Servicios de temporización

- Necesidades de temporización
  - Temporizador de disparo único
  - Temporizador repetitivo
- Temporización en ADA
- Temporización en POSIX 1003.1a
- Temporización en POSIX 1003.1b
  - Reloj de referencia
  - Creación, programación y destrucción de temporizadores
  - Otras llamadas de utilidad

Servicios de temporización

- Tarea típica:

```
Hacer siempre
    Esperar momento apropiado
    Realizar operaciones
Fin hacer
```

- Según causa del desbloqueo:
  - Tareas activadas por eventos (“event driven”)
  - Tareas activadas por tiempo
- Soluciones para tareas activadas por eventos:
  - Muestreo (“polling”)
  - Respuesta a interrupciones

Servicios de temporización

- Servicios de temporización requeridos:
  - Acciones periódicas
  - Aviso aislado
    - Retraso
    - Sobretiempo
- Diferentes referencias
  - Relativa: “Esperar 1 ms” (desde ahora…)
  - Absoluta: “Esperar hasta las 13:30”
- Soluciones (tarea activada por tiempo):
  - Temporización por programa (“ballast coding”)
  - Reloj de tiempo real y temporizadores

Temporización por programa

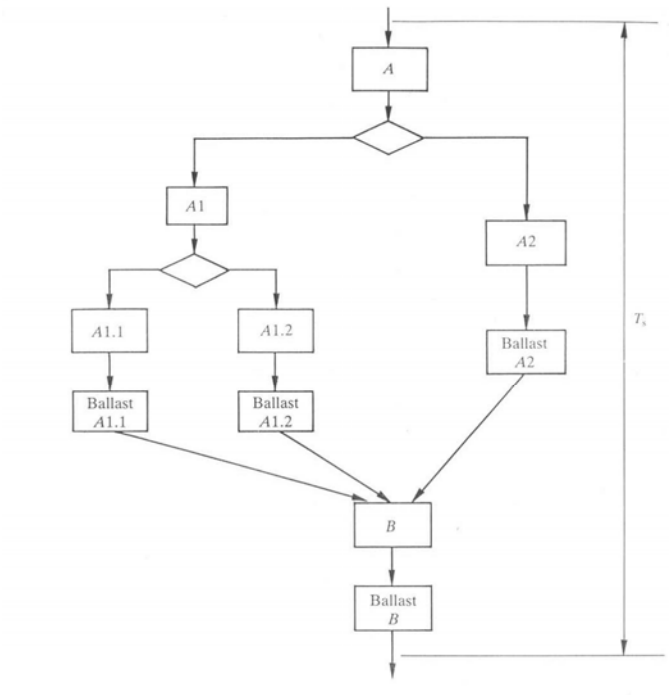

Fig. 4.2 Ballast coding.

Servicios de temporización

- Reloj de tiempo real y temporizadores
  - Reloj de tiempo real:
    - Referencia de tiempos (hora y calendario)
    - Permite medir intervalos
    - Permite el funcionamiento de temporizadores
  - Temporizadores: Generan eventos de temporización
    - Repetitivos
    - Disparo único, con referencia absoluta o relativa

Temporización periódica basada en esperas

- ¿Esperas relativas para temporización periódica?
- Objetivo: Tan precisa como el reloj de referencia
- Primer intento:

```
Hacer Siempre
  Control
  Esperar(T)
Fin hacer
```

  - El ciclo es mayor que T

- Segundo intento:

```
Hacer Siempre
  Control
  Esperar(T-D)
Fin hacer
```

  - Coste D no constante
  - Otras perturbaciones

Temporización periódica basada en esperas

- Tercer intento:

```
Hacer Siempre
  T1 = reloj()
  Control
  T2 = reloj()
  D = T2-T1
  Esperar(T-D)
Fin hacer
```

- Errores locales:
  - Granularidad del reloj
  - Latencia de interrupciones
  - Expulsión por tareas más prioritarias
- Los errores locales son acumulativos
- No sucede con
  - Temporizadores periódicos
  - Temporizadores de disparo único con referencia absoluta

Temporización en ADA

- Package **Calendar** (obligatorio):
  - **Time**: Tipo que almacena un instante de tiempo.
  - **Duration**: Tipo que almacena un resultado de cálculos de tiempo
    - Número real en **punto fijo**. Indica segundos.
    - **Granularidad** de 20 ms como máximo
    - **Extensión** de -86400.0 a 86400.0 como mínimo
    - La resta de dos **Time** puede generar un resultado **Duration**
  - **Clock:** función que permite tener acceso al tiempo actual
- "Package" **Real_time**: opcional, con mayor resolución del reloj

Temporización en ADA

- Intervalo de tiempo invertido en un conjunto de sentencias:

```
declare
    t1, t2: Time;
    intervalo: Duration;
begin
    t1 := Clock;
    <más sentencias>
    t2 := Clock;
    intervalo := t2 - t1;
end;
```

- Retraso de tiempo **relativo** (sentencia **delay**):

```
<sentencias1>
delay 10.0;
<sentencias2>
```

