ActressMas Framework

When one task wants to communicate with another, it sends it a message, rather than contacting it directly. The messages are put on a queue, and the receiving task (known as an "actor" or "agent") pulls the messages off the queue one at a time to process them.

This message-based approach has been applied to many situations, from low-level network sockets (built on TCP/IP) to enterprise wide application integration systems (for example MSMQ or IBM WebSphere MQ).

From a software design point of view, a message-based approach has a number of benefits:

- You can manage shared data and resources without locks.
- You can easily follow the "single responsibility principle", because each agent can be designed to do only one thing.
- It encourages a "pipeline" model of programming with "producers" sending messages to decoupled "consumers", which has additional benefits:
- The queue acts as a buffer, eliminating waiting on the client side.
- It is straightforward to scale up one side or the other of the queue as needed in order to maximize throughput.
- Errors can be handled gracefully, because the decoupling means that agents can be created and destroyed without affecting their clients.

From a practical developer's point of view, when writing the code for any given actor, you don't have to hurt your brain by thinking about concurrency. The message queue forces a "serialization" of operations that otherwise might occur concurrently. And this in turn makes it much easier to think about (and write code for) the logic for processing a message, because you can be sure that your code will be isolated from other events that might interrupt your flow.

With these advantages, it is not surprising that when a team inside Ericsson wanted to design a programming language for writing highly-concurrent telephony applications, they created one with a message-based approach, namely Erlang. Erlang has now become the poster child for the whole topic, and has created a lot of interest in implementing the same approach in other languages.

F# has a built-in agent class called MailboxProcessor. These agents are very lightweight compared with threads - you can instantiate tens of thousands of them at the same time.

These are similar to the agents in Erlang, but unlike the Erlang ones, they do not work across process boundaries, only in the same process. And unlike a heavyweight queueing system such as MSMQ, the messages are not persistent. If your app crashes, the messages are lost.

The agent encapsulates a message queue that supports multiple-writers and a single reader agent. Writers send messages to the agent by using the Post method and its variations. The agent may wait for messages using the Receive or TryReceive methods or scan through all available messages using the Scan or TryScan method.

Actress is a C# port of the F# MailboxProcessor.

Actress has been extended to *ActressMas* to facilitate the development of multiagent systems.

References

Fsharpforfunandprofit, *Messages and Agents*, https://fsharpforfunandprofit.com/posts/concurrency-actor-model, 2010.

D. Delimarsky, *Control.MailboxProcessor<'Msg> Class (F#)*, Micrososft, https://msdn.microsoft.com/en-us/visualfsharpdocs/conceptual/control.mailboxprocessor%5B%27msg%5D-class-%5Bfsharp%5D, 2017.

K. Thompson, *Actress*, https://github.com/kthompson/Actress, 2017.

F. Leon, *ActressMas*, https://github.com/florinleon/ActressMas, 2018.