TP 3: Unix Sockets

Nicolas Mastropasqua

Programación sobre redes

November 2, 2020

1 Sockets Unix

1.1 Comunicación entre procesos

La comunicación entre procesos puede darse a través de distintos esquemas, como el intercambio de mensajes o bien el uso de memoria compartida.
Durante la primer parte de la materia exploramos la segunda estrategia utilizando threads que comparten memoria. Para esto, vimos que es necesario garantizar cierta coherencia en el acceso de los recursos compartidos. Más allá de poder aplicar algoritmos para sincronizar procesos, estudiamos distintas primitivas de sincronización, como los semáforos. Este último objeto permite lograr múltiples esquemas de sincronización, situación que se observó en los distintos ejercicios realizados.
A su vez, el esquema anterior supone una complejidad adicional importante producto de la necesidad de garantizar coherencia. La alternativa, el pasaje de mensajes, propone, en principio, una mayor simplicidad y la posibilidad de comunicar procesos alojados en **distintos hosts**. Una forma de conseguir esto es a través de la abstracción denominada **sockets**, que permite la comunicación bidireccional entre procesos. Dicho todo lo anterior, en este trabajo se propondrá el desarrollo de una aplicación que utilice sockets de UNIX [1] y que permita a varios clientes conectarse a un servidor de chat con algunas funcionalidades simples.

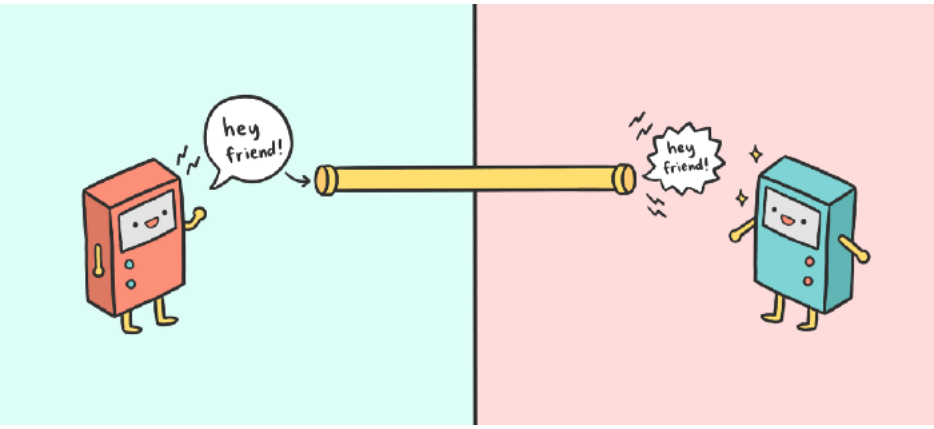

Figure 1: Source

2 Enunciado

2.1 Requisitos básicos

La aplicación que se pide desarrollar consta de un programa cliente y otro servidor que se comunican a través de *sockets* de UNIX en el dominio de Internet (alcanza con que funcione en una red local). Si bien en el trabajo habrá lugar para que cada grupo implemente las funcionalidades que desee, se pide cumplir con el siguiente conjunto de características básicas.

2.1.1 Enviar mensajes a participantes

El servidor debe soportar la interacción con n clientes de forma concurrente, siendo $n > 2$ un valor arbitrario elegido por el grupo. Luego, cada mensaje enviado por un participante debe poder ser leído por el resto.

OPCIONAL: Se podría implementar el envío de mensajes privados a otros usuarios que se encuentren online.

2.1.2 Login de usuarios

El servidor debe exigir a los clientes que se identifiquen con un *nickname* antes de poder hacer uso del mismo.No pueden existir dos clientes online con el mismo *nickname*.

2.1.3 Ejecución de comandos

El servidor permitirá que los clientes registrados exitosamente puedan ejecutar comandos especiales. Para lo anterior, el usuario indicará que entra al modo consola utilizando "/" seguido de algún comando reconocido por el servidor.
En particular, se pide implementar como mínimo el comando /list que le devuelve al cliente todos los participantes del chat.
Además de esto, cada grupo deberá proponer un comando o funcionalidad adicional para el chat.

2.2 Manejo de errores

Es importante considerar distintos casos que podrían causar problemas o sean indeseables, de manera que su aplicación sea lo más robusta posible. En este sentido, habría que tener en cuenta no solo posibles problemas de establecimiento de conexión o bien de desconexiones por parte de clientes, si no, por ejemplo, uso incorrecto de la aplicación (ingresar comandos inválidos, ingreso al chat con nickname invalido, etc).

Por otro lado, es muy probable que encuentren la necesidad de tener alguna estructura de datos, compartida por todos los threads del servidor, con información de cada cliente. Como vimos en la materia, esto podría traer situaciones indeseables como **race conditions** o **inconsistencias**. Para el caso de las inconsistencias, habría que identificar situaciones en donde distintos threads del servidor operen concurrente sobre la estructura compartida de clientes (por ejemplo, un thread eliminando un cliente y otro agregando uno).
Concretamente, se pide que el grupo argumente si es posible que existan inconsistencias o race conditions en su implementación. De ser así, se espera que puedan proponer una solución utilizando algún esquema de sincronización visto en clase. Se sugiere utilizar **pthreads mutex**[1] como primera opción, pero también pueden considerar **objetos atómicos** o bien **semáforos UNIX**.

3 Entrega

Para implementar los requisitos del sistema se debe utilizar C/C++. Para ello, están disponible los archivos entregados como base que podrían ser de ayuda como guía para la implementación.
Además de cumplir con los requisitos básicos planteados en el enunciado, se tendrá en cuenta las funcionalidades adicionales propuestas por el grupo, la legibilidad del código y el uso de recursos vistos en clase.

La entrega del tp deberá ser subida al *Classroom* en la tarea correspondiente. El formato de entrega será un archivo zip con el apellido de los integrantes del grupo, que no puede exceder las dos personas. La fecha limite propuesta para la entrega sera el **13 de Noviembre** hasta las 23:59:59. Cualquier trabajo entregado fuera del plazo se considera reentrega.

[1] `https://computing.llnl.gov/tutorials/pthreads/#Mutexes`

References

[1] Brian "Beej Jorgensen" Hall *Beej's Guide to Network Programming Using Internet Sockets.*