Технічні науки

УДК 004.852

Зеленін Віктор Анатолійович
бакалавр комп'ютерних наук
Національного технічного університету України
«Київський політехнічний інститут імені Ігоря Сікорського»
Зеленин Виктор Анатольевич
бакалавр компьютерных наук
Национального технического университета Украины
«Киевский политехнический институт имени Игоря Сикорского»
Zelenin Victor
bachelor of computer science of
The National Technical University of Ukraine
«Kyiv Polytechnic Institute»

МОДЕЛЬ АКТОРІВ В ПАРАЛЕЛЬНОМУ ПРОГРАМУВАННІ
МОДЕЛЬ АКТОРОВ В ПАРАЛЛЕЛЬНОМ ПРОГРАММИРОВАНИИ
THE MODEL OF ACTORS INTO CONCURRENT PROGRAMMING

Анотація. В даній роботі розглядаються підходи до розробки паралельних систем на базі акторів. Дана модель спрощує написання багатопоточних програм та надає можливість до відносно простого масштабування системи.

Ключові слова: актор, реактивне програмування, асинхронні повідомлення, Akka.

Аннотация. В данной работе рассматриваются подходы к разработке параллельных систем на базе акторов. Данная модель упрощает написание многопоточных программ и предоставляет возможность к относительно простому масштабированию системы.

Ключевые слова: актор, реактивное программирование, асинхронные сообщения, Akka.

Summary. Thesis considers approaches to the development of parallel systems based on actors paradigm. This model simplifies writing concurrent programs and provide an opportunity to scale system relatively simple.

Key words: actor, reactive programming, asynchronous messages, Akka.

З стрімким розвитком технологій розвивається і модель програмування. На разі застосування паралельних обчислень є актуальним питанням в інформаційних технологіях. Найскладнішим завданням в багатопоточній програмі є організація доступу до спільних змінних ресурсів.

Існує чимало різноманітних підходів до синхронізації доступу до спільних ресурсів, але існує парадигма акторів, яка дозволяє позбавитися спільного стану між конкуруючими потоками. Модель акторів слідує вимогам реактивного програмування, яке заключається в реактивному маніфесті. Сам по собі, реактивний маніфест - це набір правил та домовленостей, які допомагають ефективно розробляти та підтримувати складні інформаційні системи. Щоб певна програма називалася реактивною, вона повинна мати малий час відгуку, бути еластичною та жорсткою, спілкуватися асинхронними повідомленнями.

Актор - це незалежна самостійна обчислювальна одиниця. Актори інкапсулюють стан та поведінку через обмін асинхронними повідомленнями. У відповідь на отримане повідомлення, актор може:

- відправити визначену кількість повідомлень іншим акторам;
- створити певну кількість нових акторів;
- визначити поведінку для опрацювання наступного отриманого повідомлення [1].

Повідомлення надсилаються асинхронно до актора, який в свою чергу повинен десь їх зберігати під час обробки інших повідомлень. Таким сховищем виступає "поштова скринька" актора, окрім того кожен актор має свій власний адрес. Схема взаємодії акторів наведена на рисунку 1.

