DOCUMENTACIÓN DEL PROYECTO

Simulador de Cliente-Servidor con Algoritmo NAT en Ubuntu Linux Usando el Lenguaje de Programación C++

Brayan Mejía Mora
Yael Alfredo Salazar Aguilar
Mauricio Salas García
15/12/2023

Contents

Objetivos

- Comprender el funcionamiento del algoritmo NAT (Traducción de Direcciones de Red)
- Comprender los fundamentos de la arquitectura de Cliente-Servidor
- Explorar más acerca de las herramientas que un sistema operativo puede ofrecer

Contexto

La arquitectura cliente-servidor es un modelo ampliamente utilizado en el que los dispositivos, denominados "clientes", realizan peticiones a un servidor centralizado para obtener servicios, recursos o datos. Esta relación se establece a través de una red de comunicación, como Internet, que permite la interconexión de dispositivos a escala global.

En el contexto de Internet, el cliente-servidor es fundamental para la mayoría de las interacciones en línea. Los navegadores web, por ejemplo, actúan como clientes al solicitar páginas web a servidores remotos. Los servidores, a su vez, responden proporcionando la información solicitada, como páginas web, archivos o datos.

Por esta razón, se decidió crear una simulación de pequeña escala de un modelo cliente-servidor usando el lenguaje de programación C++ implementado en Linux.

Cliente

Un cliente es cualquier dispositivo o aplicación que solicita algún servicio, recursos o datos a otro dispositivo centralizado llamado servidor. Por esto mismo, se creó un programa que simula un cliente el cual le solicita al servidor realizar la conversión de una cadena, una vez realizada esta petición, el servidor devuelve una respuesta, la cadena convertida en mayúsculas o minúsculas simulando que se realizó un algún tipo de servicio.

Habiendo obtenido una visión clara de lo que es un cliente, es crucial profundizar algunos detalles técnicos que permitan comprender en código el funcionamiento de un cliente.

Primero, hay que tener en cuenta que sólo existe un programa que crea un solo cliente, para simular que existen otros clientes tan solo basta ejecutar otra instancia del mismo programa, no hace falta cambiar nada dentro del código, todo esto es posible porque nos basamos en la idea que cada programa que se ejecuta en un

sistema operativo en realidad es un proceso, y todos los procesos tiene un identificador único, por ende, se creyó conveniente hacer uso de este identificador, justificando que en la vida real uno nunca sabe en la mayoría de veces cual es indicador que se asignó cuando se realiza alguna solicitud.

Tomando la idea anterior, se hizo lo mismo para los programas del Router y del Servidor; con ello se resuelve la tarea de estar asignando identificadores únicos.

Teniendo identificadores únicos para cada programa, entra en cuestión la forma de comunicar los datos de un programa a otro, ahora ya no es tan simple como pasar de un valor a otro como si fuera una variable, son programas distintos. La manera para resolver esto, es justamente como lo hacen los sistemas linux, usar archivos como memoria compartida. De esta manera, se puede establecer una conexión entre el programa cliente y el router.

Sin embargo, aún falta algo muy importante, por ahora, ya se cuenta con un identificador único para cada programa, ya hay una manera de compartir recursos, pero aún falta la forma para comunicar que un proceso ya pude hacer uso de ese recurso compartido, ya sea que lea el archivo o escriba en el, porque de lo contrario, ambos procesos entrarían en conflicto para realizar una simple tarea.
Aunque, hay más de una manera para realizar esta comunicación, por ejemplo, los sockets, se decidió implementar las señales como medio de comunicación entre dos programas. Al ser una pequeña simulación, las señales son suficientes para este modelo. Las señales no son otra cosa, que un programa que ya viene integrado en linux que permite comunicar información sustancial entre procesos, las señales empleadas para este programa son SIGTERM y SIGUSR1; para el cliente, se uso unicamente la señal SIGTERM, tanto para enviar un petición como para recibir una respuesta.

Cabe destacar, que para que siempre que se crea un cliente, y finaliza su servicio, es el momento cuando se eliminan los archivos que llevaron entablar dicha comunicación entre el cliente y el router. Otro punto clave, para realizar esta comunicación entre cliente y servidor es el intermediario, llamado Router.