- Retraso de tiempo **absoluto** (sentencia **delay until)**:

```
Comienzo := Clock;
<Sentencias1>
delay until Comienzo + 10.0;
<Sentencias2>
```


Temporización en ADA

- Temporización periódica:
  - Los retrasos **delay** introducen incertidumbres y deriva **acumulativa**.
  - Esto no sucede con una referencia absoluta: **delay until**

```
declare
    Proximo: Time;
    Intervalo: constant Duration := 7.0;
begin
    Proximo := Clock + Intervalo;
    loop
            <sentencias>
            delay until Proximo;
            Proximo := Proximo + Intervalo;
    end loop;
end;
```

- Sobretiempo con la sentencia **select**:

```
select
    accept entrada1(I: Integer) do
    <sentencias>
    end entrada;
or
    delay 5.0;
    <sentencias alternativas>
end select;
```

- Sobretiempo para abortar una actividad ya comenzada:

```
select
    delay 0.5;
    <sentencias A>
then abort
    <sentencias B>
end select;
```

Temporización en POSIX 1003.1a

- Reloj de referencia:

  time_t time(time_t *tiempo);

- Retraso relativo y temporizador:

  unsigned long sleep(unsigned segs);
  unsigned alarm(unsigned segs);

- Problemas:
  - Poca resolución (segundos)
  - No hay temporizadores periódicos ni con referencia absoluta
  - **alarm** y **sleep** pueden interaccionar (SIGALRM)

Temporización en POSIX 1003.1b

- Mejoras:
  - Mayor resolución (hasta nanosegundos)
  - Temporizadores con diversos modos
  - Asignación flexible de señales a eventos
  - Detección de “overruns”
- Reloj de referencia (CLOCK_REALTIME):

  int clock_gettime(clockid_t reloj, struct timespec *tiempo);
  int clock_getres(clockid_t reloj, struct timespec *resol);

- Retraso relativo más preciso:

  int nanosleep(struct timespec *retraso, struct timespec *queda);

Temporización en POSIX 1003.1b

- Temporizadores:
  - Creación: Hay que especificar el reloj de referencia y el modo de notificación del evento

```
int timer_create(clockid_t reloj, struct sigevent *aviso,
                 timer_t *tempo);
```

  - Programación:
    - Dos valores: **it_value** e **it_interval**, en **struct itimerspec**
    - Flag TIMER_ABSTIME

```
int timer_settime(timer_t *tempo, int flags,
                  const struct itimerspec *spec,
                  struct itimerspec *spec_ant);
```


Temporización en POSIX 1003.1b

- Programación de temporizadores:
  - Si **it_value** = 0:
    - Temporizador desactivado
  - Si **it_value** != 0:
    - **it_interval** = 0: Temporizador de un solo disparo
    - **it_interval** != 0: Temporizador periódico
  - Significado:
    - **it_value**: Define el instante del primer evento
    - **it_interval**: Define el periodo de activación
  - Flag TIMER_ABSTIME: **it_value** se interpreta como un instante de tiempo absoluto
- No es necesario destruir un temporizador para reprogramarlo

Programación de un temporizador

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

timer_t mitemp;
struct sigaction accion;
siginfo_t info;
struct sigevent eventos;
struct itimerspec prog;
struct timespec ciclo;
sigset_t ev_tempo;
......
/* Programación: Primer disparo a los 1sg
      250mseg.; ciclo de 1 sg. 250 mseg. */
ciclo.tv_sec = 1;
ciclo.tv_nsec = 250000000L;
prog.it_value = ciclo;
prog.it_interval = ciclo;

/* SIGRTMIN de tiempo real */
accion.sa_sigaction = manejador;
accion.sa_flags = SA_SIGINFO;
sigemptyset(&accion.sa_flags);
sigaction(SIGRTMIN, &accion, NULL);

/* El temporizador avisará con una señal SIGRTMIN */
eventos.sigev_signo = SIGRTMIN;
eventos.sigev_notify = SIGEV_SIGNAL;
eventos.sigev_value.sival_ptr = (void *)&mitemp;

/* Bloqueo de SIGRTMIN para usar sigwaitinfo */
sigemptyset(&ev_tempo);
sigaddset(&ev_tempo, SIGRTMIN);
sigprocmask(SIG_BLOCK, &ev_tempo, NULL);

/* Temporizador creado y activado */
timer_create(CLOCK_REALTIME, &eventos, &mitemp);
timer_settime(mitemp, 0, &prog, NULL);

(…)
/* Esperando señal (vencimiento del temporizador) */
sigwaitinfo(&ev_tempo, &info);
(…)

/*Destrucción del temporizador */
timer_delete(mitemp);
....
```


Otras llamadas útiles

- Lectura del estado del temporizador:
  - **it_value**: Tiempo hasta próxima activación
  - **it_interval**: Intervalo real programado

    int timer_gettime(timer_t tempo, struct itimerspec *queda);

- Lectura de la cuenta de “overrun”:
  - Número de señales generadas desde que se encoló la última recibida:

    int timer_getoverrun(timer_t *tempo);

- Convertir fecha a **time_t**:

  time_t mktime(struct tm *fecha);