Universidad de Alcalá

Departamento de Automática

ARQUITECTURA DE REDES
Laboratorio

Práctica 4: Ejercicios de aplicación de HTTP

Grado en Ingeniería Informática
Curso 2022/23

1. OBJETIVOS.

El objetivo de esta práctica es que el alumno llegue a conocer los principales conceptos relacionados con la comunicación de procesos mediante Sockets utilizando protocolos *http*.

2. ACTIVIDADES.

- El alumno deberá estudiar los ejercicios cuyo código se adjunta.
- Escribirá los programas en lenguaje C que sean requeridos, y que nos permitirán responder a peticiones de un navegador Web.
- Deberá compilar y depurar los ejercicios propuestos utilizando lenguaje C sobre el S.O Linux.
- Deberá estudiar los resultados obtenidos con las capturas de tráfico de red obtenidas con la aplicación Wireshark.

3. EJERCICIOS.

Ejercicio 1: Crear un miniservidor Web.

Tiene como objeto el estudio y desarrollo de un miniservidor Web. Una vez que hayamos escrito y desarrollado nuestro miniservidor, seremos capaces de "entrar" utilizando cualquier navegador Web, desde nuestra propia máquina (en donde estará corriendo el servidor) o desde otra máquina cualquiera. También podemos modificar su comportamiento y ver el resultado en el navegador Web.

El proceso de la práctica tendrá cuatro pasos básicos:

1. Edición y estudio de una serie de ficheros fuentes del programa servidor Socket, y ficheros auxiliares en lenguaje HTML.
2. Compilación del programa Socket sin errores.
3. Selección de un puerto libre y ejecución del miniservidor.
4. Comprobación del funcionamiento desde el propio sistema y si es posible también desde otros.

1. Edición y estudio de los ficheros fuentes:

A continuación se muestran los distintos ficheros que integran el ejercicio:

- Servidor**: *mserver.c***

```
# include <sys/types.h>
# include <sys/socket.h>
# include <sys/wait.h>
# include <netinet/in.h>
# include <netdb.h>
# include <errno.h>
# include <stdio.h>
# include <unistd.h>
# include <fcntl.h>
# include <signal.h>
# include <string.h>
# include <stdlib.h>
```

```
# define FICHERO  "mserver.log"
# define HEAD     "head.html"
# define TAIL     "tail.html"

static void usage(char *name)
{
   fprintf(stderr, "%s: Uso %s <port>\n", name, name);
   exit (1);
}

static void ChildHasDied(int num)
{
int   status;
pid_t pid;

   pid = wait(&status);
   if (signal(SIGCHLD, ChildHasDied) == SIG_ERR)
      fprintf(stderr, "Error al dar de alta la accion.\n");
   return;
}

static void FinishAction(int num)
{
   fprintf(stderr, "Programa finalizado correctamente.\n\n");
   exit(0);
}

static void net_server(int sock)
{
# define BUFF_LEN 8192
int   len;
char  formato[BUFF_LEN];
char  cadena[BUFF_LEN];
FILE *fp = NULL;

   fp = fopen(FICHERO, "w");
   len = fcntl(sock, F_SETFD, O_NDELAY | O_NONBLOCK);

   do
   {
      len = read(sock, cadena, BUFF_LEN);
      sprintf(formato, "%%%d.%ds", len, len);
      if (fp)
        {
         fprintf(fp, formato, cadena);
         printf(formato, cadena);
        }
      else
         printf(formato, cadena);
    }while (len == BUFF_LEN);
   fclose(fp);

   fp = fopen(HEAD, "r");
   if (fp)
     {
      while(fgets(cadena, sizeof(cadena), fp) != NULL)
         write(sock, cadena, strlen(cadena));
      }
   fclose(fp);
```

```
   fp = fopen(FICHERO, "r");
   if (fp)
     {
      while(fgets(cadena, sizeof(cadena), fp) != NULL)
         write(sock, cadena, strlen(cadena));
     }
   fclose(fp);

   fp = fopen(TAIL, "r");
   if (fp)
     {
      while(fgets(cadena, sizeof(cadena), fp) != NULL)
         write(sock, cadena, strlen(cadena));
     }
   fclose(fp);

   close(sock);
}

static int net_slisten(short port)
{
int sock, new_sock;
struct sockaddr_in sin, from;
unsigned int len = sizeof(from);
int OptVal = 1;
int OptSize = sizeof(OptVal);

   bzero((char *) &sin, sizeof (sin));
   sin.sin_family = AF_INET;
   sin.sin_port = htons(port);

   sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
   if (sock < 0)
      exit(-1);

   if (setsockopt(sock, SOL_SOCKET, SO_REUSEADDR,
      (char *)&OptVal, OptSize) < 0) exit(-3);

   if (bind(sock, (struct sockaddr *)&sin, sizeof(sin)) < 0) exit(-2);

   listen (sock, 4);
   for (;;)
   {
      len = sizeof(from);
      new_sock = accept(sock, (struct sockaddr *)&from, &len);
      if (new_sock < 0)
      {
         if (errno == EINTR)
            continue;
         exit(-4);
      }

      switch (fork())
      {
         case -1:           /* ¿error? */
            exit(-5);

         case 0:            /* proceso hijo */
            close(sock);
            return(new_sock);
```

```
         default:           /* proceso padre */
            close(new_sock);
      }
   }
}

int main(int argc, char **argv)
{
int sock;
short port;

   if (signal(SIGTERM, FinishAction) == SIG_ERR)
      fprintf(stderr, "Error al dar de alta la accion.\n");

   if (signal(SIGCHLD, ChildHasDied) == SIG_ERR)
      fprintf(stderr, "Error al dar de alta la accion.\n");

   if (argc != 2) usage(argv[0]);
   port = atoi(argv[1]);
   sock = net_slisten(port);

   if (sock == -1)
   {
     perror("mserver_http: net_slisten falló.");
     exit(1);
   }
   net_server(sock);
   return 0;
}
```

