Seminario 1: Programación multihebra y semáforos.

1. Cálculo concurrente de una integral (de forma contigua)

```
// ---------------------------------------------------------------------
// Sistemas concurrentes y Distribuidos.
// Seminario 1. Programación Multihebra y Semáforos.
//
// Ejemplo 9 (ejemplo9.cpp)
// Calculo concurrente de una integral.
//
// Se completa el problema de la integral para calcular PI, de forma contigua.
// ---------------------------------------------------------------------
#include <iostream>
#include <iomanip>
#include <chrono>  // incluye now, time\_point, duration
#include <future>
#include <vector>
#include <cmath>

using namespace std ;
using namespace std::chrono;

const long m  = 1024l*1024l*1024l, // número de muestras (del orden de mil millones)
n  = 4  ;              // número de hebras concurrentes (divisor de 'm')


// ---------------------------------------------------------------------
// evalua la función $f$ a integrar ($f(x)=4/(1+x^2)$)
double f( double x )
{
return 4.0/(1.0+x*x) ;
}
// ---------------------------------------------------------------------
// calcula la integral de forma secuencial, devuelve resultado:
double calcular_integral_secuencial(  )
{
double suma = 0.0 ;                    // inicializar suma
for( long j = 0 ; j < m ; j++ )          // para cada $j$ entre $0$ y $m-1$:
{  const double xj = double(j+0.5)/m ;     //      calcular $x_j$
suma += f( xj );                    //      añadir $f(x_j)$ a la suma actual
}
```

```
return suma/m ;                         // devolver valor promedio de $f$
}

// ---------------------------------------------------------------------------
// función que ejecuta cada hebra: recibe $i$ ==índice de la hebra, ($0\leq i<n$)
double funcion_hebra( long i )
{
double suma = 0.0;
int puntos = m/n;    // Nº puntos que calcula cada hebra
//    if(i == 0){
//        for(int j = i; j < puntos; j++){
//            const double xj = double(j+0.5)/m;
//            suma += f( xj );
//        }
//   }else{
//        for(int j = (i-1)+ puntos; j < puntos+i; j++){
//            const double xj = double(j+0.5)/m;
//            suma += f( xj );
//   }
// Más eficiente:
int inicio = puntos*i;  //A partir de que punto comienza a calcular.
for(int j = inicio; j < puntos+inicio; j++){
const double xj = double(j+0.5)/m;
suma += f( xj );
}

return suma/m;
}

// ---------------------------------------------------------------------------
// calculo de la integral de forma concurrente
/*
El vector futuros debe estar declarado con el tipo de dato correcto.
En este caso, debería ser std::future<double> ya que funcion_hebra
devuelve un double.
*/
double calcular_integral_concurrente( )
{
double suma = 0.0;
future<double> futuros[n];
for (long i = 0; i < n; i++){
futuros[i] = async( launch::async, funcion_hebra, i);
```

```
}

for (int i = 0; i < n; i++)
suma += futuros[i].get();

return suma;

}
// ------------------------------------------------------------------------

int main()
{

time_point<steady_clock> inicio_sec  = steady_clock::now() ;
const double             result_sec  = calcular_integral_secuencial(  );
time_point<steady_clock> fin_sec     = steady_clock::now() ;
double x = sin(0.4567);
time_point<steady_clock> inicio_conc = steady_clock::now() ;
const double             result_conc = calcular_integral_concurrente(  );
time_point<steady_clock> fin_conc    = steady_clock::now() ;
duration<float,milli>    tiempo_sec  = fin_sec  - inicio_sec ,
tiempo_conc = fin_conc - inicio_conc ;
const float              porc        = 100.0*tiempo_conc.count()/tiempo_sec.count() ;


constexpr long double pi = 3.14159265358979323846l ;

cout << "Número de muestras (m)   : " << m << endl
<< "Número de hebras (n)     : " << n << endl
<< setprecision(18)
<< "Valor de PI              : " << pi << endl
<< "Resultado secuencial     : " << result_sec  << endl
<< "Resultado concurrente    : " << result_conc << endl
<< setprecision(5)
<< "Tiempo secuencial        : " << tiempo_sec.count()  << " milisegundos. " << endl
<< "Tiempo concurrente       : " << tiempo_conc.count() << " milisegundos. " << endl
<< setprecision(4)
<< "Porcentaje t.conc/t.sec. : " << porc << "%" << endl;
}
```

Seminario 1: Programación multihebra y semáforos.

