面向 rCore-Tutorial 操作系统的调试工具

OS 大赛报告会 13 场

陈志扬

指导教师：吴竞邦

北京工商大学计算机科学与技术系

2023 年 6 月 17 日

项目背景与目标

项目背景

- rust 操作系统相关实验上手难度较高，环境配置繁琐
- GDB TUI 不方便，VSCode 有调试器插件，但是只能调试用户态程序

工作内容

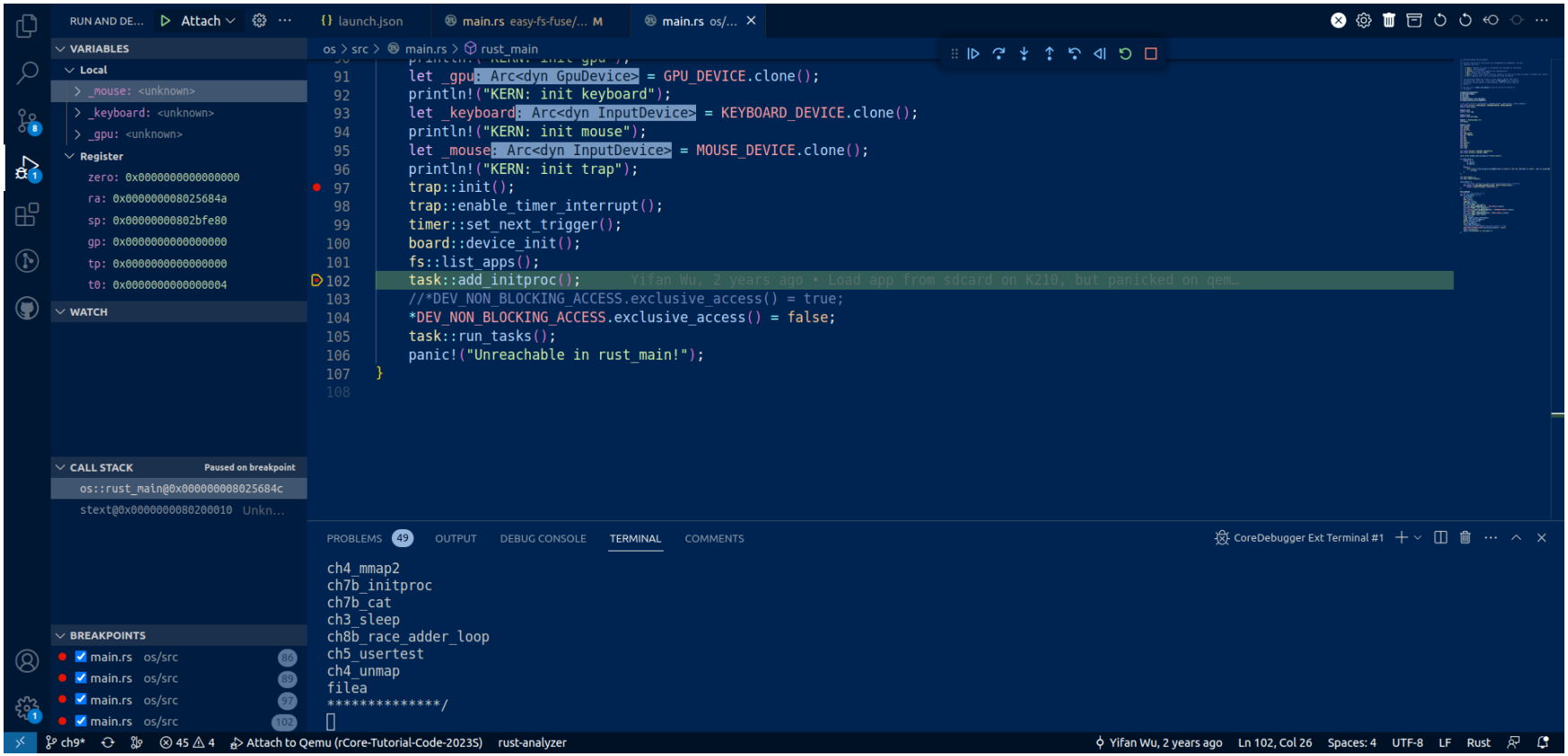
- 在已有的调试器基础上，增加操作系统调试功能，如同时设内核态用户态断点，获取当前特权级，自动更换符号表文件
- 修改 QEMU 虚拟机，从而支持基于串口的调试
- 结合 eBPF 技术，同时用静态分析和动态分析技术进行函数跟踪和信息获取（寄存器，内存，函数调用参数）

