```cpp
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <string>
#include <ctime>
#include <iomanip>
#include <algorithm>
#include <winsock2.h>

bool recibirDatos(SOCKET clientSocket, char* buffer, int bufferSize);

std::string leerArchivo(const std::string& nombreArchivo);

std::pair<bool, std::string> verificarCredenciales(const std::string& usuario, const std::string& contrasena);

void registrarLog(const std::string& mensaje);

std::string fechaYHora();

std::string traduccion(SOCKET clientSocket);

std::string buscarTraduccion(const std::string& palabra);

std::string nuevaTraduccion(SOCKET clientSocket);  // Esta es para PEDIR al cliente una nueva traducción

bool agregarNuevaTraduccion(const std::string& nuevaTraduccion); // Esta es para METER EN EL ARCHIVO la nueva traducción

std::string verRegistroActividades(SOCKET clientSocket); // Manda por varios buffer el registro completo

std::string administrarUsuarios(SOCKET clientSocket);

bool registrarNuevoUsuario(const std::string& usuario, const std::string& contrasena, const std::string& rol, int intentos);

bool usuarioExiste(const std::string& usuario); // Función para verificar si un usuario ya está registrado

std::string obtenerUsuariosBloqueados(); // Devuelve un string con los usuarios que estan bloqueados

std::string desbloquearUsuario(const std::string& usuario); // le asigna 0 intentos a un usuario con mas de 3

std::string menuConsulta();

std::string menuAdmin();

std::string mostrarMenu(std::string rol);

int main() {
    std::string ip /*= "192.168.1.34"*/;
    int puerto /*= 5005*/;
    std::cout << "Ingrese la dirección IP del servidor: ";
    std::cin >> ip;
    std::cout << "Ingrese el puerto del servidor: ";
    std::cin >> puerto;
    std::cin.ignore();

    std::string mensajeLog="";
    std::string response;
    WSADATA wsData;
    if (WSAStartup(MAKEWORD(2, 2), &wsData) != 0) {
        std::cerr << "Error al inicializar Winsock" << std::endl;
        return -1;
    }

    SOCKET serverSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (serverSocket == INVALID_SOCKET) {
        std::cerr << "Error al crear el socket del servidor" << std::endl;
        WSACleanup();
        return -1;
    }

    sockaddr_in serverAddr;
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(puerto);
    serverAddr.sin_addr.s_addr = inet_addr(ip.c_str());


```

```
77      int reuseAddr = 1;
78      if (setsockopt(serverSocket, SOL_SOCKET, SO_REUSEADDR, reinterpret_cast<const
    char*>(&reuseAddr), sizeof(reuseAddr)) == SOCKET_ERROR) {
79          std::cerr << "Error al establecer SO_REUSEADDR" << std::endl;
80          closesocket(serverSocket);
81          WSACleanup();
82          return -1;
83      }
84
85      if (bind(serverSocket, (sockaddr*)&serverAddr, sizeof(serverAddr)) == SOCKET_ERROR) {
86          std::cerr << "Error al vincular el socket a la direccion" << WSAGetLastError() <<
    std::endl;
87          closesocket(serverSocket);
88          WSACleanup();
89          return -1;
90      }
91
92      // Obtener el puerto de escucha
93      sockaddr_in boundAddr;
94      int boundAddrLen = sizeof(boundAddr);
95      if (getsockname(serverSocket, (sockaddr*)&boundAddr, &boundAddrLen) == SOCKET_ERROR) {
96          std::cerr << "Error al obtener el puerto de escucha" << WSAGetLastError() <<
    std::endl;
97          closesocket(serverSocket);
98          WSACleanup();
99          return -1;
100     }
101
102     unsigned short puertoDeEscucha = ntohs(boundAddr.sin_port);
103
104     if (listen(serverSocket, 5) == SOCKET_ERROR) {
105         std::cerr << "Error al escuchar por conexiones entrantes" << std::endl;
106         closesocket(serverSocket);
107         WSACleanup();
108         return -1;
109     }
110     mensajeLog = "============================\n =======Inicia
    Servidor=======\n============================";
111     registrarLog(mensajeLog);
112
113     std::cout << "Esperando conexiones entrantes..." << std::endl;
114     // Acá guardo el puerto de escucha
115     mensajeLog = fechaYHora() + ": Socket creado. Puerto de escucha: " +
    std::to_string(puertoDeEscucha);
