SIU-Guaramini

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 (02/05/2017). **Entrega 2:** clase 10 (16/05/2017).
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

Debido al incremento del alumnado en la facultad de ingeniería de la UBA, se nos ha solicitado la reimplementación del motor del sistema de inscripciones.

Descripción

El sistema SIU Guaraní posee una arquitectura basada en microservicios. Los microservicios son procesos pequeños y especializados, permitiendo distribuir el desarrollo de los mismos.
El servicio solicitado será el de inscripción a materias, siendo tareas como login, persistencia de datos o altas, bajas y modificaciones de usuarios, resueltos por otros procesos.

El servicio deberá poder ejecutarse en modo servidor y modo cliente. Para el segundo existirán "submodos" (a los que llamaremos tipos de usuario) de ejecución, que tendrán distintas funcionalidades según el modo con el que el usuario que lo invocó.

En la primer versión, los tipos de usuario soportados serán los siguientes:

- docente
- alumno
- admin

Las funciones que debe brindar el servicio son las siguientes:

- **listarMaterias**: Lista todas las materias cargadas al sistema, con sus correspondiente descripción y cupos.

- **listarInscripciones**: En el caso del modo "alumno", lista las materias a las que se inscribió, en el caso del docente, lista los alumnos inscriptos en la materia de la cuál es docente. En el caso del admin, listará todos los alumnos en todas las materias.

- **inscribir**: En el caso del modo alumno, recibe como argumentos el código de la materia y el código del curso. Si la inscripción es exitosa responde OK, caso contrario responde el motivo de rechazo. En el caso del docente, recibe como parámetro extra el alumno a inscribir, pudiendo sólo inscribir a alumnos en el curso donde se encuentra como docente. El usuario admin puede inscribir a un alumno en cualquier curso.

- **desinscribir**: Utilizando la misma lógica que inscribir, permite a un alumno salir de un curso del cuál se inscribió. Un docente puede desinscribir solamente a alumnos de su curso, un admin a cualquier alumno de cualquier curso.

Formato de Línea de Comandos

Servidor

El modo servidor recibirá

1. El puerto por el que escuchará conexiones
2. El nombre de un archivo TSV de donde cargará los usuarios del sistema
3. El nombre de otro archivo TSV con las materias del sistema

Por ejemplo

```
./server 8080 usuarios.txt materias.txt
```

Cliente

El modo cliente recibirá

1. La ip del servidor a conectarse
2. El puerto al cual debe conectarse
3. El modo de ejecución del servicio
4. El id del usuario (sólo aplica a modo alumno o docente)

Por ejemplo

```
./cliente 127.0.0.1 8080 alumno 1337
```

ó

```
./cliente 127.0.0.1 8080 admin
```

Formato de archivos de entrada del servidor

El archivo de usuarios está compuesto por líneas con el siguiente formato donde los items estan separados por tabs **\t** y cada línea finaliza en un **\n**:

```
<tipo-usuario>\t<id>\t<nombre>\n
```

El archivo de materias está compuesto por líneas con el siguiente formato, similar al anterior:

```
<código-materia>\t<id-curso>\t<descripcion>\t<id-docente>\t<vacantes>\n
```

Se garantiza que solo habrá un docente en cada materia y que todos los docentes listados en el archivo de materias estaran tambien listados en el archivo de usuarios.

Códigos de Retorno

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

Entrada y Salida

Servidor

El servidor deberá esperar por consola que se ingrese el caracter **q**. Cuando lo recibe, debe terminar ordenadamente, cerrando conexiones y liberando recursos. En paralelo debe atender conexiones entrantes. No se escribirá nada por salida estándar.

Mediante la salida de error se imprimirá la siguiente información donde los items están separados por un espacio y cada línea finaliza con un **\n**:

- Nueva conexión aceptada

  ```
  <tipo-usuario> <id> conectado.\n
  ```

- Comando recibido desde un cliente

  ```
  <tipo-usuario> <id> ejecuta <id-comando>.\n
  ```

- Cliente desconectado

  ```
  <tipo-usuario> <id> desconectado.\n
  ```

En caso de querer ingresar con un tipo de usuario no soportado, se imprimirá

```
<tipo-usuario> es un tipo de usuario inválido.\n
```

En caso de querer ingresar con un id de usuario no válido, se imprimirá

```
<id> es un <tipo-usuario> inválido.\n
```

En ambos casos se desconectará al cliente.

