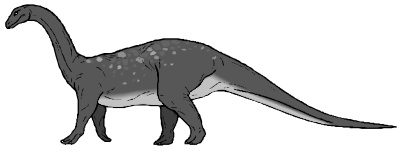
Sistemas Operativos I
Manual de prácticas

Grupo de Sistemas Operativos (DSIC/DISCA)

Práctica 5: Sincronización entre procesos



PRÁCTICA 5: SINCRONIZACIÓN ENTRE PROCESOS

INTRODUCCIÓN

En esta quinta práctica de Sistemas Operativos 1 se pretende que el alumno tenga la oportunidad de trabajar con lenguajes de programación concurrente, en este caso Ada y C con POSIX Threads. Para ello se plantea una práctica (en su mayor parte demostrativa), que combina un programa concurrente con una interfaz gráfica implementada en el lenguaje Tcl-Tk. En concreto, dicho programa es una solución al conocido problema del *Productor-Consumidor*.

No es el objetivo de la práctica que el alumno codifique completamente una solución al problema del Productor-Consumidor, pero tampoco que dicho problema se le presente totalmente resuelto. Por ello se ha optado por realizar una codificación parcial, proporcionándole la estructura y casi todo el código del programa. El alumno debe leer cuidadosamente dicho código, y, siguiendo las indicaciones que existen en el mismo, escribir las (pocas) líneas de código que faltan. Una vez codificado y compilado, se podrán probar distintos casos variando la cantidad de productores y de consumidores, así como sus velocidades relativas.

Para facilitar la comprensión de la programación concurrente, así como para que la solución al problema resulte más vistosa, se ha implementado una interfaz gráfica en Tcl-Tk, con la que las tareas del programa se comunican. De esta forma, se puede observar en pantalla el estado de cada tarea *productor* y *consumidor*, así como el estado del propio *buffer*. En el punto siguiente se introduce una breve descripción de este lenguaje, pero sólo a título de curiosidad.

¿QUÉ ES TCL-TK?

Tcl son las siglas de *Tool Command Languge*. En realidad, Tcl son dos cosas: un lenguaje para crear *scripts* y un intérprete. Como lenguaje, Tcl destaca por su sencillez y su extensa funcionalidad. Por su parte, Tk son las siglas de *ToolKit*¸ y es simplemente una extensión de Tcl que permite crear interfaces gráficas.

La definición sintáctica del lenguaje es extremadamente simple y regular, lo que facilita un rápido aprendizaje. La funcionalidad de Tcl va mucho más allá que la ofrecida por otros lenguajes. Tan simple es incrementar en uno el valor de una variable como acceder a un URL en la Web y guardar su contenido en un fichero. El intérprete es una biblioteca de funciones, no un programa. Esto permite al programador incorporar el intérprete dentro de su programa, haciendo así que toda la funcionalidad de Tcl quede disponible de inmediato a otros lenguajes tales como C o Ada, por ejemplo. No obstante, lo que más distingue a este intérprete es la facilidad de enriquecer su funcionalidad desde la aplicación donde es utilizado. La aplicación puede definir nuevos mandatos codificados en el lenguaje nativo (Ada, C, etc) y registrarlos en el intérprete. Esto ha promovido la construcción de importantes extensiones a la funcionalidad de Tcl, normalmente desarrolladas por programadores que de forma desinteresada han hecho públicas sus implementaciones. De todas estas extensiones, la más popular es Tk, una extensión que permite la construcción de interfaces gráficas de usuario, la cual se ha utilizado en esta práctica para crear la animación del problema del Productor-Consumidor.

Tcl-Tk es de dominio público. Su creador es el profesor John Ousterhout, de la Universidad de California. Actualmente, John sigue siendo liderando el desarrollo de Tcl-Tk, ahora como fundador de la empresa Scriptics. Tcl-Tk está disponible para cualquier plataforma Unix y recientemente, para plataformas Windows (3.x, 9x y NT) y Macintosh.
Para concluir, como muestra, un botón.

```
pc0601$ wish
% button .b -text hola -command {puts hola}
.b
% grid .b
%
```

DESCRIPCIÓN DEL PROBLEMA

En el problema del *Productor-Consumidor* existen dos tipos de tareas: los *productores*, que producen ítems, y los *consumidores*, que consumen dichos ítems. Entre ambos existe una zona de memoria común que almacena temporalmente los elementos producidos hasta que son extraídos para su consumo. Esta zona, denominada *buffer* se implementa normalmente mediante un vector circular, que no es más que un vector cuyos índices de incrementan de forma circular (empleando el operador módulo).

