Área de Arquitectura y Tecnología de Computadores

Universidad Carlos III de Madrid

Universidad
Carlos III de Madrid

SISTEMAS OPERATIVOS

Práctica 3. Programación de una biblioteca de gestión de stock concurrente (Multithread)

Grado de Ingeniería en Informática

Curso 2013/2014

Índice

1. Enunciado de la Práctica

Esta práctica permite al alumno familiarizarse con los servicios para la gestión de procesos ligeros (hilos o *threads*) que proporciona POSIX. Se pretende que el alumno comprenda la importancia de sincronizar los hilos cuando trabajan de forma concurrente.

Para la gestión de hilos, se utilizarán las llamadas al sistema de POSIX relacionadas, como `pthread_create` y `pthread_join`. Para la sincronización de hilos, se utilizarán *mutex* y *variables condicionales* mediante llamadas como `pthread_mutex_init`, `pthread_mutex_lock`, `pthread_mutex_unlock`, `pthread_cond_init`, `pthread_cond_wait`, `pthread_cond_signal`, `pthread_cond_destroy`.

El alumno deber diseñar y codificar, en lenguaje C y sobre sistema operativo UNIX/Linux, una librería de acceso a la base de datos de un almacén de modo que las operaciones sean consistentes incluso en caso de que sucedan de forma concurrente.

1.1. Descripción de la Práctica

Se pretende codificar una biblioteca que permita la concurrencia de la biblioteca **db_warehouse**. La biblioteca **db_warehouse** proporcionada permite gestionar los productos de un almacén así como el stock actual de cada uno de ellos, pero no soporta que varios clientes puedan acceder a la vez.

Se desea codificar una nueva interfaz que se sitúe entre el usuario y la base de datos, que dé soporte a esta concurrencia. La biblioteca a codificar, denominada **concurrent**, tiene que cumplir los siguientes requisitos:

- Debe permitir realizar operaciones concurrentes sobre la base de datos sobre distintos elementos de la misma.
- Debe mantener la coherencia de los elementos de la base de datos (tanto en la creación, como en la eliminación o la actualización de los datos).
- Se debe respetar la interfaz propuesta en el enunciado, sin posibilidad de realizar ningún cambio en la misma.

Es decir:

- La biblioteca a desarrollar debe soportar el uso de múltiples procesos ligeros “por encima” de la interfaz.
- No se pueden realizar operaciones de creación o borrado de un producto si se está realizando cualquier otra operación sobre la base de datos.
- En cambio, se pueden realizar operaciones de actualización de distintos productos de forma concurrente.

- Existirá un máximo de operaciones de consultas concurrente sobre el total de la base de datos que vendrá marcado por la constante MAX_READERS que se encuentra en el fichero `concurrent.h`.

1.1.1. Interfaz de la base de datos

La interfaz proporcionada para acceder a la base de datos se detalla a continuación. Esta interfaz ya está codificada y se proporciona junto al código fuente de la práctica pre-compilada, es decir, se podrá acceder al fichero de cabeceras, pero no al código.

NOTA: Todos los parámetros pasados a las funciones tienen que tener algún valor. En el caso de que pase algún valor a **NULL**, la función devolverá siempre un valor de **-1**.

La base de datos podrá administrar hasta 16 productos, es decir, se podrán crear o borrar productos tantas veces como se desee, pero sólo podrá haber un **máximo** de **16 productos activos**.

- `int db_warehouse_init()`
  - **Descripción:** función que inicializa la base de datos. Usar solo una vez al inicio.
  - **Valor de retorno:** 0 → ok, -1 → error
- `int db_warehouse_destroy()`
  - **Descripción:** función que elimina los recursos utilizados de la base de datos. Usar solo una vez al finalizar.
  - **Ámbito:** afecta a la totalidad de la base de datos.
  - **Valor de retorno:** 0 → ok, -1 → error
- `int db_warehouse_create_product (char *product_name)`
  - **Descripción:** función que crea un nuevo producto en el primer registro disponible del almacén. No controla la existencia de otros productos con el mismo nombre.
  - **Ámbito:** afecta a la totalidad de la base de datos.
  - **Entrada:** nombre del producto.
  - **Valor de retorno:** 0 → ok, -1 → error que significa que no existen registros libres para la creación del producto (16 productos activas).
- `int db_warehouse_get_num_products(int *num_products)`

