Práctica 4. Implementación de Sistemas de Tiempo Real.

1. Modificando ejecutivo1.cpp

```
// ---------------------------------------------------------------------
//
// 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  |
//  *---------*----------*---------*--------*
//
// ---------------------------------------------------------------------
// Actividad 1: nueva funcionalidad
// En la simulación (en ejecutivo1.cpp) cada tarea es una simple espera bloqueada
// de duración igual a su tiempo de cómputo. También hay una espera al final del
// ciclo secundario.
// - Sabemos que, en la práctica, en una ejecución el tiempo de duración actual
//   de cada una de esas esperas puede ser algo mayor que el argumento de sleep_for.
// - Copia el código en ejecutivo1-compr.cpp y ahí extiéndelo de forma que, cada
//   vez que acaba un ciclo secundario, se informe del retraso del instante final
//   actual respecto al instante final esperado.
// - La comprobación se hará al final del bucle, inmediatamente después de sleep_until.
// ---------------------------------------------------------------------

#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( 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 (en unidades de milisegundos, enteros)
const milliseconds Ts_ms( 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_ms ;

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

// ACTIVIDAD 1----------------------------------------------------------

// fin_sec = instante final
time_point<steady_clock> fin_sec  = steady_clock::now();

// Calcular la diferencia entre el instante final actual y el instante final esperado
milliseconds_f retraso = fin_sec - ini_sec;
milliseconds_f duracion_real = Ts_ms + retraso;

// Imprimir resultados después de sleep_until
cout << "Duracion esperada:    " << Ts_ms.count() << " milisegundos." << endl;
cout << "Duracion real:        " << duracion_real.count() << " milisegundos." << endl;
cout << "Ocurre un retraso de: " << retraso.count() << " milisegundos." << endl;

}
}
}
```

Práctica 4. Implementación de Sistemas de Tiempo Real.

2. ejecutivo2.cpp

```
// ---------------------------------------------------------------------------
//
// Sistemas concurrentes y Distribuidos.
// Práctica 4. Implementación de sistemas de tiempo real.
//
// Archivo: ejecutivo2.cpp
// Implementación del segundo ejemplo de ejecutivo cíclico:
//
//   Datos de las tareas:
//   ------------
//   Ta.  T    C
//   ------------
//   A  500  100
//   B  500  150
//   C  1000 200
//   D  2000 240
//  -------------
//
//  Hiperperiodo Tm = 2000ms
//
//  Planificación (con Ts == 500 ms). Se cumple que maxC(240) <= Ts <= minD(500)
//  *---------*----------*---------*--------*
//  | A B C   | A B D    | A B C   | A B    |
//  *---------*----------*---------*--------*
//
// ---------------------------------------------------------------------------
/* Actividad 2: Responde en tu portafolios a estas cuestiones:
*
* PREGUNTA: ¿ cual es el mínimo tiempo de espera que queda al final de las
* iteraciones del ciclo secundario con tu solución ?
*
* RESPUESTA: El mínimo tiempo que queda es 10 ms y ocurre en la 2a iteracion
* del ciclo secundario.
*
* PREGUNTA: ¿ sería planificable si la tarea D tuviese un tiempo cómputo de 250 ms ?
*
* RESPUESTA: En teoría si podría ser planificable, aunque debemos de tener en cuenta que
* habrá iteraciones en las que no haya tiempo de espera al final del ciclo secundario,
```

```
* como es el caso de la iteracion 2, que entre la 2 y la 3 no habría tiempo de espera
* debido a que las tareas A B y D tardan 500ms, que es tiempo del ciclo secundario.
*
*/



#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 (en unidades de milisegundos, enteros)
const milliseconds Ts_ms( 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(); TareaD();          break ;
case 3 : TareaA(); TareaB(); TareaC();          break ;
case 4 : TareaA(); TareaB();                    break ;
}

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

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

// ACTIVIDAD 1----------------------------------------------------

// fin_sec = instante final
time_point<steady_clock> fin_sec  = steady_clock::now();

// Calcular la diferencia entre el instante final actual y el instante final esperado
milliseconds_f retraso = fin_sec - ini_sec;
milliseconds_f duracion_real = Ts_ms + retraso;

// Imprimir resultados después de sleep_until
cout << "Duracion esperada:    " << Ts_ms.count() << " milisegundos." << endl;
```

```
cout << "Duracion real:         " << duracion_real.count() << " milisegundos." << endl;
cout << "Ocurre un retraso de: " << retraso.count() << " milisegundos." << endl;

}
}
}
```