Imagen que contiene Icono

El contenido generado por IA puede ser incorrecto.Dibujo en blanco y negro

El contenido generado por IA puede ser incorrecto.

Instituto Politécnico Nacional

Escuela Superior de Cómputo

Ingeniería en Sistemas Computacionales

Aplicaciones para comunicaciones en red

Academia “Sistemas Distribuidos”

Plan 2020

Práctica 15:

“**Ejemplo de Protocolo HTTP**”

2015090269

González González Armando Omar

Profesor: Ojeda Santillan Rodrigo

Contenido

[Objetivo 3](#_Toc201742584)

[Introducción 3](#_Toc201742585)

[Desarrollo 4](#_Toc201742586)

[3.1 Análisis del Código del Servidor (http.c) 4](#_Toc201742587)

[Conclusión 8](#_Toc201742588)

[Pregunta 8](#_Toc201742589)

[¿Cómo podrían generar un negocio a través de lo visto en la práctica? 8](#_Toc201742590)

[Bibliografía 8](#_Toc201742591)

## Objetivo

Ilustrar cómo se puede implementar un servidor HTTP básico utilizando el lenguaje de programación C. Este ejercicio busca demostrar el funcionamiento fundamental del protocolo HTTP mediante la creación de un servidor que escucha en el puerto 80, acepta conexiones de clientes (navegadores web) y les responde con una página HTML simple.

## Introducción

El Protocolo de Transferencia de Hipertexto (HTTP) es un protocolo de la capa de aplicación que establece las reglas y el formato para la comunicación entre clientes web, como los navegadores, y los servidores web. Es la base sobre la que se construye la World Wide Web. HTTP es un protocolo orientado a la conexión, por lo que depende de un protocolo de transporte fiable como TCP para asegurar la entrega íntegra de los datos.

En esta práctica, se construye un servidor simple en C que demuestra este flujo. El servidor se configura para escuchar conexiones entrantes en el puerto estándar de HTTP (puerto 80). Al recibir una solicitud de un cliente, el servidor no analiza la petición, sino que responde directamente con una página HTML predefinida. Esta respuesta está formateada según el estándar HTTP, incluyendo una línea de estado, cabeceras y el cuerpo del mensaje (el código HTML). Este enfoque permite comprender a bajo nivel cómo se transmiten los datos estructurados en la web.

## Desarrollo

La implementación de la práctica consiste en un único programa en lenguaje C que funciona como un servidor web minimalista. Este servidor se enlaza al puerto 80, espera conexiones y sirve una página estática a cualquier cliente que se conecte.

### **3.1 Análisis del Código del Servidor (**http.c**)**

El código del servidor realiza los siguientes pasos clave para su funcionamiento:

1. **Inclusión de Bibliotecas:** Se incluyen las cabeceras necesarias para la programación de sockets en un entorno tipo UNIX (sys/socket.h, netinet/in.h, unistd.h) y para la manipulación de cadenas (string.h).
2. **Creación del Socket:** Se crea un socket TCP (SOCK\_STREAM) para el protocolo de internet IPv4 (AF\_INET) mediante la llamada a socket().
3. **Configuración y Enlace (Bind):** Se configura una estructura sockaddr\_in para especificar que el servidor debe escuchar en el puerto 80 (htons(80)) y en todas las interfaces de red disponibles (INADDR\_ANY). La función bind() asocia el socket a esta dirección y puerto.
4. **Escucha (Listen):** Se pone el socket en modo de escucha pasiva con listen(), preparándolo para aceptar hasta 5 conexiones en cola.
5. **Preparación de la Respuesta HTTP:** Se construye una cadena de caracteres (Block) que contiene la respuesta HTTP completa. Esta respuesta incluye la línea de estado (HTTP/1.0 200 OK), las cabeceras (Content-type y Content-length) y el cuerpo del mensaje, que es el código HTML de una página simple con un fondo naranja y un botón interactivo.
6. **Bucle de Aceptación:** El servidor entra en un bucle infinito (while(1)) para atender a los clientes.
   * La función accept() bloquea la ejecución hasta que un cliente se conecta. Cuando lo hace, accept() devuelve un nuevo descriptor de socket para esa conexión específica.
   * Se utiliza write() para enviar la respuesta HTTP (la cadena Block) al cliente a través del nuevo socket.
   * Finalmente, close() cierra la conexión con ese cliente, y el servidor vuelve al inicio del bucle para esperar la siguiente conexión