  - **Descripción:** función que devuelve el número de productos activos en el almacén.
  - **Salida:** número de productos activos en el almacén.
  - **Valor de retorno:** 0 → ok, -1 → error

- `int db_warehouse_delete_product(char *product_name)`
  - **Descripción:** función que elimina el primer producto que encuentre con el mismo nombre que el solicitado.
  - **Ámbito:** afecta a la totalidad de la base de datos.
  - **Entrada:** nombre del producto.
  - **Valor de retorno:** 0 → ok, -1 → error en caso de que no se encuentre ningún producto con el nombre solicitado.
- `int db_warehouse_exists_product(char *product_name)`
  - **Descripción:** función que indica si existe un producto.
  - **Entrada:** nombre del producto a buscar.
  - **Valor de retorno:** 1 → existe, 0 → No existe
- `int db_warehouse_update_stock(char *product_name, int stock)`
  - **Descripción:** función que actualiza el stock de un producto.
  - **Ámbito:** afecta al producto solicitado.
  - **Entrada:** nombre del producto y stock del producto.
  - **Valor de retorno:** 0 → ok, -1 → error en caso de que no se encuentre ningún producto con el nombre solicitado
- `int db_warehouse_get_stock(char *product_name, int *stock)`
  - **Descripción:** función que devuelve el stock de un producto.
  - **Ámbito:** afecta al producto solicitado.
  - **Entrada:** nombre del producto.
  - **Salida:** stock del producto.
  - **Valor de retorno:** 0 → ok, -1 → error en caso de que no se encuentre ningún producto con el nombre solicitado.

- `int db_warehouse_set_internal_data(char *product_name, void *ptr, int size)`
    - **Descripción:** función que permite asociar una serie de datos a un producto concreto. Se puede utilizar para guardar con el producto datos relacionados con la sincronización de procesos ligeros.
    - **Ámbito:** afecta al producto solicitado.
    - **Entrada:** nombre del producto, puntero a los datos y tamaño de los mismos.
    - **Valor de retorno:** 0 → ok, -1 → error en caso de que no se encuentre ningún producto con el nombre solicitado.
- `int db_warehouse_get_internal_data(char *product_name, void **ptr, int *size)`
    - **Descripción:** función que devuelve los datos internos asociados a un producto.
    - **Ámbito:** afecta l producto solicitado.
    - **Entrada:** nombre del producto.
    - **Salida:** puntero a los datos y tamaño de los mismos.
    - **Valor de retorno:** 0 → ok, -1 → error en caso de que no se encuentre ningún producto con el nombre solicitado.

1.1.2. Interfaz secuencial

En este apartado se detalla la interfaz secuencial. Esta interfaz ya está codificada y se proporciona junto al código fuente de la práctica. El objetivo de esta interfaz es mostrar un ejemplo de codificación secuencial de la funcionalidad requerida, es decir, incluye todo el código que se debe implementar salvo los mecanismos de sincronización de hilos. Se debe respetar este código a la hora de implementar la versión concurrente.

- `int sequential_init()`
    - **Descripción:** función que inicializa los recursos utilizados en la biblioteca, así como la base de datos utilizada. Usar sólo una vez al inicio.
    - **Valor de retorno:** 0 → ok, -1 → error
- `int sequential_destroy()`
    - **Descripción:** función que elimina los recursos utilizados en la biblioteca, así como los utilizados en la base de datos. Usar sólo una vez al final del programa.
    - **Valor de retorno:** 0→ ok, -1→ error

