

Práctica 2.2 - Sistemas Operativos y Distribuidos

_ - Carlos Salas Alarcón - _

Cliente HTTP:

Este código implementa un cliente HTTP que se conecta a servidores web para recuperar recursos. La principal diferencia con el servidor radica en que el cliente inicia la conexión, mientras que el servidor espera pasivamente por conexiones entrantes.

La comunicación se basa en dos protocolos: TCP como protocolo de transporte e HTTP como protocolo de aplicación. La creación del socket con `AF_INET` y `SOCK_STREAM` configura una conexión TCP sobre IPv4, garantizando entrega confiable y en orden de los datos.

La estructura `sockaddr_in` se utiliza porque estamos trabajando exclusivamente con direcciones IPv4. Esta estructura contiene los campos necesarios para especificar una dirección completa: familia (siempre `AF_INET` para IPv4), dirección IP en formato numérico (obtenida mediante `inet_addr()`) y puerto en orden de bytes de red (convertido con `htons()`).

El proceso del cliente sigue una secuencia lineal: crear socket, configurar dirección del servidor, conectar, enviar petición HTTP, recibir respuesta y cerrar conexión. La petición HTTP se construye según el estándar, con el método GET, el recurso solicitado, la cabecera Host requerida por HTTP/1.1, y Connection: close para solicitar el cierre de la conexión tras la respuesta.

La función `connect()` realiza el establecimiento de la conexión TCP, mientras que `write()` y `read()` manejan el envío y recepción de datos a través del socket ya conectado. El programa principal procesa los parámetros de entrada para determinar a qué servidor conectarse y qué recurso solicitar, mostrando finalmente la respuesta completa del servidor en la salida estándar.

Servidor HTTP:

El código del servidor HTTP implementa un servidor web que espera conexiones entrantes de clientes y sirve archivos solicitados. A diferencia del cliente que inicia conexiones, el servidor opera de forma pasiva, esperando que los clientes se conecten a él.

El servidor utiliza los protocolos TCP para las conexiones de red y HTTP para la comunicación a nivel de aplicación. La creación del socket con `AF_INET` y `SOCK_STREAM` establece el uso de IPv4 y TCP. La estructura `sockaddr_in` almacena la dirección local del servidor, donde debe escuchar conexiones. Es crucial que el servidor especifique su propia dirección IP y puerto mediante `inet_addr()` y `htons()`, ya que esto determina en qué interfaz de red y puerto aceptará conexiones.

Las funciones `bind()`, `listen()` y `accept()` son exclusivas del lado del servidor. `Bind()` asocia el socket con la dirección local, `listen()` marca el socket como pasivo para aceptar conexiones, y `accept()` bloquea hasta que un cliente se conecta, devolviendo un nuevo socket para esa conexión específica.

Un aspecto importante es el uso de `fork()` para crear procesos hijos que manejen cada cliente concurrentemente. El proceso padre continúa aceptando nuevas conexiones mientras los procesos hijos atienden clientes individuales. Cada hijo cierra el socket de escucha principal y usa únicamente el socket de cliente.

La función `procesar()` recibe tres parámetros: la petición HTTP, un buffer para la respuesta, y `resp_size` que indica el tamaño máximo del buffer. Este último parámetro es necesario porque `HTTP_response` es un puntero dentro de la función, y `sizeof(HTTP_response)` devolvería el tamaño del puntero, no del buffer. Sin `resp_size`, no podríamos saber el tamaño real disponible.

El servidor lee la petición HTTP, procesa el recurso solicitado, busca el archivo correspondiente en el sistema de archivos local, y construye una respuesta HTTP con las cabeceras apropiadas y el contenido del archivo.

Diferencias entre roles y funcionamiento:

Este proyecto implementa un sistema completo cliente-servidor HTTP que funciona mediante línea de comandos. El servidor se inicia con `./servidor <IP> <puerto>` y espera conexiones, mientras el cliente se usa con `./servidor <IP> <puerto> <archivo>` para solicitar recursos. Ambos programas se comunican usando sockets TCP y el protocolo HTTP.

La implementación enseña las diferencias prácticas entre cliente y servidor. El cliente activamente conecta al servidor usando `connect()`, mientras el servidor usa `bind()`, `listen()` y `accept()` para esperar conexiones. El servidor maneja múltiples clientes simultáneos creando procesos hijos para cada conexión.

El desarrollo muestra desafíos reales como la gestión de memoria en C, especialmente al trabajar con buffers de tamaño fijo y la necesidad de pasar explícitamente tamaños entre funciones. La experiencia cubre conceptos esenciales de redes, concurrencia y programación de sistemas de forma práctica e integral.