项目背景与目标

主要技术难点

- 用两种跟踪技术（Qemu 的 gdbserver 和 eBPF Server），两个数据来源同时跟踪同一个操作系统
- 同时跟踪操作系统的内核态和用户态；
- 支持多个基于中断的 Qemu 虚拟串口传输
- 协调同步消息和异步消息
- 适配 RISC-V 处理器（PMP 机制，函数调用规范）

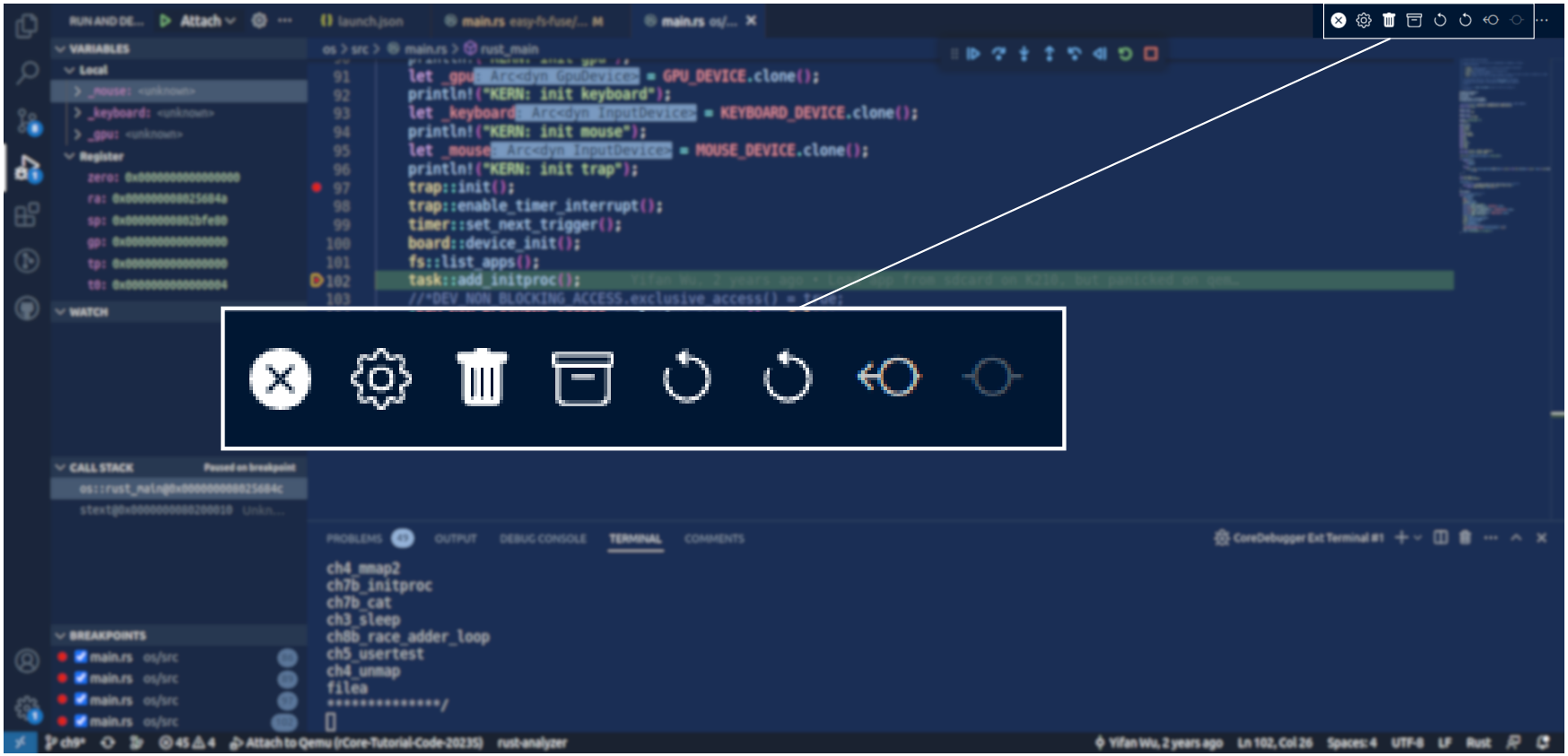
用户界面


RUN AND DE... Attach
launch.json
main.rs easy-fs-fuse/... M
main.rs os/...
os > src > main.rs > rust_main
VARIABLES
Local
_mouse: <unknown>
_keyboard: <unknown>
_gpu: <unknown>
Register
zero: 0x0000000000000000
ra: 0x000000008025684a
sp: 0x00000000802bfe80
gp: 0x0000000000000000
tp: 0x0000000000000000
t0: 0x0000000000000004
WATCH
CALL STACK Paused on breakpoint
os::rust_main@0x000000008025684c
stext@0x0000000080200010 Unkn...
BREAKPOINTS
main.rs os/src 86
main.rs os/src 89
main.rs os/src 97
main.rs os/src 102
let _gpu: Arc<dyn GpuDevice> = GPU_DEVICE.clone();
println!("KERN: init keyboard");
let _keyboard: Arc<dyn InputDevice> = KEYBOARD_DEVICE.clone();
println!("KERN: init mouse");
let _mouse: Arc<dyn InputDevice> = MOUSE_DEVICE.clone();
println!("KERN: init trap");
trap::init();
trap::enable_timer_interrupt();
timer::set_next_trigger();
board::device_init();
fs::list_apps();
task::add_initproc(); Yifan Wu, 2 years ago • Load app from sdcard on K210, but panicked on qem…
//*DEV_NON_BLOCKING_ACCESS.exclusive_access() = true;
*DEV_NON_BLOCKING_ACCESS.exclusive_access() = false;
task::run_tasks();
panic!("Unreachable in rust_main!");
}
PROBLEMS 49 OUTPUT DEBUG CONSOLE TERMINAL COMMENTS
CoreDebugger Ext Terminal #1
ch4_mmap2
ch7b_initproc
ch7b_cat
ch3_sleep
ch8b_race_adder_loop
ch5_usertest
ch4_unmap
filea
**************/
ch9* 45 4 Attach to Qemu (rCore-Tutorial-Code-2023S) rust-analyzer
Yifan Wu, 2 years ago Ln 102, Col 26 Spaces: 4 UTF-8 LF Rust

