Analyzing Multithreaded Programs [closed]

Asked 16 years, 4 months ago Modified 11 years ago Viewed 3k times

7

Closed. This question does not meet Stack Overflow guidelines. It is not currently accepting answers.

Questions asking us to **recommend or find a tool, library or favorite off-site resource** are off-topic for Stack Overflow as they tend to attract opinionated answers and spam. Instead, describe the problem and what has been done so far to solve it.

Closed 11 years ago.

Improve this question

We have a codebase that is several years old, and all the original developers are long gone. It uses many, many threads, but with no apparent design or common architectural principles. Every developer had his own style of multithreaded programming, so some threads communicate with one another using queues, some lock data with mutexes, some lock with semaphores, some use operating-system IPC mechanisms for intra-process communications. There is no design documentation, and comments are sparse. It's a mess, and it seems that whenever we try to refactor the code or add new functionality, we introduce deadlocks or other problems.

So, does anyone know of any tools or techniques that would help to analyze and document all the interactions between threads? FWIW, the codebase is C++ on Linux, but I'd be interested to hear about tools for other environments.

Update

I appreciate the responses received so far, but I was hoping for something more sophisticated or systematic than advice that is essentially "add log messages, figure out what's going on, and fix it." There are lots of tools out there for analyzing and documenting control-flow in single-threaded programs; is there nothing available for multi-threaded programs?

See also Debugging multithreaded applications

multithreading concurrency

Share

Improve this question

Follow

edited May 23, 2017 at 12:33

Community Bot

1 ● 1

asked Aug 13, 2008 at 15:09

Kristopher Johnson

82.4k ● 55 ● 251 ● 306

7 Answers

Sorted by: Highest score (default)

6

Invest in a copy of Intel's [VTune](#) and its thread profiling tools. It will give you both a system and a source level view of the thread behaviour. It's certainly not going to autodocument the thing for you, but should be a real help in at least visualising what is happening in different circumstances.

I think there is a trial version that you can download, so may be worth giving that a go. I've only used the Windows version, but looking at the VTune webpage it also has a Linux version.

Share Improve this answer Follow

answered Aug 21, 2008 at 18:38

Greg Whitfield
5,719 ● 2 ● 31 ● 32

I took VTune for a spin... and I find that I prefer Apple's Shark profiler better as I find the interface and output for intuitive and straight forward. My 2 cents. – oz10 Dec 2, 2008 at 0:37

4

As a starting point, I'd be tempted to add tracing log messages at strategic points within your application. This will allow you to analyse how your threads are interacting with no danger that the act of observing the threads will change their behaviour (as could be the case with step-by-step debugging). My experience is with the .NET platform and my favoured logging tool would be log4net since it's free, has extensive configuration options and, if you're sensible in how you implement your logging, it won't noticeably hinder your application's performance. Alternatively, there is .NET's built in Debug (or Trace) class in the System.Diagnostics namespace.

Share Improve this answer Follow

answered Aug 13, 2008 at 15:33

Wheelie
3,906 ● 2 ● 35 ● 39

Use Sysinternal's free DebugView program in conjunction with System.Diagnostics.Debug/Trace, it's a great way to get debug info from your application, and it's a lot nicer than Visual Studio's output window. – Rob Sep 22, 2008 at 2:41

I just say: [TestFixtureSetUp] public void ConfLog4Net() { log4net.Config.BasicConfigurator.Configure(new TraceAppender { Layout = new PatternLayout("[%t] %m%newline"), Threshold = Level.Info }); } – Henrik Nov 29, 2009 at 22:20

3

I'd focus on the shared memory locks first (the mutexes and semaphores) as they are most likely to cause issues. Look at which state is being protected by locks and then determine which state is under the protection of several locks. This will give you a sense of potential conflicts. Look at situations where code that holds a lock calls out to methods (don't forget virtual methods). Try to eliminate these calls where possible (by reducing the time the lock is held).

Given the list of mutexes that are held and a rough idea of the state that they protect, assign a locking order (i.e., mutex A should always be taken before mutex B). Try to enforce this in the code.

See if you can combine several locks into one if concurrency won't be adversely affected. For example, if mutex A and B seem like they might have deadlocks and an ordering scheme is not easily done, combine them to one lock initially.

It's not going to be easy but I'm for simplifying the code at the expense of concurrency to get a handle of the problem.

Share Improve this answer Follow

answered Aug 13, 2008 at 20:41

denis phillips
12.7k ● 5 ● 34 ● 47

2

This a really hard problem for automated tools. You might want to look into model checking your code. Don't expect magical results: model checkers are very limited in the amount of code and the number of threads they can effectively check.

A tool that might work for you is CHESS (although it is unfortunately Windows-only). BLAST is another fairly powerful tool, but is very difficult to use and may not handle C++. Wikipedia also lists StEAM, which I haven't heard of before, but sounds like it might work for you:

> StEAM is a model checker for C++. It detects deadlocks, segmentation faults, out of range variables and non-terminating loops.

Alternatively, it would probably help a lot to try to converge the code towards a small number of well-defined (and, preferably, high-level) synchronization schemes. Mixing locks, semaphores, and monitors in the same code base is asking for trouble.

Share Improve this answer Follow

answered Aug 21, 2008 at 19:11

Chris Conway
56k ● 43 ● 131 ● 155

1

One thing to keep in mind with using log4net or similar tool is that they change the timing of the application and can often hide the underlying race conditions. We had some poorly written code to debug and introduced logging and this actually removed race conditions and deadlocks (or greatly reduced their frequency).

Share Improve this answer Follow

answered Aug 13, 2008 at 20:27

denis phillips
12.7k ● 5 ● 34 ● 47

1

In Java, you have choices like FindBugs (for static bytecode analysis) to find certain kinds of inconsistent synchronization, or the many dynamic thread analyzers from companies like Coverity, JProbe, OptimizeIt, etc.

Share Improve this answer

Follow

answered Sep 22, 2008 at 2:19

Alex Miller

70.1k ● 25 ● 124 ● 168

1

Can't UML help you here ?

If you reverse-engineer your codebase into UML, then you should be able to draw class diagrams that shows the relationships between your classes. Starting from the classes whose methods are the thread entry points, you could see which thread uses which class. Based on my experience with Rational Rose, this could be achieved using drag-and-drop ; if no relationship between the added class and the previous ones, then the added class is not directly used by the thread that started with the method you began the diagram with. This should gives you hints towards the role of each threads.

This will also show the "data objects" that are shared and the objects that are thread-specific.

If you draw a big class diagram and remove all the "data objects", then you should be able to layout that diagram as clouds, each clouds being a thread - or a group of threads, unless the coupling and cohesion of the code base is awful.

This will only gives you one portion of the puzzle, but it could be helpful ; I just hope your codebase is not too muddy or too "procedural", in which case ...

Share Improve this answer

Follow

edited Sep 26, 2008 at 17:37

answered Sep 26, 2008 at 14:25

philant

35.7k ● 11 ● 73 ● 113