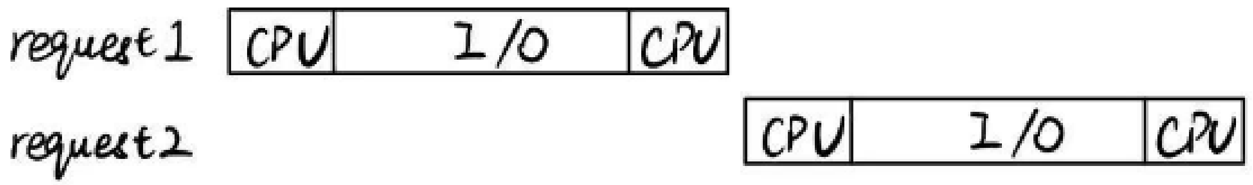
Asynchronous Programming —— Methods Only

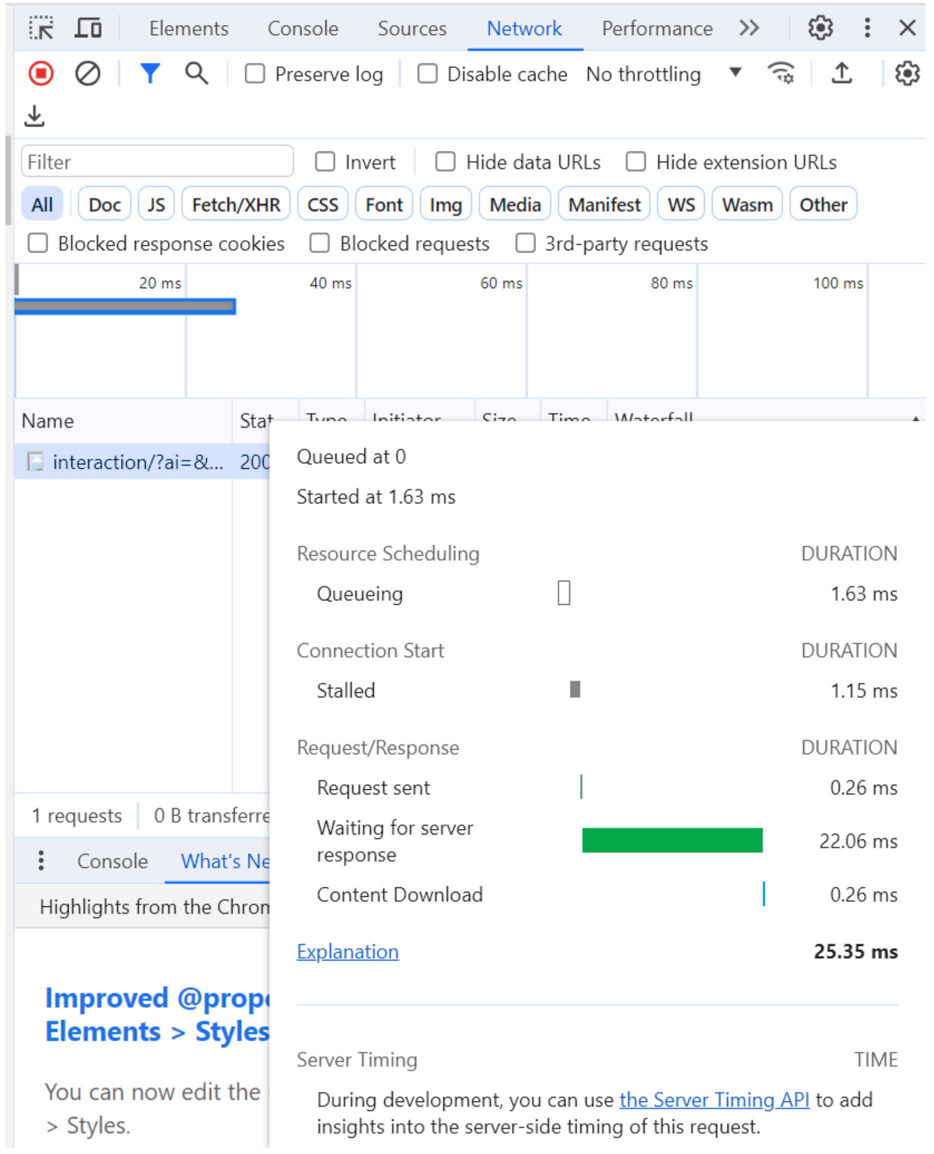
Jiaming Xu

2023.12.13

Background - I/O bound

- Web server
  - Most of the time is spent waiting for server response
  - CPU utilization is **LOW**!





