Práctica 13
Hilos

(Rev 1.0)

1 Objetivos

Que el alumno se familiarice con el concepto de hilos (*threads*), su implementación en C++ 11, y su aplicación en problemas simples.

2 Pre-laboratorio

2.1 Cuestionario

(Las preguntas de los cuestionarios vendrán en sus exámenes parciales del salón.)

1. ¿Qué es la concurrencia?
2. ¿Qué son los hilos?
3. ¿Qué son los Posix Threads?
4. ¿Los Posix Threads están implementados en Windows?
5. Mencione algunos mecanismos de sincronización de hilos.
6. ¿Qué son los deathlocks y las race-conditions?

2.2 Actividades

1.

3 Teoría

3.1

4 Actividades

Actividad 1: Primeros pasos con los hilos

En esta actividad Ud. se familiarizará con el uso simple de los hilos.

1. Ejecute el programa ***test_Hilos.cpp***. Comente los resultados. No olvide que la instrucción de compilación debe incluir a la biblioteca pthread:

```
g++ -Wall -std=c++11 -o test_Hilos.out test_Hilos.cpp -lpthread
```

2. Realice las siguientes modificaciones, ejecute y comente los resultados.

```
/*-------------------------------------------------------------------
 *  Driver program
 *-------------------------------------------------------------------*/
int main(void)
{

  std::cout << "Arrancando func1() en el fondo"
    << " y func2 () en primer plano: " << std::endl;

  std::future<int> result1 {std::async (func1)};

  func2 ();

#if 0
  int result2 = func2 ();

  int result = result1.get () + result2;

  std::cout << "\nEl resultado de func1() + func2() es: "
    << result << std::endl;
#endif

  return 0;
}
```

De los resultados anteriores, ¿para qué cree Ud. que se utiliza el método `.get()` (también existe un método que hace algo similar, `.wait()`)?

Actividad 2: Generando y consumiendo datos

En esta actividad Ud. va a escribir un pequeño programa, ***test_sincro.cpp***, donde una función generará datos (el usuario introducirá una cantidad de números), mientras que otra función los consumirá (los imprimirá en la pantalla).

1. En este tipo de aplicaciones se requiere de un búfer de datos – un área de memoria temporal donde se escriben y leen datos –. Este búfer será un vector de enteros de tamaño 5. Como excepción a la regla sobre variables globales, el búfer deberá ser global (ya que será compartido por diversos hilos).

2. De igual forma debemos utilizar un mecanismo de sincronización entre los hilos para que la escritura y lectura se haga de manera ordenada. Para ello tendremos otra variable global, `bool isReady`, que tomará el valor `false` si el búfer no se ha llenado, y `true` si el búfer se llenó[1]. Vea los siguientes pasos.

3. Basándose en la esencia del programa de la actividad anterior, escriba una función `Leer()` que lea 5 números desde el teclado y vaya guardándolos en el búfer. Antes de iniciar la lectura ponga la bandera `isReady` a `false`. Cuando termine con la lectura, ponga la bandera a `true`.

4. Escriba una función `Imprimir()` que imprima el búfer de datos. Esta función encuestará[2] a la bandera `isReady`, y mientras sea `false` no hará nada; pero en cuanto ésta tome el valor `true` será que imprimirá el búfer. Al terminó deberá poner la bandera otra vez a `false`.

5. Ud. decida si en su driver program realiza una o más secuencias de lectura y escritura. Pero para empezar bastará con una sola (para probar que todo va bien).

> Entregables: El archivo test_sincro.cpp.

1 Este es el mecanismo más simple de sincronización, y es una variante del semáforo binario.
2 El término en inglés es *polling*, y significa que se estará preguntando constantemente por la variable hasta que ésta cambie de valor. Recuerde que con hilos, el cambio de estado de la variable es asíncrono, es decir, fuera del flujo normal del programa.