Chat Soporte Tecnico

Version Final 1.0

Generated by Doxygen 1.14.0

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

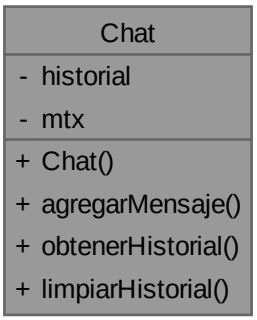
Class Documentation

3.1 Chat Class Reference

Clase gestora del historial de la conversación.

`#include <chat.h>`

Collaboration diagram for Chat:



Public Member Functions

- Chat ()

  Constructor por defecto. Inicializa un historial vacío.
- void agregarMensaje (std::string emisor, std::string texto, bool esMio)

  Inserta un nuevo mensaje en el historial de forma segura.
- std::vector< Mensaje > obtenerHistorial ()

  Obtiene una COPIA del historial actual para ser dibujada.
- void limpiarHistorial ()

  Borra todos los mensajes almacenados.

Private Attributes

- std::vector< Mensaje > historial

  Contenedor dinámico de mensajes.
- std::mutex mtx

  Semáforo de exclusión mutua.

3.1.1 Detailed Description

Clase gestora del historial de la conversación.

- Esta clase actúa como un recurso compartido entre:

1. El Hilo de Red (que escribe mensajes recibidos).
2. El Hilo Gráfico/Main (que lee mensajes para dibujarlos).

- Utiliza un Mutex para evitar Condiciones de Carrera (Race Conditions).

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Chat()

```
Chat::Chat () [inline]
```

Constructor por defecto. Inicializa un historial vacío.

3.1.3 Member Function Documentation

3.1.3.1 agregarMensaje()

```
void Chat::agregarMensaje (
            std::string emisor,
            std::string texto,
            bool esMio)
```

Inserta un nuevo mensaje en el historial de forma segura.

Agrega un mensaje al vector de forma segura (Thread-Safe).

Parameters

emisor	El nombre de quien envía.
texto	El contenido del mensaje.
esMio	Define si el mensaje lo escribí yo (true) o me llegó (false).

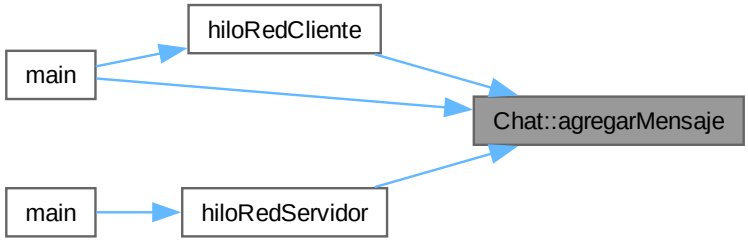
- Utiliza el patrón RAII para el bloqueo: El mutex se bloquea al crear 'lock' y se desbloquea AUTOMÁ↩ TICAMENTE cuando la función termina (se cierra la llave).

Parameters

emisor	Quién envía el mensaje.
texto	Contenido del mensaje.
esMio	Booleano para determinar el color de la burbuja en la UI.

-

Here is the caller graph for this function:



3.1.3.2 limpiarHistorial()

```
void Chat::limpiarHistorial ()
```

Borra todos los mensajes almacenados.

Vacia el vector de mensajes.

- Utilizado cuando se cambia de cliente o se cierra una sesión, para asegurar que el siguiente usuario empiece con la pantalla limpia.
- Se utiliza al cambiar de cliente para limpiar la pantalla.

Here is the caller graph for this function:

3.1.3.3 obtenerHistorial()

```
std::vector< Mensaje > Chat::obtenerHistorial ()
```

Obtiene una COPIA del historial actual para ser dibujada.

Devuelve una copia instantánea del chat actual.

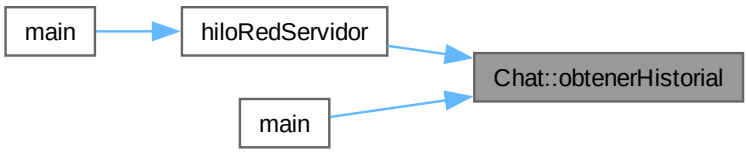
Returns

std::vector<Mensaje> Una copia segura de los mensajes.

- Se devuelve por valor (copia) y no por referencia para que el hilo gráfico pueda dibujar tranquilo sin bloquear al hilo de red por mucho tiempo.
- Es necesario bloquear también para LEER, porque si el vector se redimensiona (realloc) mientras lo leemos, el programa crashearía.
-

std::vector<Mensaje> Copia completa del historial.

Here is the caller graph for this function:



3.1.4 Member Data Documentation

3.1.4.1 historial

```
std::vector<Mensaje> Chat::historial  [private]
```

Contenedor dinámico de mensajes.

- Se eligió std::vector por sobre std::list o arreglos fijos porque:

1. Permite acceso rápido e iteración secuencial (ideal para dibujar 60 FPS).
2. Crece dinámicamente según sea necesario.

3.1.4.2 mtx

```
std::mutex Chat::mtx  [private]
```

Semáforo de exclusión mutua.

- Este objeto es CRITICO. Bloquea el acceso al vector mientras un hilo está escribiendo, obligando al otro a esperar. Sin esto, el programa crashearía al intentar leer y escribir memoria simultáneamente.

The documentation for this class was generated from the following files:

- include/chat.h
- src/chat.cpp

3.2 ClienteSocket Class Reference

Clase que representa al cliente en la arquitectura Cliente-Servidor.

```
#include <clienteSocket.h>
```

Collaboration diagram for ClienteSocket:

Public Member Functions

- ClienteSocket ()

  Constructor. Inicializa las variables en un estado seguro/nulo.
- ~ClienteSocket ()

  Destructor. Se asegura de cerrar la conexión si el objeto se destruye.
- bool crear ()

  Solicita al Sistema Operativo la creación de un endpoint de comunicación.
- bool conectar (const char *ip, int puerto)

  Intenta establecer el "Túnel" (TCP) con el servidor.
- void enviar (const char *mensaje)

  Envía datos a través del túnel.
- std::string recibir ()

  Espera y captura datos provenientes del servidor.
- void cerrar ()

  Cierra ordenadamente la conexión.

