Engineer

Adopted Swift 6 strict concurrency over C

Adopted Swift 6 complete strict concurrency with no actors, bridging a blocking C event loop to the main actor through one producer, one consumer and one ordering — after establishing that a task per event loses the ordering the interface depends on.

September 2026

strict concurrency with one ordering, and no actors

Situation. Swift 6's complete strict concurrency checking turns data races into compile errors instead of intermittent crashes. Adopting it against a C library is where it gets difficult: the core's event loop is a blocking call that must run off the main thread forever, and the values it hands back are pointers with no concurrency guarantees at all. The compiler cannot reason about any of it and will refuse everything until the boundary is described explicitly.

Task. Complete checking had to be enabled with no escape hatches, which meant designing the crossing from a blocking C loop to the main actor rather than annotating around it.

Action. The obvious approach is an actor per subsystem, and it was rejected on measurement rather than taste. Spawning a task per incoming event lets the runtime schedule them in any order, and the core's event stream is ordered — a message-changed event that overtakes the message-created event it refers to produces an interface showing an edit to something that does not exist yet. Actor reentrancy makes this worse, not better, because an actor can suspend mid-method and process another call. What replaced it is deliberately plain: one producer thread owning the blocking loop, one consumer, one queue between them, and a single hop onto the main actor at the end. Ordering is preserved because there is exactly one path and nothing overtakes anything. The unsafe types crossing that boundary are wrapped in types whose thread-safety is

asserted at the wrapper rather than assumed, and the assertion is documented with why it holds — the pointer is owned by one thread and copied before it is handed over.

Result. The application compiles under complete strict concurrency with no suppressions, and event ordering is a structural property rather than a hope. The cost is that the design is less parallel than it could be: everything funnels through one consumer, and if that consumer ever becomes a bottleneck the fix will require re-deriving which events can be reordered safely, which is exactly the analysis this avoided.

Bring this experience to your project

Use “Adopted Swift 6 strict concurrency over C” as a starting point for a similar challenge in your organisation. The services behind this work are available for your project. The shape of a system worked out on paper first, so the parts fit and the cost is known before anyone writes code.

Explore our services Get in touch →

Look behind the result

Read the situation, decisions and result in this example. We can discuss which parts transfer to your constraints and which need another approach. Adopted Swift 6 strict concurrency over C — strict concurrency with one ordering, and no actors

Explore our services Get in touch →

Want concurrency the compiler can prove is safe?

Talk through your project with the engineer responsible for delivery. We agree the scope, price basis and handover in writing before work begins.

Role context: macOS Application Engineer. It draws on Solution Architecture, Backend Engineering, Frontend Engineering, Reliability & Backups, Performance Tuning and Testing & QA. And while you are here, we are rather good at Platform & Solution Architecture, Backend & API Development and Technical Leadership & Consulting.

Get in touch Explore our services

One piece of **a longer record** — and it is all on this site.

Every entry is written the same way: the situation, the task, what we did and what changed.

See the whole record

Define the improvement you need

Describe the workflow, the existing interfaces and the users who need the change. We can define a testable delivery and a handover that lets your team maintain it.

See what it costs Get in touch →

Want concurrency the compiler can prove is safe?

Ask about the decisions behind “Adopted Swift 6 strict concurrency over C”. You will speak with the people who can explain the work and assess your own project.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/portfolio/adopted-swift-6-strict-concurrency-over-c/