Queued at 0
Started at 1.63 ms

Resource Scheduling	DURATION
Queueing	1.63 ms
Connection Start	**DURATION**
Stalled	1.15 ms
Request/Response	**DURATION**
Request sent	0.26 ms
Waiting for server response	22.06 ms
Content Download	0.26 ms
Explanation	**25.35 ms**

Server Timing TIME

During development, you can use the Server Timing API to add insights into the server-side timing of this request.

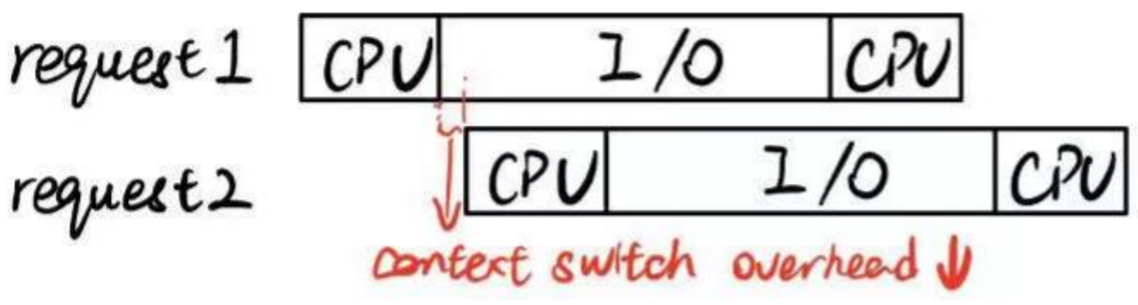
Highlights from the Chron

Improved @prope
Elements > Styles

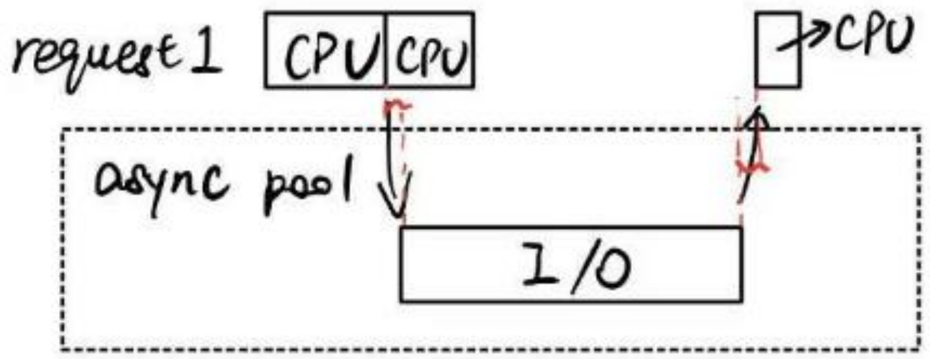
You can now edit the
> Styles.

Motivations

- **Faster and faster** - overcome CPU bottleneck
  - Non-blocking (original)
  - High performance (advanced)
  - Scalability for distributed systems (advanced)