116     registrarLog(mensajeLog);
117
118     while (true) { // Bucle externo para esperar nuevas conexiones
119         sockaddr_in clientAddr;
120         int clientAddrSize = sizeof(clientAddr);
121         SOCKET clientSocket = accept(serverSocket, (sockaddr*)&clientAddr, &clientAddrSize);
122         if (clientSocket == INVALID_SOCKET) {
123             std::cerr << "Error al aceptar la conexion entrante" << std::endl;
124             closesocket(serverSocket);
125             WSACleanup();
126             return -1;
127         }
128
129         std::cout << "Cliente conectado" << std::endl;
130
131         bool credencialesPedidas = false; // Creo el boolean para que las credenciales no
    esten en el while
132         std::string usuario;
133         std::string rolUsuario;
134         char buffer[1024];
135         while (true) {
136
137             if (!recibirDatos(clientSocket, buffer, sizeof(buffer))) {
138                 break; // Manejar error o desconexión del cliente
139             }
140
141             std::cout << "Cliente: " << buffer << std::endl;
142             std::string mensajeCliente(buffer);
143
144             if (!credencialesPedidas) { ///ESTE IF ES PARA LA PRIMERA VEZ, O SEA PARA
    INICIAR SESION, SU ELSE ES PARA ELEGIR OPCIONES
145             // Parsear el mensaje para obtener usuario y contraseña
146                 size_t pos = mensajeCliente.find('|');
147                 if (pos == std::string::npos) {
148                     std::cerr << "Mensaje invalido del cliente" << std::endl;
149                     continue;
150                 }
151                 usuario = mensajeCliente.substr(0, pos);
152                 std::string contrasena = mensajeCliente.substr(pos + 1);
153
```

```
                // Verificar las credenciales

                std::pair<bool, std::string> credenciales = verificarCredenciales(usuario, contrasena);
                rolUsuario = credenciales.second; // Esto es para filtrar las opciones según el rol

                if (credenciales.first) {
                    response = "Acceso concedido como " + credenciales.second + mostrarMenu(rolUsuario);
                    credencialesPedidas = true;
                    mensajeLog = fechaYHora() + ": Inicio de sesion - usuario: " + usuario;  // Esto va a registro.log
                    registrarLog(mensajeLog);
                } else {
                    response = credenciales.second;
                    send(clientSocket, response.c_str(), response.size(), 0);
                    break;
                }
                send(clientSocket, response.c_str(), response.size(), 0);
            } else { ///ACA SE ELIGEN OPCIONES Y SE HACEN LAS FUNCIONES NECESARIAS
                if (mensajeCliente == "1") { // Traducir palabra que recibe el servidor
                    response = traduccion(clientSocket) + mostrarMenu(rolUsuario);
                } else if (mensajeCliente == "2" && rolUsuario == "ADMIN") { // Agregar una traducción
                    response = nuevaTraduccion(clientSocket) + mostrarMenu(rolUsuario);
                } else if (mensajeCliente == "3" && rolUsuario == "ADMIN") { // Usuarios
                    response = administrarUsuarios(clientSocket) + mostrarMenu(rolUsuario);
                } else if (mensajeCliente == "4" && rolUsuario == "ADMIN") { // Ver registro de actividades
                    response = verRegistroActividades(clientSocket) + mostrarMenu(rolUsuario);
                } else if (mensajeCliente == "5") { // Cerrar Sesión
                    break; // Aqui cierra sesión
                } else { // Opción no válida
                    response = "Opcion no valida" + mostrarMenu(rolUsuario);
                }
                //response = "Mensaje recibido por el servidor";
                send(clientSocket, response.c_str(), response.size(), 0);
            }
            //send(clientSocket, response.c_str(), response.size(), 0);
        }
        mensajeLog = fechaYHora() + ": Cierre de sesion - usuario: " + usuario;
        registrarLog(mensajeLog);
        closesocket(clientSocket); // Acá cierro el socket dentro del while, asi puedo...

