



CÁTEDRA:

SISTEMAS DISTRIBUIDOS

ENUNCIADO DE TRABAJOS DE LABORATORIO

Año 2015

Versión 2.1

Docentes

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

INDICE

TRABAJO 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

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 N° 1 – Cliente / Servidor. Sockets. RPC. Threads. Concurrencia.

PRÁCTICA N° 2 – Java RMI. Concurrencia. Sincronización.

PRÁCTICA N° 3 – HTTP. HTML. CGI. AJAX.

PRÁCTICA N° 4 – Código Móvil. DFS.

Trabajo de Laboratorio 1

Cliente / Servidor. Sockets. RPC. Threads. Concurrency. 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.