Informática Industrial

Concurrencia en C++11

Concurrencia

- Actualmente casi todos los procesadores tienen dos o más núcleos. Estos núcleos realizan el procesamiento de las tareas que el sistema operativo les envía.
- En tareas de procesamiento intensivo podemos aprovecharnos del trabajo en paralelo de todos los núcleos con la programación concurrente.
- En este tema veremos como programar concurrentemente las tareas a realizar en cada núcleo mediante *threads* o hilos de ejecución.

Concurrencia

- En realidad emplearemos la programación concurrente incluso cuando dispongamos de un único procesador con un solo núcleo. Permite disponer de otros procesos listos cuando el proceso en ejecución se bloquea (p.ej. Por una espera en una operación de entrada y salida)
- No todo son ventajas en la concurrencia. Hay que programar las tareas que se realizarán concurrentemente teniendo cuidado con las condiciones de carrera y sincronización de los hilos.

Concurrencia en C++11

- C++11 introdujo una nueva biblioteca de hilos (*thread*).
- Esta biblioteca incluye llamadas al sistema para **crear, lanzar y sincronizar hilos**.
- En este tema veremos con ejemplos las características fundamentales de la biblioteca.
- Para compilar los ejemplos es necesario un compilador C++11.
- Activar la opción **-std=c++0x** o **-std=c++11**.

Hilos en C++11 Crear y lanzar un hilo

Tenemos que incluir una nueva cabecera: <thread>

Creamos un hilo y comienza a ejecutar la función hola.

```
#include <thread>
#include <iostream>

using namespace std;
void hola()
{
    cout << "Hola mundo " << endl;
}

int main()
{
    thread hilo1(hola); //crea y arranca un nuevo hilo llamado hilo1
    hilo1.join();       //main espera a que termine hilo1

    return 0;
}
```

Hilos en C++11 join()

```cpp
#include <thread>
#include <iostream>

using namespace std;
void hola()
{
    cout << "Hola mundo " << endl;
}

int main()
{
    thread hilo1(hola); //crea y arranca un nuevo hilo llamado hilo1
    hilo1.join();       //main espera a que termine hilo1

    return 0;
}
```

join() es para que main espere a que el hilo1 termine.
main estará ocioso hasta que termine hilo1.

```
Hola mundo

Process returned 0 (0x0)   execution time : 0.010 s
Press any key to continue.
```

Hilos en C++11 detach()

detach: separa la ejecución del hilo de la ejec.del main permitiendo continuar la ejecución independientemente

```
#include <thread>
#include <iostream>

using namespace std;
void hola()
{
    cout << "Hola mundo " << endl;
}

int main()
{
    thread hilo1(hola); //crea y arranca un nuevo hilo llamado hilo1
    hilo1.detach();     //Separa la ejecución del hilo de la ejec.del main
                        //permitiendo continuar la ejecución independientemente


    return 0;
}
```

Al no esperar main no da tiempo a que se produzca la salida por pantalla del hilo1

```
Process returned 0 (0x0)   execution time : 0.048 s
Press any key to continue.
```

Hilos en C++11 Paso de Parámetros al hilo

```cpp
#include <iostream>
#include <thread>
#include <string>

using namespace std;

void imprime(string msg)
{
    cout << "En funcion hilo. ";
    cout << "El mensaje es: " << msg << endl;
}

int main()
{
    string mensaje = "Informatica Industrial";
    thread hilo(imprime, mensaje);
    cout << "Hilo main. El mensaje es: " << mensaje << endl;
    hilo.join();
    return 0;
}
```

Creamos un hilo que recibirá un *string* como parámetro

```
Hilo main. El mensaje es: Informatica Industrial
En funcion hilo. El mensaje es: Informatica Industrial
```

