CS 435/535 – Fall 2024

Homework 2

The deadline for this homework is **Friday 22nd of November at 23:59 (midnight).** Please upload your solutions to LMS by the deadline.

Question 1: Concurrent Access to Linked List (40 points)

Give an example of a linked list and a sequence of memory accesses to the linked list in which the following pairs of operations can potentially result in problems (assuming no mutex/semaphore is used). Describe the problem (how it may actually occur, and what would be the consequence):
a) (8 points) Two deletes executed simultaneously
b) (8 points) An insert and a delete executed simultaneously
c) (8 points) A member and a delete executed simultaneously
d) (8 points) Two inserts executed simultaneously
e) (8 points) An insert and a member executed simultaneously

Question 2: Implementation of Read-Write Locks (60 points)

As discussed during the class, multithreaded linked lists cannot exploit the potential for simultaneous access to any node by threads that are executing Member() function if the mutex is used for the whole list. In this case, only one thread is allowed to access the entire list at any instant. The second approach was to use a mutex for each node in the list. However, this approach only allows one thread to access any given node at any instant, therefore, incurring a high overhead.

An alternative was using read-write locks. A read-write lock is somewhat like a mutex except that it provides two lock functions. The first lock function locks the read-write lock for reading, while the second locks it for writing. Multiple threads can thereby simultaneously obtain the lock by calling the read-lock function, while only one thread can obtain the lock by calling the write-lock function. Thus, if any threads own the lock for reading, any threads that want to obtain the lock for writing will block in the call to the write-lock function. Furthermore, if any thread owns the lock for writing, any threads that want to obtain the lock for reading or writing will block in their respective locking functions.

a) (30 points) Implement read-write locks with the synchronization primitives you have learned so far (e.g., mutex, semaphores, conditional variables, and barriers – you don’t need to use all of them, you have a freedom to use one or more of them based on your needs).

You should have the following functions:

```
int my_rdlock(my_rwlock_t∗ rwlock_p);
int my_wrlock(my_rwlock_t∗ rwlock_p);
int my_unlock(my_rwlock_t∗ rwlock_p);
```

(you should also have a read-write lock type *my_rwlock_t* that is going to be used in your read-write locking/unlocking functions)

With your own read-write lock functions implemented, you can protect the linked list functions that with discussed in the class (i.e., Member(), Insert(), Delete() functions) as follows:

```
my_rdlock(&rwlock);
Member(value);
my_unlock(&rwlock);
. . .
my_wrlock(&rwlock);
Insert(value);
my_unlock(&rwlock);
. . .
my_wrlock(&rwlock);
Delete(value);
my_unlock(&rwlock);
```

b) (30 points) Run the linked-list code with your read-write lock functions implementation and report the results (compare it with the pthreads own read-write lock implementation).
You should use the following parameters for the comparison:

i) 8 threads, 100.000 elements, 1,000,000 operations; 99% member, %0.5 insert and %0.5 delete;

ii) 8 threads, 100.000 elements, 1,000,000 operations; 90% member, %5 insert and %5 delete;

iii) 8 threads, 100.000 elements, 1,000,000 operations; 50% member, %25 insert and %25 delete;