01.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: ejecutivo1.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 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( 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 ;

         // 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;
      }

   }
}
```