|  |  |
| --- | --- |
|  | Realización de un *proxy* |
|  |  |
| P.1.3 | Laboratorio de Redes y Servicios |
|  | INTEGRANTES DEL GRUPO:  Nombre: MANUEL MONTOYA CATALÁ  Nombre: ANDRÉS BEATO OLLERO |

ÍNDICE

[INTRODUCCIÓN 2](#_Toc388111358)

[1.1 Introducción HTTP: 3](#_Toc388111359)

[1.2 Modelo de conexiones TCP realizadas por HTTP: 4](#_Toc388111360)

[1.3 Solución al tipo de conexiones: 5](#_Toc388111361)

[Implementación Básica del proxy 6](#_Toc388111362)

[Mejoras 8](#_Toc388111363)

[ejEMPLO de EJECUCIÓN 10](#_Toc388111364)

[Codigo Servidor bbdd 11](#_Toc388111365)

[Codigo Servidor PROXY 12](#_Toc388111366)

[SUITE DE Funciones 18](#_Toc388111367)

[Archivo de cabeceras. 22](#_Toc388111368)

[BIBLIOGRAFIA 23](#_Toc388111369)

# INTRODUCCIÓN

En esta práctica se va a realizar una implementación propietaria de un servidor proxy web. A grandes

rasgos, un proxy es un punto intermedio entre un ordenador conectado a Internet y el servidor que está

accediendo. Cuando se navega empleando un proxy, el navegador en realidad no accede al servidor web

directamente, sino que envía una solicitud al proxy y es este quien se conecta con el servidor y

devuelve el resultado de dicha solicitud, actuando como intermediario de la comunicación. Entre las

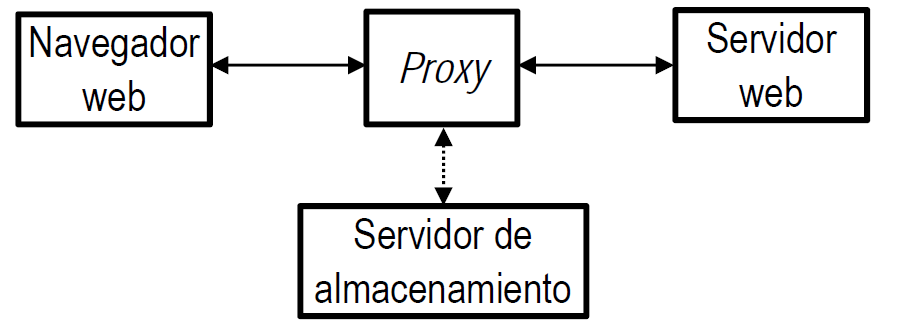
ventajas que se pueden conseguir con el uso de un proxy web se encuentran la seguridad, la rapidez

de navegación y el anonimato.

La práctica consiste en la realización de dos programas:

- Servidor proxy: Se comunica via TCP con el Servidor Web solicitado por el cliente.

- Servidor de almacenamiento: Almacenar las distintas peticiones que realizan los clientes.



Especificaciones del servidor Proxy:

- Debe comunicarse via puerto UDP 8010 con el servidor de alcamacenamiento

- No realiza caché de paginas.

- Esta escuchando en el puerto 8008 TCP

- Debe conectarse al puerto 80 del Servidor Web solicitado.

- Atender diversos usuarios mediante la funcion select().

- Si se resuelve bien la petición, es decir, obtenemos el valor “200 OK” debemos enviar un mensaje al Servidor de almacenamiento con el formato:

Fecha y hora (20 bytes) + dirección IP del cliente (20 bytes) + dirección pedida

- Controle las peticiones inválidas, de manera que cuando una petición no sea válida, devuelva el código de error “Solicitud incorrecta” (400). Es decir, cuando no nos podamos conectar al host especificado (ya sea debido al puerto o a la IP) debemos enviar al cliente un mensaje 400.

- Si el proxy está inactivo durante mas de 5 minutos debe mostrar por pantalla el mensaje

“*Proxy inactivo*”.

Especificaciones del servidor de almacenamiento:

- Este servidor estará escuchando en el puerto 8010 UDP

- Almacenará los mensajes uno tras otro en un fichero llamado “*peticiones.log”*.

Para la realizacion de la práctica hemos creado una serie de funciones para manejo de sockets y cadenas de caracteres que utilizaremos para hacer el código más legible.

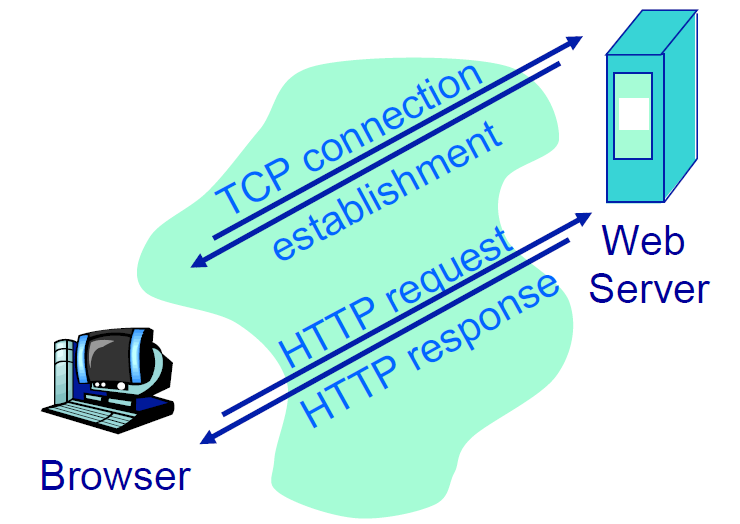
## 1.1 Introducción HTTP:

El protocolo de transferencia de hipertexto (HyperText Transfer Protocol) es un protocolo del nivel de aplicación usado para la transferencia de información entre sistemas, de forma clara y rápida. Este protocolo ha sido usado por el World-Wide Web desde 1990.

Este protocolo permite usar una serie de métodos para indicar el caracter de la petición.

Se basa en otros conceptos y estándares como Uniform Resource Identifier (URI), Uniform Resource Location (URL) y Uniform Resource Name (URN), para indicar el recurso al que hace referencia la petición.

El protocolo HTTP se basa en un paradigma de peticiones y respuestas.

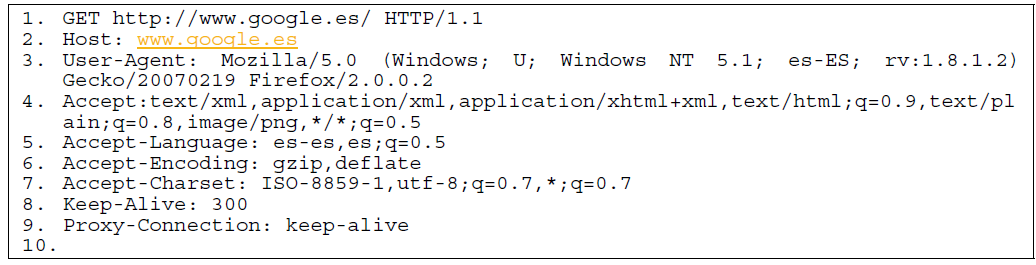
Un **cliente envía una petición** en forma de método, una URI, y una versión de protocolo seguida de los modificadores de la petición información sobre el cliente y un posible contenido al final.

El **servidor contesta con una línea de estado** que incluye la versión del protocolo y un código que indica éxito o error, seguido de la información del servidor y un posible contenido.

Si el contenido no cabe en un solo paquete, se envía en varios.

**- Peticiones:**

Los clientes envian peticiones a un Servidor HTTP para solicitar algo del mismo.

El formato de una peticion es:

**Línea de petición**

**\*(Cabeceras)**

**CRLF**

**[Contenido]**

En la linea de petición lo primero que se indica es el método que es el tipo de petición.

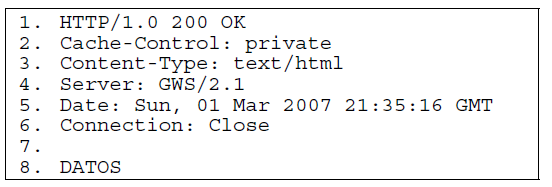
La petición más común es el GET que sirve para solicitar un recurso y tiene el formato

**GET URI Version\_HTTP**

El URI es el recurso que estamos solicitando. La principal informacion que necesitamos para establecer la conexión con el Web Server es su nombre/IP y puerto, esto lo podemos conseguir a partir del URI o tambien a través de la Cabecera “Host”.Nosotros utilizamos la segunda opción.

**- Respuestas:**

Después de recibir e interpretar una petición, el servidor debe responder con un mensaje HTTP.

Este mensaje tiene el siguiente formato:

**Línea de estado**

**\*( Cabeceras )**

**CRLF**

**[ Contenido ]**

La línea de estado indica la version del protocolo HTTP utilizado y el codigo de estado de la resolucion.

