Sistemas Operativos I
Manual de prácticas

Grupo de Sistemas Operativos (DSIC/DISCA)

Práctica 4: Programación con hilos

OBJETIVOS DE LA PRÁCTICA

Esta práctica presenta tres objetivos básicos. El primero consiste en que el alumno se habitúe a la codificación de programas concurrentes (multi-hilo), comprobando las diferencias existentes entre este tipo de programas y los programas clásicos secuenciales.

Como segundo objetivo, se permitirá al alumno practicar las dos formas básicas de implementación de programas concurrentes: con soporte del lenguaje de programación (en este caso, Ada95) y con soporte del sistema operativo (utilizando una interfaz POSIX mediante el lenguaje C).

Por último, se pretende también que el alumno se enfrente con la problemática de la sección crítica mediante un caso práctico de acceso concurrente a recursos (en este caso, variables en memoria) sin mecanismos de protección.

Para ello, la práctica se ha dividido en dos partes, cada una de ellas ilustrada por un programa concurrente a ejecutar por el alumno: “hola mundo concurrente” y “concurrencia sin protección”.

ANTES DE EMPEZAR...

Los ficheros fuente que menciona la práctica están situados en un archivo comprimido (extensión `tgz`) denominado `/practicas/asignaturas/so1/pr4/pr4.tgz`. Antes de empezar, cópiatelo en un directorio tuyo y descomprímelo. Por ejemplo:

```
bash$ cd
bash$ mkdir practica4
bash$ cp /practicas/asignaturas/so1/pr4/pr4.tgz practica4
bash$ cd practica4
bash$ tar zxvf pr4.tgz
```

Eso debería crear dos subdirectorios denominados `hmconc` y `sinprot` en el directorio `practica4` (en este caso), que contienen los archivos necesarios. A partir de ahí, puedes seguir con la práctica. Que te diviertas.

PARTE 1: HOLA MUNDO CONCURRENTE

INTRODUCCIÓN

El programa *Hola Mundo!* es habitualmente el primer ejemplo que se suele dar para ilustrar un lenguaje de programación. Este programa simplemente escribe por pantalla "hola mundo" y acaba.

En este caso, se utiliza este ejemplo clásico como primera toma de contacto de programas concurrentes. Este programa se ha bautizado como "Hola Mundo Concurrente". En concreto, se proporcionan dos versiones este programa, que permitirán ilustrar las dos aproximaciones básicas para programar con hilos de ejecución: soporte a nivel de lenguaje de alto nivel (Ada95) y soporte a nivel de primitivas del sistema operativo (hilos POSIX, programados desde C).

ACTIVIDADES

ACTIVIDAD 1: HOLA MUNDO CONCURRENTE EN C

Entra en el directorio `hmconc/c` y abre el archivo `hmconc.c` desde `emacs` (o tu editor favorito). Tómate un momento para estudiar su código. En especial, comprueba cómo la creación de hilos se realiza **explícitamente** desde el programa principal.

Pregunta: ¿Qué función se utiliza para crear cada hilo? ¿Con qué parámetros?

Intenta predecir qué se visualizará en pantalla al ejecutar dicho programa. Compílalo (mediante la orden `make` en el directorio donde reside el fichero fuente) y ejecútalo.

Pregunta: ¿Que consigue el programa principal al realiza un `pthread_join` sobre cada uno de los hilos que ha creado? Prueba qué ocurriría si no hiciera esas llamadas y comenta aquí tus resultados.

ACTIVIDAD 2: HOLA MUNDO CONCURRENTE EN ADA95

Ahora entra en el directorio `hmconc/ada` y abre el archivo `hmconc.adb`. Estudia su código un momento. Fíjate sobre todo cómo la creación de los hilos se realiza **implícitamente** mediante la declaración de variables un tipo tarea.

Pregunta: ¿Cómo se llama el tipo tarea a partir del cual se crean las tareas Ada? ¿Cómo se llaman dichas tareas? ¿Cómo podríamos crear más?

Intenta predecir qué se visualizará en pantalla al ejecutar dicho programa. Compílalo (mediante la orden `make` en el directorio donde reside el fichero fuente) y ejecútalo.

Pregunta: ¿Qué diferencias has observado respecto a la ejecución del programa en C? ¿Qué instrucción ejecuta el programa principal para esperar la finalización de las tareas?

PARTE 2: CONCURRENCIA SIN PROTECCIÓN

INTRODUCCIÓN

El problema de la *sección crítica* puede darse en sistemas en las que procesos (o hilos) concurrentes acceden sin mecanismos de protección explícitos a recursos comunes que sólo son pueden ser utilizados en serie. En recursos de este tipo, se debe garantizar de alguna manera que su acceso por parte de los procesos se serializa completamente, pues de lo contrario se puede incurrir en una inconsistencia del propio recurso. Tal vez el ejemplo más evidente de tales recursos son variables en memoria compartidas por varios procesos o hilos. Esta problemática, que se analiza con profundidad en el Tema 4 de teoría, se presenta en esta práctica de forma experimental.

