Theory Exercise 2

Tarek Auel, Tiange Hu, Benedikt Christoph Naumann, Markus Schanz

Task 1.1: Threading

Ein Monitor ist ein Konzept zur Synchronisierung zweier Prozesse/Threads, das einen gegenseitigen Ausschluss erlaubt. Ein Monitor erlaubt, dass genau ein oder kein Thread sich zur gleichen Zeit im kritischen Codebereich befindet. Befindet sich ein Thread im vom Montior geschützten Bereich, müssen alle anderen, die ebenfalls einen durch diesen Monitor geschützen Bereich betreten wollen, warten, bis der erste Prozess den kritischen Bereich verlassen hat. Ein Monitor besteht aus Variablen und Prozeduren wobei die Variablen nur durch die Prozeduren geändert werden dürfen. Die Prozeduren widerum sind durch den Monitor geschützt um die Synchronization ermöglichen und ein Auftreten von raice-conditions zu verhindern.

Task 1.2: Transparency in Java RMI

- ✅ Access transparency: is gegeben. Lokale und entferne Objekte, die in der Registery angemeldet wurden, werden über diese auf die gleiche Art und Weise angesprochen.
- ✅ Location transparency ist gegeben. Objekte werden in der Registery mit einem beliebigen Namen angemeldet. Dieser kann rein funktional sein.
- ❌ Concurrency transparency ist nicht von Java RMI gegeben. Die Ressourcen müssen dies selbst implementieren.
- ❌ Replication transparency ist nicht gegeben. Jedem Namen (logisch eine Resource) kann genau ein Objekt zugeordnet werden.
- ❌ Failure transparency ist nicht gegeben. Die Registry ist beispielsweise ein Single-Point-Of-Failure. Es sind auch keine anderen Methoden vorhanden, die Ausfälle kompensieren können
- ❌ Mobility transparency ist nicht gegeben. Zwar können Namen in der Registry neu vergeben werden, allerdings bleiben bestehende Verbindungen davon unberührt.
- ❌ Performance transparency ist nicht gegeben. Der Zugriff auf eine lokale Ressource kann deutlich schneller sein. Außerdem sind keine Mechanismen implementiert, die eine Rekonfiguration zur Performancesteigerung erlauben.
- ❌ Scaling transparency ist nicht gegeben. RMI erlaubt es nicht das verteilte System auf einfache Weise durch weitere Knoten zu ergänzen. Durch die fehlende Replication- und Mobility transparency würden neue Knoten nicht automatisch von der bestehenden Infrastruktur genutzt werden können.

Task 1.3: RMI - single-threaded vs multi-threaded

a) Steps for a single request:

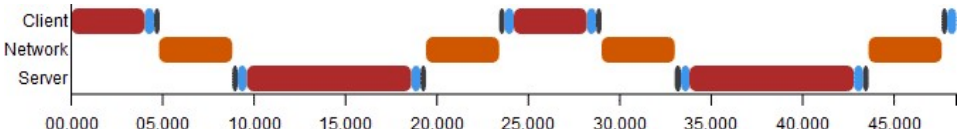
- 4ms: calculate arguments
- 0.5ms: marshalling
- 0.3ms: send
- 4ms: network transfer
- 0.3ms: receive
- 0.5ms: demarshalling
- 9ms: server processing
- 0.5ms: marshalling
- 0.3ms: send
- 4ms: network
- 0.3ms: receive
- 0.5ms: marshalling

Grouped:

- Client until send (incl.): 4.8ms

- Network 4ms
- Server processing time: 10.6ms
- Network 4ms
- Client receiving result: 0.8ms

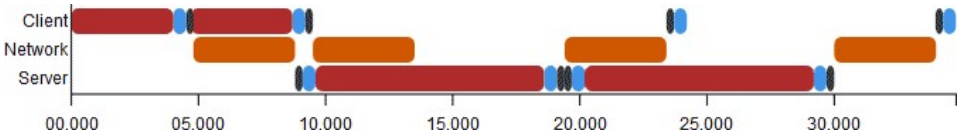
To finish one request: 24.2ms. The client is single-threaded and synchronous, so the second requests starts when the first result is received which leads to a total execution time of 24.2ms * 2 = 48.4ms



In total: 48.4ms

b)

- First request send on network at t=4.8ms, Second request send on network at t=9,6ms
- Request one is received at t=8.8ms, Second request is received at t=13,6ms
- server processes first request until t=19.4ms, and starts immediately to process the second request
- network 1st request + client gets first result, finished at 24.2 ms (not interesting)
- server processes second request starting at t=19.4 until t=30ms
- network 2nd request 4ms until t=34ms, client receives and demarshalleing 0.8ms finished at t=34.8ms



In total 34.8ms