Protocolos
De
Internet

Trabajo Práctico 1 – Parte A

Sockets

Alumnos: Soler Gonzalo
Soler Milagros
Tobio Tomás
Pejinakis Juan Ignacio
Basara Ignacio

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.
    - Presentar las respuestas en archivos ,pdf identificando alumno y grupo.
    - C-S = Cliente - Servidor

1) En comunicaciones, la concurrencia es igual de importante que la comunicación en sí. Para lograr que exista concurrencia, es decir, que el cliente pueda computar de forma simultánea y que el servidor pueda manejar esos pedidos, es necesario el uso de los sockets. Entonces, un socket es aquella estructura de datos que permite mantener una conexión activa entre cliente y servidor.

   Existen diferentes tipos de sockets, entre ellos:

   Los sockets raw son herramientas poderosas que permiten acceder directamente a la capa de red y de enlace de datos en sistemas Unix. Se utilizan para implementar protocolos de red personalizados, monitorear redes, olfatear paquetes o enviar paquetes falsificados. Estos sockets pueden ser útiles para implementar nuevos protocolos en el espacio de usuario. Al abrir un socket raw, se puede especificar si se desea que el kernel construya el encabezado IP automáticamente o si se prefiere construirlo manualmente. Esto brinda un control total sobre los campos del encabezado IP, como el TTL y la dirección IP de origen. En la capa de enlace, se debe construir manualmente la trama (frame) y, en algunos casos, se puede vincular el socket a un dispositivo de red específico. Los sockets raw también permiten recuperar información de los paquetes, como los encabezados de Ethernet e IP. Unix proporciona dos tipos de sockets raw: SOCK_PACKET y SOCK_RAW, siendo este último el más común. Estos sockets pueden ser utilizados para escuchar tráfico TCP, UDP o ICMP, creando sockets raw separados para cada protocolo.

   Los sockets de flujo son herramientas que definen flujos de comunicación confiables y bidireccionales, utilizando el protocolo TCP en la capa de transporte y el protocolo IP para el enrutamiento. Cuando

se envían mensajes a través de estos sockets, llegan al otro extremo en el mismo orden en el que fueron enviados, y sin errores.

Un ejemplo común de aplicación de los sockets de flujo es el protocolo telnet, donde todos los caracteres ingresados deben llegar al otro extremo en el mismo orden en que fueron tipeados. Los navegadores web también utilizan sockets de flujo cuando acceden a páginas web a través de los protocolos HTTP/HTTPS.

Los sockets de Datagrama son herramientas de comunicación sin conexión que utilizan el protocolo UDP en la capa de transporte y el protocolo IP para el enrutamiento. Al enviar un datagrama a través de estos sockets, no se garantiza que llegue al destino, y si llega, puede llegar fuera de secuencia. Sin embargo, si llega, los datos contenidos en el paquete no tendrán errores.

Dado que no es necesario mantener una conexión abierta como con los sockets de flujo, simplemente se arma un paquete, se le coloca una cabecera IP con la información de destino y se envía. Estos sockets se utilizan generalmente para transferencias de información por paquetes, donde no se necesita una conexión continua.

Ejemplos de aplicaciones que utilizan este tipo de socket son el tftp y el bootp. Es importante destacar que, al usar sockets de datagrama, es necesario implementar un protocolo encima de UDP si se quiere asegurar que los paquetes lleguen correctamente.

2) Existen tres estructuras necesarias para manejar sockets:

Sockaddr: Es la estructura que mantiene la información de direcciones de socket. Utiliza un char corto no signado para indicar la familia de protocolos del socket y un char de 14 bytes con la dirección y número del puerto de destino del socket.

Sockaddr_in: Es una estructura que resulta más sencilla para referirse a los elementos del socket. Utiliza:

- Un entero corto para referirse a la familia de direcciones (AF_INET)
- Un entero corto no signado para el número de puerto
- Una estructura para la dirección IP.
- Un char no signado de 8 bytes para relleno.

In_addr: Es la estructura que quedó por herencia histórica que describe la dirección IP de 4 bytes utilizando un entero largo.