Para ilustrar la necesidad de los mecanismos de protección en estos casos, se han elaborado tres programas muy similares, denominados `sinprot1`, `sinprot2` y `sinprot3`. Los tres programas poseen internamente varios hilos que acceden concurrentemente (y sin protección) a una variable compartida `V`, cuyo valor inicial es 100 en todos los programas. En concreto, cada programa posee:

- un hilo (denominado `agrega`) que, dentro de un bucle, se dedica a sumar 1 a la variable `V`;
- un segundo hilo (denominado `resta`) que, dentro de un bucle, va restando 1 a la misma variable;
- y un tercer hilo (denominado `inspecciona`) que se despierta cada segundo e imprime por pantalla el valor actual de `V`.

Existen versiones de los tres programas en Ada95 y en C/POSIX, para que se vea que el problema de la sección crítica es el mismo independientemente del modelo de programación concurrente elegido.

ACTIVIDADES

ACTIVIDAD 1: PRIMERA VERSIÓN DE SINPROT

Estudia el código de los programas `sinprot1.c` (en directorio `sinprot/c`) y `sinprot1.adb` (en el directorio `sinprot/ada`). Para ello, lo más cómodo es que tengas en pantalla dos `emacs`, uno con cada fichero. Nota que ambos programas son equivalentes. Fíjate sobre todo en el código que ejecutan las tareas `agrega` y `resta`, a continuación (primero en C y luego en Ada95):

```
void *agrega (void *argumento) {
  long int cont;

  for (cont = 0; cont < REPETICIONES; cont = cont + 1) {
     V = V + 1;
     retraso(DORMIR);
  }

  printf("-------> Fin AGREGA (V = %ld)\n", V);
  pthread_exit(0);
}
```

```
void *resta (void *argumento) {
  long int cont;

  for (cont = 0; cont < REPETICIONES; cont = cont + 1) {
     V = V - 1;
     retraso(DORMIR);
  }

  printf("-------> Fin RESTA (V = %ld)\n", V);
  pthread_exit(0);
}
```

```
task body AGREGA is
  Cont : Integer := 0;

  begin
    loop
      V := V + 1;
      delay(Dormir);
      Cont := Cont + 1;
      exit when Cont >= Repeticiones;
    end loop;

    Put("-------> Fin AGREGA  (V ="); Put(V); Put(" )");New_Line;
end AGREGA;

task body RESTA is
  Cont : Integer := 0;

  begin
    loop
      V := V - 1;
      delay(Dormir);
      Cont := Cont + 1;
      exit when Cont >= Repeticiones;
    end loop;

    Put("-------> Fin RESTA  (V ="); Put(V); Put(" )");New_Line;
end RESTA;
```

De acuerdo con el propio código, y antes de ejecutar ninguno de ambos, intenta predecir qué se verá por pantalla cuando los ejecutes. **Pregunta**: ¿Será 100 el valor final de la variable `V`? Justifica por qué piensas que sí (o que no).

Compila (mediante `make`) y ejecuta ambos programas y comenta lo que se observa en pantalla, así como el valor final de `V`. ¿Existen diferencias respecto a lo que habías predicho? ¿Cuáles son?

ACTIVIDAD 2: SEGUNDA VERSIÓN DE SINPROT

Estudia el código de los programas `sinprot2.c` (en directorio `sinprot/c`) y `sinprot2.adb` (en el directorio `sinprot/ada`). Nota que ambos son equivalentes. Fíjate especialmente en el código que ejecutan las tareas `agrega` y `resta`, a continuación (primero en C y luego en Ada95):

```
void *agrega (void *argumento) {
  long int cont;

  for (cont = 0; cont < REPETICIONES; cont = cont + 1) {
     V = V + 1;
  }

  printf("-------> Fin AGREGA (V = %ld)\n", V);
  pthread_exit(0);
}


void *resta (void *argumento) {
  long int cont;

  for (cont = 0; cont < REPETICIONES; cont = cont + 1) {
     V = V - 1;
  }

  printf("-------> Fin RESTA  (V = %ld)\n", V);
  pthread_exit(0);
}
```

```
task body AGREGA is
  Cont : Integer := 0;

  begin
    loop
      V := V + 1;
      Cont := Cont + 1;
      exit when Cont >= Repeticiones;
    end loop;

    Put("-------> Fin AGREGA  (V ="); Put(V); Put(" )");New_Line;
end AGREGA;


task body RESTA is
  Cont : Integer := 0;

  begin
    loop
      V := V - 1;
      Cont := Cont + 1;
      exit when Cont >= Repeticiones;
    end loop;

    Put("-------> Fin RESTA  (V ="); Put(V); Put(" )");New_Line;
end RESTA;
```

Piensa detenidamente qué diferencias existen entre estas versiones y las de la actividad anterior. **Pregunta**: ¿En qué pueden afectar los cambios?

De acuerdo con el propio código, y antes de ejecutar ninguno de ambos, intenta predecir qué se verá por pantalla cuando los ejecutes. **Pregunta**: ¿Será 100 el valor final de la variable `V`? Justifica por qué piensas que sí (o que no).