Private Attributes

- int clienteSocket

  Descriptor de archivo del socket.
- sockaddr_in serverAddr

  Estructura de datos para IPv4.

3.2.1 Detailed Description

Clase que representa al cliente en la arquitectura Cliente-Servidor.

- Su función principal es iniciar la comunicación, mantener el descriptor de archivo del socket y convertir los mensajes de texto de C++ en paquetes de bytes para la red.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 ClienteSocket()

```
ClienteSocket::ClienteSocket ()
```

Constructor. Inicializa las variables en un estado seguro/nulo.

Constructor.

- Inicializa el descriptor del socket en -1 para indicar que está "vacío" o "no asignado". Esto evita que intentemos cerrar o usar un socket basura por accidente.

3.2.2.2 ~ClienteSocket()

```
ClienteSocket::~ClienteSocket ()
```

Destructor. Se asegura de cerrar la conexión si el objeto se destruye.

Destructor.

- Implementa el principio RAII (Resource Acquisition Is Initialization) para evitar fugas de recursos.
- Garantiza que, si el objeto ClienteSocket se destruye (ej. al cerrar la ventana), la conexión se cierre correctamente liberando recursos del sistema.

Here is the call graph for this function:

3.2.3 Member Function Documentation

3.2.3.1 cerrar()

```
void ClienteSocket::cerrar ()
```

Cierra ordenadamente la conexión.

Cierra el descriptor de archivo.

- Envía un paquete FIN para terminar el handshake TCP y libera el descriptor.
- Libera el puerto y la memoria en el Kernel.

Here is the caller graph for this function:

3.2.3.2 conectar()

```
bool ClienteSocket::conectar (
            const char * ip,
            int puerto)
```

Intenta establecer el "Túnel" (TCP) con el servidor.

Establece la conexión física/lógica con el servidor.

- Utiliza la syscall 'connect()'. Es una operación bloqueante por defecto.

Parameters

ip	Dirección IP del servidor (ej. "127.0.0.1").
puerto	Puerto de escucha del servidor (ej. 8080).

Returns

true si el servidor aceptó la conexión.

- Convierte la IP de texto a binario.

Here is the caller graph for this function:

3.2.3.3 crear()

```
bool ClienteSocket::crear ()
```

Solicita al Sistema Operativo la creación de un endpoint de comunicación.

Solicita un socket al Kernel.

- Utiliza la syscall 'socket()'.

Returns

true si el SO nos asignó un ID de socket válido, false si falló.

- AF_INET: Indica que usaremos IPv4.
- SOCK_STREAM: Indica que usaremos TCP (protocolo fiable, ordenado, con conexión).
- 0: Protocolo IP automático.

Here is the caller graph for this function:

3.2.3.4 enviar()

```
void ClienteSocket::enviar (
            const char * mensaje)
```

Envía datos a través del túnel.

Envía bytes crudos al servidor.

- Serializa el texto y lo empuja al buffer de salida de la tarjeta de red.

Parameters

mensaje	El texto crudo (string) a enviar.

- Utiliza 'send' en lugar de 'write' porque es específico para sockets.

Parameters

mensaje	Cadena de caracteres estilo C.

Here is the caller graph for this function:

3.2.3.5 recibir()

```
std::string ClienteSocket::recibir ()
```

Espera y captura datos provenientes del servidor.

Escucha y captura la respuesta del servidor.

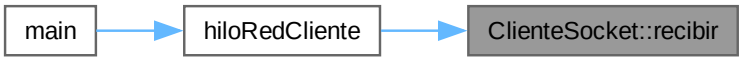
- Lee el buffer de entrada de la tarjeta de red.

  Returns

  std::string El mensaje reconstruido como objeto string de C++.

- Esta función es BLOQUEANTE por defecto (espera hasta que llegue algo).

Here is the caller graph for this function:



3.2.4 Member Data Documentation

3.2.4.1 clienteSocket

```
int ClienteSocket::clienteSocket  [private]
```

Descriptor de archivo del socket.

- En Linux, todo es un archivo. El socket se identifica con un simple número entero (int). Si es -1, significa que el socket no es válido o hubo error.

3.2.4.2 serverAddr

```
sockaddr_in ClienteSocket::serverAddr  [private]
```

Estructura de datos para IPv4.

- Contiene la Familia (AF_INET), el Puerto (htons) y la Dirección IP (inet_addr) del servidor al que nos queremos conectar.

The documentation for this class was generated from the following files:

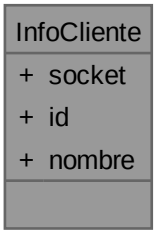
- include/clienteSocket.h
- src/clienteSocket.cpp

3.3 InfoCliente Struct Reference

Datos asociados a un cliente conectado.

```
#include <socket.h>
```

Collaboration diagram for InfoCliente:



Public Attributes

- int socket

  El ID numérico del socket (File Descriptor).
- int id

  ID único autoincremental asignado por nuestro sistema.
- std::string nombre

  Nombre para mostrar en la interfaz (ej. "Cliente 5").

3.3.1 Detailed Description

Datos asociados a un cliente conectado.

- Permite guardar información relevante del cliente para su administración.

3.3.2 Member Data Documentation

3.3.2.1 id

```
int InfoCliente::id
```

ID único autoincremental asignado por nuestro sistema.

3.3.2.2 nombre

```
std::string InfoCliente::nombre
```

Nombre para mostrar en la interfaz (ej. "Cliente 5").

3.3.2.3 socket

```
int InfoCliente::socket
```

El ID numérico del socket (File Descriptor).

The documentation for this struct was generated from the following file:

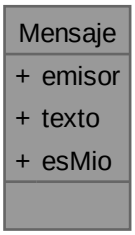
- include/socket.h

3.4 Mensaje Struct Reference

Estructura de datos simple que representa un único mensaje de texto.

`#include <chat.h>`

Collaboration diagram for Mensaje:



Public Attributes

- std::string emisor

  Nombre o ID de quien envió el mensaje (ej. "Cliente 1", "Soporte").
- std::string texto

  El contenido del mensaje.
- bool esMio

  Flag booleana para la UI: true=Derecha (Verde), false=Izquierda (Blanco).

