

# Grenoble universities

## TD-TP Synchronisation with monitors

### (Assignment Two)

Arnaud Legrand, Sacha Hunold, Benjamin Negrevergne

2010

#### I. The Reader-Writer Problem

We consider a situation in which a shared resource is concurrently accessed by two types of entities : the readers and the writers. The readers access the resource without modifying it. The writers, can alter the resource. To keep the resource in a consistent state, multiple readers may access it concurrently but writers should access it exclusively. In other terms, when a writer is using a resource, no reader or writer can access this resource.

The reader-writer problem is a classical synchronization problem. This access scheme is typically used when working with files or memory.

In the following we will consider the following functions :

- `init(reader_writer_t)` is the initialization function
- `begin_read(reader_writer_t)` and `end_read(reader_writer_t)` are the functions used by each reader before and after reading
- `begin_write(reader_writer_t)` et `end_write(reader_writer_t)` are the functions used by a writer before and after writing.

The structure `reader_writer_t` is to be defined. It must contain all the variables used by the synchronization primitives.

#### II. Implementing a thread-safe linked list

You are asked to implement a **thread-safe linked list**. A **non** thread-safe linked list implementation is included in the archive<sup>1</sup> (`linked_list.c`, `linked_list.h`).

In the linked list the `list_insert()` and `list_remove()` are considered to be *writings*, since it alter the linked list. `list_exists()` is considered to be *reading*.

The provided implementation includes correct calls to the `begin_read()`, `end_read()`, `begin_write()` and `end_write()` primitives. Your work consists in implementing these primitives so the linked list behave correctly when accessed by multiple threads.

---

1. [http://membres-lig.imag.fr/negrevergne/ens/assignment\\_two.tar.gz](http://membres-lig.imag.fr/negrevergne/ens/assignment_two.tar.gz)

```

Thread nmbr 0, won.
Thread nmbr 1, won.
THREAD 0 TIME: +0s 0.046ms, TYPE : BEGIN WRITE
THREAD 0 TIME: +0s 0.047ms, TYPE : END WRITE
THREAD 0 TIME: +0s 0.047ms, TYPE : BEGIN READ
THREAD 0 TIME: +0s 0.047ms, TYPE : END READ
THREAD 1 TIME: +0s 0.062ms, TYPE : BEGIN WRITE
THREAD 1 TIME: +0s 0.065ms, TYPE : END WRITE
THREAD 1 TIME: +0s 0.065ms, TYPE : BEGIN READ
THREAD 1 TIME: +0s 0.066ms, TYPE : END READ
THREAD 1 TIME: +0s 0.066ms, TYPE : BEGIN WRITE
THREAD 1 TIME: +0s 0.066ms, TYPE : END WRITE
THREAD 1 TIME: +0s 0.066ms, TYPE : BEGIN READ
THREAD 1 TIME: +0s 0.066ms, TYPE : END READ
THREAD 1 TIME: +0s 0.067ms, TYPE : BEGIN WRITE
THREAD 1 TIME: +0s 0.067ms, TYPE : END WRITE
THREAD 1 TIME: +0s 0.067ms, TYPE : BEGIN READ
THREAD 1 TIME: +0s 0.067ms, TYPE : END READ
THREAD 1 TIME: +0s 0.067ms, TYPE : BEGIN WRITE
THREAD 1 TIME: +0s 0.068ms, TYPE : END WRITE
THREAD 1 TIME: +0s 0.068ms, TYPE : BEGIN READ
THREAD 1 TIME: +0s 0.068ms, TYPE : END READ

```

FIGURE 1 – output of : `linked_list_simple_test`

The archive also contains a simple test file `linked_list_simple_test.c`. Read it carefully. Any successful test must return `EXIT_SUCCESS`.

**Question II.1.** *Considering that no synchronization has been implemented yet, what result could you expect from this test. If you run the test, it won't actually fail, why? What do you conclude on the validity of this test?*

The code of `begin_read()`, `end_read()`, `begin_write()` and `end_write()` contains calls to the tracing module (`reader_writer_tracing.h`, `reader_writer_tracing.c`). This module allow one to record events and to prints them afterward.