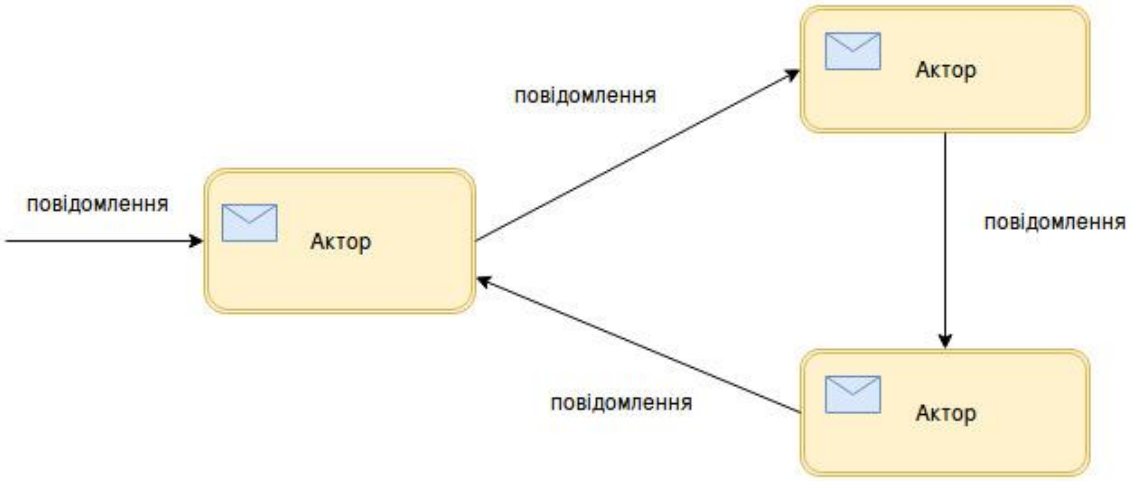

Рисунок 1. Взаємодія акторів через асинхронні повідомлення

Кожен актор може створювати дочірніх акторів та бути їх супервізором. На практиці, значно краще розділити задачу на менші керуємі підзадачі та розв'язати їх паралельно. Якщо актор має важливий стан, який не можна втратити, тоді всю задачу цілком краще віддати на виконання породженим акторам і просто стежити за виконанням задачі. Тому актори ніколи не приходять одні. Вони утворюють певні системи та ієрархії.

Всі актори дотримуються принципу «Let It Crash» і так званого шаблону Error Kernel Pattern (шаблон ядра помилок). Цей шаблон веде нас до системи акторів, де супервізор кожного рівня ієрархії у відповіді за локальні помилки [2]. В даному контексті, помилки - це виключні ситуації, що виникли у підлеглих акторів, тобто самі актори не повинні їх обробляти. У свою чергу, супервізор при виникненні помилки може відловити її і, як правило, продовжити, перезавантажити, зупинити роботу актора або ж викинути виключну ситуацію своєму супервізору, повідомляючи про падіння всієї своєї ієрархії підпорядкування. Ядро помилок самої системи акторів формується з системних акторів, які відповідають за життєвий цикл всієї системи. При цьому кожен рівень ієрархії в організаційній структурі є локальним ядром помилок, які відповідають за стратегію обробки помилок, яка визначить, як потрібно реагувати на певні помилки дочірніх акторів [3].

