# Instituto Politécnico Nacional

Escuela Superior de Cómputo


Desarrollo de Sistemas Distribuidos

Prof. **Benjamín Cruz Torres**

## Práctica No. 1
## Cuatro Relojes Independientes

Grupo: 4CV3

Equipo: 6

Integrantes:

1. Acosta Rosales Jair Sebastián
2. De Jesús López David
3. Galicia Vargas Gerardo
4. Martínez Marcos Luis Eduardo
5. Octaviano Lima Elvia Jaqueline

*Fecha: 28 de agosto de 2018*

# Práctica 1: Cuatro Relojes Independientes

Objetivo de la Práctica Que el alumno comprenda las ventajas de la programación concurrente en una computadora a través del uso de hilos (threads).

## Actividades

Desarrollar una vista con cuatro relojes digitales independientes de acuerdo a los siguientes requerimientos

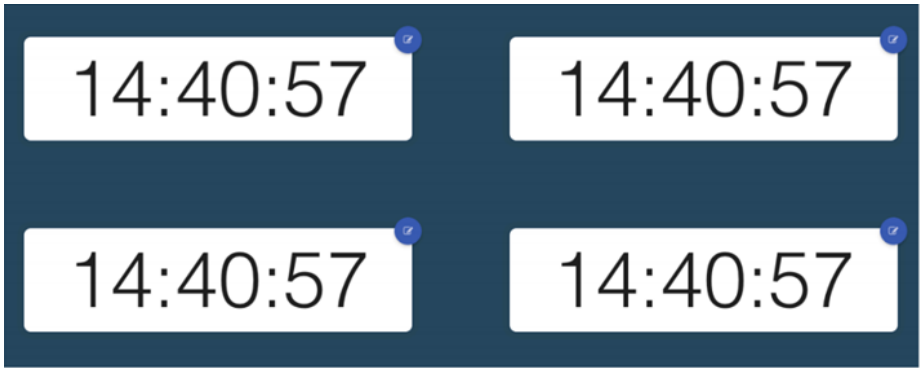



Figura 1. Cuatro relojes independientes

## Requerimientos funcionales

- Los cuatro relojes funcionan de forma independiente
- El usuario puede modificar la hora y los minutos de cada reloj.
- Inicialmente el segundero cambia cada segundo.
- El formato de hora es de 24 hrs.

## Requerimientos no funcionales

- Al pulsar el botón de modificar, el reloj correspondiente se detendrá.
- El modificar un reloj no afecta el funcionamiento de los otros tres.
- Al inicio uno de los relojes tendrá la hora local y los otros tres tendrán horas diferentes elegidas al azar.
- Se deberá cambiar la velocidad de actualización del segundero mediante código.

# Investigación

### *Socket*

"[…] Abstracción a través de la cual una aplicación puede enviar y recibir datos […] permite a una aplicación conectarse a la red y comunicarse con otras aplicaciones que están conectadas a la misma red. […]" (Calvert & Donahoo, 2008: 14).

"[…] Los procesos pueden estar ejecutándose en el mismo o en distintos sistemas, unidos mediante una red. […]" (*Ibíd.*).

## Dominios de comunicación

"[…] Los sockets se crean dentro de un dominio de comunicación […] dice dónde se encuentran los procesos que se van a intercomunicar.

Si los procesos están en el mismo sistema, el dominio de comunicación será AF_UNIX, si los procesos están en distintos sistemas y éstos se hallan unidos mediante una red TCP/IP, el dominio de comunicación será AF_INET. La idea original fue que se usase la misma interfaz también para distintas familias de protocolos. […]" (*Ibíd.*).