3.4.1 Detailed Description

Estructura de datos simple que representa un único mensaje de texto.

- Se utiliza struct en lugar de class porque solo necesitamos un contenedor de datos público sin lógica compleja interna.

3.4.2 Member Data Documentation

3.4.2.1 emisor

`std::string Mensaje::emisor`

Nombre o ID de quien envió el mensaje (ej. "Cliente 1", "Soporte").

3.4.2.2 esMio

```
bool Mensaje::esMio
```

Flag booleana para la UI: true=Derecha (Verde), false=Izquierda (Blanco).

3.4.2.3 texto

```
std::string Mensaje::texto
```

El contenido del mensaje.

The documentation for this struct was generated from the following file:

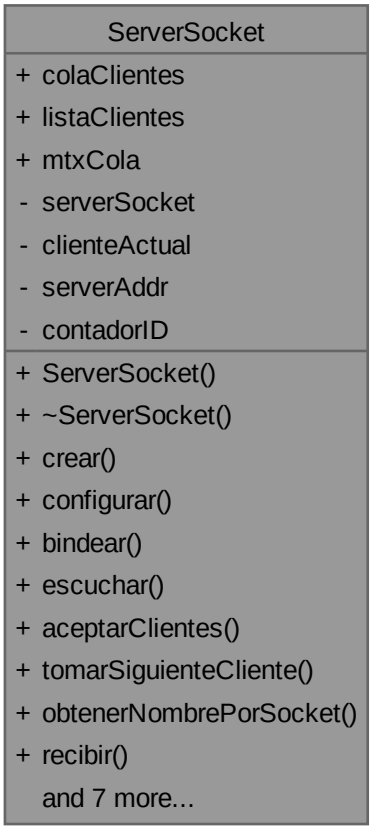
- include/chat.h

3.5 ServerSocket Class Reference

Clase administradora del servidor concurrente.

```
#include <socket.h>
```

Collaboration diagram for ServerSocket:



Public Member Functions

- ServerSocket ()

  Constructor. Inicializa el servidor en estado "apagado" y contadores en 0.
- ~ServerSocket ()

  Destructor. Cierra el socket principal para liberar el puerto del SO.
- bool crear ()

  Crea el socket del servidor usando la syscall socket().
- bool configurar (const char *ip, int puerto)

  Configura la estructura sockaddr_in con la IP y Puerto deseados.
- bool bindear ()

  Vincula el socket a la dirección IP/Puerto en el Sistema Operativo.
- bool escuchar (int espera=5)

  Pone al socket en modo pasivo para escuchar conexiones.
- void aceptarClientes ()

  Bucle infinito (para correr en un hilo aparte) que acepta conexiones.
- bool tomarSiguienteCliente ()

  Extrae al siguiente cliente de la cola y lo marca como activo.
- std::string obtenerNombrePorSocket (int socket)

  Busca en el vector listaClientes el nombre asociado a un socket.
- std::string recibir ()

  Recibe datos del cliente que está siendo atendido ACTUALMENTE.
- void enviar (const std::string &msg)

  Envía datos al cliente que está siendo atendido ACTUALMENTE.
- void cerrarCliente ()

  Cierra la conexión con un cliente específico.
- void cerrarServidor ()

  Apaga todo el servidor.
- int getClienteActual ()

  Getter para obtener el ID del socket activo.
- bool hayClientesEnCola ()

  Verifica si hay personas esperando en la fila.
- bool estoyAtendiendo ()

  Verifica si el agente está ocupado.
- void liberarClienteActual ()

  Resetea el estado del agente a "Libre".

Public Attributes

- std::queue< int > colaClientes

  Cola de espera "First-In, First-Out" (FIFO).
- std::vector< InfoCliente > listaClientes

  Base de datos en memoria de todos los conectados.
- std::mutex mtxCola

  Mutex para proteger la cola de condiciones de carrera.

Private Attributes

- int serverSocket

  Descriptor del socket principal que escucha ("El Oído").
- int clienteActual

  Socket del cliente que está siendo atendido actualmente (-1 si libre).
- sockaddr_in serverAddr

  Configuración de red (IP/Puerto).
- int contadorID

  Contador para generar IDs únicos (1, 2, 3...).

3.5.1 Detailed Description

Clase administradora del servidor concurrente.

- Responsabilidades:

1. Escuchar conexiones entrantes en un puerto específico.
2. Gestionar la concurrencia mediante hilos (Acceptor Thread vs Main Thread).
3. Administrar la cola de espera (FIFO) para atención al cliente.
4. Mantener el registro de todos los clientes conectados.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 ServerSocket()

```
ServerSocket::ServerSocket ()
```

Constructor. Inicializa el servidor en estado "apagado" y contadores en 0.

Constructor. Inicializa los descriptores en -1 (estado inválido).

3.5.2.2 ~ServerSocket()

```
ServerSocket::~ServerSocket ()
```

Destructor. Cierra el socket principal para liberar el puerto del SO.

Destructor. Llama a cerrarServidor() para asegurar limpieza de recursos. Here is the call graph for this function:

3.5.3 Member Function Documentation

3.5.3.1 aceptarClientes()

```
void ServerSocket::aceptarClientes ()
```

Bucle infinito (para correr en un hilo aparte) que acepta conexiones.

Bucle infinito que corre en un hilo secundario (Background).

- Cuando llega alguien, lo registra, lo mete a la cola y le envía señal de espera.
- Thread-Safe: Usa mtxCola al modificar la cola.
- Su única tarea es esperar a que alguien se conecte y meterlo a la cola.
- CRÍTICO: Usa Mutex para proteger el acceso a 'colaClientes' y 'listaClientes'.

Here is the caller graph for this function:

3.5.3.2 bindear()

```
bool ServerSocket::bindear ()
```

Vincula el socket a la dirección IP/Puerto en el Sistema Operativo.

"Amarra" el socket a un puerto específico de la máquina.

- Utiliza la syscall bind(). Si falla, usualmente es porque el puerto está ocupado.
- Nota técnica: Se usa '::bind' (con dos puntos) para decirle al compilador que use la función bind() original de C (global) y no std::bind de C++.