Si el cliente solicito algun recurso mediante el metido GET este estara en DATOS.

A nosotros nos interesa principalmente el codigo de estado y nada más ya que no interferiremos

## 1.2 Modelo de conexiones TCP realizadas por HTTP:

HTTP puede realizar diferentes tipos de conexiones para realizar sus peticiones:

- Peticiones no persistentes:

Solo se pide un recurso por cada conexión TCP realizada.

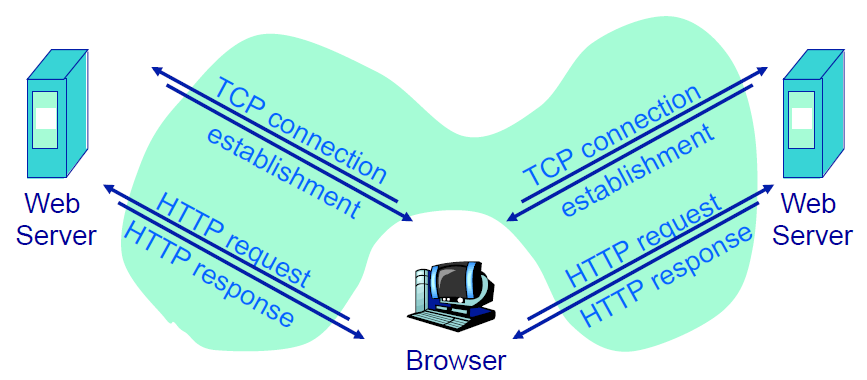
Si queremos solicitar 2 recursos tenemos que abrir 2 conexiones TCP

- Peticiones persistentes:

Podemos realizar varias peticiones HTTP sobre la misma conexión TCP.

Esto nos ahorra los tiempos de establecimiento de conexión TCP.

Ademas HTTP puede realizar diferentes conexiones TCP en paralelo para pedir varios recursos al mismo tiempo, generando varios sockets al mismo host.



Debemos tener en cuenta que existen servidores como Firefox que usarán la misma conexión TCP con el Proxy, para pedir recursos de diferentes servidores.

Sea un escenario en el cual disponemos de un cliente que solicita ver una página web en un servidor con varios recursos almacenador tambien en otros servidores, el escenario mas generico que podemos disponer es el siguiente:

**Servidor Web1**

**Servidor Web2**

**Servidor Web3**

**Cliente**

**Proxy**

Llamaremos a los socket entre Cliente y Proxy “Client sock” y a los sockets entre Servidores Web y Proxy “Server sock”. Partimos de la base de que un “Client sock” puede generar diversos “Server sock” debido a posibles multiples peticiones en paralelo. Un “Server sock” debe tener siempre asociado un “Client sock”, por otra parte, un “Client sock” puede tener asociado 0, 1 o varios “Server sock”.

## 1.3 Solución al tipo de conexiones:

La acción principal que debe realizar nuestro proxy es ser capaz de relacionar los sockets que vienen de los clientes con los sockets que genera el hacia los servidores web. Debido al gran abanico de posibilidades de conexiones, la lógica que hemos optado por realizar es totalmente generalizada y orientada a sockets.

- Cuando un usuario genera una conexión TCP (socket) con el Proxy:

🡪 Se genera un “Client sock” y guardamos contanscia de que es un cliente y su IP.

- Cuando “Client sock” solicita un recurso a algún Servidor Web:

🡪 Comprobamos si tenemos un “Server sock” asociado al “Client sock” con dicho servidor.

· Si lo tenemos 🡪 Simplemente retransmitimos el mensaje al “Server sock”.

· Si no 🡪 Creamos el “Server sock” conectandonos Servidor (si existe).

🡪 Asociamos al “Server sock” el “Client sock” (socket e IP) y viceversa.

🡪 Retransmitimos el mensaje al “Server sock”.

- Cuando recibimos información de un “Server sock”:

🡪 Simplemente retransmitimos el mensaje al “Client sock” asociado.

- Cuando un “Server sock” nos cierra la conexión:

🡪 Simplemente cerramos el socket.

- Cuando un “Client sock” nos cierra la conexión:

🡪 Cerramos dicho socket y todos los “Server sock” asociados a el.

A la hora de cerrar sockets pueden haber problemas de sincronización debido a 200 OK que quemos enviar pero ya no podemos porque el cliente cerro el socket… ( o por lo menos esa es nuestra conclusión).

Para poder realizar esta lógica necesitaremos una serie de estructuras de informacion donde poder guardar la información. Las estructuras de información más importantes son:

int related\_fd**[**MAX\_SOCKETS**];**

char Client\_IPs**[**MAX\_SOCKETS**][**30**];**

char Webs\_IPs**[**MAX\_SOCKETS**][**30**];**

La primera de ellas guarda las asociaciones entre los “Client sock” y los “Server sock”:

🡪 Por ejemplo si el “Server sock” 23, está asociado con el “Client sock” 5, tenemos que:

related\_fd**[**23**] = 5;**

De esta manera cuando nos llegue información por el “Server sock” 23, sabemos que tenemos que retrasmitirla por el “Client sock” 5.

Cuando nos llega información de un “Client sock” la reenviamos al Host que indica el mensaje HTTP, si no tenemos un “Server sock” con dicho Host, lo generamos.

Las restantes 2 estructuras simplemente guardan las IP de los sockets. La primera simplemente sirve poder tener un registro de los recursos que pide cada usuario y la segunda para saber con qué IPs está conectado cada “Client sock”.

🡪Cuando se genera un socket, ya sea de cliente o servidor, se guardan las estructuras necesarias.

🡪Cuando recibimos informacion por alguno de los sockets miramos en las estructuras si es cliente o Servidor, si es cliente obtenemos el tipo de peticion solicitada y el recurso solicitado en las variables:

char method**[**50**];** // Method receive

char Client\_Reqs**[**NUM\_USERS**][**1024**];** // Clients Requests

Si es servidor guardamos la respuesta dada en:

int response**;** // HTTP response to a request.

# Implementación Básica del proxy

En el apartado anterior hemos explicado la lógica que hemos seguido para la conexión entre los sockets entre cliente y proxy “Client sock” y aquellos entre Servidores Web y proxy “Server sock”. Este es el corazón de nuestro Servidor Proxy. A continuación explicaremos brevemente como se han realizado las especificaciones básicas pedidas, teniendo en cuenta que hemos realizado una suite de funciones que nos facilitan bastante el trabajo.

A grandes rasgos el programa inicialmente se pone a escuchar en el puerto 8008 TCP y crea el socket UDP para comunicarse en con la base de datos:

listenSocket **=** listen\_TCP\_Port **(**atoi**(**Proxy\_port**),** 20**);**

UDP\_BBDD\_sd **=** socket\_UDP **(**"127.0.0.1"**,** 8010**,** **&**BBDD\_dir\_serv**);**

Añade estos sockets y el socket del teclado (STDIN = 0) a la estructura de información del select() para la recepción de información.

FD\_SET**(**STDIN**,** **&**master\_read\_fds**);**

FD\_SET**(**listenSocket**,** **&**master\_read\_fds**);**

Después entramos en el bucle del programa capitaneado por el select() donde nos pondremos a escuchar a ver si llega información por alguno de los sockets que tenemos abiertos. Para la especificación de que si estamos 5 minutos sin recibir información imprimamos “Proxy inactivo” en pantalla, tenemos la estructura temporal time\_monsters que la igualamos a 5min en cada iteración del bucle:

time\_monster**.**tv\_sec **=** 300**;**

errs **=** select**(**fdmax**+**1**,** **&**read\_fds**,** **NULL,** **NULL,** **&**time\_monster**);**

Ahora entramos ya en la lógica principal descrita anteriormente, pongamos un par de ejemplos donde hemos quitado parte del código para centrarnos en lo que queremos mostrar:

🡪 Aceptación de un cliente:

**if** **(**i **==** listenSocket**)** **{** // NEW CONECTION CASE !!!!!

addrlen **=** **sizeof(**Clien\_addr**);**

newfd **=** accept**(**listenSocket**,** **(**struct sockaddr**\*)&**Clien\_addr**,&**addrlen**);**

strcpy**(**aux\_IP**,**inet\_ntoa**(((**struct in\_addr**)**Clien\_addr**.**sin\_addr**)))** **;**

FD\_SET**(**newfd**,** **&**master\_read\_fds**);** // add to master set

strcpy**(**Client\_IPs**[**newfd**],**aux\_IP**)** **;** // IP of the client

**}**

🡪 Conexión con un web server:

newfd **=** connect\_machine **(**web\_IP**,** aux\_port**);** // Connect to host

related\_fd**[**i**]** **=** newfd**;** // Stablish relationships

related\_fd**[**newfd**]** **=** i**;**

FD\_SET**(**newfd**,** **&**master\_read\_fds**);** // add to master set

strcpy**(**Webs\_IPs**[**newfd**],** web\_IP**);**

🡪 Manejar el caso de Peticion incorrecta por fallo de IP:

**if** **(**web\_IP**[**0**]** **==** 0**){** // If it was a bad request

bad\_request **=** 1**;**

printf**(**"IP incorrecta \n"**);**

printf**(**"PETICION INCORRECTA 400 \n\n"**);** **if** **(**sendall**(**i**,**bad\_request\_s**,**strlen**(**bad\_request\_s**),** 0**)** **==** **-**1**)** **{**

perror**(**"sendall"**);**

**}**

**break;}**

🡪 Enviar información de un socket a otro:

**if** **(**bad\_request **==** 0**){**

**if** **(**FD\_ISSET**(**related\_fd**[**i**],** **&**master\_read\_fds**)){**

**if** **(**sendall**(**related\_fd**[**i**],**recv\_buff**,**nbytes**,** 0**)** **==** **-**1**)** **{**

perror**(**"sendall"**);**

**}**

**}**

**}**

Como vemos en la función, antes de enviar la información, comprobamos que está abierto el socket al que vamos a enviársela.