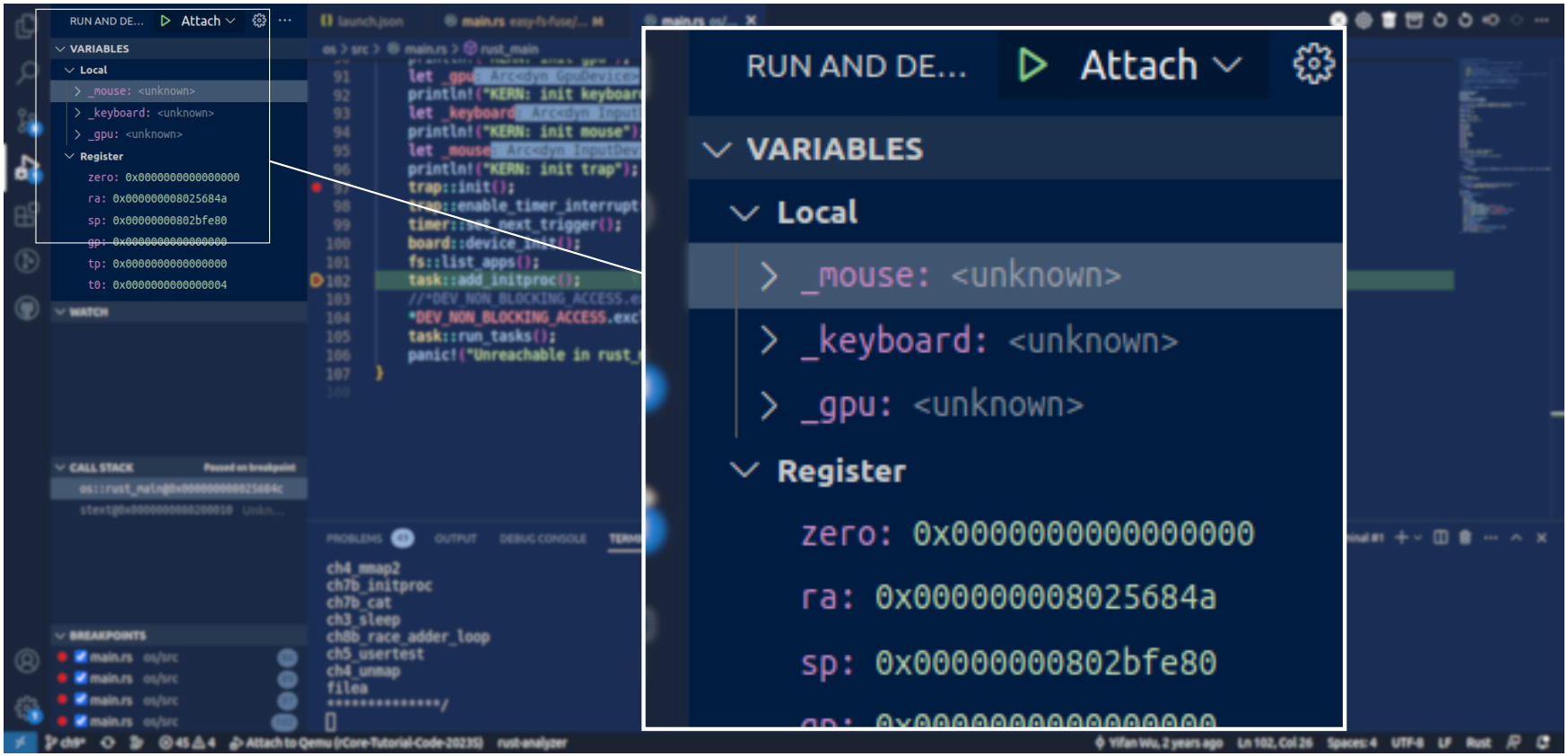
用户界面


let _gpu: Arc<dyn GpuDevice> = GPU_DEVICE.clone();
println!("KERN: init keyboard");
let _keyboard: Arc<dyn InputDevice> = KEYBOARD_DEVICE.clone();
println!("KERN: init mouse");
let _mouse: Arc<dyn InputDevice> = MOUSE_DEVICE.clone();
println!("KERN: init trap");
trap::init();
trap::enable_timer_interrupt();
timer::set_next_trigger();
board::device_init();
fs::list_apps();
ch4_mmap2
ch7b_initproc
ch7b_cat
ch3_sleep
ch8b_race_adder_loop
ch5_usertest
ch4_unmap
filea
**********/

