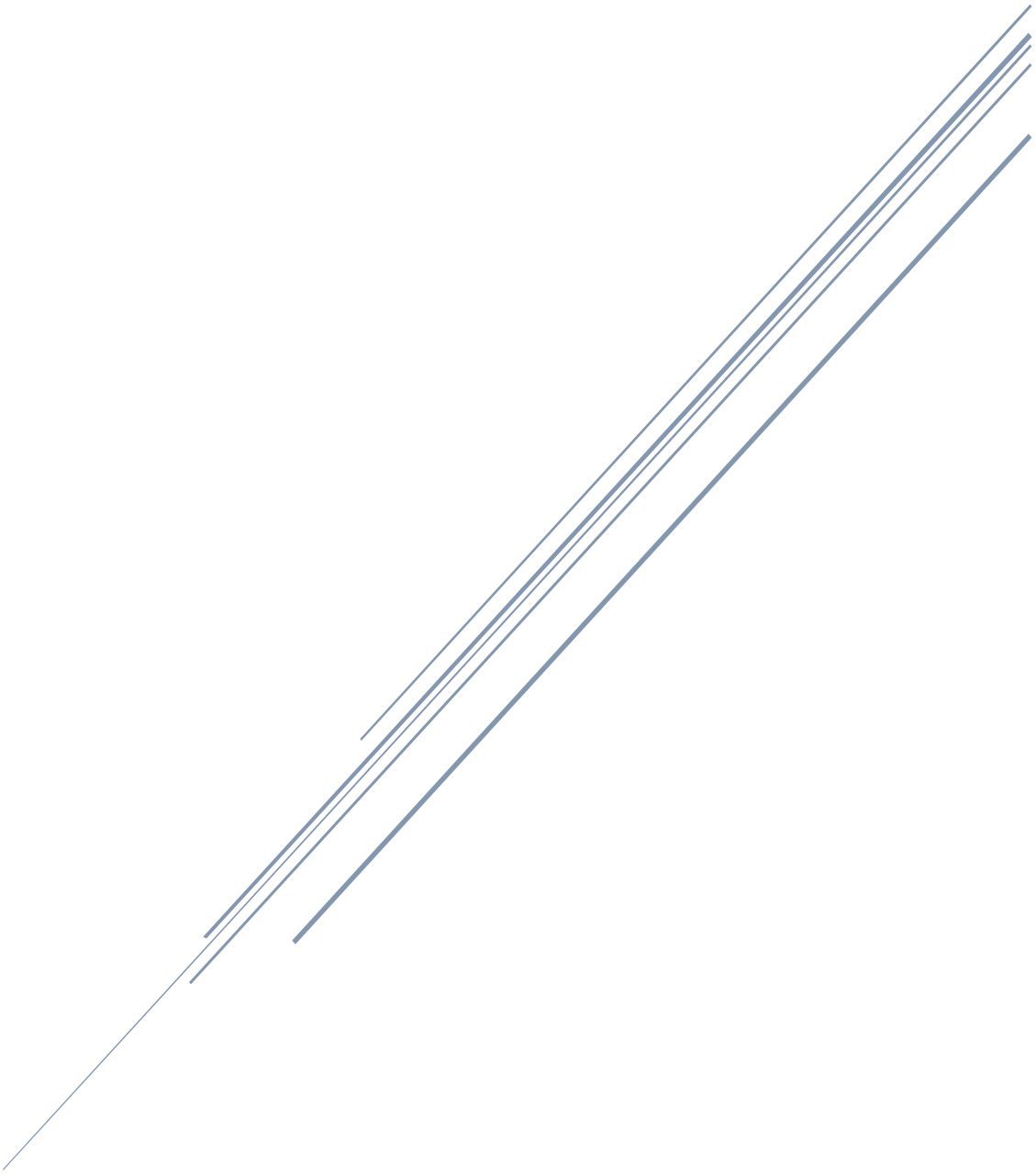
PORTAFOLIOS PRÁCTICA 4 SCD

Implementación de Sistemas de Tiempo Real

Víctor José Rubia López



UGR ETSIIT
2ºB (B2)

Actividad 1: nueva funcionalidad

A continuación, añadiré código para comprobar si el retraso que acumula la ejecución de las tareas supera 20 milisegundos. Antes de esperar hasta el inicio de la siguiente iteración del ciclo secundario añadimos lo siguiente.