🡪 Enviar información a la BBDD:

fechaActual **=** time**(**0**)** **;** // Get the time and print it

fechaPtr **=** gmtime**(&**fechaActual**)** **;**

numbytes **=** sprintf**(**send\_buff**,** "A%i-%i-%i(%i:%i:%i) "**,** fechaPtr**->**tm\_mday**,** fechaPtr**->**tm\_mon **+** 1**,** fechaPtr**->**tm\_year **+** 1900**,** fechaPtr**->**tm\_hour**,** fechaPtr**->**tm\_min**,** fechaPtr**->**tm\_sec**);**

sprintf**(**send\_buff**+**strlen**(**send\_buff**),**"%s -> %s -> %i"**,**Client\_IPs**[**related\_fd**[**i**]],**Client\_Reqs**[**related\_fd**[**i**]],**HTTP\_response**);**

numbytes **=** UDP\_send\_string**(**UDP\_BBDD\_sd**,** **&**BBDD\_dir\_serv**,** send\_buff**);**

🡪 Cerrar un “Client Sock” y todos sus “Server Sock”.

Client\_IPs**[**i**][**0**]** **=** 0**;**

close**(**i**);**

FD\_CLR**(**i**,** **&**master\_read\_fds**);**

**for** **(**j **=** 0**;** j **<** NUM\_USERS**;** j**++){**

**if** **(**related\_fd**[**j**]** **==** i**){** close**(**j**);** // bye!

FD\_CLR**(**j**,** **&**master\_read\_fds**);** // remove from master set

num\_sockets**--;**

Webs\_IPs**[**j**][**0**]** **=** 0**;**

num\_web\_sockets**--;**

related\_fd**[**j**]** **=** **-**1**;**

**}**

**}**

También llevamos la cuenta de los sockets activos en nuestro Proxy, tanto “Client sock” como “Server sock”

int num\_sockets **=** 0**;** // Number of sockets we have alive.

int num\_client\_sockets **=** 0**;**// Number of sockets we have with the clients

int num\_web\_sockets **=** 0**;**// Number of sockets we have with the webs

Cada vez que se abre o cierra un socket tenemos que dejar constancia en estas variables.

# Mejoras

En algunas de las siguientes mejoras necesitamos poder dar información al servidor a través del teclado, por lo que al igual que en la práctica anterior, incluiremos en el select() la escucha al Teclado (STDIN) y mediante la función de descomposición de cadenas **get\_word\_array()** podremos identificar los distintos comandos y parámetros introducidos.

**1)** Filtrado de Servidores Web mediante **--filtro\_servidor nombre\_servidor**

Para que solo podamos acceder al servidor dado por **nombre\_servidor** tenemos la variable

char allowed\_server**[**50**];**

Si la variable está vacía no hay filtro y si no lo está lo que haces es comprobar en cada mensaje enviado por un “Client sock” que el host destino sea el del filtro.

errs **=** 1**;**

**if** **(**allowed\_server**[**0**]** **!=**0**){** // If there is a host filter

**if** **(**strcmp**(**host\_name**,** allowed\_server**)!=** 0**){** errs **=**0**;**

**}**

**}**

Si el host no cumple el filtro no se realiza la conexión.

**2)** Limite de acceso a usuarios mediante una máscara: **--filtro\_usuario dir\_IP/mask**

Para guardar el valor de la máscara tenemos las variables:

int mask **=** **-**1**;**

char IP\_mask**[**30**];**

En la primera variable guardamos la máscara y en la segunda la IP, cuando un cliente se conecta a nosotros, generando un “Client sock” lo que hacemos es comprobar si su IP está dentro de la máscara (si hay mascara). Si no lo está, cerramos el socket.

strcpy**(**aux\_IP**,**inet\_ntoa**(((**struct in\_addr**)**Clien\_addr**.**sin\_addr**)));** // client IP

errs **=** 1**;**

**if** **(**mask **!=** **-**1**){** // If there is a mask set

**if** **(**check\_mask**(**aux\_IP**,**IP\_mask**,**mask**)==**0**){** // If we are not in the submask

close **(**newfd**);** // CLose conection

printf**(**"Usuario %s fuera de rango quiso conectarse \n"**,**aux\_IP**);**

errs **=** 0**;**

**}**

**}**

Internamente la función **check\_mask(aux\_IP,IP\_mask,mask)** convierte ambas IPs a número de 32 bits y compara sus (32 – mask) bits de mayor peso.

**3)** Limitando el número de peticiones simultáneas que puede aceptar mediante: **--max\_con NUM**

Hemos considerado el número máximo de peticiones simultaneas como el número de “Client sock”, para controlar este máximo tenemos la variable:

int max\_conn **=** **-**1**;**

Cuando recibimos un “Client sock” comprobamos el numero de sockets con clientes que tenemos y si exede el maximo impuesto pues cerramos el socket.

errs **=** 1**;**

**if** **(**max\_conn **!=** **-**1**)** **{**

**if** **(**num\_client\_sockets **>=** max\_conn**)** **{**

errs **=** 0**;**

printf**(**"Demasiadas conexiones simultadneas \n"**);**

close **(**newfd**);**

**}**

**}**

**4)** Permita que el *proxy* acceda a servidores web que se encuentren en puertos diferentes al 80, cogiendo dicho puerto de la URL.

La dirección TCP -> Host + Puerto nos lo encontramos en la segunda línea del menaje HTTP en el formato:

**Host: Nombre\_host:Puerto**

Si el puerto es el 80, el estándar de HTTP, se omite. Así pues tendremos que comprobar en el nombre del host si existe el carácter ‘:’ y de existir, separar el nombre del host del puerto y actuar en consecuencia:

errs **=** find\_char**(**host\_name**,**':'**);**

**if** **(**errs **!=-**1**)** **{** // If we have been told a port other than 80

host\_name**[**errs**]** **=** 0**;** // Erase the port part from the name

aux\_port **=** atoi**(&**host\_name**[**errs**+**1**]);** // Get the port

printf **(**"Puerto no standar %i\n"**,** aux\_port**);**

**}**

**else** **{**

aux\_port **=** 80**;**

**}**

**5)** Envió a la BBDD tanto de las peticiones como de las respuestas:

A la BBDD enviamos tanto las peticiones como las respuestas a las mismas por separado, para que la BBDD pueda diferenciarlas, añadimos una letra al comienzo del mensaje:

- R: Para peticiones

- A: Para respuestas.

fechaActual **=** time**(**0**)** **;** // Get the time and print it

fechaPtr **=** gmtime**(&**fechaActual**)** **;**

numbytes **=** sprintf**(**send\_buff**,** "R%i-%i-%i(%i:%i:%i) "**,** fechaPtr**->**tm\_mday**,** fechaPtr**->**tm\_mon **+** 1**,** **->**tm\_year **+** 1900**,** fechaPtr**->**tm\_hour**,** fechaPtr**->** tm\_min**,** fechaPtr**->**tm\_sec**);**

sprintf**(**send\_buff**+**strlen**(**send\_buff**),**"%s -> %s"**,**Client\_IPs**[**i**],**word\_array**[**1**]);**

numbytes **=** UDP\_send\_string**(**UDP\_BBDD\_sd**,** **&**BBDD\_dir\_serv**,** send\_buff**);**