Compila (mediante `make`) y ejecuta ambos programas y comenta lo que se observa en pantalla, así como el valor final de `V`. **Pregunta:** ¿Existen diferencias respecto a lo que habías predicho? ¿Cuáles son?

ACTIVIDAD 3: TERCERA VERSIÓN DE SINPROT

Estudia el código de los programas `sinprot3.c` (en directorio `sinprot/c`) y `sinprot3.adb` (en el directorio `sinprot/ada`). Nota que ambos son equivalentes. Fíjate sobre todo en el código que ejecutan las tareas `agrega` y `resta`, a continuación (primero en C y luego en Ada95):

```
void *agrega (void *argumento) {
  long int cont;
  long int inter;

  for (cont = 0; cont < REPETICIONES; cont = cont + 1) {
    inter = V;
    inter = inter + 1;
    V     = inter;
  }

  printf("-------> Fin AGREGA (V = %ld)\n", V);
  pthread_exit(0);
}
```

```
void *resta (void *argumento) {
  long int cont;
  long int inter;

  for (cont = 0; cont < REPETICIONES; cont = cont + 1) {
    inter = V;
    inter = inter - 1;
    V     = inter;
  }

  printf("-------> Fin RESTA  (V = %ld)\n", V);
  pthread_exit(0);
}
```

```
task body AGREGA is
  Cont  : Integer := 0;
  Inter : Integer;

  begin
    loop
      Inter := V;
      Inter := Inter + 1;
      V     := Inter;
      Cont  := Cont + 1;
      exit when Cont >= Repeticiones;
    end loop;

    Put("-------> Fin AGREGA  (V ="); Put(V); Put(" )");New_Line;
end AGREGA;


task body RESTA is
  Cont : Integer := 0;
  Inter : Integer;

  begin
    loop
      Inter := V;
      Inter := Inter - 1;
      V     := Inter;
      Cont := Cont + 1;
      exit when Cont >= Repeticiones;
    end loop;

    Put("-------> Fin RESTA  (V ="); Put(V); Put(" )");New_Line;
end RESTA;
```

Piensa detenidamente qué diferencias existen entre estas versiones y las de la actividad anterior. **Pregunta**: ¿Crees que esta versión se comportará como la anterior? Justifícalo.

De acuerdo con el propio código, y antes de ejecutar ninguno de ambos, intenta predecir qué se verá por pantalla cuando los ejecutes. **Pregunta**: ¿Será 100 el valor final de la variable `V`? Justifica por qué piensas que sí (o que no).

Compila (mediante `make`) y ejecuta ambos programas y comenta lo que se observa en pantalla, así como el valor final de `V`. **Pregunta:** ¿Existen diferencias respecto a lo que habías predicho? ¿Cuáles son?

ANEXO: CÓDIGO FUENTE DE LOS PROGRAMAS

HOLA MUNDO CONCURRENTE (C)

```
/*                                                                      */
/*  Hmconc (Hola Mundo Concurrente)                                     */
/*  Sistemas Operativos 1                                               */
/*                                                                      */
/*  Grupo de Sistemas Operativos DSIC/DISCA (UPV)                       */
/*  Ultima revision: 17/07/2001                                         */
/*                                                                      */

#include <stdio.h>
#include <pthread.h>


/*
    CONSTANTES

    UN_SEGUNDO   : Un segundo expresado en milisegundos
*/
#define UN_SEGUNDO         1000

/*
    TIPOS DE USUARIO

    argumentos_tarea_t : Estructura con los argumentos iniciales de
                         las tareas
*/
typedef struct argumentos_tarea_t {
  int   id;              /* Identificador de la tarea */
  int   ttl;             /* Time To Live  (numero de iteraciones) */
} argumentos_tarea_t;

/*
   FUNCIONES AUXILIARES

   retraso(int milisegunos) : Suspende a la tarea invocante durante
                              los milisegundos especificados
   imprimir(int id,
            char *cadena)   : Imprime en pantalla la cadena a partir
                              de la columna id*2
*/
void retraso (int milisegundos) {

  struct timespec ts;

  if (milisegundos > 0) {
    ts.tv_sec  = milisegundos / 1000;
    ts.tv_nsec = (milisegundos % 1000) * 1000000;
    nanosleep (&ts, NULL);
  }
}

void imprimir (int id, char *cadena) {

  int col; // columna en la que tenemos que empezar a imprimir (id*2)

  // Dejamos los espacios indicados al principio:
  for (col = 0; col < (id*2); col = col + 1) {
    fprintf(stderr, " ");
  }

  // Imprimimos el mensaje
  fprintf(stderr, "[Tarea %d]: %s\n\n", id, cadena);
}

/*
   FUNCIONES QUE EJECUTAN LAS TAREAS
   - Saluda : Nace (saluda), itera y muere
*/
void *saluda (void *argumentos) {

  // Variables locales de la tarea:
  struct argumentos_tarea_t *arg;
  int    id;
  int    ttl;
```

```
  int    i = 0;

  // Recuperamos los argumentos con los que se ha creado la tarea
  arg = (argumentos_tarea_t *) argumentos;
  id  = arg->id;
  ttl = arg->ttl;

  // Mensaje de nacimiento
  imprimir(id, "Hoooola mundo, estoy viva!!");

  // Bucle en el que se espera un poco e imprime
  // (Cada tarea itera tanto como su TTL)
  while (i < ttl) {

    retraso(UN_SEGUNDO);
    imprimir(id, "Sigo viva");
    i = i+1;
  }
  imprimir(id, "Muero");
  pthread_exit(0);
}


/*
   PROGRAMA PRINCIPAL

*/
int main (void) {

  struct argumentos_tarea_t arg1, arg2, arg3;
  pthread_t                 t_saluda1, t_saluda2, t_saluda3;
  pthread_attr_t            atributos;


  // Primero saludamos (ante todo, educacion!)
  fprintf(stderr, "[PRINCIPAL]: HOLA MUNDO!!\n\n");


  // Definimos el valor de los atributos de las tareas (por defecto)
  pthread_attr_init(&atributos);


  // Preparamos los argumentos iniciales cada tarea y la creamos:
  // (Nota: podriamos haber reutilizado "arg1" en las tres llamadas)
  arg1.id  = 1;
  arg1.ttl = 10;
  pthread_create(&t_saluda1, &atributos, saluda, &arg1);

  arg2.id  = 3;
  arg2.ttl = 5;
  pthread_create(&t_saluda2, &atributos, saluda, &arg2);

  arg3.id  = 6;
  arg3.ttl = 4;
  pthread_create(&t_saluda3, &atributos, saluda, &arg3);


  // Obligamos a que el programa principal se espere a que todas
  // las tareas finalicen
  pthread_join(t_saluda1, NULL);
  pthread_join(t_saluda2, NULL);
  pthread_join(t_saluda3, NULL);

  // Fin del programa principal
  exit(0);
}
```