Cliente

Los comandos se ingresarán en texto plano por entrada estándar con el siguiente formato

```
<comando> [<argumentos>]
```

A continuación se detalla la sintaxis de cada comando:

listarMaterias: No posee argumentos. Lista las materias cargadas al sistema, con el siguiente formato:

```
<codigo> - <descripcion>, Curso <id-curso>, <nombre-docente>, <vacantes>
vacantes.\n
```

El orden del listado es por codigo de materia y luego por curso.

listarInscripciones (todos los modos): No posee argumentos. En el caso del modo "alumno", lista las materias a las que se inscribió, con el siguiente formato

```
<codigo> - <descripcion>, Curso <id-curso>, <nombre-docente>.\n
```

En el caso del docente, lista los alumnos inscriptos en la materia de la cuál es docente. En el caso del admin, listará todos los alumnos en todas las materias. El formato de ambos es el siguiente

```
<codigo> - <descripcion>, Curso <id-curso>, <nombre-docente>:\n
<id-alumno> <nombre>\n
<id-alumno2> <nombre2>\n
```

El orden del listado es por codigo de materia y luego por curso y los alumnos por su identificador.

inscribir: En el caso del modo alumno, recibe como argumentos el código de la materia y el código del curso.

```
inscribir <codigo-materia> <id-curso>
```

Ejemplo:

```
./cliente 127.0.0.1 8080 alumno 1337
> inscribir 7542 1
```

En caso de inscripción exitosa, se devuelve el mensaje:

```
Inscripción exitosa.\n
```

Si el alumno ya estaba inscripto, la respuesta es:

```
Inscripción ya realizada.\n
```

Si la materia no existe, el mensaje devuelto es:

```
La materia <codigo-materia> no es válida.\n
```

Si el curso no existe, el mensaje devuelto es:

```
El curso <id-curso> en la materia <codigo-materia> no es válido.\n
```

Si el curso no tiene más vacantes

```
El curso <id-curso> de la materia <codigo-materia> no posee más
vacantes.\n
```

En el caso del docente, recibe como parámetro extra el alumno a inscribir, pudiendo sólo inscribir a alumnos en el curso donde se encuentra como docente.

```
Inscribir <id-alumno> <codigo-materia> <id-curso>
```

Ejemplo:

```
./cliente 127.0.0.1 8080 docente 8008
> inscribir 1337 7542 1
```

Inscribe al alumno 1337, a la materia con el código 7542, curso 1

Esto agrega dos validaciones:
Si el alumno no existe, se debe devolver

```
El alumno <id-alumno> es inválido.\n
```

Si la materia o curso no corresponden al docente, debe devolver

```
No tiene permisos para operar sobre <codigo-materia>, curso <id-curso>.\n
```

El usuario admin puede inscribir a un alumno en cualquier curso.

desinscribir: Utilizando la misma lógica que inscribir, permite a un alumno salir de un curso al que previamente se inscribió

```
desinscribir <codigo-materia> <id-curso>
desinscribir <id-alumno> <codigo-materia> <id-curso>
```

En caso de desinscripción exitosa, se devuelve el mensaje:

```
Desinscripción exitosa.\n
```

Si la inscripción no existe (ya sea porque no hubo inscripción o uno de los códigos es inválido), se devuelve el mensaje:

```
Desincripción inválida.\n
```

Si la materia o curso no corresponden al docente, debe devolver

```
No tiene permisos para operar sobre <codigo-materia>, curso <id-curso>.\n
```

Protocolo de comunicación

Formato del mensaje

Los comandos se enviarán utilizando texto plano y preconcatenando el largo del mensaje. El mismo será un entero de 4 bytes big endian.

Inicio

Al conectarse el cliente, el primer mensaje enviado por el cliente será el modo y el id (si corresponde) en texto plano, separados por un guión medio.
Por ejemplo, si se conecta un alumno con padrón 1337, el mensaje enviado será “alumno-1337”
Si se conecta un administrador, sólamente enviará “admin”

Envío de comandos

Los comandos serán enviados en mensajes de texto con el siguiente formato

```
<cod-comando>-<arg1>-<argN>...
```

Donde `cod-comando` es el código del comando a enviar.

Los códigos de comando son los siguientes:

- `lm`: listar materias
- `li`: listarInscripciones
- `in`: inscribirse
- `de`: desinscribirse

Ejemplos de Ejecución

Ejemplo 1

Sean el archivo `usuarios.txt`:

```
alumno\t1\talice\n
alumno\t2\tbob\n
docente\t3\tcraig\n
docente\t4\tmallory\n
docente\t4\teve\n
docente\t12\ttrudy\n
```

Y sea el archivo materias.txt:

```
1001\t1\tAlgoritmos I\t3\t20\n
1003\t1\tAlgoritmos III\t4\t20\n
1003\t2\tAlgoritmos III\t5\t20\n
```

```
1009\t1\tAlgoritmos IV: la batalla final\t12\t10\n
```

El servidor se ejecuta con la siguiente línea:

```
./server 8080 usuarios.txt materias.txt
```

