Programación para las Comunicaciones - Práctica 1

Implementación de un Servicio de Historial de Accesos HTTP/HTTPS

Pedro José Fernández Escámez

3 de junio de 2025

Índice

1. Introducción

Esta práctica aborda el diseño e implementación de un servicio genérico de transporte utilizando los protocolos HTTP y HTTPS. El objetivo principal es desarrollar una aplicación cliente-servidor que ofrezca un servicio de Historial de Accesos, empleando HTTP/HTTPS como mecanismo de transporte y cookies para la gestión del estado entre peticiones. La implementación se realiza sobre sockets TCP en una arquitectura multi-hilo, atendiendo peticiones seguras y no seguras en puertos distintos. Se requiere autenticación tanto de cliente como de servidor para las conexiones HTTPS, utilizando certificados X.509 generados y firmados por una Autoridad Certificadora (CA) propia.

2. Funcionalidades Implementadas

La aplicación cumple con los siguientes requisitos funcionales obligatorios:

- **Servicio de Transporte HTTP/HTTPS:** Se ha implementado un servidor capaz de atender peticiones sobre HTTP (no seguro) y HTTPS (seguro) en puertos distintos (8080 para HTTP y 4430 para HTTPS).
- **Historial de Accesos con Cookies:** El servidor gestiona un historial de los recursos solicitados por un cliente. Este historial se almacena y transmite mediante cookies HTTP.
  - Si una petición incluye una cookie de historial, el nuevo recurso solicitado se concatena al historial existente.
  - Si la petición no incluye cookie, se genera una nueva con el recurso actual.
  - El servidor es sin estado respecto al almacenamiento del historial, dependiendo enteramente de la cookie del cliente.
- **Respuesta HTML:** El servidor responde a cada petición con una página HTML que muestra el último recurso solicitado, el número de recursos en el historial y la lista completa de dichos recursos.
- **Arquitectura Multi-hilo:** Cada servidor (HTTP y HTTPS) utiliza un hilo principal para escuchar conexiones. Cada conexión aceptada es manejada por un nuevo hilo dedicado (instancia de la clase `Handler`), permitiendo la concurrencia de múltiples clientes.
- **Autenticación SSL/TLS:** Para las conexiones HTTPS, se implementa autenticación mutua (servidor y cliente). El servidor requiere que el cliente presente un certificado válido firmado por la CA de confianza.
- **Cliente de Consola:** Se ha desarrollado un cliente simple de consola que permite al usuario elegir el protocolo (HTTP o HTTPS) y especificar el recurso a solicitar. Este cliente muestra la respuesta del servidor.

Adicionalmente, se han explorado/implementado las siguientes funcionalidades opcionales:

- **Persistencia de Cookies en Cliente de Consola:** El cliente de consola (tanto para HTTP como para HTTPS) guarda la cookie recibida en un archivo local y la carga al iniciar, permitiendo mantener el historial entre sesiones del cliente.
- **Conteo de Recursos Solicitados:** El servidor lleva un conteo global de cuántas veces se ha solicitado cada recurso, mostrando esta información en la respuesta HTML (esta funcionalidad se implementó en la clase `Handler` utilizando un mapa estático y sincronizado).

2.1. Certificados X.509

Para la capa de seguridad SSL/TLS, se ha establecido una infraestructura de clave pública (PKI) propia:

1. **Autoridad Certificadora (CA):** Se generó una CA raíz utilizando OpenSSL. Esta CA es la encargada de firmar los certificados del servidor y del cliente.
   - **DN de la CA:** `CN=ca, OU=ppc, O=umu, C=ES`
2. **Certificado del Servidor:** Se generó un par de claves y una solicitud de firma de certificado (CSR) para el servidor usando `keytool`. El CSR fue firmado por la CA para producir el certificado del servidor. Este certificado, junto con la clave privada del servidor y el certificado de la CA, se almacenó en un Java KeyStore (`servidor.jks`).
   - **DN del Servidor:** `CN=localhost, OU=ppc, O=umu, C=ES`
3. **Certificado del Cliente:** De forma análoga, se generó un par de claves y un CSR para el cliente, que fue firmado por la CA. El certificado resultante, su clave privada y el certificado de la CA se almacenaron en otro Java KeyStore (`client.jks`).
   - **DN del Cliente:** `CN=Pepo Cliente, OU=ppc, O=umu, C=ES`
4. **TrustStores:** Se crearon Java TrustStores (`servertrust.jks` para el servidor y `clienttrust.jks` para el cliente) que contienen únicamente el certificado público de la CA. Esto permite a cada entidad verificar la autenticidad de los certificados presentados por la otra parte.

El proceso de generación siguió las pautas de creación de una CA con OpenSSL y la generación/firma de certificados de entidad final con `keytool` (para la gestión de claves y CSRs) y OpenSSL (para la firma por parte de la CA).

3. Implementación

3.1. Organización del Código

El sistema se compone de las siguientes clases principales, siguiendo un diseño modular cliente-servidor:

- `Application`: Clase principal que inicia los servidores HTTP y HTTPS en hilos separados. También contiene la lógica del cliente de consola interactivo, permitiendo al usuario realizar peticiones.
- `HTTPServer`: Hilo que actúa como servidor HTTP. Escucha en el puerto 8080, acepta conexiones y delega cada una a un nuevo hilo `Handler`.
- `SSLServer`: Hilo que actúa como servidor HTTPS. Configura el contexto SSL/TLS con los keystores y truststores necesarios, habilita la autenticación de cliente, y escucha en el puerto 4430. Al igual que `HTTPServer`, delega las conexiones a hilos `Handler`.
- `HTTPClient`: Encapsula la lógica para enviar peticiones HTTP al servidor. Gestiona la conexión, el envío de la petición (incluyendo cookies) y la recepción e impresión de la respuesta. Implementa persistencia de cookies.
- `SSLClient`: Similar a `HTTPClient`, pero para conexiones HTTPS. Configura el contexto SSL/TLS del cliente con sus keystores y truststores para la autenticación mutua. Implementa persistencia de cookies.

- `Handler`: Clase que procesa las peticiones de los clientes, tanto HTTP como HTTPS. Es instanciada por `HTTPServer` y `SSLServer`. Extrae el recurso, maneja las cookies para el historial, y genera la respuesta HTML. Implementa el conteo de accesos a recursos.

Diagrama de Clases (Conceptual):

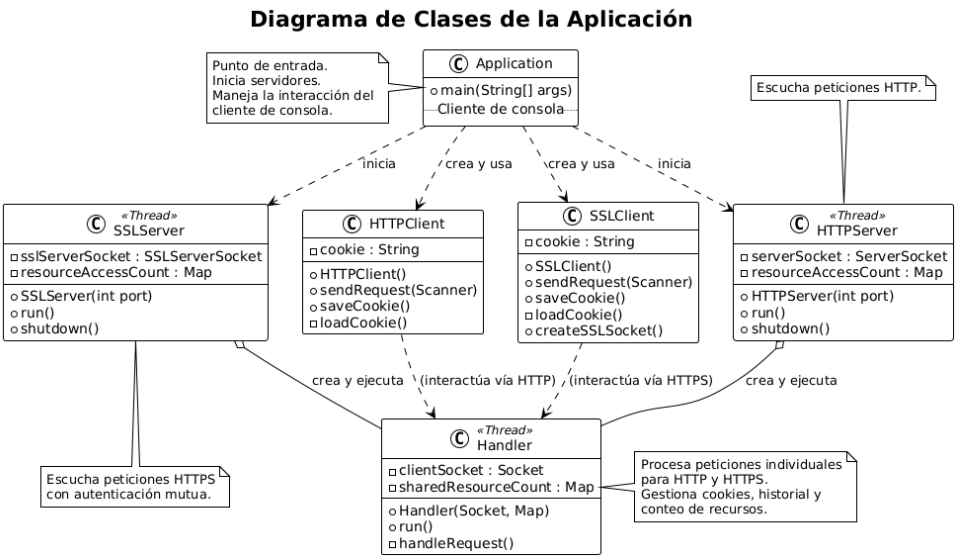

Figura 1: Diagrama de clases simplificado del sistema.