Here is the caller graph for this function:

3.5.3.3 cerrarCliente()

```
void ServerSocket::cerrarCliente ()
```

Cierra la conexión con un cliente específico.

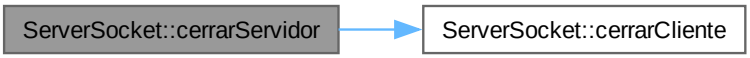
Here is the caller graph for this function:

3.5.3.4 cerrarServidor()

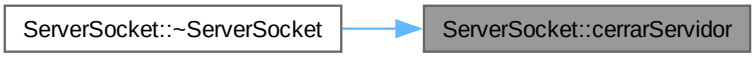
```
void ServerSocket::cerrarServidor ()
```

Apaga todo el servidor.

Here is the call graph for this function:



Here is the caller graph for this function:



3.5.3.5 configurar()

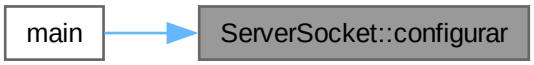
```
bool ServerSocket::configurar (
            const char * ip,
            int puerto)
```

Configura la estructura sockaddr_in con la IP y Puerto deseados.

Prepara la estructura de dirección.

- Utiliza htons y inet_pton para asegurar que los datos sean legibles por la red (Big Endian).

Here is the caller graph for this function:



3.5.3.6 crear()

```
bool ServerSocket::crear ()
```

Crea el socket del servidor usando la syscall socket().

Solicita un socket maestro al Kernel.

Returns

true si se creó el descriptor correctamente.

- AF_INET: IPv4.
- SOCK_STREAM: TCP (Orientado a conexión).

Here is the caller graph for this function:

3.5.3.7 enviar()

```
void ServerSocket::enviar (
            const std::string & msg)
```

Envía datos al cliente que está siendo atendido ACTUALMENTE.

Here is the caller graph for this function:

3.5.3.8 escuchar()

```
bool ServerSocket::escuchar (
            int espera = 5)
```

Pone al socket en modo pasivo para escuchar conexiones.

Pone el socket en modo pasivo (escucha).

Parameters

espera	Tamaño del backlog (cuántas conexiones pendientes admite el kernel).
espera	Longitud de la cola de conexiones pendientes del Kernel (Backlog).

Here is the caller graph for this function:

3.5.3.9 estoyAtendiendo()

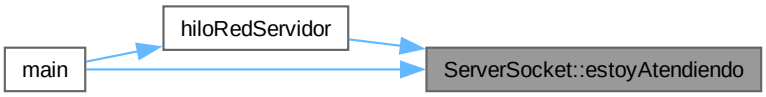
```
bool ServerSocket::estoyAtendiendo ()
```

Verifica si el agente está ocupado.

Returns

true si clienteActual != -1.

Here is the caller graph for this function:

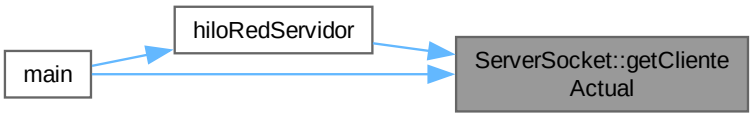


3.5.3.10 getClienteActual()

```
int ServerSocket::getClienteActual ()
```

Getter para obtener el ID del socket activo.

Here is the caller graph for this function:



3.5.3.11 hayClientesEnCola()

```
bool ServerSocket::hayClientesEnCola ()
```

Verifica si hay personas esperando en la fila.

Returns

true si la cola no está vacía. Thread-Safe.

Here is the caller graph for this function:

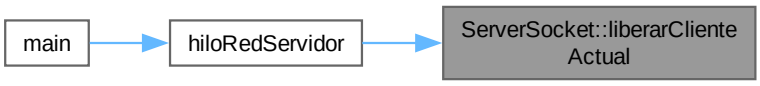
3.5.3.12 liberarClienteActual()

```
void ServerSocket::liberarClienteActual ()
```

Resetea el estado del agente a "Libre".

- Se llama cuando el cliente actual se desconecta o termina la sesión.

Here is the caller graph for this function:



3.5.3.13 obtenerNombrePorSocket()

```
std::string ServerSocket::obtenerNombrePorSocket (
            int socketBuscado)
```

Busca en el vector listaClientes el nombre asociado a un socket.

Búsqueda lineal en el vector de clientes.

Parameters

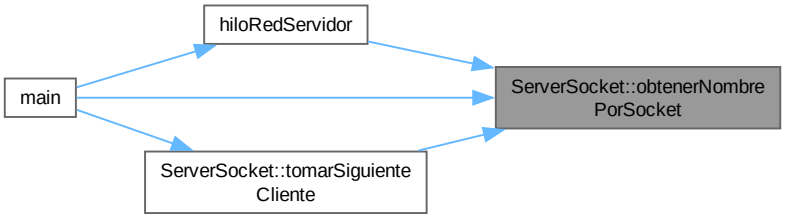
socket	El ID del socket a buscar.

Returns

El nombre del cliente o "Desconocido".

- Itera sobre la listaClientes para encontrar el nombre asociado a un socket.

Here is the caller graph for this function:



3.5.3.14 recibir()

```
string ServerSocket::recibir ()
```

Recibe datos del cliente que está siendo atendido ACTUALMENTE.

Lee datos del cliente actual.

- Usa memset para limpiar el buffer antes de leer, evitando basura de memoria.

Here is the caller graph for this function:

3.5.3.15 tomarSiguienteCliente()

```
bool ServerSocket::tomarSiguienteCliente ()
```

Extrae al siguiente cliente de la cola y lo marca como activo.

Saca al siguiente cliente de la cola y empieza la sesión.

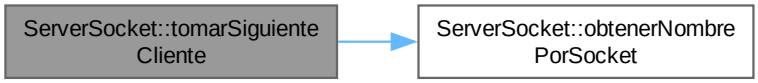
- Thread-Safe: Usa mtxCola.

  Returns

  true si había alguien en la cola y se pudo tomar, false si estaba vacía.