Router

Una de las partes más importantes del proyecto, ha sido la de la creación del Router, a pesar de que no es completamente un router en nuestro proyecto, hemos decidido seguir los estándares usados en las redes para poder asimilar esta simulación más a la vida real, por lo que, la parte del proyecto que se encarga de comunicar a el cliente (o el proceso que envía el mensaje), con el servidor (o el proceso que recibe y modifica el mensaje), esto la hace mediante la metodología

NAT, la cuál permite comunicar varios clientes con un servidor dando a el servidor solo una dirección, la cuál pertenece al router, de esta manera, se logran crear varias direcciones dentro de un lugar aislado como una casa, sin embargo, todas las solicitudes se hacen a través de una sola dirección, que es la del router.

Siguiendo esta metodología, en la parte del router se propuso lograr este mismo objetivo para comunicar dos procesos, sin que el proceso que actuará como servidor se entere de ninguna información más que del mensaje del proceso que lo envía, para este propósito se han usado hilos, señales y el uso de memoria compartida.
El proceso del NAT inicia escribiendo su id, para después crear un hilo, el cuál inicia su vida esperando una señal de terminar que es enviada por el cliente, en el momento en el cuál se recibe esta señal, un manejador la captura para poder ejecutar las acciones correspondientes, ya que esta señal indica que un cliente le ha enviado una petición a el router. La función "`clientRequest`" funciona como capturadora de la señal, dados sus parámetros, nos permite conocer un poco de información sobre el proceso que envió la señal, lo que le interesa a la función es poder acceder al mensaje que se desea enviar, y para ello accede a la memoria compartida que se crea entre el proceso y el router, la cuál es mediante una archivo, recordando que este archivo tiene como nombre la PID del proceso que envía el mensaje, de esta manera, al utilizar el manejador, se puede conocer la PID del proceso remitente, y a su vez, el nombre del archivo para acceder a la memoria compartida. Con estos datos, el NAT creará una nueva área de memoria compartida, la cuál servirá como puente para comunicarse con el servidor, esto lo hace mediante la función "`setPortSignal`" la cuál crea un archivo con el nombre de un puerto que se le asignará por el sistema, normalmente dependiendo de la cantidad de solicitudes simultáneas que se tengan dependerá el nombre el cuál consta de la letra "P" + un número de puerto por el cuál se realizará la conexión con el servidor. En este archivo, se copia el mensaje exactamente como se tiene en la memoria compartida del cliente.

Posteriormente, si se logró crear el archivo correctamente y copiar el mensaje, se le mandará una señal al proceso que actúa como servidor para poder hacer una solicitud a el servidor, sin embargo, para lograr una conexión más eficiente, le envía dentro de la señal el puerto al que se conectará, de esta manera no se necesitan utilizar tablas para poder comparar la solicitud con el nombre del remitente, y a el servidor solo se le pasa la dirección del router, por lo que la comunicación se hace de manera efectiva con el servidor.

El servidor se deberá encargar de procesar la información y enviarla de regreso con una señal, (Se explica más adelante como lo hace) teniendo eso en cuenta, el router se queda esperando la señal de regreso del servidor, cuando la recibe, recuerda el puerto con el cuál se comunicó con el servidor, con toda esa información, en la