3.2. Protocolo de Comunicación

La comunicación se basa en el protocolo HTTP/1.1 sobre TCP.

- El cliente (de consola o navegador) establece una conexión TCP (o TLS sobre TCP para HTTPS) con el servidor.
- El cliente envía una petición HTTP GET, por ejemplo:

```
GET /recurso_solicitado HTTP/1.1
Host: localhost
Cookie: historial=recurso1,recurso2
Connection: close
```

Listing 1: Ejemplo de Petición HTTP GET

- El servidor procesa la petición:
  1. Extrae el recurso de la línea de petición.
  2. Extrae la cookie ”historial”si existe.
  3. Actualiza/crea el historial concatenando el nuevo recurso.
  4. Incrementa el contador para el recurso solicitado.
- El servidor envía una respuesta HTTP, por ejemplo:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Set-Cookie: historial=recurso1,recurso2,recurso_solicitado; Path=/; SameSite=Lax
Connection: close
```

```
<!DOCTYPE html>... (cuerpo HTML con la informacion del historial) ...
```

Listing 2: Ejemplo de Respuesta HTTP

Para HTTPS, el intercambio de datos HTTP es precedido por un handshake SSL/TLS donde se negocian los parámetros de seguridad y se realiza la autenticación mutua mediante los certificados X.509.

Diagrama de Secuencia (Conceptual):

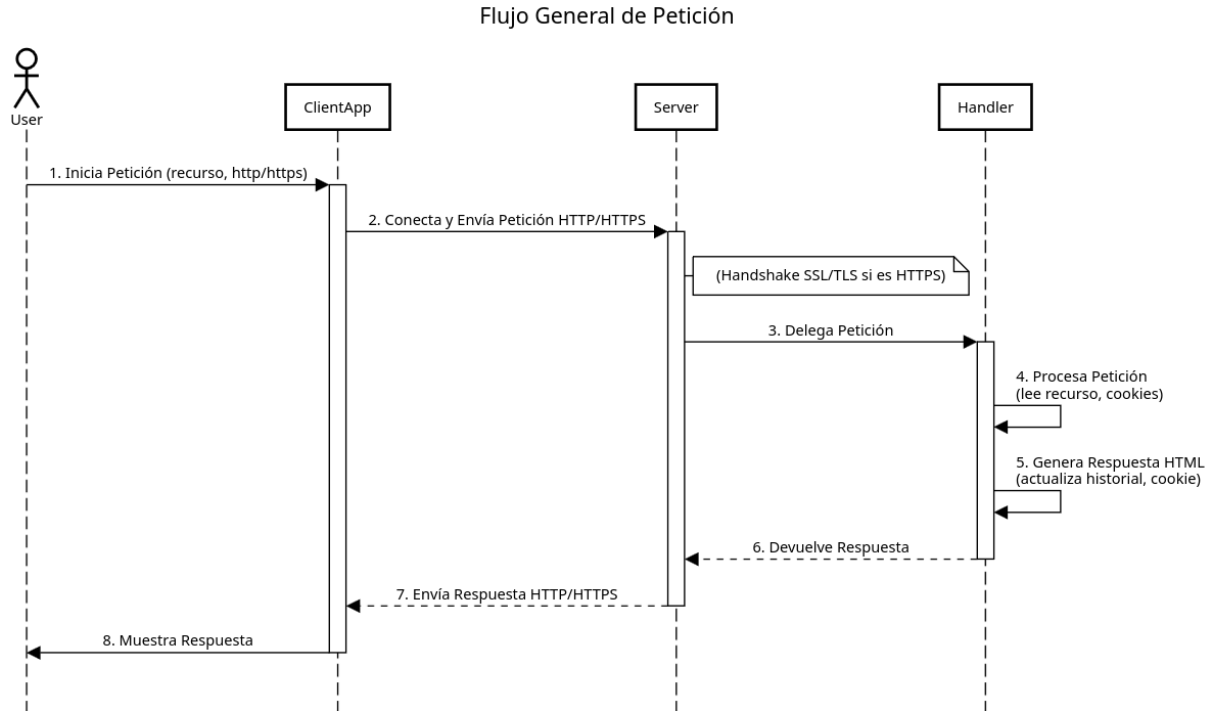

Figura 2: Diagrama de secuencia para una petición HTTP.

3.3. Operaciones de E/S

La entrada y salida de datos a través de los sockets se maneja mediante streams. Se utilizan `BufferedReader` para leer las peticiones línea a línea desde el `InputStream` del socket, y `DataOutputStream` (para el cliente) o `PrintStream` (para el servidor en el `Handler`) para escribir las respuestas en el `OutputStream`. Se utiliza `flush()` para asegurar que los datos se envían inmediatamente. Se emplea codificación UTF-8 para la correcta interpretación de caracteres.

3.4. Hilos

La aplicación utiliza múltiples hilos para lograr concurrencia y responsividad:

- La clase `Application` inicia dos hilos principales, uno para `HTTPServer` y otro para `SSLServer`.
- Cada instancia de `HTTPServer` y `SSLServer` ejecuta un bucle de escucha en su hilo. Al aceptar una nueva conexión de cliente, crea una nueva instancia de la clase `Handler` y la ejecuta en un nuevo hilo (`handler.start()`). Esto permite que el servidor principal siga escuchando nuevas conexiones mientras los hilos `Handler` procesan las peticiones existentes de forma concurrente.
