

Socket

Generado por Doxygen 1.8.3.1

Sábado, 7 de Diciembre de 2013 01:45:58

Índice general

1	Página principal	1
2	Client	3
3	HOWTO	5
3.1	Descripción del protocolo	5
3.1.1	Modo en texto llano	5
3.1.2	Envío/Recepción de mensajes	5
3.2	Compilación	6
3.2.1	Opciones de compilación	6
3.2.2	Flags de compilación y librerías	6
3.3	Ejemplos	6
3.3.1	Programar un cliente	6
4	Índice de clases	7
4.1	Lista de clases	7
5	Índice de archivos	9
5.1	Lista de archivos	9
6	Documentación de las clases	11
6.1	Referencia de la Clase Server	11
6.1.1	Descripción detallada	12
6.1.2	Documentación del constructor y destructor	12
6.1.2.1	Server	12
6.1.3	Documentación de las funciones miembro	12
6.1.3.1	freeRAM	12
6.1.3.2	getNWorkers	12
6.1.3.3	getStartedThreads	12
6.1.3.4	getStoppedThreads	12
6.1.3.5	requestExit	12
6.1.3.6	setNWorkers	13
6.1.3.7	startServer	13

6.1.4	Documentación de los datos miembro	13
6.1.4.1	nWorkers	13
6.1.4.2	shutdownServer	13
6.1.4.3	ss	13
6.1.4.4	startedThreads	13
6.1.4.5	stoppedThreads	14
6.1.4.6	workerID	14
6.2	Referencia de la Clase Socket	14
6.2.1	Descripción detallada	15
6.2.2	Documentación del constructor y destructor	15
6.2.2.1	Socket	15
6.2.3	Documentación de las funciones miembro	15
6.2.3.1	Accept	15
6.2.3.2	Bind	15
6.2.3.3	Close	15
6.2.3.4	Connect	16
6.2.3.5	Create	16
6.2.3.6	getSock	16
6.2.3.7	Listen	16
6.2.3.8	operator<<	16
6.2.3.9	operator>>	16
6.2.3.10	Receive	17
6.2.3.11	Send	17
6.2.4	Documentación de los datos miembro	17
6.2.4.1	sock	17
6.2.4.2	sockAddr	17
6.3	Referencia de la Clase SocketException	17
6.3.1	Descripción detallada	18
6.3.2	Documentación del constructor y destructor	18
6.3.2.1	SocketException	18
6.3.2.2	~SocketException	18
6.3.3	Documentación de las funciones miembro	18
6.3.3.1	description	18
6.3.4	Documentación de los datos miembro	18
6.3.4.1	message	18
6.4	Referencia de la Estructura thread_args	18
6.4.1	Descripción detallada	19
6.4.2	Documentación de los datos miembro	19
6.4.2.1	condition	19
6.4.2.2	id	19

6.4.2.3	mutex	19
6.4.2.4	s	20
6.4.2.5	serv	20
6.4.2.6	thread	20
7	Documentación de archivos	21
7.1	Referencia del Archivo /home/imanol/devel/eclipse/PAD/JChatClient/src/client.cpp	21
7.1.1	Descripción detallada	21
7.1.2	Documentación de las funciones	22
7.1.2.1	exitClient	22
7.1.2.2	main	22
7.1.2.3	processCMD	22
7.1.3	Documentación de las variables	22
7.1.3.1	connected	22
7.2	Referencia del Archivo /home/imanol/devel/eclipse/PAD/JChatClient/src/client.dox	22
7.2.1	Descripción detallada	22
7.3	Referencia del Archivo /home/imanol/devel/eclipse/PAD/JChatClient/src/howto.dox	22
7.3.1	Descripción detallada	22
7.4	Referencia del Archivo /home/imanol/devel/eclipse/PAD/JChatClient/src/include/server.h	22
7.4.1	Descripción detallada	23
7.4.2	Documentación de los 'defines'	24
7.4.2.1	CONFFILE	24
7.4.2.2	N	24
7.4.3	Documentación de las funciones	24
7.4.3.1	killThread	24
7.4.3.2	processText	24
7.4.3.3	readConf	24
7.4.3.4	WorkerThread	24
7.5	Referencia del Archivo /home/imanol/devel/eclipse/PAD/JChatClient/src/include/Socket.h	25
7.5.1	Descripción detallada	25
7.6	Referencia del Archivo /home/imanol/devel/eclipse/PAD/JChatClient/src/include/SocketException.h	25
7.6.1	Descripción detallada	26
7.7	Referencia del Archivo /home/imanol/devel/eclipse/PAD/JChatClient/src/mainpage.dox	26
7.8	Referencia del Archivo /home/imanol/devel/eclipse/PAD/JChatClient/src/makefile.dox	26
7.9	Referencia del Archivo /home/imanol/devel/eclipse/PAD/JChatClient/src/Socket.cpp	26
7.9.1	Descripción detallada	26
Índice		26

Capítulo 1

Página principal

La librería [Socket](#) es una implementación de comunicación con Sockets TCP/IP en [GNU/Linux](#).