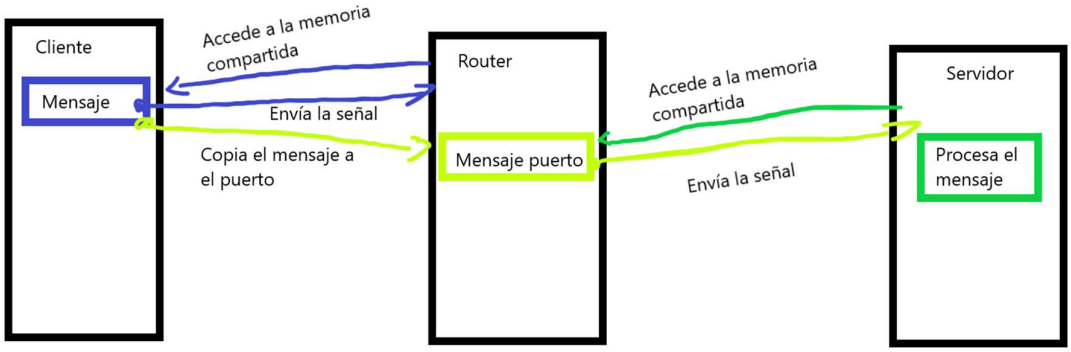
función “`sendResponse`” abre el archivo de memoria compartida entre el servidor y el router, para finalmente copiar la información a el otro archivo compartido que tenía con el cliente. Recordemos que el archivo de memoria compartida con el servidor lleva como nombre el puerto de conexión, mientras que el de la comunicación con el cliente, lleva el nombre del PID del proceso del cliente. El router logra saber qué memorias compartidas están relacionadas porque guarda las variables en el hilo de ejecución creado al inicio del router, el cuál excluye los demás hilos para poder atender solo esa petición, y al finalizar, se muere, sirviendo solo como un puente de comunicación entre las dos memorias compartidas para no perder su relación entre ellas.

Finalmente, se le envía al cliente una señal de terminado para poder indicar que ya se tiene la respuesta del servidor.

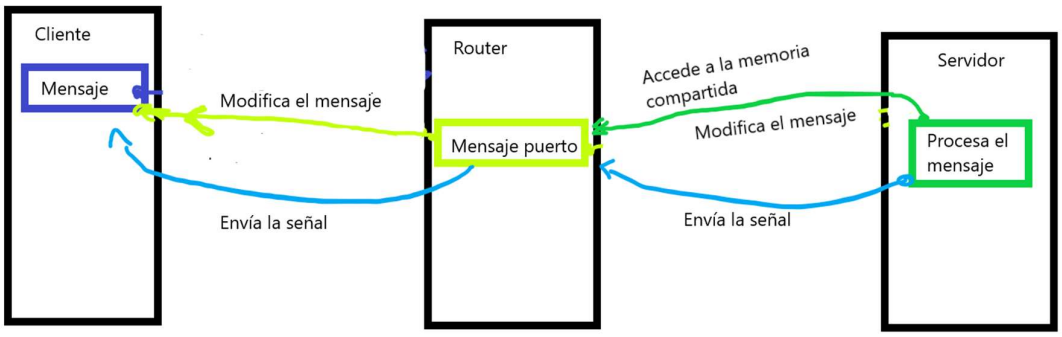
El uso de puertos en los routers NAT es muy importante, ya que de esa manera logran identificar qué solicitud se realizó con qué dirección y así cumplir con enviarle la información correcta a la dirección correcta usando su dirección propia del router para comunicarse con el servidor. Por lo que este simulador logra cumplir de manera similar a como lo haría un router en una red real.

Los hilos son necesarios, ya que normalmente en un router NAT se maneja múltiples solicitudes al mismo tiempo, y para manejar cada una de ellas, se utilizan hilos, sin embargo, para comunicar los procesos, por más rápidos que sean las personas, no pueden enviar múltiples peticiones al NAT al mismo tiempo, lo cuál termina sin tener demasiado sentido el uso de hilos, sin embargo, se han usado para preparar el sistema a una escalabilidad mayor o a una red más realista.

Finalmente, se han decidido agregar una diagramas muy sencillos del tipo sketch para explicar el envió de mensajes y su recepción:



