.UBAfiuba
FACULTAD DE INGENIERÍA

Técnicas de Programación Concurrente I

Modelo de Actores

Ing. Pablo A. Deymonnaz
pdeymon@fi.uba.ar

Facultad de Ingeniería
Universidad de Buenos Aires

.UBAfiuba FACULTAD DE INGENIERÍA

Índice

Modelo de Actores

- Similar al modelo de pasaje de mensajes.
- Desarrollado por Carl Hewitt en 1973.
- Popularizado por el lenguaje Erlang.

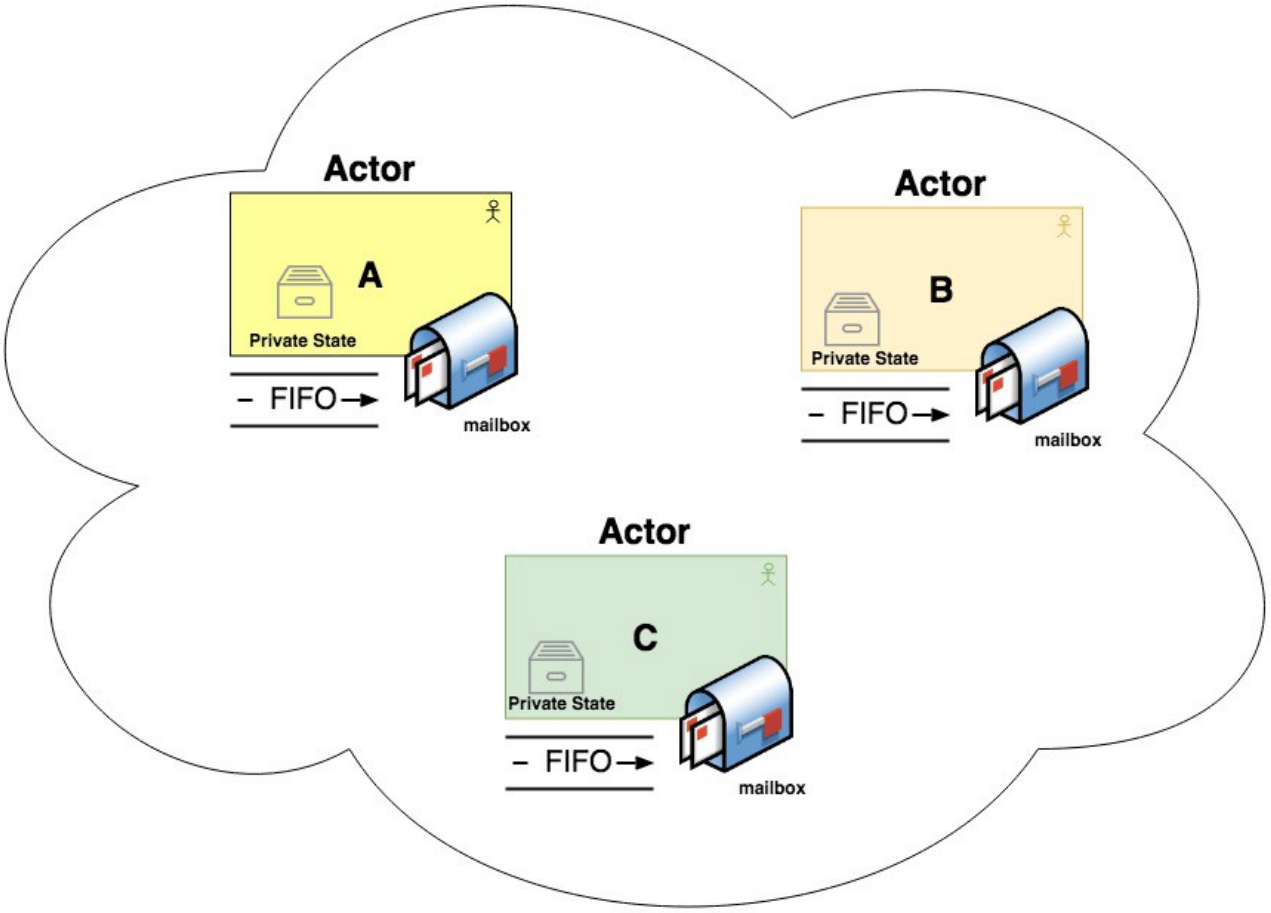
Actor

El actor es la primitiva principal del modelo. Son livianos, se pueden crear miles (en lugar de threads).
Encapsulan comportamiento y estado.
Compuesto por:

- **Dirección:** adonde enviarle mensajes.
- **Casilla de correo (mailbox):** un FIFO de los últimos mensajes recibidos.

