Trabajo Práctico Final.

SOckets

Kevin Cortes – Martin Capparelli

Sistemas Operativos – 1er. Cuatrimestre 2017

ITBA – Ing. Informática

SOckets

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

Contenidos

SOckets

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.


UDPClient.c


UDPServer.c


UDPClientMaster.c


so.db


dbmanager.c


datastructure.h

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

Trabajo

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.

Introducción

Un socket es una manera de hablar con otra computadora. Para ser más preciso, es una manera de hablar con otras computadoras usando descriptores de ficheros estándar. Todas las acciones de entrada y salida son desempeñadas escribiendo o leyendo en uno de estos descriptores de fichero, los cuales son simplemente un número entero, asociado a un fichero abierto que puede ser una conexión de red, un terminal, o cualquier otra cosa.

Ahora bien, sobre los diferentes tipos de sockets en Internet, hay muchos tipos pero sólo se describirán dos de ellos:

1. Sockets de Flujo (SOCK_STREAM): Están libres de errores: Si por ejemplo, enviáramos por el socket de flujo tres objetos "A, B, C", llegarán al destino en el mismo orden -- "A, B, C". Estos sockets usan TCP y es este protocolo el que nos asegura el orden de los objetos durante la transmisión.
2. Sockets de Datagramas (SOCK_DGRAM): Es el que utilizaremos nosotros en este trabajo práctico. Éstos usan UD, y no necesitan de una conexión accesible como los Sockets de Flujo.

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

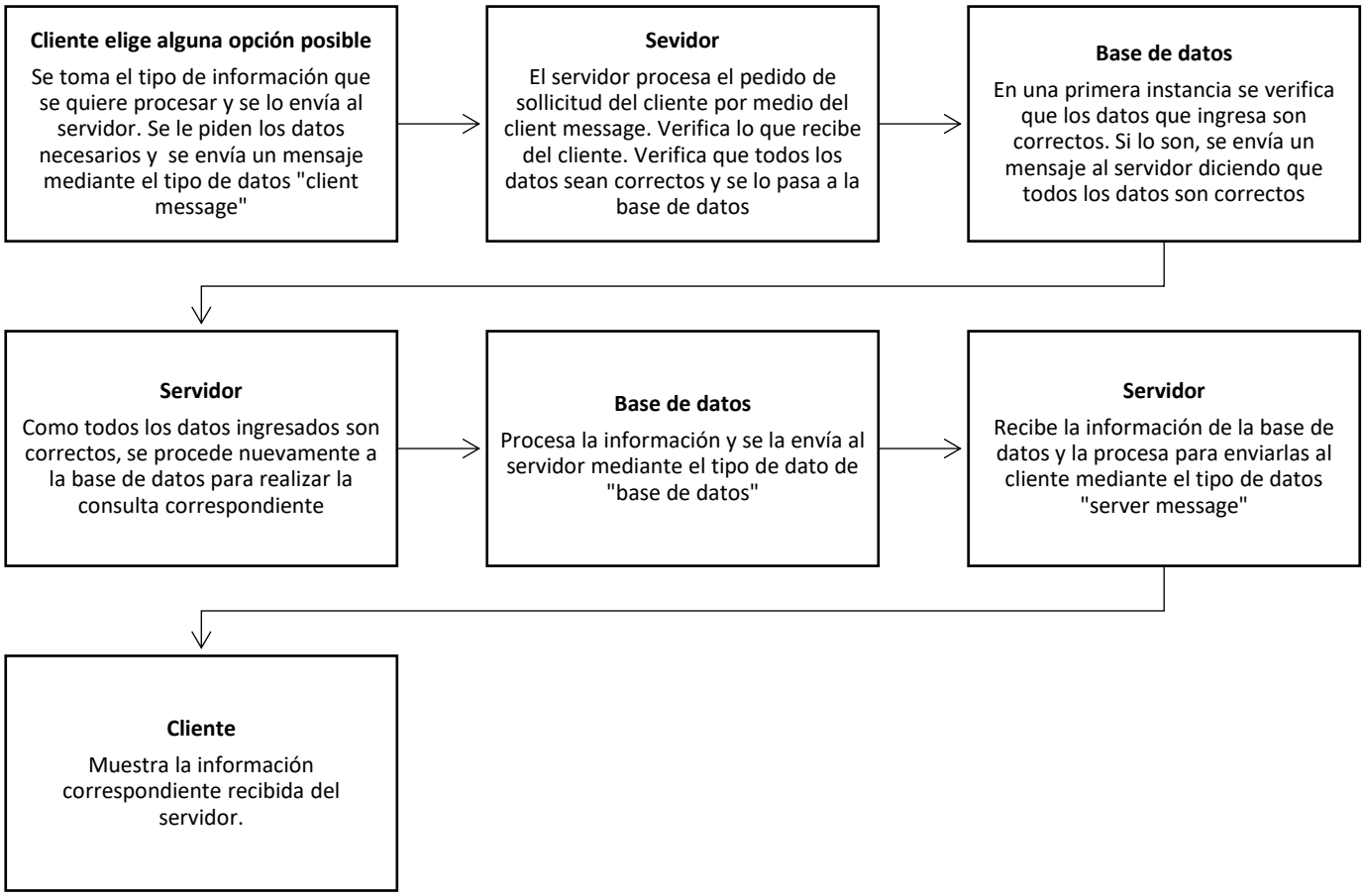
Comunicación Cliente-Servidor

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).



