Programación Concurrente y de Tiempo Real
Grado en Ingeniería Informática
Examen Final de Prácticas
17 de Enero de 2022

1. Notas de Procedimiento

1. Dispone de 180 minutos para completar el ejercicio.
2. Puede utilizar el material bibliográfico (libros) y copia de API que estime convenientes.
3. 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 subcarpeta por cada enunciado del examen, la cual contendrá a su vez el conjunto de ficheros que den solución a ese enunciado en particular. Cada fichero de código debe incluir un comentario con su nombre, apellidos y D.N.I.
4. Los productos de examen deben entregarse **única y exclusivamente** mediante la tarea de entrega habilitada para ello en el Campus Virtual, y dentro del plazo de tiempo establecido. Entregas efectuadas incumpliendo lo anterior no serán consideradas a efectos de corrección.

2. Criterios Generales de Corrección

Los códigos entregados deben satisfacer los siguientes criterios[1]:

- compilar correctamente sin errores ($-n$);
- ejecutarse sin lanzamiento de excepciones u otras anomalías graves de ejecución ($-n$);
- poseer una semántica que da soporte a la solución pedida en términos de entrada/salida, seguridad, vivacidad, rendimiento, etc. ($-n$);

[1] Entre paréntesis, se indica la puntuación a descontar derivada del incumplimiento del criterio, para un enunciado valorado con n puntos.

- recoger íntegramente todas las especificaciones establecidas en el documento de examen e implementarlas tal y como se especifican ($-1{,}5$/especificación).
- estar contenidos en ficheros conformes en número, formato y nomenclatura de nombres con lo explicitado en este documento de examen ($-n$).

3. Enunciados de Examen

1. (3.0 puntos) Los cuaterniones de *Hamilton* son una extensión del cuerpo de los números reales similar a los números complejos, ampliamente aplicados en diversas ramas de la Física, como el electromagnetismo o la teoría cuántica. El conjunto de los cuaterniones está dado por $\mathbb{H} = \{a + bi + cj + dk : a, b, c, d \in \mathbb{R}\}$, donde $i^2 = j^2 = k^2 = ijk = -1$. Definimos arbitrariamente a continuación las siguientes operaciones con cuaterniones:

   - suma: si $q_1 = a_1 + b_1 i + c_1 j + d_1 k$ y $q_2 = a_2 + b_2 i + c_2 j + d_2 k$, entonces $q_1 + q_2 = (a_1 + a_2) + (b_1 + b_2)i + (c_1 + c_2)j + (d_1 + d_2)k$
   - conjugado: si $q = a + bi + cj + dk$, entonces $q_c = a - bi - cj - dk$
   - escalado: si $q = a + bi + cj + dk$ y $p \in \mathbb{R}$, entonces $p \cdot q = (p \cdot a) + (p \cdot b)i + (p \cdot c)j + (p \cdot d)k$
   - traza: si $q = a + bi + cj + dk$, entonces $traza(q) = a + b + c + d$

   Se pide escribir una aplicación distribuida utilizando el *framework* Java-RMI que implemente las operaciones sobre cuaterniones indicadas Para ello, debe utilizar como elemento básico la siguiente interfaz (que estará en un fichero `ICuaternion.java`):

```
import java.rmi.*;
public interface ICuaternion extends Remote{
  public float[] sumCuaternion(float[] q1, float[] q2)
          throws RemoteException;
  public float[] conCuaternion(float[] q)
          throws RemoteException;
  public float[] xCuaternion(float[] q, float p)
          throws RemoteException;
  public float  tCuaternion(float[] q)
          throws RemoteException;
}
```

Especificaciones adicionales:

- No utilice paquetes (`package`);
- Servidor: código en `sCuaternion.java`; ejecución sobre el puerto por defecto; debe exportar cuatro objetos remotos distintos, que deben poseer diferentes nombres de servicio.
- Cliente: código en `cCuaternion.java`; comienza por imprimir la lista de servicios remotos, previa consulta al servidor; cada operación se realizará sobre uno de los diferentes objetos remotos; lee por teclados dos cuaterniones, pide al primer objeto servidor que los sume y muestra en pantalla los dos cuaterniones leídos y el cuaternión suma; a continuación, pide a un segundo objeto servidor conjugar el primer cuaternión y muestra el conjugado en pantalla; ahora, lee un número real, y escala el primer cuaternión con él utilizando un tercer objeto servidor, mostrando en pantalla el cuaternión escalado; finalmente, se imprime la traza del primer cuaternión, calculada por el cuarto objeto servidor. Las lecturas de datos deben ir acompañadas de mensajes impresos indicando claramente los datos que se piden.

