Departamento de Arquitectura

Redes de Computadoras
Práctico 3

Curso 2024

Objetivos

- Introducir la API de sockets del curso.
- Familiarizarse con el uso de sockets.
- Comprender las arquitecturas de las aplicaciones que usan sockets.

Duración

- 3 clases.

Ejercicio 1 La API de sockets TCP de varios lenguajes ofrece una llamada `read_line()` o equivalente, que garantiza la entrega de una línea completa que fue transmitida sobre el stream. Por ejemplo, si el stream es "`line1\nline2\n`", dos llamadas consecutivas a `read_line()` devolverían "`line1`" y "`line2`", respectivamente.

Implemente una versión que pueda funcionar de forma bloqueante y no bloqueante en un lenguaje de alto nivel. Puede asumir que cuenta con un socket ya conectado y no bloqueante. Puede utilizar llamadas al sistema para consultar la hora actual.

Ejercicio 2 Se desea implementar una aplicación con una arquitectura cliente-servidor de intercambio de mensajes que cumpla con las siguientes características:

- El servidor espera conexiones TCP en todas sus interfaces en el puerto 8081.
- Los clientes envían dos tipos de mensajes:
  1. **ECHO**: *texto*\n
  2. **EXIT**\n
- Al recibir un mensaje de tipo **ECHO** el servidor debe contestar al cliente con texto.
- Al recibir un mensaje de tipo **EXIT** el servidor deberá contestarle "**CLOSE** `ip_cliente`" donde `ip_cliente` es la dirección IP del cliente que envió el comando y cerrar la conexión con dicho cliente.

Se pide:

(a) Implemente en un lenguaje de alto nivel, utilizando las primitivas de la API de sockets del curso, el programa que ejecuta el servidor.

(b) Implemente en un lenguaje de alto nivel, utilizando las primitivas de la API de sockets del curso, el programa que ejecuta un cliente que envía los mensajes 1 y 2.

Ref. Primer parcial 2019

Ejercicio 3 Sea *ceroconf* un protocolo de descubrimiento de servicios. En este protocolo un servicio se anuncia en la red local enviando mensajes UDP broadcast al puerto 1234. Estos mensajes tienen el formato "`nombre_de_servicio\n numero_de_puerto`", y deben ser enviados cada 30 segundos. Se asume que el servicio a anunciar es un servidor TCP escuchando en el puerto indicado.

Sea *tcp_cam* un servicio que permite acceder a una cámara web mediante TCP. Para esto el servicio acepta conexiones TCP y hace streaming en cada conexión leyendo los datos a enviar desde el archivo /dev/video0. Este servicio se anuncia a si mismo usando *ceroconf* con el nombre_de_servicio "`tcp_cam`".

Sea *video_capture* una aplicación que espera la aparición de servicios *tcp_cam* en la red local (anunciados con *ceroconf*). Cada vez que detecta una nueva instancia se conecta a ella y queda leyendo el stream, escribiéndolo a disco en un archivo. La aplicación puede descargar varios streams simultáneamente.

Se pide: Implemente el servicio *tcp_cam* y la aplicación video_capture en un lenguaje de alto nivel. Tiene disponible la APIs de sockets, concurrencia y lectura/escritura de archivos, así como bibliotecas de estructuras de datos.

Ref. Examen Julio 2017

Ejercicio 4 Se desea implementar la aplicación `redireccion` que permita implementar la redirección de un puerto en el host que la ejecuta a un puerto en otro host (port-forwarding). La aplicación será ejecutada, a modo de ejemplo, de la siguiente forma:

```
> redireccion 8080:www.debian.org:80
```

En este ejemplo, mientras se ejecuta la aplicación, las conexiones TCP al puerto 8080 del host local serán redireccionadas al puerto 80 del host www.debian.org. La implementación debe permitir conexiones simultaneas, y no debe asumir cambios en la aplicación que inicia la conexión. Asuma que se cuenta con primitivas para el manejo de *sockets* (establecimiento de conexión, envío y recepción, etc.), hilos y funciones de parsing de los comandos.

Se pide:

(a) Implemente en un lenguaje de alto nivel el comando `redireccion`.

(b) ¿Que utilidad encuentra para el comando implementado?

Ref. Examen Julio 2016

Ejercicio 5 Sea la red mostrada en al figura, donde un equipo servidor proxy recibe conexiones web desde internet (puerto 80) y conecta con equipos internos a la red, configurados con direcciones IP privadas. En internet, el DNS devuelve la IP pública del *proxy* (12.12.12.128) para estos dos nombres de dominio `www.serviciouno.uy` y `www.serviciodos.uy`.