El objetivo principal de esta librería es abstraer toda la capa de comunicación al programador, lo único que debe hacer es guardar los datos a enviar en un contenedor STL del tipo `std::string` y enviarlo a una instancia debidamente inicializada de la clase [Socket](#). Este se encarga de inicializar los sockets, atender las conexiones o realizar nuevas, etc.

Para más información, consultar el [HOWTO](#)

Autor

Imanol Barba Sabariego

Fecha

11/06/2013

Capítulo 2

Client

Ejemplo de aplicación cliente

```
#include "Socket.h"
#include <iostream>
#include "SocketException.h"
#include <sstream>
#include <signal.h>
#include <cstdlib>
#include <sys/time.h>

using namespace std;

void exitClient(int signal)
{
    cout << "Server connection terminated unexpectedly" << endl << "Exiting" << endl;
    exit(-1);
}

int main()
{
    signal(SIGPIPE, exitClient);
    signal(SIGINT, exitClient);
    Socket s;
    string send, recv, host;
    int port;
    s.Create();
    cout << "Created socket" << endl;
    cout << "Hostname: ";
    cin >> host;
    cout << "Port: ";
    cin >> port;
    cin.ignore();
    s.Connect(host, port);
    cout << "Connected" << endl;
    while(true)
    {
        cout << "> ";
        getline(cin, send);
        s >> recv;
        cout << "Received: " << recv << endl;
    }
}
```


Capítulo 3

HOWTO

Descripción de funcionamiento y uso de la librería.

```
A continuación se desglosan las instrucciones para el desarrollo con esta librería y su posterior uso.  
<br><br>
```

3.1. Descripción del protocolo

En las comunicaciones con Sockets hay un problema, es fácil saber la longitud del mensaje que vas a enviar, pero cuesta saber la longitud del mensaje que deseas recibir, ya que la transmisión se puede detener por llegar al fin del mensaje o bien por que hay problemas en la red.

Los sockets de UNIX nos dan una solución, que es usar llamadas no bloqueantes para recibir el mensaje, es decir: Si leo de un socket, y no he llenado el buffer donde guardo los datos que recibo de allí, la aplicación no se bloquea esperando recibir la suficiente cantidad de datos. A su vez, se usan llamadas a la función `select()` para comprobar si hay datos disponibles a para leer o la conexión está terminada.

Esto requiere uso de la forks e ir haciendo encuesta (polling) al socket, cosa que consume más recursos del sistema.

Debido a la complejidad añadida que supone hacerlo por esta vía, ya que la librería usa threads que son más ligeros para el sistema y la memoria RAM; se ha optado por implementar un sencillísimo protocolo de comunicación que no supone gasto de computación y repercute de forma nímia en el rendimiento de la transferencia.

3.1.1. Modo en texto llano

Este modo transmite los caracteres de texto sin ningún tipo de encriptación, por lo tanto son totalmente visibles para cualquier agente intermedio. Sin embargo, el modo sin encriptación es más rápido y ofrece el doble de velocidad de transferencia aproximadamente.

3.1.2. Envío/Recepción de mensajes

El mensaje se transmite de la siguiente forma:

1. Bob le quiere mandar a Alice un mensaje, primero primero pone el número de caracteres (bytes) que ocupa el mensaje en forma de string terminado con carácter NULL ('\0') y lo manda.
2. Bob transmite el mensaje íntegro.
3. Alice lee uno a uno los caracteres del socket hasta encontrar un carácter NULL ('\0'), entonces lee lo que ha recibido, que es la longitud del mensaje real.
4. Alice pasa a extraer los N bytes que ha leído que recibiría y obtiene el mensaje final.

3.2. Compilación

A continuación se detallan las opciones e instrucciones necesarias para compilar esta librería.