HOLA MUNDO CONCURRENTE (ADA95)

```
--
-- Hmconc (Hola Mundo Concurrente)
-- Sistemas Operativos 1
--
-- Grupo de Sistemas Operativos DSIC/DISCA (UPV)
-- Ultima revision: 14/07/2001
--

with Ada.Text_IO;
use  Ada.Text_IO;
with Ada.Integer_Text_IO;
use  Ada.Integer_Text_IO;


procedure Hmconc is

   ------------------------------------------------------------------
   -- CONSTANTES
   --
   -- UN_SEGUNDO : Retardo interno de las tareas
   --
   UN_SEGUNDO : constant Duration := Duration(1);

------------------------------------------------------------------
   -- ESPECIFICACION DEL TIPO TAREA TAREA_SALUDADORA
   --
   -- Id      : identificador de la tarea
   -- TTL     : Time To Live  (numero de iteraciones)
   --
   task type TAREA_SALUDADORA (Id: Positive; TTL: Positive);

   ------------------------------------------------------------------
   -- IMPLEMENTACION DEL TIPO TAREA TAREA_SALUDADORA
   --
   task body TAREA_SALUDADORA is

      -- Procedimientos auxiliares de la tarea
      --
      procedure Imprimir(Id: in Positive; Mensaje: in String) is
      begin
         -- Cada tarea imprime en una columna distinta (Id*2):
         --
         Set_Col(Positive_Count(Id*2));
         Put("[Tarea "); Put(Id,1); Put("]: ");
         Put(Mensaje);
         New_Line;
         New_Line;
      end Imprimir;

      -- Variables locales de la tarea
      --
      I : Integer := 0;

   -- Cuerpo de la tarea
   --
   begin

      -- Mensaje de nacimiento
      --
      Imprimir(Id, "Hoooola mundo, estoy viva!!");

      -- Bucle en el que se espera un poco e imprime
      -- Cada tarea itera tanto como su TTL
      --
      loop

         delay(UN_SEGUNDO);
         Imprimir(Id, "Sigo viva");

         I := I + 1;
         exit when I >= TTL;

      end loop;

      Imprimir(Id, "Muero");

   end TAREA_SALUDADORA;
```

```
------------------------------------------------------------------------
------------------------------------------------------------------------
-- INICIO DEL PROCEDIMIENTO PRINCIPAL
--
begin


   declare
      -- Declaracion/Creacion de las tareas:
      --
      Saluda1 : TAREA_SALUDADORA(1,10);
      Saluda2 : TAREA_SALUDADORA(3,5);
      Saluda3 : TAREA_SALUDADORA(6,4);


   begin
      Put("[PRINCIPAL]: HOLA MUNDO!!");
      New_Line;
      New_Line;
   end;


end Hmconc;
```