Ejemplo de constructor de sockadrr_in:

```
struck sockaddr_in my addr;  //Pongo nombre a la estructura

myaddr.sin_family = AF_INET;  //Siempre deberá tener este valor

my_addr.sin_port = htons(MYPORT);  //Mediante la función htons paso de ordenación de máquina a ordenación de red y defino el puerto de destino.

inet_aton("10.12.110.57", &(my_addr.sin_addr));  //Esta función convierte el primer parámetro a un entero largo y lo coloca en la dirección de memoria del segundo parámetro. De esta manera defino la dirección IP de destino.

memset(&(my_addr.sin_zero), '\0', 8);  //Esta función rellena la estructura con 0.
```

3) El modo bloqueante en sockets implica que la ejecución del programa se detenga cada vez que se llame a una función y se realice una operación. En cambio, el modo no bloqueante realiza la operación sin necesidad de parar la ejecución del programa.
   Las funciones que pueden utilizar los distintos modos son:
   - Listen() y Accept(), las cuales son utilizadas para escuchar y aceptar la conexión entre C-S y que, en modo bloqueante, paran la ejecución hasta que reciba una conexión entrante. En modo no bloqueante, si no reciben una conexión, simplemente sigue la ejecución.
   - Bind(), la cual es utilizada para asociar al socket con el puerto de la máquina local.
   - Connect(), la cual se utiliza para solicitar una conexión a un servidor. En modo bloqueante, el hilo no seguirá hasta que no se realice la conexión, sin embargo en modo no bloqueante, mientras se envían los datos, el programa continúa.
   - Send(), recv() y recvfrom(), las cuales se utilizan para enviar y recibir datos.

   Para evitar que una función se bloquee por el kernel por defecto, se utiliza la función fcntl().

   Para poder atender diferentes solicitudes de diferentes sockets mientras el servidor sigue escuchando a conexiones entrantes, se utiliza la función select(), la cual permite comprobar varios sockets al mismo tiempo.

   La función fork() es utilizada para abrir un hilo de ejecución en otro hilo de ejecución, generando un proceso "hijo" del primero que había.

4) El modelo C-S en modo de Concurrencia Real es aquel en el que el servidor puede manejar múltiples solicitudes de clientes simultáneamente, o sea, puede procesar solicitudes varias sin que se produzcan conflictos o bloqueos en el mismo procesamiento (contrario al modo iterativo, en que se procesa una solicitud a la vez).

Este modelo se puede acceder al hacer uso de hilos o copias del servidor (zombies). Cada solicitud se maneja en un hilo o proceso separado, lo que permite que varias solicitudes se procesen de manera concurrente.
Este modelo mejora la eficiencia y la escalabilidad del servidor, lo que permite que el sistema pueda manejar un mayor número de solicitudes de clientes sin afectar su rendimiento, por lo que también hace que su costo sea mayor.

