Concurrency & Synchronization Algorithms

Alicia y Bernardo

Descripción

Son vecinos, en dos casas que comparten el patio. Cada uno tiene un perro. Los perros desean salir al patio. Debemos sincronizarlos para que no saquen a los perros a la vez, ya que los perros se pelean. El recurso es el patio y la sección crítica, pasear al perro.

Código

```
-- Extensión del algoritmo: Charlie y Carla, con capacidad de patio 2.
with Ada.Text_IO;
use Ada.Text_IO;
-- Main
procedure main is
capacidadPatio: Integer;
-- semaforo
 task semaforo is
    entry init (x:Integer);
    entry p;
    entry v;
 end semaforo;
 task body semaforo is
     s: Integer;
 begin
     accept init (x:Integer) do
       s:= x;
      end init;
     loop
       select
         when s>0 => accept p do
           s := s-1;
         end p;
       or accept v do
           s := s+1;
         end v;
       end select;
    end loop;
 end semaforo;
 -- recursos que vinculan al patio
 procedure Alice is
```

```
begin
   loop
      semaforo.p;
      Put_Line("Alice is walking her dog"); -- sección crítica
      delay(4.0);
      semaforo.v;
      -- otras tareas
   end loop;
end Alice;
procedure Bernardo is
begin
  loop
      semaforo.p;
      Put_Line("Bernardo is walking his dog"); -- sección crítica
      delay(4.0);
      semaforo.v;
      -- otras tareas
  end loop;
end Bernardo;
procedure Charlie is
begin
  loop
      semaforo.p;
      Put_Line("Charlie is walking his dog"); -- sección crítica
      delay(4.0);
      semaforo.v;
      -- otras tareas
  end loop;
end Charlie;
procedure Carla is
begin
  loop
      semaforo.p;
      Put_Line("Carla is walking her dog"); -- sección crítica
      delay(4.0);
      semaforo.v;
      -- otras tareas
   end loop;
  end Carla;
-- paseos en el patio
procedure paseo_perro_patio is
  Task pasea_perro_Alicia;
  Task pasea_perro_Bernardo;
```

```
  Task pasea_perro_Charlie;
  Task pasea_perro_Carla;
  Task body pasea_perro_Alicia is
  begin
       Alice; -- proc Alicia
  end pasea_perro_Alicia;
  Task body pasea_perro_Bernardo is
  begin
       Bernardo; -- proc Bernardo
  end pasea_perro_Bernardo;
  Task body pasea_perro_Charlie is
  begin
       Charlie; -- proc Charlie
  end pasea_perro_Charlie;
  Task body pasea_perro_Carla is
  begin
       Carla; -- proc Carla
  end pasea_perro_Carla;
 begin
   Put("");
 end paseo_perro_patio;
begin
  -- suponemos que el paseo del perro sin importar quien lo pasee, lleva el mismo tiempo
  capacidadPatio:= 2;
  semaforo.init(capacidadPatio); -- inicializo semaforo
  paseo_perro_patio; -- se pasean los perros
end main;
```

Filósofos Comensales

Descripción

Existen 5 filósofos (procesos) sentados en una mesa, y en ella hay 5 palitos chinos (recursos) para comer arroz. Cada palito es compartido con el filósofo de al lado, tanto a la derecha como a la izquierda. Su vida es pensar y comer repetidamente. Los palitos se toman y devuelven estrictamente de a 1. En el caso que los 5 filósofos tomen el palito izquierdo, quedan todos bloqueados (deadlock).
Para evitarlo, no puede haber más de 2 filósofos comiendo a la vez (4 palitos). Entonces, la solución es repartir los 5 palitos entre los 4 filósofos que lleguen primero, y el último filósofo espera a que alguno termine de comer.