        std::cout << "Esperando nuevas conexiones..." << std::endl; // Volver a esperar nuevas conexiones
    }

    closesocket(serverSocket);
    WSACleanup();

    return 0;
}
// Funcion para recibir datos del cliente
bool recibirDatos(SOCKET clientSocket, char* buffer, int bufferSize) {
    memset(buffer, 0, bufferSize);
    int bytesReceived = recv(clientSocket, buffer, bufferSize - 1, 0);
    if (bytesReceived == SOCKET_ERROR) {
        std::cerr << "Error al recibir datos del cliente" << std::endl;
        return false;
    } else if (bytesReceived == 0) {
        std::cout << "Cliente desconectado" << std::endl;
        return false;
    }
    return true;
}
// Funcion para leer un archivo
std::string leerArchivo(const std::string& nombreArchivo) { // Le mando por parámetro el nombre del archivo
    std::ifstream archivo(nombreArchivo); // Leer archivo
    std::string contenido; // String para cargar los datos (ACA VAN A ESTAR LOS STRINGS PARA COMPARAR)

    if (archivo.is_open()) { // Si el archivo está abierto
        std::string linea; // Línea temporal
        while (std::getline(archivo, linea)) { // Acá igualo la línea temporal a la siguiente línea del archivo (SE REEMPLAZA)
            contenido += linea + "\n"; // Le meto al string para cargar datos la línea temporal
        }
        archivo.close();
    } else {
```

```
        std::cerr << "Error al abrir el archivo" << std::endl;
    }

    return contenido;
}
// Retorna un boolean para verificar si se aceptan las credenciales, y otro con el rol del usuario para los menús
std::pair<bool, std::string> verificarCredenciales(const std::string& usuario, const std::string& contrasena) {
    std::string contenidoArchivo = leerArchivo("credenciales.txt");

    std::istringstream archivoStream(contenidoArchivo); // Acá guardo las credenciales para estructurar funciones
    std::string linea;
    std::vector<std::string> lineas;

    bool usuarioBloqueado = false;
    bool usuarioEncontrado = false;
    bool accesoConcedido = false;

    std::string rolArchivoReturn = "";// Declaro rol para que me lo tome el return

    while (std::getline(archivoStream, linea)) { // Llamo una línea de las credenciales
        std::istringstream lineaStream(linea); // Lunciones para la línea
        std::string usuarioArchivo, contrasenaArchivo, rolArchivo, intentosArchivo; // Strings usuario|contraseña|rol|intentos
        std::getline(lineaStream, usuarioArchivo, '|');
        std::getline(lineaStream, contrasenaArchivo, '|');
        std::getline(lineaStream, rolArchivo, '|');
        std::getline(lineaStream, intentosArchivo);
        if (usuarioArchivo == usuario) {
            usuarioEncontrado = true;
            int intentos = std::stoi(intentosArchivo); // Convertir el valor a entero

            if (intentos >= 3) {
                usuarioBloqueado = true;
                rolArchivoReturn = "usuario bloqueado";
            } else if (contrasenaArchivo == contrasena) {
                intentos = 0;
                accesoConcedido = true;
                rolArchivoReturn = rolArchivo;
            } else if(intentos == 2){
                rolArchivoReturn = "Datos de usuario incorrectos\nUSUARIO BLOQUEADO";
                intentos++; /// Acá está el control de veces que se logeó mal
            } else {
                rolArchivoReturn = "Datos de usuario incorrectos";
                intentos++;
            }

            linea = usuarioArchivo + "|" + contrasenaArchivo + "|" + rolArchivo + "|" + std::to_string(intentos); // Paso a int intentos
        }

        lineas.push_back(linea); // Agregar la línea a la lista
    }

    if (!usuarioEncontrado) {
        std::cout << "Usuario no encontrado" << std::endl;
    }

    if (!usuarioBloqueado && !accesoConcedido) {
        std::ofstream archivo("credenciales.txt"); // Escribimos en el archivo
