Programación Concurrente y de Tiempo Real
Grado en Ingeniería Informática
Examen Final de Prácticas (**MODELO C**)
Febrero de 2021

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. Se prohíbe el uso de apuntes, *pen-drives* y similares.
3. Todos los ficheros de código generados deben incluir un comentario que incluya nombre, apellidos y D.N.I.
4. 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.
5. **No se aceptarán bajo ningún concepto entregas efectuadas por medios diferentes a la tarea de subida, ni fuera del plazo de examen establecido. Por tanto, debe asegurarse de entregar sus productos en el plazo establecido y mediante la tarea habilitada a tal efecto.**

2. Criterios de Corrección

Veáse la ficha de la asignatura, apartado Sistema de Evaluación, subapartado Criterios Generales de Evaluación.

3. Enunciados de Examen

1. (TAD Cola Sincronizada con C++, 3.0 puntos) Deseamos disponer de una implementación basada en arrays del TAD cola que sea segura frente a concurrencia, utilizando para ello el lenguaje C++. Las operaciones que

la implementación debe soportar son las habituales de este TAD, y debe incluir la operación de imprimir la cola. Efectúe la implementación del TAD indicado de forma que los observadores y modificadores sean seguros frente a concurrencia. Incluya un programa principal que haga uso de una de estas colas, si se crea con un tamaño igual a diez componentes, y que tenga no menos de dos hebras que trabajen concurrentemente sobre la cola. Asegúrese de incluir en el programa principal impresiones de la cola que permitan comprobar de manera inequívoca la coherencia de la misma frente a accesos concurrentes. Guarde el código en `secQueue.cpp`. NOTA: el código debe compilar obligatoriamente con `g++ -std=c++11 -pthread secQueue.cpp`

2. (Procesamiento Distribuido con MPJ-Express, 3.5 puntos) Se desea escribir un programa utilizando MPJ-Express que efectúe el siguiente procesamiento sobre una matriz cuadrada de 4×4 elementos de números enteros positivos en el rango $[0, 1]$:

   - existirá un proceso master que enviará a cada proceso slave una columna de la matriz.
   - cada proceso slave efectuará la suma de los componentes de la columna recibida, y remitirá esa suma al proceso master; antes de hacerlo, imprimirá en pantalla el valor de la suma.
   - el proceso master efectuará a su vez el producto de los valores enviados por los procesos slaves, y la presentará en pantalla.
   - el proceso master cargará en su código los datos de la matriz.

   Guarde su código en un fichero llamado `sumMulMatDistribuida.java`. Este código deberá incluir de forma obligatoria un comentario donde se indicarán las órdenes de compilación y de ejecución con las cuales propone que su código sea procesado.

3. (Filtrado 1D con Java, 3.5 puntos) Considere una colección de valores en el intervalo [-4, 4], de tamaño 10^4 valores, que representamos por $x[n]$. Un filtrado unidimensional discreto de tales valores mediante un filtro $[l_0, l_1, l_2]$ proporciona una nueva señal $w[n]$ según la siguiente ecuación:

$$w[i] = (x[i-3] \cdot l_0) + (x[i] \cdot l_1) + (x[i+3] \cdot l_2)$$

   Se pide escribir un programa en Java que efectúe de forma paralela el filtrado de la colección de valores de acuerdo a las especificaciones siguientes:

   - tareas paralelas soportadas por implementación de la interfaz `Runnable`.
   - división manual de la nube de datos original entre ocho tareas paralelas.
   - lectura del filtro por teclado, con componentes en rango [-1, 1].
   - todos los datos son de tipo entero.
   - condición de frontera a imponer: cilíndrica.
   - los datos filtrados $w[n]$ deben mantenerse en el intervalo [-4, 4].

- procesamiento de las tareas mediante un ejecutor de clase `ThreadPoolExecutor`.
- dos modos de ejecución distintos, a elegir por el usuario:
  - automático: carga de la señal original de 10^4 valores con datos aleatorios en el rango indicado, y toma de tiempos con resolución de nanosegundo; el programa imprime el tiempo total de ejecución paralelo.
  - manual: el programa lee por teclado el tamaño de la señal original, y la propia señal; termina imprimiendo la señal original y la señal convoluta.

Guarde su código en `1DFilter.java`