CONCURRENCY MODELS AND THEIR MATURITY LEVELS IN JVM WORLD

Kiev 2021

softserve

AGENDA

1. Concurrency and Parallelism
2. Concurrency models
3. Maturity level of concurrency models
4. Process-based concurrency model
5. Thread-based concurrency model
6. Coroutine/Fiber/Actor-based concurrency models
7. Questions

CONCURRENCY AND PARALLELISM

Concurrency is a logical separation of progam structure into separate logic tasks that can exist and be processed at the same time.

Parallelism is the task of running multiple **computations simultaneously.**

CONCURRENCY MODELS IN JVM WORLD

A concurrency model **specifies how execution elements in the the system collaborate to complete the tasks they are are given.**

Maturity level of concurrency model informally defines the efficiency of resource utilization required by the execution elements.

PROCESS-BASED CONCURRENCY MODEL

Child level of maturity, the best choice for simple cloud microservices

Logical tasks are mapped to processes as execution elements. Examples: Apache, Ruby on Rails (partially), Python (partially), Ocaml (partially), Some Cloud-based architectures

Advantages: no shared data, no concurrency problems inside process

Disadvantages: no possibility to write a complex data-handling system, dependency of database or other stateful application, big memory overhead

THREAD-BASED CONCURRENCY MODEL

Teenager level of maturity, the best choice to serve a few complex requests

Logical tasks are mapped to threads as execution elements. Examples: Java threads, Posix threads, Windows threads, OpenMP

Advantages: lower memory consumption (in comparison with process model), better CPU utilization

Disadvantages: data racing, need of synchronization, frequent context switching, limited number of threads, ”possible callback hell”

Thread-based concurrency model can be improved by using **tasks.** Tasks can be executed in thread pools by using Executor (in Java).

Advantages: better resources utilization in cases where no blocking operations used

Disadvantages: same problems with resource synchronization, problems with resources utilization in case of blocking operations, that may cause the increase of threads in thread pool used to execute tasks

FIBER/COROUTINE/ACTOR-BASED CONCURRENCY MODELS

Adult level of maturity, the best choice to serve many requests, for example chat application

Logical tasks are mapped to lightweight special threads as execution elements. Examples: Kotlin coroutines, Scala ZIO fibers, D fibers, Go goroutines, Scala aktors

Advantages: lower memory consumption (in comparison with process model), better CPU utilization, low rate of context switching, lower probability of data racing, possibility to avoid ”callback hell”

Disadvantages: synchronization is still somewhere needed, special transaction handling is required, sometimes problems with debug, sometimes the bridge between normal code and coroutines code is required (Kotlin)

Coroutines and fibers are the logical evolution of tasks. They can be suspended and resumed so there is a possibility to avoid complex callbacks there.

Coroutines and fibers are almost the same except some **minor differences** in design.

Coroutines and fibers models usually uses channels for data exchange. It prevents concurrency issues.

Aktor is a coroutine (fiber) with dedicated channel for retrieving messages. Aktor model is less flexible and more heavy than coroutines-based.

In JVM world the best coroutine implementation is Kotlin coroutines. The best actor implementation is Akka framework.

QUESTIONS