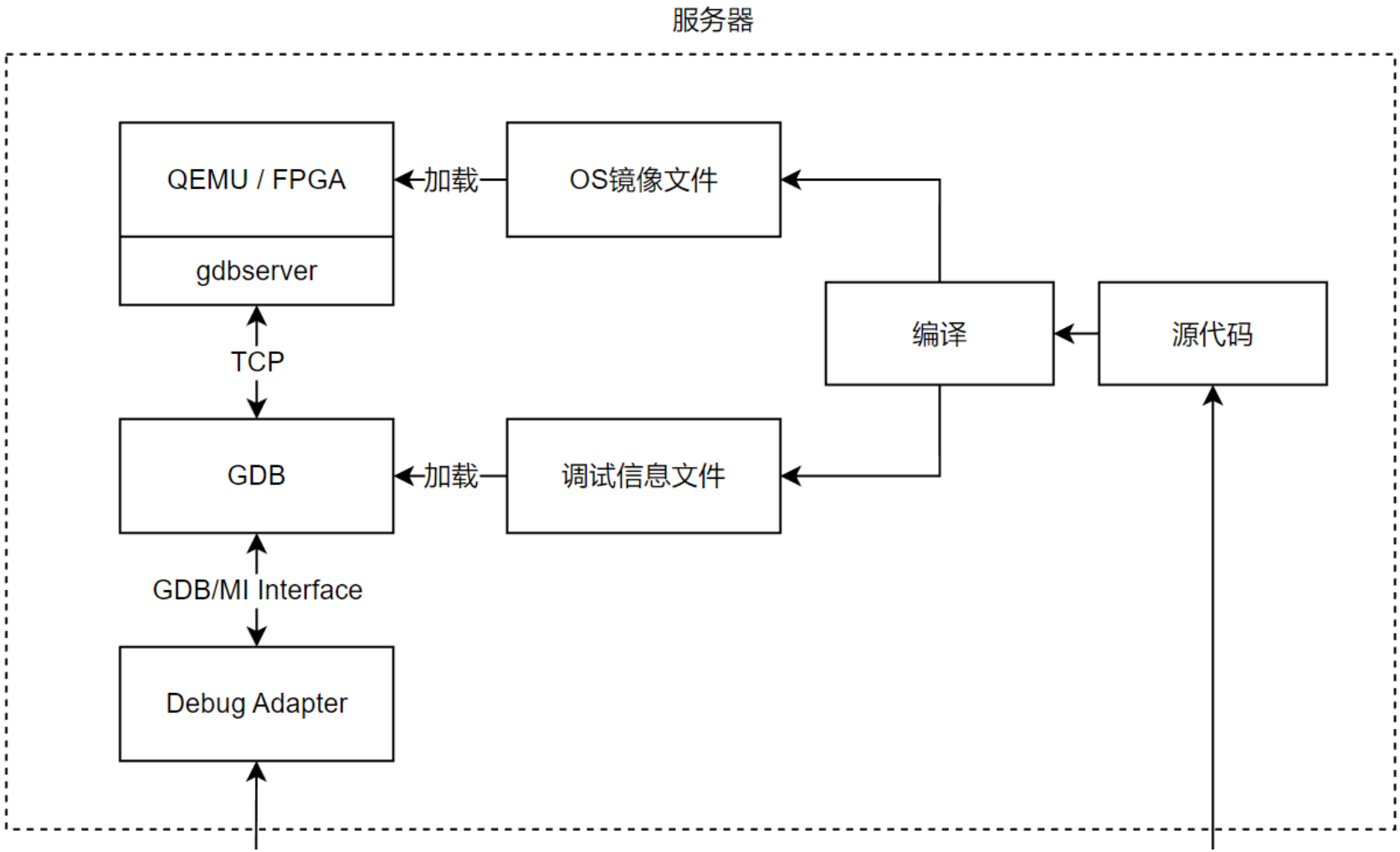
用户界面


let _gpu: Arc<dyn GpuDevice> = GPU_DEVICE.clone();
println!("KERN: init keyboard");
let _keyboard: Arc<dyn InputDevice> = KEYBOARD_DEVICE.clone();
println!("KERN: init mouse");
let _mouse: Arc<dyn InputDevice> = MOUSE_DEVICE.clone();
println!("KERN: init trap");
trap::init();
trap::enable_timer_interrupt();
timer::set_next_trigger();
board::device_init();
fs::list_apps();
task::add_initproc();
ch4_mmap2
ch7b_initproc
ch7b_cat
ch3_sleep
ch8b_race_adder_loop
ch5_usertest
ch4_unmap
filea
**********/

用户界面


RUN AND DE...
Attach
VARIABLES
Local
_mouse: <unknown>
_keyboard: <unknown>
_gpu: <unknown>
Register
zero: 0x0000000000000000
ra: 0x000000008025684a
sp: 0x00000000802bfe80