        for (const std::string& lineaModificada : lineas) { // Recorro cada línea modificada
            archivo << lineaModificada << "\n"; // La imprimo en el archivo
        }
    }
    // Devolver un par (bool, string) donde el bool indica si las credenciales son válidas
    // y el string contiene el rol del usuario
    return std::make_pair(usuarioEncontrado && accesoConcedido, rolArchivoReturn);
}
// Esto es para guardar en el archivo de registro
void registrarLog(const std::string& mensaje) {
    std::ofstream archivoLog("registro.log", std::ios::app); // Abre el archivo en modo de adjuntar
    if (archivoLog.is_open()) {
        archivoLog << mensaje << std::endl; // Escribe el mensaje en el archivo log
        archivoLog.close(); // Cierra el archivo
    } else {
        std::cerr << "Error al abrir el archivo de registro.log" << std::endl;
    }
}
// Esto devuelve la fecha y la hora actual en formato string
std::string fechaYHora(){
    // Obtener la fecha y la hora actual del sistema
```

```
    std::time_t tiempoActual = std::time(nullptr);
    std::tm* tiempoInfo = std::localtime(&tiempoActual);

    // Formatear la fecha y la hora como una cadena
    std::stringstream ss;
    ss << std::put_time(tiempoInfo, "%Y-%m-%d %H:%M");
    std::string fechaYHora = ss.str();

    return fechaYHora;
}
/// Opción 1: Traducir
std::string traduccion(SOCKET clientSocket) {
    // Puedes recibir datos adicionales del cliente y enviar respuestas aquí
    char buffer[1024];
    // Pedir la palabra del cliente
    send(clientSocket, "Ingrese una palabra en ingles ", strlen("Ingrese una palabra en ingles "), 0);

    // Recibe la palabra en inglés del cliente
    if (!recibirDatos(clientSocket, buffer, sizeof(buffer))) {
        return "ERROR al recibir datos";
    }
    // Ahora con la palabra en mano, la mando en la funcion de buscar traduccion
    std::string palabra=buffer;
    if (palabra == "/salir") return "";

    // Convertir la palabra a minúsculas
    std::transform(palabra.begin(), palabra.end(), palabra.begin(), ::tolower);


    // La traducción no existe en el archivo
    std::string resultado = buscarTraduccion(palabra);
    if (resultado.empty()) {
        return "No fue posible encontrar la traducción para: " + palabra;
    }

    // Realiza la traducción y envía la respuesta al cliente
    return resultado;
}
// Retorna la palabra en español enviándole la palabra en inglés por parametro (si no encuentra retorna error)
std::string buscarTraduccion(const std::string& palabra) {
    const std::string archivoTraducciones = "traducciones.txt";

    std::ifstream archivo(archivoTraducciones); // Abro el archivo en modo lectura

    if (!archivo.is_open()) {
        return "Error: No se pudo abrir el archivo";
    }

    std::string linea;
    while (std::getline(archivo, linea)) {
        std::istringstream ss(linea);
        std::string palabraEnIngles, traduccion;

        if (std::getline(ss, palabraEnIngles, ':') && std::getline(ss, traduccion)) {
            // Eliminar espacios en blanco al principio y al final de las cadenas
            palabraEnIngles.erase(0, palabraEnIngles.find_first_not_of(" \t\r\n"));
            palabraEnIngles.erase(palabraEnIngles.find_last_not_of(" \t\r\n") + 1);

            traduccion.erase(0, traduccion.find_first_not_of(" \t\r\n"));
            traduccion.erase(traduccion.find_last_not_of(" \t\r\n") + 1);
            std::cout << palabra << std::endl;
            // Comparo con la palabra qwue mandé por parametro
            if (palabraEnIngles == palabra) {
                return palabra + " en ingles es " + traduccion + " en español.";
            }
        }
    }

    return "No se encontro traduccion para la palabra " + palabra;
}
/// Opción 2: Agregar traducción
std::string nuevaTraduccion(SOCKET clientSocket) {
    char buffer[1024];
    // Pedir la nueva traducción al cliente