- `int sequential_create_product(char *product_name)`
  - **Descripción:** función que crea un producto en la base de datos. En caso de encontrar otro producto con el mismo nombre, se considera que la creación se ha realizado con éxito.
  - **Entrada:** nombre del producto.
  - **Valor de retorno:** 0 → ok, -1→ error
- `int sequential_get_num_products(int *num_products)`
  - **Descripción:** función que consulta en la base de datos el número de productos activos en el almacén.
  - **Salida:** devuelve el número de productos activos en el almacén.
  - **Valor de retorno:** 0→ ok, -1 → error
- `int sequential_delete_product(char *product_name)`
  - **Descripción:** función que borra un producto de la base de datos. En caso de no encontrarse un producto con el mismo nombre, se considera que el borrado se ha realizado con éxito.
  - **Entrada:** nombre del producto.
  - **Valor de retorno:** 0 → ok, -1→ error
- `int sequential_increment_stock(char *product_name, int stock, int *updated_stock)`
  - **Descripción:** función que incrementa el stock de un producto y devuelve el stock actualizado.
  - **Entrada:** nombre del producto y el stock a incrementar.
  - **Salida:** devuelve el stock actualizado.
  - **Valor de retorno:** 0 → ok, -1 → error (producto no encontrado, etc.)
- `int sequential_decrement_stock(char *product_name, int stock, int *updated_stock)`
  - **Descripción:** función que decrementa el stock de un producto y devuelve el stock actualizado.
  - **Entrada:** nombre del producto y el stock a restar.
  - **Salida:** devuelve el stock actualizado.
  - **Valor de retorno:** 0→ ok, -1 → error (producto no encontrado, etc.)

- `int sequential_get_stock(char *product_name, int *stock)`
  - **Descripción:** función que devuelve el stock de un producto.
  - **Entrada:** nombre del producto.
  - **Salida:** devuelve el stock actualizado.
  - **Valor de retorno:** 0→ ok, -1 → error (producto no encontrado, etc.)

1.1.3. Interfaz concurrente

En este apartado se detalla la interfaz a codificar. Se debe respetar, codificando el contenido de sus funciones para permitir el acceso concurrente a la base de datos del almacén.

- `int concurrent_init()`
  - **Descripción:** función que inicializa los recursos utilizados en la biblioteca, así como la base de datos utilizada y los mecanismos de control de concurrencia. Usar solo una vez al inicio. No debe controlar ningún tipo de concurrencia.
  - **Control de concurrencia:** no debe controlar ningún tipo de concurrencia.
  - **Valor de retorno:** 0 → ok, -1 → error
- `int concurrent_destroy()`
  - **Descripción:** función que elimina los recursos utilizados en la biblioteca, así como los utilizados en la base de datos y los mecanismos de control de concurrencia. Usar solo una vez al final del programa.
  - **Control de concurrencia:** no debe controlar ningún tipo de concurrencia.
  - **Valor de retorno:** 0 → ok, -1 → error
- `int concurrent_create_product(char *product_name)`
  - **Descripción:** función que crea un producto en la base de datos utilizando concurrencia. En caso de encontrar otro producto con el mismo nombre, se considera que la creación se ha realizado con éxito. Debe mantener la coherencia de la BD. No permite realizar otras operaciones al mismo tiempo.
  - **Control de concurrencia:** no debe permitir que se realice ninguna otra operación sobre la base de datos al mismo tiempo.
  - **Entrada:** nombre del producto.
  - **Valor de retorno:** 0 → ok, -1 → error
- `int concurrent_get_num_products(int *num_products)`
  - **Descripción:** función que consulta en la base de datos el número de productos activos utilizando gestión de la concurrencia. Debe mantener la coherencia de la BD.
  - **Control de concurrencia:** solo debe permitir que se produzcan otras operaciones de consulta sobre la base de datos (hasta un máximo de MAX_READERS). Las operaciones permitidas serán obtener número de productos y cualquier operación sobre un producto concreto.
  - **Salida:** devuelve el número de productos existentes en el almacén.
  - **Valor de retorno:** 0 → ok, -1 → error