- Se utiliza un ExecutorService (`Executors.newFixedThreadPool`) en las clases servidor para gestionar de forma más eficiente los hilos de los Handlers.

4. Ejecución

Para ejecutar la práctica:

1. **Compilar el código fuente:** Asegurarse de que todas las clases Java (`Application.java`, `HTTPServer.java`, `SSLServer.java`, `Handler.java`, `HTTPClient.java`, `SSLClient.java` y sus paquetes correspondientes) están compiladas.
2. **Disponibilidad de Certificados:** Los archivos de keystore y truststore (`servidor.jks`, `client.jks`, `servertrust.jks`, `clienttrust.jks`) deben estar en el subdirectorio `certs/` relativo a la ubicación desde donde se ejecuta la aplicación, o las rutas en el código deben ser ajustadas.
3. **Ejecutar la Clase Principal:** Ejecutar la clase `application.Application`. Esto iniciará los servidores HTTP y HTTPS.

```
java application.Application
```

Listing 3: Ejecución desde la línea de comandos (ejemplo)

4. **Interacción con el Cliente de Consola:**
   - La consola mostrará un mensaje preguntando el protocolo a utilizar (`http/https/salir`).
   - El usuario introduce el protocolo deseado.
   - Posteriormente, se solicita el recurso a pedir (ej. `/pagina1.html`).
   - La respuesta del servidor (cabeceras HTTP y cuerpo HTML) se mostrará en la consola del cliente.
5. **Acceso mediante Navegador Web:**
   - Para HTTP: Abrir un navegador y acceder a `http://localhost:8080/recurso_deseado`.
   - Para HTTPS: Abrir un navegador y acceder a `https://localhost:4430/recurso_deseado`.
     - Es necesario haber importado previamente el certificado de la CA (`cacert.pem`) en el almacén de autoridades de confianza del navegador.
     - Si se desea probar la autenticación de cliente desde el navegador, el certificado de cliente (exportado a formato PKCS#12, e.g., `client_exportado.p12`) debe importarse en el almacén personal de certificados del navegador.

Los servidores permanecerán activos hasta que se cierre la aplicación `Application` (por ejemplo, escribiendo salir en el prompt del cliente de consola o interrumpiendo el proceso).