    send(clientSocket, "Nueva traduccion palabraEnIngles:traduccionEnEspanol: ", strlen("Nueva traduccion palabraEnIngles:traduccionEnEspanol: "), 0);

    if (!recibirDatos(clientSocket, buffer, sizeof(buffer))) {
        return "ERROR al recibir datos";
    }

    std::string nuevaTraduccion=buffer; // Ejemplo: dog:perro
```

```
385     if (nuevaTraduccion == "/salir") return "";
386 
387     // Verificar el formato de inserción
388     size_t pos = nuevaTraduccion.find(':');
389     if (pos == std::string::npos || pos == 0 || pos == nuevaTraduccion.length() - 1) {
390         return "No fue posible insertar la traduccion. El formato de insercion debe ser palabraEnIngles:traduccionEnEspanol";
391     }
392 
393     std::string palabraEnIngles = nuevaTraduccion.substr(0, pos);
394     std::string traduccionEnEspanol = nuevaTraduccion.substr(pos + 1); // Separo las palabras para buscarlas en el archivo
395 
396     // Verificar si la traducción ya existe
397     if (buscarTraduccion(palabraEnIngles) == traduccionEnEspanol) {
398         return "Ya existe una traducción para " + palabraEnIngles + ": " + traduccionEnEspanol;
399     }
400     if (agregarNuevaTraduccion(nuevaTraduccion)) {
401         return "Nueva traduccion insertada correctamente";
402     } else {
403         return "Error al insertar la nueva traduccion";
404     }
405 }
406 // Función para agregar una traducción al archivo de traducciones
407 bool agregarNuevaTraduccion(const std::string& nuevaTraduccion) {
408     std::ofstream archivoTraducciones("traducciones.txt", std::ios::app);
409 
410     if (!archivoTraducciones) {
411         return false;
412     }
413     std::string aMinuscula = nuevaTraduccion;
414     std::transform(aMinuscula.begin(), aMinuscula.end(), aMinuscula.begin(), ::tolower);
415     archivoTraducciones << aMinuscula << std::endl;
416 
417     archivoTraducciones.close();
418 
419     return true;
420 }
421 /// Opción 4: Lee el archivo de registro y lo retorna como varios string de 1024 caracteres
422 std::string verRegistroActividades(SOCKET clientSocket) {
423     std::ifstream archivoRegistro("registro.log");
424     if (!archivoRegistro) {
425         return "ERROR: no hay archivo de registro";
426     }
427 
428     std::string linea;
429     std::string contenidoRegistro;
430     while (std::getline(archivoRegistro, linea)) { // Acá voy agregando al string final cada linea que lee el getline
431         contenidoRegistro += linea + '\n';
432     }
433     archivoRegistro.close();
434     char buffer[1024];
435     std::string tempContenidoRegistro;
436     while (contenidoRegistro.size()>1023){ // Mientras el string final sea mayor al tamaño del buffer, le resto los primeros 1024 char
437         tempContenidoRegistro = contenidoRegistro.substr(0, 1024); // Le agrego el $ para decirle al cliente que aún hay que mostrar más registro
438         contenidoRegistro = contenidoRegistro.substr(1024);
439         send(clientSocket, tempContenidoRegistro.c_str(), tempContenidoRegistro.size(), 0);
440         recibirDatos(clientSocket, buffer, sizeof(buffer));
441     }
442     return contenidoRegistro;
443 }
444 /// Opción 3: Submenú de usuarios
445 std::string administrarUsuarios(SOCKET clientSocket) {
446     char buffer[1024];
447     send(clientSocket, "\nUsuarios:\na. Alta\nb. Desbloqueo\n", strlen("\nUsuarios:\na. Alta\nb. Desbloqueo\n"), 0);
448 
449     if (!recibirDatos(clientSocket, buffer, sizeof(buffer))) {
450         return "ERROR al recibir datos"; // Manejar error o desconexión del cliente
451     }
452     std::string opcion(buffer); // "a" o "b"
453     if (opcion == "/salir") return "";
454 
455     if (opcion == "a") { // Alta
456         send(clientSocket, "Ingrese el nombre de usuario y contrasena separados por '|' (Ejemplo: usuario|contrasena): ", strlen("Ingrese el nombre de usuario y contraseña separados por '|' (Ejemplo: usuario|contraseña): "), 0);
