DeBondis.com

Ejercicio N°3

Objetivos	• Diseño y construcción de sistemas con acceso distribuido • Encapsulación de Threads y Sockets en Clases • Definición de protocolos de comunicación • Protección de los recursos compartidos • Uso de buenas prácticas de programación en C++
Instancias de Entrega	**Entrega 1:** clase 8 (04/10/2016). **Entrega 2:** clase 10 (18/10/2016).
Temas de Repaso	• Definición de clases en C++ • Contenedores de STL • Excepciones / RAII • Move Semantics • Sockets • Threads
Criterios de Evaluación	• Criterios de ejercicios anteriores • Eficiencia del protocolo de comunicaciones definido • Control de paquetes completos en el envío y recepción por Sockets • Atención de varios clientes de forma simultánea • Eliminación de clientes desconectados de forma controlada

Índice

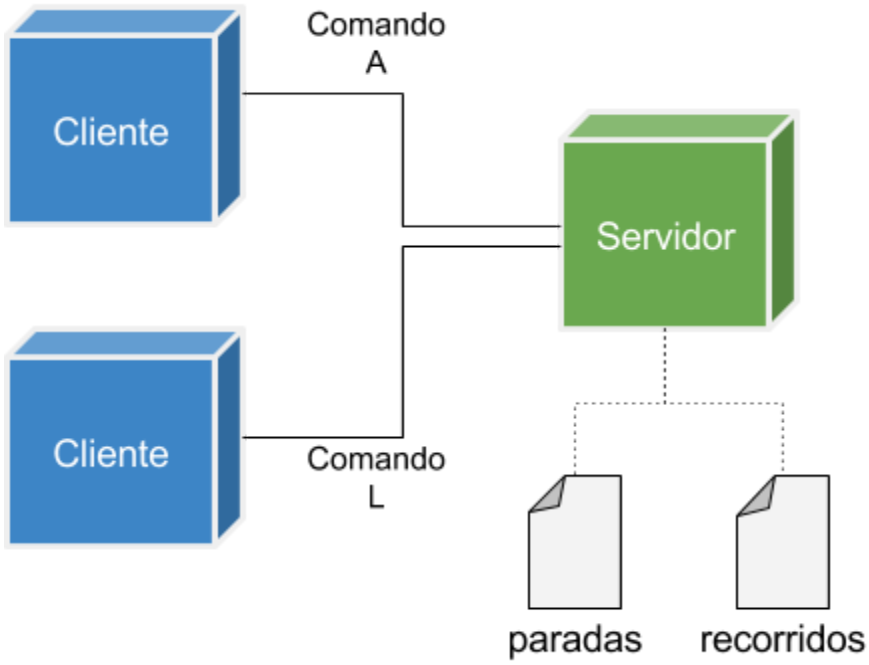
Introducción

El gobierno de una muy importante ciudad reconoció, a partir de infinitas quejas de los contribuyentes, que el tránsito es un caos. Con intenciones de calmar las quejas (ya que resolver el problema del tránsito requiere de una empresa mucho mayor) ha decidido contratarnos para desarrollar una aplicación. Luego de varias reuniones, y sin estar claros los objetivos del sistema, se decidió realizar una propuesta innovadora que fue presentada y aprobada por unanimidad. Se propone desarrollar una aplicación que permita a los usuarios de colectivos de la ciudad decidir qué línea utilizar a partir de algunas consultas online.

Descripción

Nuestro sistema constará de dos aplicaciones: un cliente y un servidor. El servidor tomará información de las paradas disponibles y recorridos de colectivos desde archivos que él mismo posee. Luego, atenderá sucesivos clientes que podrán interactuar mediante comandos.
Los clientes son ejecutados de forma remota y, mediante los comandos acceden al servidor para indicar la posición de los colectivos de una línea o consultar por aquellos colectivos más convenientes de acuerdo con el horario.
Cada cliente recibe información de entrada estándar y envía mensajes al servidor en formato binario, como se detalla en las secciones siguientes.
Las respuestas del servidor, serán también en formato binario y el cliente deberá imprimir por consola notificaciones al usuario con del resultado del comando. El formato de estas notificaciones también se detallará más adelante.

