6

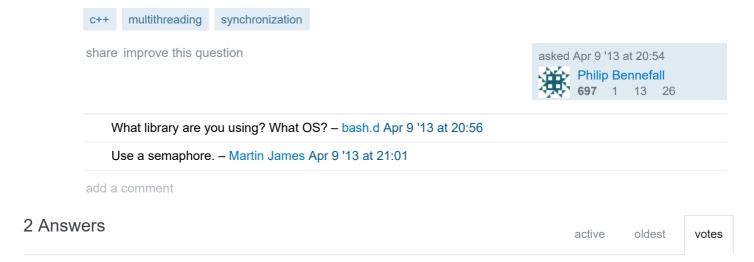
5



I have a problem with a deadlock in my code related to the use of condition variables. This is more of a design question than a pure code question. I have no problem actually writing code once I understand the correct design. I have the following scenario:

- 1. Thread A waits on a condition variable.
- 2. Thread B calls notify all, and thread A wakes up.
- This is of course what I want to happen, and is what does happen when everything works as expected. But sometimes, I get the following scenario instead:
 - 1. Thread A executes the code right before it begins to wait on the condition variable.
 - 2. Thread B calls notify all, thinking that thread A is waiting.
 - 3. Thread A begins waiting on the condition variable, not realizing that thread B already told it to stop waiting. Deadlock.

What is the best way to resolve this? I can't think of a reliable way to check whether thread A is actually waiting, in order to know when I should call notify_all in thread B. Do I have to resort to timed lock? I would hate to.



During the period just before Thread A waits on condition variable it must be holding a mutex. The easiest solution is to make sure that Thread B is holding the same mutex at the time it calls notify all. So something like this:

```
std::mutex m;
std::condition_variable cv;
int the_condition = 0;

Thread A: {
    std::unique_lock<std::mutex> lock(m);
    do something
    while (the_condition == 0) {
        cv.wait(lock);
    }
```

```
now the_condition != 0 and thread A has the mutex
do something else
} // releases the mutex;

Thread B: {
   std::unique_lock<std::mutex> lock(m);
   do something that makes the_condition != 0
   cv.notify_all();
} // releases the mutex
```

This guarantees that Thread B only does the notify_all() either before Thread A acquires the mutex or while Thread A is waiting on the condition variable.

The other key here, though, is the while loop waiting for the_condition to become true. Once A has the mutex it should not be possible for any other thread to change the_condition until A has tested the_condition, found it false, and started waiting (thus releasing the mutex).

The point is: what you are really waiting for is for the value of the_condition to become non-zero, the std::condition_variable::notify_all is just telling you that thread B thinks thread A should wake up and retest.

share improve this answer

edited Apr 9 '13 at 21:14

answered Apr 9 '13 at 21:05

Wandering Logic

2,459 1 11 21

+1 Note that std::condition_variable::wait also has an overload that accepts a predicate. So instead of while(the_condition == 0) cv.wait(lock); you could instead write cv.wait(lock, [&]{ return the_condition != 0; }); — Andrew Durward Apr 9 '13 at 21:16

@AndrewDurward: That's sweet! I didn't know that. - Wandering Logic Apr 9 '13 at 21:18

Ah yes, of course. Ridiculously simple. Just lock the same mutex that the condition variable uses, when changing the condition to actually be true in thread B. I have implemented your suggestion in my code and it solves the problem. Thank you! - Philip Bennefall Apr 9 '13 at 21:59

add a comment