

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS
Information and Computer Science Department
ICS431: Operating System
Project Two (Term 062)

Project Title:

Read/Write Locks with Writer Precedence

Project Objective:

- To demonstrate your understanding of *Thread Creation and Execution* concept.
- To demonstrate your understanding of *Thread synchronization*
- To demonstrate your understanding of reading/writing problem

Project Duration:

You have 21 days (3 Weeks) to finish this project starting on the date of posting this project. The exact deadline will be announced through the WebCT.

Project Submission:

You are requested to submit both hardcopy and softcopy. Hardcopy should contain cover page, your source code, and sample outputs. Softcopy (source code only) will be submitted through WebCT.

Project Rules:

1. Submitting your project after the deadline will result in ZERO mark.
2. Any cheating in the project will result to ZERO mark.
3. You are allowed to work in teams of Two.

Programming Assignment:

This project is basically a programming assignment. Everything should be written in C language. Your code will be tested on the Solaris operating system or on the Linux operating system. So, you have to be sure that your code should be compiled and run on both operating systems. The programming assignment will be evaluated on several factors:

- **Documentation:** your code must include a design document describing your solution and must be well-commented.
- **Correctness:** your code must do what it supposed to do.
- **Structure:** your code should be broken into functions.
- **Style:** your code should be easy to read, well indented, well commented, and use clear, self-explanatory variable and function names.

Academic Honesty

This really should not be an issue, but recurring events have made the following necessary. You are encouraged to discuss the project material and concepts with other students in the class. However, all work that you submit must be your own. **Under no circumstances may you look at anyone else's code or show anyone else your code.** And while you may discuss the concepts and techniques used in your project, you may not discuss implementation details of the project itself.

If you are caught copying or otherwise turning in work that is not solely your own, you will fail the course and a letter will be sent to your Department, CCSE, and to the Registrar office.

The bottom line is that you are expected to conduct yourself as a person of integrity — you are expected to adhere to the highest standards of academic integrity. This means that plagiarism in any form is completely unacceptable. As a (soon to be) computing professional, I encourage you to consult the code of ethics appropriate to your discipline.

The Assignment

Implement a mutex that distinguishes between read locks and write locks. Multiple read locks may be granted simultaneously (to distinct threads), but only one write lock may be granted at any one moment. Also, a write lock cannot be granted if there are active read locks. Writers should be given precedence. If a writer is waiting while a reader holds the lock, new readers should wait for the writer. When a lock is released, any waiting writers should be given priority. A reader may upgrade a read lock to a write lock. The upgrade must be performed atomically, in the sense that there must be no window between the release of the read lock and the granting of the write lock. If a deadlock is detected, the `upgrade()` function should return false (with the read lock still held), If it succeeds, it should return true, with the read lock converted atomically to a write lock.

This assignment is to be written in the C programming language. You will implement a `RWMutex` abstraction that has the following functions (see below). You should define a `RWMutex` type (`rwmutex_t`) and pass it as the first parameter to each of the functions. The prototypes for the `RWMutex` abstraction are as follows:

```
rwmutex_t * create_rwlock ( )
```

Allocate and initialize a new `rwmutex`.

```
void write_lock (rwmutex_t *)
```

Get a write lock to the mutex. When the function returns, the thread has a write lock to the mutex. The function should block until a write lock is obtained.

```
void read_lock (rwmutex_t *)
```

Get a read lock to the mutex. When the function returns, the thread has a read lock to the mutex. The function should block if there are other threads that have a write lock. Several threads can hold a read lock at the same time.

```
int upgrade (rwmutex_t *)
```

Upgrade a thread from read lock to write lock. You can assume that a thread requesting this upgrade already has a read lock. If there are other threads that have a write lock and upgrade is not possible then this function should return 0 and should NOT block. If upgrade is possible (there are no read or write locks) then this function should upgrade the read lock to write lock atomically and return 1.

```
void read_unlock (rwmutex_t *)
```

Release the read lock on the `rwmutex`. You can assume that a thread calling this functions already has a read lock on the mutex.

```
void write_unlock (rwmutex_t *)
```

Release the write lock on the `rwmutex`. The thread calling this functions should have a write lock on the mutex.

Implementation Requirements

The entire implementation should be in the file `rwmutex.c`. You may NOT change the existing functions' parameters or return types. A header file named `rwmutex.h` is required to declare the `rwmutex` type and all the function prototypes. All operations need to be atomic which will require you to use mutexes and conditional variables. You will also need to use `pthread_equal()` function to check if two thread ids are same.