dit
UPM

Tiempo real

Juan Antonio de la Puente
DIT/UPM

Transparencias basadas en el capítulo 12 del libro de A. Burns y A. Wellings *Real-Time Systems and Programming Languuages*, 3ª edición (2001)

Tiempo real

- **Objetivo**
  - Comprender el papel del tiempo en el diseño y realización de sistemas de tiempo real

- **Contenido**:
  - Sistemas de referencia de tiempo
  - Relojes, retardos y límites temporales *(time-outs)*
  - Requisitos temporales
  - Ámbitos temporales
  - Tolerancia de fallos

Necesidades

- Acceso al **tiempo real**
  - leer el paso del tiempo en relojes
  - retrasar la ejecución de los procesos durante un tiempo
  - definir límites temporales para la ocurrencia de un suceso (time-outs)
- Representación de los **requisitos temporales**
  - períodos de activación
  - plazos de ejecución
- Análisis del cumplimiento de los requisitos temporales
  - lo veremos en el próximo tema

Tiempo universal

- TU ó UT *(Universal Time)*
  - tiempo solar medio en el meridiano 0 (Greenwich)
    - » definido en 1884 (entonces se llamaba GMT)

> *1 s = 1/86 400 de un día solar medio*

    - » definición oficial hasta 1955

- Muy impreciso y difícil de determinar
  - la duración del día no es constante
    - » pérdida de energía por las mareas
    - » otros fenónemos irregulares

Tiempo de efemérides

- Año trópico
  - tiempo transcurrido entre dos pasos de la Tierra por el punto γ

> *1s = 1/31 566 925,9747 del año trópico 1900*

    - » definición oficial entre 1955 y 1967

- Correcciones del tiempo universal (UT0)
  - UT1: UT0 corregido por el movimiento de los polos
  - UT2: UT1 corregido por variaciones en la rotación de la Tierra

Tiempo atómico

- Los relojes atómicos proporcionan medidas estables y precisas

> *1 s = 9 192 631 770 períodos de la radiación correspondiente a la transición entre los dos niveles hiperfinos del estado fundamental del átomode Cesio 133 en reposo a una temperatura de 0 K*

  - » definición oficial (SI) desde 1967
  - » precisión del orden de 10-13 (1 s en 300 000 años)

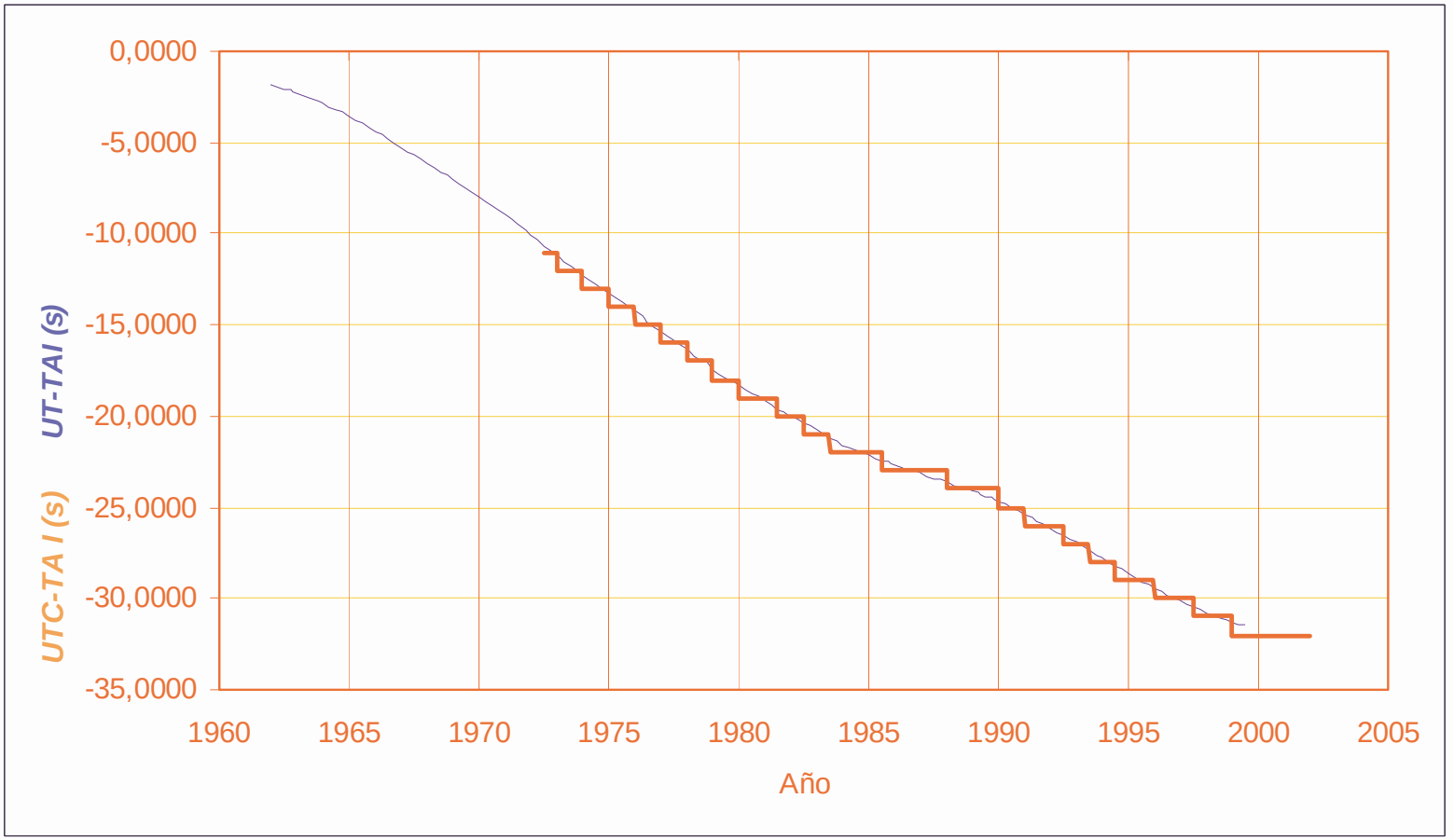
- **TAI (tiempo atómico internacional)**
  - definido en 1970
  - sincronizado con UT en 1958-01-01:00:00:00
    - » mantenido por una red coordinada por el BIMP *(Bureau International de Mésures et Poids)*

