02.pdf

fer__luque

Sistemas Concurrentes y Distribuidos

2º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

```
// ---------------------------------------------------------------------------
//
// 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
//  -------------
//  El ciclo principal dura T_M = 2000 ms ya que
//  T_M = mcm(500, 500, 1000, 2000) = 2000

//  Diseñamos una planificación manualmente de forma que:
//    - Cada tarea se ejecuta una única vez completamente dentro de cada
repetición de su período
//    - Si una tarea se inicia dentro de una iteración del ciclo secundario, acaba
antes del final
//      dicha iteración

//  Planificación (con Ts == 500 ms)
//  *---------*----------*---------*--------*
//  | A B D   | A B C    | A B     | A B C  |
//  *---------*----------*---------*--------*
//     490        450       250       450

// Cuestiones:
// Mínimo tiempo de espera que queda al final de las iteraciones del ciclo
secundario:
// El tiempo más corto de espera se encontraría en el priner ciclo secundario,
pues sería únicamente de
// 10 ms frente a los: 50ms del segundo, 250ms del tercero y 50 ms del cuarto


// Sí que seguiría siendo planificable si la tarea D tuviese un tiempo de cómputo
de 250ms, pues se
// podría mantener incluso la misma solución, lo único que cambiaría es que el
primer ciclo secundario,
// en lugar de durar 490 ms duraría 500 ms, es decir el ciclo completo


// Historial:
// Creado en Diciembre de 2017
// ---------------------------------------------------------------------------

#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(); TareaD(); break ;
            case 2 : TareaA(); TareaB(); TareaC(); break ;
            case 3 : TareaA(); TareaB();           break ;
            case 4 : TareaA(); TareaB(); TareaC(); break ;
         }

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

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

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

         steady_clock::duration duracion = fin_sec - ini_sec;

         if (milliseconds_f(duracion).count() > milliseconds_f(20).count()) {
            cerr << "El retraso es demasiado grande" << endl;
            exit(-1);
         }
         else
            cout << "El retraso es de " << milliseconds_f(duracion).count() << "
```

```
milisegundos." << endl;
      }

   }
}
```