Operating Systems – spring 2022
Assignment 8

Instructor: Hans P. Reiser

Submission Deadline: Monday, March 14th, 2022 – 23:59

A new assignment will be published every week, right after (or a bit before) the last one was due. It must be completed before its submission deadline. (Hard deadlines, no extension.)

T-Questions are theory homework assignments. **The answers to the assignments must be uploaded to Canvas (in the quiz).**

P-Questions are programming assignments. Download the provided template from Canvas. Do not fiddle with the compiler flags. Submission instructions can also be found in first assignment.

The topic of this assignment synchronization.

T-Question 8.1: Deadlocks

a. Enumerate and explain the 4 necessary conditions for a deadlock. **2 T-pt**

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

true	false	
☐	☐	They have high overhead due to the system calls they use
☐	☐	They waste a lot of CPU time if critcal sections have a long execution time
☐	☐	They have high 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. The producer/consumer problem is also known as the **1 T-pt**

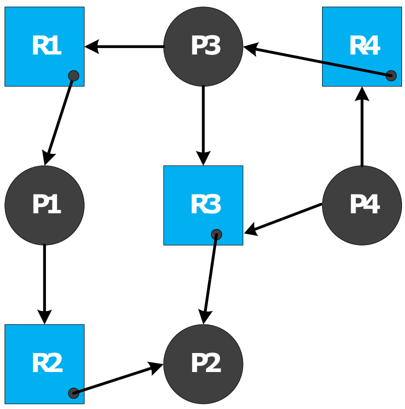
true	false	
☐	☐	readers/writers problem
☐	☐	mutual exclusion problem
☐	☐	bounded buffer problem
☐	☐	deadlock problem

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

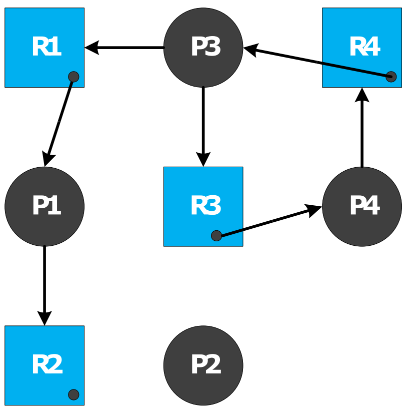
true	false	
☐	☐	It can only be used on user mode, not in kernel mode
☐	☐	It can only be used on multi-core CPUs, not on single-core CPUs
☐	☐	It is not suitable for synchronization between multiple threads on multiple CPU cores
☐	☐	It is typically implemented using system calls and thus has the overhead of a system call

T-Question 8.2: 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**



b. Does the situation change after $P2$ releases its resources and $R3$ is granted to $P4$? Explain! **2 T-pt**



P-Question 8.1: Simple Head 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 4. 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 that problem using mutexes. (You may also choose to use any other mechanisms, such as atomic instructions (CAS etc.), but such solutions will likely be more complex).

a. Add correct coordination to `malloc.c`, such that `my_malloc` and `my_free` can be used in multiple, concurrent threads. **5 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`. Note: You should consider that `dumpAllocator` might be called from within your `my_malloc` or `my_free` function for debugging purposes! **1 P-pt**

P-Question 8.2: 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