Comunicación entre dos clientes y servidor:

```c
#include <stdio.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h> // read(), write(), close()
#define MAX 80
#define PORT 8080
#define SA struct sockaddr

// Función para el chat entre el cliente y el servidor
void func(int connfd)
{
    char buff[MAX];
    int n;
    // Loop para el chat
    while (1)
    {
        bzero(buff, MAX);

        // Leo el mesaje del cliente y lo copio en un buffer
        read(connfd, buff, sizeof(buff));
        // si el mensaje dice "exit" salgo y cierro conexión
        if (strncmp(buff, "exit", 4) == 0)
        {
            printf("Salgo del servidor...\n");
            break;
        }
        // Muestro el buffer con los datos del cliente
        printf("Del cliente: %s\t: ", buff);
        bzero(buff, MAX);
        n = 0;
        // Copio el mensaje del servidor en el buffer
        while ((buff[n++] = getchar()) != '\n')
            ;

        // y envío el buffer al cliente
```

```
        write(connfd, buff, sizeof(buff));
    }
}


int main()
{
    int sockfd, connfd, len;
    struct sockaddr_in servaddr, cli;

    // Creo un socket y verifico que se cree bien
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1)
    {
        printf("Falla la creación del socket...\n");
        exit(0);
    }
    printf("Socket creado...\n");
    bzero(&servaddr, sizeof(servaddr));

    // asigno IP, PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(PORT);

    // Bind a la dirección IP y lo verifico
    if ((bind(sockfd, (SA *)&servaddr, sizeof(servaddr))) != 0)
    {
        printf("Falla socket bind ...\n");
        exit(0);
    }
    printf("Se hace el socket bind ..\n");
    while (1)
    {
        if ((listen(sockfd, 5)) != 0)
        // El servidor está en modo escucha (pasivo) y lo verifico
        {
            printf("Falla el listen ...\n");
            exit(0);
        }
        printf("Servidor en modo escucha ...\n");
        len = sizeof(cli);
        connfd = accept(sockfd, (SA *)&cli, &len);
        if (connfd < 0)
        {
            printf("Falla al aceptar datos en el servidor...\n");
            exit(0);
        }
        else
```

```
            printf("El servidor acepta al cliente ...\n");
        int pid = fork();
        if (pid == 0)
            func(connfd);
    }

    close(sockfd);
}
```

```
o-1-grupo-4/TP1$
tomastobio@DESKTOP-2UT9IVO:/mnt/c/Users/Usuario/Documents/GitHub/trabajo-practic
o-1-grupo-4/TP1$ ./server
Socket creado...
Se hace el socket bind ..
Servidor en modo escucha ...
El servidor acepta al cliente ...
Servidor en modo escucha ...
Del cliente: Hola
        : Buenas tardesz
El servidor acepta al cliente ...
Servidor en modo escucha ...
Del cliente: Hola
        : Buenas noches
Del cliente: Nos vemos
        : Hasta luego
Del cliente: Hasta la proxima
        : SALIR
Del cliente:    :
```

```
o-1-grupo-4/TP1$
tomastobio@DESKTOP-2UT9IVO:/mnt/c/Users/Usuario/Documents/Github/trabajo-practic
o-1-grupo-4/TP1$ ./client1
Socket creado ..
Conectado al servidor..
Ingrese texto: Hola
Servidor : Buenas tardesz
Ingrese texto: Nos vemos
Servidor : Hasta luego
Ingrese texto:
```

```
o-practico-1-grupo-4/TP1$ ./client2
Socket creado ..
Conectado al servidor..
Ingrese texto: Hola
Servidor : Buenas noches
Ingrese texto: Hasta la proxima
Servidor : SALIR
Cliente cierra conexión...
tomastobio@DESKTOP-2UT9IVO:/mnt/c/Users/Usuario/Documents/Github/trabaj
```

5) La función close() permite cerrar el descriptor de archivo del socket. Impide que se realicen más lecturas o escrituras en el socket. Sin embargo, no realiza ningún tipo de control de cómo se cierra el socket.
Para añadir control, se utiliza la función shutdown() que dependiendo del segundo parámetro, indica cómo lo cierra:

0—No permite recibir más datos
1—No permite enviar más datos
2—No permite enviar ni recibir datos (igual que close())

Con esta función se logra evitar que se cierre la conexión cuando se está en medio de un envío o recibimiento de datos. Sin embargo, shutdown() no libera el descriptor. Para eso se debe usar close().

6) Prueba con Servidor UDP / Cliente UDP

Servidor

```c
#include <stdio.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h> // read(), write(), close()
#define MAX 80
#define PORT 8080
#define SA struct sockaddr

void func(int sockfd, struct sockaddr_in cli)
{
    char buff[MAX];
    int n, len;
    struct sockaddr_in cliaddr; // Crear una variable para almacenar la dirección del remitente

    // Loop para el chat
    for (;;) {
        bzero(buff, MAX);

        // Leo el mesaje del cliente y lo copio en un buffer
        len = sizeof(cli);
        n = recvfrom(sockfd, buff, sizeof(buff), 0, (struct sockaddr *)&cli, &len);
        // Muestro el buffer con los datos del cliente
        printf("Del cliente: %s\t: ", buff);
        bzero(buff, MAX);
        n = 0;
        // Copio el mensaje del servidor en el buffer
        while ((buff[n++] = getchar()) != '\n');

        // y envío el buffer al cliente
```

```
        sendto(sockfd, buff, sizeof(buff), 0, (struct sockaddr *)&cli,
sizeof(cli));

        // si el mensaje dice "SALIR" salgo y cierro conexión
        if (strncmp("SALIR", buff, 4) == 0) {
            printf("Salgo del servidor...\n");
            break;
        }
    }
}

// Función principal
int main()
{
    int sockfd, len;
    struct sockaddr_in servaddr, cli;

    // socket create and verification
    sockfd = socket(AF_INET, SOCK_DGRAM, 0); // UDP
    if (sockfd == -1) {
        printf("Falla la creación del socket...\n");
        exit(0);
    }
    else
        printf("Socket creado...\n");
        bzero(&servaddr, sizeof(servaddr));

    // asigno IP, PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(PORT);

    // Bind del nuevo socket a la dirección IP y lo verifico
    if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
        printf("Falla socket bind ...\n");
        exit(0);
    }
    else
        printf("Se hace el socket bind ..\n");

    // Llamo a la función para el chat entre el cliente y el servidor
    func(sockfd, cli);

    // Cierro el socket al terminar el chat
    close(sockfd);
}
```

