Arquitectura de Sistemas

Práctica 11: El problema lectores/escritores

Gustavo Romero López

Updated: 23 de mayo de 2025

Departamento de Ingeniería de Computadores, Automática y Robótica

Objetivos

- Resolver el problema lectores/escritores de la forma más eficiente y/o justa posible.
- Para ello utilice cualquiera de los mecanismos de sincronización y exclusión mutua vistos hasta ahora.
- Pruebe los cerrojos específicos de lectura/escritura de pthread y compare los resultados con los suyos.
- Para facilitar el proceso de arranque partiremos de analizar un par de soluciones incorrectas:
  - `le.cc`: condición de carrera + inanición.
  - `mutex.cc`: sincronización demasiado restrictiva.

Semáforos (POSIX)

`#include <semaphore.h>` Cabecera C/C++.

`sem_t` Tipo semáforo.

`sem_init(sem, attr, valor)` Crea un semáforo `sem` inicializado a `valor` y con los atributos `attr`.

`sem_destroy(sem)` Destruye el semáforo `sem`.

`sem_wait(sem)` Si el valor del semáforo `sem` es positivo lo decrementa y retorna inmediatamente. En otro se bloquea hasta poder hacerlo.

`sem_trywait(sem)` Versión no bloqueante de `sem_wait(sem)`. En cualquier caso retorna inmediatamente. Es necesario comprobar la salida antes de continuar.

`sem_post(sem)` Incrementa el valor del semáforo `sem`. En caso de cambiar a un valor positivo desbloquea a algun llamador de `sem_wait(sem)`.

Cerrojos Lector/Escritor (pthread)

`pthread_rwlock_t` Tipo cerrojo lector/escritor.

`pthread_rwlock_init(rwlock, attr)` Inicializa el cerrojo lector/escritor `rwlock` con los atributos `attr`.

`pthread_rwlock_destroy(rwlock)` Destruye el cerrojo `rwlock`.

`pthread_rwlock_rdlock(rwlock)` Adquiere el cerrojo `rwlock` para lectura.

`pthread_rwlock_tryrdlock(rwlock)` Intenta adquirir el cerrojo `rwlock` para lectura.

`pthread_rwlock_wrlock(rwlock)` Adquiere el cerrojo `rwlock` para escritura.

`pthread_rwlock_trywrlock(rwlock)` Intenta adquirir el cerrojo `rwlock` para escritura.

`pthread_rwlock_unlock(rwlock)` Libera el cerrojo `rwlock` en función de la versión de aquisición ejecutada con anterioridad, ya sea lector o escritor.

Mutex de C++

`#include <mutex>` Cabecera.

`std::mutex` Nombre de la clase.

- `lock()` Adquiere el semáforo. Si otra hebra lo ha bloqueado previamente entonces bloquea a la hebra actual hasta que la propietaria del semáforo lo deja libre.
- `unlock()` Desbloquea el semáforo.

`std::lock_guard` Envoltorio RAII que permite adquirir un semáforo en el bloque en ejecución.

Semáforos de C++

`#include <semaphore>` Cabecera.

`std::binary_semaphore` Nombre de la clase.

`std::counting_semaphore` Nombre de la clase.

`acquire()` Intenta decrementar el semáforo, en caso de éxito continua, en caso de fallo se bloquea.

`release()` Incrementa el valor del semáforo.

- Sólo en C++20: `-std=c++20`

`std::shared_mutex` de C++

`#include <shared_mutex>` Cabecera.

`std::shared_mutex` Nombre de la clase.

`lock()` Adquiere el semáforo de manera exclusiva. Si otra hebra lo ha bloqueado previamente entonces bloquea a la hebra actual hasta que la propietaria del semáforo lo deja libre.

`unlock()` Desbloquea el semáforo de manera exclusiva.

`lock_shared()` Adquiere el semáforo de forma compartida. Si otra hebra lo ha bloqueado previamente de manera exclusiva entonces bloquea a la hebra actual hasta que la propietaria del semáforo lo deja libre.

`unlock_shared()` Desbloquea el semáforo para uso compartido.

`std::shared_lock` Envoltorio RAII que permite adquirir un semáforo de forma compartida.

`std::unique_lock` Envoltorio RAII que permite adquirir un semáforo de forma exclusiva y traspasar su propiedad.

Copie el programa le.cc y **verifique** que la secuencia de ejecución no es correcta porque existen condiciones de carrera.

```
void seccion_critica(char c)
{
    for (char i = 0; i < 10; ++i)
        std::cout << c;
    std::cout << '\n';
}

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

void lector(char c)
{
    latch.arrive_and_wait();
    while (true)
```

```
    {
        seccion_critica(c);
    }
}

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

void escritor(char c)
{
    latch.arrive_and_wait();
    while (true)
    {
        seccion_critica(c);
    }
}
```

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

int main()
{
    for (size_t i = 0; i < N / 2; ++i)
    {
        std::thread(lector, '0' + i).detach();
        std::thread(escritor, 'a' + i).detach();
    }

    std::this_thread::sleep_for(12ms);
}

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

Copie el programa mutex.cc y **verifique** que está libre de condiciones de carrera aunque es una solución incorrecta.

```
void lector(char c)
{
    latch.arrive_and_wait();
    while (true)
    {
        std::lock_guard<std::mutex> lock(mutex);
        seccion_critica(c);
    }
}

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

void escritor(char c)
```

```
{
    latch.arrive_and_wait();
    while (true)
    {
        std::lock_guard<std::mutex> lock(mutex);
        seccion_critica(c);
    }
}
```

Trabajo: escriba su propia versión

- **Modifique** le.cc de forma que esté libre de condiciones de carrera y además permita el paralelismo entre lectores.
- En clase hemos visto dos soluciones: Bacon y Stallings.
- Compare con las otras soluciones.
- Pista: a lo mejor le puede venir bien un **interruptor**...

```
class interruptor
{
public:
    void lock(std::mutex &llave)
    {
        std::lock_guard<std::mutex> lock(mutex);
        if (++contador == 1)
            llave.lock();
    }

    void unlock(std::mutex &llave)
    {
        std::lock_guard<std::mutex> lock(mutex);
        if (--contador == 0)
            llave.unlock();
    }
```

```
protected:
    std::mutex mutex;
    size_t contador = 0;
};
```

Trabajo: torno.cc

- Si en su solución ha empleado un interruptor puede que exista inanición de los escritores.
- **Modifique** su solución de forma que siga libre de condiciones de carrera, permita el paralelismo entre escritores y esté libre de inanición.
- Pista: a lo mejor le puede venir bien un **torno**.

ejemplo de torno

```
semáforo s = 1;
...
s.esperar();
s.señalar();
...
```

Trabajo: conseguir justicia

- **Modifique** su solución de forma que funcione permitiendo una ejecución equilibrada de lectores y escritores.
- Evite las versiones que explícitamente favorecen a alguna de las partes.
- H. Ballhausen propuso una solución interesante desde el punto de vista de la equidad.[1]

[1] https://arxiv.org/pdf/cs/0303005

Trabajo: ¿soluciones demasiado restrictivas?

- Busque entre las soluciones que proporcionamos si existe alguna que sea demasiado restrictiva tal como `mutex.cc`.