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <string.h>

int udp\_socket, udp\_cliente;

char Block[1000] = "";

struct sockaddr\_in servidor, cliente;

int main()

{

int lbind, llisten;

udp\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

if (udp\_socket == -1)

{

perror("\nError al tratar de abrir socket...");

exit(-1);

}

else

{

perror("\nExito al abrir socket...");

servidor.sin\_family = AF\_INET;

servidor.sin\_port = htons(80);

servidor.sin\_addr.s\_addr = INADDR\_ANY;

lbind = bind(udp\_socket, (struct sockaddr \*)&servidor, sizeof(servidor));

if (lbind == -1)

{

perror("\nError en el bind");

exit(0);

}

else

{

perror("\nExito en bind");

llisten = listen(udp\_socket, 5);

if (llisten == -1)

{

perror("\nError en el listen");

exit(0);

}

else

{

perror("\nExito en el listen");

sprintf(

Block,

"HTTP/1.0 200 OK\r\n"

"Content-type: text/html\r\n"

"Content-length: %d\r\n\r\n"

"<!doctype html>"

"<head>"

"<title>""HTTP""</title>"

"</head>"

"<body bgcolor=""orange"">"

"<h1>"

"Prueba practica HTTP"

"</h1>"

"<button type=button onclick=document.getElementById('ejemplo').innerHTML=""Date()"">"

"Fecha y hora"

"</button>"

"<p id=""ejemplo"">"

"</p>"

"</body>"

"</html>",

300

);

while (1)

{

int size\_cliente = sizeof(struct sockaddr\_in);

udp\_cliente = accept(udp\_socket, (struct sockaddr \*)&cliente, &size\_cliente);

if (udp\_cliente == -1)

{

perror("\nError en el accept");

exit(0);

}

else

{

perror("\nExito en el accept");

write(udp\_cliente, Block, strlen(Block));

close(udp\_cliente);

}

}

close(udp\_socket);

return 0;

}

}

}

}

A screenshot of a computer program

AI-generated content may be incorrect.  
Figura 1. Ejecución del programa en la terminal.

A screenshot of a computer

AI-generated content may be incorrect.  
Figura 2. Ejecución del HTML en el navegador, enviado en el programa por protocolo HTTP.

## Conclusión

La práctica se completó exitosamente, logrando implementar un servidor HTTP funcional en lenguaje C capaz de servir una página HTML simple a un navegador web. El desarrollo permitió comprender a bajo nivel el flujo de trabajo de un servidor web: la creación de un socket, su enlace a un puerto, la escucha de conexiones, la aceptación de clientes y el envío de una respuesta formateada según el protocolo HTTP.

Se concluye que, aunque los servidores web modernos son sistemas mucho más complejos, sus fundamentos se basan en los mismos principios de programación de sockets vistos en esta práctica. Este ejercicio ha sido fundamental para desmitificar el funcionamiento de la web y entender la estructura de una respuesta HTTP, incluyendo la línea de estado, las cabeceras y el cuerpo del mensaje.

## Pregunta

### ¿Cómo podrían generar un negocio a través de lo visto en la práctica?

Aunque un servidor tan simple no es un producto comercial, el principio de servir contenido web de forma optimizada sí lo es. Un negocio podría especializarse en crear servidores web ultraligeros y de alto rendimiento para dispositivos de Internet de las Cosas (IoT). Estos servidores, escritos en C para un uso mínimo de recursos, podrían ofrecer interfaces de configuración web para dispositivos embebidos, vendiéndose como una solución de software licenciada a fabricantes de hardware.

## Bibliografía

[1] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1," Internet Engineering Task Force (IETF), Jun. 1999. [En línea]. Disponible: <https://www.rfc-editor.org/rfc/rfc2616>.

[2] W. R. Stevens, B. Fenner, and A. M. Rudoff, UNIX Network Programming, Volume 1: The Sockets Networking API, 3rd ed. Boston: Addison-Wesley Professional, 2003.

[3] J. F. Kurose and K. W. Ross, Computer Networking: A Top-Down Approach, 8th ed. Boston: Pearson, 2021.