Implementación

A continuación, daremos una breve explicación de las librerías utilizadas. Trataremos de no ahondar tanto en este tema ya que pertenece a librerías ajenas a nuestro TP (sin embargo, consideramos necesario explicar brevemente cómo ha sido utilizado).

Estructura sockaddr_in

```
#include <netinet/in.h>

struct sockaddr_in {
    short            sin_family;   // e.g. AF_INET
    unsigned short   sin_port;     // e.g. htons(3490)
    struct in_addr   sin_addr;     // see struct in_addr, below
    char             sin_zero[8];  // zero this if you want to
};

struct in_addr {
    unsigned long s_addr;  // load with inet_aton()
};
```

- **Sin_family**: Familia de la dirección
- **Sin_port**: Puerto
- **Sin_addr**: Dirección de internet
- **Sin_zero**: Del mismo tamaño que el struct sockaddr. Puede ser configurada con ceros usando las funciones memset() o bzero()

Funciones inet_addr y htons

```
udpServerAddr->sin_addr.s_addr = inet_addr(serverIP);
udpServerAddr->sin_port = htons(serverPort);
```

Por un lado, la función inet_addr() convierte una dirección IP en un entero largo sin signo (unsigned long int).

Es un tipo de conversión de nodo a variable larga de Red

Funcion socket(…)

```
#include <sys/types.h>
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
```

- **Domain:** Se podrá establecer como `AF_INET` o `AF_UNIX` (si se desea crear sockets para la comunicación interna del sistema). Éstas son las más usadas, pero no las únicas.

- **Type:** Aquí se debe especificar la clase de socket que queremos usar (de Flujos o de Datagramas). Las variables que deben aparecer con `SOCK_STREAM` o `SOCK_DGRAM` según queramos usar sockets de Flujo o de Datagramas, respectivamente.
- **Protocol.**

La función `socket()` nos devuelve un descriptor de socket, el cual podremos usar luego para llamadas al sistema. Si nos devuelve *-1*, se ha producido un error (obsérvese que esto puede resultar útil para rutinas de verificación de errores

Función bind()

```
#include <sys/types.h>
#include <sys/socket.h>

int bind(int fd, struct sockaddr *my_addr,int addrlen);
```

- **fd.** Es el descriptor de fichero socket devuelto por la llamada a `socket()`.
- **my_addr.** es un puntero a una estructura sockaddr
- **addrlen.** contiene la longitud de la estructura sockaddr a la cuál apunta el puntero `my_addr`. Se debería establecer como `sizeof(struct sockaddr)`.

La llamada bind() se usa cuando los puertos locales de nuestra máquina están en nuestros planes (usualmente cuando utilizamos la llamada listen()). Su función esencial es asociar un socket con un puerto (de nuestra máquina). Análogamente socket(), devolverá -1 en caso de error.

Un aspecto importante sobre los puertos y la llamada bind() es que todos los puertos menores que 1024 están reservados. Se podrá establecer un puerto, siempre que esté entre 1024 y 65535 (y siempre que no estén siendo usados por otros programas).

Función connect()

```
#include <sys/types.h>
#include <sys/socket.h>

int connect(int fd, struct sockaddr *serv_addr, int addrlen);
```

- **fd.** Debería configurarse como el fichero descriptor del socket, el cuál fue devuelto por la llamada a `socket()`.
- **serv_addr.** Es un puntero a la estructura sockaddr la cuál contiene la dirección IP destino y el puerto.
- **addrlen.** Análogamente de lo que pasaba con `bind()`, este argumento debería establecerse como `sizeof(struct sockaddr)`.

La función `connect()` se usa para conectarse a un puerto definido en una dirección IP. Devolverá -1 si ocurre algún error.

Bibliografía

1. https://www.gnu.org/software/libc/manual/html_node/Inet-Example.html
2. https://cs.uns.edu.ar/~ldm/mypage/data/rc/apuntes/introduccion_al_uso_de_sockets.pdf
3. http://www.thegeekstuff.com/2011/12/c-socket-programming/
4. http://es.tldp.org/Tutoriales/PROG-SOCKETS/prog-sockets.html