Práctica 2: Resolución de problemas de sincronización mediante esperas activas.

Programación de Sistemas Concurrentes y Distribuidos

Dpto. de Informática e Ingeniería de Sistemas,
Grado de Ingeniería Informática
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

1. Objetivos

En esta práctica se estudiará la resolución de problemas de sincronización (por condición y exclusión mutua) mediante esperas activas.

En concreto los objetivos de esta práctica son:

- comprender y profundizar en la sincronización de procesos mediante esperas activas,
- implementar sincronización mediante el uso de instrucciones de tipo `test-and-set`,
- y profundizar en el modelo de concurrencia de C++.

2. Trabajo previo a la sesión en el laboratorio

Antes de la correspondiente sesión en el laboratorio, cada estudiante deberá leer el enunciado, analizar los problemas que en él se proponen y realizar un diseño previo de las soluciones sobre las que va a trabajar. Los resultados de su trabajo de análisis y diseño los tendrá que expresar en un documento que entregará en papel a los profesores antes del inicio de la sesión (en el mismo laboratorio). El documento debe contener como mínimo el nombre completo y el NIP del estudiante y, para cada ejercicio,

- la descripción de los datos compartidos, enumeración de los procesos que los comparten, e indicación de si requieren sincronización o no.
- un esbozo de alto nivel del código de los procesos indicando las zonas que están afectadas por la sincronización (ya sea zona de exclusión mutua o espera sincronizada).

La entrega de este documento es un pre-requisito para la realización y evaluación de la práctica.

3. Ejercicio 1

Considérese el entorno de datos siguientes, preparado para trabajar con matrices de 512^2 datos de tipo $float$

```
const int N = 512;
typedef float Mat[N][N];      //matriz NxN
typedef float Vect[N];        //vector de dim. N

//-----------------------------------------------------
//Pre:  true
//Post: mod_Vect(x) = sqrt(sum_{α=0}^{N-1} v[α]^2)
//Com:  Devuelve el módulo del vector

float mod_Vect(const Vect x);
//-----------------------------------------------------
//Pre:  0 ≤ f1 ≤ f2 ≤ N − 1
//Post: ∀α ∈ [f1, f2].pMV[α] = sum_{β=0}^{N-1} M[α][β] · x[β]
//Com:  Calcula el producto, parcial, matriz por vector
//      (solo considera las filas en el rango [f1, f2])

void prod_Mat_Vect(const Mat A, const Vect x,
                   const int f1, const int f2, Vect& pMV);
```

El ejercicio pide calcular el módulo del vector resultante de multiplicar una matriz por un vector. Para ello, 16 procesos distintos llevan a cabo el cálculo del producto de la matriz por el vector: cada proceso se implementará mediante un `thread` y realizará el producto parcial de 32 filas. Una vez que todos han acabado, un proceso `informador` calcula su módulo y lo muestra por la salida estándar. Con el objetivo de obtener e informar del resultado tan pronto como sea posible, el programa principal ejecutará el `join` con los 17 threads *una vez el proceso informador haya mostrado el resultado buscado.*

Es preciso llevar a cabo un análisis detallado de los aspectos de sincronización (datos compartidos, posibles interferencias, esperas necesarias, etc.), plantear un diseño de la solución en consecuencia y llevar a cabo una implementación utilizando las esperas activas donde sea necesaria la sincronización.

El fichero fuente que contenga el procedimiento `main` de este ejercicio deberá llamarse `main_p2_e1.cpp`. El valor de este ejercicio será el 75 % de la nota total. **La entrega de esta parte es obligatoria**.

4. Ejercicio 2

Se trata de hacer una segunda versión del programa anterior en el que además de realizar la tarea anterior queremos saber cuál ha sido el thread más lento. Para ello, cada thread, además de realizar el trabajo del ejercicio anterior, debe actualizar los parámetros `tMax` y `idMasLento` de manera que, una vez terminado el programa, las variables que el programa principal pasa por referencia a los threads contengan el tiempo máximo requerido por los threads, así como el identificador del que más haya tardado[1].

