Universidad de Los Andes
Facultad de Ingeniería y Ciencias Aplicadas
Ciencias de la Computación

Tarea 3

Sistemas Operativos y Redes

Emilio Vélez

“Some people goes to High School,

I go to School High”

Sir Calvin Cordozar Broadus Jr.

Índice

Descripción General:

El presente trabajo tiene por finalidad obtener un archivo desde la carpeta “myFiles” de algún cliente sin la necesidad que esté en el mismo directorio, ordenador, país, etc. y meterla a nuestra propia carpeta de archivos compartidos. O sea, mediante el uso de Internet. La complejidad del problema radica en los siguientes problemas:

- No sabemos quién o quienes tienen el archivo.
- No es la idea que el servidor guarde los archivos, debemos obtenerlos directamente de su dueño.
- El programa debe funcionar para varios clientes al mismo tiempo, deberemos usar threads para entregar un servicio personalizado, privado y en tiempo real, no por turnos ya que aplicaría en tiempos de espera muy altos para N grande usuarios.
- Se debe pautear muy bien el flujo de datos, ya que de nada nos sirve recibir información mezclada, ilógica, fuera de contexto etc.

Para enfrentar estos problemas, este programa funciona en base a un servidor con una jerarquía de conocimiento sobre la existencia de archivos, usuarios con senda dirección, con control sobre la cantidad de usuarios conectados máxima, la capacidad de desconectar a quien sea.

A diferencia del servidor, el cliente posee información limitada, solo conoce originalmente lo que existe en su propia carpeta de compartidos. Es por eso que mediante comandos de solicitud le informa mediante un canal propio (servidor-cliente único) que desea obtener información. El servidor, si es que está apropiadamente conectado con el usuario, procederá a responder o conversar por la misma vía mediante comandos o líneas de información conformadas por un header (primeros 3 o 4 chars) posee un flag definiendo el tipo de información que viene seguido de esta misma. El programa cuenta con un sistema de sockets que envían entre ellos mensajes TCP/IP, garantizando el orden de envío, pero no otros problemas surgidos y que explicare en respectivo tema.

Luego de el cliente solicitar información sobre los archivos disponibles, puede elegir entre los que concordaron con su búsqueda y con el comando -d descargar el archivo. Para dejar de lado la conexión con el servidor e iniciar una con el cliente o seed, el servidor como última acción le envía a cada uno la información necesaria del otro para abrir un nuevo socket mediante un nuevo thread, o sea funciona como un servicio de citas por internet.

Finalmente, el usuario seed, recibirá un comando de orden -u + información tal como ip, puerto arbitrario común para los dos, nombre del archivo requerido. Abrirá un

socket y se conectará al servidor momentáneo creado por el cliente solicitante, el cual solo permitirá una conexión en el puerto acordado por el servidor principal para los dos, siendo su ip publica el host.

Paquetes de Información:

Por default los sockets de Python envían y reciben mensajes bajo el protocolo TCP/IP. Gracias a esto nos aseguramos de que el orden de envío y recepción es el mismo. Ahorrándonos problemas como:

-Hola Juan ¿Cómo estás?

-Muy bien, ¿y tú?

-Hola Pedro

Sin embargo, para mantener el orden lógico de la conversación no nos sirve solo esta garantía. Se nos puede colar información y tener problemas como:

- ¿Por dónde has viajado?

-Miami

-Punta cana

-Italia

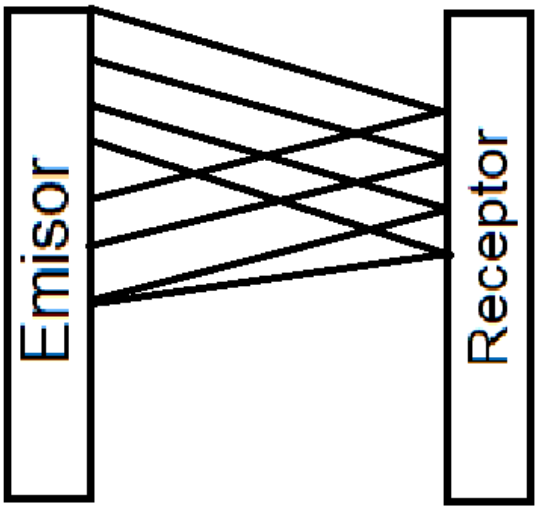
- Genial, ¿Cuál es el que más te gustó de esos?

