Operating Systems – spring 2024
Tutorial-Assignment 9

Instructor: Hans P. Reiser

Submission Deadline: Monday, March 18th, 2024 – 23:59

A new assignment will be published every week. It must be completed before its submission deadline (late policy for programming assignments: up to two days, 10% penalty/day)

Lab Exercisess are theory and programming exercises discussed in the lab class. They are not graded, but should help you solve the graded questions and prepare for the final exam. Make sure to read and think about possible solutions before the lab class.

T-Questions are theory homework assignments and need to be answered directly on Canvas (quiz).

P-Questions are programming assignments. Download the provided template from Canvas. Do not fiddle with the compiler flags.

The topic of this assignment is synchronization.

Lab 9.1: Synchronization Primitives

a. Distinguish the various types of synchronization objects and summarize their respective operations' semantics: spinlocks, counting semaphores, binary semaphores, and mutex objects.

b. What is the difference between a reentrant and a non-reeentrant mutex?

c. The following code causes a deadlock. Explain why! How can this problem be solved (in a way such that all access to `value` is still protected by a mutex)

```
#include <pthread.h>

pthread_mutex_t lock;
int value = 10;

void initialize ()
{
    pthread_mutex_init(&lock, NULL);
}

void dosomethingelse() {
    pthread_mutex_lock(&lock);
    if(value==100) { value = 52; }
    pthread_mutex_unlock(&lock);
}

void dosomething() {
    pthread_mutex_lock(&lock);
    value = value + 10;
    dosomethingelse();
    value = value − 10;
    pthread_mutex_unlock(&lock);
}

int main() {
    initialize ();
    dosomething();
}
```

d. Using a CPU register for a spinlock's variable would be much faster than the implementation with a variable in main memory. Why would such a spinlock not work?

Lab 9.2: Producer-Consumer Problem

a. Solve the producer-consumer problem for the following buffer using a single pthread mutex and two semaphores:

```
#define BUFFER_SIZE 10
int buffer[BUFFER_SIZE];
int index = 0; // Current element in buffer
```

b. Assume you have only a single producer and a single consumer. Can you simplify the code from part (a) in that case?

c. Assume you have modified the code such that only two semaphores are used, but the mutex lock has been removed from the solution of part (a). What can go wrong now, if you have a single consumer, but multiple producers?

Lab 9.3: Synchronous vs. asynchronous IPC

a. How can you perform asynchronous interprocess communication if your operating system only provides synchronous IPC mechanisms?

b. How can you provide synchronous IPC if your operating system only offers asynchronous IPC mechanisms?

T-Question 9.1: Deadlocks

a. Lets assume your code has the following methods:

   (a) A function x() that locks – incrementally in this order – mutex A, B, and C

   (b) A function y() that locks – incrementally in this order – mutex B and D

   (c) A function z() that locks – incrementally in this order – mutex C and D.

   There is no lock preemption, i.e. each lock is held until it is released by the function. After locking the respective locks, each function performs some operations on a shared state and then releaseses all its locks before returning.

   Further assume multiple (> 3) concurrent threads invoking these three functions. Is this system at risk for deadlocks? Explain how (or why not)! **3 T-pt**

b. What is the main disadvantage of spinlocks? **1 T-pt**

true	false	
☐	☐	They minimize wasting CPU time as they interact with the scheduler using a system call.
☐	☐	They may waste a lot of CPU time if critical sections have a long execution time
☐	☐	They have little overhead in case there is very little lock contention
☐	☐	They can only be used on systems with a single CPU core

c. What are the two main types of semaphores? **1 T-pt**

true	false	
☐	☐	spinning and blocking
☐	☐	binary and counting
☐	☐	with and without internal state
☐	☐	bounded and unbounded

d. In the context of the second reader-writer problem, which of the following statements are true? **2 T-pt**

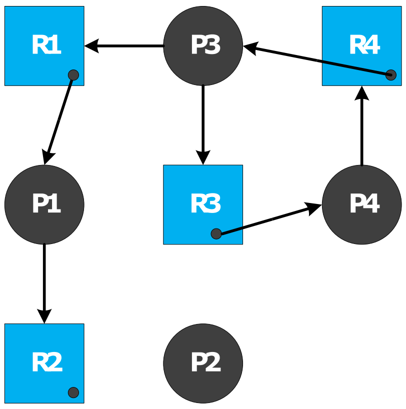
true	false	
☐	☐	Only one reader can access the critical section at a time.
☐	☐	While a writer is waiting to access the critical section, no additional readers will be allowed to enter the critical section.
☐	☐	Readers and writers can access the critical section concurrently.
☐	☐	Only one writer can access the critical section at a time.

e. What is one main disadvantage of disabling interrupts as mechanism for implementing mutual exclusion? **1 T-pt**

true	false	
☐	☐	It can only be used in kernel mode, not in user mode
☐	☐	It can only be used on multi-core CPUs, not on single-core CPUs
☐	☐	Mutual exclusion based on disabling interrupts provides reentrant (recursive) locks
☐	☐	It is typically implemented using system calls and thus has the overhead of a system call

T-Question 9.2: Resource Allocation Graph

a. Describe the situation shown in the resource allocation graph. Has a deadlock occurred in the above situation? Why, or why not? **2 T-pt**



P-Question 9.1: Simple Heap Allocator – Thread-Safe

Download the template **p1** for this assignment from Canvas. You may only modify and upload the file `malloc.c`. The template of this assignment is the same as the sample solution of Assignment 5. One problem with that solution is that it does not support multithreading, as the heap managed by `my_malloc` is a global, shared data structure, and the allocator lacks any mechanisms to coordinate the access to that shared data.

In this question you will solve this problem using mutexes.

a. Add correct coordination to `malloc.c`, such that `my_malloc` and `my_free` can be used in multiple, concurrent threads. **4 P-pt**

The main program uses multiple threads for testing. Usually, running main without adding coordination will result in a segmentation fault due to memory corruption. But note that this behaviour is highly nondeterministic.

b. Add correct coordination to the function `dumpAllocator`, to make sure that it does not print garbage or cause a segmentation fault in case of concurrent calls to `my_alloc` or `my_free`. **1 P-pt**

c. Make sure that turning on debug output (change the DEBUG define to 1) does not cause a deadlock. (Depending on how you solved the previous parts, this may or may not require additional work **1 P-pt**

P-Question 9.2: Multi-Mutex

Download the template **p2** for this assignment from Canvas. You may only modify and upload the file `multi_mutex.c`.

To prevent deadlocks one of the four necessary deadlock conditions must be broken. One method to achieve this is to acquire multiple locks 'atomically', that is acquire all or none. In this question you will write a mutex wrapper that locks multiple pthread mutexes.

a. Write a function that unlocks all pthread mutexes in the supplied `mutexv` array. The number of mutexes is given in `mutexc`. The function should return 0 on success, -1 otherwise. **1 P-pt**

```
int multi_mutex_unlock(pthread_mutex_t **mutexv, int mutexc);
```

b. Write a function that tries to lock all of the supplied mutexes, or none, if one of the mutexes cannot be acquired. That is, on failure, the function should release all previously acquired mutexes. The function should return 0 on success, -1 otherwise. It shall NOT block if some mutex is currently not available. **3 P-pt**

```
int multi_mutex_trylock(pthread_mutex_t **mutexv, int mutexc);
```

Total:
10 T-pt
10 P-pt