Concurrency in Chromium:

How I learned to stop worrying and love Sequences

cfredric@

Goals

- Give intuition on Chromium's concurrency model
- Show some useful tips/tricks for working with concurrency in Chromium

Non-goals:

- Explain how things are implemented, in depth
- Show all APIs related to concurrency

Agenda

- What's the problem?
- Chromium's solution
  - Vocabulary
  - Guts
  - Usage patterns

Chromium's Architecture

- Chromium consists of multiple processes:
  - Browser process, renderer processes, utility processes (network process, data decoder process, etc.)
- Each process consists of multiple threads:
  - Main thread (also called UI thread in browser process)
  - IO thread (for IPC, not file/network IO)
  - Other special purpose threads
  - A pool of general-purpose threads

Intra-process Parallelism (in general)

All threads of a given process share the same address space (modulo thread-local storage [TLS]).

How can threads avoid data races, in general?

Multiple approaches:

- Access memory from multiple threads simultaneously
  - "Communicate by sharing memory"
  - Must use mutexes, condvars, etc. to ensure safety
- Send data between threads, without sharing memory
  - "Share memory by communicating"
  - Must use message-passing between threads
- Hybrid

Intra-process Parallelism in Chromium

Chromium uses the hybrid approach, with a strong preference for message-passing:

- Send data and tasks between threads, instead of using locks to synchronize.
- Locks/condition variables exist, but are rarely needed.

The End

~~The End~~

Why not stop here?

- Threads are too coarse-grained & heavy-weight
- Chromium has many independent streams of work to do at a given time
    - Need a way to take independent streams of work and load-balance them between threads

Chromium's concurrency vocabulary

- `Task`: a basic unit of work.
  - Think `OnceCallback` and `RepeatingCallback`.
- Physical thread: an OS thread.
  - Think pthreads on POSIX.
- base::Thread: Chromium's abstraction over physical threads.
  - Platform-agnostic.
- `Sequence`: a "virtual thread"; a "stream of work".
  - An environment that executes a series of tasks in order.
  - Not associated with any particular physical thread.

How does Chromium execute Sequences?

- ❌ One thread : one Sequence
  - Idea: make a new thread to handle each Sequence
  - Too much overhead
- ❌ One thread : many Sequences
  - Idea: each thread owns a set of Sequences that it executes
  - Hard to load-balance
- ✅ Many threads : many Sequences
  - Idea: threads share Sequences, pick one to execute when scheduling the next task
  - Can "move" a Sequence from a busy thread to an idle one => easy to load-balance
  - Doesn't require large number of threads (good for low-end devices)

Why are we here, again?

- Started by discussing **safe** concurrent programming
- Got sidetracked about **efficient** concurrent programming
    - ignored how to make it safe, oops

How to use Sequences safely?

- Goal: use the properties of Sequences to protect against data races
  - Know: data races occur if data is accessed by more than one thread at a time
  - Know: tasks from a given Sequence can execute on only one thread at a time
- => If all the code that accesses an object is on the same Sequence, it's impossible to have a data race involving that object
- => Want something to ensure that whenever we access an object, we do so from a consistent Sequence
  - `SEQUENCE_CHECKER` is built for this!
  - `GUARDED_BY_CONTEXT` makes it impossible to forget to do this check (fails at compile-time).
  - More flexible than `ThreadChecker`, since it doesn't care what physical thread it's on.