SIN PROTECCIÓN 1 (C)

```
/*                                                                      */
/*  Sinprot1 (Acceso concurrente sin proteccion, Ejemplo 1)             */
/*  Sistemas Operativos 1                                               */
/*                                                                      */
/*  Grupo de Sistemas Operativos DSIC/DISCA (UPV)                       */
/*  Ultima revision: 14/07/2001                                         */
/*                                                                      */
#include <stdio.h>
#include <pthread.h>

/*
    CONSTANTES

    REPETICIONES : Numero de veces que se suma/resta 1 a V
    DORMIR       : Milisegundos que la tarea que suma/resta duerme
                   en cada iteracion

    UN_SEGUNDO   : Un segundo expresado en milisegundos
    CUENTA_ATRAS : Tiempo que dura la inspeccion del valor de V
*/
#define REPETICIONES       1000
#define DORMIR                1
#define UN_SEGUNDO         1000
#define CUENTA_ATRAS         30

/*
   VARIABLES GLOBALES (COMPARTIDAS)

   V : Variable compartida por las tareas

*/
long int V = 100; // Valor inicial


/*
   FUNCIONES AUXILIARES

   retraso(int milisegunos) : Suspende a la tarea invocante durante
                              los milisegundos especificados
*/
void retraso (int milisegundos) {

  struct timespec ts;

  if (milisegundos > 0) {
    ts.tv_sec  = milisegundos / 1000;
    ts.tv_nsec = (milisegundos % 1000) * 1000000;
    nanosleep (&ts, NULL);
  }
}
```

```
/*
   FUNCIONES QUE EJECUTAN LAS TAREAS

   - Agrega : Ejecuta un bucle donde incrementa la variable V
   - Resta  : Ejecuta un bucle donde decrementa la variable V
   - Inspecciona : Imprime cada segundo el valor de V
*/
void *agrega (void *argumento) {

  long int cont;

  for (cont = 0; cont < REPETICIONES; cont = cont + 1) {
     V = V + 1;
     retraso(DORMIR);
  }

  printf("-------> Fin AGREGA (V = %ld)\n", V);
  pthread_exit(0);
}

void *resta (void *argumento) {

  long int cont;

  for (cont = 0; cont < REPETICIONES; cont = cont + 1) {
     V = V - 1;
     retraso(DORMIR);
  }

  printf("-------> Fin RESTA  (V = %ld)\n", V);
  pthread_exit(0);
}

void *inspecciona (void *argumento) {

  int cont = CUENTA_ATRAS;

  while (cont > 0) {

    retraso(UN_SEGUNDO);

    fprintf(stderr, "Inspeccion (%d): Valor actual de V = %ld\n",
            cont, V);
    cont = cont - 1;
  }

  fprintf(stderr, "-------> VALOR FINAL: V = %ld\n\n", V);
  pthread_exit(0);
}



/*
   PROGRAMA PRINCIPAL

*/
int main (void) {

  pthread_attr_t atributos;
  pthread_t      t_agrega, t_resta, t_inspecciona;


  // Definimos el valor de los atributos de las tareas (por defecto)
  pthread_attr_init(&atributos);


  // Creamos las tres tareas con dichos atributos
  pthread_create(&t_agrega, &atributos, agrega, NULL);
  pthread_create(&t_resta, &atributos, resta, NULL);
  pthread_create(&t_inspecciona, &atributos, inspecciona, NULL);


  // Obligamos a que el programa principal se espere a que las
  // tres tareas finalicen
  pthread_join(t_agrega, NULL);
  pthread_join(t_resta, NULL);
  pthread_join(t_inspecciona, NULL);

  // Fin del programa principal
  exit(0);
}
```

SIN PROTECCIÓN 1 (ADA95)

```
--
-- Sinprot1 (Acceso concurrente sin proteccion, Ejemplo 1)
-- Sistemas Operativos 1
--
-- Grupo de Sistemas Operativos DSIC/DISCA (UPV)
-- Ultima revision: 14/07/2001
--
with Ada.Text_IO;
use  Ada.Text_IO;
with Ada.Integer_Text_IO;
use  Ada.Integer_Text_IO;

procedure SinProt1  is

   ------------------------------------------------------------------
   -- CONSTANTES
   --
   -- Repeticiones : Numero de veces que se suma/resta 1 a V
   -- Dormir       : Segundos que la tarea duerme en cada iteracion
   -- Un_Segundo   : Retardo interno de la tarea inspecciona
   -- Cuenta_Atras : Tiempo que dura la inspeccion del valor de V
   --
   Repeticiones : constant Integer  := 1000;
   Dormir       : constant Duration := 0.001;
   Un_Segundo   : constant Duration := Duration(1);
   Cuenta_Atras : constant Integer  := 30;

   ------------------------------------------------------------------
   -- VARIABLES GLOBALES (COMPARTIDAS)
   --
   -- V            : Variable compartida entre las tareas
   --
   V : Integer := 100;   -- Valor inicial

   ------------------------------------------------------------------
   -- ESPECIFICACION DE LAS TAREAS
   --

   -- AGREGA
   --
   task AGREGA;

   -- RESTA
   --
   task RESTA;

   -- INSPECCIONA
   --
   task INSPECCIONA;

   ------------------------------------------------------------------
   -- IMPLEMENTACION DE LAS TAREAS
   --

   -- AGREGA
   --
   task body AGREGA is

      -- Variables locales de la tarea
      --
      Cont : Integer := 0;

   -- Cuerpo de la tarea
   --
   begin

      loop

         V := V + 1;
         delay(Dormir);

         Cont := Cont + 1;
         exit when Cont >= Repeticiones;

      end loop;

      Put("-------> Fin AGREGA  (V ="); Put(V); Put(" )");New_Line;

   end AGREGA;
```

```
   -- RESTA
   --
   task body RESTA is

      -- Variables locales de la tarea
      --
      Cont : Integer := 0;

   -- Cuerpo de la tarea
   --
   begin

      loop

         V := V - 1;
         delay(Dormir);

         Cont := Cont + 1;
         exit when Cont >= Repeticiones;

      end loop;
      Put("-------> Fin RESTA  (V ="); Put(V); Put(" )");New_Line;

   end RESTA;


   -- INSPECCIONA
   --
   task body INSPECCIONA is

      -- Variables locales de la tarea
      --
      Cont : Integer := Cuenta_Atras; -- Iteraciones de la tarea

   -- Cuerpo de la tarea
   --
   begin

      loop
         delay(Un_Segundo);

         Put("Inspeccion ("); Put(Cont,2); Put("): Valor actual de V =");
         Put(V); New_Line;

         Cont := Cont - 1;
         exit when Cont = 0;

      end loop;

      Put("-------> VALOR FINAL: V =");Put(V);
      New_Line; New_Line;

   end INSPECCIONA;




------------------------------------------------------------------------
------------------------------------------------------------------------
-- INICIO DEL PROCEDIMIENTO PRINCIPAL
--
begin
   null;
end SinProt1;
```