3.2.1. Opciones de compilación

Los siguientes #defines establecen la configuración en tiempo de compilación

- `RSALength` : La longitud de la llave RSA que se usará.
- `AESLength` : La longitud de la llave AES que se usará.
- `CONFFILE` : La ruta absoluta o relativa al fichero de configuración del servidor (véase: `conf`).
- `PUBLICKEY` : La ruta absoluta o relativa a la llave pública RSA. Existe en [server.h](#) y [client.cpp](#)
- `PRIVATEKEY` : La ruta absoluta o relativa a la llave pública RSA. Existe en [server.h](#) y [client.cpp](#)

3.2.2. Flags de compilación y librerías

Flags de compilación necesarios:

- `-I/carpeta/con/cabeceras_de/crypto++ -I/carpeta/con/cabeceras/de_la/librería/Socket`
- `-L/carpeta/con/la_librería/crypto++`
- `-lcryptopp -lpthread`

Véase: `makefile`

3.3. Ejemplos

A continuación se detalla el código para programar una aplicación servidor cliente sencilla

3.3.1. Programar un cliente

Véase: [Client](#)

Capítulo 4

Índice de clases

4.1. Lista de clases

Lista de las clases, estructuras, uniones e interfaces con una breve descripción:

Server	Clase de aplicación servidor	11
Socket	Clase del socket	14
SocketException	Clase de Excepción de Sockets	17
thread_args	Argumentos de los threads	18

Capítulo 5

Indice de archivos

5.1. Lista de archivos

Lista de todos los archivos con descripciones breves:

/home/imanol/devel/eclipse/PAD/JChatClient/src/client.cpp	
Fichero de implementación de un cliente	21
/home/imanol/devel/eclipse/PAD/JChatClient/src/Socket.cpp	
Fichero de implementación de la clase Socket	26
/home/imanol/devel/eclipse/PAD/JChatClient/src/include/server.h	
Header de la clase Server	22
/home/imanol/devel/eclipse/PAD/JChatClient/src/include/Socket.h	
Header de la clase Socket	25
/home/imanol/devel/eclipse/PAD/JChatClient/src/include/SocketException.h	
Header de excepción de Sockets	25

Capítulo 6

Documentación de las clases

6.1. Referencia de la Clase Server

Clase de aplicación servidor.

```
#include <server.h>
```

Métodos públicos

- [Server \(\)](#)
Constructor de la clase [Server](#).
- int [getNWorkers \(\)](#)
Getter del número de threads activos.
- void [setNWorkers \(int n\)](#)
Setter del número de threads activos.
- list< pthread_t * > * [getStartedThreads \(\)](#)
Getter de la pila de threads activos.
- list< pthread_t * > * [getStoppedThreads \(\)](#)
Getter de la pila de threads terminados.
- void [startServer \(string i, int p\)](#)
Método de inicialización del servidor.
- void [freeRAM \(list< pthread_t * > *threadList\)](#)
Método para liberar memoria de threads.
- void [requestExit \(\)](#)
Método para terminar el servidor.

Atributos privados

- int [nWorkers](#)
Contador de threads.
- bool [shutdownServer](#)
Variable de apagado.
- int [workerID](#)
Contador de ID de thread.
- list< pthread_t * > [stoppedThreads](#)
Pila de threads terminados.
- list< pthread_t * > [startedThreads](#)

Pila de threads empezados.

- [Socket ss](#)

Socket de comunicación.

6.1.1. Descripción detallada

Clase de aplicación servidor.

Esta clase define un objeto con los métodos y atributos necesarios para lanzar una aplicación servidor y atender las conexiones. Para realizar la comunicación con el cliente, usa un objeto de la clase [Socket](#)

6.1.2. Documentación del constructor y destructor

6.1.2.1. `Server::Server() [inline]`

Constructor de la clase [Server](#).

Inicializa los argumentos iniciales del servidor

6.1.3. Documentación de las funciones miembro

6.1.3.1. `void Server::freeRAM (list< pthread_t * > * threadList)`

Método para liberar memoria de threads.

Este método se llama para que libere la memoria de todos los threads que se le proporcionan por argumento

Parámetros

<i>threadList</i>	Contenedor con los threads a liberar
-------------------	--------------------------------------

6.1.3.2. `int Server::getNWorkers ()`

Getter del número de threads activos.

Devuelve el número de threads activos en ese instante, por tanto, del número de conexiones que están siendo atendidas.

6.1.3.3. `list<pthread_t*>* Server::getStartedThreads ()`

Getter de la pila de threads activos.

Devuelve un contenedor con la lista de threads que estan activos, para terminarlos en caso de que el programa finalice prematuramente

6.1.3.4. `list<pthread_t*>* Server::getStoppedThreads ()`

Getter de la pila de threads terminados.

Devuelve un contenedor con la lista de threads que han terminado, para poder liberar la memoria que se le ha asignado

6.1.3.5. `void Server::requestExit ()`

Método para terminar el servidor.

Este método inicia la secuencia de finalización del servidor

6.1.3.6. void Server::setNWorkers (int *n*)

Setter del número de threads activos.

