Programación de Sockets

Informe del Trabajo Práctico Final

Sistemas Operativos

Integrantes:

Martin Victory 56086,
Florencia Cavallin 56015
Martín Grabina 57360

Entrega: Martes 6 de febrero de 2018
Coloquio: Miércoles 7 de febrero de 2018

Introducción y objetivos

En el siguiente informe se detallan todos los aspectos tenidos en cuenta durante el desarrollo del trabajo práctico de sockets en cuestión.
El mismo tuvo una duración aproximada de dos semanas de desarrollo y se realizó grupalmente, dividiendo tareas equitativamente, mediante Git y algunas practicas de programacion consensuadas.
Los objetivos de este trabajo son:

- Adquirir los conocimientos básicos necesarios para poder utilizar primitivas de sincronización y comunicación de procesos en ANSI C.
- Conocer y comprender las distintas formas de comunicación entre procesos mediante sockets.
- Generar una aplicación cliente-servidor que atienda clientes en simultáneo garantizando la integridad de los datos.

Instalación

Se detallan los pasos a seguir para instalar el sistema utilizando la misma computadora que albergue el servidor y los clientes. Se puede ejecutar de manera remota tomando los recaudos suficientes. Es por esto que se utiliza el nombre de host “localhost” y se utiliza arbitrariamente el puerto 8000 (se podria utilizar otro).

1. Descargar la carpeta del sistema y ubicarse en la misma
2. Ejecutar el comando *‘make all’* para compilar todos los archivos necesarios e instalar la base de datos. Se recomienda ejecutar previamente un *‘make clean’* para borrar archivos precompilados que puedan traer problemas de versiones anteriores.
3. Ejecutar *‘./server 8000’* esto iniciará el servidor en el puerto 8000.
4. En otra pestaña o ventana de la terminal ejecutar *‘./client localhost 8000’* esto conectará al terminal cliente con el servidor en puerto 8000 bajo el nombre de localhost.
5. Utilizar el sistema libremente siguiendo los pasos que se indican.

Problemas durante el desarrollo

A medida que el sistema crecía se procedió a probar todo el sistema en conjunto, con voluntarios que nunca lo habian probado, con el fin de encontrar errores de todo tipo, inclusive de experiencia de usuario.
En un principio para lograr la concurrencia empezamos haciendo procesos paralelos, pero despues debimos modificarlo a threads para poder sincronizarlos de manera más fácil.
A su vez, la cantidad de archivos a compilar e instalar era un problema ya que muchas veces se olvidaba de ejecutar o limpiar alguno y generaba problemas de lógica, lo cual se soluciono con un makefile correspondiente con divisiones que permitan ejecutar distintos

comandos según necesidad, por que por ejemplo el servidor tarda en compilar casi el doble que el cliente, entonces si se modifica algo del cliente no es necesario volver a compilar el servidor y perder el tiempo.
Dedicamos tambien los ultimos momentos de desarrollo para testeo profundo y hacer un sistema robusto que valide los datos y evite problemas inesperados.

Funcionamiento del servidor

Para cumplir los objetivos del trabajo, el funcionamiento del servidor debía estar totalmente desacoplado de los clientes, siendo estos procesos diferentes. Por lo tanto la única comunicación que establecen es mediante el uso de sockets.

Primero se debe ejecutar el servidor quien se configura como tal abriendo un socket y queda a la espera de los procesos clientes. Cuando se ejecuta un cliente, el servidor lo acepta y abre un nuevo *thread* el cual se ocupara de manejar todas las consultas que el cliente realiza. Al salir el cliente del sistema, el thread termina. Mientras tanto, el *thread* principal del servidor sigue a la espera de nuevos clientes y por cada uno realiza el mismo procedimiento. De esta manera, obtenemos nuestro servidor concurrente.

Para lograr una correcta comunicación entre cliente-servidor, se definió un protocolo entre ambos con funciones limitadas y específicas. Dada la configuración de los sockets elegida, este protocolo está basado en la comunicación TCP donde el cliente realiza un pedido al servidor y luego el servidor le responde si fue exitosa la operación o no.

A la hora del manejo de la base de datos, se decidió tener una única instancia de esta abierta en el servidor. Se inicializa previo a recibir a los clientes, y se almacena de manera global para que luego cada thread encargado de manejar un cliente tenga acceso a la misma. Debido a esta solución surge un problema: ¿que pasa si dos clientes quieren acceder a la base de datos en el mismo instante? Para resolver esto se implementó un sistema de mutex, donde el uso de la base de datos es considerado zona crítica y por lo tanto solo un thread puede ejecutar sus funciones a la vez.

Finalmente, para darle mas funcionamiento al proceso servidor, y no quedar únicamente a la espera de nuevos clientes, se decidió agregarle una pequeña interfaz en la cual se le pueden pedir comandos como: cantidad de usuarios conectados, o cerrar el servidor. Para esto al iniciar el servidor se crea un nuevo *thread* el cual se encarga de recibir comandos y brindar la información. Una vez ejecutando el servidor se pueden escribir los siguientes comandos: help, users, quit.

Funcionamiento del cliente

El funcionamiento del cliente es simple, al iniciar establece la comunicación con el servidor por medio de un socket, y luego ejecuta su interfaz gráfica. Por cada comando ejecutado por el usuario, el proceso cliente se realiza una petición al servidor, estas estan especificadas por el mismo protocolo descrito en el servidor.

Decisiones de diseño

Se decidio dedicarle tiempo al diseño de la experiencia del usuario, teniendo en cuenta que es un sistema de reserva de vuelos, deberá poder utilizarse fácilmente por cualquier persona. Entonces se decidió aprovechar el esquema mental del usuario tipo, utilizando carteles con distintos colores asociados dependiendo de su significado (Ej. Errores color rojo).

Decidimos que todos los asientos sean de la misma categoría y tengan el mismo precio, simulando una línea aérea low-cost, para facilitar cuestiones en todos los ámbitos del desarrollo (desde base de datos hasta interfaz de usuario) y de la misma manera, todos los aviones tienen la misma configuración de asientos arbitrariamente 6 asientos con pasillo por fila y los mismos son numerados sin letras.

Base de datos

Para la base de datos utilizamos la librería sqlite version 3, con las siguientes configuraciones de tablas.

Vuelo

- Numero de vuelo, entero
- Origen, cadena de caracteres
- Destino, cadena de caracteres
- Precio, entero
- Cantidad de asientos, entero
- Fecha, cadena de caracteres

Reservación

- Número de reserva, entero
- Numero de vuelo asociado, entero
- Nombre, cadena de caracteres
- Estado de la reserva, cadena de caracteres
- Número de asiento, entero

Asiento

- Numero de vuelo asociado, entero
- Número de asiento, entero

Conclusión

Llegamos a la conclusión que, como siempre, con mas tiempo y dedicación este tipo de herramientas no tienen limite y se pueden potenciar aún más de lo que lo utilizamos en este proyecto, sin embargo esto funciono exitosamente para cumplir los objetivos propuestos anteriormente en relativamente poco tiempo, de manera organizada y libre.

Pudimos investigar de manera individual y discutir grupalmente el funcionamiento de los sockets en un sistema de tipo cliente-servidor, entender mejor cómo funcionan muchos de

los sistemas que utilizamos dia a dia y cubrir nuestros déficits de conocimientos específicos requeridos para Protocolos de Comunicación.

Bibliografía utilizada

- Wikipedia
- OSDev
- StackOverflow
- Manual de linux