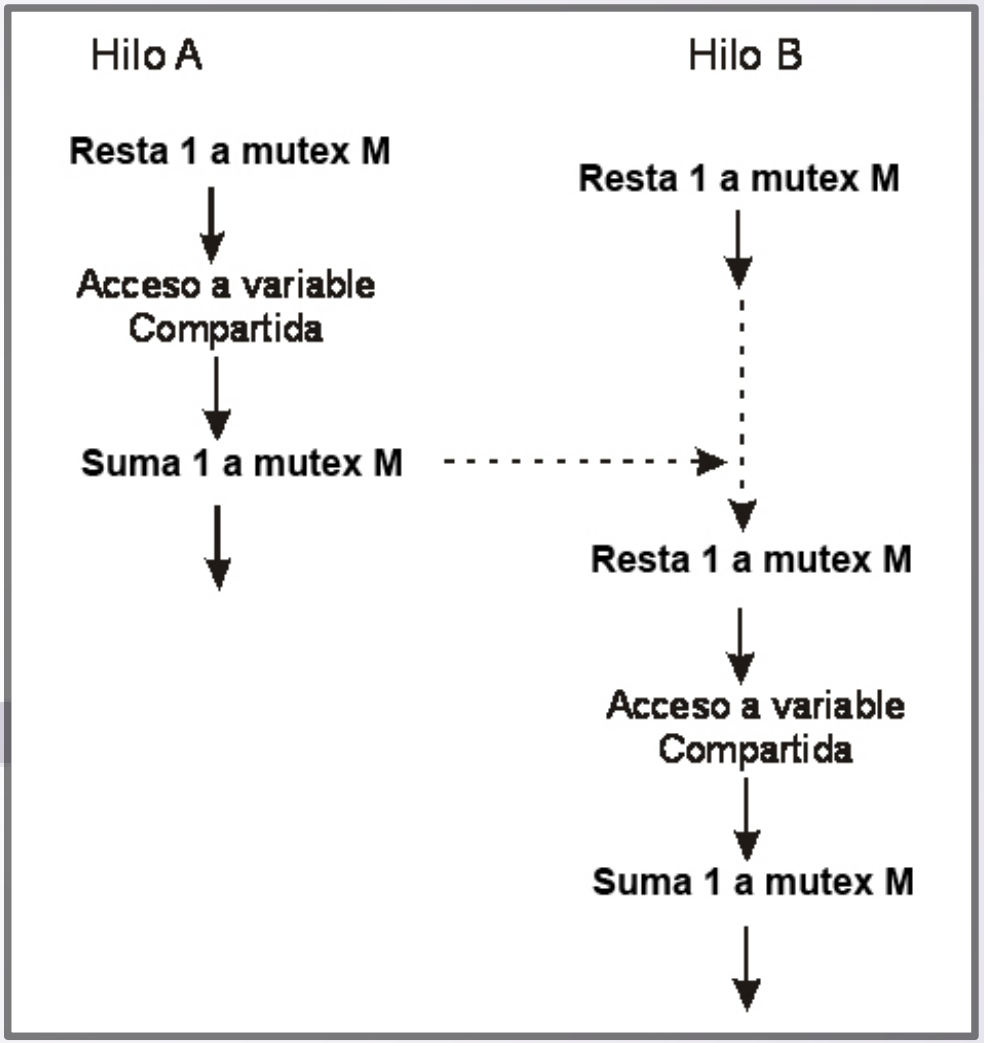
MUTEX

MUTEX

Cuando se tiene acceso a un recurso compartido (sección crítica) la ejecución debe ser atómica, es decir, la ejecución del acceso no debe ser interrumpido por otro hilo que desea acceder simultáneamente al mismo recurso compartido.

- Para sincronizar el uso de variables compartidas están los mutex (abreviatura de la exclusión mutua).
- Un mutex sirve para asegurarse de que sólo un hilo a la vez puede acceder a la variable compartida.
- Un mutex tiene dos estados: cero y uno.
- Solo un hilo a la vez puede decrementar un mutex que estaba en estado 1.

Si varios hilos intentan decrementar un mutex en estado 1, debido a que un solo hilo puede decrementarlo, los otros hilos permanecen a su vez bloqueados a la espera de poder restar 1 al mutex.



Pasos para utilizar el mutex:

- Restar 1 (lock) al mutex asignado al recurso compartido
- Acceder al recurso compartido
- Sumar 1 (unlock) al mutex

El uso del mutex es de carácter **consultivo**, en vez de obligatorio. Es decir que un hilo es libre de ignorar el uso de un mutex. Con el fin de manejar con seguridad las variables compartidas, todos las hilos deben **cooperar** en el uso de un mutex, respetando las reglas de su uso.

Inicialización de Mutex estático

Un mutex es una variable del tipo `pthread_mutex_t`

Para la asignación estática del mutex hacemos:

```
#include <pthread.h>
pthread_mutex_t mtx = PTHREAD_MUTEX_INITIALIZER;
```

Antes de restar o sumar 1 a un mutex debemos inicializarlo.

Al inicializar un mutex en forma estática este solo puede tener los atributos por defecto.

Resta de un mutex

```
#include <pthread.h>

int pthread_mutex_lock(pthread_mutex_t *mutex);
```

Devuelve 0 si tuvieron éxito y un número positivo en caso de error

La función `pthread_mutex_lock()` resta 1 a un mutex.
Si el mutex está en estado 1, la función `pthread_mutex_lock()` resta 1 al mutex (pasa a estado 0) y vuelve inmediatamente.
Si el mutex está en estado 0 por otro hilo, la función `pthread_mutex_lock()` bloquea el hilo hasta que el mutex pase a estado 1, en cuyo momento se resta 1 al mutex (pasando a estado 0) y la función retorna.