自动编译、加载内核并启动调试



遇到的问题

GDB 无法设置断点

- 解决办法：修改编译参数
- 在 release 编译模式下保留所有调试信息并关闭编译器的优化

没有操作系统相关的调试功能

- 通过修改 Debug Adapter 来实现内核态用户态切换跟踪，特权级检测等功能

修改编译参数

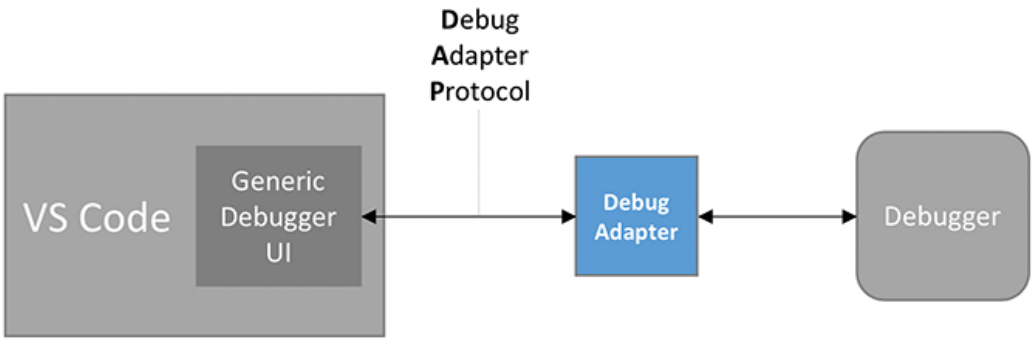
修改编译参数和链接脚本

- ‘debug=true‘ 保留调试信息
- ‘opt-leve1=0‘ 最低优化等级
- ‘linker.ld‘ 保留 *.debug 段

修改后带来的问题

- 应用程序占用的磁盘空间显著增加，导致 easy-fs-fuse（用于将应用程序打包为文件系统镜像）崩溃
  - 因此，需要将磁盘镜像的空间调大
- 用户栈溢出
  - 调整 ‘USER_STACK_SIZE‘ 等参数