Tiempo universal coordinado

- El TAI se aparta lentamente del UT
  - » La duración del día solar medio va en aumento
  - » En 2001, la diferencia es aproximadamente de 32 s

- **UTC *(Universal Time Coordinated)***
  - – Definido en 1972
  - – UTC = TAI + H
    - » H se elige de forma que |UT2 - UTC| ≤ 0,9 s
  - – Se añade un *segundo intercalar* al UTC cuando es necesario.
    - » 30 de junio o 31 de diciembre a las 24:00
    - » Lo decide el IERS *(International Earth Rotation Service)*
  - – La hora oficial (TO u OT) en cada país se basa en el UTC
    - » TO = UTC + Z + C

Diferencias entre UT, UTC y TAI



El calendario

- Calendarios antiguos: solares y lunares
  - año, mes, semana
- Calendario juliano
  - 1 año = 365,25 días
    - » un día adicional (año bisiesto) cada 4 años
- **Calendario gregoriano** (1582)
  - 1 año = 365,2425 días
    - » se suprimen 3 días cada 400 años
    - » sólo los múltiplos de 100 que también lo son de 400 son bisiestos
  - se mantiene un error de 1 día cada 2500 años, aproximadamente

Referencias de tiempo

- El UTC es adecuado para la comunicación con personas
- Pero presenta saltos (segundos intercalares) que lo hacen inadecuado para medidas de tiempo precisas
  - peor todavía si consideramos el calendario y la hora oficial
- EL TAI es una referencia más adecuada para controlar la ejecución de tareas de tiempo real
- Lo que hace falta es una referencia de tiempo
  - estable: sin variaciones grandes a lo largo del tiempo
  - exacta: diferencia con el TAI acotada
    - » excepto una constante: puede tener una *época* (origen) cualquiera
  - precisa: la diferencia entre dos lecturas sucesivas está acotada
  - monótona no decreciente: sin saltos hacia atrás

Relojes

- Los relojes son módulos de hardware y software que permiten medir el tiempo real.
- Pueden ser internos o externos
- **Características importantes**:
  - Características estáticas (representación del tiempo)
    - » Resolución
    - » Intervalo de valores
  - Características dinámicas
    - » Granularidad
    - » Exactitud
    - » Estabilidad

      variación o movimiento

$$1-\rho \leq \frac{\tau(t')-\tau(t)}{t'-t} \leq 1+\rho$$

Relojes en Ada

En Ada hay dos paquetes predefinidos que proporcionan funciones de reloj:

- Ada.Calendar
  - Define un tipo abstracto Time
  - La función Clock da un valor de tiempo para uso externo
  - Los intervalos de tiempo se representan con el tipo predefinido Duration
- Ada.Real_Time
  - Define un tipo abstracto Time
  - La función Clock da un valor monótono, sin saltos
  - Los intervalos de tiempo se representan con el tipo abstracto Time_Span

Intervalos de tiempo

- El tipo Duration representa intervalos de tiempo en segundos

- Es un tipo de coma fija:

  type Duration **is delta** ... **range** ...;

  – Su resolución, Duration'Small, no debe ser mayor que 20ms (se recomienda que no sobrepase los 100µs)

  – El intervalo de valores debe comprender ±1 día (-86_400.0 .. +86_400.0)

Ada.Calendar (1)

```
package Ada.Calendar is

   type Time is private;

   subtype Year_Number      is Integer  range 1901..2099;
   subtype Month_Number     is Integer  range 1..12;
   subtype Day_Number       is Integer  range 1..31;
   subtype Day_Duration     is Duration range 0.0..86_400.0;

   function Clock return Time;

   function Year   (Date : Time) return Year_Number;
   function Month  (Date : Time) return Month_Number;
   function Day    (Date : Time) return Day_Number;
   function Seconds(Date : Time) return Day_Duration;

   procedure Split(Date    : in  Time;
                   Year    : out Year_Number;
                   Month   : out Month_Number;
                   Day     : out Day_Number;
                   Seconds : out Day_Duration);
```

Ada.Calendar (2)

```
   function Time_Of
     (Year    : Year_Number;
      Month   : Month_Number;
      Day     : Day_Number;
      Seconds : Day_Duration := 0.0)
     return    Time;

   function "+" (Left : Time;     Right : Duration) return Time;
   function "+" (Left : Duration; Right : Time)     return Time;
   function "-" (Left : Time;     Right : Duration) return Time;
   function "-" (Left : Time;     Right : Time)     return
   Duration;

   function "<"  (Left, Right : Time) return Boolean;
   function "<=" (Left, Right : Time) return Boolean;
   function ">"  (Left, Right : Time) return Boolean;
   function ">=" (Left, Right : Time) return Boolean;

   Time_Error : exception;
end Ada.Calendar;
```