457 
458         if (!recibirDatos(clientSocket, buffer, sizeof(buffer))) {
459             return "ERROR al recibir datos"; // Manejar error o desconexión del cliente
```

```
        }

        std::string datosUsuario(buffer);
        if (datosUsuario == "/salir") return "";
        size_t pos = datosUsuario.find('|');
        if (pos == std::string::npos) {
            return "Error al dar de alta el nuevo usuario: datos incompletos";
        }

        std::string nuevoUsuario = datosUsuario.substr(0, pos);
        std::string nuevaContrasena = datosUsuario.substr(pos + 1);
        std::string mensajeError = "";

        if (usuarioExiste(nuevoUsuario)){
            mensajeError = "Error: El usuario '" + nuevoUsuario + "' ya existe.";
            return mensajeError;
        } else if (nuevoUsuario == "" || nuevaContrasena == ""){
            mensajeError = "Error: Campos ingresados incorrectamente";
            return mensajeError;
        } else {
            // Registrar el nuevo usuario, establecer intentos a 0, rol a CONSULTA
            registrarNuevoUsuario(nuevoUsuario, nuevaContrasena, "CONSULTA", 0);
            return "Usuario agregado con exito";

        }
    } else if (opcion == "b") { // Desbloqueo de Usuario
        // Aquí debes implementar la lógica para mostrar los usuarios bloqueados y permitir al cliente seleccionar uno para desbloquear
        std::string usuariosBloqueados = obtenerUsuariosBloqueados();
        if (usuariosBloqueados.empty()) {
            return "No se encontraron usuarios bloqueados";
        }
        std::string listaDeUsuarios = "\nUsuarios Bloqueados: " + usuariosBloqueados + "\nIngrese el nombre de usuario a desbloquear: ";
        send(clientSocket, listaDeUsuarios.c_str(), listaDeUsuarios.size(), 0);

        if (!recibirDatos(clientSocket, buffer, sizeof(buffer))) {
            return "ERROR al recibir datos"; // Manejar error o desconexión del cliente
        }

        std::string usuarioADesbloquear(buffer);
        if (usuarioADesbloquear == "/salir") return "";

        // Realizar la lógica de desbloqueo del usuario
        std::string desbloqueado = desbloquearUsuario(usuarioADesbloquear);

        if (desbloqueado=="Ok") {
            return usuarioADesbloquear + " desbloqueado correctamente";
        } else if (desbloqueado == "File"){
            return "Error al desbloquear el usuario";
        } else if (desbloqueado == "No"){
            return "El usuario no se encuentra bloqueado";
        }
    } else {
        return "Opción no válida";
    }
}

// Función para registrar un nuevo usuario en el archivo
bool registrarNuevoUsuario(const std::string& usuario, const std::string& contrasena, const std::string& rol, int intentos) {
    // Abre el archivo en modo de escritura (appended)
    std::ofstream archivoCredenciales("credenciales.txt", std::ios::app);
    if (!archivoCredenciales) {
        return false; // No se pudo abrir el archivo
    }

    // Escribe el nuevo usuario en el archivo con el formato deseado
    archivoCredenciales << usuario << "|" << contrasena << "|" << rol << "|" << intentos << std::endl;

    // Cierra el archivo
    archivoCredenciales.close();

    // El usuario se registró correctamente
    return true;
}

// Función para verificar si un usuario ya existe en el archivo
bool usuarioExiste(const std::string& usuario) {
    // Abre el archivo en modo de lectura
    std::ifstream archivoCredenciales("credenciales.txt");
    if (!archivoCredenciales) {
        return false; // No se pudo abrir el archivo
```

```
    }

    std::string linea;
    while (std::getline(archivoCredenciales, linea)) {
        size_t pos = linea.find('|');
        if (pos != std::string::npos) {
            std::string usuarioExistente = linea.substr(0, pos);
            if (usuarioExistente == usuario) {
