Socket

Generado por Doxygen 1.8.3.1

Martes, 26 de Noviembre de 2013 11:34:23

Índice general

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 incializada 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

Fichero de configuración

Descripción de la sintaxis y parámetros del fichero de configuración

```
    A continuación se detalla el uso del fichero de configuración del servidor.
    <br><br>
```

3.1. Sintáxis

La sintaxis es muy sencilla:

```
PARÁMETRO<WHITESPACE>=<WHITESPACE>VALOR

            donde <WHITESPACE> pueden ser espacios, tabulaciones o saltos de línea.
            <br>
```

3.2. Parámetros

A continuación se detallan los posibles parámetros y qué valores aceptan.

3.2.1. bind-ip

Contiene la dirección IP que será asigada al Socket del servidor, por tanto, la dirección donde escuchará conexiones. Acepta un string del tipo W.X.Y.Z donde W,X,Y,Z son números de 0 a 255.

3.2.2. port

El puerto donde el servidor acepta conexiones. Acepta un número de 1 a 65535.

3.3. Ejemplos

```
bind-ip = 127.0.0.1
port = 3001
```

Capítulo 4

HOWTO

Descripción de funcionamiento y uso de la librería.

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

4.1. 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 comprovar 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.

4.1.1. Modo en texto llano

Este modo transmite los carácteres 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.

4.1.2. 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 carácteres (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 carácteres 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.

4.2. Compilación

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

4.2.1. 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: Fichero de configuración).
- 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

4.2.2. 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

4.3. Ejemplos

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

4.3.1. Programar un servidor

Véase: Client

4.3.2. Programar un cliente

Véase: Server

Capítulo 5

Server

Ejemplo de aplicación servidor

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

Server *serv;

void killThread(thread_args *t_arg)
{
        (t_arg->s)->Close();
        pthread_mutex_lock(t_arg->mutex);
        t_arg->serv->setNWorkers(t_arg->serv->getNWorkers()-1);
        cout « "Worker " « t_arg->id « ": connection terminated" « endl;
        pthread_mutex_unlock(t_arg->mutex);
        pthread_cond_signal(t_arg->condition);
        t_arg->serv->getStartedThreads()->remove(t_arg->
      thread);
        t_arg->serv->getStoppedThreads()->push_back(t_arg->
      thread);
        if(t_arg->s != 0)
        {
                delete t_arg->s;
                t_arg->s = 0;
        }
        if(t_arg != 0)
        {
                delete t_arg;
                t_arg = 0;
        }
        pthread_exit(NULL);
}

void *WorkerThread(void* args)
{
        struct thread_args *t_arg = (struct thread_args*)args;
        while(true)
        {
                string message;
                *(t_arg->s) » message;
                cout « "Worker " « t_arg->id « " received: " « message « endl;
                string send = "You said: ";
                send += message;
                *(t_arg->s) « send;
        }
        killThread(t_arg);
}

void stopServer(int signal)
{
        serv->requestExit();
}

void processText(string *str)
{
        for(int i = 0; i < str->length(); i++)
        {
                if((*str)[i] == 32 || (*str)[i] == 10 || (*str)[i] == 11)
                {
                        str->erase(i--,1);
                }
        }
}
```

```
bool readConf(string *ip, int *port)
{
        *ip = "";
        *port = 0;
        ifstream confFile;
        confFile.open(CONFFILE);
        if(!confFile.is_open())
        {
                cout « "Error opening configuration file" « endl;
                return false;
        }
        string parameter;
        while(true)
        {
                getline(confFile, parameter, '=');
                processText(&parameter);
                if(confFile.eof())
                {
                        break;
                }
                if(parameter == "bind-ip")
                {
                        confFile » *ip;
                }
                else if(parameter == "port")
                {
                        confFile » *port;
                }
        }
        confFile.close();
        if(*ip == "" || *port == 0)
        {
                return false;
        }
        return true;
}