Comentarios a Ada.Calendar

- Los valores del tipo Time combinan la fecha y la hora
- La hora se da en segundos desde medianoche
  - cuando hay un segundo intercalar se llega a 86_400.0
- El reloj se supone sincronizado con una referencia externa (UTC o TO)
  - los detalles se dejan para el entorno (SO)

Ejemplo

```
declare
   use Ada.Calendar;
   Start, Finish      : Time;
   Interval           : Duration;
begin
   Start  := Clock;
   -- instrucciones cuya duración se mide
   Finish := Clock;
   Interval := Finsih - Start;
end;
```

Ada.Real_Time (1)

```
package Ada.Real_Time is

   type Time is private;
   Time_First : constant Time;
   Time_Last  : constant Time;
   Time_Unit  : constant := -- real number;

   type Time_Span is private;
   Time_Span_First :  constant Time_Span;
   Time_Span_Last  :  constant Time_Span;
   Time_Span_Zero  :  constant Time_Span;
   Time_Span_Unit  :  constant Time_Span;

   Tick : constant Time_Span;
   function Clock return Time;

   function "+"  (Left : Time;      Right : Time_Span) return Time;
   function "+"  (Left : Time_Span; Right : Time)      return Time;
   function "-"  (Left : Time;      Right : Time_Span) return Time;
   function "-"  (Left : Time;      Right : Time) return Time_Span;

   function "<"  (Left, Right : Time) return Boolean;
   function "<=" (Left, Right : Time) return Boolean;
   function ">"  (Left, Right : Time) return Boolean;
   function ">=" (Left, Right : Time) return Boolean;
```

Ada.Real_Time (2)

```
function "+" (Left, Right : Time_Span)              return Time_Span;
function "-" (Left, Right : Time_Span)              return Time_Span;
function "-" (Right : Time_Span)                    return Time_Span;
function "*" (Left : Time_Span; Right : Integer)    return Time_Span;
function "*" (Left : Integer;   Right : Time_Span)  return Time_Span;
function "/" (Left, Right : Time_Span)              return Integer;
function "/" (Left : Time_Span; Right : Integer)    return Time_Span;
function "abs" (Right : Time_Span)                  return Time_Span;

function "<"  (Left, Right : Time_Span) return Boolean;
function "<=" (Left, Right : Time_Span) return Boolean;
function ">"  (Left, Right : Time_Span) return Boolean;
function ">=" (Left, Right : Time_Span) return Boolean;

function To_Duration  (TS : Time_Span)  return Duration;
function To_Time_Span (D : Duration)    return Time_Span;

function Nanoseconds  (NS : integer)    return Time_Span;
function Microseconds (US : integer)    return Time_Span;
function Milliseconds (MS : integer)    return Time_Span;
```

Ada.Real_Time (3)

```
   type Seconds_Count is new Integer range ...;

   procedure Split (T : Time;
                    SC : out Seconds_Count;
                    TS : out Time_Span);
   function Time_Of (SC : Seconds_Count; TS : Time_Span)
     return Time;

end Ada.Real_Time;
```

Comentarios sobre Ada.Real_Time (1)

- El tipo Time representa valores de tiempo absolutos.
  - Un valor *T* de tipo Time representa un intervalo de duración
    $$[E + T \cdot \text{Time\_Unit}, E + (T+1) \cdot \text{Time\_Unit}]$$
    - » Time_Unit no debe ser mayor de 20ms..
    - » El valor de *E* no está definido
  - El intervalo de valores del tipo Time debe alcanzar al menos 50 años desde el arranque del sistema.
- El tipo Time_Span representa intervalos de tiempo.
  - Un valor *S* de tipo Time_Span representa un intervalo de duración igual a $S \cdot$ Time_Span_Unit .
    - » Time_Span_Unit =Time_Unit
    - » Duration'Small debe ser un múltiplo entero de Time_Span_Unit.
  - El intervalo de valores del tipo Time_Span debe abarcar por lo menos -3600..+3600 s .

Comentarios sobre Ada.Real_Time (2)

- La función Clock proporciona el tiempo absoluto transcurrido desde la época.

- Tick es el valor medio del intervalo durante el cual el valor de Clock permanece constante. No debe ser mayor de 1ms
  - Se recomienda que el valor de Tick sea igual al de Time_Span_Unit, o un múltiplo exacto de éste.

- El valor de Clock no debe disminuir en ningún momento (es decir, el reloj es monótono no decreciente).

Ejemplo

```
declare
     use Ada.Real_Time;
     Start, Finish : Time;
     Frame         : Time_Span := Milliseconds(10);
begin
   Start := Clock;
   -- instrucciones
   Finish := Clock;
   if Finish - Start > Frame then
      raise Time_Error; -- excepción definida por el programador
   end if;
end;
```

Medida del tiempo de ejecución

- Se usa una técnica de *doble bucle*
- Sólo es válido si la secuencia de instrucciones se ejecuta de una vez (sin ceder el procesador a otras tareas o al núcleo)

```
declare
   T0, T1, T2     : Ada.Real_Time.Time;
   Execution_Time : Ada.Real_Time.Time_Span;
   N : constant Positive := ...;
begin
   T0 = Ada.Real_Time.Clock;
   for i in 1 .. N loop
      -- secuencia de instrucciones
   end loop;
   T1 = Ada.Real_Time.Clock;
   for i in 1 .. N loop
      null;
   end loop;
   T2 = Ada.Real_Time.Clock;
   Execution_Time = ((T1 - T0) - (T2 - T1))/N;
end;
```

