实验报告

211240045 杨镇源

实验进度：

PA3.1 完成了所有必做内容；
PA3.2 完成了所有必做内容；
PA3.3 完成了所有必做内容。

必做题：

必做题 1：理解上下文结构体的前世今生。

__am_irq_handle(Context *c)中上下文结构指针 c 指向的上下文结构体的定义在 riscv32-nemu.h 中，但是该上下文结构体真正的使用则在在 trap.S 中。

在 trap.s 中定义了 mcause，mstatus 和 mepc 的偏移量，对应着 Context 结构体中定义变量的顺序，随后通过 sw 和 lw 调取 cpu 中控制寄存器的值，即可对上下文结构体成员赋值。

trap.S 中有我们需要实现的新指令，而 riscv32-nemu.h（上下文）会利用 trap.S 中的汇编语句来进行保存现场的操作，然后可以通过实现的新指令进入系统调用。

必做题 2：从 Nanos-lite 调用 yield()开始, 到从 yield()返回的期间, 这一趟旅程具体经历了什么? 软(AM, Nanos-lite)硬(NEMU)件是如何相互协助来完成这趟旅程的?

从 yield()调用开始，NEMU 开始处理 yield()中的内联汇编代码，随后在调用到 ecall 指令之后直接跳转到在 mtvem 存储的地址。

之后调用 trap.s，trap.s 首先开辟了一块栈空间，随后将当前现场的寄存器值全部通过 sw 存储在栈中，随后将三个状态寄存器 mcause,mstatus 和 mepc 存入通用寄存器中，随后根据 AM 中 Context 上下文结构体声明的变量顺序，依次将状态寄存器存入 Context 中。

之后便运行注册的回调函数，其中不同类型的异常会分别产生不同的效果，在此处我们调用的异常类型为 YIELD，随后会调用 do_event()对异常进行处理，之后再次跳回 trap.s 进行现场复原，将之前在栈中存储的 mstatus 和 mepc 的值重新放回控制寄存器中，通用寄存器同理，然后返回。

必做题 3：hello 程序是什么, 它从而何来, 要到哪里去。

hello 程序的 ELF 文件初始时存储在 ramdisk.img 上，其代码、数据等有关信息都在 ELF 文件中。通过 naive_uload()方法将对应信息加载到内存后，即可以

在 riscv32-nemu 架构之下运行。其之所以能加载到相应位置，是因为 ELF 文件中已经做出了相应规定。

第一条指令在__start 函数中，后即读取 entry 的地址，并利用函数跳转至该入口执行目标文件。

第一个 Hello World 通过 write 直接调用系统调用函数 write 向 stdout 输出对应的值，nanos-lite 通过调用 serial_write，serial_write 通过使用 abstract machine 中的 ioe 向屏幕输出。后续的字符串通过 printf 输出，printf 通过 malloc 申请一块缓冲区，而 malloc 会调用系统调用 brk 来进行堆区管理，之后将字符逐一写入缓冲区，而写出换行符或者缓冲区满时会进行刷新，调用 write 向 stdout 输出。

必做题 4：仙剑奇侠传究竟如何运行。

通过阅读 Pal 中的 PAL_SplashScreen()可以得出，mgo.mkf 是通过 PAL_MKFReadChunk(buf,32000,SPRITENUM_SPLASH_CRANE,gpGlobals→f.fpMGO)函数进行解析的，而该函数的使用离不开库函数和系统调用函数 fseek()和 read()等的支持。

这两个库函数离不开 libos 中_syscall_的支持，通过进入系统层，将操作权归还操作系统得以成功调用这两个函数。

然后在 AM 中 call Interrupt/Expection，通过查看系统异常号，在操作系统中使用正确的系统调用函数 SYS_seek()和 SYS_fread()，后进一步调用 fs_seek()，fs_read()，最后通过 trm.c 到底层的 nemu 里进行有关运算。

这样，库函数的文件读取才能顺利跳转至__am_asm_trap，再跳转至__am_irq_handler，进而跳转至 do_event，进行仙鹤像素信息的读取，自后再跳转回原仙剑程序继续执行。

实验心得：我恨新冠 QAQ！！！