int main()
{
        string ip;
        int port;
        serv = new Server();
        signal(SIGINT, stopServer);
        if(!readConf(&ip,&port))
        {
                cout « "Configuration couldn't be loaded" « endl;
                return -1;
        }
        serv->startServer(ip,port);
        delete serv;
        return 0;
}
```

Capítulo 6

Índice de clases

6.1. Lista de clases

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

Capítulo 7

Indice de archivos

7.1. Lista de archivos

Lista de todos los archivos con descripciones breves:

Capítulo 8

Documentación de las clases

8.1. 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 memória 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.

8.1.1. 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

8.1.2. Documentación del constructor y destructor

8.1.2.1. Server::Server ( ) [inline]

Constructor de la clase Server.

Incializa los argumentos inciales del servidor

8.1.3. Documentación de las funciones miembro

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

Método para liberar memória de threads.

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

Parámetros

threadList	Contenedor con los threads a liberar

8.1.3.2. 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.

8.1.3.3. 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

8.1.3.4. 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

8.1.3.5. void Server::requestExit ( )

Método para terminar el servidor.

Este método inicia la secuencia de finalización del servidor

8.1.3.6. 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

n	Nuevo número de threads activos

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

Método de inicialización del servidor.

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

Parámetros

i	IP donde se aceptan las conexiones
p	Puerto donde se aceptan las conexiones

8.1.4. Documentación de los datos miembro

8.1.4.1. 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 antendidas simultáneamente.

8.1.4.2. 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.

8.1.4.3. 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 el 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 comuncación.

8.1.4.4. 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 produciria al finalizar el programa, por tanto no es relevante.

8.1.4.5. 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.

8.1.4.6. 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 de los siguientes ficheros:

- /home/imanol/devel/eclipse/PAD/Socket/src/include/server.h
- /home/imanol/devel/eclipse/PAD/Socket/src/server.cpp

8.2. 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.

8.2.1. 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.

8.2.2. Documentación del constructor y destructor

8.2.2.1. Socket::Socket ( )

Constructor de la clase Socket.

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

8.2.3. Documentación de las funciones miembro

8.2.3.1. 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

clientSock	Instáncia de Socket que comunica con el cliente entrante

8.2.3.2. 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

address	Dirección IP asignada
port	Puerto asignado

8.2.3.3. 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.

8.2.3.4. 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 atienda conexiones.

Parámetros

hostname	Hostname al que conectarse
port	Puerto al que conectarse

8.2.3.5. 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.

8.2.3.6. int Socket::getSock ( )

Getter para el file descriptor del socket.

Este método devuelve el file descriptor del socket.

8.2.3.7. 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

backlog	Número máximo de conexiones **en espera**

8.2.3.8. 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

text	Mensaje a enviar

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

Método para recibir mensajes.

Este método recibe un mensaje de longitud arbitrária con o sin encriptación según las opciones de compilación usando el protocolo implementado.

Parámetros

text	Mensaje a recibir

8.2.3.10. 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

buff	Buffer donde se almacena el mensaje recibido
length	Longitud del mensaje a recibir

8.2.3.11. 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 especifidada 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

buff	Buffer con el mensaje a enviar
length	Longitud del mensaje

8.2.4. Documentación de los datos miembro

8.2.4.1. int Socket::sock [private]

Descriptor del fichero del socket.

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

8.2.4.2. 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/Socket/src/include/Socket.h
- /home/imanol/devel/eclipse/PAD/Socket/src/Socket.cpp

8.3. 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.

8.3.1. Descripción detallada

Clase de Excepción de Sockets.

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

8.3.2. Documentación del constructor y destructor

8.3.2.1. SocketException::SocketException (string *m*) [inline]

Constructor de la clase.

Inicializa el mensaje

Parámetros

m	Mensaje inicial

8.3.2.2. SocketException::~SocketException ( ) [inline]

8.3.3. Documentación de las funciones miembro

8.3.3.1. string SocketException::description ( ) [inline]

Getter del atributo message.

Devuelve el mensaje asociado a la excepción producida

8.3.4. Documentación de los datos miembro

8.3.4.1. 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/Socket/src/include/SocketException.h

8.4. 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.

8.4.1. 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

8.4.2. Documentación de los datos miembro

8.4.2.1. 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.

8.4.2.2. int thread_args::id

ID del thread.

Identifica al thread con un ID único

8.4.2.3. 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

8.4.2.4. 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.

8.4.2.5. 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.

8.4.2.6. 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/Socket/src/include/server.h

Capítulo 9

Documentación de archivos

9.1. Referencia del Archivo /home/imanol/devel/eclipse/PAD/Socket/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.
- int main ()

  Método principal del cliente.

9.1.1. 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.

9.1.2. Documentación de las funciones

9.1.2.1. void exitClient (int *signal*)

Método para terminar el cliente.

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

Parámetros

signal	Parámetro que captura el signal recibido

9.1.2.2. 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.

9.2. Referencia del Archivo /home/imanol/devel/eclipse/PAD/Socket/src/client.dox

9.2.1. Descripción detallada

Autor

Imanol Barba Sabariego

Fecha

13/06/2013

9.3. Referencia del Archivo /home/imanol/devel/eclipse/PAD/Socket/src/conf.dox

9.3.1. Descripción detallada

Autor

Imanol Barba Sabariego

Fecha

13/06/2013

9.4. Referencia del Archivo /home/imanol/devel/eclipse/PAD/Socket/src/howto.dox

9.4.1. Descripción detallada

Autor

Imanol Barba Sabariego

Fecha

13/06/2013

9.5. Referencia del Archivo /home/imanol/devel/eclipse/PAD/Socket/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.

9.5.1. 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.

9.5.2. Documentación de los 'defines'

9.5.2.1. #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.

9.5.2.2. #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.

9.5.3. Documentación de las funciones

9.5.3.1. 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

t_arg	Puntero al struct que contiene los argumentos del thread

9.5.3.2. 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

str	Línea de texto a procesar

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

Método de lectura de configuración.

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

Parámetros

ip	Puntero donde se almacena la IP leída
port	Puntero donde se almacena el puerto leído

9.5.3.4. 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

args	Puntero al struct de los argumentos del thread casteado a tipo void*

9.6. Referencia del Archivo /home/imanol/devel/eclipse/PAD/Socket/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.

9.6.1. 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

9.7. Referencia del Archivo /home/imanol/devel/eclipse/PAD/Socket/src/include/SocketException.h

Header de excepción de Sockets.