- `int concurrent_delete_product(char *product_name)`
    - **Descripción:** función que borra un producto de la base de datos utilizando control de concurrencia. En caso de no encontrarse un producto con el mismo nombre, se considera que el borrado se ha realizado con éxito. Debe mantener la coherencia de la BD. No permite realizar otras operaciones al mismo tiempo.
    - **Control de concurrencia:** no debe permitir que se realice ninguna otra operación sobre la base de datos al mismo tiempo.
    - **Entrada:** nombre del producto.
    - **Valor de retorno:** 0 → ok, -1 → error
- `int concurrent_increment_stock(char *product_name, int stock, int *updated_stock)`
    - **Descripción:** función que incrementa el stock de un producto y devuelve el stock actualizado. Debe permitir actualizar o leer otros productos en paralelo.
    - **Control de concurrencia:** no debe permitir que se realice ninguna otra operación sobre el producto afectado al mismo tiempo. Contará como una operación de consulta sobre la base de datos (hasta un máximo de MAX_READERS).
    - **Entrada:** nombre del producto y el stock a incrementar.
    - **Salida:** devuelve el stock actualizado.
    - **Valor de retorno:** 0 → ok, -1 → error (producto no encontrado, etc.)
- `int concurrent_decrement_stock(char *product_name, int stock, int *updated_stock)`
    - **Descripción:** función que decrementa el stock de un producto y devuelve el stock actualizado. Debe permitir actualizar o leer otros productos en paralelo.
    - **Control de concurrencia:** no debe permitir que se realice ninguna otra operación sobre el producto afectado al mismo tiempo. Contará como una operación de consulta sobre la base de datos (hasta un máximo de MAX_READERS).
    - **Entrada:** nombre del producto y el stock a restar.
    - **Salida:** devuelve el stock actualizado.
    - **Valor de retorno:** 0 → ok, -1 → error (producto no encontrado, etc.)
- `int concurrent_get_stock(char *product_name, int *stock)`
    - **Descripción:** función que devuelve el stock de un producto. Debe permitir actualizar o leer otros productos en paralelo.

- **Control de concurrencia:** debe permitir que se realicen otras consultas simultáneas sobre el producto, pero ninguna otra operación de actualización sobre el producto afectado al mismo tiempo. Contará como una operación de consulta sobre la base de datos (hasta un máximo de MAX_READERS).
- **Entrada:** nombre del producto.
- **Salida:** devuelve el stock actualizado.
- **Valor de retorno:** 0 → ok, -1 → error (producto no encontrado, etc.)

1.2. Código Fuente de Apoyo

Para facilitar la realización de esta práctica se dispone del fichero *ssoo_p3_concurrency_2014.tar.gz* que contiene código fuente de apoyo. Para extraer su contenido ejecutar lo siguiente:

```
tar zxvf ssoo_p3_concurrency_2014.tar.gz
```

Al extraer su contenido, se crea el directorio *ssoo_p3_concurrency/*, donde se debe desarrollar la práctica. Dentro de este directorio se habrán incluido los siguientes ficheros:

ssoo_p3_concurrency/Makefile

Fichero de compilación (para ejecutar "`make`", para borrar los ficheros "`make clean`").

ssoo_p3_concurrency/lib/

Directorio con las bibliotecas del sistema

ssoo_p3_concurrency/lib/libdb_warehouse.a

Biblioteca para el manejo de la base de datos compilada en las aulas de la asignatura.

ssoo_p3_concurrency/lib/libdb_warehouse-64bits.a

Biblioteca para el manejo de la base de datos compilada en una máquina de 64 bits. Si se quiere utilizar, renombrarla a **libdb_warehouse.a** sustituyendo a la anterior.

ssoo_p3_ concurrency/include/

Directorio con los ficheros de cabeceras.

ssoo_p3_concurrency/include/db_warehouse.h

Fichero de cabeceras con las funciones de manejo de la base de datos del almacén.

ssoo_p3_concurrency/include/sequential.h

Fichero de cabeceras con las funciones de manejo secuencial de ejemplo.

ssoo_p3_concurrency/include/concurrent.h

Fichero de cabeceras con las funciones de manejo de la biblioteca concurrente.

ssoo_p3_concurrency/sequential.c

Fichero que contiene las funciones para la utilización de la librería del almacén de forma secuencial. Se trata de un código cuya funcionalidad es exactamente la misma que la requerida salvo por los mecanismos de sincronización ausentes.

ssoo_p3_concurrency/concurrent.c

Fichero que contiene las funciones para la utilización concurrente de la librería del almacén. Este es el **ÚNICO FICHERO QUE SE DEBE MODIFICAR**. SE DEBE RESPETAR Y UTILIZAR EL CÓDIGO YA EXISTENTE a la hora de codificar la librería concurrente.

ssoo_p3_concurrency/sequential_example.c

Fichero de ejemplo que utiliza las funciones descritas en la biblioteca secuencial. Es sólo un ejemplo de funcionamiento, las pruebas que se realicen al código entregado serán más complejas.

ssoo_p3_concurrency/concurrent_example.c

Fichero de ejemplo que utiliza las funciones descritas en la biblioteca concurrente. Es sólo un ejemplo de funcionamiento, las pruebas que se realicen al código entregado serán más complejas.