Relojes en Java

- Reloj de hora del día (Java)
  - java.lang.System.currentTimeMillis da el número de milisegundos transcurridos desde UT 1970-01-01:00:00
  - java.util.Date usa este valor para dar la fecha y hora del día
    - » muchos detalles complicados (por ejemplo, el mes va de 0 a 11)
- Tipos de datos de alta resolución (RT Java)
  - representación del tiempo con resolución de 1 ns
- Relojes y temporizadores (RT Java)
  - adecuados para sistemas de tiempo real

Tiempo de alta resolución

```
public abstract class HighResolutionTime
   implements java.lang.Comparable {  ...}

public class AbsoluteTime extends HighResolutionTime {...}

public class RelativeTime extends HighResolutionTime {...}

public class RationalTime extends RelativeTime {...}
```

- Un valor de tiempo se compone de un número entero de milisegundos y un número entero de nanosegundos
- El tiempo absoluto se cuenta desde 1970-01-01:00:00:00
- Un valor de tiempo racional representa un intervalo compuesto de subintervalos con una frecuencia determinada
- Las definiciones de estas clases contienen contructores, operaciones aritméticas y otras

Relojes

```
public abstract class Clock
{
  public Clock();
  public static Clock getRealtimeClock();
  public abstract RelativeTime getResolution();
  public AbsoluteTime getTime();
  public abstract void getTime(AbsoluteTime time);
  public abstract void setResolution(RelativeTime resolution);
}
```

- Siempre hay disponible al menos un reloj de tiempo real
- Puede haber clases derivadas que definen relojes especializados

Ejemplo

```
AbsoluteTime start, finish;
RelativeTime interval;
Clock clock = Clock.getRealtimeClock();

start = clock.getTime();
// secuencia de instrucciones
finish = clock.getTime();
interval = finish.subtract(start);
```

Temporizadores

```
public abstract class Timer extends AsyncEvent
{
   protected Timer(HighResolutionTime t, Clock c,
                   AsyncEventHandler handler);
   ...
   public void reschedule(HighResolutionTime time);
   public void start();
}

public class OneShotTimer extends Timer {...}
public class PeriodicTimer extends Timer {...}
```

Relojes en C / POSIX

Hay también dos tipos

- **Reloj calendario (ANSI C)**
  - Proporciona valores de tiempo con resolución de 1s
- **Relojes de tiempo real (POSIX)**
  - Se pueden definir distintos relojes
  - Por lo menos debe haber uno denominado ***CLOCK_REALTIME***
  - La resolución de la representación es de 1ns
  - La granularidad depende de la implementación
    - » como máximo 20 ms

Reloj-calendario

- Proporciona una medida de tiempo con resolución de 1s

```
#include <time.h>
time_t time (time_t *tloc);
```

- El tipo *time_t* representa un número entero de segundos
- El tiempo se mide desde las 0h UT del 1 de enero de 1970
- Hay otras operaciones y otros tipos que facilitan el uso de unidades corrientes (año, mes, día, hora, ...)

Relojes de tiempo real en POSIX

- El tiempo se representa mediante el tipo *timespec*

```
struct timespec {
   time_t tv_sec;    /* segundos */
   long   tv_nsec }  /* nanosegundos */
```

- El tipo *clockid_t* sirve para identificar distintos relojes
- *CLOCK_REALTIME* es una constante de este tipo
- La resolución máxima del reloj es de 20ms
- Puede haber otros relojes

Operaciones con relojes

- Leer la hora:

```
#include <time.h>
int clock_gettime (clockid_t clockid,
                   struct timespec *tp);
```

- Poner en hora

```
#include <time.h>
int clock_settime (clockid_t clockid,
                   const struct timespec *tp);
```

- Resolución del reloj

```
#include <time.h>
int clock_getres (clockid_t clockid,
                  struct timespec *res);
```

Índice

- **Medida del tiempo y relojes**
- **Retardos**
- Límites temporales *(time-outs)*
- Requisitos temporales
- Tolerancia de fallos

Retardos

- Un **retardo** suspende la ejecución de una tarea durante un cierto tiempo
- Hay dos tipos
  - **Retardo relativo**: la ejecución se suspende durante un intervalo de tiempo relativo al instante actual
  - **Retardo absoluto**: la ejecución se suspende hasta que se llegue a un instante determinado de tiempo absoluto

Retardo relativo en Ada

- En Ada hay una instrucción

  delay *expresión;*

  que suspende la ejecución de la tarea que la invoca durante un intervalo de tiempo
- La *expresión* es de tipo Duration
- Una instrucción delay con argumento cero o negativo no produce ningún retardo

Retardo relativo en C/POSIX

- La función

```
unsigned int sleep (unsigned int seconds);
```

  suspende el *thread* que la invoca durante un número entero de segundos

- La función

```
#include <time.h>
int nanosleep (const struct timespec *rqtp,
               struct timespec *rmtp;)
```

  permite especificar retardos con mayor precisión
  La duración del retardo es **rqtp*

Retardo absoluto en Ada

- La instrucción

  delay until *expresión*;

  suspende la ejecución de la tarea que la invoca hasta que el valor del reloj sea igual al especificado por la *expresión*
- La expresión es de uno de estos tipos:
  - Ada.Calendar.Time
  - Ada.Real_Time.Time
- Se usa el reloj correspondiente al tipo Time utilizado
- Si se especifica un tiempo anterior al valor actual del reloj, no se produce ningún retardo