Establece el número de threads activos, para poder cambiarlo cuando alguno de los threads activos finaliza

Parámetros

<i>n</i>	Nuevo número de threads activos
----------	---------------------------------

6.1.3.7. void Server::startServer (string *i*, int *p*)

Método de inicialización del servidor.

Inicializa el servidor en el puerto e IP especificados para empezar a recibir conexiones entrantes

Parámetros

<i>i</i>	IP donde se aceptan las conexiones
<i>p</i>	Puerto donde se aceptan las conexiones

6.1.4. Documentación de los datos miembro

6.1.4.1. int Server::nWorkers [private]

Contador de threads.

Esta variable se encarga de mantener la cuenta de threads activos, por tanto, el número de conexiones que estan siendo atendidas simultáneamente.

6.1.4.2. bool Server::shutdownServer [private]

Variable de apagado.

Esta variable controla el apagado del servidor, al ponerla a true, la siguiente iteración del bucle que atiende las conexiones no se producirá y el programa terminará.

6.1.4.3. Socket Server::ss [private]

Socket de comunicación.

Esta variable contiene el objeto de la clase [Socket](#) que la aplicación servidor usa para poder atender las peticiones. Su función es quedarse escuchando en el puerto e IP introducidas en el fichero de configuración y crear un objeto de la clase [Socket](#) para cada petición de cada cliente nuevo, siendo este último objeto creado el que se usa para la comunicación.

6.1.4.4. list<pthread_t*> Server::startedThreads [private]

Pila de threads empezados.

Esta variable contiene una lista de threads que han empezado su ejecución. Si el programa finalizara prematuramente, se liberarían los punteros de los threads almacenados en esta pila.

NOTA: No se liberará la memoria asignada a los argumentos de los threads, dando lugar a memory leaks; sin embargo, esto se produciría al finalizar el programa, por tanto no es relevante.

6.1.4.5. `list<pthread_t*> Server::stoppedThreads` [private]

Pila de threads terminados.

Esta variable contiene una lista de threads que han finalizado su ejecución. A cada iteración del bucle que atiende conexiones, se libera toda la memoria de los threads que hay almacenados aquí.

6.1.4.6. `int Server::workerID` [private]

Contador de ID de thread.

Esta variable contiene el ID del próximo thread que se creará, por tanto, indica el número de conexiones que han sido atendidas desde el inicio del servidor

La documentación para esta clase fue generada a partir del siguiente fichero:

- `/home/imanol/devel/eclipse/PAD/JChatClient/src/include/server.h`

6.2. Referencia de la Clase Socket

Clase del socket.

```
#include <Socket.h>
```

Métodos públicos

- `Socket ()`
Constructor de la clase `Socket`.
- `void Create ()`
Método para crear sockets.
- `void Bind (string address, int port)`
Método para hacer bind a una dirección y puerto.
- `void Listen (int backlog)`
Método para escuchar conexiones.
- `void Accept (Socket &clientSock)`
Método para aceptar conexiones.
- `void Connect (string hostname, int port)`
Método para efectuar conexiones.
- `void Close ()`
Método para cerrar sockets.
- `int getSock ()`
Getter para el file descriptor del socket.
- `const Socket & operator<< (const string &)`
Método para enviar mensajes.
- `const Socket & operator>> (string &)`
Método para recibir mensajes.

Métodos privados

- `int Receive (char *buff, int length)`
Método para recibir un mensaje de longitud conocida.
- `int Send (const char *buff, int length)`
Método para enviar un mensaje de longitud conocida.

Atributos privados

- `int sock`
Descriptor del fichero del socket.
- `struct sockaddr_in sockAddr`
Estructura de dirección de socket.

6.2.1. Descripción detallada

Clase del socket.

Esta clase define un objeto con los métodos y atributos necesarios para realizar comunicación encriptada o en texto llano a través de un socket TCP/IP en un entorno UNIX abstrayendo la implementación de sockets y de encriptación al programador.

6.2.2. Documentación del constructor y destructor

6.2.2.1. `Socket::Socket ()`

Constructor de la clase `Socket`.

Inicializa el file descriptor del socket y prepara la memoria donde se almacenarán las llaves públicas.

6.2.3. Documentación de las funciones miembro

6.2.3.1. `void Socket::Accept (Socket & clientSock)`

Método para aceptar conexiones.

Este método bloquea el thread que lo ejecuta hasta que recibe una conexión entrante, que almacena como instancia de la clase `Socket` en la referencia proporcionada por argumento.

