C Programming in Linux Tutorial #034 – Socket Programming

En este tutorial, exploraremos cómo realizar programación de sockets en C en el entorno Linux. La programación de sockets permite que los programas se comuniquen a través de una red utilizando protocolos como TCP e UDP.

¿Qué es un socket?

Un socket es un punto final en una comunicación entre dos programas a través de una red. Los sockets permiten que los datos se transmitan entre procesos en la misma máquina o en máquinas diferentes utilizando protocolos de comunicación.

Tipos de sockets

- **Sockets de flujo (Stream Sockets):** Utilizan el protocolo TCP para una comunicación orientada a conexión y confiable.
- **Sockets de datagramas (Datagram Sockets):** Utilizan el protocolo UDP para una comunicación sin conexión y sin garantía de entrega.

Programación de un socket TCP

A continuación, se muestra cómo implementar un servidor y un cliente usando sockets TCP.

1. Servidor TCP

Primero, crearemos un servidor que escuche en un puerto y acepte conexiones de clientes.

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

#define PORT 8080
#define BUFFER_SIZE 1024

int main() {
    int server_fd, new_socket;
    struct sockaddr_in address;
    int addrlen = sizeof(address);
    char buffer[BUFFER_SIZE] = {0};

    // Crear el socket
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }

    // Configurar la dirección del servidor
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

    // Enlazar el socket
      if (bind(server_fd, (struct sockaddr *)&address,
sizeof(address)) < 0) {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }

    // Escuchar por conexiones
    if (listen(server_fd, 3) < 0) {
        perror("listen");
        exit(EXIT_FAILURE);
    }

    // Aceptar una conexión
```

```
     if ((new_socket = accept(server_fd, (struct sockaddr
*)&address, (socklen_t*)&addrlen)) < 0) {
        perror("accept");
        exit(EXIT_FAILURE);
    }

    // Leer datos del cliente
    read(new_socket, buffer, BUFFER_SIZE);
    printf("Mensaje del cliente: %s\n", buffer);

    // Enviar una respuesta al cliente
     send(new_socket, "Hola desde el servidor", strlen("Hola
desde el servidor"), 0);
    printf("Mensaje enviado al cliente\n");

    // Cerrar el socket
    close(new_socket);
    close(server_fd);

    return 0;
}
```

2. Cliente TCP

Ahora, crearemos un cliente que se conecte al servidor y envíe un mensaje.

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

#define PORT 8080
#define BUFFER_SIZE 1024

int main() {
    int sock = 0;
    struct sockaddr_in serv_addr;
```

```
    char *message = "Hola desde el cliente";
    char buffer[BUFFER_SIZE] = {0};

    // Crear el socket
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket creation error");
        exit(EXIT_FAILURE);
    }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    // Convertir IPv4 e IPv6 direcciones desde texto a binario
     if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr)
<= 0) {
        perror("invalid address / address not supported");
        exit(EXIT_FAILURE);
    }

    // Conectar al servidor
      if (connect(sock, (struct sockaddr *)&serv_addr,
sizeof(serv_addr)) < 0) {
        perror("connection failed");
        exit(EXIT_FAILURE);
    }

    // Enviar un mensaje al servidor
    send(sock, message, strlen(message), 0);
    printf("Mensaje enviado al servidor\n");

    // Leer la respuesta del servidor
    read(sock, buffer, BUFFER_SIZE);
    printf("Respuesta del servidor: %s\n", buffer);

    // Cerrar el socket
    close(sock);

    return 0;
}
```

Programación de un socket UDP

Ahora, implementaremos un servidor y un cliente utilizando sockets UDP.

1. Servidor UDP

El servidor UDP recibirá mensajes de clientes y enviará una respuesta.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h&gt>

#define PORT 8080
#define BUFFER_SIZE 1024

int main() {
    int sockfd;
    struct sockaddr_in server_addr, client_addr;
    socklen_t addr_len = sizeof(client_addr);
    char buffer[BUFFER_SIZE] = {0};

    // Crear el socket
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }

    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(PORT);

    // Enlazar el socket
      if (bind(sockfd, (struct sockaddr *)&server_addr,
sizeof(server_addr)) < 0) {
        perror("bind failed");
```

```
        exit(EXIT_FAILURE);
    }

    // Recibir mensaje del cliente
    recvfrom(sockfd, buffer, BUFFER_SIZE, 0, (struct sockaddr
*)&client_addr, &addr_len);
    printf("Mensaje recibido del cliente: %s\n", buffer);

    // Enviar una respuesta al cliente
    sendto(sockfd, "Hola desde el servidor UDP", strlen("Hola
desde el servidor UDP"), 0, (struct sockaddr *)&client_addr,
addr_len);
    printf("Respuesta enviada al cliente UDP\n");

    // Cerrar el socket
    close(sockfd);

    return 0;
}
```

2. Cliente UDP

El cliente UDP enviará un mensaje al servidor y recibirá una respuesta.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h&gt>

#define PORT 8080
#define BUFFER_SIZE 1024

int main() {
    int sockfd;
    struct sockaddr_in server_addr;
    char *message = "Hola desde el cliente UDP";
    char buffer[BUFFER_SIZE] = {0};
```

```
    // Crear el socket
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT);

    // Convertir IPv4 e IPv6 direcciones desde texto a binario
    if (inet_pton(AF_INET, "127.0.0.1", &server_addr.sin_addr)
<= 0) {
        perror("invalid address / address not supported");
        exit(EXIT_FAILURE);
    }

    // Enviar un mensaje al servidor
      sendto(sockfd, message, strlen(message), 0, (struct
sockaddr *)&server_addr, sizeof(server_addr));
    printf("Mensaje enviado al servidor UDP\n");

    // Recibir una respuesta del servidor
    recvfrom(sockfd, buffer, BUFFER_SIZE, 0, NULL, NULL);
    printf("Respuesta del servidor UDP: %s\n", buffer);

    // Cerrar el socket
    close(sockfd);

    return 0;
}
```

Consideraciones

La programación de sockets es fundamental para crear aplicaciones de red. TCP proporciona una comunicación confiable, mientras que UDP es más adecuado para aplicaciones que requieren velocidad y pueden tolerar pérdidas de datos.