Fundamentos de los Sistemas Operativos

Práctica 4: Hilos

El objetivo de esta actividad es iniciarte en la programación concurrente, a través de la creación de hilos que tendrán que coordinarse entre sí para poder resolver una misión común.

Después de haber trabajado en la práctica 3 con ficheros, ahora tendrás oportunidad de manejar procesos ligeros o hilos. Para ello utilizarás la biblioteca ***pthreads***, que es un estándar dentro de UNIX/Linux.

Recuerda que en el Campus Virtual tienes el material de apoyo para realizar esta práctica.

1 Introducción

El objetivo de esta actividad es iniciarte en la programación concurrente, a través de la creación de hilos que tendrán que coordinarse entre sí para poder resolver una misión común.

La biblioteca *pthreads* se creó a principios de los años 90 para que los desarrolladores de aplicaciones en el sistema operativo UNIX pudieran crear procesos ligeros dentro de sus aplicaciones. Con el tiempo, *pthreads* se ha convertido en un estándar POSIX y por tanto cualquier sistema operativo estándar UNIX debería ofrecer esta API. *Pthreads* ofrece servicios para crear hilos y además implementa un modelo de sincronización entre hilos basado en cerrojos y variables condición.

Los objetivos de aprendizaje de esta práctica son:

- Saber crear hilos y terminarlos (*pthread_create*, *pthread_exit, pthread_cancel*).
- Saber sincronizar hilos con la función *pthread_join*.
- Adiestrarte en la resolución de ejercicios de sincronización entre hilos mediante cerrojos y variables condición.

Esta práctica tiene asociada una entrega y por tanto forma parte de la calificación final.

2 Requisitos previos

Para abordar esta práctica es muy conveniente haber completado la Práctica 3. También debes saber cómo funcionan los cerrojos y las variables condición como herramientas de sincronización entre procesos o hilos (Tema 3 de la teoría).

3 Plan de actividades y orientaciones

Para realizar esta práctica deberás conocer los servicios básicos de manejo de hilos de la biblioteca ***pthreads***. El material que puedes emplear para documentarte es el siguiente:

- [1] https://computing.llnl.gov/tutorials/pthreads/.
  Es un tutorial sobre *pthreads* en inglés. Aunque puedes analizar y estudiar el tutorial al completo, para poder desarrollar la práctica que te proponemos simplemente necesitarás trabajar los apartados 1 al 6 del tutorial.
- [2] Linux System Programming, 2nd Edition. Disponible en la biblioteca de la ULPGC. Puedes encontrar información relacionada con la biblioteca *pthreads* en la sección final del capítulo 7, concretamente en el apartado titulado *Pthreads*. Para poder afrontar la actividad práctica te bastará con trabajar los apartados explicados desde la página 226 hasta el ejemplo que se muestra en la página 235.

Recuerda igualmente que siempre puedes encontrar ayuda específica sobre una función si consultas la página de manual desde el shell, por ejemplo: **man pthread_create**.

Además, en el material de esta actividad en *Moodle* se añaden unos ficheros fuentes en C necesarios para realizar las tareas. Descárgalos para que puedas realizar las tareas.

Actividades	*Objetivos / orientaciones*
Leer documentación sobre la biblioteca *pthreads*	Conocer el funcionamiento básico de la librería. Antes de empezar a resolver los problemas propuestos deberías leer al menos una de las dos fuentes recomendadas [1] [2].
Sesiones prácticas	Habrá tres sesiones prácticas en el laboratorio. El profesor realizará ejemplos de uso de la librería.
Ejercicios de entrenamiento	Es muy importante que realices los ejercicios propuestos en esta ficha. Están diseñados para aprender los aspectos esenciales del uso de la librería y te facilitarán la realización del ejercicio entregable.
Ejercicio entregable	Debes realizar la tarea propuesta en esta ficha y entregarla en Moodle.

4 Ejercicios de entrenamiento

En este apartado te proponemos una hoja de ruta para que te adiestres de forma autónoma en los objetivos de la práctica. Te recomendamos que hagas todos los ejercicios en el orden en el que están propuestos.

