GSoC 2022 Proposal | Rewrite PJDFSTest suite

Introduction

This project aims to rewrite the PJDFSTest suite. Today, the tests are written in a mix of shell script and C. This approach has provided some flexibility and usability, allowing to use syscalls within a shell environment. However, it also has disadvantages, the main ones being performance, code duplication and higher entry barrier for potential contributors. We want to improve the test suite, by switching to a unique language, Rust. Rust has numerous advantages, in particular, fearless multithreading, low-level handling and increased safety.

TL;DR Rewrite the tests in Rust, with a custom-built test runner for running them standalone, and rely on Kyua for running tests along with ATF support to get reports.

whoami?

Name: Sayafdine Said
GitHub: musikid
Discord: musikid.#7043
Matrix: musikid64
Mail address: musikid@outlook.com
Timezone: Paris, France (UTC+2)

Currently a computer science student at Sorbonne University, I am proficient in Rust, Python, C/C++ and JavaScript. I run Arch Linux and FreeBSD, so I am pretty familiar with the Unix environment. I wrote fixes for some open source projects, but never been more involved. I also did a bit of yak shaving, fancy being the most complete project I produced. This GSoC is the perfect occasion to be involved in the open source community, particularly for the FreeBSD project, that I love.

Timeline

After implementing the test collection along with the fixtures, I aim to write a single-threaded test runner. Then, as a secondary objective, I will add ATF support to rely on Kyua test runner, to inherit from its high quality reporting. Finally, I will try to add support for multithreading to our test runner.

Adding multithreading and ATF support are not top priorities, though, the main one being to write the tests.

NOTE: All the time intervals imply writing the documentation along with the code.

Community bonding (May 20)

- Get more familiar with the current codebase
- Review the missing syscalls
- Discuss on details

1st week (June 13)

- Iterate on the project's design

2nd week - 4th week (June 20)

- Implement test collection
- Implement fixtures

5th - 7th weeks (July 11)

- Implement test runner
- Start writing the tests

Phase 1 evaluation period (July 25)

8th - 9th weeks (August 1)

- Add ATF support (not a priority)
- Continue writing the tests

10th - 12th weeks (August 15)

- Add multithreading support (not a priority)
- Continue writing the tests

13th week - End (September 4)

- Document extensively
- Write the eventual missing tests

Details

Test collection

The project will not rely on the Rust testing framework, therefore, we will need to do the test collection ourselves. Since we plan to adopt an approach similar to Criterion, we will need to write macros. Criterion collects the tests in a group (`criterion_group!`), which is turned into a function, to finally aggregate all these functions into the main one (`criterion_main!`). We want to adopt a similar approach, with some nuances, however.

Since we want to be able to list the tests, collecting them in a function seems more troublesome. Instead, we will collect them in a slice, along with/within a structure for easing listing them. The structure could be constructed using an attribute macro, or plain syntax if I'm running out of time.

Fixtures

We can take inspiration from pytest and rstest, and use an attribute macro to add the fixtures to the test's parameters. Though, since they will certainly be harder to write with attribute (procedural) macros, I might switch to using plain functions, if I'm running out of time.

Macros

`pjdfs_group!` Make a group of tests.

`pjdfs_main!` Make the main function.

`#[pjdfs_test]` Declare a test. It might come with other attributes to declare conditions for ATF. For example, some tests need to be run as root.

`#[fixture]` Declare a fixture.

Command-line arguments

The program will support ATF, so the command-line interface should be compatible with it. ATF has a really simple interface, consisting only of two running modes:

- `-l`, to list all the tests and their conditions.
- `[-r resfile] [-s srcdir] [-v var1=value1 [.. -v varN=valueN]] test_case`, to run a test case.

Layout

Currently, tests are organized by syscalls, and the rewrite should adopt a similar approach. Like explained in the previous section, we declare the tests by groups.

For example:

symlink/mod.rs

```
pjdfs_group!(symlink, return_enoent, return_eaccess);
```

main.rs

```
pjdfs_main!(symlink);
```

Relevant links

https://github.com/pjd/pjdfstest/issues/59

https://github.com/musikid/pytest-atf.git

https://www.freebsd.org/cgi/man.cgi?query=atf-test-program&sektion=1&apropos=0&manpath=FreeBSD+13.0-RELEASE+and+Ports

https://www.freebsd.org/cgi/man.cgi?query=atf-test-case&sektion=4&apropos=0&manpath=FreeBSD+13.0-RELEASE+and+Ports