeof(servaddr);
    memset(&servaddr, 0, sizeof(servaddr));
    memset(&cliaddr, 0, sizeof(cliaddr));
    servaddr.sin family    = AF INET; // IPv4
    servaddr.sin addr.s addr = INADDR_ANY;
    servaddr.sin port = htons(8080);
    sockfd = socket(AF INET, SOCK STREAM, 0);
    bind(sockfd, (struct sockaddr *)&servaddr,l2);
    listen(sockfd,5);
    while(0<(connfd=accept(sockfd,(struct sockaddr *)&cliaddr, &len))) {
        if (0 == (pid = fork())) {
                        memset(buffer, 0, sizeof(buffer));
                        read(connfd,buffer,1024);
                        printf("Client : %s\n", buffer);
                        write(connfd,hello,1024);
                        printf("Hello sent by: %d.\n",(int) getpid());
                        close(sockfd);
                        close(connfd);
                }
        }
}
```

Coding Guidelines

- Handle error after socket system calls using perror. In case of read/write failures, close socket and exit.
- Remember use of INADDR_ANY in bind() to bind to all local interfaces
- Use of connect() uses a random free port.
- Distinguish between shutdown() and close() and use appropriately.
- Use strlen(buf) +1 as size in send() for string data.
- Close socket on exit.

Summary

- What is Socket Programming
- Datagram Vs Stream Sockets
- How to Create a UDP Socket
- How to Create a TCP Socket
- How to Handle Concurrent Servers
- Coding Guidelines for Socket Programming.

Any Questions ?

Thank you!