Crear hilos

Como primer ejercicio te proponemos que crees un programa en C que lance un hilo. Tanto el hilo principal como el hilo creado deben mostrar, con un retardo de un segundo, N mensajes por pantalla. Una posible salida por pantalla sería la siguiente:

```
Hilo principal: Iteración 1
```

```
Hilo principal: Iteración 2
Hilo hijo: Iteración 1
Hilo principal: Iteración 3
Hilo hijo: Iteración 2
…
…
```

Recuerda que para trabajar con la biblioteca *pthreads*, hay que incluir la cabecera **<pthreads.h>**. Además, para compilar los programas en C, hay que añadir la opción del compilador "**`-lpthread`**". Esta opción incluye la biblioteca *pthreads* en la generación del binario ejecutable.

Sincronización básica con pthread_join()

Escribe un programa que lance varios hilos según se explica a continuación:

En primer lugar, hay que lanzar veintiséis hilos. Cada uno de ellos escribirá por pantalla diez veces una letra mayúscula de la "A" a la "Z" (sin la eñe).

Cuando esos veintiséis hilos hayan finalizado, se escribirá un mensaje por pantalla indicando esta circunstancia. A continuación comenzará a ejecutarse una nueva batería de hilos, que en esta ocasión escribirán diez veces las letras minúsculas de la "a" a la "z". Cada hilo se encargará de escribir una letra diez veces.

Cuando los hilos de las minúsculas hayan terminado todos, se escribirá un mensaje y se lanzarán diez hilos que escribirán diez veces las cifras del "0"al "9". Tras ello se escribirá un mensaje de despedida y finalizará la ejecución del programa.

Para resolver esta tarea, habrá que crear hilos con **`pthread_create()`** y sincronizarlos con **`pthread_join()`**.

Problemas al manipular variables compartidas

Como hemos explicado en la teoría de la asignatura, el acceso no controlado a datos compartidos puede ocasionar problemas. En los ficheros fuentes que proporcionamos tenemos el programa **`compartida.c`**, en el que se lanzan dos hilos que están modificando continuamente una variable compartida, mientras la función *main()* visualiza periódicamente el valor actual de la variable.

Si la variable se modificara de forma atómica, a lo largo de su vida sólo podría adquirir los valores 1, 2 y 3. Sin embargo, los accesos concurrentes sin control provocan que adquiera casi cualquier valor.

Añade código en los accesos compartidos para tratar de que las modificaciones a la variable sean atómicas. Utiliza un booleano compartido (solución incorrecta) y después implementa el algoritmo de Peterson (solución correcta para secciones críticas de dos procesos).

Sincronización básica con cerraduras

Resuelve los dos ejercicios anteriores empleando como herramienta de sincronización cerrojos y variables condición.

5 Ejercicio entregable

El ejercicio entregable consistirá en desarrollar dos aplicaciones multihilo que resuelven problemas de sincronización: en la primera se regulará el acceso concurrente a un búfer circular; la segunda será un problema de sincronización que elegirás dentro de un catálogo.

La primera parte contribuirá en un 60% a la calificación de la práctica, mientras que la segunda parte aportará un 40%. Sin embargo, para aprobar la práctica es necesario superar ambas partes.

Primera parte: búfer circular

Dentro del material de la actividad publicado en *Moodle* hay un programa en C en el que se accede a un búfer circular. El búfer tiene capacidad para almacenar un número máximo de elementos que en nuestro caso particular serán de tipo "entero largo" (en C se corresponde con el tipo **`long)`** El programa accede al búfer circular para insertar o extraer un elemento. El búfer y las funciones que acceden al búfer están definidas en el fichero **`Buffer_Circular.h`** y consiste en dos operaciones:

- **`Inserta_Item(void *elemento)`**, sirve para insertar el elemento apuntado por elemento en el búfer circular.
- **`Extrae_Item(void *elemento)`**, sirve para extraer un elemento y almacenarlo en la zona de memoria apuntada por elemento.

Estas operaciones de inserción y extracción están implementadas de tal forma que su ejecución concurrente da problemas. Para evidenciar estos problemas se han introducido retardos en las operaciones de inserción en el búfer cuando el número a insertar es impar, como puede verse en el fichero de implementación **`Buffer_Circular.c`**. En consecuencia, cuando varios hilos concurrentes insertan y extraen elementos del búfer, el resultado de las operaciones puede ser incorrecto o dicho de otra forma, no podemos garantizar el correcto funcionamiento de las operaciones, dando lugar, por ejemplo, a la pérdida de elementos del búfer. Esto puede comprobarse compilando el programa y observando el resultado por pantalla.