En una solución correcta a dicho problema, el acceso al buffer debe ser realizado en exclusión mutua, manteniendo además como restricciones evidentes que un productor debe suspenderse si intenta insertar un elemento cuando el buffer está lleno, y que un consumidor debe asimismo suspenderse si trata de extraer un elemento estando el buffer vacío.

LA INTERFAZ GRÁFICA

Esta interfaz muestra en pantalla el estado de las tareas así como el estado del buffer.

En la parte derecha se representan las secciones de código de las tareas productoras y consumidoras, así como la sección en la que se encuentra cada una de ellas.

- Las secciones `Produciendo`, `Consumiendo` y `Otras Acciones` son privadas para cada tarea.
- Las secciones `Insertando` y `Extrayendo` son los lugares donde las tareas acceden al buffer. Si la solución es correcta, tan sólo una tarea podrá encontrarse en estas secciones.
- La secciones `Esperando Insertar` y `Esperando Extraer` indican que la tarea está esperando que el buffer esté libre o bien que el buffer esté no vacío (consumidores) o no lleno (productores)

En la parte superior izquierda se representa el estado del buffer.

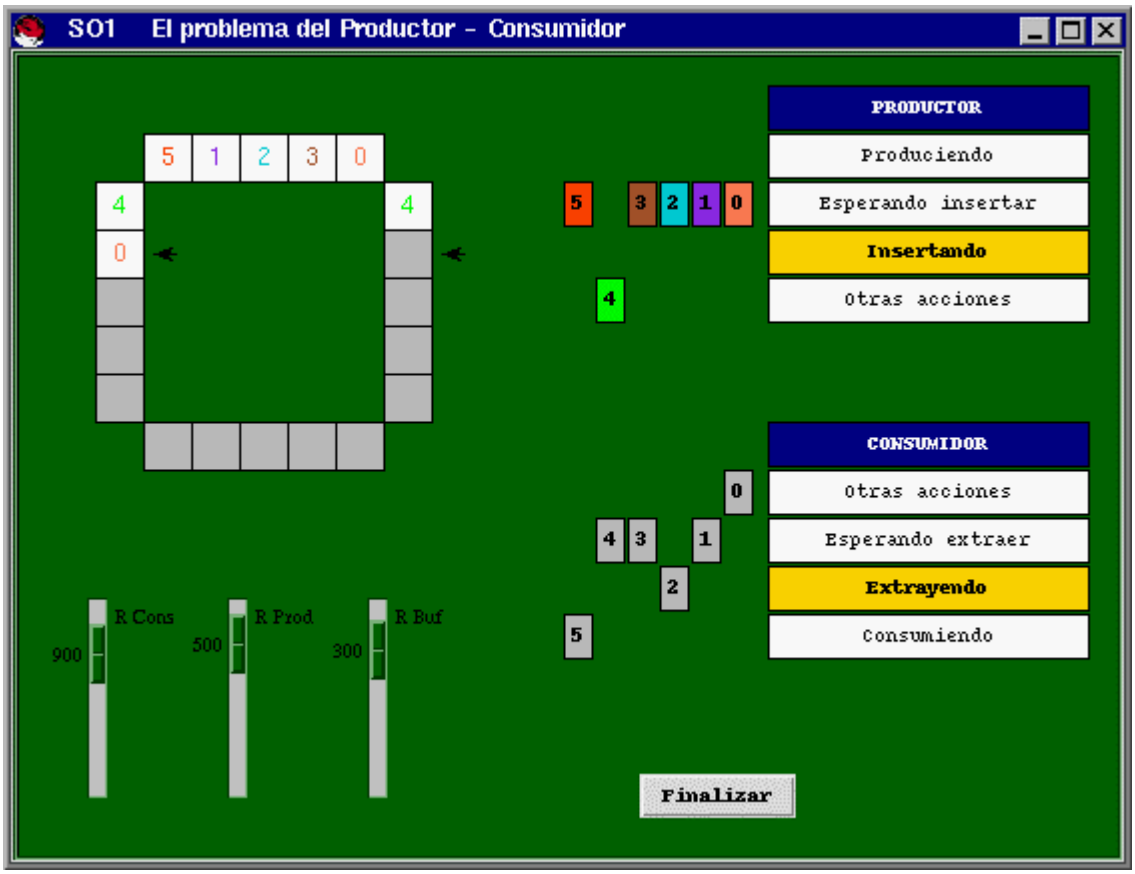
- El buffer está implementado sobre un vector de tamaño finito, tratado de forma circular, de ahí su representación
- La flecha exterior indica la posición donde se depositará el próximo elemento.

