Control 1
Redes
Plazo de entrega: 10 de abril de 2024

José M. Piquer

Intro

El control se responde en forma individual, sin preguntas entre Uds o a terceros. Todas las dudas pregúntenlas en el foro de U-cursos y el equipo docente les responderá. Entreguen las tres preguntas en un PDF con su nombre por U-cursos. Pueden (y deben) buscar material en Internet para responder, pero citen siempre las fuentes que utilizan. Pueden usar chatGPT para ayudarlos, pero, no pueden entregar su respuesta directamente, deben explicar y analizar todo lo que les propone. Recuerden que chatGPT se equivoca, alucina y miente con mucha convicción. Siempre recuerden mencionarlo si lo utilizan.

En épocas de chatGPT estamos obligados a evaluar mal respuestas que antes habríamos aceptado como casi correctas. Si Uds entregan un párrafo de argumentos correctos pero no responden nunca la pregunta específica con argumentos técnicos que corresponden a ese caso, hoy en día tienen cero puntos.

Un ejemplo, decir algo como:"existen algoritmos propuestos con mejoras para TCP para enlaces de alto delay como CUBIC y BBR, que buscan resolver este problema" antes les habrían dado algún puntaje (significaba que habían hecho una búsqueda en google al menos). Hoy en día vale cero.

Las preguntas de los controles las chequeamos con chatGPT antes de ponerlas, y la idea es que chatGPT no logre sacarse un 4.0.

Entonces, la recomendación es: contesten la pregunta específica del enunciad. Las frases periféricas que buscan adornar sin ir al detalle técnico no aportan en nada, de hecho más bien perjudican, por que mientras más hablan sin ir al punto, más generan la impresión en el corrector que no conocen la respuesta (y suenan igual que chatGPT).

P1: sockets

Para la Tarea 1, propusimos usar el servidor de eco que viene en el código de ejemplos: `server_echo_udp2.py` para que probaran localmente.

Sin embargo, en esos mismos ejemplo viene un servidor mucho más simple: `server_echo_udp.py`

Analice las diferencias entre ambos y pruebe si la segunda versión también serviría para la Tarea 1 (en ambos casos, hay que modificar el tamaño máximo del mensaje que se recibe desde el socket).

Responda las siguientes preguntas:

1. ¿Cuál de las dos es mejor solución para eco?
2. En un servidor general (no de eco), ¿Cuándo deberíamos usar un estilo o el otro de solución?
3. Si en el caso de la Tarea 1 el servidor debe atender muchos clientes simultáneos enviando archivos muy grandes, ¿cambia su selección de mejor solución? ¿Por qué?
4. El cliente de eco que Uds tienen que modificar para la Tarea 1 envía un mensaje inicial y espera la respuesta (para descartarlo) antes de entrar al ciclo principal. ¿Por qué hace eso? ¿Sería necesario para ambos tipos de servidores?
5. Si ahora uso un socket TCP (SOCK_STREAM) en vez de UDP, ¿qué pasa con las dos versiones de servidores discutidas aquí?

P2: NTP

En NTP, existe una versión del servidor donde simplemente hace *broadcast* hacia la red, enviando un paquete UDP con la hora actual a todos los dispositivos que están conectados en la red local en ese momento y están escuchando en el puerto NTP.

Esta versión no es la favorita hoy en día, por que no permite mantener mucha precisión en el ajuste de los relojes.

Responda las siguientes preguntas:

1. ¿Qué limita la precisión en este caso?
2. ¿A qué precisión puedo aspirar?
3. Proponga ideas para mejorar esa limitación

4. ¿Existen escenarios en que el esquema de *broadcast* igual sea mejor?

5. Si recibo una hora que es anterior a la mía (mi reloj está "adelantado"), ¿puedo retroceder el reloj?

P3: Arquitectura de Servidores

Una empresa de sistemas de vigilancia quiere generar un servidor de video que permita mirar muchas cámaras dentro de una organización, desde Internet. Para esto, necesitan programar un servidor que recibe clientes desde Internet que miran el conjunto de las cámaras, y el mismo servidor es cliente de múltiples cámaras IP (ver diagrama).

Para esto, el servidor de video debe manejar muchos sockets conectados a las cámaras (digamos, más de 20) y aceptar conexiones de Internet de clientes que quieren ver esas cámaras en tiempo real.

El problema es que estas cámaras soportan un sólo cliente TCP a la vez y envían datos que sólo transmiten los cambios en el video (las zonas que se mueven), para ahorrar capacidad y ancho de banda.

Entonces, el servidor debe ser el único cliente que se conecta a las cámaras, y arma en memoria el video a transmitir para cada una. Pero, hacia Internet debe ser capaz de atender múltiples clientes. Mientras más clientes se puedan tener en Internet, mejor.

Tienen cuatro propuestas de implementación que les han traído sus ingenieros para atender a los clientes de Internet:

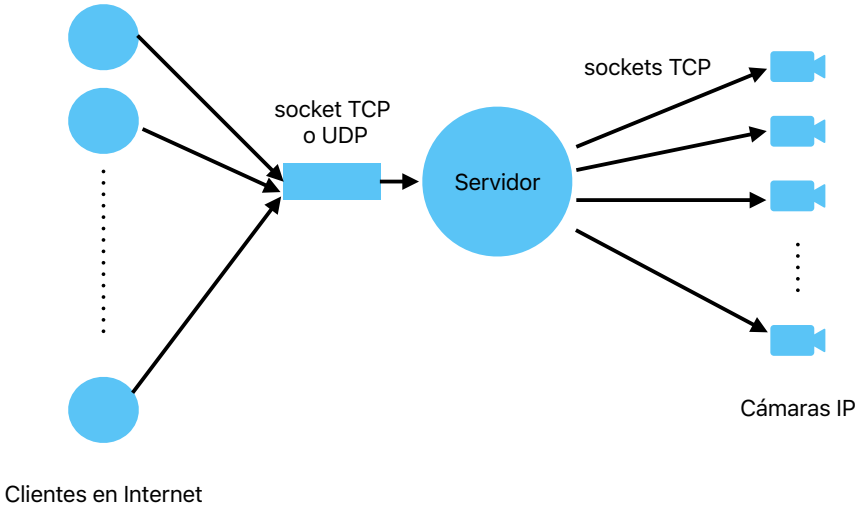
1. Usar procesos pesados: cada cliente es atendido por un servidor dedicado, con un socket TCP (SOCK_STREAM) para cada uno.

2. Usar threads: cada cliente es atendido por un thread dedicado, con un socket TCP (SOCK_STREAM) para cada uno.

3. Usar select: todos los clientes son atendidos por el mismo proceso que usa select() para definir a qué socket atender, cada cliente tiene un socket TCP (SOCK_STREAM) propio.

4. Socket común: todos los clientes comparten el mismo socket UDP (SOCK_DGRAM) para enviar sus requerimientos, el servidor es un proceso que va atendiendo los requerimientos uno por uno.

Analice las distintas alternativas y discuta sus pro y contras. Elija la alternativa que Ud usaría en un caso así y justifique su elección.


sockets TCP
socket TCP
o UDP
Servidor
Cámaras IP
Clientes en Internet