Blocking semantics
immediately or *synchronously*



Non-blocking semantics
purely asynchronously

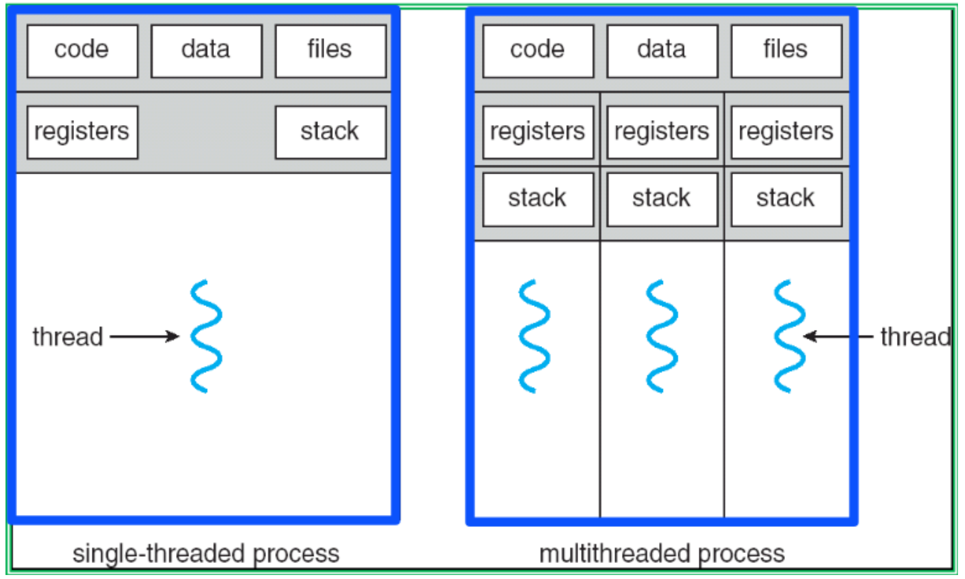
Developments

- Processes
- Threads
  - Callbacks
  - **Futures & Promises**
  - Reactive Extensions
- **Coroutines**

← Programming patterns
(could also apply to coroutines)

Processes

- Instance of a running program with restricted privilege
  - Executable → New process
  - Owns registers, stack, file descriptors, and network connections
  - Shares heap
- Good protection, but poor communication
  - Protected from each other with unique address space
  - Inter-process communication
    - Signals (for events)
    - Semophores (numbers only)
    - Shared memory (fd/mmap) (synchronization)
    - Pipes (unidirectional, only parent-child)
    - Sockets (poor performance)
    - Message queues (restricted volume)



Threads

- Single unique execution context – lightweight, safety concerned

Overhead of context switching

- Processes

  trap into kernel

  registers (%rip, %rsp, …)

  file descriptors

  TLB

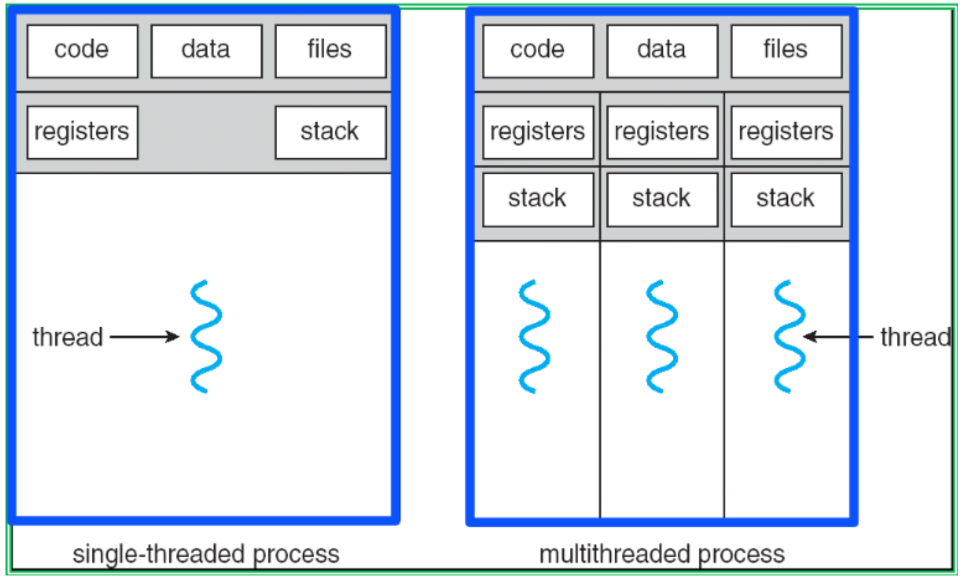
  MMU

  cache (if using virtual addresses)