- La flecha interior indica la posición de donde se extraerá el próximo elemento.
- Los elementos en blanco indican los elementos que actualmente están dentro del buffer.
- El valor numérico de cada elemento corresponde con el número de la tarea productor que lo depositó.

En la parte inferior izquierda hay tres escalas que permiten modificar las velocidades de las tareas. Sus unidades son milisegundos.

- El valor de escala R.Prod es consultado por los productores antes de las secciones `Produciendo` y `Otras Acciones`, utilizándose este valor para retardar la ejecución de estas secciones durante los milisegundos especificados.
- El valor de escala R.Cons es consultado por los consumidores antes de las secciones `Consumiendo` y `Otras Acciones`, utilizándose este valor para retardar la ejecución de estas secciones durante los milisegundos especificados.
- El valor de la estala R.Buffer es consultado por ambos tipos de tareas antes de las secciones `Insertando` y `Extrayendo`, utilizándose este valor para retardar la ejecución de estas secciones durante los milisegundos especificados.

Es importante destacar que no hay ningún retardo asociado con las secciones `Esperando Insertar` y `Esperando Extraer`, por lo que si una tarea se detiene en estas secciones es debido exclusivamente al estado actual del buffer.



ACTIVIDADES DE LABORATORIO

En primer lugar se deben copiar los ficheros fuente de la solución:

```
$ cp –r /practicas/asignaturas/so1/pr5  .
```

Esta acción creará un directorio denominado pr5, cuyo contenido será el siguiente

```
bash$ ls -RF pr5
pr5:
lib/  prodconsada/  prodconsthreads/

pr5/lib:
animacion.adb  bibdemop.tcl   tcl_ada.adb
animacion.ads  c-animacion.c  tcl_ada.ads

pr5/prodconsada:
Makefile  demoada*  prodcons.adb

pr5/prodconsthreads:
Makefile  demothreads*  prodcons.c
```

El directorio lib constituye el soporte para la animación del problema. En los directorios prodconsada y prodconsthreads están las soluciones incompletas al problema codificadas en Ada y C respectivamente (prodcons.adb y prodcons.c), así como las soluciones completas en forma de ficheros ejecutables (demoada y demothreads).

Una vez hecho esto, podemos estructurar las actividades de laboratorio en varias fases, que deben abordarse en orden. Estas fases pueden realizarse en paralelo sobre las dos versiones del programa, o bien seleccionando primero un lenguaje de programación y a continuación el otro.

1. Ejecutar el programa de muestra (`demoada para el lenguaje Ada, demothreads` para C con POSIX threads). Una vez visto el resultado final que se espera, el alumno debe **estudiar el código del programa suministrado**, compilarlo tal como está y ejecutarlo. Se comprobará que, evidentemente, el ejecutable obtenido ahora no hace lo mismo que el programa de muestra.

   La compilación se realiza en el directorio donde está el programa con el mandato `make` o bien con la entrada `compile` del menú `tools` de `emacs`, estando editando el fichero fuente.

2. Completar la implementación del fichero fuente, el cual codifica el caso sencillo en el cual sólo hay un productor y un consumidor. Nótese que los puntos donde hay que añadir o modificar líneas de código se han indicado mediante un comentario con la cadena @@@.

3. Verifique el correcto funcionamiento de su programa, observando como influyen las distintas duraciones que pueden modificarse en la animación. En particular, debe conseguirse:

   - Una situación en que el buffer se llene rápidamente
   - Una situación en la que el buffer permanece siempre prácticamente vacío

- Una situación de equilibrio, en la que el buffer se encuentre aproximadamente con diez elementos.
- Una situación en la que el productor esté insertando y el consumidor esté extrayendo.