-Paris

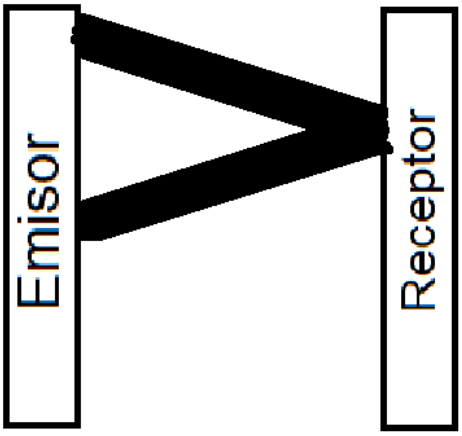
-LondresPuntaCana

Podemos ver que no esperó toda la respuesta y paso a la siguiente pregunta, también que el lector no alcanzo a despejar el buffer antes que llegara el siguiente paquete y mezclo información.

Para solucionar esto, transforme toda lista de datos a envía en solo uno el cual será transformado en lista nuevamente en el receptor. Funciona bajo una lógica de “si no lo dijiste todo en el instante después no puedes” y “pregunta única-respuesta única-pregunta única-respuesta única….”



En este caso dos paquetes llegan al mismo tiempo y el lector lo toma como uno solo, por lo tanto puedo recibir información como “-q Paris-u 2000 192.168.0.1-qLondres”, la cual no me sirve.

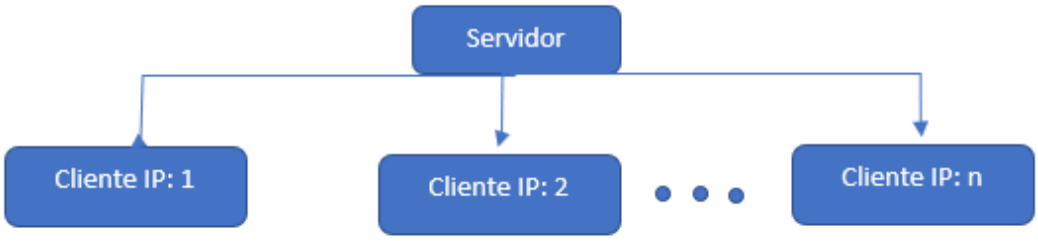


Así me aseguro de que nada pueda ponerse entremedio.

Mapa del Programa:

Solicitud y Respuesta búsqueda:

Búsqueda de archivos:



Descarga:

Ejemplo de descarga:

El cliente 2 tendrá un archivo llamado test.txt que el cliente 1 no tiene.

Pasos:

-Abrir Server.py [estado: running…]

-Conectar cliente1 (si busca ahora no encontrara el archivo dado que c2 no está disponible

-C1 recibe orden del servidor de mostrarle todos sus archivos compartidos.

-Conectar c2

-C2 recibe orden de mostrar sus archivos compartidos.

-C1 input: “-s test”

-Servidor recibe el input de C1 y le envía lista de los archivos relacionados. “-l lista de archivos”

-c1 elige cual quiere y ejecuta la orden de descarga: “-d 5” por ejemplo.

-servidor revisa en su mapa buscando los datos del dueño del archivo 5 o test.txt y le envía un comando de descarga con la información a c1. A la vez le envía a c2 una orden de ser seed “-u + la información de donde conectarse y que archivo mandar”. A ambos se les envía el mismo puerto para coordinar.

-c1 y c2 hacen un thread uno de recepción y el otro de envío.

-finaliza y c1 ya tiene test.txt en su carpeta myFiles