Synchronization with monitors

Master M1 MOSIG, Grenoble University

2015

Abstract

This lab aims at improving your understanding of synchronization problems and solutions. In this lab, you will use some of the primitives provided by the POSIX Threads library, and, in particular:

- `pthread_mutex_init()`
- `pthread_mutex_destroy()`
- `pthread_mutex_lock()`
- `pthread_mutex_unlock()`
- `pthread_cond_init()`
- `pthread_cond_destroy()`
- `pthread_cond_wait()`
- `pthread_cond_broadcast()`
- `pthread_cond_signal()`

You will need to consult the corresponding manual pages (e.g., `man 3 pthread_mutex_lock`) to learn about their precise semantics[1].

I. Getting started...

Question I.1: *When do you need synchronization among threads?*

Question I.2: *How does a mutex work? If there are multiple threads waiting for a mutex, when the* `unlock` *function is executed, which one will be deblocked? The WC example.*

Question I.3: *How does a monitor work? The parking or producer/consumer example.*

Question I.4: *How do you implement priorities between threads? The restaurant example.*

II. The Dining Philosophers

Several (N) philosophers meet for a dinner and sit at a round table. On the table, there are N bowls of rice. Each philosopher sits in front of a bowl of rice. In between each one of the bowls there is one single chopstick. Therefore there are also N chopsticks.

[1]Note: There are many others functions (and related man pages) in the Pthreads library. To get a full list, type: `man -k pthread`

During the dinner, the philosophers may either be **thinking** or **eating**. When a philosopher is thinking, it can last for a random amount of time. When he is eating, it can last for a small fixed amount of time.

A philosopher can think whenever he wants, however if he wants to eat he first needs to pick up the two chopsticks next to him (the one on the left and the one on the right). If at least one of the two chopsticks is not available (because a neighbor is currently using it), the philosopher must wait.

The above described problem can be modeled with threads. Each philosopher is a thread. Synchronization is mandatory to make sure that no chopstick is simultaneously picked up by two different philosophers.

Question II.1: *The behavior of each philosopher will be described by the function executed by his thread. Describe informally the contents of this function.*

Question II.2: *A first approach to address the dining philosophers problem consists in using N mutexes, with each mutex protecting the access to one chopstick. However, an implementation based on this approach is somewhat error-prone. What kind of issues can you envision with such an approach? (Note that you are not required to implement this approach.)*

Question II.3: *Design a an approach based on monitors to address the dining philosophers problem, avoiding the issues described in the previous question. Write a program to implement your approach and test it.*

Question II.4: *Does your design suffer from potential* starvation *issues? First, precisely explain what starvation means in the context of the dining philosophers. Then, consider how your design behaves regarding starvation:*

- *If you believe that your design is immune to starvation issues, explain why (i.e., prove it).*
- *Otherwise, explain how starvation issues may occur and propose a new design to prevent them (implement it and prove it).*