- Esta función es llamada por el 'Main Loop'.
- CRÍTICO: Usa Mutex porque modifica la misma cola que usa aceptarClientes().

Here is the call graph for this function:



Here is the caller graph for this function:

3.5.4 Member Data Documentation

3.5.4.1 clienteActual

```
int ServerSocket::clienteActual  [private]
```

Socket del cliente que está siendo atendido actualmente (-1 si libre).

3.5.4.2 colaClientes

```
std::queue<int> ServerSocket::colaClientes
```

Cola de espera "First-In, First-Out" (FIFO).

- Almacena los sockets de los clientes que están esperando turno. Es pública para depuración, pero debería accederse con cuidado.

3.5.4.3 contadorID

```
int ServerSocket::contadorID  [private]
```

Contador para generar IDs únicos (1, 2, 3...).

3.5.4.4 listaClientes

```
std::vector<InfoCliente> ServerSocket::listaClientes
```

Base de datos en memoria de todos los conectados.

- Se usa para buscar el nombre de un cliente a partir de su socket.

3.5.4.5 mtxCola

```
std::mutex ServerSocket::mtxCola
```

Mutex para proteger la cola de condiciones de carrera.

- Vital porque el Hilo Aceptador (background) mete gente a la cola y el Hilo Principal (Main) saca gente de la cola. Sin esto, el programa crashea.

3.5.4.6 serverAddr

```
sockaddr_in ServerSocket::serverAddr  [private]
```

Configuración de red (IP/Puerto).

3.5.4.7 serverSocket

```
int ServerSocket::serverSocket  [private]
```

Descriptor del socket principal que escucha ("El Oído").

The documentation for this class was generated from the following files:

- include/socket.h
- src/socket.cpp

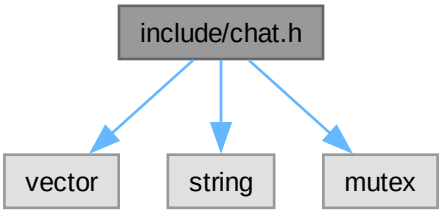
Chapter 4

File Documentation

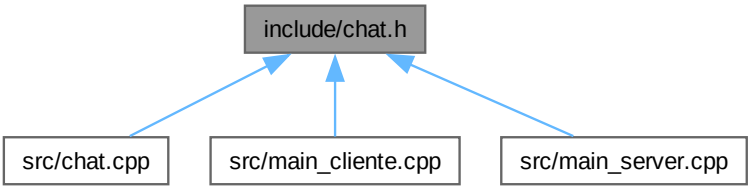
4.1 include/chat.h File Reference

```
#include <vector>
#include <string>
#include <mutex>
```

Include dependency graph for chat.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct Mensaje

  Estructura de datos simple que representa un único mensaje de texto.
- class Chat

  Clase gestora del historial de la conversación.

4.2 chat.h

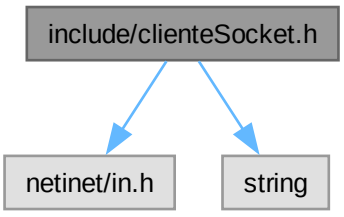
Go to the documentation of this file.

```
00001
00012
00013 #ifndef CHAT_H
00014 #define CHAT_H
00015
00016 #include <vector>
00017 #include <string>
00018 #include <mutex>
00019
00026 struct Mensaje{
00027
00028     std::string emisor;
00029     std::string texto;
00030     bool esMio;
00031
00032 };
00033
00042 class Chat{
00043
00044     private:
00051         std::vector<Mensaje> historial;
00052
00059         std::mutex mtx;
00060
00061     public:
00065         Chat(){};
00066
00073         void agregarMensaje(std::string emisor, std::string texto, bool esMio);
00074
00081         std::vector<Mensaje> obtenerHistorial();
00082
00088         void limpiarHistorial();
00089
00090 };
00091
00092 #endif
```

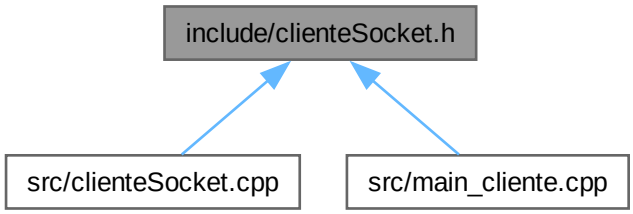
4.3 include/clienteSocket.h File Reference

```
#include <netinet/in.h>
#include <string>
```

Include dependency graph for clienteSocket.h:



This graph shows which files directly or indirectly include this file:



Classes

- class ClienteSocket

  Clase que representa al cliente en la arquitectura Cliente-Servidor.

4.4 clienteSocket.h

Go to the documentation of this file.

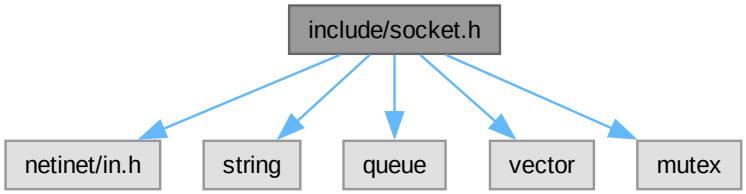
```
00001
00012
00013 #ifndef CLIENTESOCKET_H
00014 #define CLIENTESOCKET_H
00015
00016 #include <netinet/in.h> // Estructuras necesarias para direcciones de internet (sockaddr_in)
00017 #include <string>       // Para manejar cadenas de texto dinámicas (std::string)
00018
00026 class ClienteSocket{
00027     private:
00033         int clienteSocket;
00034
00040         sockaddr_in serverAddr;
00041
00042     public:
00043
00047         ClienteSocket();
00048
00054         ~ClienteSocket();
00055
00061         bool crear();
00062
00070         bool conectar(const char* ip, int puerto);
00071
00077         void enviar(const char* mensaje);
00078
00084         std::string recibir();
00085
00090         void cerrar();
00091
00092 };
00093
00094 #endif
```

4.5 include/socket.h File Reference

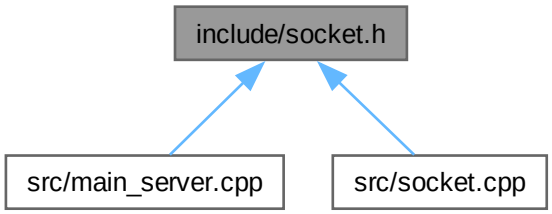
```
#include <netinet/in.h>
#include <string>
```

```
#include <queue>
#include <vector>
#include <mutex>
```

Include dependency graph for socket.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct InfoCliente

  Datos asociados a un cliente conectado.
- class ServerSocket

  Clase administradora del servidor concurrente.