## Tipos de *sockets*

"[…] Diferentes tipos de sockets corresponden a diferentes familias de protocolos y diferentes pilas de protocolos dentro de una familia. […] Los principales tipos de sockets en TCP/IP actualmente son los *stream sockets* y *datagram sockets*. Los sockets de flujo usan TCP como el protocolo terminalterminal (con IP debajo) y así provee un servicio confiable de flujo de bytes. […] Los sockets de datagrama utilizan UDP (de nuevo, con IP debajo) y así provee un servicio de mejor esfuerzo que las aplicaciones pueden usar para enviar mensajes individuales de alrededor de 65500 bytes de longitud. […] Un socket TCP/IP es identificado como único por una dirección de Internet, un protocolo terminal-terminal (TCP o UDP) y un número de puerto. […]" (Calvert & Donahoo, 2008: 14).

## Tipos de *sockets* en el dominio *AF_INET*

### *Sockets Stream*

"[…] hacen uso del protocolo TCP, el cual nos provee un flujo de datos bidireccional, secuenciado, sin duplicación de paquetes y libre de errores. […]" (*Sockets*, 2008: 1).

### *Sockets Datagram*

"[…] hacen uso del protocolo UDP, el cual nos provee un flujo de datos bidireccional, pero los paquetes pueden llegar fuera de secuencia, pueden no llegar o contener errores.
Por lo tanto, el proceso que recibe los datos debe comprobar la secuencia, eliminar duplicados y asegurar la integridad.
Se llaman también "sockets sin conexión", porque no hay que mantener una conexión activa, como en el caso de sockets stream.
Son utilizados para transferencia de información paquete por paquete.
[Los programas] implementan un protocolo encima de UDP que realiza control de errores. […]" (*Ibíd.*).

### Sockets Raw

"[…] son para el usuario más común, provistos principalmente para aquellos interesados en desarrollar nuevos protocolos de comunicación o para hacer uso de facilidades ocultas de un protocolo existente. […]" (*Ibíd.*).

El envío de archivos a través de la red es una característica importante para la gran mayoría de las aplicaciones que hoy día se utilizan (blogs, redes sociales, mensajería instantánea, declaración de impuestos, educación en línea, etc.), sin embargo, no todas las aplicaciones disponibles permiten el envío de archivos de gran tamaño (p.e. El correo electrónico no permite enviar archivos de más de 10 o 25 MB). Esto hace necesario el desarrollo de aplicaciones que permitan transferir archivos sin importar el tamaño de éstos.

Un socket queda definido por un par de direcciones IP local y remota, un protocolo de transporte y un par de números de puerto local y remoto. Para que dos programas puedan comunicarse entre sí es necesario que se cumplan ciertos requisitos:

- Que un programa sea capaz de localizar al otro.
- Que ambos programas sean capaces de intercambiarse cualquier secuencia de octetos, es decir, datos relevantes a su finalidad.

Para ello son necesarios los tres recursos que originan el concepto de socket:

- Un protocolo de comunicaciones, que permite el intercambio de octetos.
- Un par de direcciones del Protocolo de Red (Dirección IP, si se utiliza el Protocolo TCP/IP), que identifica la computadora de origen y la remota.
- Un par de números de puerto, que identifica a un programa dentro de cada computadora.

Los sockets permiten implementar una arquitectura cliente-servidor o peer to peer. La comunicación debe ser iniciada por uno de los programas que se denomina programa cliente. El segundo programa espera a que otro inicie la comunicación, por este motivo se denomina programa servidor.

Un socket es un proceso o hilo existente en la máquina cliente y en la máquina servidora, que sirve en última instancia para que el programa servidor y el cliente lean y escriban la información. Esta información será la transmitida por las diferentes capas de red.