SIN PROTECCIÓN 2 (C)

```
/*                                                                      */
/*  Sinprot2 (Acceso concurrente sin proteccion, Ejemplo 2)             */
/*  Sistemas Operativos 1                                               */
/*                                                                      */
/*  Grupo de Sistemas Operativos DSIC/DISCA (UPV)                       */
/*  Ultima revision: 14/07/2001                                         */
/*                                                                      */
#include <stdio.h>
#include <pthread.h>

/*
    CONSTANTES

    REPETICIONES : Numero de veces que se suma/resta 1 a V
    UN_SEGUNDO   : Un segundo expresado en milisegundos
    CUENTA_ATRAS : Tiempo que dura la inspeccion del valor de V
*/
#define REPETICIONES  300000000
#define UN_SEGUNDO         1000
#define CUENTA_ATRAS         30

/*
   VARIABLES GLOBALES (COMPARTIDAS)

   V : Variable compartida por las tareas
*/
long int V = 100; // Valor inicial


/*
   FUNCIONES AUXILIARES

   retraso(int milisegunos) : Suspende a la tarea invocante durante
                              los milisegundos especificados
*/
void retraso (int milisegundos) {

  struct timespec ts;

  if (milisegundos > 0) {
    ts.tv_sec  = milisegundos / 1000;
    ts.tv_nsec = (milisegundos % 1000) * 1000000;
    nanosleep (&ts, NULL);
  }
}

/*
   FUNCIONES QUE EJECUTAN LAS TAREAS

   - Agrega : Ejecuta un bucle donde incrementa la variable V
   - Resta  : Ejecuta un bucle donde decrementa la variable V
   - Inspecciona : Imprime cada segundo el valor de V
*/
void *agrega (void *argumento) {

  long int cont;

  for (cont = 0; cont < REPETICIONES; cont = cont + 1) {
     V = V + 1;
  }

  printf("-------> Fin AGREGA (V = %ld)\n", V);
  pthread_exit(0);
}

void *resta (void *argumento) {

  long int cont;

  for (cont = 0; cont < REPETICIONES; cont = cont + 1) {
     V = V - 1;
  }

  printf("-------> Fin RESTA  (V = %ld)\n", V);
  pthread_exit(0);
}
```

```
void *inspecciona (void *argumento) {

  int cont = CUENTA_ATRAS;

  while (cont > 0) {

    retraso(UN_SEGUNDO);

    fprintf(stderr, "Inspeccion (%d): Valor actual de V = %ld\n",
            cont, V);
    cont = cont - 1;
  }

  fprintf(stderr, "-------> VALOR FINAL: V = %ld\n\n", V);
  pthread_exit(0);
}

/*
   PROGRAMA PRINCIPAL

*/
int main (void) {

  pthread_attr_t atributos;
  pthread_t      t_agrega, t_resta, t_inspecciona;


  // Definimos el valor de los atributos de las tareas (por defecto)
  pthread_attr_init(&atributos);


  // Creamos las tres tareas con dichos atributos
  pthread_create(&t_agrega, &atributos, agrega, NULL);
  pthread_create(&t_resta, &atributos, resta, NULL);
  pthread_create(&t_inspecciona, &atributos, inspecciona, NULL);


  // Obligamos a que el programa principal se espere a que las
  // tres tareas finalicen
  pthread_join(t_agrega, NULL);
  pthread_join(t_resta, NULL);
  pthread_join(t_inspecciona, NULL);

  // Fin del programa principal
  exit(0);
}
```

