Async Signal Safe Functions vs Dyld Lazy Binding

This post describes a gotcha when calling async signal safe functions. This issue mostly comes up in the context of crash reporters (see Implementing Your Own Crash Reporter) but I've split it out into its own post because:

- It can be relevant in other situations.
- I didn't want to further increase the size of the above-mentioned post, which is already quite large.

The executive summary here is that, when you rely on async signal safe functions, you must be careful about dyld lazy binding. A corollary is that **implementing your own crash reporter is much harder than you might think**. The point covered here is just one gotcha of many; see Implementing Your Own Crash Reporter for a more comprehensive list.

IMPORTANT This post discusses implementation details of the dynamic linker, dyld. That implementation has changed many times in the past, and is likely to change again in the future. In fact, it's within the bounds of possibility that dyld might change to eliminate lazy binding entirely, and thus completely obviate this problem. If you'd like to check on the current state of affairs, or have a follow-up question about anything I've raised here, please start a new thread with the *Debugging* tag.

Share and Enjoy
—
Quinn "The Eskimo!" @ Developer Technical Support @ Apple
`let myEmail = "eskimo" + "1" + "@" + "apple.com"`

The Problem

Consider the following code:

```
extern pid_t gSignalPID;

static void mySigAlarmHandler(int sigNum) {
    gSignalPID = getpgrp();
}

static void installSigAlarmHandler(void) {
    (void) signal(SIGALRM, mySigAlarmHandler);
}
```

This *should* be safe. `SIGALRM` is an async signal, but the signal handler only calls `getpgrp`, which is documented to be async signal safe.

However, if you step through this routine at the assembly language level, you'll find that things are not so simple. The `installSigAlarmHandler` function does not trap into the kernel directly. Rather, the actual kernel trap is in a `getpgrp` function that's implemented in the System framework (aka `libSystem`). The program containing `installSigAlarmHandler` can't call this `getpgrp` function directly. Instead it calls a stub function that was inserted into the program at link time. And that stub function doesn't call `getpgrp` directly either. Rather, it bounces through some layers of indirection that eventually call through to a dyld helper function, `dyld_stub_binder`. That function looks up the `getpgrp` symbol in System framework, patches the stub to call that function directly for future invocations, and then calls through to `getpgrp` for this invocation. This whole process is known as **lazy binding**.

This raises the question, is `dyld_stub_binder` async signal safe? It turns out this is a hard question to answer. As you might expect for a complex program that supports multi-threading, dyld uses a lock to protect its internal data structures. The presence of this lock suggests that `dyld_stub_binder` might not be async signal safe. This lock is currently a recursive lock, so you're unlikely to deadlock — which is the failure mode you'll see, for example, when you call `malloc` or the Objective-C runtime from this context — but there's a chance that the async signal might be delivered at a point where dyld's data structures are inconsistent. That would be bad.

Furthermore, there are two cases where the fact that this is a recursive lock won't protect you from deadlocks:

- When you suspend a thread (using `thread_suspend`) that's in an unknown state.
- When implementing your own crash reporter.

The first case is relatively rare, but I do see it from time to time. For example, some folks have code that periodically backtraces the main thread to look for excessive main thread latency, and they suspend the main thread while doing the backtrace to guarantee consistent results.

When you suspend a thread in an unknown state, there's a possibility that this thread is inside `dyld_stub_binder`. If so, it might be holding dyld's recursive lock. If your thread then calls an async signal safe function, it may end up calling `dyld_stub_binder` itself, which will deadlock.

This is a particular concern when implementing your own crash reporter because in that case your signal handler is expected to suspend all threads except the current one to reduce the possibility of it being called by multiple threads. Implementing Your Own Crash Reporter discusses that issue in more depth.

The Solution

The solution here is to carefully audit any code on the async signal path to ensure that:

- It only calls async signal safe functions.
- Those functions were called elsewhere in your program before the signal handler was installed.

IMPORTANT In the above, *program* refers to the Mach-O image containing your signal handler. Remember that each Mach-O image gets its own dyld stub functions, and thus calling an async signal safe function in Mach-O image A won't bind the stub in Mach-O image B. For example, if your app calls `-[NSFileHandler readDataOfLength:]`, which calls `read` internally, that won't bind your app's `read` stub but rather the stub within the Foundation shared library.

Note I suspect that Foundation's call to `read` doesn't use lazy binding because Foundation exists within the dyld shared cache, but my point is still valid: The fact that Foundation has called `read` doesn't bind *your* `read` stub.

Change History

- 2019-05-14 — First posted.
- 2019-05-15 — Clarified that the actual implementation of `getpgrp` is in the System framework. Minor editorial changes.
- 2021-02-27 — Fixed the formatting. Minor editorial changes.

Debugging

Asked 1 year ago by eskimo

Reply to this question