Sequences, visualized

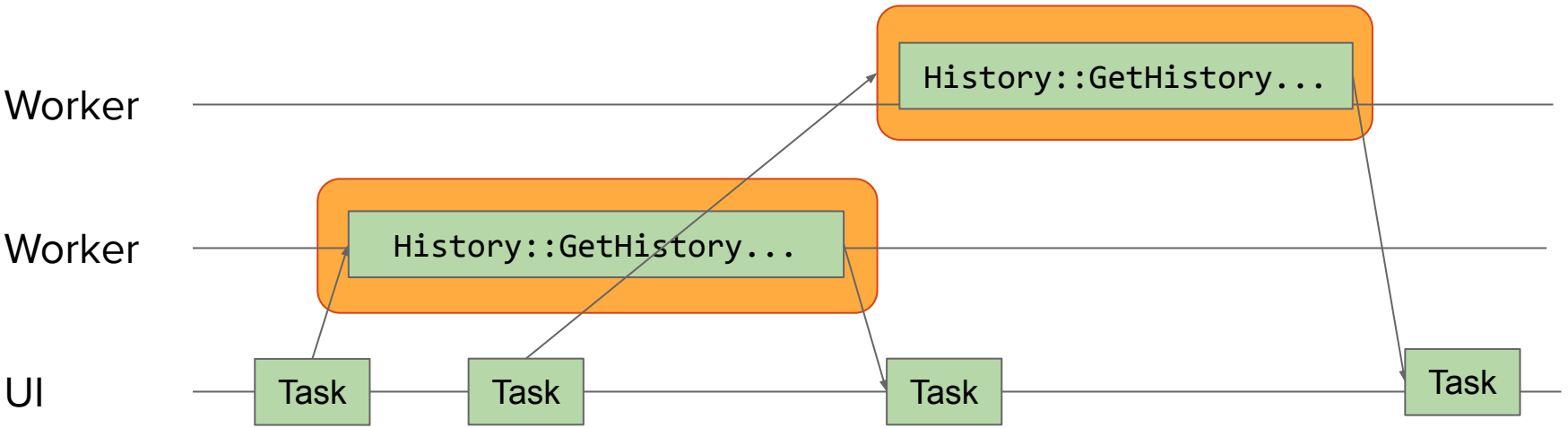

Image credit: Life of a Process, Chrome U 2019

`Sequence` internals

A class that is:

- A `TaskSource`
  - Provides stream of tasks to threading infrastructure.

And has:

- A `SequenceToken`
  - Wrapper around an int.
  - Each instance gets a unique token.
- A `SequenceLocalStorageMap`
  - Like thread-local storage, but for `Sequence`s

How does the infra use Sequences?

- Scheduler ensures that a Sequence only executes one task at a time.
- Before a Sequence's next `Task` is executed, its `SequenceToken` and `SequenceLocalStorage` are put into TLS.
    - => each thread has a unique "currently running Sequence"

Who creates Sequences?

- Sequences are integrated in `ThreadPool`/`TaskRunner` infrastructure
- Sequences get automatically created by:
  - base::ThreadPool::Post[Delayed]Task
  - base::Create[Updateable]SequencedTaskRunner
  - base::CreateSingleThreadTaskRunner

How do I send a task from my Sequence to another?

- I don't care what Sequence I use:
  - `ThreadPool::Post[Delayed]Task` (creates a new Sequence)
- To a specific sequence:
  - `SequencedTaskRunner::Post[Delayed]Task`
  - `SingleThreadTaskRunner::Post[Delayed]Task`
- `SequenceBound<T>` can help call methods/ctor/dtor on a specific sequence.

How do I run tasks on "my" Sequence?

- Run a task on some other sequence, then come back:
  - `TaskRunner::PostTaskAndReply[WithResult]`
  - `ThreadPool::PostTaskAndReply[WithResult]`
- Run something on "my" sequence, asynchronously:
  - `SequencedTaskRunnerHandle::Get()->Post[Delayed]Task`

I don't know what Sequence I need to run on!

- You might not have to do anything!
  - Often APIs implicitly use sequences properly.
  - E.g. `mojo::Receiver::Bind` by default schedules message events on the sequence that called `Bind`.

References

- Threading and Tasks in Chrome
- Threading and Tasks FAQ
- Share Memory By Communicating - The Go Programming Language
- The Chromium Chronicle #1: Task Scheduling Best Practices
- Chrome U 2019: Life of a Process (slides)
- Callbacks in Chromium

Appendix

- Jobs (post_job.h)
  - Power-user API, for bulk-processing with minimal scheduling overhead