

PV168

Parallelism

Parallelism In Modern Applications

- Approaches:
 - explicit parallelism
 - thread pools
 - functional paradigm
- Code example
 - Available [here](#)

Explicit Parallelism

- Uses synchronization primitives
 - mutex, monitors, locks, atomics
- Low-level approach
- Easy to create a bug
 - Better to avoid

Explicit Parallelism

- What is the difference between mutex and monitor?
 - Mutex is used to control the access to the critical section.
 - Monitor is a combination of mutex and condition variable.
- Condition variable is another synchronization primitive
 - Sends notifications between threads.

Thread Pools

- Separates thread management and the job specification
- Either *run-and-forget*
- ... or retrieve the result via `Future<T>`
- The synchronization is done via thread-safe queue
 - and via the `Future<T>` class when used
- Suitable for *stateful* classes
 - Resembles to *message passing* in OOP


Thread Pools

- Sounds easy, right?

Thread Pools

- One should guarantee at most one operation runs concurrently on the instance.
 - Synchronization must be done also between threads and the *stateful* object.
- It can be achieved by adding a queue to each object.
- Furthermore, the queue in the *Thread Pool* becomes "the queue of queues."

Functional Approach

- Avoids manual thread management
- Useful when data are independent to each other
 - stream-based data processing
 - without side-effects
 -  ➡ not useful for *stateful* objects
- Read-only accesses to *stateful* objects are fine

Other

- fibres/coroutines
- message passing
- processes

Fibres/Coroutines

- Asynchronicity independent to threads
- Virtual function stacks
- I.e., server may have each transaction in a separate fiber.
 - It would not be possible with 1:1 mapping threads to transactions.
 - Remember - threads are just processes, hence large.

Message Passing

- Similar to the *Thread Pool* approach.
- No data sharing between classes.
- Everything passed via messages.
- The motivation is to prepare your environment for the next step

Processes

- Do not create threads, run multiple processes
 - In case of crash, just once instance is down.
 - Better scaling - adhoc management
- Multiprocessor execution
 - may run in different data centers
- The protocol defines the correctness.