为 rCore 实现更多系统调用

罗崚骁　　宋香君

2020 年 5 月 16 日

课程设计目标

- rCore 自编译
  - musl-libc 完善
  - Make, CMake
  - Rustc, Cargo
  - 文件系统支持

目前进展

运行 shell 脚本

- ash + 脚本文件名运行脚本（sys_dup，fcntl）
- 命令替换（Command Substitution）
  - 诸如 echo $(echo 233) 这样的命令
  - 需要对管道增加阻塞
- 可以跑通 make 的 configure 脚本和构建脚本

目前进展

GNU Make

- 能够在 rCore 内部缺少 make 的情况下构建，configure+build.sh
- 基本功能正常运行，可以使用 3.78.1 的 make 构建出 3.79 的 make
- 支持增量构建
  - 增加了文件系统对修改时间的记录，完善了对 stat 的实现，顺便还能支持 touch

目前进展

libc-test

- Makefile 正常执行
- 给出了所有测例的执行结果
  - 258 个测例成功
  - 215 个测例失败，其中 18 个测例无法正常结束
  - 失败的测例主要是 pthread（缺少信号机制支持）、math（大量计算错误，140 个左右）相关
  - 修好了全部信号量相关的测例

后续任务

Cargo

- 目前只能新建项目，还无法成功使用 Cargo 构建

```
[DEBUG][0,1] 0:1:1 syscall id 293 begin
[ INFO][0,1] pipe2: fds: 0x7fffffffa088, flags: 0x80000
[ INFO][0,1] pipe: created rfd: 3 wfd: 4
[ INFO][0,1] => Ok(0)
[DEBUG][0,1] 0:1:1 syscall id 7 begin
[ INFO][0,1] poll: ufds: 0x7fffffffa048, nfds: 1, timeout_msecs: 0xffffffffffffffff
[ INFO][0,1] poll: fds: [PollFd { fd: 3, events: IN, revents: (empty) }]
```

