A Threading Snarl

- Michael Stahl, Red Hat, Inc.
- 2014-09-04

Overview

1. Introduction
2. Threading Architecture
3. Common Anti-Patterns
4. Threading Constraints

Introduction

- Why a talk about threading?
  - years ago somebody at Sun had the glorious idea to test OOo by remote UNO connection
  - JUnitTests still provide important test coverage of UNO API today
  - what could possibly go wrong?

Issues with Multi-Threading

- too much code needs to be **thread-aware**
  - VCL
  - every method of every UNO component
- lots of fragile, un-testable locking all over the place that doesn't actually work
  - locking invariants usually undocumented
  - additional complexity
  - performance impact of per-method locking / atomic ref-counting (200k `osl_acquireMutex` calls on start-up)
- very little actual scalability is achieved
  - from the UI, most things happen on main thread (reliable!)
- … but remote UNO connections quite unreliable

Threading Architecture(s)

- 2 threading architectures:
- VCL: originally single-threaded => big global lock (SolarMutex/SalYieldMutex)
  - VCL does not want to be a **thread-safe** UI toolkit, but is **thread-aware**
    - no single event handling / “GUI” thread
  - "Multithreaded toolkits: A failed dream?" – Graham Hamilton
- UNO: fine-grained per-component locking
  - UNO components have to be thread-safe
    - similar to COM "multi-threaded apartment" model
- inherent conflict is often resolved by using SolarMutex to lock UNO components

Common Anti-Patterns: Missing Lock

- race due to forgetting to lock mutex
  - happens surprisingly often
  - every UNO method implementation needs a lock
- forgetting to lock mutex in / around C++ destructor
  - esp. in applications where dtor un-registers in core model
  - make sure member / superclass destruction is also covered!
    - `sw::UnoImplPtr`

Common Anti-Patterns: Deadlock

- AB-BA deadlock of 2 threads between 2 mutexes {A,B}
  - Thread 1 locks mutex A
  - Thread 2 locks mutex B
  - Thread 1 tries to lock mutex B and sleeps
  - Thread 2 tries to lock mutex A and sleeps

Example:

```
void SomeComp::foo()
{
  MutexGuard g;
  ...
  callEventListeners();
}
```

- need to unlock `MutexGuard` before calling out!
  - [in practice, cannot unlock SolarMutex...]

Common Anti-Patterns: Deadlock Via Recursive Mutex

- `osl::Mutex` is recursive, so instead of trivial self-deadlocks we get very subtle deadlocks!

"A correct and well understood design does not require recursive mutexes."

– David Butenhof

```
void SomeComp::foo() {
  {
    MutexGuard g;
    ...
  }
  //don't call with lock
  callEventListeners();
}
void SomeComp::bar() {
  MutexGuard g;
  ...
  foo(); // oops!
}
```

Common Anti-Patterns: Racy Reference Counting

- The `uno::Reference` uses thread-safe atomic instructions
- But:
  careful when converting C++ pointer to `uno::Reference`!
  - valid if newly created (ref-count == 0)
  - valid if thread already owns a `uno::Reference` to it
  - in all other cases: use `uno::WeakReference` for thread safety!
- for examples see i#105557, fdo#72695

Common Anti-Patterns: Thread Not Joined

- A thread is spawned without any protocol for its lifetime
- keeps running during shutdown...
  - accesses objects that are already deleted by exit handlers...

UNO Bridges & Bindings (1)

- UNO remote bridges (URP): reader / writer threads
- Thread-Affine UNO-UNO purpose bridge: 2 threads
- Java JNI and URP bridges:
  - finalizers - typically run in separate finalizer thread [implementation dependent], call `XInterface::release()`
  - currently `AsynchronousFinalizer` actually moves the finalizer to yet another thread… [both bridges]
- CLI bridge (cli_ure):
  - finalizers may be called on separate thread and call `XInterface::release()`

UNO Bridges & Bindings (2)

- Python:
  - famous "Global Interpreter Lock" ... should not cause deadlocks, as it is dropped before calling UNO methods
  - PyUNO finalizer thread
- C++/Java/CLI/Python extensions can spawn threads
- BASIC:
  - inherently single-threaded, runtime calls `Reschedule()` periodically
- OLE Automation: wraps COM object around UNO object or the other way, seems to have no obvious threading issue

VCL

- Main thread is running event loop, and always holds SolarMutex except when event loop calls `Yield()`
- dialogs are executed → `Yield()` → SolarMutex released!
  - [important if the dialog spawns worker threads...]
- `SolarMutexReleaser` – scary...
- Application::Reschedule() – internal API to release SolarMutex
- XToolkit::reschedule() – public UNO API to release SolarMutex
  - [actually called by some bundled extensions]
- `com.sun.star.awt.AsyncCallback` service allows moving work to main thread from remote UNO
  - can work around some threading bugs

Java UI Toolkits

- Swing UI (could be used by extensions):
  - (mostly) not thread-safe, all events are delivered to one event handling thread [which is not the main thread!]
  - if a Swing event handler calls some UNO method it will happen on separate event handling thread
- SWT UI (dito):
  - no idea, hope nobody is using that in extension

Unix: GTK+ / Qt

- GTK+ **thread-aware** (`gdk_thread_enter/leave`)
  - SolarMutex hooked into GTK+, `GDK_THREADS_MUTEX`
  - guarantee Gtk+ and VCL have same idea whether mutex is locked, for code that calls into Gdk/Gtk+ w/o VCL being involved
  - (although some Gtk related libs may release the mutex at unfortunate times…)
  - https://developer.gnome.org/gdk3/stable/gdk3-Threads.html
- Qt single threaded – all event handling/UI on main thread
  - how does that work? – badly! can't use KDE dialogs unless glib main loop integration allows foisting SolarMutex on Qt with `g_main_context_set_poll_func`
  - http://qt-project.org/doc/qt-4.8/thread-basics.html

Mac OS X

- Cocoa is (mostly) not thread-safe
  - ... except some low-level classes (once you spawn a `NSThread`)
- events get delivered to main thread
- `NSView`'s "graphic states" and `NSGraphicsContext` are **thread-affine**
  - `NSView` mostly restricted to main thread
- https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/Multithreading/ThreadSafetySummary/ThreadSafetySummary.html

Win32

- COM: main thread in STA, other threads in MTA (`oslCreateThread`)
- COM STA components (clipboard, drag&drop, file picker) apparently require running in separate thread
- DDE is **thread-affine**
  - everything happens on thread calling `DdeInitialize`
  - and via Window messages
- Win32 Windows are **thread-affine**, which is a real problem...
  - construction, destruction, events all on same thread
  - VCL has to create all Windows on main thread
    - which cannot actually work currently...

Win32 VCL Window Deadlock

```
void
pseudo-win32-message-loop()
{
  SolarMutexReleaser r;
  while (msg=GetMessage()) {
    switch (msg) {
      case FOO:
      {
        SolarMutexGuard g;
        ...
      }
      case SAL_MSG_CREATEFRAME:
      ... // no mutex needed
    }
  }
}
```

```
void SomeUNOcomponent
     ::makeMeAView()
{
  SolarMutexGuard g;
  Window *w = new Window;
}

Window::Window()
{
  m_pSalFrame = (SalFrame*)
    SendMessage(
      SAL_MSG_CREATEFRAME);
  // <- deadlock here
}
```

"I'm worn, tired of my mind
I'm worn out, thinking of why
I'm always so unsure"
– Portishead, "Threads"

- Thanks for listening