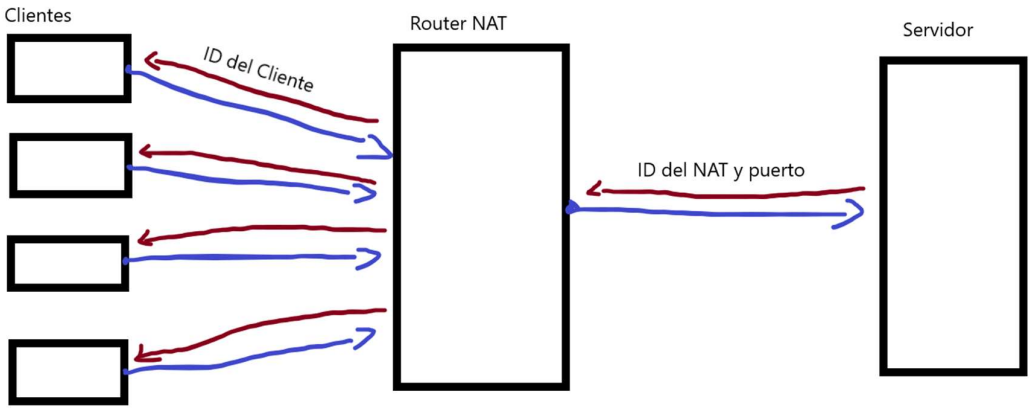
En este diagrama se muestra como una señal es enviada a el router, como este copia el mensaje a la memoria compartida de un puerto, y le envía este mensaje a el servidor por medio de una señal, finalmente el servidor procesa el mensaje y continúa con el regreso del mensaje:



En esta parte, solamente va modificando las partes y enviando una señal de que el mensaje original se ha modificado con el que el servidor proceso.

Con los diagramas antes propuestos se logra explicar de manera más simple lo que se explicó sobre el funcionamiento del NAT elaborado para el proyecto.

Finalmente se elaboró un diagrama para explicar el funcionamiento del NAT:



Con las descripciones dadas anteriormente, se logra entender que el NAT lo que hace es comunicar varios clientes con un servidor, cambiando sus identificadores de cada cliente por el del NAT mismo, junto con un puerto de conexión.

Servidor

El servidor es la parte que se encarga de convertir la cadena mandada por el cliente, pasando una cadena de caracteres a mayúsculas o a minúsculas, esto no se logra con una conexión directa con el cliente, sino con la conexión de Servidor-Router. Esto se hace a través de señales POSIX y archivos, los cuales proporcionan los ID necesarios y el número de puerto.

Para comenzar, primero se cuenta con el Main, en él, primero se obtiene el pid del proceso actual y se asigna al idServer, así, este id será utilizado como el id del servidor y es con el que el router podrá trabajar, y llama a la función setSignal. También llama la función getrid y guarda el valor en una variable, muestra el idServer y llama a la función server.

Ahora, también se encuentra la función “setSignal”, la cual abre un archivo llamado “sid”, si tiene éxito, en él guardará el idServer. Si no puede abrir el archivo, mandará un mensaje de error. También se tiene la función “getrid”, esta función intenta abrir un archivo llamado “rid”, si lo logra abrir, lee el archivo y devuelve el id, el cual es el id del router. Si no lo logra abrir, manda un mensaje de error.

Ahora, se tiene la función “showMessage”, la cual intenta abrir un archivo llamado “P” seguido por el número del puerto, si logra abrirlo, lee la cadena que se encuentra en el archivo. Si no lo consigue, entonces manda un mensaje de error.

Así mismo, se cuenta con la función llamada “getPortSignal”, esta función es la que se encarga de transformar la cadena. Primero va a leer la cadena que se encuentra en el archivo, y va a revisar letra por letra para cambiarla por mayúscula o minúscula, dependiendo de cómo se encuentre la cadena original, si está en mayúscula, pasará a minúscula y si está en minúscula, pasará a mayúscula. Si hay un error a la hora de abrir el archivo, entonces mandará un mensaje de error y regresará false, sino devolverá true.

Continuando, se tiene la función “routerRequest”, esta función es un manejador de señales, este manejador es utilizado con la forma extendida, esto permite enviar ciertas cosas que serían imposibles con una manera simplificada, como enviar el ID. En este caso espera la señal SIGTERM, cuando esta señal llega, imprime la información de router y del número de puerto. Luego llama a la función “getPortSignal” y envía la señal SIGURS1 al router con el mismo número de puerto.

Finalmente se encuentra la función “server”, esta función se encarga de configurar al manejador de señales “routerRequest” para la señal SIGTERM, después entra en un bucle infinito en el que se queda esperando por solicitudes del router. Esto permite que el servidor siempre esté activo y solo se apague cuando se solicite.

Video

https://youtu.be/kZ-BGE3YY28