JActor2 Revisited

JActor2 is a robust and high-performance alternative to threads and locks.

Jactor2 Revisited focuses on a subset of the API that is easy to learn but reasonably comprehensive.

By Bill la Forge, 2014

Issues with Threads & Locks

- Reworking software to take advantage of many-core computers is difficult.
- High-speed memory cache typically must be reloaded after a thread is unblocked, and this slows things down significantly.
- Testing provides no assurance that production code contains no race conditions or potential deadlocks.

JActor2 Model: Message Passing

- Messages are passed asynchronously between reactors, which process them one at a time.
- Message processing is done by blades. Every Blade has an associated reactor.
- When a blade variable is private and only updated when a message is processed, race conditions will not occur.
- The use of reactors often improves vertical scaling, as any number of threads can be used to good effect.

The JActor2 Model: Message Types

- The three types of messages are requests, responses and signals.
- When a request is sent, either a response or an exception is always returned. (Exception handlers are used to catch exceptions, which otherwise recursively propagate to the source of the request.) Programing then is similar to OO.
- Signals are used to pass events. If an exception occurs when processing, the exception is logged.

JActor2 Model: Isolation of Requests

- Once processing has begun for a request, no other request will be processed until a response is returned.
- Processing a request often means sending requests to other reactors and processing their responses.
- Responses (and Signals) then are processed in the order they are received, while subsequent requests are held in a separate queue.
- There is no interaction between requests sent to the same reactor—they are isolated.

JActor2 Model: The 4 States of a Reactor

1. Idle – A reactor has no messages to process.
2. Pending – A reactor has not completed the current request but there are no responses or signals to process. There may however be pending requests.
3. Ready – Either the reactor has completed the last request and has requests that it can process or the reactor has responses or signals to process.
4. Active – The reactor is currently processing a message.

JActor2 Model: The Work Queue

- When a message is passed to a reactor and the reactor is not active, the reactor is added to a work queue.
- A pool of threads all try to read reactors from the same work queue. When read, if the reactor is already active then the thread just reads another reactor.
- The reactor's messages are processed until the reactor becomes idle or pending, after which the thread reads the next reactor.

JAactor2 Model: Message Buffering

- Requests and responses are not sent immediately, but grouped into buffers and passed only after the current message is processed. This lowers the cost of message passing.
- When the last buffer is sent, if the destination reactor is not active, the current reactor is added to the work queue and the destination reactor is processed instead for an improvement in overall performance.
- Signals are not buffered, but passed immediately to their target reactor.

Deadlocks

- When two threads each hold a lock and try to acquire the lock held by the other thread, you get a deadlock.
- Similarly you can have two reactors which send requests to each other but will not process any subsequent requests until they get a response from the reactor.
- The same happens with actors, only nobody calls them deadlocks.

Partial Ordering

- Deadlocks are avoided by observing a partial ordering. So for a given set of locks, they will always be acquired in the same order.
- Partial orderings are addressed when designing the software. Problems arise when maintaining a large program over an extended period of time. Locking order documents may not be maintained, may not be consulted for every change or may not even exist.
- Testing doesn't help either, as deadlocks may occur infrequently.

JActor2 Model: Partial Ordering

- If reactors never send requests to other reactors which have sent them a request, even indirectly, then there will be no deadlocks.
- This partial ordering of which reactors send to which other reactors is tracked at runtime. And any attempt to send a request which violates the partial ordering observed to date raises a runtime exception.
- System tests with reasonable coverage will now detect partial ordering failures, adding significantly to overall robustness.

https://github.com/laforge49/JActor2