PRELIMINARY my-draft-colorrgb0.85,0.85,0.85 my-draft-color

Actors: A Message Passing Single-Thread Semantics for SMP Environments

David McClain, Refined Audiometrics Laboratory, LLC
1st Draft

February 17, 2018

Contents

1 Introduction to Actors

The Actors system provides for an indefinite number of message handlers in SMP environments with limited threading resources. The only limit imposed on the number of Actors is the size of virtual memory; typically 10's of millions of Actors can be deployed. Each Actor message handler offers single-thread semantics to the programmer, for mutable local state. Every Actor has a private message queue to which other code can send messages which prompt some action by the Actor. The format of messages is entirely open, and depends on the programmer's needs and conventions.

SMP systems offer multi-core CPU's with multi-threading, each processing core executing code for one thread at a time, all cores operating in parallel. This imparts complexities to designs, especially when mutable state must be shared among threads. Actors help to simplify programming by keeping much of the mutable state private to each Actor, and ensuring that any one Actor will only execute on one CPU core at a time.

Real computers impose limits on the number of threads that can be spawned, typically in the range of 1-2 thousand threads. With Actors, we can perform with many times more active message handlers, all multiplexed on a limited pool of Executive threads, the pool ranging in size from 1 thread to as many as you like, but usually not more than the number of processor cores.

It is generally quite impossible to predict when calling some function will cause a thread to block, waiting for some event. By the same measure, it is generally impossible to know if any called function will return quickly or not, which could be seen as a form of blocking. The *Halting Problem* shows the impossibility of predicting behavior in general.

Hence, sometimes Actors will hog a thread or block it from continuing. Other threads will pick up and run on the same and other cores of the CPU. If every Executive thread running Actor handlers ends up blocking, then the Actor system will throw up a restart dialog asking if you want to create an additional Executive thread, kill an Executive thread that might be hanging, or just wait a bit longer in hopes that the blocking eases with more time.

The Actors system enforces single thread semantics by disallowing the Actor's message handling code from executing on more than one Executive thread at a time. Conversely, the number of disparate Actors that can be running in parallel is limited only by the size of the Executive Pool and the number of CPU cores.

When Actor code knows that it could block waiting, there are several approaches that can be used to avoid physically blocking the Executive thread. Generally, these all become variants of callback continuations that can be called when the blocking activity finishes, either by directly invoking the continuations, or by sending a message to the Actor.

The Actor system implements a form of *Erlang* Selective Receive (`RECV`) that sets up a transient selective message responder as its continuation. Incoming messages not matching one of the selective cases are stashed in order of arrival for later processing after a `RECV` has been satisfied or it times out.

Other macros have been defined as a simple extension to the `=BIND` system outlined by Paul Graham in his *"On Lisp"* book, in the later chapters discussing nondeterminism. In using these macros the Actor does not perform a `RECV` operation, but rather depends on the blocking code to restart the Actor through a captured continuation. But in order to enforce single-thread semantics inside the Actor code such continuations are themselves frequently translated into messages sent to the Actor.

Actors are represented as instances of a CLOS class with an associated code pointer pointing at the message handling code. These objects also contain the private message queue, a busy bit, an SMP-safe property list, and support slots for `RECV` when active.

When a message arrives at an Actor's message queue, the actor is placed onto the ready queue, if not already enqueued nor active, for execution by any Executive thread. Once started running, the Actor takes messages from its message queue and performs the indicated actions, repeatedly, until the message queue becomes empty. Afterward, the Actor is marked not-busy, and retired.

It is important to remember that Actors are not Threads. And hence, an Actor cannot be killed. If you kill a running Actor thread, you are really killing the underlying Executive thread. Actors are just CLOS objects with associated functional behavior. You can't kill an Actor, any more than you can kill any other Function. You can disable an Actor from future activity by simply not sending it any more messages. But if you happen to kill off all the Executive threads, a new pool will be regenerated at the next message sent to an Actor. No special action is needed to restore the Executive thread pool.

Actors can change their behavior at any time by performing a `BECOME`, which physically alters the code pointer stored in the Actor's CLOS object. It's old pointer value is returned in case you need to later restore the original behavior.

To preserve and respect the single-thread semantics provided by Actors, outside code should refrain from altering anything in the CLOS object, or any other mutable state, belonging to the Actor, except for the contents of the property list. All other interactions with Actors should be specified by `SEND` messages.