El *servidor proxy* debe recibir y procesar los accesos a los dos dominios mencionados. Como los dominios son atendidos por diferentes servidores, se deben redirigir los pedidos al servidor que corresponda.

El *servidor proxy* debe funcionar de manera transparente, es decir, los clientes deberán poder usar el servicio web como si estuviesen directamente conectados a los servidores finales.

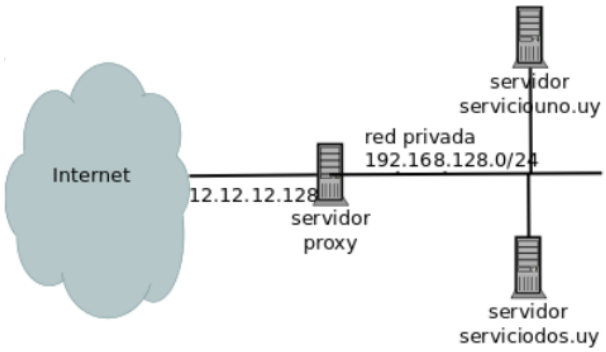

Figura 1: Arquitectura proxy

Se pide:

(a) Indique de que forma el servidor proxy puede determinar a que servidor desea acceder el cliente.

(b) Implemente en un lenguaje de alto nivel, utilizando las primitivas de sockets, el programa que ejecuta el servidor proxy. El servidor proxy deberá ser capaz de atender múltiples clientes simultáneamente. Defina claramente (pero no implemente) las funciones que utilice para parsear los mensajes HTTP.

Ref. Examen Diciembre 2017

Ejercicio 6 Sea la red mostrada en la figura 2, donde un equipo analizador analiza los logs recibidos por los equipos servidores web y en caso de encontrar actividad sospechosa genera una alerta. Esta alerta debe ser recibida por todos los administradores de la red (Admin1, ..., Admin N). Cada servidor web envía las entradas de log, a medida que se producen, al analizador. Esto se hace mediante una conexión TCP al puerto 2562 que permanece abierta mientras el servidor web está activo. Los administradores escuchan las alertas por UDP en el puerto 8523.

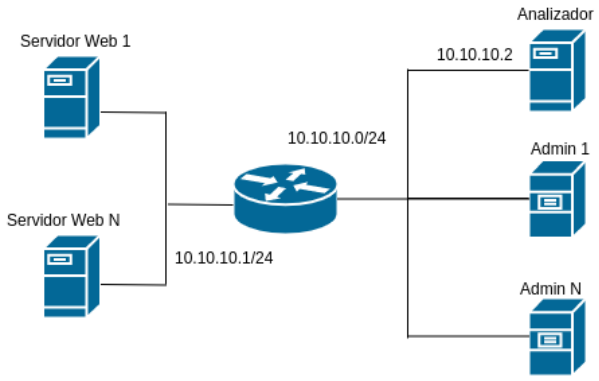

Figura 2: Arquitectura

El archivo de log sigue el formato de los Apache access log, un ejemplo del contenido de este archivo son las siguientes entradas (líneas):

```
177.38.32.248 - - [04/May/2019:00:00:02 -0300] "GET /centos/repomd.xml HTTP/1.1"
200 3445\n
2620:113:80c0:8::20 - - [04/May/2019:00:00:02 -0300] "GET /opensuse/ HTTP/1.1"
200 1190\n
2001:41c9:1:3ce::1:10 - - [04/May/2019:00:00:02 -0300] "GET /raspbian/ HTTP/1.1"
200 1660\n
```

Se pide:

(a) Implemente en un lenguaje de alto nivel, utilizando las primitivas de la API de sockets, el programa que ejecuta el analizador. Deberá ser capaz de atender múltiples conexiones de los servidores simultáneamente.

Para analizar las entradas de log cuenta con la función `analizar_solicitud(req_http):bool` que recibe como entrada una solicitud HTTP (línea del log) y retorna true en caso de que sea sospechosa. Además, se cuenta con la función `crear_alerta(solicitud_http):string` que crea la carga útil de una alerta para dicha solicitud. Puede asumir implementadas las funciones de manejo de strings que considere necesarias, pero debe explicar su funcionamiento.

(b) Implemente en un lenguaje de alto nivel, utilizando las primitivas de la API de sockets, el procedimiento `enviar_log(archivo_log)` que utilizan los servidores web para enviar las entradas de log al analizador. Para esto dispone de la llamada bloqueante `read_line(file):string`, que consume y devuelve una línea completa de un archivo.

Ref. Examen Julio 2020

Ejercicio 7 Se desea implementar un programa que permita descargar una página web desde un servidor Web que soporta el protocolo HTTP1.0.