后续任务

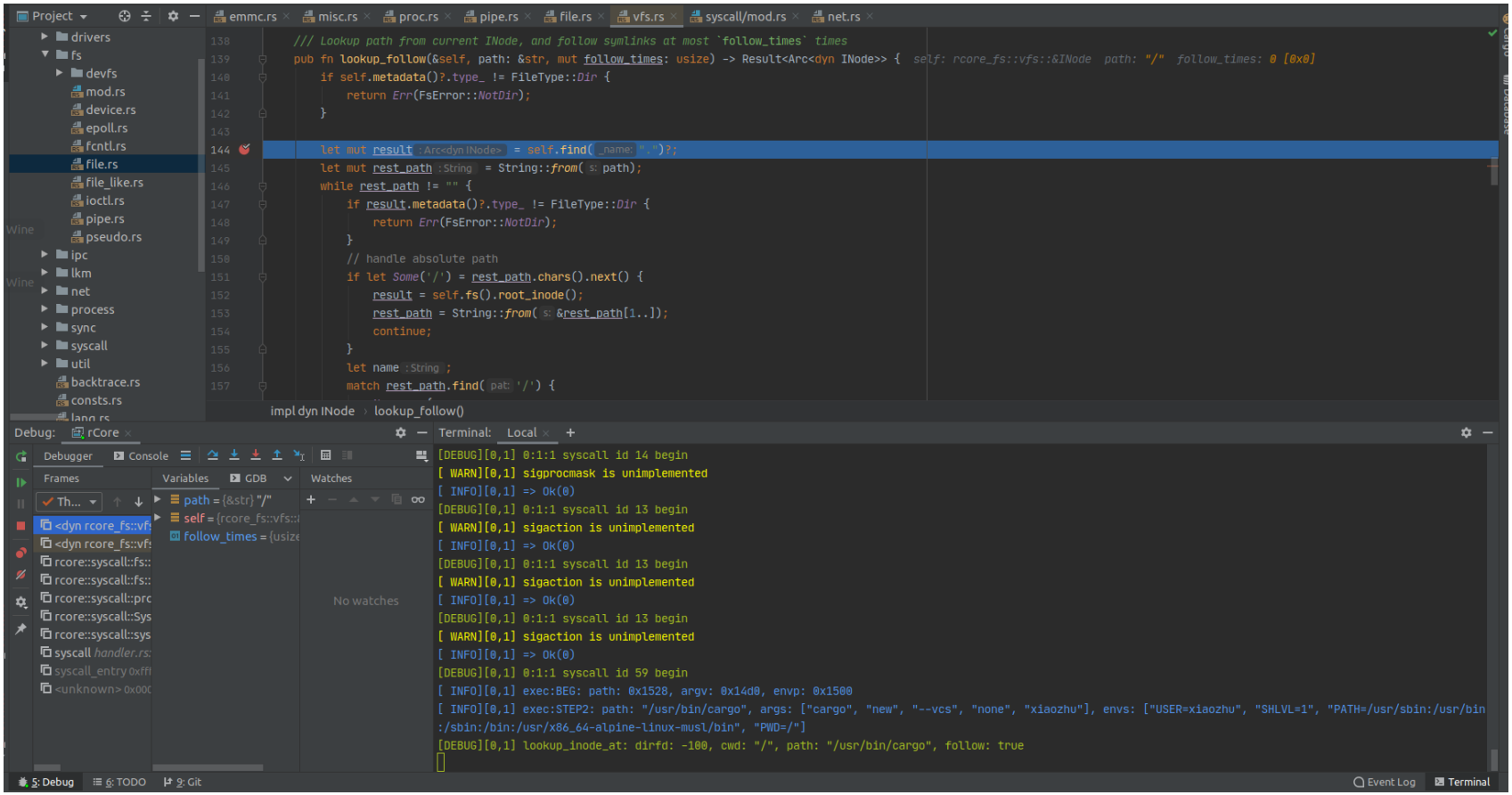
libc-test

- 主要针对 pthread 和数学的相关测例进行修复
- 增加信号机制，实现一些和信号相关的系统调用
- 编写文档，总结 rCore 开发微小的经验

调试方法

- 除了可以开启记录模式打印每条系统调用外，还可以用 gdb 通过相应端口连接到 qemu 进行调试
- 使用 CLion 提供的 gdb remote debug 可以在 IDE 中更方便地定位到错误
- 不过目前 rCore 的 Debug 版也开了 O2，调试时困难变大

调试方法


/// Lookup path from current INode, and follow symlinks at most `follow_times` times
pub fn lookup_follow(&self, path: &str, mut follow_times: usize) -> Result<Arc<dyn INode>> {
if self.metadata()?.type_ != FileType::Dir {
return Err(FsError::NotDir);
}
let mut result = self.find(".")?;
let mut rest_path = String::from(path);
while rest_path != "" {
if result.metadata()?.type_ != FileType::Dir {
return Err(FsError::NotDir);
}
// handle absolute path
if let Some('/') = rest_path.chars().next() {
result = self.fs().root_inode();
rest_path = String::from(&rest_path[1..]);
continue;
}
let name
match rest_path.find('/') {
impl dyn INode
lookup_follow()
Debug:
rCore
Debugger
Console
Frames
Variables
GDB
Watches
path = {&str} "/"
self = {rcore_fs::vfs::
follow_times = {usize
No watches
Terminal:
Local
[DEBUG][0,1] 0:1:1 syscall id 14 begin
[WARN][0,1] sigprocmask is unimplemented
[INFO][0,1] => Ok(0)
[DEBUG][0,1] 0:1:1 syscall id 13 begin
[WARN][0,1] sigaction is unimplemented
[INFO][0,1] => Ok(0)
[DEBUG][0,1] 0:1:1 syscall id 13 begin
[WARN][0,1] sigaction is unimplemented
[INFO][0,1] => Ok(0)
[DEBUG][0,1] 0:1:1 syscall id 13 begin
[WARN][0,1] sigaction is unimplemented
[INFO][0,1] => Ok(0)
[DEBUG][0,1] 0:1:1 syscall id 59 begin
[INFO][0,1] exec:BEG: path: 0x1528, argv: 0x14d0, envp: 0x1500
[INFO][0,1] exec:STEP2: path: "/usr/bin/cargo", args: ["cargo", "new", "--vcs", "none", "xiaozhu"], envs: ["USER=xiaozhu", "SHLVL=1", "PATH=/usr/sbin:/usr/bin
:/sbin:/bin:/usr/x86_64-alpine-linux-musl/bin", "PWD=/"]
[DEBUG][0,1] lookup_inode_at: dirfd: -100, cwd: "/", path: "/usr/bin/cargo", follow: true
5: Debug
6: TODO
9: Git
Event Log
Terminal

问题选讲

IO 阻塞与进程死锁

- 在有阻塞的 IO 中，目前的设计里 inode 的实现中看不见进程，并不能释放进程锁
- 于是，设想有如下情景：进程 2 由进程 1 fork 出来，并且有连接他们的管道，进程 1 读进程 2 写。进程 1 先读，保持着锁进入阻塞；进程 2 在写之前需要调用 getppid 获取父进程 id，便发生了死锁。
- 解决方法：考虑到进程的 pid 其实是定值，不必要用锁保护，可以和进程并列保存。
- 后续：IO 阻塞时释放进程锁

结束

谢谢观看