El actor supervisor puede crear otros actores hijo.

Actor


Actor
A
Private State
FIFO
mailbox
Actor
B
Private State
FIFO
mailbox
Actor
C
Private State
FIFO
mailbox

Actor

- Son aislados de otros actores: no comparten memoria.
- El estado privado solo puede cambiarse a partir de procesar mensajes.
- Pueden manejar un mensaje por vez.
- En un sistema distribuido, la dirección del actor puede ser una dirección remota.

Mensajes

- Los actores se comunican entre ellos solamente usando mensajes.
- Los mensajes son procesados por los actores de forma asincrónica.
- Los mensajes son estructuras simples inmutables.

.UBAfiuba FACULTAD DE INGENIERÍA

Índice

Framework Actix

- Usa **tokio** y futures como runtime de sustento. Se ejecutan dentro del Sistema de Actores.
- El núcleo es el tipo *Arbitrer*: un thread que crea un event loop por debajo y provee un handler.
- Cada actor se ejecuta dentro de un arbitrer.
- El handler se usa para enviar mensajes al actor.
- Se ejecutan en un contexto de ejecución *Context<A>*, separado para cada uno.

Crear un actor

- Crear un tipo de dato. Debe implementar el trait **Actor**.
- Definir un mensaje e implementar el handler para ese tipo del actor: **Handler<M>** Los mensajes pueden ser manejados de forma asincrónica.
- Crear el actor y hacer spawn en uno de los árbitros.

Ciclo de vida del actor

- **Iniciado (Started):** con el método *started()* (implementación default en el trait Actor). El contexto del actor está disponible.
- **En ejecución (Running):** es el estado siguiente a la ejecución de started(). Puede estar en este estado de forma indefinida.
- **Parando (Stopping):** puede pasar a este este estado en las siguiente situaciones:
  - Llamando **Context::stop** en el mismo actor.
  - ningún otro actor lo referencia
  - no hay objetos registrados en el contexto
- **Detenido (Stopped):** desde el estado anterior no modificó su situación. Es el último estado de ejecución.

Dirección

Los actores son referenciados únicamente por la dirección.

```
struct MyActor;
impl Actor for MyActor {
  type Context = Context<Self>;
}

let addr = MyActor.start();
```

Mensaje

Un actor se comunica con otro enviando mensajes.
Todos los mensajes son tipados, deben implementar el trait *Message*.
Message::Result define el tipo de retorno.

```
impl Message for Ping {
type Result = Result<bool, std::io::Error>;
}
```

Enviar mensaje

Varias formas:

- **Addr::do_send(M):** ignora errores en el envío del mensaje. Si la casilla de mensajes está cerrada, se descarta. No retorna el resultado.
- **Addr::try_send(M):** trata de enviar el mensaje inmediatamente. Si la casilla de mensajes está llena o cerrada, retorna *SendError*
- **Addr::send(M):** retorna un objeto *future* que resultado del proceso de manejo del mensaje.

Contexto

Los actores mantienen el contecto interno de ejecución, o estado.
Permite al actor determinar su dirección, cambiar los límites de la casilla de mensajes, o detenerse.
Los mensajes llegan a la casilla primero, luego el contexto de ejecución llama al handler específico.

Arbiter

Provee el contexto de ejecución asincrónica para los actores, funciones y futures.
Alojan el entorno donde se ejecuta el actor.
Realizan varias tareas: crear un nuevo Thread del SO, ejecutar un event loop, crear tareas asincrónicas en ese event loop.

Bibliografía

- **The Complete Rust Programming Reference Guide**, Rahul Sharma, Vesa Kaihlavirta, Claus Matzinger.
- **An Actor-Based Programming System**, Roy J. Byrd, Stephen E. Smith, S. Peter de Jong. IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598
- **Seven Concurrency Models in Seven Weeks: When Threads Unravel (The Pragmatic Programmers)**, Paul Butcher
- Actix: *https://actix.rs/book/actix/*