Cliente

```c
#include <arpa/inet.h> // inet_addr()
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h> // bzero()
#include <sys/socket.h>
#include <unistd.h> // read(), write(), close()
#define MAX 80
#define PORT 8080
#define SA struct sockaddr

void func(int sockfd, struct sockaddr_in servaddr)
{
    char buff[MAX];
    int n;
    for (;;) {
        bzero(buff, sizeof(buff));
        printf("Ingrese texto : ");
        n = 0;
        while ((buff[n++] = getchar()) != '\n');
        sendto(sockfd, buff, sizeof(buff), 0, (SA*)&(servaddr),
sizeof(servaddr));
        bzero(buff, sizeof(buff));
        recvfrom(sockfd, buff, sizeof(buff), 0, NULL, NULL);
        printf("Servidor : %s", buff);
        if ((strncmp("SALIR",buff, 4)) == 0) {
            printf("Cliente cierra conexión...\n");
            break;
        }
    }
}

int main()
{
    int sockfd, connfd;
    struct sockaddr_in servaddr, cli;

    // socket: creo socket y lo verifico
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd == -1) {
        printf("Falla la creación del socket...\n");
        exit(0);
    }
    else
        printf("Socket creado ..\n");
    bzero(&servaddr, sizeof(servaddr));
```

```c
    // asigno IP, PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
    servaddr.sin_port = htons(PORT);

    // Conecto los sockets entre cliente y servidor
    if (connect(sockfd, (SA*)&servaddr, sizeof(servaddr))
        != 0) {
        printf("Falla de conexión con servidor...\n");
        exit(0);
    }
    else
        printf("Conectado al servidor..\n");

    // Función para el chat
    func(sockfd,servaddr);

    //Cierro el socket
    close(sockfd);
}
```

```
tomastobio@DESKTOP-2UT9IVO:/mnt/c/Users/Usuario/Documents/github/trabajo-practi
co-1-grupo-4/TP1$ ./server
Socket creado...
Se hace el socket bind ..
Del cliente: hola
        : buenas tardes
Del cliente: como estas? 23
        : SALIR
Salgo del servidor...
tomastobio@DESKTOP-2UT9IVO:/mnt/c/Users/Usuario/Documents/github/trabajo-practi
co-1-grupo-4/TP1$
```

```
tomastobio@DESKTOP-2UT9IVO:/mnt/c/Users/Usuario/Documents/Github/trabajo-practico-1-
grupo-4/TP1$ ./client
Socket creado ..
Conectado al servidor..
Ingrese texto : hola
Servidor : buenas tardes
Ingrese texto : como estas? 23
Servidor : SALIR
Cliente cierra conexión...
tomastobio@DESKTOP-2UT9IVO:/mnt/c/Users/Usuario/Documents/Github/trabajo-practico-1-
grupo-4/TP1$
```

