Protocolos De Internet

Trabajo Práctico 1 – Parte A

Alumnos: Pablo Lamela – Facundo Monti – Mateo Villanueva

Profesor: Ing. Javier A. Ouret

Año: 2024

1) Explicar qué es un socket y los diferentes tipos de sockets.
2) Cuáles son las estructuras necesarias para operar con sockets en el modelo C-S y cómo se hace para ingresar los datos requeridos. Explicar con un ejemplo.
3) Explicar modo bloqueante y no bloqueante en sockets y cuáles son los “sockets calls” a los cuales se pueden aplicar estos modos. Explicar las funciones necesarias.
4) Describir el modelo C-S aplicado a un servidor con Concurrencia Real. Escribir un ejemplo en lenguaje C.
5) ¿Cómo se cierra una conexión C-S? Métodos que eviten la pérdida de información.
6) Probar el código del chat entre cliente y servidor. Cambiar tipo de socket y volver a probar.

1) Un socket es un concepto fundamental en la comunicación de redes y se utiliza para establecer una conexión entre dos programas en una red. Funciona como un punto final para enviar y recibir datos entre dos nodos en una red.
   Existen diferentes tipos de sockets, los más comunes son:
   1) Socket de flujo (Stream Socket):
   -Se basa en el protocolo TCP (Transmission Control Protocol).
   -Proporciona una conexión orientada a la conexión y confiable entre dos hosts.
   -Garantiza que los datos se entreguen en el orden correcto y sin errores.
   2) Socket de datagrama (Datagram Socket):
   -Utiliza el protocolo UDP (User Datagram Protocol).
   -Proporciona una conexión sin conexión, lo que significa que los paquetes de datos pueden llegar en un orden diferente al enviado o pueden perderse.
   -Es más rápido, pero menos confiable que el socket de flujo.
   3) Socket crudo (Raw Socket):
   -Permite el acceso directo a los paquetes de red a un nivel más bajo que los sockets de flujo y datagrama.
   -Se utilizan para la implementación de protocolos de red personalizados y para tareas de diagnóstico avanzadas.
2) La primera estructura principal es un descriptor de socket, que simplemente es un int.

   La segunda estructura es sockaddr, la cual contiene información de direcciones de socket para diversos tipos de este. Tiene el siguiente formato:

```c
struct sockaddr{
    unsigned short sa_family;
    char sa_data[14];
}
```

sa_family indica la familia de direcciones utilizada en una dirección de socket específica. Esta familia de direcciones define el tipo de dirección de red que se está utilizando, como IPv4, IPv6, u otras familias de direcciones de red. Y sa_data contiene una dirección IP y un numero de puerto.

En tercer lugar, tenemos la estructura sockaddr_in que simplemente es una estructura paralela a sockaddr que fue creada con el objetivo de que sea más práctico rellenar la información que debería ir en sa_data. Así es su formato:

```c
struct sockaddr_in {
    short int sin_family; // familia de direcciones, AF_INET
    unsigned short int sin_port; // Número de puerto
    struct in_addr sin_addr; // Dirección de Internet
    unsigned char sin_zero[8]; // Relleno para preservar el tamaño original de struct sockaddr
};
```

Y, por último, en la estructura de sockaddr_in vemos que aparece otra llamada in_addr. Esta solamente esta formada por un unsigned long que se utiliza para almacenar la dirección ip.

A continuación, podemos ver como se haría para ingresar la información mencionada en una variable de tipo struct sockaddr_in:

```c
struct sockaddr_in server_addr;
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(PORT);
server_addr.sin_addr.s_addr = inet_addr("127.0.0.0");
bzero(&(server_addr.sin_zero), 8); // pongo en 0 el resto de la estructura
```

El uso de la función htons() es para pasar de la ordenación de bytes de la maquina a la utilizada en la red.