Aproximación con retardo relativo

- La instrucción

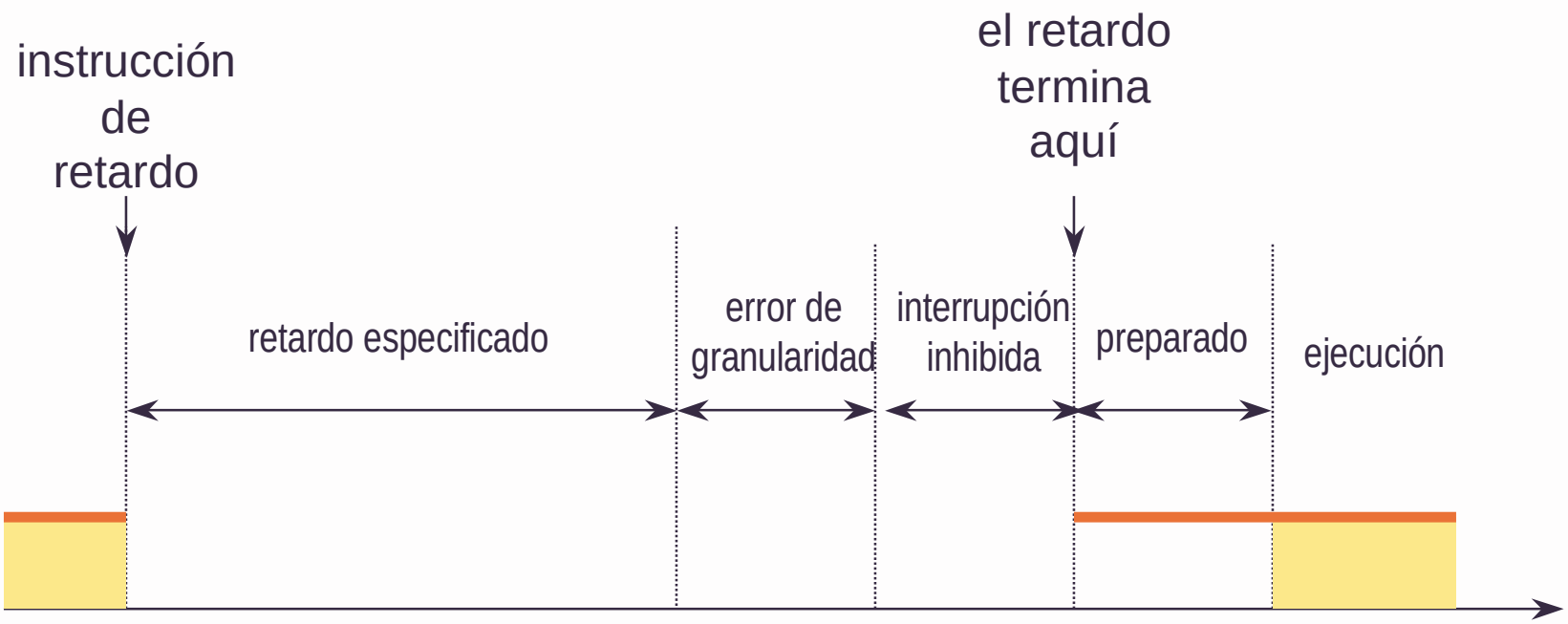
  delay until *T*;

  se puede aproximar mediante

  delay (*T* - Clock);

  pero el efecto no es exactamente el mismo
- La expresión (*T* - Clock) tendría que ejecutarse de forma atómica

Ejecución de un retardo



Índice

- **Medida del tiempo y relojes**
- Retardos
- **Límites temporales *(time-outs)***
- Requisitos temporales
- Tolerancia de fallos

Limitación del tiempo de espera

- A menudo conviene limitar el tiempo durante el cual se espera que ocurra un suceso
- **Ejemplos**:
  - Acceso a una sección crítica:
    La espera está limitada por la duración de la sección crítica
  - Sincronización condicional
    - » Ada: Llamada a una entrada protegida con barreras
    - » POSIX: Espera en variable de condición o un semáforo
  - Cita entre dos tareas
  - Ejecución de una acción

Ejemplo

```
task Controller is
   entry Call (T : Temperature);
end Controller;


task body Controller is
   -- declaraciones
begin
   loop
      accept Call (T : Temperature) do
         New_Temp := T;
      end Call;
      -- otras acciones
   end loop;
end Controller;
```

Aceptación temporizada

- Se puede especificar una acción alternativa en caso de que la llamada no se reciba dentro de un cierto intervalo mediante una **aceptación temporizada:**

```
select
   accept Call (T : Temperature) do
      New_Temp := T;
   end Call;
or
   delay 10.0;
   -- acción alternativa
end select;
```

- El retardo puede ser también absoluto

Llamada temporizada

- Se puede limitar el tiempo que tarda en aceptarse la llamada mediante una **llamada temporizada**

```
loop
   -- leer el nuevo valor de T
   select
      Controller.Call(T);
   or
      delay 0.5;
      -- acción alternativa
   end select;
end loop;
```

- Aquí también puede usarse un retardo absoluto

Llamada condicional

- Se usa cuando se quiere ejecutar una acción alternativa si la llamada no se acepta inmediatamente

```
select
   Controller.Call(T);
else
   -- acción alternativa
end select;
```

Llamada temporizada y condicional a entradas protegidas

- La llamada temporizada se puede usar para limitar el tiempo de espera en una entrada protegida

```
select
   P.E(...); -- P es un objeto protegido
or
   delay 0.5;
end select;
```

- También se pueden hacer llamadas condicionales a entradas protegidas

Acciones temporizadas

- Se puede usar una **transferencia asíncrona de control** (ATC) para limitar el tiempo de ejecución de una acción:

```
select
   delay 0.1;
then abort
   -- acción
end select;
```

