SISTEMAS OPERATIVOS

TRABAJO PRÁCTICO Nº 1
INTER-PROCESS COMMUNICATION

ALUMNOS

- 52056 – Juan Marcos Bellini
- 55821 – Diego María De Grimaudet De Rochebouët
- 55824 – Juan Li Puma
- 58241 – Ishbir Singh

TABLA DE CONTENIDOS

INTRODUCCIÓN

El presente informe trata acerca de la entrega preliminar del primer trabajo práctico de Sistemas Operativos. La misma consta de de la idea general del proyecto, el diseño conceptual, y la capa de comunicación entre el cliente y el servidor. El trabajo consiste en aprender a utilizar los distintos mecanismos de IPC (*inter-process communication*) presentes en un sistema POSIX.

Dentro de las características del proyecto se encuentran la existencia de más de un proceso cliente (todos ellos concurrentes), conectándose a un proceso servidor (que también es concurrente). La comunicación entre los clientes y el servidor es a través de una capa de comunicación con dos implementaciones posibles, *Named Pipes* (o *FIFOs*) y *Sockets TPC*. Además, el servidor se comunica con servicios de datos y de *logging* independientes.

A continuación se puede ver un breve esquema de la arquitectura general del *software*.

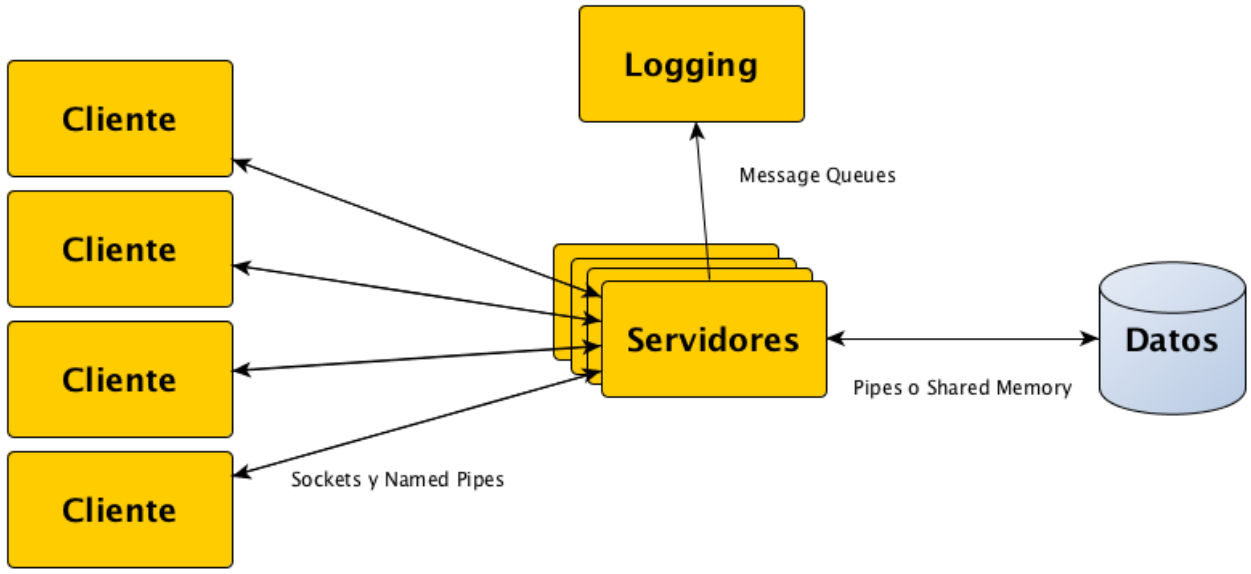

FIG. 1: ESQUEMA GENERAL DEL *SOFTWARE*

ALCANCE DEL PROYECTO

El *software* a desarrollar tiene la finalidad de crear una herramienta que pueda ayudar en la comercialización de bebidas alcohólicas en una modalidad de *delivery*. Los clientes de la empresa pueden utilizar la aplicación para consultar precios y/o *stock*, y realizar compras. A su vez, los dueños del comercio, pueden actualizar sus productos ofrecidos, sus precios, y sus depósitos, y consultar y actualizar los pedidos pendientes.

