```
// ---------------------------------------------------------------------------
// Sistemas concurrentes y Distribuidos.
// Seminario 1. Programación Multihebra y Semáforos.
//
// Ejemplo 9 (ejemplo9.cpp)
// Calculo concurrente de una integral. Plantilla para completar.
//
// Historial:
// Creado en Abril de 2017
// ---------------------------------------------------------------------------

#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, ($0\leq i<n$)
 * Forma 1 de calcular la hebra
 * El bucle recorre desde el valor que recibe (i), hasta m y se amplía de n en n.
 * En la variable suma se almacena el valor de la suma de j entre m.
 */
double funcion_hebra(long i){
    double suma = 0.0;
    for(unsigned long j=i; j < m; j+=n)
        suma += f((j+double(0.5))/m);
    return suma;
}
/**
 * Forma 2 de calcular la hebra
 * Planteo el for de otra forma, yendo desde m/n multiplicado por el valor de i
(argumento de la función), hasta m/n multiplicado por i+1.
 * En la variable suma se almacena el valor de la suma de j entre m.
 */
double funcion_hebra(long i){
    double suma=0.0;
    for(unsigned long j = ((m/n)*i) + 1; j<(m/n)*(i+1); j++){
        suma += f((j+double(0.5))/m);
    }
    return suma;
}
```

```
/**
 * Cálculo de forma concurrente de una integral.
 * Uso un vector de futuros donde almacenaré dentro de un bucle los distintos valores del
número PI en distintas posiciones del vector.
 * Sumaré los valores obtenidos y almacenados en el vector en la variable total.
 */
double calcular_integral_concurrente(){
    double total = 0.0;
    future<double> future[n];

    for(int i = 0; i < n; i++)
        future[i]=async(launch::async, funcion_hebra, i);

    for(int i = 0; i < n; i++)
        total += future[i].get();

    return total/m;
}
// -----------------------------------------------------------------------------

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 double pi = 3.141592653589793238461 ;

  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;
}
```

Salida de ejecución de la Forma 1

```
Número de muestras (m)    : 1073741824
Número de hebras (n)      : 4
Valor de PI               : 3.14159265358979312
Resultado secuencial      : 3.14159265358998185
Resultado concurrente     : 3.14159265358978601
Tiempo secuencial         : 1645.8 milisegundos.
Tiempo concurrente        : 504.96 milisegundos.
Porcentaje t.conc/t.sec. : 30.68%
```

El bucle recorre desde el valor que recibe (i), hasta m y se amplía de n en n. En la variable suma se almacena el valor de la suma de j entre m.

Salida de ejecución de la Forma 2

```
Número de muestras (m)   : 1073741824
Número de hebras (n)     : 4
Valor de PI              : 3.14159265358979312
Resultado secuencial     : 3.14159265358998185
Resultado concurrente    : 3.1415926409939634
Tiempo secuencial        : 1401.2 milisegundos.
Tiempo concurrente       : 499.35 milisegundos.
Porcentaje t.conc/t.sec. : 35.64%
```

Planteo el for de otra forma, yendo desde m/n multiplicado por el valor de i (argumento de la función), hasta m/n multiplicado por i+1. En la variable suma se almacena el valor de la suma de j entre m.