ANEXO

CÁTEDRA:

SISTEMAS DISTRIBUIDOS

ENUNCIADO DE TRABAJOS DE LABORATORIO

Año 2015

Versión 2.2

Docentes

Profesor Adjunto: **Mg. Ing. Ricardo Antonio López**
Jefe de Trabajos Prácticos: **Lic. Cristian Javier Parise**

INDICE

TRABAJOS DE LABORATORIO 3
Objetivo general de los Trabajos Prácticos de Laboratorio. 3
Características 3
Forma de aprobación 3
Lista de Trabajos de Laboratorio 3

Trabajo de Laboratorio 1 4
Cliente / Servidor. Sockets. RPC. Threads. Concurrencia. RFS. 4

Trabajo de Laboratorio 2 6
Java RMI. Concurrencia. Sincronización. Transacciones 6

TRABAJOS DE LABORATORIO

Objetivo general de los Trabajos Prácticos de Laboratorio.

- Conocimiento y realización de trabajos sobre arquitectura Cliente / Servidor con diferentes tecnologías.
- Conocimiento de la problemática de Sincronización.
- Manejo de la generación de páginas WEB dinámicas.
- Manejo de Código móvil y Sistemas distribuidos de archivo.

Características

El Informe deberá reunir las siguientes características:

1. Cada Grupo presentará su informe a efectos de su calificado por el profesor. Los trabajos que no reúnan los requisitos mínimos serán devueltos para su corrección.
2. Deberá ser presentado a la cátedra confeccionado en grupos **no mayores de dos alumnos**, con discusión individual por alumno.
3. Para su preparación e impresión, el trabajo práctico deberá ser entregado de la siguiente forma:
   - En formato HTML o PDF, con un índice que refleje su estructura. Se incluirá una portada que deberá identificar a los integrantes del grupo y contener la firma de los mismos.
   - Toda la bibliografía utilizada deberá ser referenciada indicando título y autor, en una sección dedicada a tal efecto.
   - El programa de aplicación que implementa la solución.
   - El código fuente debe estar debidamente comentado. La solución debe ser desarrollada utilizando el lenguaje de programación indicado. También se debe incluir el makefile correspondiente o instrucciones o script para su correcta compilación, además del propio batch de prueba de ser necesario.

Forma de aprobación

Se tendrá en cuenta para la aprobación del trabajo práctico y los integrantes del grupo:

- Funcionamiento de la aplicación desarrollada. Se evaluará si la funcionalidad cumple con lo solicitado. En caso de que así no sea, el trabajo práctico se considerará desaprobado.
- Estructura general de la presentación, su legibilidad y facilidad de lectura y comprensión.
- Contenido del informe y el uso de la información técnica para elaborarlo.
- Evaluación del grupo como un todo y a cada uno de sus integrantes.

Lista de Trabajos de Laboratorio

PRÁCTICA Nº 1 – Cliente / Servidor. Sockets. RPC. Threads. Concurrencia.

PRÁCTICA Nº 2 – Java RMI. Concurrencia. Sincronización.

PRÁCTICA Nº 3 – HTTP. HTML. CGI. AJAX.

PRÁCTICA Nº 4 – Código Móvil. DFS.

Trabajo de Laboratorio 1

Cliente / Servidor. Sockets. RPC. Threads. Concurrencia. RFS.

1. Tome el código provisto en la **carpeta p11.**

   a) Analice los fuentes client.c y server.c y modifíquelos para que la consulta del cliente y la respuesta del servidor sea más interactiva (cambiando texto y/o requerimiento del cliente al servidor).

   b) Analice los fuentes client2.c y server2.c para ver su funcionamiento. Modifique el tamaño de los buffers para que sean de longitud fija: 10^3, 10^4, 10^5 y 10^6 bytes. Explique las diferencias obtenidas al ejecutar en cada caso.

2. Tome el código provisto en la **carpeta p12.**

   a) Analice el código provisto en la **carpeta p12** para luego responder las siguientes consignas:

      1) Definir brevemente qué es un servidor con estados y qué es un servidor sin estados.

      2) Explicar si el servidor implementado en **p12** es de la clase de servidores con o sin estado.

   b) Tomando el código analizado realice un servidor opuesto al ya implementado (Si es sin estado realice uno con estado o viceversa).

3. Dada la siguiente especificación RPC:

```
/* rfs.x */

typedef opaque file_data<>;

struct open_record
{
   string file_name<>;
   int flags;
};

struct read_record
{
   int fd;
   int count;
};

program RFS
{
   version RFS_VERS_1
   {
      int RFS_OPEN(open_record r) = 1;
      file_data RFS_READ(read_record r) = 2;
      int RFS_CLOSE(int fd) = 3;
   } = 1;
} = 0x20000001;
```

   a) Agregue la operación RFS_WRITE al .x de la especificación e implemente la solución completa.

   b) Implemente el equivalente en Java con sockets.

   c) Comparar y comentar la complejidad de ambas implementaciones.

4. Teniendo en cuenta el servidor del ejercicio anterior, 3.b:

   a) Pruebe cancelar un cliente cuando se está en el medio del funcionamiento y vuelva a arrancar el cliente. Observe y documente qué ocurre.

b) Pruebe cancelar el servidor cuando se está en el medio del funcionamiento y vuelva a arrancar el cliente. Observe y documente qué ocurre.

c) Determine si es un servidor con estados o sin estados.

5. Modifique la solución 3.b), y utilice los threads de Java para que se puedan responder requerimientos de clientes de manera concurrente.

