EXAMEN PRÁCTICA 1 - SCD - (22/10/14) - CARLOS UREÑA
SINCRONIZACIÓN. HEBRAS Y SEMÁFOROS POSIX (EXAMEN TIPO A)

El tiempo de realización del examen es de 60 minutos. La primera parte se responderá contestando en el propio folio del examen. La segunda parte se realiza partiendo de la solución proporcionada por el alumno a la práctica. Se debe subir el código generado a Tutor antes del plazo que el profesor indique.

1. Corrección de errores (6 puntos)

Accede a Tutor y descarga el archivo que el profesor te ha proporcionado (A1-ebs.java). El código que contiene trata de dar solución a un problema clásico de sincronización productor-consumidor en utilizando para ello semáforos POSIX.

1a.1. Intenta compilar el código proporcionado. Comprueba que hay diversos errores básicos de sintaxis que impiden que el código genere un ejecutable. Encuentra dichos errores y arréglalos hasta que el código compile correctamente. Indica en el folio del examen el número de línea y cómo debería quedar ésta una vez solventado el problema.

1a.2. Ejecuta el programa y observa la salida que genera. El algoritmo presenta un error en su diseño. Explica brevemente en el folio del examen por qué la salida proporcionada no es correcta.

1a.3. Observa el código del programa ejecutado. Encuentra y corrige el error en el diseño del algoritmo que impide que el programa muestre una salida coherente. Señala en el folio del examen en qué línea está el problema y cómo debería quedar ésta para solucionar el error.

1a.4. Se desea depurar el algoritmo proporcionado para que la escritura y la lectura de un dato en el buffer (no la generación del dato, sino la escritura/lectura en el buffer de éste) sea una sentencia atómica, es decir, no pueda haber ninguna otra hebra accediendo al buffer mientras que ésta se ejecuta. Describe brevemente en el folio qué cambios harías en el código para llevarlo a cabo.

2. Problema de de los Fumadores (4 puntos)

Partiendo del algoritmo desarrollado durante la Práctica 1 para resolver el problema de los Fumadores usando semáforos, se pide modificar el algoritmo para satisfacer los siguientes requisitos.

Deseamos añadir una nueva hebra a nuestro problema llamada *proveedor*, que genera un ingrediente en bucle. Una vez ha generado uno, el proveedor despierta al estanquero y le pasa el ingrediente, el cual lo colocará en el mostrador y despertará al fumador correspondiente.

La hebra proveedora no generará nuevos ingredientes hasta que se haya retirado del mostrador el ingrediente ya producido. Las hebras de los fumadores no cambian con respecto al problema original. Recuerda controlar la sincronización entre todas las hebras mediante el uso de semáforos para evitar incoherencias en la salida o interbloqueos (*deadlocks*).

```
// Prácticas de SCD - Curso 2014-15
// Práctica 1 - prueba de evaluación
// Grupo A3 (Viernes 11:30-13:30)
// Prueba tipo A
// Ejercicio 1 - con errores básicos y de sincronización

#include <iostream>  // incluye 'std::cout'
#include <stdlib.h>  // incluye 'rand' y 'srand'
#include <unistd.h>  // incluye 'usleep( <millonésimas-de-segundo> )'
#include <time.h>    // incluye 'time'
#include <pthread.h>
#include <semaphore.h>

using namespace std ;

const unsigned
   num_items = 30 ,
   tam_vec   = 5  ;
int
   vector[tam_vec] ;
unsigned
   primera_libre   = 0,
   primera_ocupada = 0;
pthread_t
   hebra_productor,
   hebra_consumidor ;
sem_t
       puede_escribir,
       puede_leer,
       mutex ;

//----------------------------------------------------------------------

int producir_dato()
{
   static int contador = 1 ;
   usleep( 1000UL*( 20UL+(rand()%80UL) ) ) ;
   return contador ++ ;
}

//----------------------------------------------------------------------

void consumir_dato( int dato )
{
   sem_wait( &mutex );
   cout << "consumiendo dato: " << dato << endl ;
   sem_post( &mutex ) ;
   usleep( 1000UL*( 20UL+ (rand()%80UL) ) );
}

//----------------------------------------------------------------------

int * funcion_productor( void * )
{
   int dato ;

   for( unsigned i = 0 ; i < num_items ; i++ )
   {
      dato = producir_dato() ;
      sem_wait( &puede_escribir ) ;
      vector[primera_libre] = dato ;
      primera_libre = (primera_libre + 1) % tam_vec ;
      sem_post( &puede_leer ) ;
   }
   return NULL ;
}

//----------------------------------------------------------------------
```

```
void * funcion_consumidor( void * )
{
   int dato ;

   for( unsigned i = 0 ; i < num_items ; i++ )
   {
      sem_wait( &puede_leer );
      dato = vector[primera_ocupada] ;
      primera_ocupada = (primera_ocupada + 1) % tam_vec ;
      sem_post( & puede_escribir ) ;
      consumir_dato( dato ) ;
   }
   return NULL ;
}

//----------------------------------------------------------------------

int main()
{
   srand( time(NULL) );

   sem_init( &puede_escribir, 0, tam_vec);
   sem_init( &puede_leer, 0, tam_vec);
   sem_init( &mutex, 0, 1 ) ;

   pthread_create( hebra_productor,  NULL, funcion_productor,  NULL );
   pthread_create( &hebra_consumidor, NULL, funcion_consumidor, NULL );

   pthread_join( hebra_productor,  NULL ) ;
   pthread_join( hebra_consumidor, NULL ) ;

   sem_destroy( &puede_escribir );
   sem_destroy( &puede_leer );
}
```

SOLUCIÓN (ERRORES COMPILACIÓN)

1. Línea 53. La función función_productor está indicada en la cabecera como una función que devuelve un (int *), cuando debería ser un (void *).

   *void * funcion_productor(void *)*

2. Línea 95. La inicialización de la hebra hebraproductor está recibiendo el argumento sin ser referencia, falta el &.

   pthread_create(&hebra_productor, NULL, funcion_productor, NULL);

SOLUCIÓN (ERROR DISEÑO)

Línea 92. El semáforo puede_leer está inicializado al tamaño del vector, el algoritmo así no funciona. El semáforo puede_escribir sí debe estar inicializado a tam_vec, pero el puede_leer debería estar inicializado a 0. Lo correcto sería...

sem_init(&puede_escribir, 0, 0);