4.6 socket.h

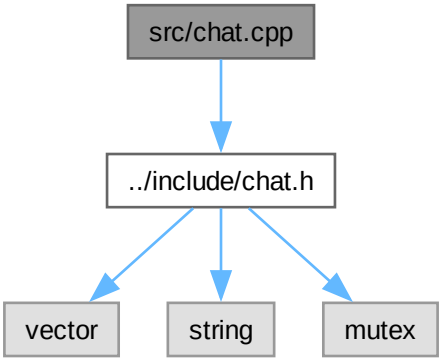
Go to the documentation of this file.

```
00001
00013
00014 #ifndef SERVERSOCKET_H
00015 #define SERVERSOCKET_H
00016
00017 #include <netinet/in.h>
00018 #include <string>
00019 #include <queue>
00020 #include <vector> // Necesario para std::vector
00021 #include <mutex>
```

```
00022
00028 struct InfoCliente {
00029     int socket;
00030     int id;
00031     std::string nombre;
00032 };
00033
00043 class ServerSocket {
00044 private:
00045     int serverSocket;
00046     int clienteActual;
00047     sockaddr_in serverAddr;
00048     int contadorID;
00049
00050 public:
00056     std::queue<int> colaClientes;
00057
00062     std::vector<InfoCliente> listaClientes;
00063
00069     std::mutex mtxCola;
00070
00074     ServerSocket();
00075
00079     ~ServerSocket();
00080
00085     bool crear();
00086
00090     bool configurar(const char* ip, int puerto);
00091
00096     bool bindear();
00097
00102     bool escuchar(int espera = 5);
00103
00109     void aceptarClientes();
00110
00116     bool tomarSiguienteCliente();
00117
00123     std::string obtenerNombrePorSocket(int socket);
00124
00128     std::string recibir();
00129
00133     void enviar(const std::string& msg);
00134
00138     void cerrarCliente();
00139
00143     void cerrarServidor();
00144
00148     int getClienteActual();
00149
00154     bool hayClientesEnCola();
00155
00160     bool estoyAtendiendo();
00161
00166     void liberarClienteActual();
00167 };
00168
00169 #endif
```

4.7 src/chat.cpp File Reference

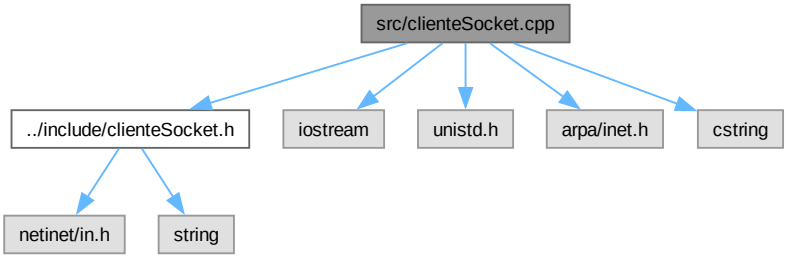
```
#include "../include/chat.h"
```
Include dependency graph for chat.cpp:



4.8 src/clienteSocket.cpp File Reference

Implementación de la comunicación de red lado Cliente.

```
#include "../include/clienteSocket.h"
#include <iostream>
#include <unistd.h>
#include <arpa/inet.h>
#include <cstring>
```
Include dependency graph for clienteSocket.cpp:



4.8.1 Detailed Description

Implementación de la comunicación de red lado Cliente.

Author

Valencia Cedeño Marcos Gael

Peralta Ordóñez Jesús

Esteves Flores Andrés

Version

1.0

Date

06/01/2026

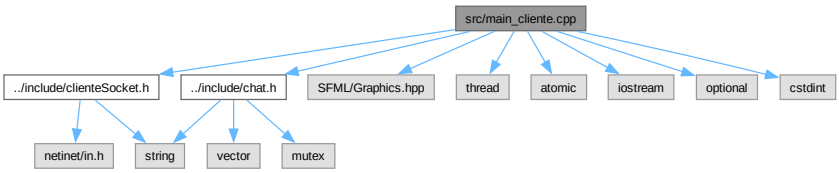
- Este archivo traduce las solicitudes de alto nivel de nuestra aplicación a llamadas de bajo nivel (syscalls) del sistema operativo Linux/Unix.

4.9 src/main_cliente.cpp File Reference

Punto de entrada de la aplicación Cliente (Usuario).

```
#include "../include/clienteSocket.h"
#include "../include/chat.h"
#include <SFML/Graphics.hpp>
#include <thread>
#include <atomic>
#include <iostream>
#include <optional>
#include <cstdint>
```

Include dependency graph for main_cliente.cpp:



Functions

- std::atomic< bool > enEspera (true)

  Bandera de estado compartida entre hilos (Thread-Safe).
- void hiloRedCliente (ClienteSocket *cliente, Chat *manager)

  Función del Hilo Secundario: Escucha al servidor.
- int main ()

  Hilo Principal (UI Thread).

4.9.1 Detailed Description

Punto de entrada de la aplicación Cliente (Usuario).

Author

Valencia Cedeño Marcos Gael

Peralta Ordóñez Jesús

Esteves Flores Andrés

Version

1.0

Date

06/01/2026

- Este archivo maneja la Interfaz Gráfica (SFML) para el usuario final.
- Implementa el protocolo de comunicación (/WAIT, /START) para bloquear o desbloquear la interacción según la disponibilidad del agente.

4.9.2 Function Documentation

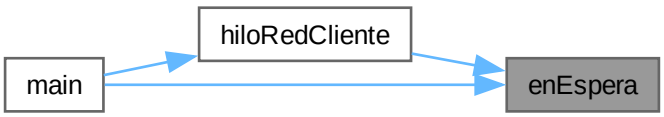
4.9.2.1 enEspera()

```
std::atomic< bool > enEspera (
            true )
```

Bandera de estado compartida entre hilos (Thread-Safe).