## Introducción

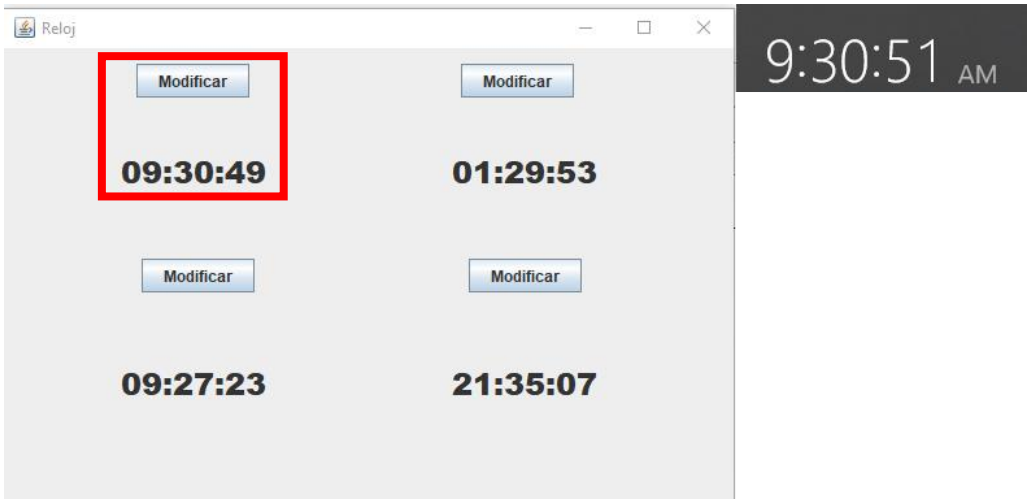
Un *thread* es un único flujo de control dentro de un programa. Algunas veces es llamado contexto de ejecución porque cada *thread* debe tener sus propios recursos, como el *program counter* y el *stack* de ejecución, como el contexto de ejecución. Sin embargo, todo *thread* en un programa aún comparte muchos recursos, tales como espacio de memoria y archivos abiertos. Los *threads* también son llamados procesos livianos (*lightweight process*).

Debido a que cada hilo tiene su propio contexto, en la presente práctica se utilizan para desarrollar una aplicación de cuatro relojes ejecutándose concurrentemente, cada uno modificable de forma independiente a los demás y con la posibilidad de realizar cálculos y actualizar sus valores en intervalos distintos de tiempo.

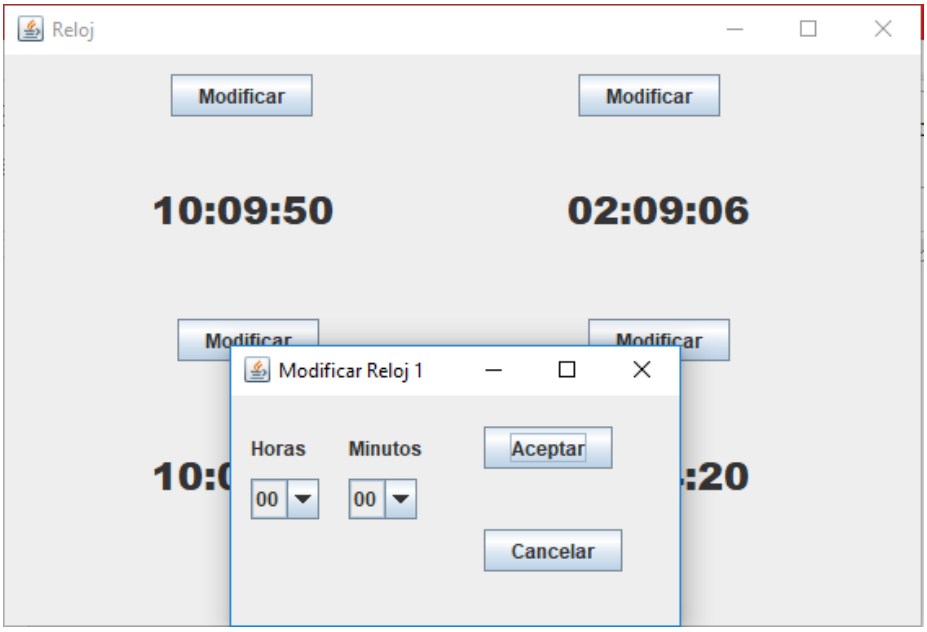
## Desarrollo de la práctica

Tras investigar cómo se utilizan los hilos en Java para actualizar periódicamente el valor de determinadas variables, se elaboró la interfaz a través del entorno gráfico de NeatBeans; posteriormente se fueron agregando las funcionalidades:

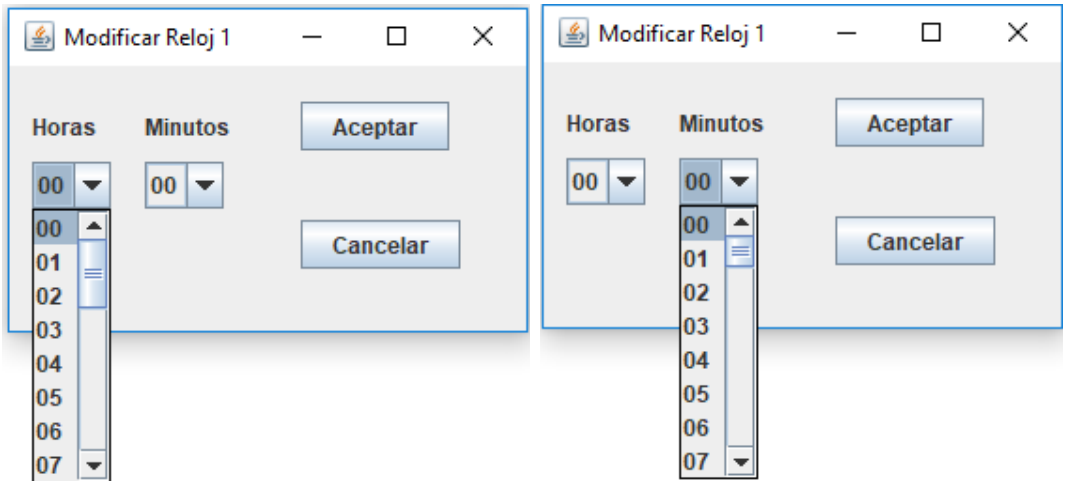
- Instancias de Relojes, los cuales contienen las variables propias del tiempo del reloj (horas, minutos y segundos)
- Instancias de una interfaz auxiliar para ingresar cambios en los valores de horas y minutos, cada instancia ligada a un reloj en particular.



