**Trabajo Práctico Final Sistemas Operativos**

**SOCKETS**

Grupo integrado por:

* Cortés Rodríguez, Kevin Imanol
* Capparelli, Martín Federico

Fecha de entrega: Martes 11 de Julio, 2017

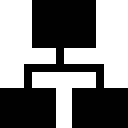
Fecha de Exposición oral: Miércoles 12 de Julio, 2017

# Objetivos

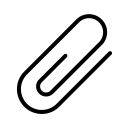
El objetivo de este trabajo consistió en la implementación de un cliente y servidor para atender a varios clientes en simultáneo mediante el uso de sockets.

# Contenido

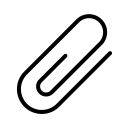
El siguiente trabajo práctico cuenta con los siguientes archivos.



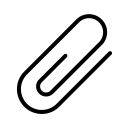
datastructure.h



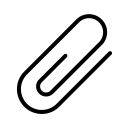
UDPClient.c



UDPClientMaster.c

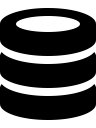


UDPServer.c



dbmanager.c

so.db



# Pasos para la ejecución

1. Situarse en la carpeta en particular (SOckets)
2. Abrir consola y hacer “make”
3. Ejecutar ./udpServer 23000
4. En otra consola, situarse en la carpeta SOckets y hacer ./udpClient 23000
5. En otra consola, situarse en la carpeta SOckets y hacer ./udpClientMaster 23000

# Entregables

* **UDPClient**: Lado del cliente normal. Se pueden hacer 4 tipos de consultas básicas (reservar asiento de un vuelo, cancelar la reserva de un vuelo, ver los ocupados de un vuelo y ver los vuelos disponibles). Envía un mensaje al servidor del tipo “client message”.
* **UDPServer**: Recibe todas las consultas del cliente, y se comunica con el manejador de base de datos. La información recibida de la base de datos es del tipo DBStructure, por lo tanto, luego hay que hacer un parseo al tipo de dato correcto (server message). Devuelve la información al cliente con el tipo “server message”.
* **UDPClientMaster**: Lado del cliente master. Se pueden hacer consultas, altas y bajas de vuelos y usuarios. Envía un mensaje al servidor del tipo “client message”.
* **so.db**: Contiene toda la información relativa a las tablas
* **dbmanager**: Es llamado por el servidor y procesa la consulta en conjunto con la base de datos (almacenada en so.db). La información la devuelve al servidor con un tipo de dato especial (DBStructure).
* **dataestructure**: Contiene la estructura relativa al envío y recepción de mensajes entre los dos tipos de clientes y servidores

# Introducción

Se ha diseñado una plataforma cliente-servidor utilizando sockets y la herramienta sqlite. La plataforma es de reserva de asientos de avión, donde el usuario no solo puede reservar asientos, sino que requiere de un registro previo.

Existen dos tipos de clientes: el cliente común y el cliente master. El común hace reservas, se puede registrar y cancelar vuelos. El cliente master tiene un poco de mayor administración de la base de dato, haciendo altas y bajas de vuelos y usuarios.

Con respecto a la base de datos, hemos tratado de mantenerlo sencillo, adaptándonos a las consignas del enunciado. Sin embargo, el código está diseñado para que se puedan hacer otras consultas; es decir, el código implementado no está acoplado al 100% con las consultas que se verán en este trabajo práctico.

El trabajo ha sido dividido en 3 secciones. En una primera instancia, se diseñó e implementó toda la base de datos y se diseñó en un archivo .c aparte todas las funciones que luego serían consultadas por los clientes.

En una segunda instancia, diseñamos toda la comunicación con sockets. Además, hicimos un trabajo de investigación en paralelo con respecto a los protocolos de comunicación que se podrían utilizar para la comunicación cliente-servidor.

Por último, hicimos un merge de la base de datos con el servidor y el cliente.

**Aclaración:** En este informe no haremos tanto enfoque en las librerías utilizadas para la implementación de sockets, sino que detallaremos la idea detrás del código implementado.

# Base de datos.

El archivo adjunto en el trabajo práctico **tables.sql** describe como están construidas las tablas. Existen 4 tablas:

1. **Asientos**: Describe todos los asientos que están ocupados. Una reserva está constituida principalmente por la identificación del usuario al que está reservado el asiento, la fecha del vuelo, el número de vuelo, entre otros. En el caso de querer hacer una reserva en el cuál no existe el vuelo, o el usuario, no se podrá realizar la reserva. El usuario en ese caso será notificado mediante una notificación por consola.
2. **Cancelados**: Esta tabla es cargada cuando se cancela alguna de las reservas. Esta tabla es consultada por el cliente master en el caso de querer tener un registro previo de todas las cancelaciones realizadas hasta el día de la fecha
3. **Cliente**: Todo usuario que quiera hacer una reserva necesita que esté registrado en la base de datos. El cliente tiene una identificación única que puede ser el documento nacional de identidad.
4. **Vuelos**: Todos los vuelos tienen un origen y un destino. Además, tienen un código identificatorio y el nombre de la aerolínea que lleva a cabo el vuelo

# Comunicación Cliente-Servidor

## Principios básicos

La comunicación entre el cliente y el servidor se da mediante el protocolo de comunicación UDP (user datagram protocol). Hemos utilizado este protocolo ya que, para dicho protocolo alcanzaba para mostrar la implementación de sockets. Sin embargo, podemos tranquilamente cambiar el protocolo de comunicación fácilmente indicándolo en **UDPClient**, **UDPClientMaster** y **UDPServer**. Dicho protocolo trabaja con paquetes enteros, no con bytes individuales como TCP. Una aplicación que emplea el protocolo UDP intercambia información en forma de bloques de bytes, de forma que, por cada bloque de bytes enviado de la capa de aplicación a la capa de transporte, se envía un paquete UDP. Tampoco emplea control del flujo ni ordena los paquetes, aunque la gran ventaja es que provoca poca carga adicional en la red ya que es sencillo y emplea cabeceras muy simples.

## ¿Cómo se da la comunicación?

Al ejecutar el programa (tanto cliente como servidor), hay que indicar en que puerto hay que conectarse. Por default, hemos establecido el puerto 23000 (aunque puede ser otro, claramente).