Bringing Tier-1 support for **64-bit RISC-V Linux** to **Rust**

We are bringing the 64-bit RISC-V Linux port of Rust to Tier-1 ("guaranteed to work") from its current Tier-2 status.

Toolchains like Ferrocene already enable Rust adoption in mission-critical systems.

The partnership between Ferrous Systems and Codethink is enabling adoption on a wider range of embedded hardware and operating systems.

```
lib.rs  U ×
src > lib.rs > ...
//! Parsers for various /proc files
//!
//! We cover formats such as those used in `/proc/meminfo` and `/proc/vmstat`

use std::path::{Path, PathBuf};

/// A parser for files which are key/value sequences
///
/// We parse lines approximating:
/// `^\s*(?P<key>\S+)\s*:?\s*(?P<value>\S+).*$`
2 implementations | 5 references
pub struct KVParser {
    location: PathBuf,
    colons: bool,
    whitespace_in_values: bool,
    buffer: [u8; 8192], // A buffer so we don't allocate at parse time
}

impl KVParser {
    ▶ Run Doctest
    /// Prepare a new `KVParser` ready for use
    ///
    /// We can choose to use colons as the separator, and whether or
    /// not to allow for whitespace in values.
    ///
    /// ```
    /// # use procparser::KVParser;
    /// let parser = KVParser::new("/proc/meminfo", true, false);
    /// ```
    0 references
    pub fn new(location: impl AsRef<Path>, colons: bool, whitespace_in_values: bool) -> Self {
        Self {
            location: location.as_ref().to_path_buf(),
            colons,
            whitespace_in_values,
            buffer: [0; 8192],
        }
    }

    ▶ Run Doctest
    /// Change the filename associated with this parser
    ///
    ///
    /// # use procparser::KVParser;
    /// # use std::path::Path;
    /// let mut parser = KVParser::new("/proc/meminfo", true, false);
    ///
    /// parser.set_location("/proc/status");
    ///
    /// assert_eq!(parser.location(), Path::new("/proc/status"))
    /// ```
    0 references
    pub fn set_location(&mut self, location: impl AsRef<Path>) {
        self.location = location.as_ref().to_path_buf();
    }

    ▶ Run Doctest
    /// Read and parse the file we've been prepared for
    ///
    /// The number of KV pairs which were passed to the callback will be returned
    /// on success.
    ///
    /// ```
    /// # use procparser::KVParser;
    /// let mut parser = KVParser::new("/proc/meminfo", true, false);
    ///
```

Rust:

... is an open-source **systems programming language** designed to empower everyone to build reliable and efficient software:
Safety - Performance - Productivity

... is a **statically-typed compiled language.**

Its powerful language-level static analysis prevents use-after-free, double-free, shared mutability, threading race hazards, and more. The libraries and binaries (known as crates) are composed of modules, which have clear type-checked public, semi-private and private APIs.

... is **cross-platform** and supports a wide range of Operating Systems:

- **Windows, Linux, macOS**
- **FreeBSD/NetBSD/OpenBSD/Illumos**
- **VxWorks, QNX, LynxOS-178**
- **RTOS like ThreadX, FreeRTOS or RTEMS, and**
- **bare-metal platforms with no OS or heap**

- **Cargo**, the package manager and build system
- **rustdoc**, the documentation generator
- **a test system**, which supports unit tests integration tests and doc tests
- a linter, **clippy**, which spots code which is technically correct but not idiomatic

... uses LLVM and is a cross-compiler out-of-the-box. It supports a wide range of CPU architectures:

- Intel **i686** and **x86-64**
- **RISC-V** 32-bit and 64-bit
- **Arm** Cortex-M, Cortex-R and Cortex-A
- **SPARC**
- **PowerPC**, **MIPS**, and more!

www.codethink.co.uk

ferrous-systems.com