Parámetros

<code>clientSock</code>	Instancia de <code>Socket</code> que comunica con el cliente entrante
-------------------------	---

6.2.3.2. `void Socket::Bind (string address, int port)`

Método para hacer bind a una dirección y puerto.

Este método asigna el socket a una dirección IP (y por tanto, a una interfaz de red en concreto) y un puerto, para que posteriormente actúe de servidor escuchando conexiones.

Parámetros

<code>address</code>	Dirección IP asignada
<code>port</code>	Puerto asignado

6.2.3.3. `void Socket::Close ()`

Método para cerrar sockets.

Este método cierra el socket para que no se pueda escribir ni leer más en él, para liberarlo del kernel y terminar la conexión TCP.

6.2.3.4. void Socket::Connect (string *hostname*, int *port*)

Método para efectuar conexiones.

Este método sirve a los sockets que actúan como cliente para poder efectuar conexiones a otro socket que esté escuchando en la dirección y puerto especificados y atiende conexiones.

Parámetros

<i>hostname</i>	Hostname al que conectarse
<i>port</i>	Puerto al que conectarse

6.2.3.5. void Socket::Create ()

Método para crear sockets.

Crea un file descriptor para un socket que por defecto no está conectado ni asignado a ninguna dirección.

6.2.3.6. int Socket::getSock ()

Getter para el file descriptor del socket.

Este método devuelve el file descriptor del socket.

6.2.3.7. void Socket::Listen (int *backlog*)

Método para escuchar conexiones.

Este método configura el socket para que se ponga en modo escucha y así pueda atender conexiones entrantes.

Parámetros

<i>backlog</i>	Número máximo de conexiones en espera
----------------	--

6.2.3.8. const Socket & Socket::operator<< (const string & *text*)

Método para enviar mensajes.

Este método envía el mensaje que se le proporciona a través del [Socket](#) con o sin encriptación según las [opciones de compilación](#) usando el [protocolo](#) implementado.

Parámetros

<i>text</i>	Mensaje a enviar
-------------	------------------

6.2.3.9. const Socket & Socket::operator>> (std::string & *text*)

Método para recibir mensajes.

Este método recibe un mensaje de longitud arbitraria con o sin encriptación según las [opciones de compilación](#) usando el [protocolo](#) implementado.

Parámetros

<i>text</i>	Mensaje a recibir
-------------	-------------------

6.2.3.10. `int Socket::Receive (char * buff, int length) [private]`

Método para recibir un mensaje de longitud conocida.

Este método se usa para recibir un mensaje de la longitud que se especifica por argumento y almacenarlo en el buffer proporcionado.

Este método garantiza que todo el mensaje se recibirá entero aunque la red no admita una longitud de paquete tan grande.

Parámetros

<code>buff</code>	Buffer donde se almacena el mensaje recibido
<code>length</code>	Longitud del mensaje a recibir

6.2.3.11. `int Socket::Send (const char * buff, int length) [private]`

Método para enviar un mensaje de longitud conocida.

Este método se usa para enviar un mensaje de la longitud especificada en el argumento y devuelve el número de bytes enviados.

Este método garantiza que todo el mensaje se enviará entero aunque la red no admita una longitud de paquete tan grande.

Parámetros

<code>buff</code>	Buffer con el mensaje a enviar
<code>length</code>	Longitud del mensaje

6.2.4. Documentación de los datos miembro

6.2.4.1. `int Socket::sock [private]`

Descriptor del fichero del socket.

Esta variable contiene el file descriptor del socket abierto por el SO.

6.2.4.2. `struct sockaddr_in Socket::sockAddr [private]`

Estructura de dirección de socket.

Este struct es usado por el SO para gestionar la dirección del socket abierto.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- [/home/imanol/devel/eclipse/PAD/JChatClient/src/include/Socket.h](#)
- [/home/imanol/devel/eclipse/PAD/JChatClient/src/Socket.cpp](#)

6.3. Referencia de la Clase SocketException

Clase de Excepción de Sockets.

```
#include <SocketException.h>
```

Métodos públicos

- [SocketException](#) (string m)

Constructor de la clase.

- `~SocketException ()`
- `string description ()`

Getter del atributo message.

Atributos privados

- `string message`

Mensaje asociado a la excepción producida.

6.3.1. Descripción detallada

Clase de Excepción de Sockets.

Definición de las excepciones lanzadas por la librería de comunicación TCP/IP

6.3.2. Documentación del constructor y destructor

6.3.2.1. `SocketException::SocketException (string m) [inline]`

Constructor de la clase.

Inicializa el mensaje

Parámetros

<i>m</i>	Mensaje inicial
----------	-----------------

6.3.2.2. `SocketException::~~SocketException () [inline]`

6.3.3. Documentación de las funciones miembro

6.3.3.1. `string SocketException::description () [inline]`

Getter del atributo message.