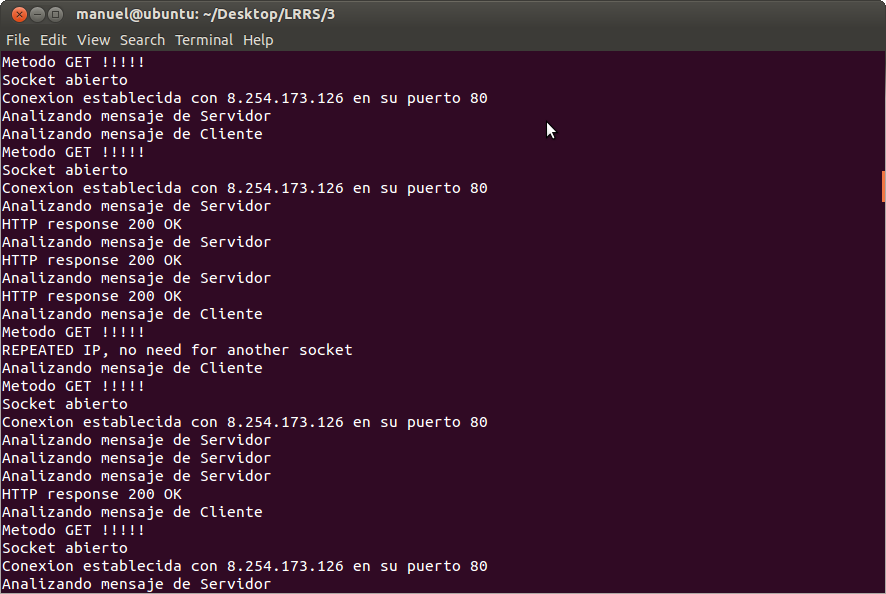
strcpy**(**Client\_Reqs**[**i**],** word\_array**[**1**]);**

Además en las respuestas también indicamos el código de la respuesta.

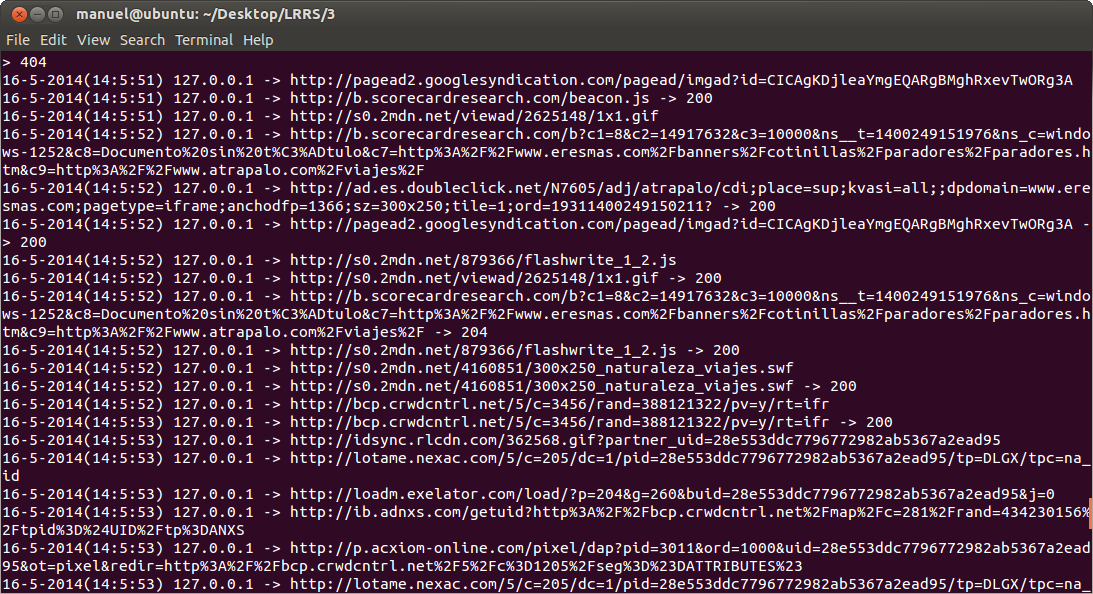
# ejEMPLO de EJECUCIÓN

Vamos a navegar un poco por Internet y ver lo que nos encontramos.

**🡪 Mensajes sobre conexiones del Servidor:**



**🡪 Mensajes sobre las URL pedidas y sus respuestas de la BBDD.**



# Codigo Servidor bbdd

El Servidor de la BBDD simplemente tiene que ponerse a escuchar en el puerto 8010 UDP y escribir en un fichero la información que se le va mandando. En nuestro caso hacemos diferenciación entre los datos que son peticiones y los datos que son respuestas. Para ello el Proxy añade una letra al principio del mensaje:

- R: Para peticiones

- A: Para respuestas.

#include "sockets\_cab.h"

#define MAXLINEA 1024

int main**(**int argc**,** char **\***argv**[]){**

int sockfd**;** // descriptor de socket

// estructura que almacena la direccion IP y numero de puerto del cliente

struct sockaddr\_in dir\_client**;**

int addr\_len**,** numbytes**,** nbytes**;**

char buf**[**MAXLINEA**];** // Datos recibidos

char aux\_buf**[**MAXLINEA**];** // Datos recibidos

sockfd **=** listen\_UDP\_Port **(**8010**);** // Escuchamos en el puerto dicho

addr\_len **=** **sizeof(**struct sockaddr**);**

//-----> File variables <------

FILE **\***pf**;**

char peticiones**[**40**]** **=** "peticiones.log"**;**

char respuestas**[**40**]** **=** "respuestas.log"**;**

printf**(**"BBDD Esperando datos ...\n"**);**

**while** **(**1**)** **{**

numbytes **=** recvfrom**(**sockfd**,** buf**,** MAXLINEA**,** 0**,** **(**struct sockaddr **\*)&**dir\_client**,** **(**socklen\_t **\*)&**addr\_len**);**

**if** **(**numbytes **==** **-**1**)** **{**

perror**(**"recvfrom"**);**

exit**(**1**);**

**}**

buf**[**numbytes**]** **=** 0**;**

**if** **((**buf**[**0**]** **==** 'R'**)||(**buf**[**0**]** **==** 'A'**)** **){**

**if** **(**buf**[**0**]** **==** 'R'**){** // Store a request

pf **=** fopen**(**peticiones**,**"a"**);**

**}** **else** **if** **(**buf**[**0**]** **==** 'A'**)** **{**

pf **=** fopen**(**respuestas**,**"a"**);**

**}**

**if** **(**pf **==** **NULL){**

perror **(**"fopen"**);**

**}**

nbytes **=** sprintf**(**aux\_buf**,**"%s \n"**,** buf**+**1**);**

// +1 for erasing the initial letter

fwrite**(**aux\_buf**,sizeof(**char**),**nbytes**,**pf**);**

printf**(**"%s"**,** aux\_buf**);**

fclose**(**pf**);**

**}**

**}**

close**(**sockfd**);**

**return** 0**;**

**}**

# Codigo Servidor PROXY

#include "sockets\_cab.h"

#define IDLE 0

#define SHOW\_USERS 1

#define CONNECT\_ALL 1

int num\_palabras**;**

int num\_users **=** 0**;** // Number of users in the system.

int fdmax**;** // maximum file descriptor number

int main**(**int argc**,** char **\*\***argv**)** **{**

int errs**,**i**,**j**;**

char **\*** Proxy\_port **=** "8008"**;**

char **\***web\_IP**;** // IP of the web we are connecting to

int nbytes**,** len**;**

int descSocket**,**listenSocket**,** newfd **;** // Socket descriptor

struct sockaddr\_in Clien\_addr**;**

socklen\_t addrlen**;**

int aux\_int **=** 0**;**

int num\_sockets **=** 0**;** // Number of sockets we have alive.

int num\_client\_sockets **=** 0**;** // Number of sockets we have with the clients

int num\_web\_sockets **=** 0**;** // Number of sockets we have with the webs

time\_t fechaActual **;** // Timing variables

struct tm **\*** fechaPtr**;**

struct timeval time\_monster**;**

char send\_buff**[**10025**];** // Send buffer

char recv\_buff**[**10025**];** // Receive buffer

char host\_name**[**50**];**

char method**[**50**];** // Method receivec

int response**;** // HTTP response to a request.

