实验1代码分析报告

于新雨 计25 2022010841

程序结构

文件组成

```
supervisor-rv
├── LICENSE
├── README.md
├── kernel
│   ├── Makefile
│   ├── debug
│   ├── include
│   │   ├── common.h
│   │   ├── exception.h
│   │   └── serial.h
│   ├── kern
│   │   ├── evec.S
│   │   ├── init.S
│   │   ├── kernel32.ld
│   │   ├── kernel64.ld
│   │   ├── shell.S
│   │   ├── test.S
│   │   ├── trap.S
│   │   └── utils.S
│   ├── kernel.elf
│   └── obj
│       ├── evec.o
│       ├── init.o
│       ├── shell.o
│       ├── test.o
│       ├── trap.o
│       └── utils.o
├── misc
│   └── cryptonight.cpp
├── obj
│   ├── evec.o
│   ├── init.o
│   ├── shell.o
│   ├── test.o
│   ├── trap.o
│   └── utils.o
└── term
    ├── bytedump
    └── term.py
```

程序结构

term

- `term.py` 为终端程序，用于与内核进行交互，主要流程分为如下步骤：
    - 获取用户输入参数，根据他初始化串口，和 kernel 建立连接
    - 验证所依赖的程序（CMD_ASSEMBLER,CMD_DISASSEMBLER,CMD_BINARY_COPY）路径的有效性
    - 和用户和 kernel 之间进行交互，完成用户输入的指令的解析和执行
- `bytedump` 为一个辅助程序，用于将二进制文件转换为十六进制字符串
- 初始化串口并且建立连接
    - 可以通过 TCP 或者 Serial 两种方式连接，由用户输入参数决定，在后续的操作中可以看到有 inp outp 两个对象用于读写数据，他们都是 TCP 或者 Serial 所建的 socket 等连接通道
- 验证所依赖的程序（CMD_ASSEMBLER,CMD_DISASSEMBLER,CMD_BINARY_COPY）路径的有效性
    - 程序的前缀是 `"riscv64-unknown-elf-"` 或者环境变量中 `'GCCPREFIX'`，CMD_ASSEMBLER,CMD_DISASSEMBLER,CMD_BINARY_COPY 分别对应的程序是 前缀+`as` 前缀+`objdump` 前缀+`objcopy`，通过 `程序 --version` 的方式检测路径是否正确
- 和用户和 kernel 之间进行交互，完成用户输入的指令的解析和执行
    - 通过 `raw_input` 来和用户进行交互，通过 TCP 或者 Serial 通道来和 kernel 进行交互，通过 `inp` 和 `outp` 来读写数据
    - 根据用户输入的指令执行 `run_D`,`run_G`,`run_A` 等函数，进一步获取用户数据和 kernel 交互，具体功能如下：
        - `run_T`: 打印页表信息（如 virtual_addr, physical_addr,访问权限等）
        - `run_A`: 用户可以输入汇编代码，通过 `CMD_ASSEMBLER` 编译为二进制代码写到参数中的地址里面
        - `run_F`: 从文件加载汇编代码，并将汇编后的机器码写入指定的内存地址
        - `run_R`: 读取并打印所有 RISC-V 通用寄存器（x1 到 x31）的值
        - `run_D`: 接收地址，字节数两个参数，打印给定地址处的若干字节
        - `run_U`: 从指定地址读取内存并反汇编指令
        - `run_G`: 从内存中的指定地址开始执行用户代码，此外，还会记录运行时间和 TrapException 并且输出给用户
    - 其中指令的格式更贴近于 windbg 的调试指令，和 gdb 的指令相比略简单一些

kernel

- 入口
    - evec.S: 在 `.text.init` 段中，位于 `0x80000000`，用于跳转到 START 函数初始化内核
- 初始化
    - init.S:
        - bss 段：存放了 Thread Control Block Table（TCBT），当前现场的 tcb 地址
        - data 段：存放了用户和内核代码，用户和内核栈的页表
        - rodata 段： 存放了初始化的时候输出的字符串 "monitor_version"
        - text 段：初始化代码做了以下几件事：

        - 清空 bss 段
        - 设置内核栈 用户栈，具体来说，sp 指向 kernel32.ld 中的 KERNEL_STACK_INIT，用 USER_STACK_INIT 初始化用户态的 sp fp
        - 在内核栈顶设置一个空闲的中断帧，并且保留在 TCBT 中
        - 初始化页表
        - 初始化物理内存访问权限，将所有页设置为可读可写可执行，但是这样在真实世界中是不安全的
        - 准备启动：打印信息标志初始化完成，跳转到 shell