- Se usa std::atomic<bool> en lugar de un bool normal para evitar "Condiciones de Carrera".
- True: El cliente está en la cola de espera (Pantalla Bloqueada).
- False: El cliente está siendo atendido (Pantalla Libre).

Here is the caller graph for this function:



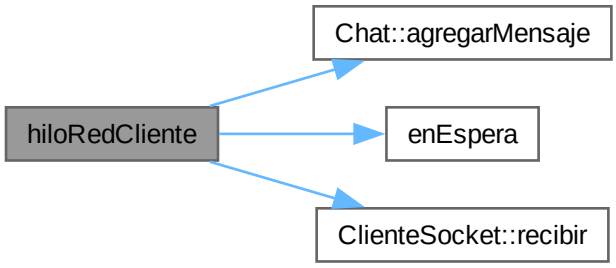
4.9.2.2 hiloRedCliente()

```
void hiloRedCliente (
            ClienteSocket * cliente,
            Chat * manager)
```

Función del Hilo Secundario: Escucha al servidor.

- Se encarga de recibir mensajes y decidir si son TEXTO para el chat o COMANDOS para cambiar el estado de la aplicación.

Here is the call graph for this function:



Here is the caller graph for this function:

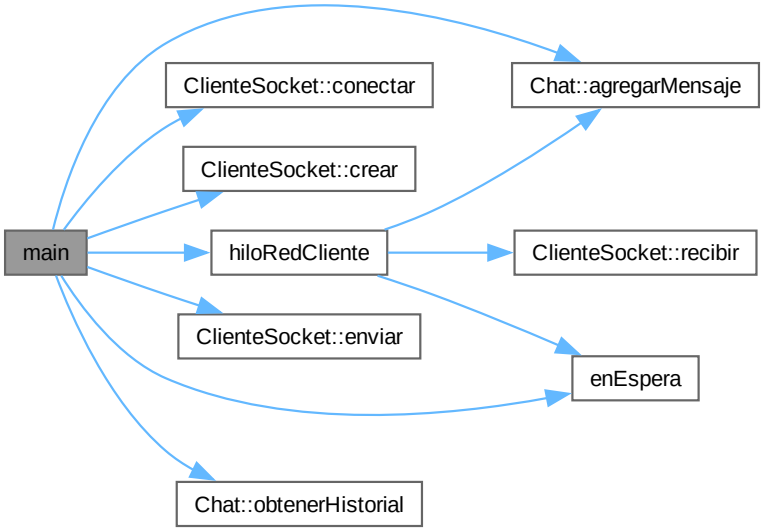
4.9.2.3 main()

```
int main ()
```

Hilo Principal (UI Thread).

- Dibuja la ventana, maneja el input del usuario y renderiza el overlay de bloqueo.

Here is the call graph for this function:

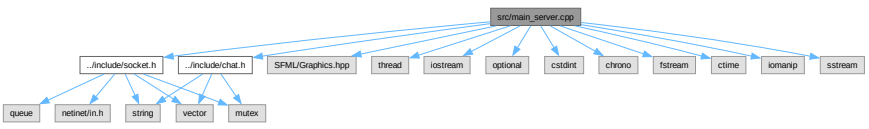


4.10 src/main_server.cpp File Reference

Punto de entrada de la aplicación Servidor (Agente de Soporte).

```
#include "../include/socket.h"
#include "../include/chat.h"
#include <SFML/Graphics.hpp>
#include <thread>
#include <iostream>
#include <optional>
#include <cstdint>
#include <chrono>
#include <fstream>
#include <ctime>
#include <iomanip>
#include <sstream>
```

Include dependency graph for main_server.cpp:



Functions

- std::string obtenerTimestamp ()

Genera una marca de tiempo actual (Timestamp).

- void generarTicket (int id, std::string nombre, const std::vector< Mensaje > &historial)

  Crea un archivo de tipo ticket (.txt) al finalizar la sesión.
- void hiloRedServidor (ServerSocket *servidor, Chat *manager)

  Función ejecutada por el Hilo Lector (Reader Thread).
- int main ()

  Hilo Principal (UI Thread).

4.10.1 Detailed Description

Punto de entrada de la aplicación Servidor (Agente de Soporte).

Author

Valencia Cedeño Marcos Gael

Peralta Ordóñez Jesús

Esteves Flores Andrés

Version

1.0

Date

06/01/2026

- Este archivo orquesta los tres componentes principales:

1. La Interfaz Gráfica (GUI) con SFML.
2. La lógica de red en segundo plano (Hilos).
3. La gestión de archivos para generar los Tickets de reporte.

4.10.2 Function Documentation

4.10.2.1 generarTicket()

```
void generarTicket (
            int id,
            std::string nombre,
            const std::vector< Mensaje > & historial)
```

Crea un archivo de tipo ticket (.txt) al finalizar la sesión.

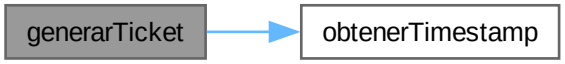
- Esta función garantiza la PERSISTENCIA de los datos. Si el programa se cierra, la conversación queda guardada en disco.

Parameters

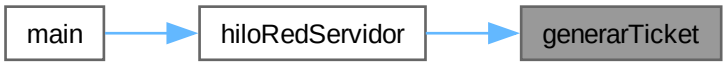
id	ID numérico del cliente.
nombre	Nombre legible (ej. "Cliente 5").

historial	Vector con todos los mensajes de la sesión.