```
#include <string>
```

Clases

- class SocketException

  Clase de Excepción de Sockets.

9.7.1. 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.

9.8. Referencia del Archivo /home/imanol/devel/eclipse/PAD/Socket/src/main_server.cpp

Implementación del servidor.

```
#include "server.h"
#include <sys/types.h>
#include <unistd.h>
#include <sstream>
```

Funciones

- void killThread (thread_args *t_arg)

  Método de finalización de Threads.
- void * WorkerThread (void *args)

  Método gestión de conexiones.
- void stopServer (int signal)

  Método de detención del servidor.
- void processText (string *str)

  Método auxiliar de procesado de texto.
- bool readConf (string *ip, int *port)

  Método de lectura de configuración.
- int main ()

  Método principal del servidor.

Variables

- Server * serv

  Instancia de la clase Server.

9.8.1. Descripción detallada

Implementación del servidor.

Autor

Imanol Barba Sabariego

Fecha

13/06/2013

En este fichero se implementa un servidor TCP/IP usando las clases Socket y Server.

9.8.2. Documentación de las funciones

9.8.2.1. 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

t_arg	Puntero al struct que contiene los argumentos del thread

9.8.2.2. int main ( )

Método principal del servidor.

Lee la configuración usando readConf() y incializa el servidor.

9.8.2.3. 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

str	Línea de texto a procesar

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

Método de lectura de configuración.

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

Parámetros

ip	Puntero donde se almacena la IP leída
port	Puntero donde se almacena el puerto leído

9.8.2.5. void stopServer (int *signal*)

Método de detención del servidor.

Este método detiene el servidor al recibir el signal SIGINT del SO.

Parámetros

signal	Parámetro que captura el signal recibido

9.8.2.6. 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

args	Puntero al struct de los argumentos del thread casteado a tipo void*

9.8.3. Documentación de las variables

9.8.3.1. Server* serv

Instancia de la clase Server.

Esta variable contiene la instancia de la clase Server que aquí se usa e inicializa.

Se trata de una variable global para poder así usarla en las funciones que reciban signals del SO.

9.9. Referencia del Archivo /home/imanol/devel/eclipse/PAD/Socket/src/mainpage.dox

9.10. Referencia del Archivo /home/imanol/devel/eclipse/PAD/Socket/src/makefile.dox

9.11. Referencia del Archivo /home/imanol/devel/eclipse/PAD/Socket/src/server.cpp

Fichero de implementación de la clase Server.

```
#include "server.h"
```

9.11.1. Descripción detallada

Fichero de implementación de la clase Server.

Autor

Imanol Barba Sabariego

Fecha

13/06/2013

En este fichero se implementan los métodos de la clase Server definidos en server.h

9.12. Referencia del Archivo /home/imanol/devel/eclipse/PAD/Socket/src/server.dox

9.12.1. Descripción detallada

Autor

Imanol Barba Sabariego

Fecha

13/06/2013

9.13. Referencia del Archivo /home/imanol/devel/eclipse/PAD/Socket/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>
```

9.13.1. 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