A continuación se incluye un pequeño diagrama de la estructura pedida:



Formato de Línea de Comandos

Servidor

El servidor recibirá como parámetros:

1. El número de puerto en el que escuchará conexiones;
2. El nombre de un archivo que contendrá identificadores de todas las paradas de colectivos de la ciudad, y las distancias entre las mismas. A modo de simplificación, las distancias serán expresadas en términos del tiempo promedio que le toma a un colectivo llegar desde una parada a la otra. Se asumirá que todos los colectivos demoran siempre la misma cantidad de tiempo establecida.
3. El nombre de un archivo que contendrá los recorridos de todas las líneas de colectivos participantes del sistema.

Por ejemplo:

```
./server 1234 paradas.txt recorridos.txt
```

Cliente

Las aplicaciones cliente, recibirán:

1. La dirección IP del servidor.
2. El número de puerto del servidor al que deben conectarse.

Por ejemplo:

```
./client 172.217.29.14 443
```

Códigos de Retorno

En cualquier caso, tanto servidor como cliente deben retornar 0.

Entradas y Salidas

Servidor

El servidor deberá esperar hasta leer un caracter ‘q’ por entrada estándar. Cuando lo recibe debe terminar ordenadamente. Mientras está esperando el caracter ‘q’ recibirá conexiones de distintos clientes, que deberá atender en paralelo, contestando lo que se describe en cada comando a continuación .
No se imprime nada por salida estándar en el servidor.
Mediante salida de error, se imprimen los siguientes mensajes de información:

- Al recibir una nueva conexión:
  - `Cliente conectado.\n`
- Al recibir un comando desde el cliente:
  - `Comando <ID-comando> recibido.\n`
- Al detectar una nueva conexión:
  - `Cliente desconectado.\n`

Archivos de entrada del servidor

Se utilizan dos archivos de texto plano para definir las paradas y recorridos mediante su lectura línea a línea.

Archivo de paradas:

El servidor recibe como segundo parámetro un archivo de texto con las paradas que hay en la ciudad, y el tiempo que tardan los colectivos en ir de una parada a otra. En cada registro del archivo de paradas se indican los identificadores de dos paradas y los segundos que se tarda en ir de una a otra, como tres **enteros** separados por espacios:

```
<id-parada-1> <id-parada-2> <segundos-de-1-a-2>\n
```

Archivo de recorridos de líneas de colectivo:

El tercer parámetro que recibe el servidor es el nombre de otro archivo de texto con los recorridos de las líneas de colectivo que se registraron en el sistema. En cada registro del archivo de recorridos se indica la línea de colectivos seguida de una lista de identificadores de paradas, todos los datos como **enteros** separados por espacios:

```
<línea-de-colectivos> <id-parada-1> <id-parada-2> … <id-parada-n>\n
```

Cliente

La aplicación cliente utilizará la entrada estándar para recibir los distintos comandos que serán transmitidos al servidor.
Todo comando recibido por entrada estándar posee la forma:

```
[dd/mm/yyyy-hh:mm:ss] <ID-comando> <parámetro-1> <parámetro-2> …\n
```

La cantidad de argumentos para cada comando es variable pero se puede inferir en la siguiente sección. Los registros se leen línea a línea, siendo cada comando procesado antes de avanzar con la siguiente. Las respuestas del servidor deben ser esperadas e impresas por salida estándar con los siguientes mensajes:

- Comando A
  - `Un colectivo de la línea <ID-línea> ha sido agregado.\n`
- Comando F
  - `Faltan <cant-minutos> minutos para que llegue el siguiente colectivo.\n`
- Comando L
  - `La línea <ID-línea> tardará <cant-minutos> minutos y <cant-segundos> segundos en llegar a destino.\n`
- Comando R
  - `El colectivo de la línea <ID-línea> tardará <cant-minutos> minutos y <cant-segundos> segundos en llegar a destino.\n`

Frente a todos los comandos, se puede recibir un mensaje de error por parte del servidor. Si esto ocurre se debe imprimir:

```
Error.\n
```

Luego de procesar todos los datos, la aplicación debe terminar ordenadamente su ejecución.

Comunicación Cliente-Servidor

La comunicación cliente-servidor se establece mediante un protocolo binario de paquetes. Los paquetes tendrán un tipo que se puede identificar mediante el primer byte. Luego, dependiendo del tipo, se puede determinar la cantidad de argumentos del paquete con sus respectivos tamaños para continuar con el parseo, procesamiento y despacho de respuesta.
Todo mensaje del cliente, posee una respuesta del servidor que respetan la estructura de paquetes.

Cliente

Existen dos modos para la aplicación cliente: Empresa y Pasajero. Ambos poseen comandos bien diferenciados cuyos IDs y parámetros se detallan a continuación.