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

Con esto, guardamos el instante después de haber ejecutado las tareas. Por lo tanto, ya tenemos para poder calcular el informe del retraso del instante final actual respecto al instante final esperado. Lo hacemos mediante la siguiente línea.

```
auto retraso = despues - ini_sec;

cout << "El retraso actual es de " << (nanoseconds(retraso).count() / 10000
00.0) << " milisegundos" << endl;
```

Como ya tenemos el valor del retraso, a continuación, escribiremos un bloque condicional para cumplir con lo que se nos pide, salir del programa si el retraso supera los 20 milisegundos.

```
if((despues - ini_sec) > milliseconds(20)){
   cout << "El tiempo de retraso es mayor que 20 ms" << endl;
   return -1;
}
```

Con esto hecho verificamos el correcto funcionamiento de la actividad, modificando algún valor de la tabla.

```
-------------------------------------
Comienza iteración del ciclo principal.

Comienza iteración 1 del ciclo secundario.
   Comienza tarea A (C == 200 ms.) ... fin.
   Comienza tarea B (C == 80 ms.) ... fin.
   Comienza tarea C (C == 50 ms.) ... fin.
El retraso actual es de 81.0026 milisegundos
El tiempo de retraso es mayor que 20 ms
```

Código de la solución propuesta

```
// -----------------------------------------------------------------------------
//
// Sistemas concurrentes y Distribuidos.
// Práctica 4. Implementación de sistemas de tiempo real.
//
// Archivo: ejecutivo1-compr.cpp
// Implementación del primer ejemplo de ejecutivo cíclico:
//
//   Datos de las tareas:
//   ------------
//   Ta.  T    C
//   ------------
//   A  250  100
//   B  250   80
//   C  500   50
//   D  500   40
//   E 1000   20
//  -------------
//
//  Planificación (con Ts == 250 ms)
//  *---------*----------*---------*--------*
//  | A B C   | A B D E  | A B C   | A B D  |
//  *---------*----------*---------*--------*
//
//
// Historial:
// Creado en Diciembre de 2018
// -----------------------------------------------------------------------------

#include <string>
#include <iostream> // cout, cerr
#include <thread>
#include <chrono>   // utilidades de tiempo
#include <ratio>    // std::ratio_divide

using namespace std ;
using namespace std::chrono ;
using namespace std::this_thread ;

// tipo para duraciones en segundos y milisegundos, en coma flotante:
typedef duration<float,ratio<1,1>>    seconds_f ;
typedef duration<float,ratio<1,1000>> milliseconds_f ;

// -----------------------------------------------------------------------------
// tarea genérica: duerme durante un intervalo de tiempo (de determinada duración)

void Tarea( const std::string & nombre, milliseconds tcomputo )
{
   cout << "   Comienza tarea " << nombre << " (C == " << tcomputo.count() << " ms.
) ... " ;
   sleep_for( tcomputo );
   cout << "fin." << endl ;
}

// -----------------------------------------------------------------------------
// tareas concretas del problema:

void TareaA() { Tarea( "A", milliseconds(200) );  }
void TareaB() { Tarea( "B", milliseconds( 80) );  }
void TareaC() { Tarea( "C", milliseconds( 50) );  }
void TareaD() { Tarea( "D", milliseconds( 40) );  }
```

```
void TareaE() { Tarea( "E", milliseconds( 20) );  }

// -----------------------------------------------------------------------------
// implementación del ejecutivo cíclico:

int main( int argc, char *argv[] )
{
   // Ts = duración del ciclo secundario
   const milliseconds Ts( 250 );

   // ini_sec = instante de inicio de la iteración actual del ciclo secundario
   time_point<steady_clock> ini_sec = steady_clock::now();

   while( true ) // ciclo principal
   {
      cout << endl
           << "-------------------------------------" << endl
           << "Comienza iteración del ciclo principal." << endl ;

      for( int i = 1 ; i <= 4 ; i++ ) // ciclo secundario (4 iteraciones)
      {
         cout << endl << "Comienza iteración " << i << " del ciclo secundario." <<
endl ;

         switch( i )
         {
            case 1 : TareaA(); TareaB(); TareaC();           break ;
            case 2 : TareaA(); TareaB(); TareaD(); TareaE(); break ;
            case 3 : TareaA(); TareaB(); TareaC();           break ;
            case 4 : TareaA(); TareaB(); TareaD();           break ;
         }

         // calcular el siguiente instante de inicio del ciclo secundario
         ini_sec += Ts ;

         // Obtenemos el instante de después de haber hecho las tareas para realiza
r la medición

         time_point<steady_clock> despues = steady_clock::now();

         //Informamos del retraso actual

         auto retraso = despues - ini_sec;

         // Calculamos el informe del retraso del instante final actual respecto al
 esperado

         cout << "El retraso actual es de " << (nanoseconds(retraso).count() / 1000
000.0) << " milisegundos" << endl;

         // Realizamos la medición y si es mayor a 20 milisegundos salimos
         if((despues - ini_sec) > milliseconds(20)){
            cout << "El tiempo de retraso es mayor que 20 ms" << endl;
            return -1;
         }

         // esperar hasta el inicio de la siguiente iteración del ciclo secundario
         sleep_until( ini_sec );
      }
   }
}
```

