Native I/O backend for FS2 JVM

Name: Onur Şahin
E-mail: sahinonur2000@hotmail.com
Address: Richard Wagner Straße 66, 79104 Freiburg, Germany
Website: https://blog.aiono.dev/
Github: https://github.com/onsah/

Abstract

fs2 is a "Functional, effectful, concurrent" streaming I/O library. It allows building and transforming arbitrarily complex streams possibly with side effects. With fs2, you can build streams with functional style while keeping constant memory usage and linear time complexity.

Currently, fs2 makes use of `NIO` in it's JVM implementation for the networking API. `NIO` uses a lot of synchronization (locking) and causes indirection which result with non-trivial performance penalty. The project aims to implement it's I/O functionality using direct OS APIs such as `epoll`/`kqueue` for the JVM implementation. The goal is to reduce locking and indirection to increase the performance. The Scala Native fs2 implementation already uses `epoll`/`kqueue`.

How it will improve fs2?

- **Efficiency**: `fs2` on JVM will perform better due to removed overhead.
- **Reduced Hardware Costs**: Because the hardware is utilized better, cheaper hardware is sufficient.
- **Tighter runtime integration**: Because I/O operations are implemented natively, runtime can be more strategical.

Related Work

It's a part of a larger project in `cats-effect` about integrating I/O into runtime by merging compute tasks with I/O in a single thread. A detailed discussion can be found in https://github.com/typelevel/cats-effect/discussions/3070.

A part of the larger project, a `PollingSystem` is already implemented in `cats-effect` and it's being used in `fs2` (implementation PR in cats-effect). Currently the `PollingSystem` for JVM is implemented using NIO `Selector` API. This project will implement another `PollingSystem` based on `epoll`/`kqueue`. So the project will be very similar to the linked PR.

Mentors

- Arman Bilge (Website)
- Antonio Jimenez (Github)

Deliverables

1. `epoll` based `PollingSystem` in `cats-effect`.
3. `kqueue` based `PollingSystem` in `cats-effect`.
2. `fs2` migration to `PollingSystem`.
4. Performance benchmarking against the previous `PollingSystem`.
5. Documentation: Both internal and external.

Proposal Timeline

This project has three main parts:

- `epoll` polling system in `cats-effect`
- `kqueue` polling system in `cats-effect`
- Using `epoll`/`kqueue` polling systems in `fs2` for networking.

I believe having one vertical slice of the implementation will help getting faster feedback and improve the overall development process. Therefore I will first implement `epoll` polling system and use it in some parts of `fs2`. Then once it works properly, I will work on implementing `kqueue` polling system. Then I will work on remaining places to migrate in `fs2`.

Mentors informed me that, migrating all appropriate modules in `fs2` into `PollingSystem` might not be realistic for the project duration. Therefore, I added some modules into the timeline as optional.

May 1 - May 10:

- Get familiar with `fs2` and `cats-effect`. Knowing how tools are used will help me see retain big picture when delving into the implementation details.
- Setup codebases locally. I already did this for `fs2` when I previously contributed.
- Keep close communication with mentors. Regularly ask questions I have regarding project design and details.

May 10 - June 2

- Investigate how networking I/O us implemented in `cats-effect` and how NIO is used in the runtime (this PR).
- Study existing migration to `PollingSystem` in `fs2` (this PR).
- Study `epoll` and `kqueue` APIs. Possibly implement prototype programs using them.
- Study how to call system APIs from Scala. This will be necessary to utilize native I/O APIs from within `cats-effect`. Decide what technology to use. One possibility is to use JNR.

June 2 - June 6 (Official coding period starts)

- Create a Github project under the Typelevel organization. Each milestone will be opened as an issue and will be linked to this project.
- Setup infrastructure to call native code from `cats-effect` in JVM implementation.

June 7 - June 29

- Implement `epoll` polling system in `cats-effect` for JVM.
- Perform automated and manual tests. Write additional automated tests if necessary.

June 30 - July 27

- Migrate `SocketGroup` / `Socket` in `fs2` to use `PollingSystem`.
- Migrate `UnixSockets` in `fs2` to use `PollingSystem`.
- Ensure proper documentation is in place.
- Test that `SocketGroup`, `Socket` and `UnixSockets` modules work correctly with `epoll` `PollingSystem`. Write automated tests.
- (Optional) if there is enough time, also migrate `DatagramSockets` and `Process` modules.

July 28 - August 10

- Implement `kqueue` polling system in `cats-effect` for JVM.
- Test that `SocketGroup`, `Socket` and `UnixSockets` modules work correctly with `kqueue` `PollingSystem`. Write automated tests.

August 11 - August 17

- Benchmark `epoll`/`kqueue` polling systems against NIO based polling system.
- Create a results report out of the benchmark outcomes.

August 18 - August 30

- Document new polling systems including internal implementation for their implementation details.

- Perform final testing to ensure no regressions happened and there are performance gain.
- Optimize and refactor the codebase.
- Cleanup the codebase for final submission. Ensure no commented code or useless comments exist.

Availability

Generally I am available around 25-30 hours weekly during the coding period.

Currently I am in my 4th semester of my Masters. Normally I would start thesis around June, but if my proposal is accepted I will delay my thesis process so that I start working on it after GSOC. Therefore my thesis won't cause any availability issues for GSOC.

I will probably have one final exam during the coding period. To prevent any surprises I extend the standard coding period by one week in my proposal.

I have a part time job which takes around 10-15 hours weekly. At the time coding period starts, my responsibilities from it will be low because it's a university student job which is more intense during the first half of the semester. Considering I will have only one lecture and my part time job, I can comfortably give 25-30 hours weekly to this project.

Project Management

Publishing the Code

- I plan to open one pull request per milestone.
- Pull requests will be merged into the project specific branch first.
- Once all milestones are implemented, a final merge to the main branch will be performed.

Best Practices

- I will follow project's contribution rules such as formatting, and certain coding conventions.
- I will clarify anything uncertain by discussing them with my mentor and community.

Sharing

- During the project, I plan to write about it in my blog so that I can share the knowledge I gained with others. Also it will help me to mentally organize the project and serve as a documentation.

My Background

I am passionate about systems programming and functional programming, with a strong interest in building scalable and efficient systems. As a demonstration of this, I developed Flux_rs, a toy programming language implementation (with bytecode VM) that deepened my understanding of language design and runtime mechanics.

Open source has a very special for me since I started programming, and I actively contribute to projects that align with my interests in systems programming and developer tools. Some highlights include:

- Maintained a Gnome Shell Extension with 16 stars
- Contributed to Nu Shell's standard library, enhancing usability for a growing community.
- Making multiple contributions to Jakt typechecker.

Professionally, I have 2 years of experience using Scala, including working with Scala Futures, thread pools, and non-blocking I/O in production environments. This hands-on experience has given me a solid grasp of concurrency models and performance considerations in real-world systems.

While I'm still deepening my familiarity with Cats, I've been actively learning the ecosystem through starter guides and practical contributions. To demonstrate my commitment to the Typelevel

ecosystem, I engaged early by contributing to fs2. Under the mentorship of Arman Bilge, I successfully implemented a non-blocking process API, with my changes merged in https://github.com/typelevel/fs2/pull/3539 and https://github.com/typelevel/fs2/pull/3548.. This experience not only strengthened my understanding of functional effect systems but also showcased my ability to deliver meaningful contributions to open-source projects.

I'm excited to continue contributing to Typelevel and further grow my expertise in functional programming through GSoC.