- Es útil para detectar y recuperar fallos

Aplicación al cómputo impreciso

- Se trata de ejecutar rápidamente un parte obligatoria de un cálculo, y de iterar sobre una parte opcional que mejora el resultado mientras haya tiempo

```
begin
   -- parte obligatoria
   select
      delay until Completion_Time;
   then abort
      loop
         -- mejorar el resultado
      end loop;
   end select;
end;
```

Espera temporizada en C/POSIX (1)

- La función

```
#include <pthread.h>
int pthread_cond_timedwait
                  (pthread_cond_t *cond,
                 pthread_mutex_t *mutex,
          const struct timespec *abstime);
```

  permite limitar el tiempo durante el cual se espera una condición

- El límite es absoluto y su valor es **abstime*

Espera temporizada en C/POSIX (2)

- Para limitar el tiempo de espera de una señal se usa

```
#include <signal.h>
int sigtimedwait (const sigset_t *set,
                  siginfo_t *info,
        const struct timespec *timeout);
```

- Aquí *timeout* es un intervalo de tiempo relativo

Temporizadores en POSIX

- Se pueden crear temporizadores asociados a relojes
- Cada temporizador se identifica mediante un valor del tipo *timer_t*
- El tiempo de espera se especifica mediante un valor de tipo *itimerspec:*

```
struct itimerspec {
  struct timespec it_interval; /* período */
  struct timespec it_value} /* expiración */
```

Creación y destrucción de temporizadores

- La función

```
#include <signal.h>
#include <time.h>
int timer_create (clockid_t clock_id,
                  struct sigevent *evp,
                      timer_t *timerid);
```

  crea un temporizador asociado al reloj *clock_id*
  - **evp* indica el tipo de notificación que se produce al expirar el temporizador
  - el identificador del temporizador se devuelve en **timer_id*
- Se puede destruir un temporizador con

```
int timer_delete (timer_t timerid);
```

Armar un temporizador

- Se usa la función

```
int timer_settime (timer_t timerid,
                   int flags,
    const struct itimerspec *value,
         struct itimerspec *ovalue);
```

- La temporización puede ser relativa o absoluta, según el valor de *flag*
- El funcionamiento se repite periódicamente si *value.it_period* > 0
- En **ovalue* se devuelve el valor que quedaba de la temporización anterior

Índice

- **Medida del tiempo y relojes**
- Retardos
- Límites temporales *(time-outs)*
- **Requisitos temporales**
- Tolerancia de fallos

Requisitos temporales

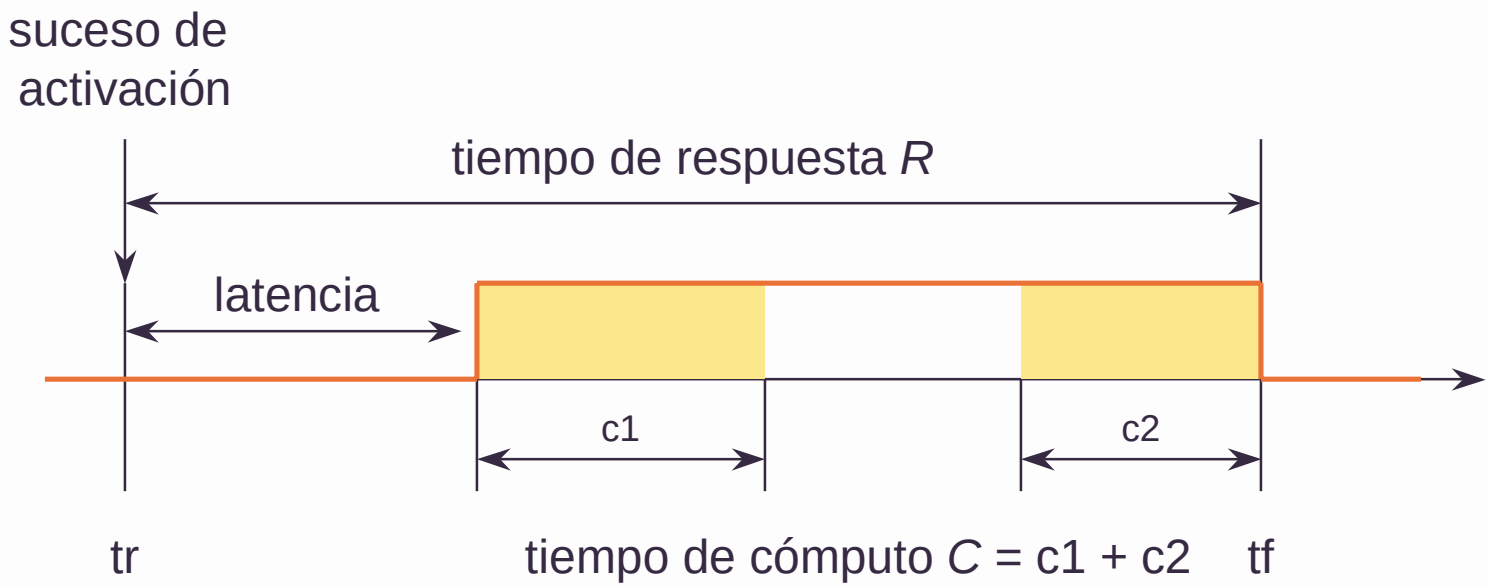
Hay dos formas de enfocar este tema:

- **Métodos formales:**
  - Especificar las propiedades temporales con un modelo formal
  - Validar la especificación
  - Comprobar que la implementación satisface las propiedades temporales
- **Métodos analíticos:**
  - Analizar las propiedades temporales desde el punto de vista de la planificación de las tareas

