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, mainly by switching to a unique language. After further discussions, we agreed on using Rust, given its numerous advantages, particularly fearless multithreading, low-level handling and 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.

Who am I?

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

Fluent in French, English and Spanish

I wrote fixes for various open source projects, but never been more involved. I also did a bit of yak shaving (meh), fancy being the most complete one. This GSoC is the perfect occasion to be more involved in the open source community, particularly for the FreeBSD project, that I like.

Timeline

I aim to write first a single-threaded test runner, along with fixtures to be shared between the tests. Then, as a secondary objective, I will also 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.

Community bonding

- Get more familiar with the current codebase
- Review the missing syscalls

1st week (June 13)

- Iterate on the project's design

2nd week - 4rd week (June 20)

- Implement the test collection
- Implement fixtures

5th - 7th weeks (July 11)

- Implement test runner

Phase 1 evaluation period (July 25)

8th - 9th weeks (August 1)

- Add ATF support

10th - 13th weeks (August 15)

- Add multithreading support

13th week - End (September 4)

- Document extensively

Architecture

Test collection

The project will not rely on the Rust testing framework, therefore we will to need to do the test collection ourselves. Since we plan to adopt an approach similar to Criterion, we will need to write ~~procedural~~ 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.

Fixtures

We could take inspiration from pytest and rstest, and use an attribute macro to add the fixtures to the test's parameters.

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);
```

Interface

Macros

Command-line arguments

The program should support ATF, so the command-line interface shoud 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.

Relevant links

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

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