Prueba con Servidor TCP / Cliente TCP

Servidor

```c
#include <stdio.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h> // read(), write(), close()
#define MAX 80
#define PORT 8080
#define SA struct sockaddr

// Función para el chat entre el cliente y el servidor
void func(int connfd)
{
    char buff[MAX];
    int n;
    // Loop para el chat
    while (1)
    {
        bzero(buff, MAX);

        // Leo el mesaje del cliente y lo copio en un buffer
        read(connfd, buff, sizeof(buff));
        // si el mensaje dice "exit" salgo y cierro conexión
        if (strncmp(buff, "exit", 4) == 0)
        {
            printf("Salgo del servidor...\n");
            break;
        }
        // Muestro el buffer con los datos del cliente
        printf("Del cliente: %s\t: ", buff);
        bzero(buff, MAX);
        n = 0;
        // Copio el mensaje del servidor en el buffer
        while ((buff[n++] = getchar()) != '\n')
            ;

        // y envío el buffer al cliente
        write(connfd, buff, sizeof(buff));
    }
}


int main()
{
    int sockfd, connfd, len;
    struct sockaddr_in servaddr, cli;
```

```
    // Creo un socket y verifico que se cree bien
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1)
    {
        printf("Falla la creación del socket...\n");
        exit(0);
    }
    printf("Socket creado...\n");
    bzero(&servaddr, sizeof(servaddr));

    // asigno IP, PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(PORT);

    // Bind a la dirección IP y lo verifico
    if ((bind(sockfd, (SA *)&servaddr, sizeof(servaddr))) != 0)
    {
        printf("Falla socket bind ...\n");
        exit(0);
    }
    printf("Se hace el socket bind ..\n");
    while (1)
    {
        if ((listen(sockfd, 5)) != 0)
        // El servidor está en modo escucha (pasivo) y lo verifico
        {
            printf("Falla el listen ...\n");
            exit(0);
        }
        printf("Servidor en modo escucha ...\n");
        len = sizeof(cli);
        connfd = accept(sockfd, (SA *)&cli, &len);
        if (connfd < 0)
        {
            printf("Falla al aceptar datos en el servidor...\n");
            exit(0);
        }
        else
            printf("El servidor acepta al cliente ...\n");
        int pid = fork();
        if (pid == 0)
            func(connfd);
    }

    close(sockfd);
}
```

Cliente

```c
#include <arpa/inet.h> // inet_addr()
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h> // bzero()
#include <sys/socket.h>
#include <unistd.h> // read(), write(), close()
#define MAX 80
#define PORT 8080
#define SA struct sockaddr
void func(int sockfd)
{
    char buff[MAX];
    int n;
    for (;;) {
        bzero(buff, sizeof(buff));
        printf("Ingrese texto: ");
        n = 0;
        while ((buff[n++] = getchar()) != '\n');
        write(sockfd, buff, sizeof(buff));
        bzero(buff, sizeof(buff));
        read(sockfd, buff, sizeof(buff));
        printf("Servidor : %s", buff);
        if ((strncmp(buff, "SALIR", 4)) == 0) {
            printf("Cliente cierra conexión...\n");
            break;
        }
    }
}

int main()
{
    int sockfd, connfd;
    struct sockaddr_in servaddr, cli;

    // socket: creo socket y lo verifico
    sockfd = socket(AF_INET, SOCK_STREAM, 0); //
    if (sockfd == -1) {
        printf("Falla la creación del socket...\n");
        exit(0);
    }
    else
        printf("Socket creado ..\n");
    bzero(&servaddr, sizeof(servaddr));

    // asigno IP, PORT
```

```c
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
    servaddr.sin_port = htons(8080);

    // Conecto los sockets entre cliente y servidor
    if (connect(sockfd, (SA*)&servaddr, sizeof(servaddr)) != 0) {
        printf("Falla de conexión con servidor...\n");
        exit(0);
    }
    else
        printf("Conectado al servidor..\n");

    // Función para el chat
    func(sockfd);

    //Cierro el socket
    close(sockfd);
}
```

```
tomastobio@DESKTOP-2UT9IVO:/mnt/c/Users/Usuario/Documents/GitHub/trabajo-practic
o-1-grupo-4/TP1$ gcc servidor_concurrente_real.c -o server
tomastobio@DESKTOP-2UT9IVO:/mnt/c/Users/Usuario/Documents/GitHub/trabajo-practic
o-1-grupo-4/TP1$ ./server
Socket creado...
Se hace el socket bind ..
Servidor en modo escucha ...
El servidor acepta al cliente ...
Servidor en modo escucha ...
Del cliente: Buenas tardes
        : Buenas noches
Del cliente: Mensaje 2
        : Respuesta 2
Del cliente: Gracias
        : SALIR
```

```
tomastobio@DESKTOP-2UT9IVO:/mnt/c/Users/Usuario/Documents/Github/trabajo-practic
o-1-grupo-4/TP1$ ./client1
Socket creado ..
Conectado al servidor..
Ingrese texto: Buenas tardes
Servidor : Buenas noches
Ingrese texto: Mensaje 2
Servidor : Respuesta 2
Ingrese texto: Gracias
Servidor : SALIR
Cliente cierra conexión...
```