//-----> Variables for decomposing messages <------

char **\*\***word\_array**;**

char **\*\***line\_array**;**

int num\_lines**;**

int num\_words**;**

// Variables for storing the conexions between sockets 0 <-o-> 0

int related\_fd**[**NUM\_USERS**];** // This array related the fd of the proxy. If i is conected to j,

// related\_fd[i] = j and related\_fd[j] = i

int aux\_port**;**

char aux\_IP**[**100**];**

char Client\_IPs**[**NUM\_USERS**][**30**];** // CLients IPs sorted by fd

char Client\_Reqs**[**NUM\_USERS**][**1024**];** // CLients Requests sorted by

char Webs\_IPs**[**NUM\_USERS**][**30**];** // Webpages IPs sorted by fd

fd\_set master\_read\_fds**,** read\_fds**;** // Master and temp file descriptor read list for select()

fd\_set master\_write\_fds**,** write\_fds**;** // Master and temp file descriptor write list for select()

fd\_set master\_exept\_fds**,** exept\_fds**;** // Master and temp file descriptor exept list for select()

//-----> HTTP values <------

char methods**[**8**][**10**]** **=** **{**"HEAD"**,**"GET"**,**"POST"**,**"PUT"**,**"DELETE"**,**"TRACE"**,**"OPTIONS"**,**"CONNECT"**};**

char method\_req**[**30**];**

int HTTP\_response**;**

char bad\_request**;** // FLAG TO KNOW IF WE GOT A BAD RESQUEST

char bad\_request\_s**[**1000**]** **=** "HTTP/1.1 400 Bad Request/n"

"Date:Sun,18Oct201210:36:20GMT/n"

"Server:Apache/2.2.14(Win32)/n"

"Content-Length:100/n"

"Content-Type:text/html;charset=iso-8859-1/n"

"Connection:Closed/n/n"

"<html><head><title>400BadRequest</title></head>"

"<body><h1>BadRequest motherfoca</h1></body></html>"

**;**

//----> UDP server variables <----

int UDP\_BBDD\_sd**;** //descriptor de socket

struct sockaddr\_in BBDD\_dir\_serv**;** // estructura que almacena la direccion IP y numero de puerto del servidor

int numbytes**;**

// Advanced Stuff variables

int mask **=** **-**1**;**

char IP\_mask**[**30**];**

int max\_conn **=** **-**1**;** // Maximum number of client connections

char allowed\_server**[**50**];**

// \*\*\*\*\*\*\*\*\* Process the given parameters \*\*\*\*\*\*\*\*

**if** **(**argc **<** 1**)** **{** // If we are not given the basic parameters

printf**(**"Not enough parameters \n"**);**

exit**(-**1**);**

**}**

//----> Initialice variables <------

**for** **(**i **=** 0**;** i **<** NUM\_USERS**;** i**++){**

related\_fd**[**i**]** **=** **-**1**;**

Client\_IPs**[**i**][**0**]** **=** 0**;**

Webs\_IPs**[**i**][**0**]** **=** 0**;**

**}**

allowed\_server**[**0**]** **=** 0**;**

time\_monster**.**tv\_sec **=** 300**;**

//----> LISTEN AT THE PROXY PORT <------

listenSocket **=** listen\_TCP\_Port **(**atoi**(**Proxy\_port**),** 20**);**

**if** **(**listenSocket **>** fdmax**){**

fdmax **=** listenSocket**;**

**}**

num\_sockets**++;**

//----> Create the BBDD UDP socket <-----

UDP\_BBDD\_sd **=** socket\_UDP **(**"127.0.0.1"**,** 8010**,** **&**BBDD\_dir\_serv**);**

**if** **(**UDP\_BBDD\_sd **>** fdmax**){**

fdmax **=** UDP\_BBDD\_sd **;**

**}**

num\_sockets**++;**

// Reset the fds event structures

FD\_ZERO**(&**master\_read\_fds**);**

FD\_ZERO**(&**master\_write\_fds**);**

FD\_ZERO**(&**master\_exept\_fds**);**

// Add the listen Socket and STDIN to the master set

FD\_SET**(**STDIN**,** **&**master\_read\_fds**);**

FD\_SET**(**listenSocket**,** **&**master\_read\_fds**);**

**while(**1**){**

read\_fds **=** master\_read\_fds**;**

write\_fds **=** master\_write\_fds**;**

exept\_fds **=** master\_exept\_fds**;**

time\_monster**.**tv\_sec **=** 300**;**

bad\_request **=** 0**;** // NO Bad request yet

errs **=** select**(**fdmax**+**1**,** **&**read\_fds**,** **NULL,** **NULL,** **&**time\_monster**);**

**if** **(**errs **==** **-**1**)** **{**

perror**(**"select"**);**

exit**(**1**);**

**}**

**else** **if** **(**errs **==** 0**){**

printf**(**"Proxy inactivo \n"**);**

**}**

**else** **{**

// printf("Sockets -> Total %i / Clientes %i / Webs %i \n",num\_sockets,num\_client\_sockets,num\_web\_sockets);

// run through the existing connections looking for data to read

**for(**i **=** 0**;** i **<=** fdmax**;** i**++)** **{**

**if** **(**FD\_ISSET**(**i**,** **&**read\_fds**))** **{** // Check for reading on every socket

**if** **(**i **==** listenSocket**)** **{** // NEW CONECTION CASE !!!!!

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* NEW CLIENT CONNECTION \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Same client might have several socket connections.

addrlen **=** **sizeof(**Clien\_addr**);**

newfd **=** accept**(**listenSocket**,** **(**struct sockaddr **\*)&**Clien\_addr**,&**addrlen**);**

errs **=** 1**;**

**if** **(**max\_conn **!=** **-**1**)** **{**

**if** **(**num\_client\_sockets **>=** max\_conn**)** **{**

errs **=** 0**;**

printf**(**"Demasiadas conexiones simultadneas \n"**);**

**}**

**}**

**if** **(**errs **==** 0**)** **{**

close **(**newfd**);**

**}**

**else{**

**if** **(**newfd **==** **-**1**)** **{** //\*\*\*\*\*\* ERROR \*\*\*\*\*\*\*\*\*\*\*

perror**(**"accept"**);**

**}**

**else** **{** //\*\*\*\*\* GOOD CONECTION \*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \*\*\*\*\*\*\*\*\*\*\*\* CHECK THE USER IS INSIDE THE MASK \*\*\*\*\*\*\*\*\*\*\*

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

strcpy**(**aux\_IP**,**inet\_ntoa**(((**struct in\_addr**)**Clien\_addr**.**sin\_addr**)))** **;** // IP of the client

errs **=** 1**;**

**if** **(**mask **!=** **-**1**){** // If there is a mask set

**if** **(**check\_mask**(**aux\_IP**,**IP\_mask**,**mask**)==**0**){** // If we are not in the submask

close **(**newfd**);** // CLose conection

printf**(**"Usuario %s fuera de rango quiso conectarse \n"**,**aux\_IP**);**

errs **=** 0**;**

**}**

**}**

**if** **(**errs **==** 1**){** // If its a valid client

FD\_SET**(**newfd**,** **&**master\_read\_fds**);** // add to master set

**if** **(**newfd **>** fdmax**)** **{** // keep track of the maximum fd

fdmax **=** newfd**;**

**}**

num\_sockets**++;**

num\_client\_sockets**++;**

aux\_port**=** **(**int**)**Clien\_addr**.**sin\_port**;** // Port of the client we are connected to

// printf("\*\*\*\* New client connection from %s on its port %i \*\*\*\* \n",aux\_IP,aux\_port );

strcpy**(**Client\_IPs**[**newfd**],**aux\_IP**)** **;** // IP of the client

**}**

**}**

**}** // If we havent overflowed the number of client sockets connections

**}**

**else** **if** **(**i **==** STDIN**)** **{** // RECEIVED DATA FROM KEYBOARD

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* RECEIVE INFO FROM STDIN \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// We only get here when [ENTER] is pressed

fgets**(**send\_buff**,** **sizeof(**send\_buff**),** stdin**);**

len **=** strlen**(**send\_buff**)** **-** 1**;**

**if** **(**send\_buff**[**len**]** **==** '\n'**){**

send\_buff**[**len**]** **=** '\0'**;**

**}**

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* COMMANDS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// printf("recibida cadena \n");

word\_array **=** get\_word\_array **(**send\_buff**,** **&**num\_words**);**

// printf("Palabra %s \n", word\_array[0]);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* --filtro\_usuario \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**if** **(**strcmp**(**word\_array**[**0**],**"--filtro\_usuario"**)==**0**){**

errs **=** find\_char **(**word\_array**[**1**],** '/'**);**

// printf("Possition %i \n",errs);

word\_array**[**1**][**errs**]** **=** 0**;**

strcpy**(**IP\_mask**,** word\_array**[**1**]);**

mask **=** atoi**(&(**word\_array**[**1**][**errs **+** 1**]));**

printf**(**"Establecida mascara %s/%i \n"**,** IP\_mask**,** mask**);**

**}**

**if** **(**strcmp**(**word\_array**[**0**],**"--max\_con"**)==**0**){**

max\_conn **=** atoi**(**word\_array**[**1**]);**

printf**(**"Establecido numero maximo de conexiones %i \n"**,** max\_conn**);**

**}**

**if** **(**strcmp**(**word\_array**[**0**],**"--filtro\_servidor"**)==**0**){**

strcpy**(**allowed\_server**,** word\_array**[**1**]);**

printf**(**"Establecido servidor %s \n"**,** allowed\_server**);**

**}**

**}**

**else** **{**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* RECEIVE INFO FROM CLIENT OR WEB \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**if** **(**FD\_ISSET**(**i**,** **&**master\_read\_fds**)){**

nbytes **=** recv**(**i**,** recv\_buff**,** **sizeof(**recv\_buff**),** 0**);** // Read data

recv\_buff**[**nbytes**]** **=** 0**;**

**if** **(**nbytes**<=** 0**)** **{** // Connection closed

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* CLIENT OR WEB CLOSED CONNECTION \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**if** **(**nbytes**<** 0**)** **{**

perror**(**"recv"**);**

**}**

close**(**i**);** // bye!