- Archivo de cabecera: ***"head.html"***

```
HTTP/1.1 200 OK
Content-Type: text/html; charset="ISO-8859-15"

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

<html>
 <head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>
Peticion del navegador
</title>
<style type="text/css">
 pre { font-size: 12pt; font-weight: bold; background-color: #ffefcf;
}
</style>
</head>
<center>
<h1>Peticion del navegador</h1>
 </center>
<pre class="pre">
```

- Archivo de cola: ***"tail.html"***

```
</pre>
</html>
```

- Archivo de compilación: ***"Makefile.m"***

```
mserver_http.lx: mserver.c
<tabulador>gcc -g -ansi -pedantic $~ -o $@
```

2. Compilación del programa Socket sin errores:

Para ello, abriremos un terminal, nos colocaremos en la carpeta elegida para destino de los anteriores ficheros y teclearemos simplemente:

```
make -f Makefile.m
```

3. Selección de un puerto libre y ejecución del miniservidor.

Necesitaremos elegir un puerto TCP que esté libre. Los números que suelen estar libres comienzan aproximadamente en 2000. Para comprobar que efectivamente un determinado puerto está libre, por ejemplo 2500, podemos usar el siguiente comando:

```
netstat -an | grep 2500
```

Si el anterior comando no contesta nada, es que efectivamente el puerto está libre y lo podemos usar. Suponiendo que el puerto 2500 está libre, arrancaremos entonces nuestro miniservidor con el siguiente comando:

```
./mserver_http.lx 2500
```

4. Comprobación del funcionamiento desde el propio sistema (y si es posible también desde otros).

4.1 Conexión del navegador Web con el servidor:

a) Probaremos entonces que nuestro navegador se conecta al servidor. Para ello, del menú de *Aplicaciones*, podemos seleccionar el *FireFox* y arrancarlo. Observaremos la respuesta que llega al navegador. En la barra de direcciones URL podemos escribir lo siguiente:

http://localhost:2500/ o también http://127.0.0.1:2500/

b) Podemos también conectar nuestro navegador al servidor de cualquier otro PC del laboratorio con tal que lo tenga arrancado. Necesitaremos dos datos: la dirección IP de ese PC y el número de puerto TCP que el usuario haya reservado para el miniservidor. Suponiendo que el PC tiene como dirección IP 172.29.40.40 y que el puerto es el 7700, escribiremos en la barra de direcciones lo siguiente:

http://172.29.40.40:7700/

4.2. Respuesta del miniservidor Web:

Como podemos observar, el miniservidor nos devuelve como respuesta justamente los mismos comandos que el navegador Web le envió a él. Ese es el comportamiento de que hemos dotado a nuestro miniservidor.