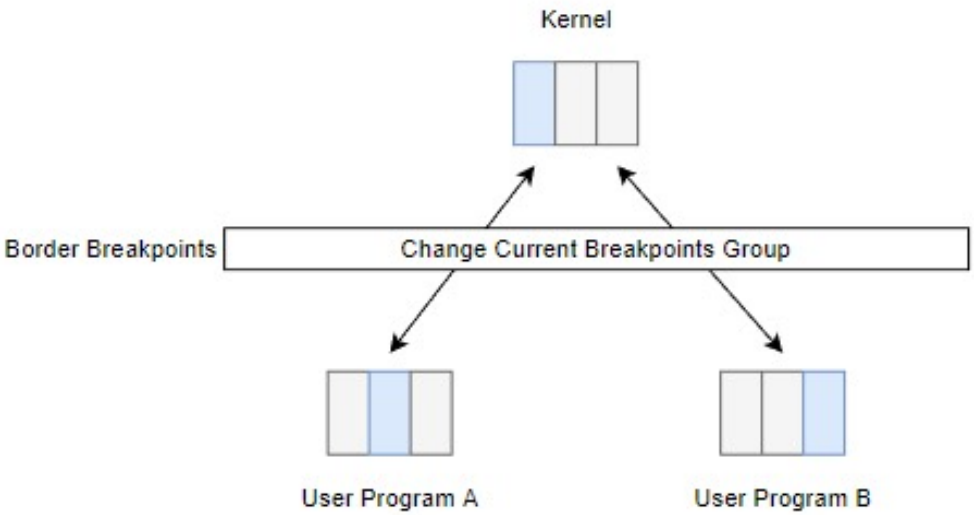
改进 Debug Adapter



需要完善并实现的协议

- Debug Adapter Protocol
- GDB Machine Interface
- Remote Serial Protocol

解决内核态用户态的断点冲突



存在的问题

由于 GDB 限制，无法在内核态设置用户态代码的断点

原因

特权级切换时，TLB 刷新

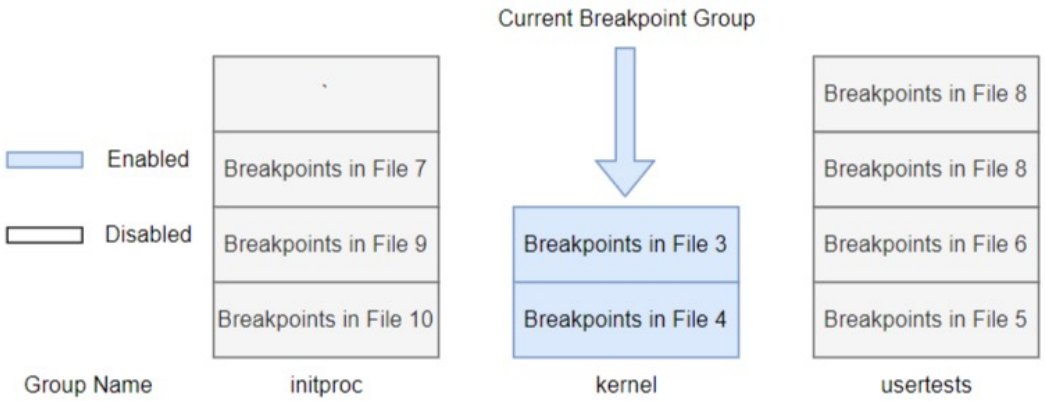
解决方法

暂存，待时机合适再设置

关键问题

暂存断点的策略，恢复断点的时机

断点组暂存机制



1. 分组缓存所有断点的信息
2. 若用户设置的断点不属于当前断点组，不令 GDB 设置