Seguiremos en detalle este último enfoque

Atributos temporales

- Los **atributos temporales** de una secuencia de instrucciones definen un **marco temporal** para su ejecución



Parámetros temporales

- **Parámetros**

D	Plazo de respuesta $(R \leq D)$
L	Tiempo límite $(t_f \leq L)$
$\mathbf{J_{min}}$	Latencia mínima
$\mathbf{J_{max}}$	Latencia máxima
C	Tiempo de cómputo máximo

- **Activación**
  - **Periódica**

T	Período
O	Fase

  - **Aperiódica** — Irregular, a ráfagas, estocástica
  - **Esporádica** — **T** Separación mínima

Requisitos temporales de tareas

- Los marcos temporales suelen ir asociados a tareas o procesos
- Generalmente se trata de
  - Ejecutar **tareas periódicas**
  - Ejecutar **tareas esporádicas** cuando ocurren los sucesos correspondientes
  - Completar la ejecución de todas las tareas dentro de su **plazo de respuesta**
- A veces se exige que la entrada o salida de una tarea se efectúe a intervalos regulares
  La desviación se llama variación o ***jitter***

Criticidad

Una tarea de tiempo real puede ser

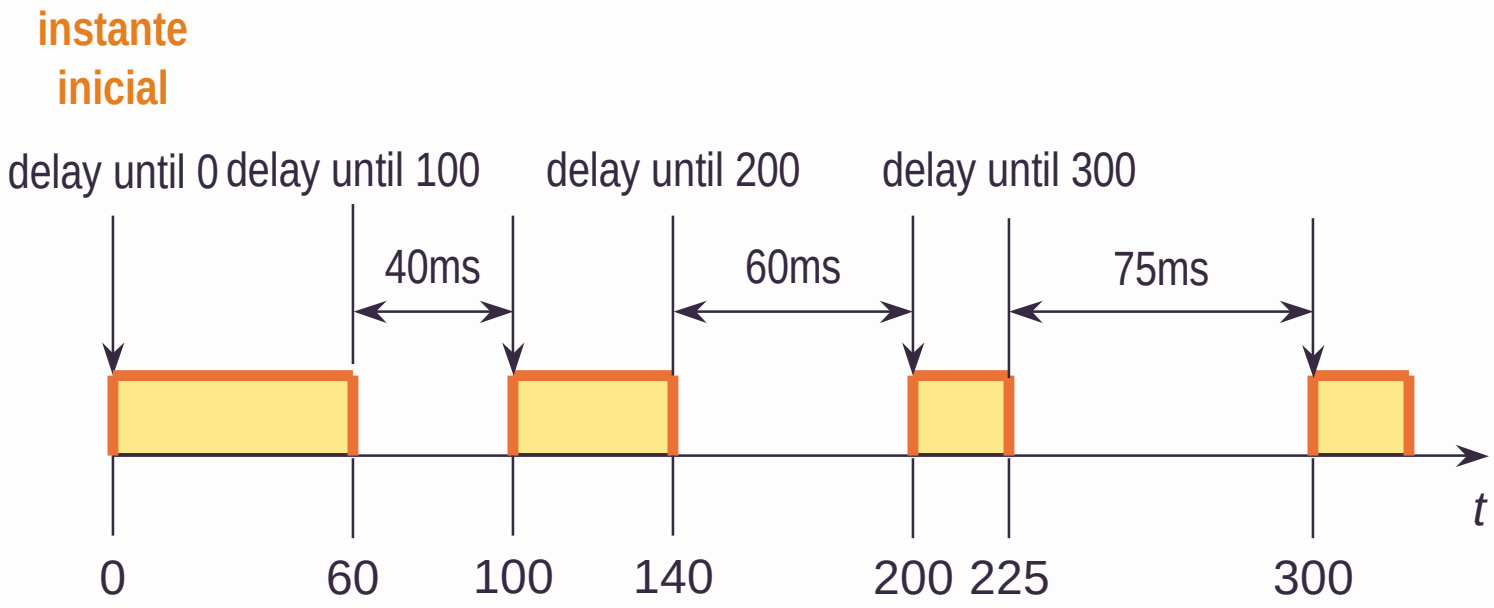
- **Crítica** *(hard)* : No se puede admitir que se sobrepase el plazo de respuesta especificado ni una sola vez
- **Acrítica** *(soft)* : Es admisible que se sobrepase el plazo ocasionalmente
- **Firme** *(firm)* : El plazo no es crítico, pero una respuesta tardía no sirve para nada
- **Interactiva** : No se especifican plazos de respuesta, sino tiempos de respuesta medios

Tareas periódicas en Ada

Se construyen con un **retardo absoluto:**

```
use Ada.Real_Time;
task body Periodic is
  Period    : constant Time_Span := ...;
  Next_Time :          Time      := ...;
begin
  -- iniciación
  loop
    delay until Next_Time;
    -- acción periódica
    Next_Time := Next_Time + Period;
  end loop;
end Periodic;
```

Ejecución de una tarea periódica



Tarea periódica con límite de tiempo

```
use Ada.Real_Time;
task body Periodic is
   Period           : constant Time_Span := ...;
   Budget           : constant Time_span := ...;
   Next_Time, Limit :          Time      := ...;
   Overrun          : exception;
begin
   -- iniciación
   loop
      delay until Next_Time;
      Limit := Clock + Budget;
      select
         delay until Limit;
         raise Overrun;
      then abort
         -- acción periódica
      end select;
      Next_Time := Next_Time + Period;
   end loop;
exception
   when Overrun => ...;
end Periodic;
```