6. Concurrencia.

a) Disparar dos clientes a la vez y verificar (documentar) si las solicitudes de ambos clientes son atendidas por el mismo o por distintos hilos en el servidor

b) Determine si la implementación del ejercicio 3.- tiene un thread por requerimiento, por conexión o por recurso (en términos del cap. 6 de Coulouris).

c) Transferir un archivo de gran tamaño (GBs) completo en la versión con RPC y en la versión con Java Sockets (utilizando las primitivas de lectura o escritura con un tamaño de buffer igual en ambos casos) y cronometrar (que el mismo cliente muestre cuanto demoró) en ambos casos, sacando las conclusiones del caso respecto a la performance de cada solución.

7. Proponga una modificación del ejercicio 3.- de manera tal que se tenga un conjunto (pool) de threads creados con anterioridad a la llegada de los requerimientos y donde se administren los threads de acuerdo a las llegadas de requerimientos ¿Sería útil cambiar la cantidad de threads administrados de esta manera? Justifique.

Trabajo de Laboratorio 2

Java RMI. Concurrencia. Sincronización. Transacciones

1. Basándose en el proyecto java rmisum, implemente dos servidores con Java RMI, cada uno de ellos implementando funciones distintas (uno de ellos suma y resta y el otro multiplicación y división). El cliente debe tomar de la línea de comando tanto el indicador de operación como los operandos a utilizar. Valiéndose de un debugger y/o un analizador de protocolos, informar:
    a. En cuáles puertos atiende cada uno de los objetos remotos de los servidores?.
    b. Cuál es el puerto en el que atiende RMIRegistry?.
    c. Identificar el mensaje que transfiere el cliente con RMIRegistry.
    d. Existe identificación de puertos.?.
    e. Los mensajes hacia RMI Registry se valen de TCP o UDP.?
2. Implemente con Java RMI el servidor de archivos remoto del punto 3.b de la práctica anterior. Debe tener las cuatro operaciones básicas: open, close, read y write.
    a. Compare la especificación de interfaz entre cliente y servidor de la implementación con RMI con la basada en sockets de Java. Efectúe los comentarios que haya lugar.
    b. Compare la complejidad de la implementación con Java sockets y con RMI tanto para el servidor como para el cliente. Efectúe los comentarios que haya lugar.
3. Haga que el servidor responda a requerimientos en forma concurrente. Defina un experimento con ayuda del analizador de protocolos, donde se pueda comprobar que se tiene un servidor concurrente.
4. Responda las siguientes cuestiones:
    a. Cuáles son los problemas que tiene un servidor concurrente?
    b. Qué mecanismos se tienen disponibles con Java RMI?
    c. Podría considerar que RPC y RMI tienen el mismo nivel de abstracción? Justifique.
    d. Identifique similitudes y diferencias entre RPC y Java RMI.
    e. Determine si el servidor concurrente con Java RMI tiene un thread por requerimiento, por conexión o por recurso (en términos del cap. 6 de Coulouris).
5. Elabore un escenario donde pueda verificarse si RMI otorga un servicio concurrente en forma automática para el acceso a objetos remotos. En especial otorgue evidencia que las variables de instancia de un determinado objeto remoto, no se “contamina” con sus equivalentes de otras instancias del mismo objeto a las que se está accediendo en forma concurrente.
6. ¿Hay una manera sencilla de establecer un timeout para el objeto/clase que hace un RMI?. Implemente una solución.
7. Implemente un servidor de hora con Java RMI utilizando el algoritmo de Cristian. Al utilizar la hora patrón, tome un grupo de valores (por ejemplo 5), eligiendo el que haya demandado menor tiempo de tránsito de los mensajes.
8. Desarrolle una aplicación que se comunique con un servidor NTP de internet y que extraiga 8 muestras del retardo con que correspondería ajustar la hora obtenida. Presentar en pantalla una tabla con cada muestra del tiempo obtenido del servidor en formato YYYY/MM/DD HH:MM:SS.mmm en relación con cada retardo.
9. Transacciones:

    a) Explicar cómo el protocolo de compromiso de dos fases para transacciones anidadas, se asegura que si la transacción de nivel superior se compromete, todas las descendientes se comprometen o se abortan.

    b) Utilizando la API de Java para transacciones (JTA), hacer una aplicación que establezca conexión a cinco diferentes bases de datos de postgres (banco1, banco2, …,banco5) para

efectuar una transacción bancaria depositando un monto en el banco1 y extrayendo el dinero por partes iguales de los bancos2 a banco4 de una cuenta que debe ser especificada por el usuario.
Los bases de datos a crear contendrán todas el mismo esquema consistente en una sola tabla:

Cuentas (id, titular, fecha_creacion, bloqueada (boolean), saldo (real))

(en el aula virtual se deja el script de creación *dbCreate.sql* de las 5 bases de datos con las mismas cuentas iniciales y diferentes saldos)

Utilizar como punto de partida los fuentes de la carpeta *jta* del aula virtual, que permite establecer una conexión jdbc a postgres y generar una transacción JTA con un solo branch a la base banco1)
La aplicación deberá:

- Solicitar al usuario un monto.
- Solicitar al usuario un número de cuenta para depositar (banco1) y un número de cuenta para extraer (banco2 a banco 5).

Solicitar al usuario importe a transferir, la cuenta origen y la cuenta destino

begin Tx.
depositar dinero al banco1 cuenta destino
direro_a_retirar = dinero /4
retirar dinero_a_retirar del banco2 cuenta origen (sitio2)
retirar dinero_a_retirar del banco3 cuenta origen
retirar dinero_a_retirar del banco4 cuenta origen
retirar dinero_a_retirar del banco5 cuenta origen
si no hay excepciones(no existe cuenta db, no existe cuenta cr, cuenta bloqueada, no alcanza el saldo en alguno de los banco, etc)
commit Tx.
sino
rollback Tx

Mostrar el resultado del saldo para las cuentas luego de la transferencia.
Pedir además una confirmación final para que la transacción se comprometa o se aborte.