UA

Universidad
de Alicante

ESCUELA POLITÉCNICA SUPERIOR

SISTEMAS OPERATIVOS Y DISTRIBUIDOS

Curso académico 2024 / 2025
Semestre 1
Grado en Ingeniería en Inteligencia Artificial
Grupo 1

Profesora: Iren Lorenzo Fonseca
Alumno: Santiago Álvarez Geanta

Universitat d'Alacant
Universidad de Alicante

Septiembre - Octubre 2024

ÍNDICE

Librerías utilizadas

Antes de empezar con el código de nuestro Cliente y nuestro Servidor utilizaremos las siguientes librerías en ambos ficheros **.c** .

```c
// SOCKET
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/types.h>

// READ & WRITE
#include <fcntl.h>
#include <string.h>

// PROCESOS
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
```

Creación Cliente HTTP

Comenzamos creando una función de tipo void, nuestra función se encargará de crear el socket de nuestro cliente y conectarlo con el correspondiente servidor. Dentro de esta función declaramos las siguientes variables:

```c
void ClienteHTTP(char *dirIP, int puerto, char *recurso) {

    int sc; // Socket Clientes

    // Variables de Read - Write
    int bytesLeidos;
    char buffer[1024];

    // reutiliza una dirección de red asociada a un socket
    static struct sockaddr_in sa;
```

La primera, **sc**, guardará el descriptor de archivo generado por la llamada a la función socket(). La segunda y la tercera serán utilizadas a la hora de llamar a las funciones read() o write(). La última se encargará de almacenar toda la información (IPv4, puerto y dirección IP). Respecto a la dirección IP usaremos **inet_addr()** para realizar la conversión binaria (es necesario incluir el .h **<arpa/inet.h>**.

```c
// Crear el socket cliente
sc = socket(AF_INET, SOCK_STREAM, 0);
if (sc < 0) {
    perror("Error al crear el socket clienteHTTP\n");
    exit(1);
}
```

Respecto a la creación del socket y la conexión con el servidor, lo primero que haremos será crear un socket mediante la llamada **socket()**, esta función recibirá el protocolo de IP utilizado, y los datos almacenados en sa (puerto 9999 y dirección local 127.0.0.1).

```c
// Coenctamos el cliente con la dirección IP del servidor:
if (connect(sc, (struct sockaddr *)&sa, sizeof(sa)) < 0) {
    perror("Error al conectar con el servidor\n");
    close(sc);
    exit(1);
} else {
    printf("Conexión establecida\n");
}
```

Una vez creado el socket, procederemos a realizar un **connect()** para conectar el socket del cliente con los **datos de dirección de nuestro servidor** almacenados en la variable **sa**.

Cuando el cliente se haya conectado al servidor, será el servidor el encargado de tratar la petición que hay escrita en el socket del cliente. Una vez tratada la respuesta, se sobreescribirá en el socket del cliente y finalmente leeremos (en nuestra función **ClienteHTTP()**) el contenido que hay en el socket (respuesta del servidor). Por último cerramos el socket de nuestro cliente y daremos por finalizada la tarea.

```c
// Tiempo dónde el servidor esta realizando la petición GET


// Leemos el contenido escrito en el Socket del cliente
while ((bytesLeidos = read(sc, buffer, sizeof(buffer))) > 0) {
    printf("%s", buffer);
}

// Finalizamos el cliente
close(sc);
```

Creación Servidor HTTP

Respecto a la creación del socket, lo primero que haremos será crear un socket mediante la llamada socket(). Creado el socket, procederemos a realizar un **bind()** para asociar una dirección y puerto con el socket del servidor. El siguiente paso será establecer un **listen()** que nos permitirá atender a un número máximo de personas simultáneamente.

Finalmente utilizaremos un bucle while **True** que siempre estará activo (nuestro servidor tendrá que estar en constante funcionamiento). Dentro del bucle estaremos llamando a la función **accept()**, la cual se quedará a la espera de que algún cliente se conecte en la dirección asociada del servidor. En el momento que se recibe un cliente, nuestra función nos devolverá su socket con el que podremos tratar la solicitud enviada mediante un read() (en nuestro caso la solicitud siempre será **Google.html**). Es importante mencionar que para cada cliente que acepte el servidor realizaremos una llamada a fork() para que sea el proceso hijo quien trate a este cliente en concreto. También cabe mencionar que dentro de nuestro proceso hijo cerraremos el socket del servidor y trataremos únicamente con el socket del cliente que se haya conectado. Además, dentro del hijo llamaremos a nuestra función **procesarCliente()** la cual se encargará de leer el socket del hijo y posteriormente escribir en él el recurso pedido (el contenido de nuestro archivo Google.html).

Finalmente, enviaremos el contenido leído con read() mediante la llamada a **send()**. Una vez tratada la petición cerraremos el socket del cliente para evitar problemas posteriores.

```c
// Creación del socket, devuelve el descriptor del socket
ss = socket(AF_INET, SOCK_STREAM, 0);
if (ss < 0) {
    perror("Error en socket() servidorHTTP()\n");
    exit(1);
}

// Asociamos el socket con la dirección IP y puerto
if (bind(ss, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0) {
    perror("Error en bind() servidorHTTP()\n");
    exit(1);
}

// Disponibilidad para recibir peticiones de servicio
if (listen(ss, numClientesAtendidos) < 0) {
    perror("Error en listen() servidorHTTP()\n");
    exit(1);
}
```

```c
while(1) {
    // Aceptamos la conexión entrante
    sCliente = accept(ss, (struct sockaddr*)&client_addr, &client_len);

    if (sCliente < 0) {
        perror("Error en accept() servidorHTTP()\n");
        continue;
    } else {
        printf("Cliente conectado\n");
    }

    pid_t pid = fork();
    if (pid == 0) {
        close(ss); // Ya no necesitamos el socket de escucha
        procesarCliente(sCliente);
        exit(0);
    }
    else{
        close(sCliente);
    }
}
close(ss);
}
```

```c
void tratarPeticion(int sCliente) {

    // Abrimos archivo .HTML -> obtenemos su descriptor de Archivos
    int archivoPedido = open("Google.html", O_RDONLY);

    // Leemos el descriptor de archivo almacenando su informacion en el buffer
    while ((bytesLeidos = read(archivoPedido, buffer, sizeof(buffer))) > 0) {
        // Enviamos el contenido del buffer al socketCliente
        send(sCliente, buffer, bytesLeidos, 0);
    }
}

void procesarCliente(int sCliente) {
    tratarPeticion(sCliente);
    close(sCliente);    // Tratada la petición cerramos nuestro socketCliente
}
```

Santiago Álvarez Geanta Grupo 1

Universidad de Alicante