Función `pthread_mutex_trylock()`

La función **`pthread_mutex_trylock()`** es la misma que **`pthread_mutex_lock()`**, excepto que si el mutex está en estado 0, **`pthread_mutex_trylock()`** falla, y vuelve con el error EBUSY.

```
#include <pthread.h>
int pthread_mutex_trylock(pthread_mutex_t * mutex);
```

Devuelve 0 si tiene éxito, o el error EBUSY indicando que otro hilo tiene el mutex.

Función `pthread_mutex_trylock()`

La función **`pthread_mutex_trylock()`** es la misma que **`pthread_mutex_lock()`**, excepto que si el mutex está en estado 0, **`pthread_mutex_trylock()`** falla, y vuelve con el error EBUSY.

```
#include <pthread.h>
int pthread_mutex_trylock(pthread_mutex_t * mutex);
```

Devuelve 0 si tiene éxito, o el error EBUSY indicando que otro hilo tiene el mutex.

Suma a un mutex

```
#include <pthread.h>

int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

Devuelve 0 si tiene éxito, un número positivo en caso de error.

La función `pthread_mutex_unlock()` suma 1 al mutex cuyo estado previo es cero.

Es incorrecto sumar 1 a un mutex que no está en estado 0.

Inicialización mutex dinámico

```
#include <pthread.h>

int pthread_mutex_init(pthread_mutex_t *mutex, const
pthread_mutex attr_t *attr);
```

Devuelve 0 si tiene éxito o un error positivo en caso de error

El argumento de mutex identifica el mutex para ser inicializado.

Si attr es NULL se tomaran los atributos por defecto

Atributos Mutex

El argumento `mtxattr` es un puntero a un objeto `pthread_mutexattr_t` que previamente se debe declarar para definir los atributos del mutex

```
#include <pthread.h>
pthread_mutexattr_t mtxattr;
```

Inicialización de atributos de un mutex

```
#include <pthread.h>
int pthread_mutexattr_init(pthread_mutexattr_t
*attr);
```

Devuelve 0 si tiene éxito, un número positivo en caso de error

Seteo de tipo en atributos

```
#include <pthread.h>

int pthread_mutexattr_settype(pthread_mutexattr_t
*attr, int type);
```

Devuelve 0 si tuvo éxito o un error positivo en caso de error

Type:
THREAD_MUTEX_ERRORCHECK
PTHREAD_MUTEX_RECURSIVE
PTHREAD_MUTEX_DEFAULT
PTHREAD_MUTEX_NORMAL

Tipos de mutex

PTHREAD_MUTEX_ERRORCHECK: Realiza comprobación de errores en todas las operaciones. Este tipo de mutex es más lento que un mutex normal, pero puede ser útil como una herramienta de depuración.

PTHREAD_MUTEX_RECURSIVE: Cuando un hilo adquiere por primera vez el mutex , el contador de lock se establece en 1. Cada operación posterior de lock hecha por el mismo hilo incrementa el contador , y cada operación de unlock decrementa el contador. El mutex se libera sólo cuando el contador cae a 0.

En mutex estáticos;THREAD_RECURSIVE_MUTEX_INITIALIZER_NP

PTHREAD_MUTEX_DEFAULT: Es el tipo predeterminado de mutex, si usamos PTHREAD_MUTEX_INITIALIZER o especificamos attr como NULL en una llamada a pthread_mutex_init().
En Linux, un mutex PTHREAD_MUTEX_DEFAULT se comporta como un mutex PTHREAD_MUTEX_NORMAL.

Destrucción de un mutex

```
#include <pthread.h>
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

Devuelve 0 si tiene éxito, un número positivo en caso de error

Es seguro destruir un mutex sólo cuando esta en estado 1, y ningún un hilo tratará subsecuentemente de restarle 1 (lock).

Si el mutex se encuentra en una región de memoria asignada dinámicamente, entonces debe ser destruido antes de la liberación de esa región de la memoria.

Una mutex que ha sido destruido con `pthread_mutex_destroy()` posteriormente se puede reinicializar con `pthread_mutex_init()`.

No es necesario ejecutar `pthread_mutex_destroy()` si el mutex se ha inicializado estáticamente (`PTHREAD_MUTEX_INITIALIZER`)

Interbloqueos de Mutex

A veces, un hilo necesita acceder simultáneamente a dos o más recursos compartidos, cada uno de ellos se rige por un mutex por separado.

El lock y unlock de los mutex puede producir interbloqueos

Hilo A

```
1. pthread_mutex_lock(mutex1);
2. pthread_mutex_lock(mutex2);
```

Blocks

Hilo B

```
1. pthread_mutex_lock(mutex2);
2. pthread_mutex_lock(mutex1);
```

Blocks

Para evitar esto todos los hilos deben restar (lock) y sumar (unlock) los mutex en el mismo orden. A esto se llama jerarquía de mutex.

Costo de usar un mutex:

Los programas que utilizan mutex consumen más ciclos de reloj, por tanto, insume más tiempo su ejecución.

Errores:

Un solo hilo **no** puede restar 1 (lock) al mismo mutex dos veces (sin hacer una suma).

Un hilo **no** puede sumar 1 (unlock) a un mutex que no está actualmente en estado 0.

Bibliografía

Kerrisk, Michael. *The linux programming Interface.* 2011. **Capítulo 30.1.**