How do I choose between the strong and weak versions of compare-exchange?

Raymond C

March 30th, 2018

Last time, we left with the question of when you should prefer the strong version of compare-exchange as opposed to the weak version.

It comes down to whether spurious failures are acceptable and how expensive they are.

In the example given in the presentation, the cost of a spurious failure is very low:

```
do { new_n->next = old_h; }
while (!head.compare_exchange_strong(old_h, new_n));
```

Recovering from a spurious failure is just updating a single variable and retrying the operation. Removing the nested loop embedded in the strong compare-exchange simplifies the outer loop.

On the other hand, if recovering from the failure requires a lot of work, such as throwing away an object and constructing a new one, then you probably want to pay for the extra retries inside the strong compare-exchange operation in order to avoid an expensive recovery iteration.

And of course if there is no iteration at all, then a spurious failure could be fatal. Consider the lock-free singleton construction pattern:

```
std::atomic<Widget*> cachedWidget;

Widget* GetSingletonWidget()
{
 Widget* widget = cachedWidget;
 if (!widget) {
  widget = new(std::nothrow) Widget();
  if (widget) {
   Widget* previousWidget = nullptr;
   if (!cachedWidget.compare_exchange_strong(previousWidget,
widget)) {
    // lost the race - destroy the redundant widget
    delete widget;
    widget = previousWidget;
   }
  }
 }
 return widget;
}
```

If we were to switch to `compare_exchange_weak`, then a spurious failure would mean that the value of `cachedWidget` was `nullptr`, but we failed to exchange anyway. This means that we would think that we lost the race against another thread and return the `previousWidget` as the singleton. But in the case of a spurious failure, the `previousWidget` will still be `nullptr`, causing the code to create a Widget, think it was redundant, throw away the created Widget, and then return `nullptr`. This is bad news for the `GetSingletonWidget` function.

Choosing between the strong and weak versions of compare-exchange requires you to understand what your algorithm does in the case of a spurious failure.

Raymond Chen

Follow

Tagged Code

Read next

The MIPS R4000, part 1: Introduction

Here we go again.

Raymond C

April 2, 2018

0 comment

The MIPS R4000, part 2: 32-bit integer calculations

The usual suspects.

Raymond C

April 3, 2018

0 comment

0 comments

Comments are closed. Login to edit/delete your existing comments

Archive

Relevant Links

Categories