El programa será invocado como `wget URL PORT`, por ejemplo:

```
wget example.com/index.html 80
```

Al ejecutarse el programa escribirá a disco el archivo html y todas las imágenes (que usen el tag <img>) que esta página contenga. Todas las imágenes deberán descargarse en paralelo. No hay limitaciones arbitrarias sobre el tamaño de los objetos que se descargan.

Se pide:

(a) Implemente en un lenguaje de alto nivel el programa descrito. Dispone de la API de sockets del curso, estructuras de datos estándar, funciones para parsear strings, etc. También dispone de una función `list=find_images(filename)` que recibe el nombre de un archivo en formato `html` y devuelve una lista de urls de imágenes que aparecen en tags <`img`>.

(b) Explique las modificaciones necesarias para que su programa soporte el protocolo HTTP1.1.

Ref. Examen Diciembre 2018

Ejercicio 8 Se desea implementar un servicio de transmisión de video en línea (*streaming*) para usuarios en Internet. El mismo utilizará los protocolos TCP y UDP y buscará distribuir el video entre todos los clientes conectados simultáneamente. Además, se desea contar con una aplicación cliente para reproducir los videos recibidos.

Descripción del servidor

El servidor es la aplicación que permite hacer streaming de video. Este servicio tiene la capacidad de realizar la transmisión utilizando el protocolo TCP o UDP.

Para el caso de TCP, el servidor acepta conexiones TCP en una dirección IP y puerto conocidos. Por cada conexión se realiza el *streaming* leyendo los datos a enviar usando la llamada `getNextFrame()`. El *streaming* hacía un cliente finaliza cuando el cliente cierra la conexión.

Para el protocolo UDP el servidor espera por un mensaje de solicitud en una dirección IP y puerto conocidos y realiza el *streaming* hacia el cliente que hizo el pedido leyendo los datos a enviar usando la llamada `frame=getNextFrame()`. El cliente debe renovar la solicitud del streaming cada 30 segundos y el servidor debe finalizar la transmisión del video a un cliente si no ha recibido solicitudes por mas de 90 segundos.

Descripción del cliente

El cliente realizará la reproducción de los streams recibidos. Para esto permitirá al usuario seleccionar el servicio TCP o UDP. Luego de seleccionada la opción el cliente se conecta al servidor y queda leyendo el stream, reproduciendo el video en pantalla. Para desplegar un frame en pantalla se dispone de la llamada `displayFrame(frame)`.

Para el caso UDP el cliente debe renovar su suscripción con el servidor cada 30 segundos.

Se pide:

(a) Implemente el servidor y el cliente en un lenguaje de alto nivel a su elección. Deberá definir el mensaje de suscripción para el caso UDP. Puede asumir que el tamaño del frame menor que el tamaño máximo de un segmento UDP. Recuerde que la API de sockets TCP no es basada en mensajes sino en streams. Deberá tener esto en cuenta para el envío de los frames de video.

(b) Explique detalladamente como debería cambiar su solución si el supuesto anterior ("el tamaño de un frame de video es siempre menor que el tamaño máximo de un segmento UDP") deja de ser válido. Su propuesta debe ser robusta a la pérdida o reordenamiento de paquetes. Justifique.

(c) Suponga que el streaming se hace sobre una red inalámbrica con muchas pérdidas. ¿Qué se puede esperar cuando se usa UDP? ¿Y TCP?

(d) Se desea distribuir un mismo video entre muchos usuarios en la red local, qué opciones hay? ¿Y si los usuarios están en internet?

Ref. Obligatorio 2, 2017

Ejercicio 9 Escriba dos programas, `client.c` y `server.c`, en lenguaje C, con las siguientes funcionalidades:

(a) El programa `client.c` abre una conexión a un servidor TCP, en IP y puerto conocidos, y le envía el string "`0123456789`" una vez por segundo. Todo lo que llega por la conexión abierta se imprime en consola, cada lectura en una línea nueva.

(b) El programa `server.c` implementa un *echo server*, esto es, un servidor que responde todo lo que llega enviando una copia hacia el emisor. El servidor acepta una conexión por vez.

(c) Además, discuta las siguientes preguntas:

i. ¿Qué tipo de sockets usó, bloqueantes o no bloqueantes? ¿Qué sucedería si en su programa cambiara la configuración de los sockets, cómo se comportaría?

ii. Suponga que mientras los programas están corriendo alguien desconecta un cable de red que los comunica, ¿qué sucedería?

iii. ¿Qué pueden decir sobre lo que el cliente imprime en la consola? ¿Las líneas son todas iguales? ¿Pueden haber huecos en la numeración? ¿Pueden haber líneas vacías?

Nota: el ćodigo C debe ser lo más correcto posible.