2. Entrega

2.1. Puntuación de la práctica

A continuación se indica un desglose de cómo será puntuada la práctica:

40% - Operaciones a nivel global de almacén: crear, borrar y obtener número de productos. (Control de concurrencia de actualización y consulta a nivel global).

40% - Operaciones a nivel de producto: Incrementar, decrementar y consultar el stock de los productos. (Control de concurrencia de actualización y consulta a nivel global y de producto).

20% - Memoria.

2.2. Pruebas a realizar

El código entregado debe ser capaz de superar las siguientes pruebas, como mínimo:

- Se deben poder lanzar operaciones de actualización de la base de datos del almacén de forma concurrente (crear y borrar productos) manteniendo las condiciones de concurrencia, es decir, que no se realicen dos escrituras a la vez.

- Se deben poder lanzar operaciones de consulta de la base de datos del almacén de forma concurrente (obtener número de productos) manteniendo las condiciones de concurrencia, en este caso será posible que las lecturas se realicen de forma simultánea.

- En caso de que haya operaciones de lectura y escritura lanzadas de forma simultánea, las operaciones de lectura deberán esperar por las operaciones de escritura que ya se encuentren en curso y viceversa.

- En caso de que sean operaciones sobre productos, bien sean actualizaciones (incremento y decremento de stock) o consultas (obtener stock) debe respetar el mismo orden indicado anteriormente, en este caso, tan solo afectando al producto sobre la que se realicen las operaciones y no sobre el almacén completo, es decir, podrán ser modificados varios productos de forma simultánea, pero no un mismo producto por varios usuarios.

- Se debe tener siempre en cuenta que el número de lectores concurrentes sobre la base de datos será siempre de MAX_READERS como máximo. Es decir, se podrán hacer hasta MAX_READERS consultas simultáneas a la base de datos, considerando una consulta de la base de datos como: obtener el número total de

productos o cualquier operación (consulta o actualización de stock) sobre un producto concreto.

2.3. Plazo de entrega

La fecha límite de entrega de la práctica se podrá consultar en AULA GLOBAL.

La entrega de las prácticas ha de realizarse de forma electrónica. En AULA GLOBAL se habilitarán unos enlaces a través de los cuales podrá realizar la entrega de las prácticas. En concreto, **se habilitará un entregador para el código de la práctica, y otro de tipo TURNITIN** para la memoria de la práctica.

2.4. Ficheros a entregar

Se debe entregar un archivo comprimido en formato zip con el nombre ssoo_p3_AAAAAAAAA_BBBBBBBBB.zip donde A…A y B…B son los NIAs de los integrantes del grupo. En caso de realizar la práctica en solitario, el formato será ssoo_p3_AAAAAAAAA.zip. El archivo zip se entregará en el entregador correspondiente al código de la práctica. El archivo debe contener:

- **concurrent.c**
- **autores.txt**

El fichero autores.txt deberá contener el nombre y NIA de cada autor en líneas separadas, ejemplo:

AUTOR1 NIA1
AUTOR2 NIA2

Para comprimir mediante línea de comandos, se puede utilizar el comando:

```
zip ssoo_p2_AAAAAAAAA_BBBBBBBBB concurrent.c autores.txt
```

La memoria se entregará en formato PDF en un fichero llamado **ssoo_p3_AAAAAAAAA_BBBBBBBBB.pdf**. Solo se corregirán y calificarán memorias en formato pdf. Tendrá que contener al menos los siguientes apartados:

- **Descripción del código** detallando las principales funciones implementadas. NO incluir código fuente de la práctica en este apartado. Cualquier código será automáticamente ignorado.

- **Batería de pruebas** utilizadas y resultados obtenidos. Se dará mayor puntuación a pruebas avanzadas, casos extremos, y en general a aquellas pruebas que garanticen el correcto funcionamiento de la práctica en todos los casos. Hay que tener en cuenta:
    - Que un programa compile correctamente y sin advertencias (*warnings*) no es garantía de que funcione correctamente.
    - Evite pruebas duplicadas que evalúan los mismos flujos de programa. La puntuación de este apartado no se mide en función del número de pruebas, sino del grado de cobertura de las mismas. Es mejor pocas pruebas que evalúan diferentes casos a muchas pruebas que evalúan siempre el mismo caso.
- **Conclusiones**, problemas encontrados, cómo se han solucionado, y opiniones personales.