2. (3.5 puntos) Utilizando la biblioteca de paso de mensajes MPJ-Express, escriba un programa en Java que efectúe aritmética matricial de matrices cuadradas de tipo `float` de acuerdo a las especificaciones que se indican, para las matrices A y B siguientes:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \; B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

   - No utilice paquetes (`package`);
   - El programa *master* (`rank=0`) determinará en su código las matrices indicadas y las enviará a tres procesos *slaves* (`rank=1, 2 ,3`) utilizando obligatoriamente comunicación con los métodos `Send/Recv`.
   - Los procesos slaves efectuarán respectivamente las siguientes operaciones sobre las dos matrices recibidas:
     - *Slave* con `rank=1`: suma las matrices y muestra el resultado en pantalla.
     - *Slave* con `rank=2`: multiplica las matrices y muestra el resultado en pantalla.
     - *Slave* con `rank=3`: transpone la matriz A y la envía de vuelta al *master*, que la presenta en pantalla.
   - Las impresiones de los resultados deben in acompañadas de texto que ilustre claramente el contenido que se imprime. Las matrices deben imprimirse obligatoriamente en formato de matriz (cuadrado de números).
   - Incluya en su programa como comentarios las órdenes de línea de comando necesarias para compilar y ejecutar su programa. Esta última debe incluir el valor del *flag* –`np` para crear el árbol de procesos que propone. **La ausencia de estas órdenes invalida por completo**

el ejercicio, con independencia de los criterios generales de corrección.

- Guarde el código en `cMat.java`.

3. (3.5 puntos) Dos sensores de infrarrojos proporcionan señales discretas $\zeta(i), \rho(i)$ de n componentes con intensidades variables enteras $\zeta(i), \rho(i) \in \mathbb{Z}_5 = \{0, 1, 2, 3, 4\}$. Estas señales, para poder ser utilizadas, necesitan ser integradas a una nueva señal $\chi : \mathbb{Z}_5^n \times \mathbb{Z}_5^n \to \mathbb{Z}_5^n$, de acuerdo con la siguiente transformada:

$$\chi(i) = (\zeta(i) + \zeta(i+1)) + (\rho(i) + \rho(i-1))$$

Desarrolle utilizando el lenguaje Java un programa que permita calcular la señal $\chi(i)$ de forma paralela mediante división del dominio de datos entre tareas paralelas de acuerdo al siguiente conjunto de especificaciones:

- No utilice paquetes (`package`);
- Guarde el código en `sensor.java`.
- Tareas paralelas modeladas mediante herencia de `Thread`.
- Condiciones de frontera para las señales $\zeta(i), \rho(i)$: nulas (vecinos de los extremos son cero).
- Señales de entrada y salida almacenadas respectivamente en arrays `senalEntradaZeta`, `senalEntradaRho` y `senalSalidaXi`.
- Dos modos de ejecución, a elegir por el usuario en un menú de opciones:
  - Modo de ejecución manual, donde:
    - El usuario introduce por teclado los datos siguientes: tamaño de las señales n, las propias señales $\zeta(i), \rho(i)$ y el número de hebras paralelas h que desea utilizar. Debe haber mensajes claros en pantalla de solicitud de los diferentes datos indicados.
    - El programa muestra en pantalla las señales originales $\zeta(i), \rho(i)$ y la señal transformada $\chi(i)$, en formato de vector de números.
  - Modo de ejecución automático: carga las señales $\zeta(i), \rho(i)$ con datos aleatorios, determina el número de tareas consultando el número de núcleos lógicos al sistema operativo, utiliza una señal de tamaño $n = 10^7$, y muestra en pantalla los tiempos de cálculo secuencial versus paralelo de la señal $\chi(i)$ en nanosegundos, con una salida igual a la siguiente, donde k, p son los tiempos obtenidos:
    - `Tiempo de ejecución secuencial:` k `nanosegundos`
    - `Tiempo de ejecución paralelo :` p `nanosegundos`