**Question II.2.** *Fig. 1 is a possible output of `./linked_list_simple_test` 2. Using this output, how can you verify the consistency of the calls to `begin_read()`, `end_read()`, `begin_write()` and `end_write()`.*

**Question II.3.** *Write a naïve implementation of `begin_read()`, `end_read()`, `begin_write()` and `end_write()` in a new file named `reader_writer_1.c` using only mutexes<sup>2</sup>.*

**Question II.4.** *Write a test program using the tracing module to check the consistency of the calls to `begin_read()`, `end_read()`, `begin_write()` and `end_write()`.*

**Question II.5.** *What could be the definition of efficiency in this particular problem? In synchronization problems in general? How would you evaluate the efficiency of this solution according to your previous definition?*

---

2. You must uncomment the corresponding line in the `Makefile`

```

THREAD 0 TIME: +0s 0.068ms, TYPE : BEGIN READ
THREAD 1 TIME: +0s 0.069ms, TYPE : WAITS TO WRITE
THREAD 2 TIME: +0s 0.070ms, TYPE : BEGIN READ
THREAD 0 TIME: +0s 0.071ms, TYPE : END READ

THREAD 0 TIME: +0s 0.083ms, TYPE : BEGIN READ
THREAD 2 TIME: +0s 0.085ms, TYPE : END READ

THREAD 2 TIME: +0s 0.123ms, TYPE : BEGIN READ
THREAD 0 TIME: +0s 0.127ms, TYPE : END READ

THREAD 0 TIME: +0s 0.143ms, TYPE : BEGIN READ
THREAD 2 TIME: +0s 0.145ms, TYPE : END READ

THREAD 2 TIME: +0s 0.161ms, TYPE : BEGIN READ
THREAD 0 TIME: +0s 0.168ms, TYPE : END READ
...
THREAD 2 TIME: +0s 0.183ms, TYPE : END READ
THREAD 1 TIME: +0s 0.184ms, TYPE : BEGIN WRITE
THREAD 1 TIME: +0s 0.187ms, TYPE : END WRITE

```

FIGURE 2 – problematic behavior

### III. Improving concurrency

The previous solution does not meet the requirements in terms of concurrency since two readers cannot read the list simultaneously.

**Question III.1.** *Implement new `begin_read()`, `end_read()`, `begin_write()` and `end_write()` functions in `reader_writer_2.c` to improve the concurrent capabilities of your thread-safe list. You may use conditions.*

**Question III.2.** *Write a test program that verify that readers can access the list concurrently. Note that this test will fail with your first implementation and pass with your second implementation.*

### IV. Improving fairness

**Question IV.1.** *Introducing the events `WAITS TO READ` and `WAITS TO WRITE` for threads waiting in the functions `begin_read()` and `begin_write()`. Spot the fairness issue in Fig. 2.*

**Question IV.2.** *Discuss about the **fairness** of your previous solution.*

**Question IV.3.** *Build a fair solution for the reader writer problem in `reader_writer_3.c`. Write a corresponding test.*

*Hint : You must not mix the readers which arrived before a writer with the ones arriving after.*

Build an archive file (.tar.gz) including the following files :

- Your 3 reader\_writer implementations named `reader_writer_X.c`, one for each part of the assignment (part 1 : thread-safeness, part 2 : concurrency and part 3 : fairness)
- test programs, named `test_name_test.c`, at least 3.
- A Makefile that compile each test with each one of your implementation. (Therefore if you have completed the assignment, typing `make` should build at least 9 binary files).
- Any file that was in the provided archive file.
- A report pdf file (~ 5 pages) with the description of each test, its output for some implementations of the reader writer module and a detailed interpretation of the results.

E-mail the archive at : [Arnaud.Legrand@imag.fr](mailto:Arnaud.Legrand@imag.fr) **AND** [Benjamin.Negrevergne@imag.fr](mailto:Benjamin.Negrevergne@imag.fr)

If you cannot complete the assignment or if you cannot complete it on time : pick one or two exercise you are comfortable with and complete them (this includes the writing).