Devuelve el mensaje asociado a la excepción producida

6.3.4. Documentación de los datos miembro

6.3.4.1. `string SocketException::message [private]`

Mensaje asociado a la excepción producida.

La documentación para esta clase fue generada a partir del siguiente fichero:

- `/home/imanol/devel/eclipse/PAD/JChatClient/src/include/SocketException.h`

6.4. Referencia de la Estructura `thread_args`

Argumentos de los threads.

```
#include <server.h>
```

Atributos públicos

- pthread_mutex_t * **mutex**
Variable de control de la exclusión mútua entre threads.
- pthread_cond_t * **condition**
Variable de notificación a otros threads.
- pthread_t * **thread**
Puntero al thread.
- **Socket** * **s**
Puntero al socket.
- **Server** * **serv**
Puntero al servidor.
- int **id**
ID del thread.

6.4.1. Descripción detallada

Argumentos de los threads.

Este struct define los argumentos que recibe un thread abierto por la aplicación servidor al recibir una conexión entrante

Autor

Imanol Barba Sabariego

Fecha

11/06/2013

6.4.2. Documentación de los datos miembro

6.4.2.1. pthread_cond_t* thread_args::condition

Variable de notificación a otros threads.

Esta variable se usa para notificar a otros threads cuando deben realizar otras acciones.

Actualmente se usa para notificar al thread principal cuando el thread que lo invoca ha terminado, en caso de que el principal haya quedado bloqueado y no admita más conexiones.

6.4.2.2. int thread_args::id

ID del thread.

Identifica al thread con un ID único

6.4.2.3. pthread_mutex_t* thread_args::mutex

Variable de control de la exclusión mútua entre threads.

Esta variable se usa para bloquear otros threads en operaciones de exclusion mútua donde se modifican variables compartidas

6.4.2.4. `Socket*` `thread_args::s`

Puntero al socket.

Esta variable representa el puntero al socket que proviene de la conexión entrante recibida por el servidor. Con este, el thread puede recibir y enviar los datos.

6.4.2.5. `Server*` `thread_args::serv`

Puntero al servidor.

Esta variable representa el puntero al servidor de la aplicación. Con este puntero, los threads se mueven a la pila de threads terminados una vez terminan la ejecución para que el thread principal (el propio servidor), vaya liberando la memoria asignada.

6.4.2.6. `pthread_t*` `thread_args::thread`

Puntero al thread.

Esta variable representa el puntero del propio thread. Al finalizar, este enviará su puntero a la lista de threads terminados, donde la memoria asignada al thread se destruirá.

La documentación para esta estructura fue generada a partir del siguiente fichero:

- `/home/imanol/devel/eclipse/PAD/JChatClient/src/include/`[server.h](#)

Capítulo 7

Documentación de archivos

7.1. Referencia del Archivo `/home/imanol/devel/eclipse/PAD/JChatClient/src/client.cpp`

Fichero de implementación de un cliente.

```
#include "Socket.h"  
#include <iostream>  
#include "SocketException.h"  
#include <sstream>  
#include <signal.h>  
#include <cstdlib>  
#include <sys/time.h>
```

Funciones

- void `exitClient` (int signal)
Método para terminar el cliente.
- void `processCMD` (string cmd)
- int `main` ()
Método principal del cliente.

Variables

- bool `connected`

7.1.1. Descripción detallada

Fichero de implementación de un cliente.

Autor

Imanol Barba Sabariego

Fecha

13/06/2013

En este fichero se implementa un cliente para poder usar con el servidor creado, usando la clase `Socket`.

7.1.2. Documentación de las funciones

7.1.2.1. void exitClient (int *signal*)

Método para terminar el cliente.

Este método se usa para terminar el cliente inmediatamente en el caso que el servidor cierre la conexión de forma inesperada, capturando el signal SIGPIPE.

Parámetros

<i>signal</i>	Parámetro que captura el signal recibido
---------------	--

7.1.2.2. int main ()

Método principal del cliente.

Este método inicializa el [Socket](#), establece la conexión y realiza las acciones que se le hayan programado para comunicarse con el servidor.

