Programación Concurrente y de Tiempo Real
Grado en Ingeniería Informática
Examen Final de Prácticas de la Asignatura
Febrero de 2013

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. 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 cual 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% bajos 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 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. **(Matriz-Vector)** Se desea disponer de un programa que realice de forma paralela, supuesto un procesador *multi-core,* el producto de una matriz por un vector $A \cdot b = y$ de acuerdo al siguiente esquema: [3.4p]

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

El programa debe identificar el número de núcleos disponibles en la computadora en que se ejecuta y dividir el problema en tantas partes como núcleos existan de forma automática. Utilice para ello la ecuación siguiente, suponiendo el coeficiente de bloqueo igual a cero.

$$Nt = \frac{N_{nd}}{1 - Cb}$$

El programa creará -mediante la técnica de herencia de la clase `Thread`- tantos *thread*s como Nt indique, asignando a cada uno una parte del problema a través del constructor de clase. Cada *thread* debería tener una prioridad igual a 6. Incorpore un menú de usuario con dos opciones: una que permita introducir las dimensiones, la matriz y el vector manualmente, y otra que lea las dimensiones y rellene ambos objetos de forma aleatoria mediante una instancia de la clase `Random`. Guarde su trabajo en `matVectorConcurrente.java`.

2. **(Creatinina-IMC)** La ecuación de *Cockroft-Gault* permite estimar el aclaramiento de creatinina (estado de la función renal) en el ser humano a partir de una determinación de creatinina en sangre de acuerdo a la siguiente expresión: [3.3p]

$$Aclaramiento = \frac{(140 - Edad) \times Peso(kg)}{72 \times Creatinina(mg/dl)} \times 0.85 \; si \; es \; mujer$$

De igual forma, el índice de masa corporal (IMC) permite conocer el grado de sobrepeso en el ser humano de acuerdo a la siguiente expresión:

$$IMC = \frac{Peso(kg)}{Altura^2(m)}$$

Se desea disponer de una calculadora remota para determinar remotamente ambos parámetros para un paciente de acuerdo a la siguiente interfaz:

```
import java.rmi.*;
public interface fRenal extends Remote
{
 public float aclarCreat(Datos t) throws RemoteException;
 public float creatininaMedia() throws RemoteException;
 public float indMasaCorporal(float peso, float altura)throws RemoteException;
}
```

Los datos se transfieren desde al cliente al servidor en una clase llamada `Datos.java`. Se pide escribir, con base en la interfaz proporcionada, un código de cliente (`cRenal.java`) y servidor (`sRenal.java`) que permita implantar una aplicación RMI completa para efectuar ambas determinaciones en red, suponiendo como puerto de comunicaciones en el lado del servidor el puerto número 1066. El sistema deberá cumplir las especificaciones siguientes:

- El cliente leerá desde teclado los valores de creatinina, edad, peso, altura y sexo.
- El cliente determinará remotamente el aclaramiento a través del método `aclarCreat(Datos)`, e imprimirá un mensaje en el terminal de acuerdo a la siguiente tabla:

Aclaramiento	Mensaje a Imprimir
≥ 90	Función normal
$60 - 89$	Daño renal leve
$30 - 59$	Daño renal moderado
$15 - 29$	Daño renal grave
< 15	Fallo Renal

- El cliente también imprimirá la creatinina media utilizando el método `creatininaMedia()`.
- El cliente determinará remotamente el IMC a través del método `indMasaCorporal(peso, altura)`, e imprimirá un mensaje en el terminal de acuerdo a la siguiente

tabla:

IMC	Mensaje a Imprimir
< 16.00	Delgadez Severa
$16.00 - 16.99$	Delgadez Moderada
$17 - 18.49$	Delgadez Aceptable
$18.50 - 24.99$	Peso Normal
$25.00 - 29.99$	Sobrepeso
$30.00 - 34.99$	Obesidad Tipo I
$35.00 - 40.00$	Obesidad Tipo II
> 40.00	Obesidad Tipo III

- Genere una política de seguridad que únicamente permite la comunicación a través del puerto 1066 y guárdela en un fichero `seg.policy`.
- Además, el servidor almacenará los diferentes valores de creatinina de las múltiples consultas que recibe, y enviará a los clientes el valor medio de creatinina a través del método `creatininaMedia()`.
- Debe entregar los ficheros `fRenal`, `Datos`, `sRenal` y `cRenal`.

3. **(LíneaCajas)** Escriba ahora un monitor java que simule una línea de 20 cajas de un hipermecado. Cada caja admite una cola de diez clientes en espera. Los clientes se dirigen a una caja libre (menos de diez clientes en cola), si la hay. En caso contrario, deben esperar. Utilizando cerrojos de clase `ReentrantLock` y variables `Condition`, implante el monitor pedido. Realice el diseño de los clientes de la línea de cajas utilizando objetos que implementen la interfaz `Runnable`, y controle su ciclo de vida mediante un `ThreadPoolExecutor`. Guarde el monitor en `linealCajas.java` y el diseño de hilos en `clientCajas.java`. [3.3p]

(ESPACIO PARA ANOTACIONES DE CORRECCIÓN. NO ESCRIBA AQUÍ)