El objetivo de esta tarea es añadir a las rutinas de los hilos unas operaciones de sincronización que resuelvan el problema para:

1. Permitir que un mismo hilo pueda insertar o extraer N elementos, en vez de sólo uno tal y como está contemplado en el código suministrado.
2. Garantizar la correcta inserción o extracción de elementos en el búfer.

Nota: En el archivo **`leeme.txt`** se describe cómo compilar y ejecutar el código C suministrado. Este código está organizado en tres archivos: **`Buffer_Circular.h`**, **`Buffer_Circular.c`** y **`test_hilos.c`.**

Segunda parte: implementar otro problema de sincronización

Se trata de elegir algún problema clásico de sincronización entre procesos e implementarlo mediante un conjunto de hilos de *pthreads*. En donde haga falta sincronizar procesos, se utilizarán cerrojos y variables condición.

Cada problema tiene una dificultad diferente y por ello una puntuación máxima distinta. Estos son los problemas propuestos y su puntuación relativa en una escala de 1 a 4:

- Lectores/Escritores (preferencia lectores=1; preferencia escritores=2)
- Procesos en equilibrio (1.5)
- Los fumadores (2)
- El taller de costura (2)
- Los filósofos sin interbloqueo (2.5)
- El barbero dormilón (2.5)
- El puente estrecho (inanición no resuelta=2.5; sin inanición= 4)
- El baño mixto (inanición no resuelta=2.5; sin inanición= 4)
- Caníbales y misioneros (4)

La definición de los problemas la puedes encontrar en la Web de la Asignatura (http://sopa.dis.ulpgc.es/fso/teoria/Lista%20ejercicios%20concurrencia.pdf).

Entrega del ejercicio

Recuerda que debes entregar todos los ficheros fuentes que implementan la solución a los ejercicios entregables, junto con la ficha correspondiente en formato PDF. Empaqueta todos los ficheros en un archivo **tar.gz** o **zip** y súbelo al Campus Virtual.

En la ficha, no olvides explicar bien qué estrategia has seguido para solucionar los problemas de sincronización.

6 Criterios de evaluación

La práctica 4 se evaluará siguiendo los criterios que se enumeran a continuación y que se encuentran agrupados en dos grandes bloques:

- Requisitos imprescindibles **(5 puntos)**.
- Calidad del código **(5 puntos) (60% primera parte; 40% segunda parte)**.

Requisitos imprescindibles (5 puntos)

Tal y como se indica en el enunciado de la práctica, el ejercicio entregable consiste en desarrollar dos aplicaciones multihilo que resuelven problemas de sincronización: la primera regula el acceso concurrente a un búfer circular; la segunda es un problema de sincronización que se debe elegir dentro de un catálogo. Para superar la práctica, ambas aplicaciones deben funcionar de acuerdo con las especificaciones indicadas en la ficha:

- Acceso concurrente al búfer circular: Permitir que un mismo hilo pueda insertar o extraer N elementos y garantizar la correcta inserción o extracción de elementos en el búfer.
- Problema de sincronización: el problema de sincronización seleccionado debe cumplir las especificaciones del enunciado.

Adicionalmente se deben cumplir los siguientes requisitos:

- El código está correctamente separado en módulos (.h,.c).
- Se documentan las pruebas realizadas.
- Se ha entregado una ficha en PDF, entendible y conforme con la plantilla.
- La ficha describe cómo debe compilarse y probarse el código.
- La entrega se ha realizado como un archivo *zip*, *tar* o *gzip*.

Calidad del código (5 puntos)

- **Mantenibilidad**: legibilidad del código (identificadores entendibles para variables, funciones, constantes, etc.); organización modular; reutilización de código (no hay bloques duplicados de código); facilidad de parametrización/configuración; rutinas para testeo del código; etc. **(1,5 puntos)**.
- **Eficiencia**: implementación eficiente en cuanto a tiempo de ejecución y a consumo de recursos, especialmente en lo que se refiere al manejo de herramientas de sincronización **(2,5 puntos)**.
- **Robustez**: comprobaciones de errores, mensajes de error, comprobación de límites de arrays, cuidado con las variables sin inicializar, cuidado con el uso de punteros, etc. **(1 punto)**.