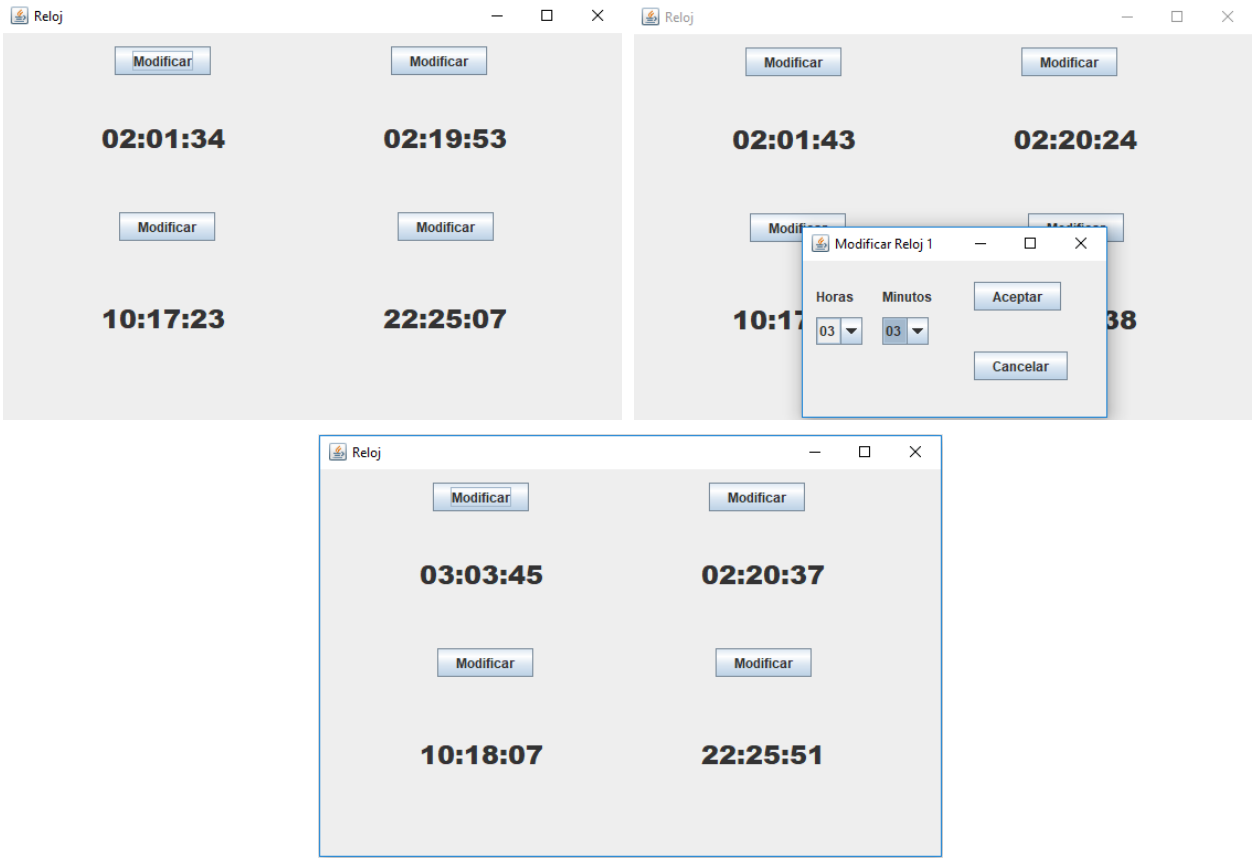

Como se observa en la captura de pantalla, cada reloj se inicializa con una hora determinada, tras lo cual, se van actualizando cada segundo y en específico el reloj marcado toma la hora del sistema para su inicialización.



Cada reloj se puede modificar al dar clic en su correspondiente botón de Modificar, lo cual abre una nueva ventana de Modificar Reloj.




En cada ventana de Modificar Reloj se puede ingresar tanto un valor para Horas, como para Minutos. Tras dar clic en el botón Aceptar, los valores seleccionados se transferirán a la ventana inicial de Reloj.



Finalmente, entre las variables globales del programa se encuentran algunas relativas al tiempo de retardo entre cada ejecución de las operaciones de cada hilo (parámetro que se le envía al método *sleep()* de cada hilo), por lo que, al modificar cada uno de estos valores (inicialmente en 1000, que corresponde a mil milisegundos, es decir, un retardo de un segundo) se modifica el tiempo en que se vuelven a ejecutar los cálculos de determinado hilo.

## Conclusiones

Los hilos permiten la ejecución de bloques de código o instrucciones de forma concurrente, pudiendo generar resultados y actualizaciones en interfaces gráficas en tiempos teóricamente iguales o bien en intervalos de tiempo definidos, con todos los hilos en ejecución a un mismo tiempo.

Gracias a este funcionamiento concurrente, sistemas más complejos pueden compartir recursos y realizar tareas de forma distribuida, es decir, la concurrencia es una de las funcionalidades básicas para implementar sistemas distribuidos.

## Bibliografía

L.-Calvert, K. & J.-Donahoo, M. (2008). *TCP/IP Sockets in Java: Practical Guide for Programmers*. (2ª Ed.). E.U.A.: Ed. Morgan Kauffman Publishers. ISBN: 1-55860-685-8.

*Sockets*. (2008). Buenos Aires: Universidad Tecnológica Nacional Recuperado de: http://www2.electron.frba.utn.edu.ar/~mdoallo/descargas/redes.pdf