7.1.2.3. void processCMD (string *cmd*)

7.1.3. Documentación de las variables

7.1.3.1. bool connected

7.2. Referencia del Archivo /home/imanol/devel/eclipse/PAD/JChatClient/src/client.dox

7.2.1. Descripción detallada

Autor

Imanol Barba Sabariego

Fecha

13/06/2013

7.3. Referencia del Archivo /home/imanol/devel/eclipse/PAD/JChatClient/src/howto.dox

7.3.1. Descripción detallada

Autor

Imanol Barba Sabariego

Fecha

13/06/2013

7.4. Referencia del Archivo /home/imanol/devel/eclipse/PAD/JChatClient/src/include/server.h

Header de la clase [Server](#).

```
#include "Socket.h"
#include "SocketException.h"
#include <iostream>
#include <sstream>
#include <signal.h>
#include <fstream>
#include <list>
```

Clases

- class [Server](#)
Clase de aplicación servidor.
- struct [thread_args](#)
Argumentos de los threads.

'defines'

- #define [N](#) 5
Numero de conexiones permitidas activas (en espera o activas)
- #define [CONFFILE](#) "socket.conf"
Ruta al fichero de configuración.

Funciones

- void [killThread](#) ([thread_args](#) *t_arg)
Método de finalización de Threads.
- void * [WorkerThread](#) (void *args)
Método gestión de conexiones.
- void [processText](#) (string *str)
Método auxiliar de procesado de texto.
- bool [readConf](#) (string *ip, int *port)
Método de lectura de configuración.

7.4.1. Descripción detallada

Header de la clase [Server](#).

Autor

Imanol Barba Sabariego

Fecha

11/06/2013

En este fichero se define la clase [Server](#) y algunos métodos globales usados por ésta para la gestión de threads y otros aspectos.

7.4.2. Documentación de los 'defines'

7.4.2.1. #define CONFFILE "socket.conf"

Ruta al fichero de configuración.

Ruta relativa o absoluta al fichero de configuración, de no existir o ser inválido el programa no funcionará.

7.4.2.2. #define N 5

Numero de conexiones permitidas activas (en espera o activas)

Esta constante controla cuantas conexiones puede haber en espera o cuantas puede haber establecias en cualquier momento: habrá N activas y N en espera como mucho, no N en espera o activas.

7.4.3. Documentación de las funciones

7.4.3.1. void killThread (thread_args * t_arg)

Método de finalización de Threads.

Éste método se ejecuta para liberar la memoria de los argumentos del thread y mandarlo a la pila de threads terminados.

Parámetros

<i>t_arg</i>	Puntero al struct que contiene los argumentos del thread
--------------	--

7.4.3.2. void processText (string * str)

Método auxiliar de procesado de texto.

Éste método usa para procesar las entradas de texto del fichero de configuración para adaptarlas a un formato adecuado.

De momento se limita a eliminar whitespace (tabulaciones, saltos de línea y espacios).

Parámetros

<i>str</i>	Línea de texto a procesar
------------	---------------------------

7.4.3.3. bool readConf (string * ip, int * port)

Método de lectura de configuración.

Éste método lee el fichero de configuración especificado para obtener parámetros de configuración para el funcionamiento del servidor.

Parámetros

<i>ip</i>	Puntero donde se almacena la IP leída
<i>port</i>	Puntero donde se almacena el puerto leído

7.4.3.4. void* WorkerThread (void * args)

Método gestión de conexiones.

Éste método es el que los threads ejecutan al crearse, aquí es donde se define el comportamiento del servidor, ya que cada conexión se gestionará como aquí se detalla.

Por defecto, el comportamiento que lleva programado es el de un servidor "echo", esperará que el cliente le envíe un mensaje y responderá con el mismo mensaje.

Parámetros

<i>args</i>	Puntero al struct de los argumentos del thread casteado a tipo void*
-------------	--

7.5. Referencia del Archivo /home/imanol/devel/eclipse/PAD/JChatClient/src/include/Socket.h

Header de la clase [Socket](#).

```
#include <iostream>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <string>
#include <sys/types.h>
#include <unistd.h>
```

Clases

- class [Socket](#)

Clase del socket.

7.5.1. Descripción detallada

Header de la clase [Socket](#).

Autor

Imanol Barba Sabariego

Fecha

12/06/2013

En este fichero se define la clase [Socket](#), que es la clase que se abstraer toda la comunicación con sockets al programador

7.6. Referencia del Archivo /home/imanol/devel/eclipse/PAD/JChatClient/src/include/SocketException.h

Header de excepción de Sockets.

```
#include <string>
```

Clases

- class [SocketException](#)

Clase de Excepción de Sockets.

7.6.1. Descripción detallada

Header de excepción de Sockets.

Autor

Imanol Barba Sabariego

Fecha

10/06/2013

En este fichero se define la clase [SocketException](#) para el control de excepciones producidas por la librería.

