Departamento de Computación,
Facultad de Ciencias Exactas y Naturales,
Universidad de Buenos Aires

TP2 - Pthreads

Sistemas Operativos

Primer Cuatrimestre de 2015

Apellido y Nombre	LU	E-mail
Cisneros Rodrigo	920/10	rodricis@hotmail.com
Rodríguez, Agustín	120/10	agustinrodriguez90@hotmail.com
Tripodi, Guido	843/10	guido.tripodi@hotmail.com

Contents

1 Introducción

2 Desarrollo y Resultados

3 Parte I – Desarrollo de Read-Write Lock

3.1 Ejercicios

- **Ejercicio 1** En primer lugar, deberán implementar un Read-Write Lock libre de inanición utilizando únicamente Variables de Condición POSIX y respetando la interfaz provista en los archivos backend-multi/RWLock.h y backend-multi/RWLock.cpp

3.2 Resultados y Conclusiones

3.2.1 Ejercicio 1

Nuestra implementación del Read-Write Lock se baso en el pseudocodigo implementado en el libro *The Little Book of Semaphores* al resolver la inanición producida en el problema de $Readers - writers$.

Se utilizaron 3 Semaphores, los cuales son:

- roomEmpty
- turnstile
- readers_mutex

Y ademas, un entero denominado *readers*

Comenzando por la implementación de los lectores, el pseudocodigo del libro mencionado es el siguiente:

```
turnstile.wait()
turnstile.signal()

readSwitch.lock(roomEmpty)
    # critical section for readers
readSwitch.unlock(roomEmpty)
```

De aquí nuestro código implementado fue el siguiente:

READERS LOCK

```
pthread_mutex_lock(&turnstile);
pthread_mutex_unlock(&turnstile);

pthread_mutex_lock(&readers_mutex);
readers++;
pthread_mutex_unlock(&readers_mutex);
```

Como se puede observar en el codigo del lock del read, el lector realiza un lock (wait) y unlock (signal) del Semaphores *turnstile* para tener su turno y que ningún otro lo saque.
Por consiguiente, se realiza el lock del mutex que se encuentra vinculado al entero *readers*, ya que este será aumentará su cantidad en 1 para que de esta manera nadie pueda modificarlo, y luego es liberado dicho mutex (*readers_mutex*).

Luego, nuestro $READ\ UNLOCK$ fue el siguiente:

READERS UNLOCK

```
pthread_mutex_lock(&readers_mutex);
readers--;
if (readers == 0) {
    pthread_cond_signal(&room_empty);
}
pthread_mutex_unlock(&readers_mutex);
```

En esta implementación, primero se realiza un lock del Semaphore vinculado al entero *readers* ya que este disminuirá en 1.
Luego, se realiza una consulta chequeando el valor del entero, si este es 0 se le dara un signal al Semaphore *room_empty* para notificarle al escritor que ya no queda ningun lector y puede proceder a escribir.
Por último, se libera el mutex *readers_mutex*.

Continuando con el escritor, el pseudocodigo fue el siguiente:

```
turnstile.wait()
roomEmpty.wait()
    # critical section for writers
turnstile.signal()

roomEmpty.signal()
```

De aquí, nuestra implementación final fue:

WRITERS LOCK

```
pthread_mutex_lock(&turnstile);
pthread_mutex_lock(&readers_mutex);
while(readers != 0)
      pthread_cond_wait(&room_empty, &readers_mutex);
pthread_mutex_unlock(&readers_mutex);
```

Inicialmente en nuestra implementación del WRITE LOCK, se realiza un lock del Semaphore *turnstile* para que nadie pueda quitarle el turno, se realiza un lock del mutex vinculado al entero, y luego se ingresa a un ciclo siempre que *readers* sea distinto de 0, esto se realiza para luego poder ejecutar la funcion *pthread_cond_wait* para que esta misma tenga un funcionamiento correcto y seguro al chequear la condición sobre $room_empty$.
Una vez que se salga del ciclo o no se ingrese al mismo se libera el mutex $readers_mutex$.

Luego, la implementación del unlock fue:

WRITERS UNLOCK

```
pthread_mutex_unlock(&turnstile);
```

Para la implementación del unlock del writer solo se libera el Semaphore *turnstile*.

De esta manera, con dicha implementación, siempre que llegue un escritor el mismo tendrá su turno sin producirse inanición. Ya que, en caso de haber lectores y llegar un escritor, estos terminaran de leer y en caso de llegar nuevos lectores deberán esperar a que el escritor finalice su ejecución.

4 Parte II: Desarrollo de Backend Multithreaded

4.1 Ejercicios

- **Ejercicio 2** En segundo lugar, deberán implementar el servidor de backend multithreaded inspirándose en el código provisto y lo desarrollado en el punto anterior.

4.2 Resultados y Conclusiones

4.2.1 Ejercicio 3

5 Bibliografía

- Cátedra de Sistemas Operativos - Clases teóricas y prácticas (1º Cuatrimestre 2015)
- The Little Book of Semaphores