PA3 实验报告

191240046 孙博文

完成情况：完成所有必做题，通过Online-Judging的所有样例。

必答题 - 理解计算机系统

理解上下文结构体的前世今生

- 上下文结构指针`c`在哪里？是怎么来的？在哪里赋值？
  - 首先，`int`指令将`eflags`/`cs`/`eip`压栈，作为`c->eflags`/`c->cs`/`c->eip`成员
  - 之后，通过设置好的异常入口地址，根据异常号分别跳转到`trap.S`中的`__am_vecsys`/`__am_vectrap`/`__am_irq0`/`__am_vecnull`函数，分别将各自的异常号压栈，成为`c->irq`成员
  - 这四个函数都跳转到`__am_asm_trap`函数，通过`pushal`将通用寄存器压栈，压入`0`作为`c->cr3`，最后向栈中压入`esp`作为结构体指针`c`的地址，作为参数传给`__am_irq_handle(Context *c)`函数
- 总之，上下文结构体是在栈上构造的。只要按照压栈的顺序组织结构体，就可以正确地解读出上下文结构体的各个成员的内容。

理解穿越时空的旅程

- 从调用`yield()`开始：
- 首先执行内联汇编的`int $0x81`指令，指令通过异常号和`LDTR`寄存器的值寻找到对应的IDT门描述符，从中组合出到跳转的pc地址，进行跳转。
- 跳转到`trap.S`中的`__am_vecsys`/`__am_vectrap`/`__am_irq0`/`__am_vecnull`， 再跳转到`__am_asm_trap`函数，正确地组织好上下文结构体
- 调用`__am_irq_handle`，最终会调用在nanos_lite中，`init_irq()`调用的`cte_init()`中注册的操作系统的回调函数，然后返回一个上下文结构体指针`c`。
- 回到`__am_irq_handle`， 返回到`__am_asm_trap`， 通过一系列`pop`，然后执行`iret`，将栈上的上下文结构体恢复到寄存器中。
- 跳转回进入时候的pc地址。

hello程序是什么，它从何而来，要到哪里去

- hello程序一开始是以二进制文件的格式存储在了磁盘的`ramdisk.img`当中。
- 在编译的时候，通过`resources.S`中的`.incbin "build/ramdisk.img"`将它链接到了`nanos-lite-x86-nemu.bin`的`.data`节。
- 在nemu运行时，ramdisk的数据都被存储在了内存盘当中。在实现文件系统之前，只有一个文件，于是可以直接从ramdisk的开头读取程序；对于实现文件系统之后，则根据文件系统中的文件表，找到相应的文件名，盘内偏移量以及程序大小。
- 通过`loader`函数将内存盘的文件的程序段加载到内存当中的指定位置。

- 程序的头表指定了程序的入口地址`_start()`，我们将程序的入口地址作为一个函数指针，并调用这个函数，就可以进入。随后将调用`call_main()`函数，然后进入`main()`。
- 打印字符经历了什么：
  - 程序调用`printf()`函数，通过对格式字符串`fmt`的翻译得到一个纯字符串输出。
  - 将字符串的地址作为参数，调用`_write()`函数，并最终调用`_syscall_(SYS_write, fd, buf, count)`，在寄存器中设置好参数，通过`int 0x80`向串口这个抽象文件中进行写入的系统调用。
  - 进入系统调用的中断处理程序，再调用操作系统的系统事件回调函数，进行系统调用的事件分发，调用`do_syscall()`，这最终会调用`serial_write()`函数。它调用`abstract machine`中封装的`putch`，逐字输出字符串。`putch()`函数会通过`outb(SERIAL_PORT, ch)`中内联的汇编`outb`指令，让cpu向串口的端口输出字符。

仙剑奇侠传究竟如何运行

- 首先在`PAL_Init_Globals()`函数中调用`UTIL_OpenRequiredFile()`，最终通过系统调用打开并读取`mgo.mkf`，并将数据保存在`gpGlobals->f.fpMGO`。`fopen()`和`fread（）`最终会通过系统调用nanos-lite中的`fs_open()`/`fs_read()`函数，从ramdisk中找到相应文件并读取。
- 申请一段内存空间，将`lpSpriteCrane`，即仙鹤的像素信息，设置在这段内存的某个位置。
- 通过`PAL_MKFReadChunk()`将`f.fpMGO`中仙鹤的像素信息读取到`lpSpriteCrane`中。
- 将仙鹤的像素信息通过`SDL_BlitSurface()`拷贝到屏幕画布上的对应位置。
- 通过`SDL_UpdateRect()`将当前的屏幕信息输出。它将先使用调色板，将8位颜色转换为32位颜色，然后调用`ConvertPixelsARGB_ABGR()`将图像的颜色设置正确。然后调用`NDL_DrawRect()`，进行`write()`系统调用，将像素的内容写入到显存的抽象文件中。
- 这最终会调用nanos-lite中的`fb_write()`函数。它将通过调用abstract-machine中的`io_write(AM_GPU_FBDRAW)`， 将参数组织为结构体`AM_GPU_FBDRAW_T ctl`，然后调用`__am_gpu_fbdraw()`向显存中的像素对应位置写入像素的颜色。
- out指令向`SYNC_ADDR`输出1，表示屏幕更新