- 交互态：shell
    - 在该程序的 bss 段中保留了所有的用户态寄存器的值
    - 首先从串口读入一个输入符号，并且进行解析，根据解析的结果执行相应的操作，具有如下的功能
    - 打印页表 (OP_T) ，利用了 satp 寄存器含有当前页表的物理地址
    - 打印寄存器：(OP_R) 从 uregs 中读取所有的寄存器的值并且依次输出给用户
    - 写入内存：(OP_A) 获得用户传入的 addr, num 参数，然后依字节写入内存
    - 运行用户输入程序：(OP_G)
        - 首先读取用户程序开始地址，发送 SIG_TIMERSET 信息告诉 term 我们程序开始执行了
        - 然后在 uregs 中保存栈指针，把 bss 段保存的各个寄存器赋值给各个寄存器，把 `.USERRET2` 作为返回地址赋值给 ra 然后跳转到用户程序地址开始执行，由于用户态最后一个指令一定是 `jr ra` 所以我们可以在 `.USERRET2` 这里捕获到用户态的结束，他会把所有当前的寄存器存到 bss 段的 uregs 中，恢复当前 sp,发送停止计时信号给 term 然后回到交互循环

- 异常处理

- trap.S:
    - 在异常处理的时候，会根据 mcause 的值进行不同的处理，比如 mcause 为 `EX_INT_FLAG | EX_INT_MODE_MACHINE | EX_INT_TYPE_TIMER` 时，表示是时钟中断，会进入 HANDLE_TIMER 处理，如果是其他异常，会进入 `HANDLE_INT` `HANDLE_ECALLL` 等处理
    - 在 HANDLE_TIMER 处理中，如果是超时，发送超时信号 `SIG_TIMEOUT` 给 term ,表示用户程序超时
    - 在 EXCEPTION_HANDLER 处理中，会根据 mcause 的值，发送不同的信号给 term，表示用户程序异常结束

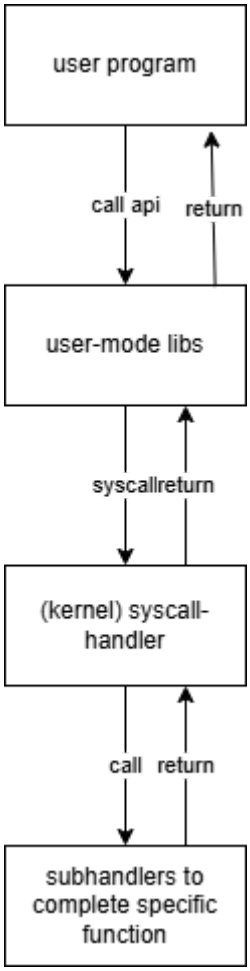
- linker
    - 以 kernel32.ld 为例
    - 定义了内核的起始函数（START），内核栈的起始地址，架构和输出格式等信息
    - 此外，给出了各段的地址和名称等信息

交互方式

控制流：用户态和内核交互

- 在通常的操作系统中，用户态程序和内核交互是以如下方式进行，图中给出的调用栈，以 linux 系统中的 ioctl routine 为例，用户态通过 `ioctl(ioctlcode,arg2,arg3...)` api 进行 ioctl syscall，在调用到的

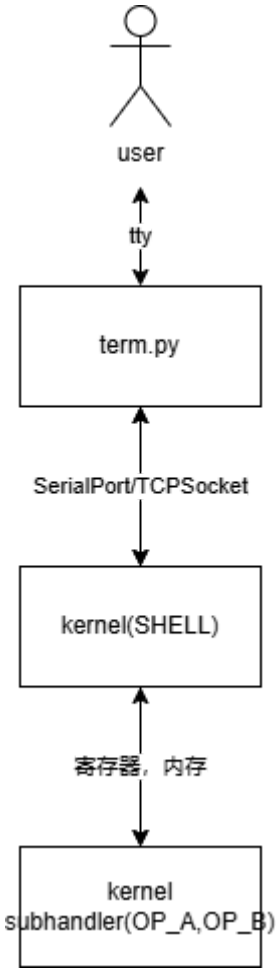
内核模块的 ioctl_handler 函数中，根据 ioctlcode 调用不同的派发函数（subhandler）进行具体功能的处理，最后返回给用户态



- 本实验中，term.py 类似于一个大的用户态 handler，通过 tty 和用户交互，根据用户输入的指令和地址之类的信息判断需要调用的内核 subhandler，然后发送的 'D','A','G' 等指令可以类比为上述的 ioctlcode，内核解析这些发送的字符，调用到对应的 OP_D,OP_A,OP_G 函数，这些可以类比上文 subhandler，在 subhandler 中继续解析其余参数，完成具体的功能，最后返回给用户态
- 但是和 linux 系统不同的是，term.py 和内核之间的交互是通过 TCP 或者 Serial 通道进行的，而不是通过 syscall 进行的，这样的设计使得 term.py 和内核之间的交互不是基于调用关系，而是数据传输关系

数据流

交互如图所示


user
tty
term.py
SerialPort/TCPSocket
kernel(SHELL)
寄存器，内存
kernel
subhandler(OP_A,OP_B)