                archivoCredenciales.close();
                return true; // El usuario ya existe en el archivo
            }
        }
    }

    archivoCredenciales.close();
    // El usuario no fue encontrado en el archivo
    return false;
}
// Devuelve los usuariosBloqueados, si no hay, devuelve un error
std::string obtenerUsuariosBloqueados() {
    std::ifstream archivoCredenciales("credenciales.txt");
    if (!archivoCredenciales) {
        return "Error: No se pudo abrir el archivo de credenciales";
    }

    std::string usuariosBloqueados;
    std::string linea;
    while (std::getline(archivoCredenciales, linea)) {
        size_t pos = linea.find('|');
        if (pos != std::string::npos) {
            std::string usuario = linea.substr(0, pos);
            // Obtener intentos como entero
            int intentos = std::stoi(linea.substr(linea.find_last_of('|') + 1));
            // Si el usuario tiene 3 intentos o más, considerarlo bloqueado
            if (intentos >= 3) {
                if (!usuariosBloqueados.empty()) {
                    usuariosBloqueados += ", ";
                }
                usuariosBloqueados += usuario;
            }
        }
    }

    archivoCredenciales.close();

    if (usuariosBloqueados.empty()) {
        return "";
    }

    return usuariosBloqueados;
}
// Retorna mensaje de que se logró desbloquear al usuario/no esta bloqueado/error
std::string desbloquearUsuario(const std::string& usuario) {
    std::ifstream archivoEntrada("credenciales.txt");
    if (!archivoEntrada) {
        return "File"; // No se pudo abrir el archivo
    }
    std::string retorno = "";
    std::vector<std::string> lineas;
    std::string linea;
    std::string ultimoString = "";
    int flag = 0;
    while (std::getline(archivoEntrada, linea)) {
        size_t pos = linea.find('|');
        if (pos != std::string::npos) { // Si se encontró el '|'
            std::string nombreUsuario = linea.substr(0, pos);

            if (nombreUsuario == usuario) {
                // Encontramos el usuario, ahora lo modificamos para establecer intentos a 0
                size_t intentos = linea.find_last_of('|'); // Busco la posición del último
'|', significa que lo que siga despues van a ser los intentos
                ultimoString = linea.back();
                if (intentos != std::string::npos && ultimoString == "3") { // Si se
encontró el nuevo '|' y el ultimo caracter es 3
                    // Encuentra la posición del último '|'
                    linea.replace(intentos + 1, std::string::npos, "0");
                    flag++;
                }
            }
        }
        lineas.push_back(linea);
    }

    archivoEntrada.close();
```

```
    if (flag == 0){
        return "No";
    }
    // Volver a escribir todas las líneas en el archivo
    std::ofstream archivoSalida("credenciales.txt");
    if (!archivoSalida) {
        return "File"; // No se pudo abrir el archivo
    }

    for (const std::string& linea : lineas) {
        archivoSalida << linea << '\n';
    }

    archivoSalida.close();

    return "Ok";
}
// Menús
std::string menuConsulta() {
    std::string menu;
    menu += "\n";
    menu += "Menu de Consulta:\n";
    menu += "1. Traducir\n";
    menu += "5. Cerrar Sesion\n";
    return menu;
}

std::string menuAdmin() {
    std::string menu;
    menu += "\n";
    menu += "Menu de Administrador:\n";
    menu += "1. Traducir\n";
    menu += "2. Nueva Traduccion\n";
    menu += "3. Usuarios:\n";
    menu += "   a. Alta\n";
    menu += "   b. Desbloqueo\n";
    menu += "4. Ver Registro de Actividades\n";
    menu += "5. Cerrar Sesion\n";
    return menu;
}

std::string mostrarMenu(std::string rol){
    std::string retorno="\n";
    if (rol == "ADMIN"){
        retorno += menuAdmin();
    } else if (rol == "CONSULTA"){
        retorno += menuConsulta();
    }
    return retorno;
}

```