// printf("\*\*\*\*\*Socket %i cerrada, conectado a %i\*\*\*\*\*\n",i,related\_fd[i]);

FD\_CLR**(**i**,** **&**master\_read\_fds**);** // remove from master set

num\_sockets**--;**

related\_fd**[**i**]** **=** **-**1**;**

**if** **(**Client\_IPs**[**i**][**0**]** **==** 0**){** // If its not a client socket

Webs\_IPs**[**i**][**0**]** **=** 0**;**

num\_web\_sockets**--;**

**}else{**

Client\_IPs**[**i**][**0**]** **=** 0**;**

num\_client\_sockets**--;**

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXx

// Since the client closed the conection, we close the sockets associated with webs

**for** **(**j **=** 0**;** j **<** NUM\_USERS**;** j**++){**

**if** **(**related\_fd**[**j**]** **==** i**){** // If the client originated those web sockets we erase them

close**(**j**);** // bye!

FD\_CLR**(**j**,** **&**master\_read\_fds**);** // remove from master set

num\_sockets**--;**

Webs\_IPs**[**j**][**0**]** **=** 0**;**

num\_web\_sockets**--;**

related\_fd**[**j**]** **=** **-**1**;**

// CLOSING SOCKETS THIS WAY ORIGINATES A BIT OF A MISTAKE COZ THE WEB SENDS US STUFF AFTER WE

// HAVE CLOSED THE SOCKET. We solve it with a security check of IFSET

**}**

**}**

**}**

**}**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* GOT GOOD DATA FROM CLIENT OR WEB \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**else** **{**

**if** **(**Client\_IPs**[**i**][**0**]** **!=** 0**){** // IF ITS A CLIENT !!!!!!!!!

printf**(**"Analizando mensaje de Cliente\n"**);**

// printf("\n\*\*\*\*\*\* Bytes received %i \*\*\*\*\*\*\*\*\* \n%s\n", nbytes, recv\_buff); //Print information received

**for** **(**j **=** 0**;** j **<** 8**;** j**++)** **{** // CHECK METHOD REQUESTED

**if** **(**strcmp2**(**recv\_buff**,**methods**[**j**])** **==** 0**){**

strcpy**(**method\_req**,** methods**[**j**]);** // GET METHOD

printf**(**"Metodo %s !!!!! \n"**,**method\_req**);**

**break;**

**}**

**}**

// FOR A EXTENDER VERSION IT SHOULD CHECK THE METHOD AND ACT ACCORDINGLY, HERE WE JUST

// ACK LIKE ITS "GET" METHOD

line\_array **=** get\_line\_array **(**recv\_buff**,** **&**num\_lines**);**

word\_array **=** get\_word\_array **(**line\_array**[**0**],** **&**num\_words**);** // GET THE RESOURCE

// SEND REQUESTED DATA TO BBDD

// Get the date.

fechaActual **=** time**(**0**)** **;** // Get the time and print it

fechaPtr **=** gmtime**(&**fechaActual**)** **;**

numbytes **=** sprintf**(**send\_buff**,** "R%i-%i-%i(%i:%i:%i) "**,** fechaPtr**->**tm\_mday**,** fechaPtr**->**tm\_mon **+** 1**,**

fechaPtr**->**tm\_year **+** 1900**,** fechaPtr**->**tm\_hour**,** fechaPtr**->**tm\_min**,** fechaPtr**->**tm\_sec**);**

// The initial R means its a request

sprintf**(**send\_buff**+**strlen**(**send\_buff**),**"%s -> %s"**,**Client\_IPs**[**i**],**word\_array**[**1**]);**

numbytes **=** UDP\_send\_string**(**UDP\_BBDD\_sd**,** **&**BBDD\_dir\_serv**,** send\_buff**);**

strcpy**(**Client\_Reqs**[**i**],** word\_array**[**1**]);** // Store the requested resource

word\_array **=** get\_word\_array **(**line\_array**[**1**],** **&**num\_words**);** // GETTING HOST NAME

strcpy**(**host\_name**,** word\_array**[**1**]);**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* CHECKIN FOR DIFFERENT PORTS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

errs **=** find\_char**(**host\_name**,**':'**);**

**if** **(**errs **!=-**1**)** **{** // If we have been told a port other than 80

host\_name**[**errs**]** **=** 0**;** // Erase the port part from the name

aux\_port **=** atoi**(&**host\_name**[**errs**+**1**]);** // Get the port

printf **(**"Puerto no standar %i\n"**,** aux\_port**);**

**}**

**else** **{**

aux\_port **=** 80**;**

**}**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* CHECKIN WEBSITE FILTER \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

errs **=** 1**;**

**if** **(**allowed\_server**[**0**]** **!=**0**){** // If there is a host filter

**if** **(**strcmp**(**host\_name**,** allowed\_server**)!=** 0**){** // If it doesnt acomplish the filter

errs **=**0**;**

**}**

**}**

**if(**errs **==** 0**){**

printf**(**"Servidor baneado mafaca !!!!! \n"**);**

**break;** // MIGHT BE BIG MISTAKE !!!!!!!!!!

**}**

web\_IP **=** get\_IP **(**host\_name**);** // GET HOST IP

**if** **(**web\_IP**[**0**]** **==** 0**){** // If it was a bad request

bad\_request **=** 1**;**

printf**(**"IP incorrecta \n"**);**

printf**(**"PETICION INCORRECTA 400 \n\n"**);** //Print information received

**if** **(**sendall**(**i**,**bad\_request\_s**,**strlen**(**bad\_request\_s**),** 0**)** **==** **-**1**)** **{**

perror**(**"sendall"**);**

**}**

**break;**

**}**

// printf("Conectando a %s con IP %s\n", host\_name, web\_IP);

// WE CHECK IF THE USER IS ALREADY CONNECTED TO THAT IP TO GET THE STUFF

errs **=** 1**;**

**for** **(**j **=** 0**;** j **<** NUM\_USERS**;** j**++){**

**if** **(**related\_fd**[**j**]** **==** i**){** // If this is a socket connection

**if** **(**strcmp**(**web\_IP**,** Webs\_IPs**[**j**])** **==** 0**)** **{** // If we are connected to that IP

printf**(**"REPEATED IP, no need for another socket \n"**);**

// STILL HAS THE PROBLEM THAT IF A CLIENT HAS DIFERENT SOCKETS, EACH SOCKET WILL DO A TCP CONNECTION

// TO THE SAME IP BUT THATS HTTP PROBLEM I GUESS. BUT THATS NOT OUR PROBLEM (Concurrent HTTP connection).

errs **=** 0**;**

**break;**

**}**

**}**

**}** // for end

**if** **(**errs **==** 1**){** // If we are not connected to that IP that hold the resource

// IF RESOURSE IS OF TIPE: // http://www.uah.es:8080/resource

// we will have to conect to that specific port... so we check the format.

// COntains Host:-> www.uah.es:8080

newfd **=** connect\_machine **(**web\_IP**,** aux\_port**);** // Connect to host

**if** **(**newfd **==** **-**1**){** // CANNOT CONECT TO SERVER

bad\_request **=** 1**;**

printf**(**"Puerto incorrecto \n"**);**

printf**(**"PETICION INCORRECTA 400 \n\n"**);** //Print information received

**if** **(**sendall**(**i**,**bad\_request\_s**,**strlen**(**bad\_request\_s**),** 0**)** **==** **-**1**)** **{**

perror**(**"sendall"**);**

**}**

**break;**

**}**

related\_fd**[**i**]** **=** newfd**;** // Stablish relationships

related\_fd**[**newfd**]** **=** i**;**

// printf("\*\*\*\*\*ORIGEN %i , DESTINO CREADO %i\*\*\*\*\*\n",i,related\_fd[i]);

FD\_SET**(**newfd**,** **&**master\_read\_fds**);** // add to master set

**if** **(**newfd **>** fdmax**)** **{** // keep track of the maximum fd

fdmax **=** newfd**;**

**}**

num\_sockets**++;**

num\_web\_sockets**++;**

strcpy**(**Webs\_IPs**[**newfd**],** web\_IP**);**

**}** // Conect to the host if

**}** // If its a client

**else** **{** // IF ITS A SERVER!!!!!!!!!

printf**(**"Analizando mensaje de Servidor \n"**);**

**if** **(**strcmp2**(**recv\_buff**,**"HTTP/"**)** **==** 0**){** // If its the begining of a response message, not part of it

// This checking might fail if the content has the words HTTP/ ->

line\_array **=** get\_line\_array **(**recv\_buff**,** **&**num\_lines**);**

word\_array **=** get\_word\_array **(**line\_array**[**0**],** **&**num\_words**);** // GET THE ANSWERE