Hilos en C++11 Paso de Parámetros al hilo

```cpp
#include <iostream>
#include <thread>
using namespace std;

void imprime()
{
    cout << "Este es el primer hilo y solo imprime este mensaje en pantalla" << endl;
}

void multiplicax2(int n)
{
    int doble;
    doble= 2 * n;
    cout << "Este es el segundo hilo" << endl;
    cout << "El doble de "<< n << " es " << doble << endl;
}

int main()
{
    int a,b;
    thread hilo1(imprime);          // genera un nuevo hilo que llama a la funcion imprime()
    thread hilo2(multiplicax2,7);   // genera un nuevo hilo que llama a la funcion multiplicax2(7)

    cout << "Los tres hilos ejecutandose concurrentemente...\n";
    // sincronizamos los hilos para que main espere a que terminen los otros
    hilo1.join();                // espera a que finalice hilo1
    hilo2.join();                // espera a que finalice hilo2

    cout << "Ya han acabado hilo1 e hilo2\n";

    return 0;
}
```

Creamos dos hilos. Uno sin parámetros y otro con un parámetro

Hilos en C++11 Paso de Parámetros al hilo

```
#include <iostream>
#include <thread>
using namespace std;

void imprime()
{
    cout << "Este es el primer hilo y solo imprime este mensaje en pantalla" << endl;
}

void multiplicax2(int n)
{
    int doble;
    doble= 2 * n;
```

```
Este es el primer hilo y solo imprime este mensaje en pantallaLos tres hilos eje
cutandose concurrentemente...
Este es el segundo hilo

El doble de 7 es 14
Ya han acabado hilo1 e hilo2
```

```
    thread hilo1(imprime);          // genera un nuevo hilo que llama a la funcion imprime()
    thread hilo2(multiplicax2,7);   // genera un nuevo hilo que llama a la funcion multiplicax2(7)

    cout << "Los tres hilos ejecutandose concurrentemente...\n";
    // sincronizamos los hilos para que main espere a que terminen los otros
    hilo1.join();                   // espera a que finalice hilo1
    hilo2.join();                   // espera a que finalice hilo2

    cout << "Ya han acabado hilo1 e hilo2\n";

    return 0;
}
```

Hilos en C++11

¿Cuántos hilos se pueden lanzar concurrentemente en una máquina?

```
#include <iostream>
#include <thread>


using namespace std;
int main()
{
    cout << "Numero de hilos = "
              <<  thread::hardware_concurrency() << endl;
    return 0;
}
```

```
Numero de hilos = 8

Process returned 0 (0x0)   execution time : 0.124 s
Press any key to continue.
```

Hilos en C++11

Condiciones de carrera

```
using namespace std;

int total=0;

void incrementaTotal()
{
    for (int i = 0; i < 100000; i++)
        total++;
}

int main()
{
    thread hilo1(incrementaTotal),hilo2(incrementaTotal);
    hilo1.join();
    hilo2.join();
    cout << "total: " << total << "\n";
    return 0;
}
```

¿Qué resultado proporciona? ¿200.000?

Hilos en C++11

Condiciones de carrera

```
using namespace std;

int total=0;

void incrementaTotal()
{
    for (int i = 0; i < 100000; i++)
        total++;
}

int main()
{
    thread hilo1(incrementaTotal),hilo2(incrementaTotal);
    hilo1.join();
    hilo2.join();
    cout << "total: " << total << "\n";
    return 0;
}
```

¿Qué resultado proporciona? 106.071, 113.070, 111.524,…

Hilos en C++11

Condiciones de carrera

```
#include <iostream>
#include <thread>
#include <mutex>

using namespace std;

mutex mu;
int total=0;

void incrementa()
{
    for (int i = 0; i < 100000; i++)
    {
        mu.lock();
        total++;
        mu.unlock();
    }
}

int main()
{
    thread hilo1(incrementa),hilo2(incrementa);
    hilo1.join();
    hilo2.join();
    cout << "total: " << total << "\n";
    return 0;
}
```

Utilizaremos semáforos mutex *(mutual exclusion)*

Tenemos que regular el acceso a la variable total porque los incrementos no son operaciones atómicas.

Ahora siempre proporciona el resultado 200.000.

Ojo! Los mutex ralentizan pues hacen la ejecución de las secciones secuencial