> **Nota:**
> Los modos descriptos son meramente indicativos ya que la aplicación cliente no requiere ningún parámetro para entrar en cada modo ni hace distinción alguna respecto de los comandos permitidos.

Modo Empresa

Comando 'A' - Inicio de Recorrido de Colectivo

Las empresas le avisan al servidor cada vez que un colectivo sale de la terminal (primera parada del recorrido). Para ello reciben por entrada estándar el horario actual (este horario NO se debe calcular en la aplicación), y lo envían al servidor, junto con el identificador del comando (un caracter 'A') y la línea de

colectivos según:

```
[dd/mm/yyyy-hh:mm:ss] A <línea>
```

Ejemplo de línea a leer por entrada estándar:

```
[20/09/2016-18:00:00] A 33
```

La aplicación deberá traducir la fecha a formato de tiempo UNIX, y enviarla junto con el identificador del comando y el número de línea de colectivos al servidor en BigEndian. Tanto el timestamp como el número de línea serán enteros sin signo de 32 bits, mientras que el caracter ocupa 1 Byte. La distribución de campos en el paquete es la siguiente:

'A' (1 byte)	Tiempo-unix (4 bytes - BigEndian)	Linea (4 bytes - BigEndian)

Esto es, los datos del ejemplo anterior se traducirán en el siguiente paquete (en hexadecimal) que serán enviados al servidor:

0x41 (1 byte)	0x57E17920 (4 bytes - BigEndian)	0x00000021 (4 bytes - BigEndian)

El servidor responde oportunamente al comando, indicando que el pedido fue procesado.

Modo Pasajero

Comando 'F' - Tiempo Faltante

a. Recibe de entrada estándar un timestamp, el identificador del comando, la línea de colectivo sobre la que está consultando el pasajero, y la parada en la que se encuentra el mismo. Por ejemplo:

```
[20/09/2016-18:00:00] F 33 14
```

b. El cliente deberá convertir la fecha a tiempo UNIX y enviarla junto con los demás campos, todos como enteros no signados de 32 bits (uint32_t) cada uno (salvo el caracter indicador del comando, que se enviará como un solo byte). El paquete final del ejemplo anterior sería:

0x46 (1 byte)	0x57E17920 (4 bytes - BigEndian)	0x00000021 (4 bytes - BigEndian)	0x0000000E (4 bytes - BigEndian)

El servidor le deberá contestar con un mensaje que diga cuánto falta para que llegue el colectivo que está esperando.

Comando 'L' - Línea Más Rápida

1. Indica qué colectivo debería tomar para llegar más rápido a destino.
2. La aplicación leerá el timestamp, el identificador del comando, el identificador de la parada donde se encuentra el pasajero, y el identificador de la parada donde quiere ir:

```
[20/09/2016-18:00:00] L 15 18
```

3. Este comando no toma en cuenta el horario actual ni los colectivos en circulación sino que se concentra en las líneas de colectivos y sus recorridos.

4. Se debe respetar la serialización propuesta en el comando anterior para enviar la consulta al servidor. Para el ejemplo anterior:

0x4C (1 byte)	0x57E17920 (4 bytes - BigEndian)	0x0000000F (4 bytes - BigEndian)	0x00000012 (4 bytes - BigEndian)

La aplicación cliente imprimirá por salida estándar un mensaje con la respuesta del servidor, que corresponde a la línea con el recorrido más corto entre las paradas especificadas.

Comando 'R' - Colectivo Recomendado

2. En este caso, la consulta es un poco más inteligente que la del comando 'L'. En el caso anterior, se tenía en cuenta solamente los recorridos de las líneas de colectivo, pero en este caso también se utilizará la información provista por comandos 'A'.
3. La aplicación leerá un timestamp, el identificador del comando, los identificadores de las paradas origen y destino:

   ```
   [20/09/2016-18:00:00] R 15 18
   ```
4. Se debe respetar la serialización propuesta en el comando anterior para enviar la consulta al servidor. Para el ejemplo anterior:

0x52 (1 byte)	0x57E17920 (4 bytes - BigEndian)	0x0000000F (4 bytes - BigEndian)	0x00000012 (4 bytes - BigEndian)

La aplicación cliente imprimirá por salida estándar un mensaje con la respuesta del servidor, que corresponde al colectivo con el cual el pasajero va a llegar más temprano a su destino.