SIN PROTECCIÓN 2 (ADA95)

```
--
-- Sinprot2 (Acceso concurrente sin proteccion, Ejemplo 2)
-- Sistemas Operativos 1
--
-- Grupo de Sistemas Operativos DSIC/DISCA (UPV)
-- Ultima revision: 14/07/2001
--
with Ada.Text_IO;
use  Ada.Text_IO;
with Ada.Integer_Text_IO;
use  Ada.Integer_Text_IO;

procedure SinProt2  is



   ------------------------------------------------------------------
   -- CONSTANTES
   --
   -- Repeticiones : Numero de veces que se suma/resta 1 a V
   -- Un_Segundo   : Retardo interno de la tarea inspecciona
   -- Cuenta_Atras : Tiempo que dura la inspeccion del valor de V
   --
   Repeticiones : constant Integer  := 1000000000;
   Un_Segundo   : constant Duration := Duration(1);
   Cuenta_Atras : constant Integer  := 30;
```

```
-------------------------------------------------------------------
-- VARIABLES GLOBALES (COMPARTIDAS)
--
-- V            : Variable compartida entre las tareas
--
V : Integer := 100;   -- Valor inicial

-------------------------------------------------------------------
-- ESPECIFICACION DE LAS TAREAS
--

-- AGREGA
--
task AGREGA;

-- RESTA
--
task RESTA;

-- INSPECCIONA
--
task INSPECCIONA;

-------------------------------------------------------------------
-- IMPLEMENTACION DE LAS TAREAS
--
-- AGREGA
--
task body AGREGA is

   -- Variables locales de la tarea
   --
   Cont : Integer := 0;

-- Cuerpo de la tarea
--
begin

   loop

      V := V + 1;
      Cont := Cont + 1;

      exit when Cont >= Repeticiones;

   end loop;
   Put("-------> Fin AGREGA  (V ="); Put(V); Put(" )");New_Line;

end AGREGA;

-- RESTA
--
task body RESTA is

   -- Variables locales de la tarea
   --
   Cont : Integer := 0;

-- Cuerpo de la tarea
--
begin

   loop

      V := V - 1;
      Cont := Cont + 1;

      exit when Cont >= Repeticiones;

   end loop;
   Put("-------> Fin RESTA  (V ="); Put(V); Put(" )");New_Line;

end RESTA;


-- INSPECCIONA
--
task body INSPECCIONA is

   -- Variables locales de la tarea
   --
   Cont : Integer := Cuenta_Atras; -- Iteraciones de la tarea
```

```
   -- Cuerpo de la tarea
   --
   begin

      loop
         delay(Un_Segundo);

         Put("Inspeccion ("); Put(Cont,2); Put("): Valor actual de V =");
         Put(V); New_Line;

         Cont := Cont - 1;
         exit when Cont = 0;

      end loop;

      Put("-------> VALOR FINAL: V =");Put(V);
      New_Line; New_Line;

   end INSPECCIONA;



------------------------------------------------------------------------
------------------------------------------------------------------------
-- INICIO DEL PROCEDIMIENTO PRINCIPAL
--
begin
   null;
end SinProt2;
```

SIN PROTECCIÓN 3 (C)

```
/*                                                                      */
/*  Sinprot3 (Acceso concurrente sin proteccion, Ejemplo 3)             */
/*  Sistemas Operativos 1                                               */
/*                                                                      */
/*  Grupo de Sistemas Operativos DSIC/DISCA (UPV)                       */
/*  Ultima revision: 14/07/2001                                         */
/*                                                                      */


#include <stdio.h>
#include <pthread.h>



/*
    CONSTANTES

    REPETICIONES : Numero de veces que se suma/resta 1 a V
    UN_SEGUNDO   : Un segundo expresado en milisegundos
    CUENTA_ATRAS : Tiempo que dura la inspeccion del valor de V
*/

#define REPETICIONES  300000000
#define UN_SEGUNDO         1000
#define CUENTA_ATRAS         30


/*
   VARIABLES GLOBALES (COMPARTIDAS)

   V : Variable compartida por las tareas

*/

long int V = 100; // Valor inicial



/*
   FUNCIONES AUXILIARES

   retraso(int milisegunos) : Suspende a la tarea invocante durante
                              los milisegundos especificados

*/
```

```
void retraso (int milisegundos) {

  struct timespec ts;

  if (milisegundos > 0) {
    ts.tv_sec  = milisegundos / 1000;
    ts.tv_nsec = (milisegundos % 1000) * 1000000;
    nanosleep (&ts, NULL);
  }
}



/*
   FUNCIONES QUE EJECUTAN LAS TAREAS

   - Agrega : Ejecuta un bucle donde incrementa la variable V

   - Resta  : Ejecuta un bucle donde decrementa la variable V

   - Inspecciona : Imprime cada segundo el valor de V

*/


void *agrega (void *argumento) {

  long int cont;
  long int inter;

  for (cont = 0; cont < REPETICIONES; cont = cont + 1) {
    inter = V;
    inter = inter + 1;
    V     = inter;
  }

  printf("-------> Fin AGREGA (V = %ld)\n", V);
  pthread_exit(0);
}



void *resta (void *argumento) {

  long int cont;
  long int inter;

  for (cont = 0; cont < REPETICIONES; cont = cont + 1) {
    inter = V;
    inter = inter - 1;
    V     = inter;
  }

  printf("-------> Fin RESTA  (V = %ld)\n", V);
  pthread_exit(0);
}



void *inspecciona (void *argumento) {

  int cont = CUENTA_ATRAS;

  while (cont > 0) {

    retraso(UN_SEGUNDO);

    fprintf(stderr, "Inspeccion (%d): Valor actual de V = %ld\n",
            cont, V);
    cont = cont - 1;
  }

  fprintf(stderr, "-------> VALOR FINAL: V = %ld\n\n", V);
  pthread_exit(0);
}
```

```
/*
   PROGRAMA PRINCIPAL

*/
int main (void) {

  pthread_attr_t atributos;
  pthread_t      t_agrega, t_resta, t_inspecciona;


  // Definimos el valor de los atributos de las tareas (por defecto)
  pthread_attr_init(&atributos);


  // Creamos las tres tareas con dichos atributos
  pthread_create(&t_agrega, &atributos, agrega, NULL);
  pthread_create(&t_resta, &atributos, resta, NULL);
  pthread_create(&t_inspecciona, &atributos, inspecciona, NULL);


  // Obligamos a que el programa principal se espere a que las
  // tres tareas finalicen
  pthread_join(t_agrega, NULL);
  pthread_join(t_resta, NULL);
  pthread_join(t_inspecciona, NULL);

  // Fin del programa principal
  exit(0);
}
```