7.7. Referencia del Archivo [/home/imanol/devel/eclipse/PAD/JChatClient/src/mainpage.dox](#)

7.8. Referencia del Archivo [/home/imanol/devel/eclipse/PAD/JChatClient/src/makefile.dox](#)

7.9. Referencia del Archivo [/home/imanol/devel/eclipse/PAD/JChatClient/src/Socket.cpp](#)

Fichero de implementación de la clase [Socket](#).

```
#include "Socket.h"  
#include "SocketException.h"  
#include <sstream>  
#include <strings.h>  
#include <cstdlib>  
#include <netinet/tcp.h>
```

7.9.1. Descripción detallada

Fichero de implementación de la clase [Socket](#).

Autor

Imanol Barba Sabariego

Fecha

13/06/2013

En este fichero se implementan los métodos de la clase [Socket](#) definidos en [Socket.h](#)

Índice alfabético

- ~SocketException
 - SocketException, 18
- /home/imanol/devel/eclipse/PAD/JChatClient/src/
 - Socket.cpp, 26
- /home/imanol/devel/eclipse/PAD/JChatClient/src/client.
 - cpp, 21
- /home/imanol/devel/eclipse/PAD/JChatClient/src/client.
 - dox, 22
- /home/imanol/devel/eclipse/PAD/JChatClient/src/howto.
 - dox, 22
- /home/imanol/devel/eclipse/PAD/JChatClient/src/include/
 - Socket.h, 25
- /home/imanol/devel/eclipse/PAD/JChatClient/src/include/
 - SocketException.h, 25
- /home/imanol/devel/eclipse/PAD/JChatClient/src/include/server.
 - h, 22
- /home/imanol/devel/eclipse/PAD/JChatClient/src/mainpage.
 - dox, 26
- /home/imanol/devel/eclipse/PAD/JChatClient/src/makefile.
 - dox, 26

- Accept
 - Socket, 15

- Bind
 - Socket, 15

- CONFFILE
 - server.h, 24
- client.cpp
 - connected, 22
 - exitClient, 22
 - main, 22
 - processCMD, 22
- Close
 - Socket, 15
- condition
 - thread_args, 19
- Connect
 - Socket, 15
- connected
 - client.cpp, 22
- Create
 - Socket, 16

- description
 - SocketException, 18

- exitClient
 - client.cpp, 22

- freeRAM
 - Server, 12

- getNWorkers
 - Server, 12
- getSock
 - Socket, 16
- getStartedThreads
 - Server, 12
- getStoppedThreads
 - Server, 12

- id
 - thread_args, 19

- killThread
 - server.h, 24

- Listen
 - Socket, 16

- main
 - client.cpp, 22
- message
 - SocketException, 18
- mutex
 - thread_args, 19

- N
 - server.h, 24
- nWorkers
 - Server, 13

- operator<<
 - Socket, 16
- operator>>
 - Socket, 16

- processCMD
 - client.cpp, 22
- processText
 - server.h, 24

- readConf
 - server.h, 24
- Receive
 - Socket, 16
- requestExit
 - Server, 12

- s

- thread_args, 19
- Send
 - Socket, 17
- serv
 - thread_args, 20
- Server, 11
 - freeRAM, 12
 - getNWorkers, 12
 - getStartedThreads, 12
 - getStoppedThreads, 12
 - nWorkers, 13
 - requestExit, 12
 - Server, 12
 - setNWorkers, 13
 - shutdownServer, 13
 - ss, 13
 - startServer, 13
 - startedThreads, 13
 - stoppedThreads, 13
 - workerID, 14
- server.h
 - CONFFILE, 24
 - killThread, 24
 - N, 24
 - processText, 24
 - readConf, 24
 - WorkerThread, 24
- setNWorkers
 - Server, 13
- shutdownServer
 - Server, 13
- sock
 - Socket, 17
- sockAddr
 - Socket, 17
- Socket, 14
 - Accept, 15
 - Bind, 15
 - Close, 15
 - Connect, 15
 - Create, 16
 - getSock, 16
 - Listen, 16
 - operator<<, 16
 - operator>>, 16
 - Receive, 16
 - Send, 17
 - sock, 17
 - sockAddr, 17
 - Socket, 15
- SocketException, 17
 - ~SocketException, 18
 - description, 18
 - message, 18
 - SocketException, 18
 - SocketException, 18
- ss
 - Server, 13
- startServer
 - Server, 13
- startedThreads
 - Server, 13
- stoppedThreads
 - Server, 13
- thread
 - thread_args, 20
- thread_args, 18
 - condition, 19
 - id, 19
 - mutex, 19
 - s, 19
 - serv, 20
 - thread, 20
- workerID
 - Server, 14
- WorkerThread
 - server.h, 24