3. 在特权级切换时切换符号表文件、进行断点组切换
4. 这种策略还可以应用于多处理机、多线程、多协程…

如何判断特权级是否已经切换？

RISC-V 处理器的的特殊性

- RISC-V 处理器无寄存器能反映当前特权级，需要用一些变通的办法

“边界断点”机制

- 在特权级切换的代码附近设置断点，若断点触发则特权级将切换
- 这些“边界断点”可以由 Debug Adapter 自动设置，无需用户手动设置
- 同时借助内存地址空间、文件名辅助判断

利用 eBPF 进行跟踪

GDB 的限制

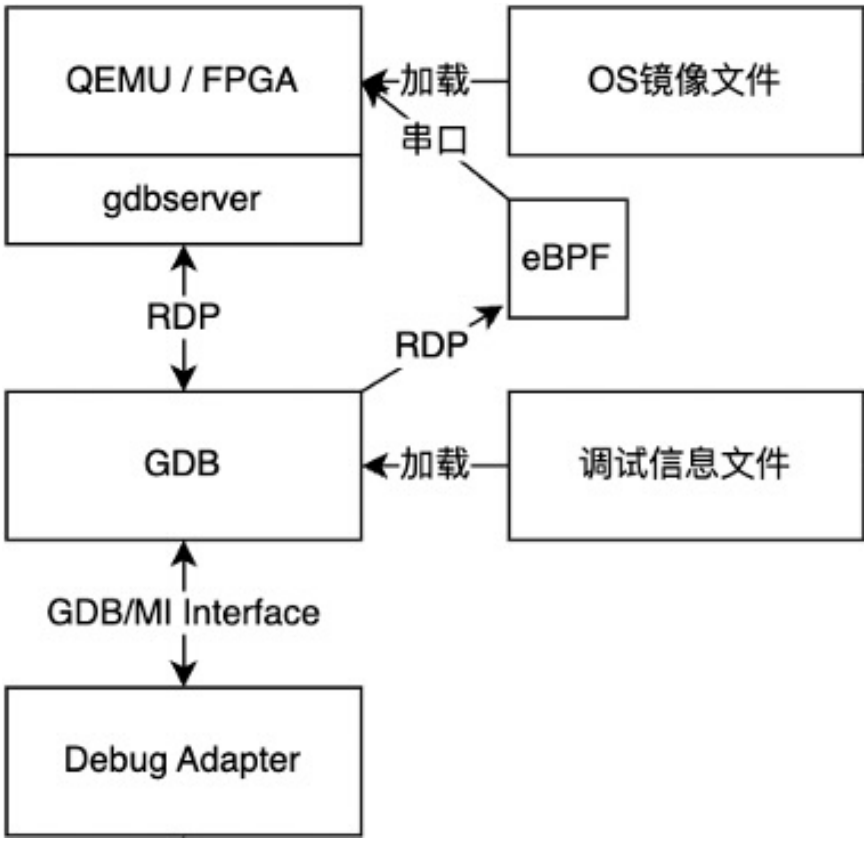
- 为了用上调试器，得改编译参数，改内核代码... 比较繁琐.
- GDB 无法跟踪 rCore 的一些重要的内核数据结构，对 rust 语言的支持也不是特别好.