HTTP\_response **=** atoi **(**word\_array**[**1**]);**

printf **(**"HTTP response %i %s \n"**,** HTTP\_response**,** word\_array**[**2**]);**

// SEND ANSWERE DATA TO BBDD

// Get the date.

fechaActual **=** time**(**0**)** **;** // Get the time and print it

fechaPtr **=** gmtime**(&**fechaActual**)** **;**

numbytes **=** sprintf**(**send\_buff**,** "A%i-%i-%i(%i:%i:%i) "**,** fechaPtr**->**tm\_mday**,** fechaPtr**->**tm\_mon **+** 1**,**

fechaPtr**->**tm\_year **+** 1900**,** fechaPtr**->**tm\_hour**,** fechaPtr**->**tm\_min**,** fechaPtr**->**tm\_sec**);**

// The initial R means its a request

sprintf**(**send\_buff**+**strlen**(**send\_buff**),**"%s -> %s -> %i"**,**Client\_IPs**[**related\_fd**[**i**]],**Client\_Reqs**[**related\_fd**[**i**]],**HTTP\_response**);**

numbytes **=** UDP\_send\_string**(**UDP\_BBDD\_sd**,** **&**BBDD\_dir\_serv**,** send\_buff**);**

**}**

**}**

// We forward the mensaje

// printf("\*\*\*\*\* Recibido desde %i hacia %i \*\*\*\*\*\n",i,related\_fd[i]);

**if** **(**bad\_request **==** 0**){**

**if** **(**FD\_ISSET**(**related\_fd**[**i**],** **&**master\_read\_fds**)){** // So that we dont send the 200 OK to a closed socket (for some reason)

// Sometimes the client closes conection before the server gives the final answere so

// we put this line to solve it.

**if** **(**sendall**(**related\_fd**[**i**],**recv\_buff**,**nbytes**,** 0**)** **==** **-**1**)** **{**

// IT was an error to write strlen(recv\_buff) coz binaries can have 0's

printf**(**"\n\*\*\*\*\*\* Bytes received %i \*\*\*\*\*\*\*\*\* \n %s \n"**,** nbytes**,** recv\_buff**);** //Print information received

perror**(**"sendall"**);**

**}**

**}**

**}**

**}**

**}** // SEGUTIRY CHECK OF IFFDSET

**}**

**}** // If its the origin of the event

**}** // For every possible socket

**}** // Else del timeout.

**}** // While(1)

**return** 0**;**

close**(**descSocket**);**

**}**

# SUITE DE Funciones

#include "sockets\_cab.h"

int listen\_TCP\_Port **(**int Port**,** int max\_conn**){**

struct sockaddr\_in Server\_addr **;** // Structure with the data about socket

int listenSocket **;** // Socket descriptor

int errores**;**

// Open the socket and checks it went well

listenSocket **=** socket**(**AF\_INET**,** SOCK\_STREAM**,** 0**);** // Socket IP, TCP

**if** **(**listenSocket **<** 0**)** **{**

fprintf**(**stderr**,** "SERVIDOR: no se ha podido abrir el socket \n"**)** **;**

exit**(-**1**)** **;**

**}**

clear\_alreadyinuse**(**listenSocket**);**

// Initialize the server addr structure

bzero **((**char**\*)** **&**Server\_addr**,** **sizeof(**Server\_addr**));**

Server\_addr**.**sin\_family **=** AF\_INET**;**

Server\_addr**.**sin\_addr**.**s\_addr**=** htonl**(**INADDR\_ANY**);**

Server\_addr**.**sin\_port **=** htons**(**Port**);**

// Bind address to machine

errores **=** bind**(**listenSocket**,(**struct sockaddr **\*)** **&**Server\_addr**,sizeof(**Server\_addr**));**

**if** **(**errores **<** 0**)** **{**

fprintf**(**stderr**,** "Error al vincular la direccion local\n"**)** **;**

**return** **-**1**;**

**}**

// Listen to incoming TCP conenctions (5 max in the queue)

errores **=** listen**(**listenSocket**,** max\_conn**);**

//printf("Somos %s escuchando en el puerto %i \n",

// inet\_ntoa(((struct in\_addr)Server\_addr.sin\_addr)) ,Port);

printf**(**"Escuchando en el puerto %i \n" **,**Port**);**

**if** **(**errores **<** 0**)** **{**

fprintf**(**stderr**,** "Error al ponerse a escuchar\n"**)** **;**

**return** **-**1 **;**

**}**

**return** listenSocket**;**

**}**

int connect\_machine **(**char **\***IP**,** int Port**){**

struct sockaddr\_in machine\_addr **;** // Structure with the data about socket

int descSocket **;** // Socket descriptor

int errores**;**

// \*\*\*\*\* Open a socket and checks it went well \*\*\*\*\*

descSocket **=** socket**(**AF\_INET**,** SOCK\_STREAM**,** 0**);** // Socket IP, TCP

**if** **(**descSocket**==** **-**1**)** **{**

fprintf**(**stderr**,** "No se ha podido abrir el socket \n"**)** **;**

**return** **-**1 **;**

**}**

fprintf**(**stderr**,** "Socket abierto \n"**)** **;**

// Initialize the Server addr structure to connect server

bzero**((**char **\*)** **&**machine\_addr**,** **sizeof(**machine\_addr**));** // Set everything to 0

machine\_addr**.**sin\_family **=** PF\_INET**;** //

machine\_addr**.**sin\_addr**.**s\_addr**=** inet\_addr**(**IP**);** // IP of the machine we will connect with

machine\_addr**.**sin\_port **=** htons**(**Port**);** // Port of the machine we will connect to

// Connect to the machine

errores **=** connect**(**descSocket**,(**struct sockaddr **\*)** **&**machine\_addr**,** **sizeof(**machine\_addr**));**

**if** **(** errores **==-**1**)** **{**

fprintf**(**stderr**,** "No se ha podido conectar con servidor\n"**)** **;**

**return** **-**1**;**

**}**

printf**(**"Conexion establecida con %s en su puerto %i \n"**,** IP**,** Port**)** **;**

**return** descSocket**;**

**}**

int sendall**(**int socket\_fd**,** char **\***buf**,** int len**,** int FLAGS**)** **{**

int bytes\_sent **=** 0**;** // how many bytes we’ve sent

int n **=** 0**;**

**while(**bytes\_sent **<** len**)** **{**

n **=** send**(**socket\_fd**,** buf **+** bytes\_sent**,** len **-** bytes\_sent**,** FLAGS**);**

**if** **(**n **==** **-**1**)** **{**

**break;**

**}**

bytes\_sent **+=** n**;**

**}**

**if** **(**n **!=** **-**1**)** **{**

**return** bytes\_sent**;**

**}else** **{**

**return** **-**1**;**

**}**

**}**

char **\*** get\_IP **(**char **\***host\_name**)** **{**

char **\***host\_IP **=** **(**char **\*)** malloc **(**50**\*sizeof(**char**));**

struct hostent **\***he**;** // Holds the addr of Server in case we are given an URL

**if** **((**host\_name**[**0**]** **<** '0'**)||(**host\_name**[**0**]** **>** '9'**)){** // If we are given a name and not IP

**if** **((**he **=** gethostbyname**(**host\_name**))** **==** **NULL)** **{**

herror**(**"gethostbyname"**);**

host\_IP**[**0**]** **=** 0**;**

**return** host\_IP**;**

**}**

host\_IP **=** inet\_ntoa**(\*((**struct in\_addr **\*)**he**->**h\_addr**));**

**}**

**else** **{**

strcpy**(**host\_IP**,** host\_name**);**

**}**

**return** host\_IP**;**

**}**

void conversor\_IaS**(**int numero**,** char **\*** cadnum**){**

int i **=** 0**,** j **=** 0**;**

char aux**[**10**];**

char convertir **=** 1**;**

**while(**convertir**){**

aux**[**i**]** **=** numero **%** 10 **+** 48**;**

numero **=** numero **/** 10**;**

i**++;**

**if** **(**numero **==** 0 **){**

convertir **=** 0**;** **}}**

**for** **(**j**=**0**;** j**<**i **;**j**++){**

**\*(**cadnum**+**j**)** **=** aux**[**i**-**j**-**1**];**

**}**

**\*(**cadnum**+**i**)** **=** 0**;**

**}**

char **\*\*** get\_word\_array **(**char **\***cadena**,** int **\***num\_palabras**){**

char **\*\*** words\_list**;**

int i **=** 0**,** num\_p **=** 0**;**

char **\***p**;**

p **=** cadena**;**

words\_list **=** **(**char **\*\*)** malloc **(sizeof(**char**\*)\***500**);**

**for** **(**i **=** 0**;** i **<** 500**;** i**++){**

words\_list**[**i**]** **=** **(**char **\*)** malloc **(sizeof(**char**)\***500**);**