3) Los modos bloqueante y no bloqueante se refieren a cómo se comportan las llamadas de funciones de sockets en términos de bloqueo del proceso mientras espera una operación de entrada/salida en el socket.
   En el modo bloqueante, las llamadas a las funciones de sockets hacen que el proceso se bloquee hasta que la operación de E/S se complete.
   En el modo no bloqueante, las llamadas a las funciones de sockets no bloquearán el proceso, incluso si la operación de E/S no se puede realizar inmediatamente. En lugar de eso, estas llamadas devolverán un valor indicando que la operación no se pudo completar en ese momento. Esto permite que el proceso realice otras tareas mientras espera que ocurra alguna actividad en el socket.
   Las funciones que pueden utilizar los distintos modos son:
   • listen() y accept(): se usan para escuchar y aceptar la conexión entre C-S. En modo bloqueante, paran la ejecución hasta que se reciba una conexión entrante. En modo no bloqueante la ejecución continúa, aunque no se haya recibido una conexión.
   • bind(): Asocia al socket con el puerto de la máquina local.
   • connect(): solicita una conexión a un servidor. En modo bloqueante, el hilo no seguirá hasta que se realice la conexión, sin embargo, en modo no bloqueante, mientras se envían los datos, el programa continúa.
   • send() y sendto(): envia datos a través de un socket. En modo bloqueante, si el socket de destino está lleno, la llamada a send() bloqueará el proceso hasta que haya espacio disponible para enviar los datos.
   • recv() y recvfrom(): se usan para recibir datos. En modo bloqueante, si no hay datos disponibles para recibir, se bloqueará el proceso hasta que se reciban datos o se produzca un error.
   El kernel puede bloquear algunas funciones por defecto. Para evitar esto se utiliza la función fcntl().
   Para que el servidor escuche a conexiones entrantes y, al mismo tiempo, atienda diferentes solicitudes de diferentes sockets se usa la función select(), que permite comprobar varios sockets al mismo tiempo.
   La función fork() sirve para abrir un hilo de ejecución en otro hilo de ejecución, generando un proceso “hijo” del primero.
4) En el modelo C-S con concurrencia real el servidor será capaz de atender múltiples clientes en simultaneo. Para lograrlo, este hará uso de la función fork() para así poder ir creando procesos hijos que manejen la conexión con los clientes. El proceso padre, se mantendrá aceptando las nuevas conexiones que luego serán manejadas por los procesos hijos. Aquí hay un ejemplo de un servidor concurrente:

```c
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
//#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/wait.h>
#include <unistd.h>

#define MY_PORT 3535
#define BACKLOG 10

int main(){

    int sockfd, new_fd;
    struct sockaddr_in my_addr;
    struct sockaddr_in client_addr;
    int sin_size;

    if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1){
        perror("socket");
        exit(1);
    }

    my_addr.sin_family = AF_INET;
    my_addr.sin_port = htons(MY_PORT);
    my_addr.sin_addr.s_addr = INADDR_ANY;
    bzero(&(my_addr.sin_zero), 8);

    if(bind(sockfd, (struct sockaddr*)&my_addr, sizeof(struct sockaddr)) == -1){
        perror("bind");
        exit(1);
    }

    if(listen(sockfd, BACKLOG) == -1){
        perror("listen");
        exit(1);
    }

    while(1){
        sin_size = sizeof(struct sockaddr_in);
        if((new_fd = accept(sockfd, (struct sockaddr*)&client_addr, &sin_size))==-1){
            perror("accept");
            continue;
        }
        printf("Server: got connection from |%s|\n", inet_ntoa(client_addr.sin_addr));

        if(!fork()){
            // lo ejecuta el proceso hijo
            if(send(new_fd, "Hello world\n", 14, 0) == -1) perror("send");
            close(new_fd);
            exit(0);
        }
        close(new_fd);
        while(waitpid(-1, NULL, WNOHANG) > 0);
    }

    return 0;
}
```

5) Para cerrar una conexión C-S podemos hacer uso de la función close() la cual toma como parámetro un descriptor de socket. Sin embargo, debemos tener en cuenta que esto produce una interrupción abrupta en la conexión ya que lo que hace es no permitir que se sigan realizando operaciones de E/S a ese descriptor de socket. Por lo tanto, para

tener un poco más de control sobre como queremos que se cierre la conexión, podemos utilizar la función shutdown(), que toma como parámetros un descriptor de socket y otro int que especifica como se debe producir la desconexión. Estos son los valores que puede tomar ese entero:

- 0 → No se permite recibir más datos
- 1 → No se permite enviar más datos
- 2 → No se permite enviar ni recibir más datos (lo mismo que close( ))

6) Lo que cambiaría en los códigos es que en vez de utilizar las funciones send() y recv() podría utilizar sendto() y recvfrom(), en la conexión con datagramas, para enviar y recibir información entre C-S. Además, del lado del servidor no hace falta hacer uso de las funciones listen() y accept() ya que sería una comunicación no orientada a la conexión.