2. Cálculo concurrente de una integral (de forma entrelazada)

```
// ----------------------------------------------------------------------------
// Sistemas concurrentes y Distribuidos.
// Seminario 1. Programación Multihebra y Semáforos.
//
// Ejemplo 9 (ejemplo9.cpp)
// Calculo concurrente de una integral.
//
// Se completa el problema de la integral para calcular PI, de forma entrelazada.
// ----------------------------------------------------------------------------
#include <iostream>
#include <iomanip>
#include <chrono>  // incluye now, time\_point, duration
#include <future>
#include <vector>
#include <cmath>

using namespace std ;
using namespace std::chrono;

const long m  = 1024l*1024l*1024l, // número de muestras (del orden de mil millones)
n  = 4  ;              // número de hebras concurrentes (divisor de 'm')



// ----------------------------------------------------------------------------
// evalua la función $f$ a integrar ($f(x)=4/(1+x^2)$)
double f( double x )
{
return 4.0/(1.0+x*x) ;
}
// ----------------------------------------------------------------------------
// calcula la integral de forma secuencial, devuelve resultado:
double calcular_integral_secuencial(  )
{
double suma = 0.0 ;                    // inicializar suma
for( long j = 0 ; j < m ; j++ )          // para cada $j$ entre $0$ y $m-1$:
{  const double xj = double(j+0.5)/m ;     //      calcular $x_j$
suma += f( xj );                    //      añadir $f(x_j)$ a la suma actual
}
```

```
return suma/m ;                    // devolver valor promedio de $f$
}

// ---------------------------------------------------------------------
// función que ejecuta cada hebra: recibe $i$ ==índice de la hebra, ($0\leq i<n$)
double funcion_hebra( long i )
{
double suma = 0.0;
for(int j = i; j < m; j+=n){
const double xj = double(j+0.5)/m;
suma += f( xj );
}
return suma/m;
}

// ---------------------------------------------------------------------
// calculo de la integral de forma concurrente
/*
El vector futuros debe estar declarado con el tipo de dato correcto.
En este caso, debería ser std::future<double> ya que funcion_hebra
devuelve un double.
*/
double calcular_integral_concurrente( )
{
double suma = 0.0;
future<double> futuros[n];
for (long i = 0; i < n; i++){
futuros[i] = async( launch::async, funcion_hebra, i);
}

for (int i = 0; i < n; i++)
suma += futuros[i].get();

return suma;

}
// ---------------------------------------------------------------------

int main()
{

time_point<steady_clock> inicio_sec  = steady_clock::now() ;
```

```
const double             result_sec  = calcular_integral_secuencial(  );
time_point<steady_clock> fin_sec     = steady_clock::now() ;
double x = sin(0.4567);
time_point<steady_clock> inicio_conc = steady_clock::now() ;
const double             result_conc = calcular_integral_concurrente(  );
time_point<steady_clock> fin_conc    = steady_clock::now() ;
duration<float,milli>    tiempo_sec  = fin_sec  - inicio_sec ,
tiempo_conc = fin_conc - inicio_conc ;
const float              porc        = 100.0*tiempo_conc.count()/tiempo_sec.count() ;


constexpr long double pi = 3.14159265358979323846l ;

cout << "Número de muestras (m)   : " << m << endl
<< "Número de hebras (n)     : " << n << endl
<< setprecision(18)
<< "Valor de PI              : " << pi << endl
<< "Resultado secuencial     : " << result_sec  << endl
<< "Resultado concurrente    : " << result_conc << endl
<< setprecision(5)
<< "Tiempo secuencial        : " << tiempo_sec.count()  << " milisegundos. " << endl
<< "Tiempo concurrente       : " << tiempo_conc.count() << " milisegundos. " << endl
<< setprecision(4)
<< "Porcentaje t.conc/t.sec. : " << porc << "%" << endl;
}
```