Código

```
with Ada.Text_IO;
use Ada.Text_IO;
with Ada.Strings.Fixed;
use Ada.Strings.Fixed;
-- Main
procedure filosofos is
 -- semáforo
 task type semaforo is
    entry init (x:Integer);
    entry p;
    entry v;
 end semaforo;
 task body semaforo is
     s: Integer;
 begin
     accept init (x:Integer) do
       s:= x;
      end init;
     loop
       select
         when s>0 => accept p do
           s := s-1;
         end p;
       or accept v do
           s := s+1;
         end v;
       end select;
    end loop;
 end semaforo;
 -- palitos
 palitos: Array(0..4) of semaforo;
 -- comedor
 comedor: semaforo;
 -- filósofo
  procedure filosofo (i:Integer) is
    izq : Integer;
    der : Integer;
  begin
    izq:= i;
    der:= (i + 1) mod 5;
    loop
      Put_Line("philosopher " & Ada.Strings.Fixed.Trim(Integer'Image(i), Ada.Strings.Left) & "
        is thinking"); -- asíncrono
```

```
        delay(3.0);
        comedor.p;
        palitos(izq).p;
        palitos(der).p;
        Put_Line("philosopher " & Ada.Strings.Fixed.Trim(Integer'Image(i), Ada.Strings.Left) &
          is eating"); -- síncrono
        delay(3.0);
        palitos(der).v;
        palitos(izq).v;
        comedor.v;
      end loop;
    end filosofo;
    -- comen los filósofos concurrentemente
    procedure comer_filosofos is
      task come_filosofo0;
      task come_filosofo1;
      task come_filosofo2;
      task come_filosofo3;
      task come_filosofo4;
      task body come_filosofo0 is
      begin
        filosofo(0);
      end come_filosofo0;
      task body come_filosofo1 is
      begin
        filosofo(1);
      end come_filosofo1;
      task body come_filosofo2 is
      begin
        filosofo(2);
      end come_filosofo2;
      task body come_filosofo3 is
      begin
        filosofo(3);
      end come_filosofo3;
      task body come_filosofo4 is
      begin
        filosofo(4);
      end come_filosofo4;
    begin
      Put("");
    end comer_filosofos;
begin
```

```
   for k in 0..4 loop
     palitos(k).init(1);
   end loop;
   comedor.init(4);
   comer_filosofos;
end filosofos;
```

Lectores y Escritores

Descripción

Existe un área compartida, que algunos procesos acceden con intención de leerla (Lectores) y otros escribirla (Escritores). Leer implica acceder al contenido, pero no implica modificar. Por lo que pueden leer muchos procesos en simultáneo. Escribir, implica modificar contenido, por lo cual nadie más puede leer y nadie más puede escribir en simultáneo.

Código

```
with Ada.Text_IO;
use Ada.Text_IO;
with Ada.Strings.Fixed;
use Ada.Strings.Fixed;
-- Main
procedure LectoresyEscritores is
 -- semáforo
  task type semaforo is
     entry init (x:Integer);
     entry p;
     entry v;
  end semaforo;
  task body semaforo is
       s: Integer;
  begin
      accept init (x:Integer) do
        s:= x;
       end init;
      loop
        select
          when s>0 => accept p do
            s := s-1;
          end p;
        or accept v do
```

```
          s := s+1;
        end v;
      end select;
    end loop;
 end semaforo;
 wrt : semaforo; -- arbitra entre escritores y entre escritores
 mutex : semaforo; -- arbitra entre lectores evitando que manipulen la variable read_cnt
  descontroladamente
 read_cnt : Integer ;
-- escritor
 procedure escritor(i: integer) is
 begin
    loop
       wrt.p;
       delay(3.0);
       Put_Line("writer " & Ada.Strings.Fixed.Trim(Integer'Image(i), Ada.Strings.Left) & " is
         writing");
       wrt.v;
       delay(3.0);
       Put_Line("writer " & Ada.Strings.Fixed.Trim(Integer'Image(i), Ada.Strings.Left) & " make
         other tasks");
    end loop;
 end escritor;
-- lector
procedure lector(i : integer) is
 begin
 loop
   mutex.p;
   read_cnt:= read_cnt + 1;
   if read_cnt = 1 Then
          wrt.p;
   end if;
   mutex.v;
   delay(3.0);
   Put_Line("reader " & Ada.Strings.Fixed.Trim(Integer'Image(i), Ada.Strings.Left) & " is
         reading");
   delay(3.0);
   mutex.p;
   read_cnt:= read_cnt - 1;
   If read_cnt = 0 Then
         wrt.v;
   end if;
   mutex.v;
```

```
  end loop;
 end lector;
 -- leen y escriben concurrentemente
  procedure leen_escriben is
      task lee_lector1;
      task lee_lector2;
      task escribe_escritor1;
      task escribe_escritor2;
      task body lee_lector1 is
      begin
        lector(1);
      end lee_lector1;
      task body lee_lector2 is
      begin
        lector(2);
      end lee_lector2;
      task body escribe_escritor1 is
      begin
        escritor(1);
      end escribe_escritor1;
      task body escribe_escritor2 is
      begin
        escritor(2);
      end escribe_escritor2;
   begin
      Put("");
   end leen_escriben;
begin
   read_cnt := 0;
   wrt.init(1);
   mutex.init(1);
   leen_escriben;
end LectoresyEscritores;
```

Productores y Consumidores

Descripción

Un buffer es un área de memoria finita provisoria para intercambiar datos.
Hay procesos que colocan datos en el buffer (Productores) y otros que extraen (Consumidores).