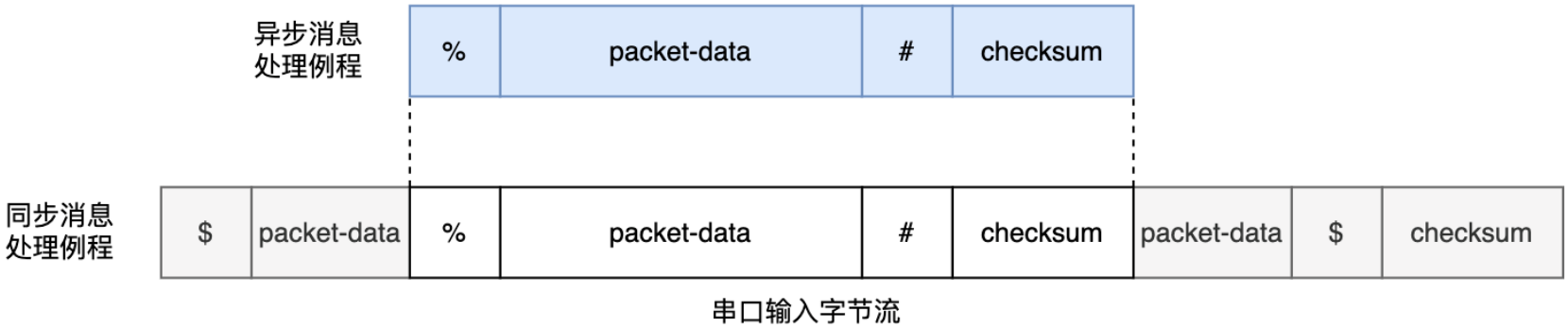
用 eBPF 技术与 gdbserver 同时进行跟踪

- eBPF 技术使得用户可以在内核执行用户自定义的程序，和 Qemu 的 gdbserver 形成很好的互补
- eBPF 程序的可移植性比较好.
- 我们基于 eBPF 实现的功能：函数调用参数获取、内存查看、寄存器查看

实现 eBPF server

1. 移植 eBPF 模块
2. 修改 Qemu 虚拟机代码，支持基于中断的多串口通信
3. 同步数据包即使被异步数据包打断也能正常传输
4. 实现 RSP 协议
5. 在 Debug Adapter 和在线 IDE 中适配 eBPF Server




异步消息
处理例程
%
packet-data
#
checksum
同步消息
处理例程
$
packet-data
%
packet-data
#
checksum
packet-data
$
checksum
串口输入字节流

总结

已完成工作

- 在已有调试器插件的基础上实现基于 Qemu 的 rust 内核在线调试工具
- 支持基于 GDB 的单步断点、内存查看、寄存器查看功能
- 支持基于 eBPF 的断点、内存查看、寄存器查看功能
- 函数调用动态跟踪
- 内核态与用户态方便的切换跟踪

总结

相关链接

- 代码仓库
  - https://github.com/chenzhiy2001/code-debug
- 在线文档
  - https://chenzhiy2001.github.io/code-debug-doc
- 欢迎大家使用我们的调试器并提出意见和建议!

Thanks!