Programación Concurrente y de Tiempo Real
Grado en Ingeniería Informática
Examen Final de Prácticas
Junio de 2014

Apellidos: Nombre:

Grupo (A ó B):

1. Notas

1. Escriba su nombre y apellidos con letra clara y legible en el espacio habilitado para ello.
2. Firme el documento en la esquina superior derecha de la primera hoja y escriba debajo su D.N.I.
3. Dispone de $2:30'$ horas para completar el ejercicio.
4. Puede utilizar el material bibliográfico (libros) y copia de API que estime convenientes. Se prohíbe el uso de apuntes, *pen-drives* y similares.
5. Recuerde que la compilación con C++11 puede requerir el uso de `-std=c++0x` en lugar de `-std=c++11`.
6. Recuerde que los ordenadores están configurados para arrancar sin memoria de la sesión anterior. Por ello, recuerde que si cambia de sistema operativ0 es posible que pierda todo el trabajo hecho. Téngalo en cuenta. Recomendamos desarrollar todo el examen bajo sistema operativo Linux.
7. Entregue sus productos, utilizando la tarea de entrega disponible en el Campus Virtual, en un fichero (`.rar, .zip`) de nombre

   `Apellido1_Apellido_2_Nombre`

   que contendrá una subcarperta por cada enunciado del examen, la cuál contendrá a su vez el conjunto de ficheros que den solución a ese enunciado en particular.

2. Criterios de Corrección

1. El examen práctico se calificará de cero a diez puntos y ponderará en la calificación final al 50 % bajo los supuestos recogidos en la ficha de la asignatura.

2. Las condiciones que una solución a un enunciado de examen práctico debe cumplir para considerarse correcta son:

   a) Los ficheros subidos a través del Campus Virtual que conforman el examen práctico se ajustan al número, formato y nomenclatura de nombres explicitados por el profesor en el documento de examen.

   b) El contenido de los ficheros es el especificado por el profesor en el documento de examen en orden a solucionar el enunciado en cuestión.

   c) Los programas elaborados por el alumno se pueden abrir y procesar con el editor y el compilador del lenguaje, y realizan un procesamiento técnicamente correcto, según el enunciado de que se trate.

   d) Se entiende por un procesamiento técnicamente correcto a aquél código de programa que compila correctamente sin errores, cuya semántica dá soporte a la solución pedida, y que ha sido escrito de acuerdo a las convenciones de estilo y eficiencia habituales en programación

3. Enunciados

1. (Factorial de Grandes Enteros, 3.4 puntos) Es sabido que la función factorial crece muy rápidamente, siendo imposible efectuar el cálculo del factorial de un número natural muy grande. Para solventar problemas de este tipo, Java incorpora (entre otras) la clase `java.math.BigInteger`, que proporciona una aritmética de precisión infinita al representar grandes enteros mediante objetos de la clase `BigInteger`. El siguiente programa en Java calcula de forma recursiva el factorial de enteros grandes haciendo uso de objetos de la clase indicada.

```
import java.math.BigInteger;
public class usoBigInteger
{

    public static void main(String[] args) {
      //Se muestra el resultado del factorial de 500 algo
      //imposible de hacer con los tipos primitivos de java
      System.out.println(factorial(new BigInteger("500")));
      System.out.println(BigSuma());
    }
      //Funcion recursiva para el calculo del factorial
    public static BigInteger factorial(BigInteger n) {
      //Si el parametro n es 1, se retorna 1 ya que el
      //factorial de 1 es 1 por definicion
      if(n.equals(BigInteger.ONE)) {
          return BigInteger.ONE;
      }
      return n.multiply(factorial(n.subtract(BigInteger.ONE)));
```

```
    }

    public static BigInteger BigSuma(){
        //Ejemplo de cómo sumar dos enteros grandes
        BigInteger a = new BigInteger("1000000");
        BigInteger b = new BigInteger("10000000");
        return (a.add(b));
    }

}
```

Se pide desarrollar un programa que resuelva este cálculo de forma paralela de acuerdo a las especificaciones siguientes:

- El programa debe particionar el trabajo entre tantas tareas `Runnable` como núcleos haya disponibles de forma automática. Puesto que el número de tareas será invariable, deberá procesarlas mediante un ejecutor adecuado a esa invariabilidad.
- Si la considera necesaria, deberá proveer exclusión mutua cuando haga falta. Para ello, utilice cerrojos de clase `ReentrantLock`.
- Todo el código necesario residirá en el fichero `facEGrandesParalelo.java`.

2. (Ibex Distribuido, 3.4 puntos) Su cliente le pide diseñar un cliente java capaz de conocer las cotizaciones remotas de los 35 principales valores bursátiles españoles. Para solucionar el problema, usted opta por desarrollar una versión simplificada del problema que inicialmente trabaja con cinco valores que se indican a continuación, junto con su acrónimo entre paréntesis: TRINCOSA (TRI), CHORIZOS NACIONALES S.A. (CHN), BANCO MALO PERO MALO (BMM), CORRUPTOLANDIA (CRR) y TOMA EL DINERO Y CORRE S.A. (TDC). Se pide escribir una arquitectura RMI distribuida que modele el problema, de acuerdo a los requerimientos siguientes:

- Ficheros `iIBEX5.java`, `sIBEX5.java` y `cIBEX5.java` contendrán la interfaz, servidor y cliente.
- El servidor correrá en el puerto 2020. Cada 100 milisegundos las cotizaciones de los cinco valore de prueba subirán o bajarán de forma aleatoria.
- El cliente deberá poder consultar los valores de todas las cotizaciones del servidor, o la de un valor en concreto, y mostrará en pantalla la información recibida.
- Las cotizaciones iniciales de los cinco valores serán fijados al arrancar el servidor de forma manual.
- El servidor debe poder procesar peticiones simultáneas de información de manera adecuada.

3. (Análisis de Rendimiento, 3.2 puntos) Deseamos comprobar de manera empírica con el lenguaje C++ que los objetos de clase `atomic` son más eficaces en términos de rendimientos que los cerrojos `mutex` sobre un recurso compartido de tipo `int`. Para ello, debe desarrollar en `analisisRendimiento.ccp` el

código necesario para ilustrar este comportamiento. El programa creará los hilos necesarios mediante funciones lambda, e imprimirá una tabla de tiempos con la estructura siguiente:

Total de Accesos al Recurso	Tiempo con Mutex	Tiempo con atomic
dato 11	dato 12	dato 13
...	...	...
dato 61	dato 62	dato 63

Para ello, es necesario saber medir tiempos de ejecución en C++. En el siguiente código, le proporcionamos un ejemplo con todo lo necesarios para ello:

```
#include <iostream>
#include <chrono>
#include <ctime>

long fibonacci(int n){
    if (n < 3) return(1);
    return fibonacci(n-1) + fibonacci(n-2);
}

int main() {
    std::chrono::time_point<std::chrono::system_clock> start, end;
    start = std::chrono::system_clock::now();
    std::cout << "f(42) = " << fibonacci(42) << '\n';
    end = std::chrono::system_clock::now();

    std::chrono::duration<double> elapsed_seconds = end-start;
    std::time_t end_time = std::chrono::system_clock::to_time_t(end);

    std::cout << "Calculo terminado a las: " << std::ctime(&end_time)
              << "Tiempo: " << elapsed_seconds.count() << "s\n";
}
```