Sea ahora un alumno que quiere ver que materias hay e inscribirse en 3 materias, Algorimos I, Algorimos III y Cocina II. Para ello, sea el archivo comandos.txt:

```
listarInscripciones\n
listarMaterias\n
inscribir 1001 1\n
inscribir 1003 2\n
inscribir 1032 1\n
inscribir 1001 1\n
listarInscripciones\n
```

Entonces el alumno ejecuta lo siguiente, donde 1 es el id de él:

```
./cliente 127.0.0.1 8080 alumno 1 < comandos.txt
```

Y la salida es:

```
1001 - Algoritmos I, Curso 1, craig, 20 vacantes.\n
1003 - Algoritmos III, Curso 1, mallory, 20 vacantes.\n
1003 - Algoritmos III, Curso 2,  eve, 20 vacantes.\n
1009 - Algoritmos IV: la batalla final, Curso 1, trudy, 10 vacantes.\n
Inscripción exitosa.\n
Inscripción exitosa.\n
La materia 1032 no es valida.\n
Inscripción exitosa.\n
1001 - Algoritmos I, Curso 1, craig, 19 vacantes.\n
1003 - Algoritmos III, Curso 1, mallory, 20 vacantes.\n
1003 - Algoritmos III, Curso 2,  eve, 19 vacantes.\n
1009 - Algoritmos IV: la batalla final, Curso 1, trudy, 9 vacantes.\n
```

Ejemplo 2

Sea el archivo `usuarios.txt`:

```
alumno\t1\talice\n
alumno\t2\tbob\n
alumno\t3\tcraig\n
docente\t4\tmallory\n
```

Y sea el archivo materias.txt:

```
1001\t1\tAlgoritmos I\t4\t2\n
```

Levantando el servidor de la misma forma que en el ejemplo 1, y ejecutando 2 clientes en paralelo de la siguiente forma:

```
./cliente 127.0.0.1 8080 alumno 1 < comandos.txt &
./cliente 127.0.0.1 8080 alumno 2 < comandos.txt
```

Donde el archivo comandos.txt contiene:

```
listarMaterias\n
```

Ambos clientes deberían retornar

```
Inscripción exitosa.\n
```

Sin embargo, si un tercer cliente se conecta después de los 2, de la siguiente forma:

```
./cliente 127.0.0.1 8080 alumno 3
```

E ingresa los siguientes comandos:

```
listarMaterias\n
inscribir 1 1001 1\n
```

La salida debiera ser la siguiente:

```
1003 - Algoritmos I, Curso 1, mallory, 0 vacantes.\n
El curso 1 de la materia 1001 no posee mas vacantes.\n
```

Notar que dado el contexto concurrente, no se puede saber qué alumno se anota primero, por lo que si se intentan anotar los 3 en simultáneo, cualquiera de los 3 puede quedar afuera, pero sólo uno de ellos. Tampoco tiene mucho sentido consultar por vacantes en un contexto concurrente, ya que no se sabe si en paralelo hay una inscripción o desinscripción.

Ejemplo 3

Sea el archivo `usuarios.txt`:

```
alumno\t1\talice\n
docente\t57\tbob\n
docente\t22\tmarge\n
docente\t1\tHuergo\n
```

Y sea el archivo materias.txt:

```
1001\t1\tAlgoritmos I\t57\t20\n
2002\t7\Laboratorio\t22\t20\n
```

Suponiendo que el servidor está levantado, el alumno 1, alice, puede inscribirse de la siguiente forma:

```
./cliente 127.0.0.1 8080 alumno 1
inscribir 1 1001 1\n
inscribir 1 2002 7\n
```

Recibiendo como respuesta

```
Inscripción exitosa.\n
Inscripción exitosa.\n
```

Si el docente Bob intenta desinscribir a Alice de todas las materias ejecutando:

```
./cliente 127.0.0.1 8080 docente 57
desinscribir 1 1001 1\n
desinscribir 1 2002 7\n
```

Sólo podrá desinscribir de la materia 1001 una única vez, por lo que su salida será

```
Desinscripción exitosa.\n
No tiene permisos para operar sobre 2002, curso 7.\n
```

Ahora supongamos que le pide ayuda al admin y este corre los mismos comandos:

```
./cliente 127.0.0.1 8080 admin
desinscribir 1 1001 1\n
desinscribir 1 2002 7\n
```

La desinscripción materia 1001 una única vez, pero la 2002 todavía es válida por lo que su salida será

```
Desinscripción inválida.\n
Desinscripción exitosa.\n
```

Restricciones

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

1. El sistema debe desarrollarse en ISO C++11.
2. Está prohibido el uso de variables globales.
3. La aplicación debe soportar clientes en simultáneo.

Referencias

[1] Ejemplo de referencia: http://en.wikipedia.org
[2] std::getline: http://www.cplusplus.com/reference/string/string/getline/
[3] std::map: http://www.cplusplus.com/reference/map/map/