[1]Se puede consultar `http://www.cplusplus.com/reference/thread/thread/` para ver cómo acceder al identificador de un thread.

Dado que todos lo threads han de acceder concurrentemente a los parámetros `tMax` y `idMasLento`, se le suministra también una variable, por referencia, del tipo `std::atomic_flag`, que permite utilizar la instrucción `test-and-set` para asegurar el acceso en exclusión mutua, tal y como se ha mostrado en clase[2].

```
void prod_Mat_Vect(const Mat A, const Vect x,
                   const int f1, const int f2, Vect& pMV.
                   std::chrono::nanoseconds& tMax,
                   std::thread::id& idMasLento,
                   std::atomic_flag& tas);
```

El fichero fuente que contenga el procedimiento `main` de este ejercicio deberá llamarse `main_p2_e2.cpp`. El valor de este ejercicio será el 25 % de la nota total. **La entrega de esta parte es opcional**.

5. Entrega de la práctica

La práctica se realizará de forma individual. Cuando se finalice se debe entregar un fichero comprimido `practica2_miNIP.zip` (donde `miNIP` es el NIP del autor de los ejercicios) con el siguiente contenido:

1. Los ficheros correspondientes al ejercicio obligatorio: al menos deberá contener `main_p2_e1.cpp` y `Makefile_p2_e1` (de manera que la ejecución de `make -f Makefile_p2_e1` generará el ejecutable `main_p2_e1`). Junto con este enunciado se ha depositado un ejemplo de fuente para la herramienta *makefile*.

2. En caso de hacer la parte optativa, los ficheros correspondientes: al menos deberá contener `main_p2_e2.cpp` y `Makefile_p2_e2` (de manera que la ejecución de `make -f Makefile_p2_e2` generará el ejecutable `main_p2_e2`)

3. Un fichero de texto denominado `autor.txt` que contendrá el NIP, los apellidos y el nombre del autor de la práctica en las primeras líneas del fichero, de manera análoga a la primera práctica.

   También deberá contener:

   - una descripción de las principales dificultades encontradas para la realización de la práctica
   - para cada ejercicio desarrollado, el listado de los nombres de los ficheros fuente que conforman la solución solicitada así como la forma de compilarlos para obtener el ejecutable correspondiente.

Para la entrega del fichero `.zip` se utilizará el comando `someter` en la máquina `hendrix.cps.unizar.es`. Los alumnos pertenecientes a grupos de prácticas cuya segunda sesión de prácticas se celebra el día 16 de octubre de 2019 deberán someter la práctica no más tarde del día 30 de octubre de 2019 a las 23:59. Los alumnos pertenecientes a grupos de prácticas cuya primera sesión de prácticas se celebra el día 23 de octubre de 2019 deberán someter la práctica no más tarde del día 5 de noviembre de 2019 a las 23:59.

[2] Véase `http://www.cplusplus.com/reference/atomic/atomic_flag/test_and_set/` para su especificación y ejemplo de uso.

Hay que asegurarse de que la práctica funciona correctamente en los ordenadores del laboratorio (vigilar aspectos como los permisos de ejecución, juego de caracteres utilizado en los ficheros, etc.). También es importante someter código limpio (donde se ha evitado introducir mensajes de depuración que no proporcionan información al usuario). El tratamiento de errores debe ser adecuado, de forma que si se producen debería informarse al usuario del tipo de error producido. Además se considerarán otros aspectos importantes como calidad del diseño del programa, adecuada documentación de los fuentes, correcto formateado de los fuentes, etc.

Para el adecuado formateado de los fuentes, es conveniente seguir unas pautas. Hay varias, y es posible que podáis configurar el entorno de desarrollo para cualquiera de ellas. Una posible, sencilla de seguir, es la "Google C++ Style Guide", que se puede encontrar en

`https://google.github.io/styleguide/cppguide.html`

Alternativamente, cualquiera que uséis en otras asignaturas de programación.