Definimos el buffer como una cola circular de datos (que se accede con mutua exclusión) donde las operaciones prioritarias son la inserción al final y la extracción del primero de la fila.
En definitiva el recurso es el buffer, los procesos son los productores y consumidores, y las condiciones de sincronización son el ingreso a la cola, cuando el buffer está lleno o vacío.

Código

```
with Ada.Text_IO;
use Ada.Text_IO;
with Ada.Strings.Fixed;
use Ada.Strings.Fixed;
-- Main
procedure ProductoryConsumidor is
 -- semáforo
  task type semaforo is
     entry init (x:Integer);
     entry p;
     entry v;
  end semaforo;
  task body semaforo is
       s: Integer;
  begin
      accept init (x:Integer) do
        s:= x;
       end init;
      loop
        select
          when s>0 => accept p do
            s := s-1;
          end p;
        or accept v do
            s := s+1;
          end v;
        end select;
      end loop;
   end semaforo;
   -- Cola circular
  task colaCircular is
      entry i_f (x: in Integer);
      entry e_p (y: out Integer);
  end colaCircular;
  task body colaCircular is
```

```
   buff: Array(0..49) of Integer;
   p_a_i: Integer; -- 0..49 -- primero a ingresar
   p_a_e: Integer; -- 0..49 -- primero a extraer
   cant: Integer; -- 0..50
begin
   p_a_i := 0;
   p_a_e := 0;
   cant := 0;
   loop
     select
       accept i_f(x: in Integer) do
         buff(p_a_i) := x;
         p_a_i:= (p_a_i + 1) mod 50;
         cant := cant + 1;
       end i_f;
       or accept e_p(y: out Integer) do
         y := buff(p_a_e);
         p_a_e:= (p_a_e + 1) mod 50;
         cant := cant - 1;
       end e_p;
     end select;
   end loop;
end colaCircular;
-- tipos
s:semaforo; -- acceso al buffer
e:semaforo; -- lleno
n:semaforo; -- vacío
-- productor
procedure productor(x: Integer) is
begin
  loop
    delay(3.0);
    Put_Line("the number is being produced " & Ada.Strings.Fixed.Trim(Integer'Image(x),
      Ada.Strings.Left)); -- asíncrono
    e.p;
    s.p;
    delay(3.0);
    colaCircular.i_f(x); -- síncrono
    Put_Line("the number has been produced and inserted into the buffer " &
      Ada.Strings.Fixed.Trim(Integer'Image(x), Ada.Strings.Left));
    s.v;
    n.v;
  end loop;
```

```
 end productor;
-- consumidor
procedure consumidor is
   x : Integer;
begin
  loop
     n.p;
     s.p;
     delay(3.0);
     colaCircular.e_p(x); -- síncrono
     Put_Line("the number has been extracted from the buffer " &
        Ada.Strings.Fixed.Trim(Integer'Image(x), Ada.Strings.Left));
     s.v;
     e.v;
     delay(3.0);
     Put_Line("the number is being consumed " & Ada.Strings.Fixed.Trim(Integer'Image(x),
        Ada.Strings.Left)); -- asíncrono
  end loop;
end consumidor;
-- producen y consumen concurrentemente
procedure producen_consumen is
 task produce1;
 task produce2;
 task produce3;
 task consumex;
 task consumey;
 task consumez;
 task body produce1 is
  begin
    productor(1);
  end produce1;
 task body produce2 is
  begin
    productor(2);
  end produce2;
 task body produce3 is
  begin
    productor(3);
  end produce3;
 task body consumex is
  begin
    consumidor;
  end consumex;
```

```
   task body consumey is
    begin
      consumidor;
    end consumey;
   task body consumez is
    begin
      consumidor;
    end consumez;
   begin
     Put("");
   end;
begin
  -- buffer de tipo colaCircular de máximo 50 enteros --
   s.init(1);
   e.init(50);
   n.init(0);
   producen_consumen;
end ProductoryConsumidor;
```