Universidad Nacional Autónoma de México
Facultad de Ciencias

Redes de Computadoras
Semestre: 2025-1

Práctica 4

Equipo Bolivar el Heroe

Bonilla Reyes Dafne - 319089660
García Ponce José Camilo - 319210536
González Peñaloza Arturo - 319091193
Main Cerezo Ashael Said- 319260658
Raudry Rico Emilio Arsenio - 318289276

TEORÍA

Por último contesta las siguientes preguntas:

1. Investiga en qué consisten las distintas versiones de HTTP.
   Respuesta:
   - HTTP/0.9. Primera versión de HTTP, era simple, pues la peticiones solo consistían en una línea de código con el único método disponible **get** seguida del path del archivo a consultar. El regreso constaba del propio archivo HTML solicitado.
   -HTTP/1.0. Una línea de código sobre el estatus de la respuesta se había incluido, al igual que el conceptos de los **headers**, haciendo flexible y extensible la transmisión de metadatos. Se podía envíar documentos además de HTML planos.
   -HTTP/1.1. Primera estandarización de HTTP. Se podía reutilizar una misma conexión. Se agregó un **Pipeline**. Introdujo mecanismos de cache. Había una negociación de contenido entre servidor y cliente. Permitía hostear diferentes dominios para misma dirección IP.
   -HTTP/2. Es un protocolo binario en lugar de uno de texto.Se pueden hacer solicitudes paralelas sobre la misma conexión. Comprime **headers** para evitar duplicidad de datos transmitidos. Permite a un servidor poblar datos en el cache de un cliente por medio de “server push”.
   -HTTP/3. Consta de la misma semántica que sus versiones anteriores, pero usa QUIC en lugar de TCP para la capa de transporte. QUIC provee una menor latencia y además ejecuta múltiples transmisiones sobre UDP e implementa una detección de pérdida de paquetes y retransmisiones independiente para cada transmisión.

2. Investiga algún protocolo similar a HTTP.
   Respuesta: SPDY fue un protocolo experimental introducido por Google a inicios de los 2010’s. Funcionaba sobre TCP/IP, resolvió el problema de datos duplicados y una mejora en la responsividad. De igual manera era capaz de mejorar el rendimiento en las comunicaciones entre servidor y cliente hasta un 64%. Cuando se introdujo HTTP/2, quedó obsoleto.

3. Aunque si bien, algunos juegos son 1 a 1, suponiendo que se puede, de qué manera se podría meter otra conexión adicional para que funcione con más personas.
   Respuesta: Podría elaborar un sistema que reciba múltiples conexiones por medio de un ServerSocket. De esta manera se podría crear un hilo por cada conexión que se encargue de la comunicación, la cual se pueda definir por medio de un protocolo sencillo pero claro sobre los movimientos y estados del juego.

4. Qué ventajas ves sobre este tipo de conexión, así como desventajas.
   Respuesta: A pesar de ya contar con poder recibir más conexiones, el manejo e incorporaciones de los nuevos jugadores dentro del juego queda pendiente.

5. Suponiendo que tuviera más ejecuciones tu servidor por otros usuarios, contesta las siguientes preguntas:
   a) Si un usuario ya tiene iniciada alguna partida y otro se conecta ¿Juego la

misma partida que el usuario 1 o se crea una nueva?

Respuesta: Se crea una nueva, se estaría jugando una partida diferente.

b) Si dos usuarios se conectan casi simultáneamente ¿a quién atiende primero el servidor?

Respuesta: La elección sería bastante arbitraria, para dar alguna preferencia se tendría que incluir un sistema de prioridades.

c) Suponiendo que n usuarios se conectan al servidor, no importa en qué orden ¿Cómo procesa las solicitudes el servidor?

Respuesta: El servidor estaría creando un juego por cada solicitud, donde cada jugador estaría compitiendo contra la computadora. Para que los jugadores interactúen entre sí, depende de la lógica del juego.

d) Propón una mejora para que se puedan atender las solicitudes de acuerdo a lo siguiente:

i. Siendo una programación sin alguna biblioteca adicional.

Respuesta: Para las condiciones de carrera y demás problemas de solicitudes que se podrían presentar, podríamos implementar alguna estructura de datos concurrente que regule dichas solicitudes por su propio diseño. De esta manera todos los jugadores estarían en el mismo juego y además tendría un orden sus turnos.

ii. Usando un framework o biblioteca adicional.

Respuesta: Haciendo uso de mecanismos de sincronización ya proporcionados por Java (como synchronized). Si tenemos una cola que guarde las solicitudes, este método manejaría su acceso para evitar problemas concurrentes.

6. Con lo investigado en la pregunta anterior, menciona algunos servicios comerciales y otros no tan comerciales donde se emplee el cliente servidor y como lo hace (a grandes rasgos, no en cuestión de programación).
   Respuesta: Dentro de los comerciales se encuentra Amazon Web Services, que provee almacenamiento y análisis de datos de las empresas contratantes por medio de APIs. Otro ejemplo sería Netflix, donde los suscriptores solicitan la visualización de contenido audiovisual en tiempo real a los servidores.

   Por el lado de los no comerciales estaría Wikipedia que provee el acceso de artículos informativos a cualquier usuario que lo solicite. También estarían los foros de preguntas donde los usuarios conversan en pro de resolver problemas o de obtener mayor información.

7. Escribe lo aprendido sobre esta práctica así como dificultades.
   Respuesta: Aprendimos cómo implementar de manera sencilla un juego mediante cliente-servidor. Además, al percatarnos de las implicaciones que trae consigo la incorporación de más jugadores, apreciamos la necesidad de protocolos para la comunicación de solicitudes y de mecanismos de sincronización que manejen el acceso a estas. No hubo dificultades en la elaboración del programa, quizás en llegar a un acuerdo sobre cómo se imprimiría estéticamente en la terminal.

Referencias

1. Jesin, A. (2014). Packet Tracer Network Simulator. Packt Publishing Ltd.
2. Bhardwaj, R. (2020). Switchport Access Mode vs Trunk Mode - IP With Ease. IP with Ease. https://ipwithease.com/switchport-trunk-mode-vs-access-mode/

3. https://developer.mozilla.org/en-US/docs/Web/HTTP/Evolution_of_HTTP