Here is the call graph for this function:



Here is the caller graph for this function:



4.10.2.2 hiloRedServidor()

```
void hiloRedServidor (
            ServerSocket * servidor,
            Chat * manager)
```

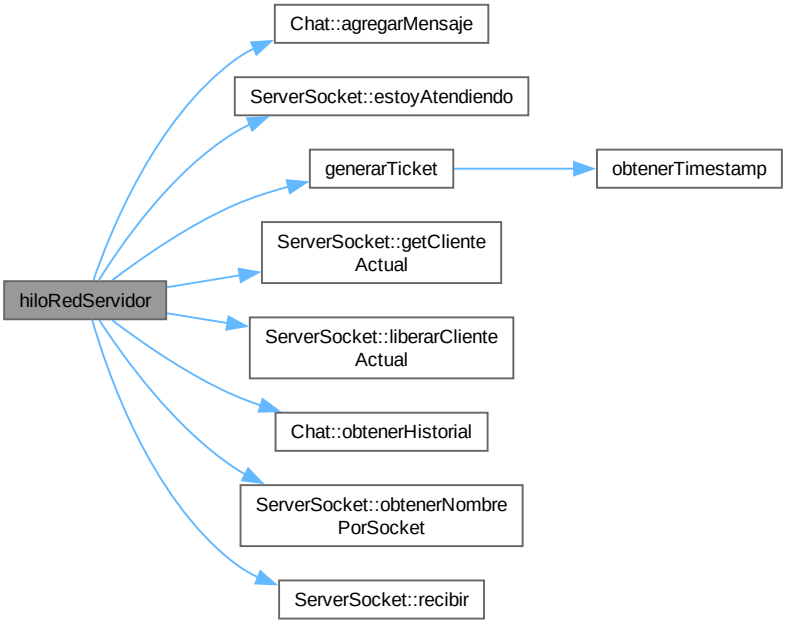
Función ejecutada por el Hilo Lector (Reader Thread).

- Se mantiene en un bucle infinito escuchando al cliente ACTIVO.
- Detecta desconexiones y dispara la generación automática del ticket.

Parameters

servidor	Puntero a la instancia del socket (para recibir datos).
manager	Puntero al gestor del chat (para guardar mensajes).

Here is the call graph for this function:



Here is the caller graph for this function:

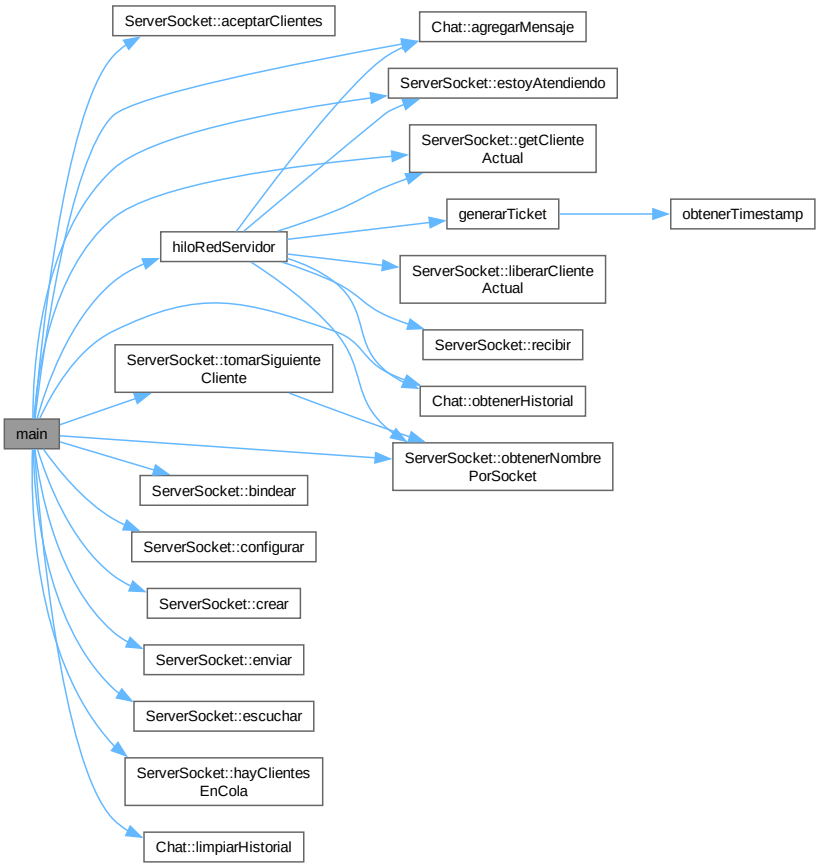
4.10.2.3 main()

```
int main ()
```

Hilo Principal (UI Thread).

- Maneja la ventana de SFML, los eventos de entrada (teclado/mouse) y la lógica de asignación de turnos ("El Portero").

Here is the call graph for this function:



4.10.2.4 obtenerTimestamp()

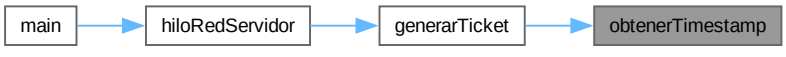
```
std::string obtenerTimestamp ()
```

Genera una marca de tiempo actual (Timestamp).

Returns

std::string Fecha y hora en formato "YYYY-MM-DD HH:MM:SS".

Here is the caller graph for this function:

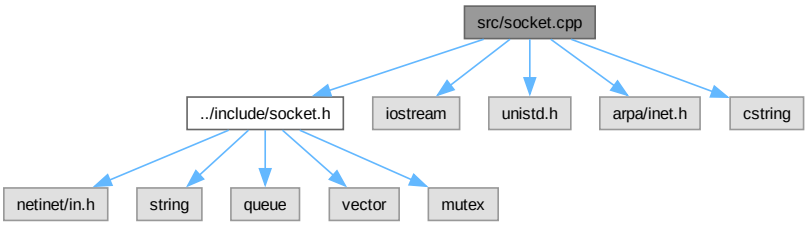


4.11 src/socket.cpp File Reference

Implementación de la lógica del servidor concurrente.

```
#include "../include/socket.h"
#include <iostream>
#include <unistd.h>
#include <arpa/inet.h>
#include <cstring>
```

Include dependency graph for socket.cpp:



4.11.1 Detailed Description

Implementación de la lógica del servidor concurrente.

Author

Valencia Cedeño Marcos Gael

Peralta Ordóñez Jesús

Esteves Flores Andrés

Version

1.0

Date

06/01/2026

- Este archivo contiene la implementación de las syscalls de red y la lógica crítica de sincronización de hilos para la cola de clientes.

Index