Tareas esporádicas en Ada (1)

El **suceso** de activación se implementa mediante un **objeto protegido**

```
protected type Event is
   procedure Signal;
   entry     Wait;
private
   Occurred : Boolean := False;
end Event;
```

Tareas esporádicas en Ada (2)

```
protected body Event is

   procedure Signal is
   begin
      Occurred := True;
   end Signal;

   entry Wait when Occurred is
   begin
      Occurred := False;
   end Wait;

end Event;
```

Tareas esporádicas en Ada (3)

La tarea esporádica se activa cuando ocurre el suceso:

```
Activation : Event;
task body Sporadic is
begin
   -- iniciación
   loop
      Activation.Wait;
      -- acción esporádica
   end loop;
end Sporadic;
```

Separación mínima (1)

```
use Ada.Real_Time.Time;
...
protected type Event is
   procedure Signal;
   entry     Wait (Event_Time : out Time);
private
   Occurred        : Boolean := False;
   Occurrence_Time : Time;
end Event;
```

Separación mínima (2)

```
protected body Event is
  procedure Signal is
  begin
     Occurred         := True;
     Occurrence_Time := Clock;
  end Signal;

  entry Wait (Event_Time : out Time)
    when Occurred is
  begin
    Occurred := False;
    Event_Time := Occurrence_Time;
  end Wait;
end Event;
```

Separación mínima (3)

```
Activation : Event;
task body Sporadic is
   Separation : Time_Span;
   Next_Time, Activation_Time : Time;
begin
   -- iniciación
   loop
      Activation.Wait (Activation_Time);
      Next_Time := Activation_Time + Separation;
      -- acción esporádica
      delay until Next_Time;
   end loop;
end Sporadic;
```

Tareas periódicas en C/POSIX (1)

- Se pueden realizar con **temporizadores**

```
#include <signal.h>
#include <time.h>
#include <pthread.h>

void * periodic {
  struct timespec period;
  struct itimerspec timer_parameters;
  timer_t timer_id;
  struct sigevent event;
  sigset signals;
  int t_signal, r_signal;
```

Tareas periódicas en C/POSIX (2)

Iniciación de datos y creación del temporizador

```
period.tv_sec  = ...;
period.tv_nsec = ...;
timer_parameters.it_interval = period;
timer_parameters.it_value = period;
t_signal = ...;
event.sigev_notify = SIGEV_SIGNAL;
event.sigevent_signo = t_signal;
sigemptyset (&signals);
sigaddset (&signals, t_signal);
pthread_sigmask (SIG_BLOCK, &signals, NULL);

timer_create (CLOCK_REALTIME, event, &timer_id);
timer_settime(timer_id,0,&timer_parameters,NULL);
```

Tareas periódicas en C/POSIX (3)

Parte cíclica del *thread* periódico

```
    while (1) {
      sigwait (&signals, &r_signal);
      /* acción periódica */
     }

  }  /* end periodic */
```

Tareas periódicas en C/POSIX (4)

```
main () {
  pthread_t periodic_id;
  pthread_attr_t periodic_attr;
  sigset_t signals;
  int t_signal;
  t_signal = ...; /* ¡la misma de antes! */
  sigemptyset (&signals);
  sigaddset (&signals, t_signal;
  pthread_sigmask (SIG_BLOCK, &signals, NULL);
  pthread_attr_init (&periodic_attr);
  pthread_create (&periodic_id, &periodic_attr,
                  periodic, NULL);
    pthread_join (&periodic_id);
}
```

Índice

- Medida del tiempo y relojes
- Retardos
- Límites temporales *(time-outs)*
- Requisitos temporales
- **Tolerancia de fallos**

Fallos temporales

Una tarea puede incumplir su plazo por varias razones, por ejemplo:

- El **tiempo de cómputo** no está bien calculado
- El **análisis** de tiempos de respuesta no es realista
- Las **herramientas** de análisis contienen errores
- No se cumplen las **hipótesis** de diseño
  (por ejemplo, separación mínima entre eventos)

En estos casos hay que **detectar** los fallos

Si el sistema es crítico, debe **recuperarse**

Detección de fallos temporales

- Fallos que se refieren al **tiempo transcurrido** (por ejemplo, incumplimiento de plazo) :
  - Transferencia de control temporizada (Ada)
  - Temporizador con manejador de señal (POSIX)
- Fallos que se refieren al **tiempo de cómputo**
  - Hace falta un reloj de tiempo de cómputo
  - No es estándar en Ada ni en POSIX
  - Hay soluciones particulares en algunos sistemas operativos

Recuperación de fallos

- Se aplican esquemas de recuperación directa o inversa
- Esquema: **Grupos de recuperación**
  - Generalmente varias tareas cooperan para conseguir una funcionalidad común
  - Grupo de recuperación: Conjunto de **tareas primarias** con una **tarea de recuperación** común
  - Cuando hay un fallo se terminan todas las tareas primarias y se arranca la tarea de recuperación
  - Esta proporciona una **funcionalidad degradada**
  - Si falla otra vez, se activa una **tarea de recuperación global** del sistema

Resumen

- Hay cuatro aspectos importantes relacionados con el tratamiento del tiempo
  - medida del tiempo mediante relojes
  - retardos
  - limitación del tiempo de espera
  - especificación de requisitos temporales
- Los atributos más importantes de un marco temporal (generalmente asociado a una tarea) son:
  - esquema de activación (periódico, esporádico)
  - plazo de terminación
  - latencia de activación
  - tiempo de cómputo máximo