4. Modifique el programa para que hayan muchos más productores que consumidores e intente conseguir las situaciones descritas en el apartado 3.

5. Modifique el programa para que hayan muchos más consumidores que productores e intente conseguir las situaciones descritas en el apartado 3.

6. Modifique el programa para que hayan seis consumidores y seis productores e intente conseguir las situaciones descritas en el apartado 3.

7. Compruebe qué ocurre si se alteran las expresiones lógicas que comprueban el estado del buffer. En particular, cometa los siguientes "errores":
   - Intercambie ambas expresiones
   - Haga ambas expresiones igual a la de los consumidores
   - Haga ambas expresiones igual a la de los productores

   Es importante constatar en estos casos que el comportamiento del programa no tiene por qué ser necesariamente erróneo. En particular, dependiendo de la velocidad relativa con la que se ejecuten los consumidores y los productores, es posible que el programa funcione "aparentemente" bien. Una pista: si el buffer se encuentra en una situación de equilibrio, es decir, ni se llena ni se vacía, es posible que el error cometido en estas expresiones no conduzca a un comportamiento erróneo.

   Estas situaciones ilustran que un programa concurrente puede manifestar o no un error cometido en función no sólo de la codificación del mismo, sino del comportamiento temporal de las tareas que se ejecutan, siendo ésta una de las mayores dificultades que tiene la programación concurrente,