**}**

i **=** 0**;**

**while** **(\***p **!=** 0**)** **{**

**if** **(\***p **==** ' '**){**

**if** **(**i **!=** 0**){** // To avoid multiple ' ' in a row error

words\_list**[**num\_p**][**i**]** **=** 0**;**

i **=** 0**;**

num\_p**++;**

**}**

**}** **else** **{**

words\_list**[**num\_p**][**i**]** **=\***p **;**

i**++;**

**}**

p**++;**

**}**

words\_list**[**num\_p**][**i**]** **=** 0**;**

num\_p**++;**

**\***num\_palabras **=** num\_p**;**

**return** words\_list**;**

**}**

char **\*\*** get\_line\_array **(**char **\***cadena**,** int **\***num\_lines**){**

char **\*\*** line\_list**;**

int i **=** 0**,** num\_p **=** 0**;**

char **\***p**;**

p **=** cadena**;**

line\_list **=** **(**char **\*\*)** malloc **(sizeof(**char**\*)\***50**);**

**for** **(**i **=** 0**;** i **<** 50**;** i**++){**

line\_list**[**i**]** **=** **(**char **\*)** malloc **(sizeof(**char**)\***500**);**

**}**

i **=** 0**;**

**while** **(\***p **!=** 0**)** **{**

**if** **(\***p **==** '\r'**){** // DETECT NEW LINE '\r\n'

p**++;**

**if** **(\***p **==** '\n'**){**

**if** **(**i **!=** 0**){** // To avoid multiple ' ' in a row error

line\_list**[**num\_p**][**i**]** **=** 0**;**

i **=** 0**;**

num\_p**++;**

**if** **(\*(**p**+**1**)** **==** '\r'**){** // If we have to saltos de carro in a row we finish this

**break;**

**}**

**}**

**}**

**}**

**else** **{**

line\_list**[**num\_p**][**i**]** **=\***p **;**

i**++;**

**}**

p**++;**

**}**

line\_list**[**num\_p**][**i**]** **=** 0**;**

num\_p**++;**

**\***num\_lines **=** num\_p**;**

**return** line\_list**;**

**}**

void clear\_alreadyinuse**(**int listener**){**

int yes**=**1**;**

**if** **(**setsockopt**(**listener**,**SOL\_SOCKET**,**SO\_REUSEADDR**,&**yes**,sizeof(**int**))** **==** **-**1**)** **{**

perror**(**"setsockopt"**);**

exit**(**1**);**

**}**

**}**

int listen\_UDP\_Port **(**int Port**){**

int sockfd**;** // descriptor de socket

struct sockaddr\_in dir\_serv**;** // estructura que almacena la direccion IP y numero de puerto del servidor

/\* Llamada a la funcion de creacion del socket \*/

**if** **((**sockfd **=** socket**(**AF\_INET**,** SOCK\_DGRAM**,** 0**))** **==** **-**1**)** **{**

perror**(**"socket"**);**

exit**(**1**);**

**}**

/\* Rellenamos la estructura del servidor \*/

dir\_serv**.**sin\_family **=** AF\_INET**;**

dir\_serv**.**sin\_port **=** htons**(**Port**);**

dir\_serv**.**sin\_addr**.**s\_addr **=** INADDR\_ANY**;** // escuchamos en todas las IPs

bzero**(&(**dir\_serv**.**sin\_zero**),** 8**);** //ponemos a 0 el resto de parametros de la estructura

/\* Llamamos a la funcion bind() \*/

printf**(**"Socket creado\n"**);**

**if** **(**bind**(**sockfd**,** **(**struct sockaddr **\*)&**dir\_serv**,** **sizeof(**struct sockaddr**))** **==** **-**1**)** **{**

perror**(**"bind"**);**

exit**(**1**);**

**}**

**return** sockfd**;**

**}**

int socket\_UDP **(**char **\***IP**,** int Port**,** struct sockaddr\_in **\*** dir\_serv**){** // estructura que almacena la direccion IP y numero de puerto del servidor

int sockfd**;** //descriptor de socket

/\* Llamada a la funcion de creacion del socket \*/

**if** **((**sockfd **=** socket**(**AF\_INET**,** SOCK\_DGRAM**,** 0**))** **==** **-**1**)** **{**

perror**(**"socket"**);**

exit**(**1**);**

**}**

/\* Rellenamos la estructura del servidor \*/

dir\_serv**->**sin\_family **=** AF\_INET**;**

dir\_serv**->**sin\_port **=**htons**(** Port**);**

dir\_serv**->**sin\_addr**.**s\_addr **=** inet\_addr**(**IP**);**//introducimos la IP desde gethostname

bzero**(&(**dir\_serv**->**sin\_zero**),** 8**);**//ponemos a 0 el resto de parametros de la estructura

**return** sockfd**;**

**}**

int UDP\_send\_string**(**int sd**,** struct sockaddr **\*** dir\_serv**,** char **\***cad**){**

int aux**;**

aux **=** sendto**(**sd**,**cad**,**strlen**(**cad**),**0**,**dir\_serv**,** **sizeof(**struct sockaddr**));**

**if** **(**aux **==** **-**1**){**

perror**(**"sendto"**);**

**}**

**return** aux**;**

**}**

// Checks wheater the IP is in the mask

int check\_mask**(**char **\***IP**,** char **\***IP\_mask**,** unsigned int mask**){**

unsigned int IPx**,** IPm**;**

IPx **=** get\_IP\_num **(**IP**)** **>>** **(**32**-**mask**);**

IPm **=** get\_IP\_num **(**IP\_mask**)** **>>** **(**32**-**mask**);**

**if** **(**IPx **==** IPm**){**

**return** 1**;**

**}**

**else** **{**

**return** 0**;**

**}**

**}**

unsigned int get\_IP\_num **(**char **\***IP**){**

unsigned int IP\_num **=** 0**;**

char **\***p**,** position **=** 0**;**

char aux\_num**[**5**];**

int i **=** 0**;**

p **=** IP**;**

**while** **(\***p **!=** 0**)** **{**

**if** **(\***p **==** '.'**){**

aux\_num**[**i**]** **=** 0**;**

IP\_num **+=** atoi**(**aux\_num**)** **<<** 8**\*(**3**-**position**);**

i **=** 0**;**

position**--;**

**}** **else** **{**

aux\_num**[**i**]** **=\***p **;**

i**++;**

**}**

p**++;**

**}**

**return** IP\_num**;**

**}**

int find\_char **(**char **\***cad**,** char car**){**

char **\***p **=** cad**;**

int i **=** 0**;**

**while** **(\***p**){**

**if** **(\***p **==** car**){**

**return** i**;**

**}**

i**++;**

p**++;**

**}**

**return** **-**1**;**

**}**

int strcmp2**(**char **\***s1**,** char **\***s2**){**

char **\*** p1**,** **\*** p2**;**

p1 **=** s1**;**

p2 **=** s2**;**

**while** **((\***p1**)&&(\***p2**)){**

**if** **(\***p1 **!=** **\***p2**){**

**return** 1**;**

**}**

p1**++;**

p2**++;**

**}**

**return** 0**;**

**}**

# Archivo de cabeceras.

#ifndef \_\_DEFS\_H\_\_

#define \_\_DEFS\_H\_\_

#include <sys/socket.h>

#include <sys/types.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <time.h>

#include <errno.h>

#include <netdb.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#endif

#define NUM\_USERS 1024

#define STDIN 0

int connect\_machine **(**char **\***IP**,** int Port**);**

int listen\_TCP\_Port **(**int Port**,** int max\_conn**);**

int sendall**(**int socket\_fd**,** char **\***buf**,** int len**,** int FLAGS**);**

void conversor\_IaS**(**int numero**,** char **\*** cadnum**);**

char **\*** get\_IP **(**char **\***host\_name**);**

void clear\_alreadyinuse**(**int listener**);**

char **\*\*** get\_word\_array **(**char **\***cadena**,** int **\***num\_palabras**);**

char **\*\*** get\_line\_array **(**char **\***cadena**,** int **\***num\_lines**);**

int listen\_UDP\_Port **(**int Port**);**

int socket\_UDP **(**char **\***IP**,** int Port**,** struct sockaddr\_in **\*** dir\_serv**);**

UDP\_send\_string**(**int sd**,** struct sockaddr **\*** dir\_serv**,** char **\***cad**);**

int check\_mask**(**char **\***IP**,** char **\***IP\_mask**,** unsigned int mask**);**

unsigned int get\_IP\_num **(**char **\***IP**);**

int find\_char **(**char **\***cad**,** char car**);**

int strcmp2**(**char **\***s1**,** char **\***s2**);**

# BIBLIOGRAFIA

[1]. Beej’s Guide to Network Programming Using Internet Sockets

[2]. HTTP Persistence & Web Caching *Jasleen Kaur*February 12, 2013