La primera línea que leemos se refiere al comando GET del protocolo HTTP. Las siguientes líneas constituyen pares (parámetro, valor) con las que el navegador se identifica a sí mismo y comunica al servidor Web ciertos datos acerca de sí mismo: por ejemplo, qué codificaciones de caracteres acepta, qué tipos de datos (son los MIME types, relacionados con la capa de presentación), etc.

***Práctica 1*: Generación de un eco parcial de respuesta al navegador.**
Modificar el programa Socket que funciona como un miniservidor Web, para que nos muestre en nuestra ventana terminal desde la que estamos ejecutando el servidor todos los comandos que el navegador Web envía, y devuelva a este únicamente:

- El nombre del comando enviado por el navegador.
- El nombre y versión del navegador.
- El nombre o dirección IP del servidor y el puerto del servidor.

Ejercicio 2: Envío de información de formularios con formato application/x-www-form-urlencoded al miniservidor Web.

El método POST es utilizado por los formularios para enviar información al servidor. Como ejemplo (que podemos modificar a nuestro gusto) podemos utilizar el siguiente fichero HTML:

- Archivo de formulario: ***"post-url.html"***

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
 <title>
  Ejemplos de formularios
 </title>
 <style type="text/css">
  p { font-size: 12pt; font-weight: bold; background-color: #ffefcf; }
</style>
</head>
<center>
<h1>Ejemplos de formularios</h1>
</center>
<FORM action="http://localhost:2500" method="post"
      enctype="application/x-www-form-urlencoded">
      <P>
      Nombre: <INPUT type="text" name="nombre"><BR>
      Apellidos: <INPUT type="text" name="apellidos"><BR>
      Correo electrónico: <INPUT type="text" name="email"><BR>
```

```
      Contraseña: <INPUT type="password" name="passwd"><BR>
      <INPUT type="radio" name="sexo" value="V"> Hombre<BR>
      <INPUT type="radio" name="sexo" value="M"> Mujer<BR>
      <INPUT type="checkbox" name="verde"> Verde<BR>
      <INPUT type="checkbox" name="rojo"> Rojo<BR>
      <INPUT type="checkbox" name="azul"> Azul<BR>
      <INPUT type="checkbox" name="amarillo"> Amarillo<BR>
      <INPUT type="submit" value="Dar de alta">
      <INPUT type="reset">
      </P>
</FORM>
</html>
```

El formato de creación de este formulario atiende al método *application/x-www-form-urlencoded.* Podemos encontrar información sobre el formato de envió del navegador hacia el servidor, consultándolo en esta página
http://www.w3.org/TR/html4/interact/forms.html#h-17.13.4.1

Para su ejecución volvemos a seguir los siguientes pasos:

a) Selección de un puerto libre y ejecución del miniservidor:
Necesitaremos elegir un puerto TCP que esté libre. Los números que suelen estar libres comienzan aproximadamente en 2000. Para comprobar que efectivamente un determinado puerto está libre, por ejemplo 2500, podemos usar el siguiente comando:

```
netstat -an | grep 2500
```

Si el anterior comando no contesta nada, es que efectivamente el puerto está libre y lo podemos usar. Suponiendo que el puerto 2500 está libre, arrancaremos entonces nuestro miniservidor con el siguiente comando:

```
./mserver_http.lx 2500
```

b) Conexión del navegador Web con el servidor:
Abriremos el fichero ***post-url.html*** mediante la opción de "*Abrir fichero*" en nuestro navegador, rellenaremos los campos y, al pulsar el botón "*Dar de alta*", observaremos los resultados.

Ejercicio 3: Envío de información de formularios con formato multipart/form-data al miniservidor Web.

El método es similar al ejercicio anterior. Para ello en esta ocasión utilizaremos el siguiente fichero en lenguaje HTML.

- Archivo de formulario: ***"post-multi.html"***

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>
```

```
  Ejemplos de formularios
</title>
<style type="text/css">
  p { font-size: 12pt; font-weight: bold; background-color: #ffefcf; }
</style>
</head>
<center>
<h1>Ejemplos de formularios</h1>
</center>
<form action="http://localhost:2500" method="post"
enctype="multipart/form-data">
      <p>
      Nombre: <input type="text" name="nombre"><br><br>
      Apellidos: <input type="text" name="apellidos"><br><br>
      Correo electrónico: <input type="text" name="email"><br><br>
      Contraseña: <input type="password" name="passwd"><br><br>
      <input type="radio" name="sexo" value="V"> Hombre<br><br>
      <input type="radio" name="sexo" value="M"> Mujer<br><br>
      <input type="checkbox" name="verde"> Verde<br><br>
      <input type="checkbox" name="rojo"> Rojo<br><br>
      <input type="checkbox" name="azul"> Azul<br><br>
      <input type="checkbox" name="amarillo"> Amarillo<br><br>
      <textarea name="eltexto" rows="5" cols="80">
Escribe algo aquí, si quieres.
      </textarea><br><br>
      Elijo algunas frutas:
      <select multiple size="4" name="seleccion-frutas">
      <option selected value="platano">Plátano</option>
      <option selected value="naranja">Naranja</option>
      <option value="pera">Pera</option>
      <option value="limon">Limón</option>
      <option value="fresa">Fresa</option>
      <option value="manzana">Manzana</option>
      <option value="ciruela">Ciruela</option>
      </select><br><br>
      Mando este fichero: <input type="file" name="fichero"><br><br>
      <input type="submit" value="Dar de alta">
      <input type="reset">
      </p>
</form>
</html>
```

El formato de creación de este formulario atiende al método *multipart/form-data*. Podemos encontrar información sobre el formato de envió del navegador hacia el servidor, consultándolo en esta página
http://www.w3.org/TR/html4/interact/forms.html#h-17.13.4.2

Los métodos utilizados para compilar y ejecutar tanto el miniservidor como el la apertura del fichero ***post-multi.html*** son similares a los del ejercicio anterior.

***Práctica 2*: Uso de Wireshark para estudiar el protocolo HTTP**

A continuación como veremos, podemos utilizar *Wireshark* para estudiar el intercambio de mensajes que se obtiene a la hora de realizar una consulta http a una página Web sencilla.

1) En primer lugar abrimos un navegador Web que esté instalado en la máquina.
2) Abrimos *Wireshark* y seleccionamos la interfaz de red adecuada. Se muestra todo el tráfico de red cursado por el PC.
3) Le indicamos que únicamente nos filtre los mensajes correspondientes al protocolo http. Escribimos "http" sin comillas en el campo correspondiente al texto y pulsamos el botón "*Apply*" para que comience a filtrar. En este momento únicamente filtrará mensajes pertenecientes al protocolo http.
4) En el navegador, accedemos a la siguiente dirección de red: ***http://localhost:2500/*** .El navegador deberá mostrar una página Web con algún texto.
5) Paramos la captura de *Wireshark*:
   La cabecera del mensaje http contiene alguna información que puede ayudar al navegador a interpretar el mensaje encapsulado. Intentad averiguar lo siguiente:
   a) ¿Qué versión de http emplea tu navegador?. ¿Qué versión de http ejecuta el servidor?.
   b) ¿Qué idiomas indica tu navegador al servidor que está dispuesto aceptar en la respuesta?. ¿Qué relación tiene esta información con el lenguaje que empleas en tu navegador y sistema operativo?.
   c) ¿Cuál es la dirección IP de tu ordenador?. ¿Y del servidor Web al que estás accediendo?.
   d) ¿Cuál es el código de estado devuelto a tu navegador por el servidor?. ¿Cuál es el significado de ese código de estado?.
   e) ¿Cuándo fue modificado por última vez el fichero html que el servidor te está devolviendo?.
   f) ¿Cuántos bytes de contenido está devolviendo el servidor al navegador?. ¿Coincide con el tamaño de la trama?. Razona tu respuesta.

6) Repetimos los pasos 4 y 5 con la apertura por parte del navegador de los ficheros ***post-url.html*** **y** ***post-multi.html***