Nota Importante:
Este comando utiliza la información provista por cada empresa de colectivo respecto de los inicios de recorrido para determinar, de entre todos los que pasan por la parada donde se encuentra el usuario, cuál es el que llegaría primero a la parada indicada. Esto difiere del comando 'L' que sólo tiene en cuenta los recorridos definidos por cada línea y no los colectivos actualmente en circulación.

Servidor

En caso de no poder responder la petición, ya sea por un argumento inválido o por algún error de entrada/salida no identificado, debe retornar el siguiente mensaje:

- **Respuesta de Error**

0xff

En caso de procesar correctamente la petición, se retorna:

- **Respuesta - Comando A**

0x00 (1 byte)	código-línea (4 bytes - BigEndian)

- **Respuesta - Comando F**

0x02 (1 byte)	segundos (4 bytes - BigEndian)

- **Respuesta - Comando L**

0x03 (1 byte)	código-línea (4 bytes - BigEndian)	segundos (4 bytes - BigEndian)

- **Respuesta - Comando R**

0x04 (1 byte)	código-línea (4 bytes - BigEndian)	segundos (4 bytes - BigEndian)

Ejemplos de Ejecución

A continuación se detalla un ejemplo de ejecución completo, incluyendo archivos de paradas y recorridos para su correcto entendimiento.

Ejemplo - 2 clientes

Dados los siguientes archivos en el servidor:

```
recorridos.txt
```

```
10 1 2 4
11 1 2 3 4
```

```
paradas.txt
```

```
1 2 600
2 3 60
2 4 60
3 4 60
```

Y las siguientes líneas en la entrada estándar de dos clientes:

```
stdin-cliente-1
```

```
[01/09/2016-17:00:00] A 11
[01/09/2016-17:08:00] A 10
```

```
stdin-cliente-2
```

```
[01/09/2016-17:10:00] F 10 2
[01/09/2016-17:10:00] F 11 2
[01/09/2016-17:10:00] L 2 4
[01/09/2016-17:10:00] R 2 4
```

El primer cliente es de tipo empresa, y agregará dos colectivos: uno que sale a las 17 horas de la terminal de la línea 11, y el otro que sale a las 17:08 de la terminal de la línea 10. Entonces la salida del cliente 1 será:

```
Colectivo de la línea 11 ha sido agregado.\n
Colectivo de la línea 10 ha sido agregado.\n
```

El segundo realiza 4 consultas a las 17:10:

- En la primera línea pregunta cuánto falta para que llegue el siguiente colectivo de la línea 10, y el servidor le contestará que faltan 8 minutos. El cliente deberá imprimir:
  - `Faltan 8 minutos para que llegue el siguiente colectivo.\n`
- En la segunda línea, pregunta cuánto falta para que llegue el siguiente colectivo de la línea 11, y el servidor le contestará que faltan 0 minutos. El cliente deberá imprimir:
  - `Faltan 0 minutos para que llegue el siguiente colectivo.\n`
- En la siguiente línea, el cliente pregunta cuál es la línea de colectivos que tarda menos entre las paradas 2 y 4. El servidor le contestará que la línea que busca es la 10, porque tarda 1 minuto. El cliente imprimirá el tiempo correspondiente a la línea de colectivos más rápida, **independientemente de en qué parte del recorrido se encuentren los colectivos de la línea**:
  - `La línea 10 tardará 1 minutos y 0 segundos en llegar a destino.\n`
- En la última línea, el cliente pregunta cuál es la línea de colectivos que le conviene tomar para ir de la parada 2 a la 4. El servidor le contestará que la línea que busca es la 11, porque tarda 2 minutos pero ya está llegando a la parada (tardará 2 minutos en total) mientras que la línea 10 tarda 1 minuto, pero el siguiente colectivo está a 8 minutos de llegar al pasajero (tardará 9 minutos en total). El cliente imprimirá el tiempo que tardará el pasajero en llegar a su destino **desde el momento de su consulta**:
  - `El colectivo de la línea 11 tardará 2 minutos y 0 segundos en llegar a destino.\n`

Restricciones

La siguiente es una lista de restricciones técnicas exigidas por el cliente:

1. El sistema debe desarrollarse en C++11.
2. Se recomienda el uso de contenedores STL siempre que sea posible.
3. Los sockets se deben emplear en modo bloqueante.
4. El servidor debe aceptar múltiples conexiones y permitir el intercambio de información con los clientes de forma concurrente.

Referencias

[1] Unix Time: https://en.wikipedia.org/wiki/Unix_time
[2] Endianness: https://en.wikipedia.org/wiki/Endianness