说明

PA3 可以从nterm运行仙剑，一切正常！

PA3 必答题

必答题(需要在实验报告中回答) - 理解上下文结构体的前世今生

你会在__am_irq_handle()中看到有一个上下文结构指针c, c指向的上下文结构究竟在哪里? 这个上下文结构又是怎么来的? 具体地, 这个上下文结构有很多成员, 每一个成员究竟在哪里赋值的? $ISA-nemu.h, trap.S, 上述讲义文字, 以及你刚刚在NEMU中实现的新指令, 这四部分内容又有什么联系?

答： c指向的上下文结构在trap.S中，它的首地址存在sp中。它是由trap.S中addi sp，sp，-CONTEXT_SIZE所分配的，每一个成员都是在jal __am_irq_handle指令前赋值的，具体而言，通用寄存器是在MAP步骤宏定义展开并一个个存储，然后再将mcause,mstatus,mepc用STORE和对应宏定义展开的偏移量存储的。$ISA-nemu.h定义了Context的各个内容顺序，让C语言代码中能够以正确顺序访问；trap.S申请空间并存储了各代码，讲义就是将trap.S的过程讲了一遍，而我刚刚在NEMU实现的新指令例如csrr则是trap.S所使用到的指令。

必答题(需要在实验报告中回答) - 理解穿越时空的旅程

从yield test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, yield test)硬(NEMU)件是如何相互协助来完成这趟旅程的? 你需要解释这一过程中的每一处细节, 包括涉及的每一行汇编代码/C代码的行为, 尤其是一些比较关键的指令/变量. 事实上, 上文的必答题"理解上下文结构体的前世今生"已经涵盖了这趟旅程中的一部分, 你可以把它的回答包含进来.

别被"每一行代码"吓到了, 这个过程也就大约50行代码, 要完全理解透彻并不是不可能的. 我们之所以设置这道必答题, 是为了强迫你理解清楚这个过程中的每一处细节. 这一理解是如此重要, 以至于如果你缺少它, 接下来你面对bug几乎是束手无策.

答： 从yield()开始，进入abstract-machine/am/src/riscv/nemu/cte.c的yield()，然后根据宏，运行了asm volatile("li a7, -1; ecall");，此时是软件调用了硬件的指令，其中li a7 -1是将系统调用号设为-1，存储在a7中（即c->GPR1），然后NEMU执行ecall指令。NEMU调用isa_raise_intr()函数，并跳转到预先设定的trap.S的位置，然会执行了"理解上下文结构体的前世今生"的过程，trap.S会调用软件的__am_irq_handle函数中，该函数根据异常的原因以及系统调用号来执行相应的操作，并调用预先设定的user_handler()函数（此处是simple_trap()），最后返回到trap.S中，trap.S最后使用了mret指令，让NEMU跳转到之前ecall时存储的位置，就返回到了yield()的下一条指令。

必答题(需要在实验报告中回答) - hello程序是什么, 它从而何来, 要到哪里去

我们知道navy-apps/tests/hello/hello.c只是一个C源文件, 它会被编译链接成一个ELF文件. 那

么, hello程序一开始在哪里? 它是怎么出现内存中的? 为什么会出现在目前的内存位置? 它的第一条指令在哪里? 究竟是怎么执行到它的第一条指令的? hello程序在不断地打印字符串, 每一个字符又是经历了什么才会最终出现在终端上?

答： 一开始hello程序在内存的ramdisk_start与ramdisk_end之间，它是在一开始运行程序的时候就被NEMU加载到内存中的。在nanos-lite的Makefile中，创建了文件resources.S，该文件指定了ramdisk.img的起始位置。hello的第一条指令在elf文件的表头elf_ehdr.e_entry所指向的位置，loader会返回这个值，然后naive_uload通过((void (*)())entry)();语句，将第一条位置当成一个函数指针来调用，从而开始执行hello。hello打印字符时，系统函数最终会调用_write()，然后通过_syscall()来调用相应的系统调用，与"理解穿越时空的旅程"不同的地方在于设定的user_handler()函数，此处设为了nanos-lite中的函数do_event()，然后该函数调用do_syscall，do_syscall再处理相应的系统调用号，此处即调用了sys_write()，它调用fs_write()，然后调用serial_write()，该函数通过putch()输出，而putch()调用了AM的outb()，从而送到NEMU的输出端口位置，NEMU再将该内容输出。

仙剑奇侠传究竟如何运行

运行仙剑奇侠传时会播放启动动画, 动画里仙鹤在群山中飞过. 这一动画是通过navy-apps/apps/pal/repo/src/main.c中的PAL_SplashScreen()函数播放的. 阅读这一函数, 可以得知仙鹤的像素信息存放在数据文件mgo.mkf中. 请回答以下问题: 库函数, libos, Nanos-lite, AM, NEMU是如何相互协助, 来帮助仙剑奇侠传的代码从mgo.mkf文件中读出仙鹤的像素信息, 并且更新到屏幕上? 换一种PA的经典问法: 这个过程究竟经历了些什么? (Hint: 合理使用各种trace工具, 可以帮助你更容易地理解仙剑奇侠传的行为) **答：** 首先，程序在PAL_MKFReadChunk中调用了fread函数，这是库函数，最终会调用到libos中的_read()系统调用，然后的过程与"hello程序是什么, 它从而何来, 要到哪里去"中_write()的过程基本是一样的，此处不再赘述，总之就是将mgo.mkf读取到了相应的内存中。