Programación Concurrente y Distribuida

Prácticas 4 y 5 Curso 2021/22 3º Curso

Threads

Introducción

La unidad mínima de ejecución en un sistema operativo se denomina *thread*. Conceptualmente un *thread* existe dentro de un proceso. Es decir, cuando se invoca un programa, el sistema operativo crea un nuevo proceso que a su vez crea un único *thread* que ejecuta secuencialmente las instrucciones del programa invocado. Dicho *thread* puede crear nuevos *threads* ejecutando el mismo programa dentro del mismo proceso, lo que implica que todos los *threads* de un proceso comparten código y memoria, aunque cada *thread* puede ejecutar una parte diferente del programa. Es decir, a diferencia de lo que ocurre cuando se crea un proceso, cuando se crea un *thread* no se hace ningún tipo de copia, sino que ambos *threads* comparten el mismo espacio de memoria, los mismos descriptores de ficheros, así como cualquier otro recurso del sistema. Si un *thread* modifica el valor de una variable, el resto de *threads* "verán" dicha variable modificada; si un *thread* cierra un fichero, el resto de *threads* no podrán leer ni escribir sobre dicho fichero.

Mientras que una aplicación se divide en procesos en tiempo de diseño, los *threads* no afectan a la arquitectura de la aplicación, sino que permiten a los procesos llevar a cabo ciertas tareas de manera no lineal (la recepción de eventos del interfaz de usuario, la recepción de datos a través de la red, etc.).

Creación de Threads

Objetivos

Esta práctica tiene como objetivos fundamentales:

- Introducir el concepto de *thread* y comprender las diferencias frente a los procesos.
- Programar procesos con varios *threads* y utilizar semáforos para su sincronización.

Entorno

Los ordenadores del laboratorio ya están preparados para realizar las prácticas. Si el alumno desea realizar las prácticas en su ordenador deberá tener instalado Linux (en el laboratorio está instalada la distribución OpenSuse 12.3 y es sobre la que se deberá garantizar el correcto funcionamiento de las prácticas).

Linux implementa *threads* a través de la librería estándar POSIX *pthreads*. Dicha librería, cuyas funciones y tipos están declarados en *<pthread.h>* no está incluida en la librería C estándar, sino en la librería `libpthread`, por lo que es necesario enlazar dicha librería incluyendo `-lpthread` en la línea de compilación.

Ejercicio 1 (mini ejemplos):

Consulte el *man* de `pthread_create`.

Escriba un proceso `p1` que cree un `thread` encargado de atender el teclado (finalizando cuando se pulse la tecla `'q'`), mientras que el principal imprima un mensaje cada segundo con el número de caracteres leídos del teclado.

Ejercicio 2 (mini ejemplos):

Escriba un proceso `p2` que cree tantos `threads` como parámetros reciba en la línea de comandos. Cada `thread` deberá imprimir en pantalla la posición y la cadena de texto del parámetro correspondiente. Todos los `threads` deberán ejecutar la misma función, que recibirá como argumentos la posición y la cadena a imprimir.

Sincronizando Threads

De manera similar a la sincronización entre procesos padre e hijos, un *thread* puede sincronizarse con la finalización de otro *thread* utilizando la función *pthread_join*.

Ejercicio 3 (mini ejemplos):

Consulte el *man* de `pthread_join`.

Escriba dos soluciones alternativas, `p3 y p4`, al grafo de precedencia del ejercicio 5 de la práctica 2 utilizando `threads.`

Semáforos con Threads

Linux incluye una librería que implementa semáforos para *threads*. El fichero de cabecera se encuentra en <*semaphore.h*>, y las principales funciones son *sem_init, sem_wait, sem_post* (el equivalente a un *signal*) y *sem_destroy.*

Ejercicio 4

Escriba un nuevo proceso `lectores` que reciba como parámetros dos valores enteros: `N1`, indicando el número total de lectores posibles; `N2` indicando el número máximo de lectores que pueden leer simultáneamente (`N2 < N1`). El `thread` *principal* deberá ser el encargado de atender al teclado y crear `N1` nuevos `threads` que simulen cada uno de los lectores. El funcionamiento de cada `thread` *lector* será el siguiente:

1. Estará en un bucle esperando a que el `thread` *principal* le indique que intente leer.
   "[Lector i] -> Esperando a intentar leer…"
   "[Lector i] -> Intentando leer…"
2. Cuando consiga acceso imprimirá un mensaje y se simulará que está leyendo hasta que se lo indique el `thread` *principal*.
   "[Lector i] -> Leyendo…"
   "[Lector i] -> Fin lectura" (vuelve al punto 1)

El `thread` *principal* mostrará , en un bucle, un menú con tres opciones:

1. *"Intentar leer"*
   > introduzca el número del lector (de 1 a N1):
2. *"Finalizar leer"*
   > introduzca el número del lector (de 1 a N1):
3. *"Salir"*

Toda la sincronización necesaria deberá implementarse con semáforos.

Ejercicio 5

Escriba un nuevo proceso `escritores` que reciba como parámetro un valor entero `N3`, indicando el número total de escritores posibles. El `thread` *principal* deberá ser el encargado de atender al teclado y crear `N3` nuevos `threads` que simulen cada uno de los escritores. El funcionamiento de cada thread *escritor* será el siguiente:

1. Estará en un bucle esperando a que el `thread` *principal* le indique que intente escribir.
   "[Escritor i] -> Esperando a intentar escribir…"
   "[Escritor i] -> Intentando escribir…"
2. Cuando consiga acceso imprimirá un mensaje y se simulará que está escribiendo hasta que se lo indique el `thread` *principal*.
   "[Escritor i] -> Escribiendo…"
   "[Escritor i] -> Fin escritura" (vuelve al punto 1)

El `thread` *principal* mostrará , en un bucle, un menú con tres opciones:

1. *"Intentar escribir"*
   > introduzca el número del escritor (de 1 a N3):
2. *"Finalizar escribir"*
   > introduzca el número del lector (de 1 a N3):
3. *"Salir"*

Toda la sincronización necesaria deberá implementarse con semáforos.

Ejercicio 6

Escriba un nuevo proceso `lectores-escritores` que incluya la funcionalidad de los dos ejercicios anteriores añadiendo la sincronización necesaria para que *lectores* y *escritores* accedan correctamente al *papel*, teniendo en cuenta que los *escritores* deben ser prioritarios respecto a los *lectores*.

Ejercicio 7: (opcional)

Modifique la solución anterior para que un *lector* sólo pueda ser adelantado por `K` procesos escritores.

Resumen

Los principales resultados esperados de esta práctica son:

- Adquirir experiencia en la programación concurrente con *threads*.
- Aprender a sincronizar *threads*.
- Resolver problemas de concurrencia utilizando semáforos.

Como trabajo adicional del alumno, se proponen las siguientes líneas:

- Resolver los ejercicios de la práctica 3 utilizando *threads*.
- Implementar otros problemas clásicos de concurrencia (productor-consumidor, filósofos, etc.) con *threads*.