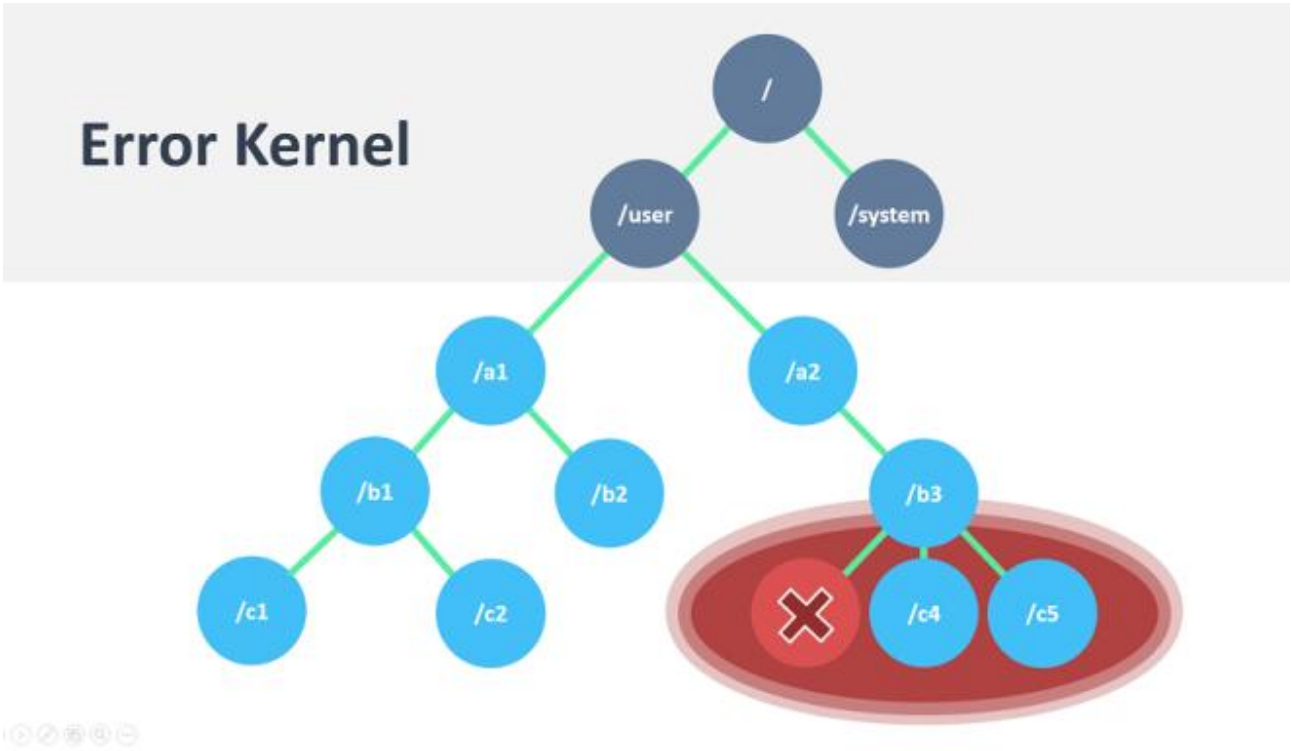

Рисунок 2. Представлення шаблону «Error Kernel»

Існує безліч реалізацій моделі акторів. Найбільш популярними є мови Erlang та Scala, та, в цілому, всі інші функціональні мови програмування, оскільки вони оперують функціями та в них відсутній стан. У контексті будь-якої функціональної мови програмування, актори – це просто функції, які виконуються в обчислювальній системі. Найвідомішою бібліотекою, яка реалізує дану модель є Akka, яка націлена на розділення всієї системи на кластер серверів [4]. Вона також підтримує реактивний маніфест, дозволяючи розгортати додатки на кластері з мінімальними змінами в коді або навіть, взагалі, без змін. Akka не вимагає серверу додатків, достатньо лише JVM, які можна об'єднати в кластер, а також реалізує модель акторів замість звичної об'єктно-орієнтованої.

Прикладом моделювання розподілених систем на базі моделі акторів може бути:

1. Електронна пошта може бути змодельована як система акторів. Клієнти представляються в якості акторів, а адреса електронної пошти – як адреса акторів;

2. Веб-сервіси з кінцевими точками можуть бути змодельовані як адреса певних окремих акторів;
3. Об'єкти з семафорами можуть бути змодельовані як послідовно-паралельний перетворювач, за умови, що їх реалізація така, що повідомлення можуть приходити постійно.

Можна виділити наступні переваги використання даної моделі програмування:

- Простота розробки багатопоточних програм;
- Масштабування. Модель акторів дозволяє створювати велику кількість акторів, кожен з яких відповідає за свою локальну задачу. Відсутність стану та асинхронний обмін повідомленнями надає можливість створювати розподілені додатки, які можуть бути масштабовані горизонтально;
- Відмовостійкість. Збій одного актора може контролюватися іншими акторами, які виконують відповідні дії для відновлення ситуації (механізм супервізорів);
- Тестування системи;
- Гнучкість.

Модель акторів – це надзвичайно зручний інструмент лише у випадках, коли використання цієї моделі доцільно. Тому слід зауважити на існуючих недоліках даної парадигми:

- Відсутня явна структура системи;
- Відсутня статична перевірка протоколів;
- Складність реалізації віддаленого розгортання та оновлення системи;
- Відсутність ефективних інструментів аналізу продуктивності системи.

В даний час паралелізм є активною областю досліджень, комп'ютери з великим числом ядер нікуди не зникнуть, і швидше за все число ядер в одному кристалі буде експоненціально збільшуватися. Відповідно, паралелізм і раніше буде залишатися областю досліджень та інноваційних рішень.

Література

1. Vernon V. Reactive Messaging Patterns with the Actor Model: Applications and Integration in Scala and Akka/ Vernon V. – Addison-Wesley, 2015 – 837 p.
2. Jesse Russel, Ronald Cuhn. The Actor Model / Jesse Russel, Ronald Cohn – VSD, 2012 – 168 p.
3. Ronald Cuhn. Reactive Design Patterns / Ronald Cuhn/ – Manning, 2017 – 392 p.
4. Raymond Roestenburg. Akka in action / Raymond Roestenburg – Minning, 2016 – 448 p.