- Threads

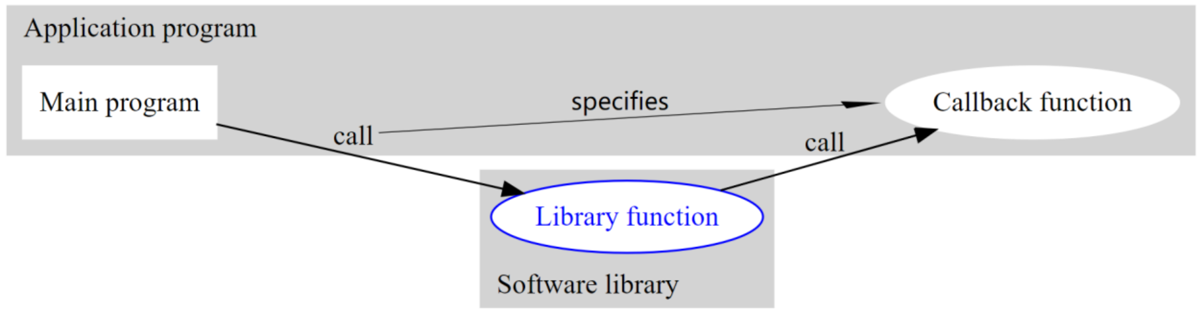
  trap into kernel

  registers (%rip, %rsp, …)



(Deferred) Callbacks

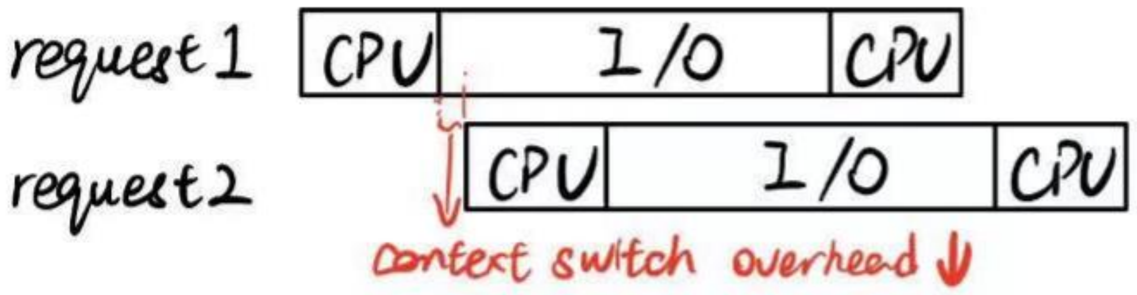
- Pass one function as a parameter to another function
- Dependency between functions, for response processing often (JavaScript for typical)



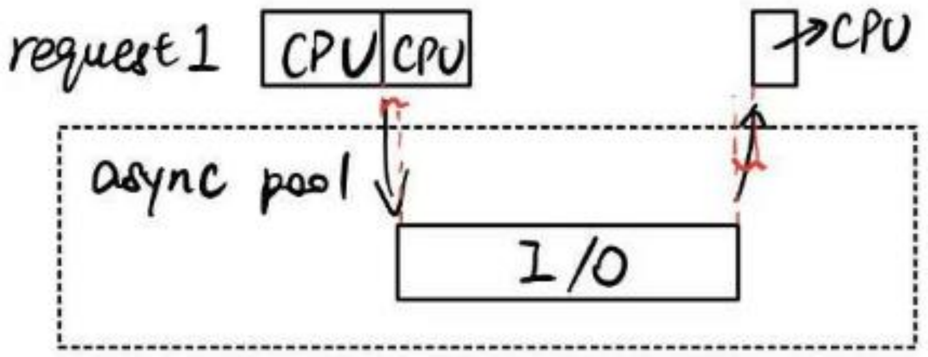
Futures and promises (async/await pattern)