-- El problema del Productor-Consumidor en Ada

```
--
-- El problema del Productor-Consumidor
-- Sistemas Operativos I
--

--with Ada.Text_IO;
--use  Ada.Text_IO;

with Animacion;
use  Animacion;

procedure Prodcons is

   ----------------------------------------------------------------------
   -- CONSTANTES
   --
   N : constant  integer := 20;    -- el tamanno del buffer



   ----------------------------------------------------------------------
   -- TIPOS BASICOS
   --
   subtype Indice           is Integer range 0..N-1;
   subtype Contador         is Integer range 0..N;
   subtype Item             is Integer;
   subtype Id_Tarea         is Integer range 0..5;
   type    Vector_De_Items  is array (Indice) of Item;



   ----------------------------------------------------------------------
   -- PROCEDIMIENTOS AUXILIARES
   --
   -- sirven para simular un cierto consumo de tiempo por parte
   -- de alguna tarea en algun punto de su ejecucion
   --
   -- consultan la animacion para conocer el retardo, el cual
   -- puede actualizarse con las escalas
   --
   procedure Otras_Acciones_Consumidor is
   begin
      delay Ani.Duracion_Consumir / 1000;
   end Otras_Acciones_Consumidor;


   procedure Otras_Acciones_Productor is
   begin
      delay Ani.Duracion_Producir / 1000;
   end Otras_Acciones_Productor;



   procedure Retardo_En_El_Buffer is
   begin
      delay Ani.Duracion_Buffer / 1000;
   end Retardo_En_El_Buffer;




   ----------------------------------------------------------------------
   -- LA ESPECIFICACION DEL TIPO PROTEGIDO BUFFER
   --
   -- Este tipo representa el buffer del problema de los
   -- productores y consumidores.
   --
   --
   protected type BUFFER is
      entry Poner (id: in Id_Tarea; x : in  Item);
      entry Sacar (id: in Id_Tarea; x : out Item);
   private
      V      : Vector_De_Items;
      Entrada  : Indice  := 0;
      Salida   : Indice  := 0;
      Contador : Integer := 0;
   end BUFFER;
```

```
-------------------------------------------------------------------
-- LA IMPLEMENTACION DEL TIPO PROTEGIDO BUFFER
--
protected body BUFFER is
                                                        -- @@@
   entry Poner (id: in Id_Tarea; x : in Item) when True  is

   begin
      Ani.Productor_Comenzando_Insercion(Id);

      V(Entrada) := X;

      Ani.Almacenar(Id,X,Entrada);

      --  @@@ Mover entrada al siguiente elemento

      Ani.Entrada_Vale(Entrada);

      --  @@@ incrementar el contador en 1

      Retardo_En_El_Buffer;
      Ani.Productor_Insercion_Terminada(Id);
   end Poner;

                                                       -- @@@
   entry Sacar (id: in Id_Tarea; x : out Item) when True is
   begin
      Ani.Consumidor_Comenzando_Extraccion(Id);

      X := V(Salida);

      Ani.Quitar(Salida);

      --  @@@ Mover salida al siguiente elemento

      Ani.Salida_Vale(Salida);

      --  @@@ decrementar el contador en 1

     Retardo_En_El_Buffer;

     Ani.Consumidor_Extraccion_Terminada(Id);

   end Sacar;

end BUFFER;



-------------------------------------------------------------------
-- CREAMOS UN BUFFER
--
El_Buffer : BUFFER;


-------------------------------------------------------------------
-- LA ESPECIFICACION DEL TIPO TAREA PRODUCTOR
--
-- Id      : la identificacion de la tarea
--
task type PRODUCTOR (id: Id_Tarea);


-------------------------------------------------------------------
-- LA IMPLEMENTACION DEL TIPO TAREA PRODUCTOR
--
task body PRODUCTOR is
   x : Item;

   -- Esta funcion es la que produce un elemento.
   -- Se retarda durante los milisegundos que indica
   -- la escala correspondiente de la animacion
   -- y contesta simplemente con el identificador
   -- de la tarea
   --
   function Producir return Integer is
      D: Duration;
   begin
      D:=Ani.Duracion_Producir;
      delay D / 1000;
      return Id;
   end Producir;
```

```
-- Cuerpo de la tarea PRODUCTOR
--
begin
   Ani.Nuevo_Productor(Id);
   loop

      Ani.Productor_Produciendo(Id);

      -- @@@ Producir un elemento en x

      Ani.Productor_Esperando_Insertar(Id);

      -- @@@ Poner x en el Buffer

      Ani.Productor_Otras_Acciones(Id);

      Otras_Acciones_Productor;

   end loop;

end PRODUCTOR;



-------------------------------------------------------------------------
-- LA ESPECIFICACION DEL TIPO TAREA CONSUMIDOR
--
-- Id      : la identificacion de la tarea
--
task type CONSUMIDOR (id: Id_Tarea);


-------------------------------------------------------------------------
-- LA IMPLEMENTACION DEL TIPO TAREA CONSUMIDOR
--
task body CONSUMIDOR is
   x : Item;

   -- Este procedimiento es el que consume un elemento.
   -- Se duerme durante las milesimas de segundo
   -- que indica la animacion
   --
   procedure Consumir(X: in Integer) is
      D: Duration;
   begin
      D:=Ani.Duracion_Consumir;
      delay D / 1000;
   end Consumir;


-- Cuerpo de la tarea PRODUCTOR
--
begin
   Ani.Nuevo_Consumidor(Id);
   loop

      Ani.Consumidor_Otras_Acciones(Id);
      Otras_Acciones_Consumidor;

      Ani.Consumidor_Esperando_Extraer(Id);

      -- @@@ sacar un elemento del Buffer

      Ani.Consumidor_Consumiendo(Id);

      -- @@@ consumir dicho elemento

   end loop;

end CONSUMIDOR;
```

```
------------------------------------------------------------------------
------------------------------------------------------------------------
-- EL INICIO DEL PROGRAMA
--
begin

   -- INICIALIZAMOS LA ANIMACION
   --
   Ani.Inicializa(N);

   declare
      -- CREAMOS LAS TAREAS
      --
    Productor_0  : PRODUCTOR(0);
    Consumidor_0 : CONSUMIDOR(0);

      -- Aqui puedes crear mas tareas
      --
   begin
      null;
   end;

   -- Aqui llegamos cuando todas las tareas terminan,
   -- cosa que nunca sucede.
   --
   -- PULSA EL BOTON FINALIZAR PARA CONCLUIR LA EJECUCION DEL PROGRAMA
   -- O BIEN UTILIZA ^C PARA FINALIZAR EN LA VENTANA
   -- DESDE LA CUAL LO HAS EJECUTADO
   --

end Prodcons;
```