DISEÑO CONCEPTUAL

El *software* cuenta con distintas aplicaciones, cada una con un rol específico. Por un lado, se cuenta con el proceso cliente, el cual es utilizado por el usuario del sistema. A través de él, se pueden realizar consultas, o realizar otro tipo de acciones. Cuando se instancia un cliente (o sea, cuando se inicia dicho proceso), se realiza una conexión a un servidor “principal” que tiene la función de esperar conexiones entrantes. Cuando llega alguna, este instancia otro servidor, el cual tiene la función de comunicarse con el cliente que previamente se conecto al servidor principal. De esta manera, se logra la concurrencia de clientes. Cada uno tiene su servidor “dedicado”.

Por otro lado, los datos se almacenan en una base de datos SQLite. Este es otro proceso servidor independiente el cual tiene como función tanto almacenar datos en disco, como también ofrecer un servicio de consulta organizada. SQLite tiene como característica resolver la concurrencia por sí mismo. Las consultas (y modificaciones) a los datos son realizadas por cada servidor instancia (llamado servidor *forkeado*). Si el usuario desea, por ejemplo, consultar el listado de productos en venta, ingresa el comando correspondiente en la aplicación cliente, ésta realiza la petición a su servidor dedicado, y éste último consulta la base de datos. Luego SQLite responde (al servidor *forkeado*), y él le transmite la información al cliente (ver **Fig. 1**).

Finalmente, para que los distintos servidores puedan comunicar su estado al usuario, se cuenta con un servicio independiente de *logging*. A través de él, se puede conocer lo que va ocurriendo durante la ejecución del sistema.

CAPA DE COMUNICACIÓN

Existen distintas formas de comunicación en el *software*. Para empezar, la comunicación entre los servidores y los clientes son a través de *named pipes*, y a través de *sockets TCP*. La elección de cada método es en el momento de compilación/*linkedición* del sistema.

Un *named pipe* (o *FIFO*), a diferencia de un *pipe* común, vive en el *file system* de la computadora. Al momento de instalar el sistema, se genera un archivo de configuración en el cual se especifica la ruta de acceso del *FIFO*. Cuando el servidor principal inicia, crea dicho *named pipe* y comienza a escuchar a través de él, esperando conexiones entrantes. Cuando un cliente inicia, escribe en el *FIFO* en cuestión, el servidor principal se *forkea*, instanciando un servidor *forkeado*, y se crea un nuevo *named pipe*, a través del cuál se pueden comunicar el cliente y el nuevo servidor.

Por otro lado, si la forma de comunicación es través de *sockets TCP*, cuando el servidor principal inicia, este crea un *socket,* configurándolo de manera que pueda escuchar a través de todas las interfaces de red disponibles (*Ethernet*, *Wi-Fi*, *Loop Back*, *Bridge*, etc.), y espera a que lleguen conexiones entrantes. Cuando un cliente inicia, este crea su propio *socket* especificando la dirección del servidor (ya sea por linea de comandos o por archivos de configuración), y se conecta con él. En ese momento, al servidor le llega una conexión entrante, por lo que éste se *forkea*, y crea un servidor dedicado (con su propia conexión con el cliente) para que se puedan comunicar independientemente.

Respecto a la comunicación con la base de datos, se decidió hacer uso de *pipes* debido a que la implementación de este tipo de comunicación es más simple que a través de *shared memory*.

Finalmente, para poder *loggear*, existen una cola de mensajes (*message queue*) entre los servidores y el servicio de *logging*.

IMPLEMENTACIÓN

A continuación se puede ver la implementación de la capa de comunicación entre el cliente y el servidor, para *named pipes*, y para *sockets TCP*