- Promised that at some point it will return
- One-to-one

- Blocking semantics
  - **Simple implementation**

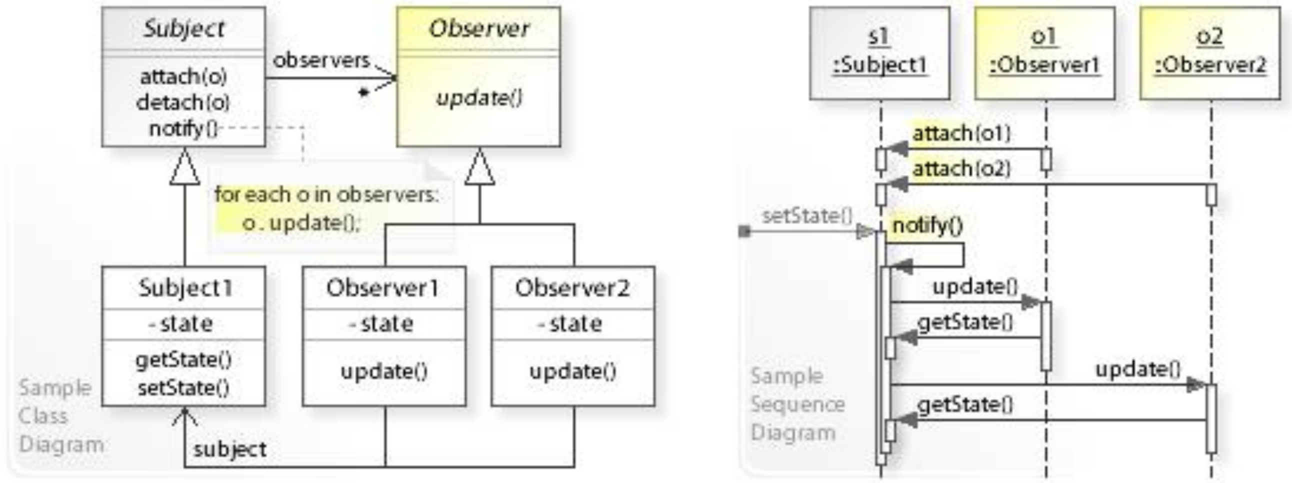


- Non-blocking semantics
  - **Shorter latency per request**



Reactive extensions (observer pattern)

- Observable stream maintaining a list of its dependents
- One object changes, dependent objects update automatically
- One-to-many



Coroutines

- Stackful coroutines – arbitrary function/**User-Level Thread**
  - Suspend at any point
  - User-space scheduling
  - Much more **lightweight** than thread
    - No need to trap into kernel
    - Just switch registers (maintain in a structure)
    - Takes < 40 cycles
- Stackless coroutines - suspendable function
  - Unsuspendable in nested call stack (share & overwrite of a single stack)
  - Much more lightweight than stackful coroutines
    - Just switch several registers
    - Takes < 10 cycles

Appendix

- C++ implementation
  - Boost library
  - Facebook Folly library

- C#, Go, Java (application level), Rust (system level)

References

- Slides of Operating Systems (Honor Track), Xin Jin
  https://pku-os.github.io/
- Asynchronous programming techniques, Kotlin Documentation
  https://kotlinlang.org/docs/async-programming.html
- Callback, Wikipedia
  https://en.wikipedia.org/wiki/Callback_(computer_programming)
- Futures and promises, Wikipedia
  https://en.wikipedia.org/wiki/Futures_and_promises
- Async/await, Wikipedia
  https://en.wikipedia.org/wiki/Async/await
- Observer pattern, Wikipedia
  https://en.wikipedia.org/wiki/Observer_pattern
- Coroutine, Wikipedia
  https://en.wikipedia.org/wiki/Coroutine
- Stackful v.s. stackless
  https://stackoverflow.com/questions/28977302/how-do-stackless-coroutines-differ-from-stackful-coroutines
- Slides of I/O, Asynchrony, and Coroutines (I), Zhenbang You
- Design patterns, Wikipedia
  https://en.wikipedia.org/wiki/Design_Patterns

Acknowledgement

- Zhenbang You

  https://github.com/ZhenbangYou