/* -- El problema del Productor-Consumidor en C */

```
/* -- */
/* -- El problema del Productor-Consumidor */
/* -- Sistemas Operativos I */
/* -- */


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



/* --------------------------------------------------------------------- */
/* -- CONSTANTES */
/* -- */
#define N 20



/* --------------------------------------------------------------------- */
/* -- LA IMPLEMENTACION DEL BUFFER con sincronizacion */
/* -- */
/* -- Estructuras de datos y las operaciones poner y sacar */
/* -- */



static int V[N];
static int entrada,salida,contador;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t  lleno = PTHREAD_COND_INITIALIZER;
pthread_cond_t  vacio = PTHREAD_COND_INITIALIZER;


void
Retardo_En_El_Buffer () {
  delay(Ani_Duracion_Buffer());
}


void
poner (int id, int x) {

  pthread_mutex_lock(&mutex);
  while (0) /* @@@ */
    pthread_cond_wait(&lleno,&mutex);

  Ani_Productor_Comenzando_Insercion(id);

  V[entrada] = x;

  Ani_Almacenar(id,x,entrada);

  /* @@@ Mover entrada al siguiente elemento  */

  Ani_Entrada_Vale(entrada);
  /* @@@ incrementar el contador en 1  */

  Retardo_En_El_Buffer();
  Ani_Productor_Insercion_Terminada(id);

  pthread_cond_broadcast(&vacio);
  pthread_mutex_unlock(&mutex);
}
```

```
void
sacar (int id, int *x) {

  pthread_mutex_lock(&mutex);
  while (0) /* @@@ */
    pthread_cond_wait(&vacio,&mutex);

  Ani_Consumidor_Comenzando_Extraccion(id);

  *x = V[salida];

  Ani_Quitar(salida);

  /* @@@ Mover salida al siguiente elemento  */

  Ani_Salida_Vale(salida);

  /* @@@ decrementar el contador en 1  */

  Retardo_En_El_Buffer();
  Ani_Consumidor_Extraccion_Terminada(id);

  pthread_cond_broadcast(&lleno);
  pthread_mutex_unlock(&mutex);
}



/* ------------------------------------------------------------------- */
/* -- LAS FUNCIONES QUE IMPLEMENTAN A UN PRODUCTOR */
/* -- */

int
producir (int id) {

  delay(Ani_Duracion_Producir());
  return id;
}


void
Otras_Acciones_Productor (){
  delay(Ani_Duracion_Producir());
}


void *
productor (void *id_ptr) {

  int x,id;

  id = * ((int *) id_ptr);
  for (;;) {

    Ani_Productor_Produciendo(id);


    /* @@@ Producir un elemento en x  */

    Ani_Productor_Esperando_Insertar(id);

    /* @@@ Poner x  en el Buffer  */

    Ani_Productor_Otras_Acciones(id);

    Otras_Acciones_Productor();
  }
}
```

```
/* ------------------------------------------------------------------- */
/* -- LAS FUNCIONES QUE IMPLEMENTAN A UN CONSUMIDOR */
/* -- */


void
consumir (int id, int x) {
  delay(Ani_Duracion_Consumir());
}


void
Otras_Acciones_Consumidor (){
  delay(Ani_Duracion_Consumir());
}


void *
consumidor (void *id_ptr) {

  int x,id;

  id = * ((int *) id_ptr);
  for (;;) {

    Ani_Consumidor_Otras_Acciones(id);

    Otras_Acciones_Consumidor();

    Ani_Consumidor_Esperando_Extraer(id);

    /* @@@ sacar un elemento del Buffer  */

    Ani_Consumidor_Consumiendo(id);

    /* @@@ consumir dicho elemento  */
  }
}

/* ------------------------------------------------------------------- */
/* -- EL PROGRAMA PRINCIPAL */
/* -- */



int
main (int argc, char *argv[]) {

  int prod0_id = 0;
  pthread_t prod0_thr;

  int cons0_id = 0;
  pthread_t cons0_thr;

  /* Aqui podemos declarar mas identificadores */


  Ani_Inicializa(N);


  pthread_create(&prod0_thr,NULL,productor,&prod0_id);
  Ani_Nuevo_Productor(prod0_id);

  pthread_create(&cons0_thr,NULL,consumidor,&cons0_id);
  Ani_Nuevo_Consumidor(cons0_id);


  /* Aqui podemos crear mas threads */


  Ani_Comenzamos();

  pthread_join(prod0_thr,NULL);
  pthread_join(cons0_thr,NULL);

  /* Aqui podemos esperar a mas threads */


}
```