Se puntuará también los siguientes aspectos relativos a la **presentación** de la práctica:

- Debe contener portada, con los autores de la práctica y sus NIAs.
- Debe contener índice de contenidos.
- La memoria debe tener números de página en todas las páginas (menos la portada).
- El texto de la memoria debe estar justificado.

El archivo pdf se entregará en el entregador correspondiente la memoria de la práctica (entregador TURNITIN).

NOTA: La única versión registrada de su práctica es la última entregada. La valoración de esta es la única válida y definitiva.

3. Normas

1) **No se permitirá la utilización de llamadas como sleep, msleep o funciones cuyo objetivo sea similar a éstas.**

2) **Las prácticas que no compilen o que no se ajusten a la funcionalidad y requisitos planteados, obtendrán una calificación de 0.**

3) **El entorno de pruebas de cara a la evaluación de la práctica será guernika. Asegúrese de que si práctica funciona correctamente en guernika mediante ssh o bien en los laboratorios del departamento de informática.**

4) **Se prestará especial atención a detectar funcionalidades copiadas entre dos prácticas. En caso de encontrar implementaciones comunes en dos prácticas, ambas obtendrán una calificación de 0.**

5) **Los programas deben compilar sin *warnings*. De lo contrario, se penalizará la nota de la práctica.**

6) **Un programa no comentado, obtendrá una calificación de 0.**

7) **La entrega de la práctica se realizará a través de Aula Gloval, tal y como se detalla en el apartado Entrega de este documento. No se permite la entrega a través de correo electrónico sin autorización previa.**

8) **Se debe respetar en todo momento el formato de la entrada y salida que se indica en cada programa a implementar.**

9) **Se debe realizar un control de errores en cada uno de los programas.**

   - **Una práctica que no supere el corrector de formato recibirá una penalización en la nota.**

Los programas entregados que no sigan estas normas no se considerarán aprobados.

4. Anexo (man function).

man es el paginador del manual del sistema, es decir permite buscar información sobre un programa, una utilidad o una función. Véase el siguiente ejemplo:

'man pthread_create' o ***'man 3 pthread_create'***

Las páginas usadas como argumentos al ejecutar *man* suelen ser normalmente nombres de programas, utilidades o funciones. Normalmente, la búsqueda se lleva a cabo en todas las secciones de manual disponibles según un orden predeterminado, y sólo se presenta la primera página encontrada, incluso si esa página se encuentra en varias secciones.

Para salir de la página mostrada, basta con pulsar la tecla 'q'.

Una página de manual tiene varias partes. Éstas están etiquetadas como NOMBRE, SINOPSIS, DESCRIPCIÓN, OPCIONES, FICHEROS, VÉASE TAMBIÉN, BUGS, y AUTOR. En la etiqueta de SINOPSIS se recogen las librerías (identificadas por la directiva *#include*) que se deben incluir en el programa en C del usuario para poder hacer uso de las funciones correspondientes.

Las formas más comunes de usar *man* son las siguientes:

- **man sección elemento:** Presenta la página de elemento disponible en la sección del manual.
- **man -a elemento:** Presenta, secuencialmente, todas las páginas del elemento disponibles en el manual. Entre página y página se puede decidir saltar a la siguiente o salir del paginador completamente.
- **man -k palabra-clave:** Busca la palabra-clave entre las descripciones breves y las páginas de manual y presenta todas las que casen.

5. Bibliografía

- El lenguaje de programación C: diseño e implementación de programas Félix García, Jesús Carretero, Javier Fernández y Alejandro Calderón. Prentice-Hall, 2002.
- The UNIX System S.R. Bourne Addison-Wesley, 1983.
- Advanced UNIX Programming M.J. Rochkind Prentice-Hall, 1985.
- Sistemas Operativos: Una visión aplicada Jesús Carretero, Félix García, Pedro de Miguel y Fernando Pérez. McGraw-Hill, 2001.
- Programming Utilities and Libraries SUN Microsystems, 1990.
- Unix man pages (`man function`)