Actividad 2: nuevo ejemplo

Ajustando la planificación T_s a 500 y ajustando los datos que nos dan en la tabla se nos queda el siguiente código.

```
// ---------------------------------------------------------------------------
//
// Sistemas concurrentes y Distribuidos.
// Práctica 4. Implementación de sistemas de tiempo real.
//
// Archivo: ejecutivo2.cpp
// Implementación del primer ejemplo de ejecutivo cíclico:
//
//   Datos de las tareas:
//   ------------
//   Ta.  T    C
//   ------------
//   A  500   100
//   B  500   150
//   C  1000  200
//   D  2000  240
//  -------------
//
//  Planificación (con Ts == 250 ms)
//  *---------*----------*---------*--------*
//  |  A  B  C|  A  B  C |  A B D  |  A   B |
//  *---------*----------*---------*--------*
//
//
// Historial:
// Creado en Diciembre de 2018
// ---------------------------------------------------------------------------

#include <string>
#include <iostream> // cout, cerr
#include <thread>
#include <chrono>   // utilidades de tiempo
#include <ratio>    // std::ratio_divide

using namespace std ;
using namespace std::chrono ;
using namespace std::this_thread ;

// tipo para duraciones en segundos y milisegundos, en coma flotante:
typedef duration<float,ratio<1,1>>    seconds_f ;
typedef duration<float,ratio<1,1000>> milliseconds_f ;

// ---------------------------------------------------------------------------
// tarea genérica: duerme durante un intervalo de tiempo (de determinada duración)

void Tarea( const std::string & nombre, milliseconds tcomputo )
{
   cout << "   Comienza tarea " << nombre << " (C == " << tcomputo.count() << " ms.) ... " ;
   sleep_for( tcomputo );
   cout << "fin." << endl ;
}

// ---------------------------------------------------------------------------
// tareas concretas del problema:

void TareaA() { Tarea( "A", milliseconds(100) );  }
void TareaB() { Tarea( "B", milliseconds(150) );  }
void TareaC() { Tarea( "C", milliseconds(200) );  }
void TareaD() { Tarea( "D", milliseconds(240) );  }

// ---------------------------------------------------------------------------
// implementación del ejecutivo cíclico:

int main( int argc, char *argv[] )
```

```
{
   // Ts = duración del ciclo secundario
   const milliseconds Ts( 500 );

   // ini_sec = instante de inicio de la iteración actual del ciclo secundario
   time_point<steady_clock> ini_sec = steady_clock::now();

   while( true ) // ciclo principal
   {
      cout << endl
           << "-------------------------------------" << endl
           << "Comienza iteración del ciclo principal." << endl ;

      for( int i = 1 ; i <= 4 ; i++ ) // ciclo secundario (4 iteraciones)
      {
         cout << endl << "Comienza iteración " << i << " del ciclo secundario." << endl ;

         switch( i )
         {
            case 1 : TareaA(); TareaB(); TareaC();           break ;
            case 2 : TareaA(); TareaB(); TareaC();           break ;
            case 3 : TareaA(); TareaB(); TareaD();           break ;
            case 4 : TareaA(); TareaB();                     break ;
         }

         // calcular el siguiente instante de inicio del ciclo secundario
         ini_sec += Ts ;

         // Obtenemos el instante de después de haber hecho las tareas para realizar la medici
ón

         time_point<steady_clock> despues = steady_clock::now();

         //Informamos del retraso actual

         auto retraso = despues - ini_sec;

         // Calculamos el informe del retraso del instante final actual respecto al esperado

         cout << "El retraso actual es de " << (nanoseconds(retraso).count() / 1000000.0) << "
 milisegundos" << endl;

         // Realizamos la medición y si es mayor a 20 milisegundos salimos
         if((despues - ini_sec) > milliseconds(20)){
            cout << "El tiempo de retraso es mayor que 20 ms" << endl;
            return -1;
         }

         // esperar hasta el inicio de la siguiente iteración del ciclo secundario
         sleep_until( ini_sec );
      }
   }
}
```

1. ¿Cuál es el mínimo tiempo de espera que queda al final de las iteraciones del ciclo secundario con tu solución?

El mínimo tiempo de espera que queda al final de las iteraciones del ciclo secundario con mi solución es de 10 milisegundos.

2. ¿Sería planificable si la tarea D tuviese un tiempo de cómputo de 250 ms?

Si la tarea D tuviese un tiempo de cómputo de 250 milisegundos, sí que sería planificable, ya que, en el ciclo A B D, Suma(C) es igual a 500, que es el tiempo que se necesita para cada ciclo secundario.