SIN PROTECCIÓN 3 (ADA95)

```
--
-- Sinprot3 (Acceso concurrente sin proteccion, Ejemplo 3)
-- Sistemas Operativos 1
--
-- Grupo de Sistemas Operativos DSIC/DISCA (UPV)
-- Ultima revision: 14/07/2001
--

with Ada.Text_IO;
use  Ada.Text_IO;
with Ada.Integer_Text_IO;
use  Ada.Integer_Text_IO;


procedure SinProt3  is

   ------------------------------------------------------------------
   -- CONSTANTES
   --
   -- Repeticiones : Numero de veces que se suma/resta 1 a V
   -- Un_Segundo   : Retardo interno de la tarea inspecciona
   -- Cuenta_Atras : Tiempo que dura la inspeccion del valor de V
   --
   Repeticiones : constant Integer  := 300000000;
   Un_Segundo   : constant Duration := Duration(1);
   Cuenta_Atras : constant Integer  := 30;


   ------------------------------------------------------------------
   -- VARIABLES GLOBALES (COMPARTIDAS)
   --
   -- V            : Variable compartida entre las tareas
   --
   V : Integer := 100;   -- Valor inicial

   ------------------------------------------------------------------
   -- ESPECIFICACION DE LAS TAREAS
   --

   -- AGREGA
   --
   task AGREGA;
```

```
   -- RESTA
   --
   task RESTA;

   -- INSPECCIONA
   --
   task INSPECCIONA;


   ------------------------------------------------------------------
   -- IMPLEMENTACION DE LAS TAREAS
   --

   -- AGREGA
   --
   task body AGREGA is

      -- Variables locales de la tarea
      --
      Cont  : Integer := 0;
      Inter : Integer;

   -- Cuerpo de la tarea
   --
   begin

      loop

         Inter := V;
         Inter := Inter + 1;
         V     := Inter;

         Cont  := Cont + 1;

         exit when Cont >= Repeticiones;

      end loop;
      Put("-------> Fin AGREGA  (V ="); Put(V); Put(" )");New_Line;

   end AGREGA;



   -- RESTA
   --
   task body RESTA is

      -- Variables locales de la tarea
      --
      Cont : Integer := 0;
      Inter : Integer;

   -- Cuerpo de la tarea
   --
   begin

      loop

         Inter := V;
         Inter := Inter - 1;
         V     := Inter;

         Cont := Cont + 1;

         exit when Cont >= Repeticiones;

      end loop;
      Put("-------> Fin RESTA  (V ="); Put(V); Put(" )");New_Line;

   end RESTA;



   -- INSPECCIONA
   --
   task body INSPECCIONA is

      -- Variables locales de la tarea
      --
      Cont : Integer := Cuenta_Atras; -- Iteraciones de la tarea

   -- Cuerpo de la tarea
```

```
   --
   begin

      loop
         delay(Un_Segundo);

         Put("Inspeccion ("); Put(Cont,2); Put("): Valor actual de V =");
         Put(V); New_Line;

         Cont := Cont - 1;
         exit when Cont = 0;

      end loop;

      Put("-------> VALOR FINAL: V =");Put(V);
      New_Line; New_Line;

   end INSPECCIONA;



------------------------------------------------------------------------
------------------------------------------------------------------------
-- INICIO DEL PROCEDIMIENTO PRINCIPAL
--
begin
   null;
end SinProt3;
```