Actividad Sockets

DESARROLLO EN SISTEMAS DISTRIBUIDOS

Link de repositorio en Github

Desarrollo

La actividad consiste en el desarrollo de un servidor y un cliente en C++ y un cliente en un lenguaje a elegir, yo elegí NodeJS.

La estructura del servidor es la siguiente:

```
int main() {
    WSADATA wsa;
    SOCKET serv_sock, client_sock;
    struct sockaddr_in server, client;
    int client_len, recv_len;
    char buffer[1024] = {0};
    int isEjecutando = 1;

    srand(time(NULL));

    printf("Inicializando Winsock...\n");
    if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0) {
        printf("Error en WSAStartup. Código de error: %d\n", WSAGetLastError())
        return 1;
    }
```

Al iniciar la ejecución se realizan distintas configuraciones, se invoca el DLL de ws2_32.dll mediante el WSAStartup enviando por parámetro la versión y el objeto de WSADATA. Esto es procedimiento estándar según la documentación oficial de Winsock. Si es exitoso, retorna 0. Se instancian también los objetos del socket del servidor y del cliente a recibir.

```
if ((sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) == INVALID_SOCKET) {
    printf("Error al crear el socket. Código de error: %d\n", WSAGetLastError());
    WSACleanup();
    return 1;
}
```

- **AF_INET** se usa para especificar la familia de direcciones IPv4.
- **SOCK_STREAM** se usa para especificar un socket de flujo.
- **IPPROTO_TCP** se usa para especificar el protocolo TCP.

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

    if (bind(serv_sock, (struct sockaddr*)&server, sizeof(server)) == SOCKET_ERROR) {
        printf("Error en bind. Código de error: %d\n", WSAGetLastError());
        closesocket(serv_sock);
        WSACleanup();
        return 1;
    }
```

Se establece “127.0.0.1” como dirección, es decir, localhost. Adress Family INET (IPv4) y se configura el puerto 8080 usando htons. La función bind se usa para enlazar a sockets orientados a la conexión o sockets sin conexión.

```
listen(serv_sock, MAX_CLIENTS);
printf("Servidor iniciado en el puerto %d\n", PORT);
client_len = sizeof(struct sockaddr_in);
```

Esta parte se explica sola. Hace que el socket escuche conexiones entrantes. *MAX_CLIENTS* indica cuántas conexiones pendientes puede tener. Luego de esto sigue el bucle principal del servidor, aceptando conexiones entrantes y enviando mensajes al cliente.

La estructura del cliente es similar, salvo que este en vez de usar bind, usa connect.

```
if (connect(sock, (struct sockaddr*)&server, sizeof(server)) < 0) {
    printf("Error en la conexión\n");
    closesocket(sock);
    WSACleanup();
    return 1;
}

while (isEjecutando) {
    printf("Menú:\n1. Generar Nombre de Usuario\n2. Generar Contraseña\n3. Salir\nIntroduce tu opción: ");
    fgets(option, sizeof(option), stdin);
    option[strcspn(option, "\n")] = 0;

    if (strcmp(option, "1") == 0) {
        printf("Introduce la longitud del nombre de usuario (5-15): ");
        fgets(length_str, sizeof(length_str), stdin);
        send(sock, "username", strlen("username"), 0);
        send(sock, length_str, strlen(length_str), 0);
    } else if (strcmp(option, "2") == 0) {
        printf("Introduce la longitud de la contraseña (8-50): ");
        fgets(length_str, sizeof(length_str), stdin);
        send(sock, "password", strlen("password"), 0);
        send(sock, length_str, strlen(length_str), 0);
    } else if (strcmp(option, "3") == 0) {
        send(sock, "exit", strlen("exit"), 0);
        isEjecutando = 0;
    } else {
        printf("Opción inválida, por favor intenta de nuevo.\n");
        continue;
    }

    recv(sock, buffer, sizeof(buffer), 0);
    printf("Respuesta del servidor: %s\n", buffer);
}

closesocket(sock);
WSACleanup();

return 0;
```

- **closesocket(socket):** Cierra el socket después de terminar la comunicación.
- **WSACleanup():** Limpia la API de Winsock y libera los recursos asignados.

En NodeJS:

```
const client = new net.Socket();

client.connect(PORT, HOST, () => {
    console.log('Conectado al servidor en ' + HOST + ':' + PORT);
    rl.question('Menú:\n1. Generar Nombre de Usuario\n2. Generar Contraseña\nIntroduce tu opción: ', (option) => {
        if (option === '1') {
            rl.question('Introduce la longitud del nombre de usuario (5-15): ', (length) => {
                client.write('username');
                client.write(length);
            });
        } else if (option === '2') {
            rl.question('Introduce la longitud de la contraseña (8-50): ', (length) => {
                client.write('password');
                client.write(length);
            });
        }
    });
```

Se instancia el socket, se declara el puerto, con client.write() se escribe al servidor y con cliente.on() se manejan eventos provenientes del servidor.