PA4 实验报告

张运吉（211300063、211300063@ smail.nju.edu.cn）

(南京大学人工智能学院, 南京 210093)

1 实验进度

我已经完成 **PA4 全部必做内容。**

2 必答题

2.1 分时多任务的具体过程

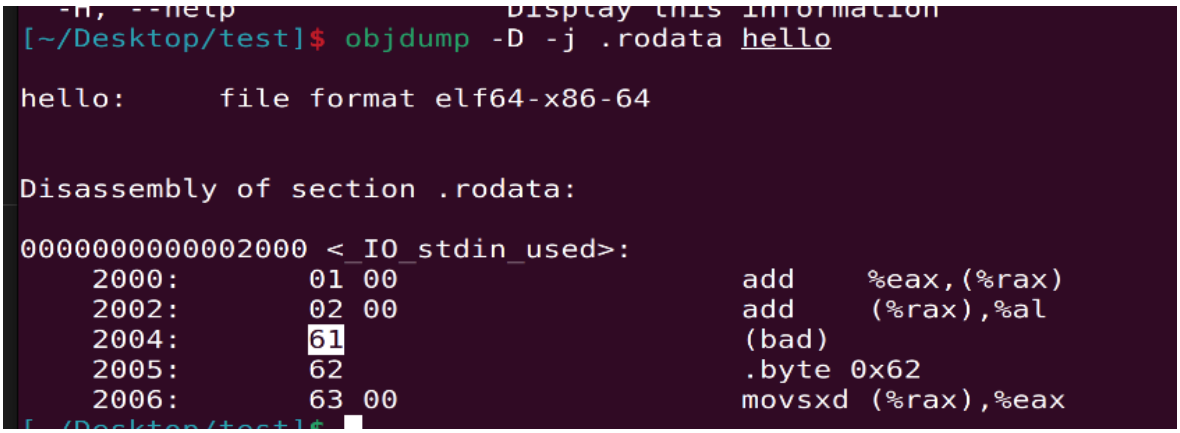
在 nanos-lite 的 main 函数中，调用 init_proc()的时候会创建 pal 和 hello 两个用户进程，具体地，通过 loader()函数把这两个程序对应的二进制文件加载到 ramdisk 中，加载的时候是以页为单位的，物理页是按照顺序在物理内存中申请，通过 map()函数建立虚拟地址到物理地址的映射 ，在 nemu 中实现了分页机制，抽象成 isa_mmu_check()和 isa_mmu_translate()函数，以后在访问用户进程对应的虚拟地址空间时，nemu 会把虚拟地址 vaddr 转化成物理地址 paddr，就这样使得 pal 和 hello 运行在分页机制上。

假设 pcb[0]对应进程是 pal，pcb[1]对应是 hello，在 nanos-lite 加载好两个进程之后，会通过 yield()进入 cte（AM）,进程控制块选择 pal 进程运行，在运行每条指令后，nemu 都会检查有没有时钟中断，如果有并且处于开中断的状态，那么会调用 isa_raise_intr()函数，这个函数最后返回异常入口地址，pc 指向这个地址，也就是调用 trap.S 中的_am_asm_trap 函数，这个函数保存当前进程的上下文，然后调用_am_irq_handle 函数，返回另一个进程的上下文，然后在_am_asm_trap 函数恢复这个进程的上下文，转而执行这个进程，这样就实现了 pal 和 hello 的分时运行。

2.2 理解计算机系统

指针 p 指向字符串常量，字符串常量存储在内存只读数据段（elf 的.rodata 节），当程序尝试对只读数据段的数据修改时，mmu 在对虚拟地址转换的过程会捕获这个非法操作，linux 操作系统会发出一个 SIGSEGV 的信号，进程收到这个信号后就打印 Segmentation fault 的错误信息。

使用 objdump 查看反汇编：



可以看到有连续四个字节 61 62 63 00（十六进制）,也就是字符串“abc”，00 代表字符串结束符。

然后使用 gdb 单步调试：

```
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from hello...
(gdb) si
The program is not being run.
(gdb) run
Starting program: /home/iuzyj/Desktop/test/hello
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Program received signal SIGSEGV, Segmentation fault.
0x0000555555555140 in main () at /home/iuzyj/Desktop/test/hello.c:3
3           p[0] = 'A';
(gdb) si

Program terminated with signal SIGSEGV, Segmentation fault.
The program no longer exists.
(gdb)
```

可以看到，进程收到一个 SIGSEGV 信号，然后触发了段错误。

3 实验过程

3.1 pa4.1

第一个必做并不是太难，参考一下 native 的代码就可以写出来，但是我在实现带参数的 context_uload 函数时遇到了困难，首先就是由于没理解讲义，导致在往栈上存参数和环境变量的时候出了很多错误，然后 busybox 的应用一直运行不起来，后来重新理解讲义，才发现是栈的生长方向弄反了，重新实现 context_uload 后就可以正确运行 busybox 了。

3.2 pa4.2

pa4.2 感觉是 pa 最难的部分了，首先看讲义理解以下分页的大致流程，然后查看 riscv 的手册了解 riscv 实现分页机制的细节，这些准备工作就花了大概两天时间，实现 map 的时候，需要注意的就是当一个进程第一次被加载的时候, 内核都会为它把页目录的那张表分配好, 并且把其物理地址存进这个进程的 as->ptr 里面, 这个就是它的页表, 这是一个两级页表, 第一级我们叫页目录, 第二级叫页表,它们结构一模一样,但是页目录的条目并不一定是有效的,拿到一个虚拟地址, 要先计算出它在页目录中的索引, 然后获得对应的 PDE, 但是 valid 位如果为 0, 我们就需要自己分配一个二级页表, 并且建立页目录与二级页表的映射, 然后再在二级页表中建立真正的物理地址与虚拟地址的映射。

正确实现 map 之后，在 nemu 中添加分页机制就差不多是一样的了，isa_mmu_translate 的功能就是把 va 装化成 pa，这和 map 函数的操作差不多。

3.3 pa4.3

第三部分是分时多任务，这部分讲义讲的比前两部分清楚，而且讲义给了伪代码，只需要在 cte 实现这些伪代码的功能即可，但是理解讲义的内容也是不容易的，我大概花了一天时间才弄清楚栈切换的原理，我是在 trap.S 这个文件中实现全部的功能的，具体地，我使用一个 t3 寄存器维护 c->np 这个变量，唯一需要注意的点就是讲义上说的“为了维护 c->np, 你可能需要在切换到内核栈之后马上保存少部分 GPR, 或者借助全局变量来帮助你”，这一部分比较痛苦的就是 debug 了，这里我无法使用输出调试法，只能配合生成的反汇编文件自己一步一步推演。

4 实验心得

PA 终于结束了，回顾整个 pa，从 pa0 搭环境时的一脸懵逼，到逐渐掌握 pa 的正确写法，这一过程我学到的不只有计算机系统的底层知识，还有一些和编程有关的技巧，比如基础设施的使用，一些工具的合理使用，都会使我们的编程事半功倍，还有就是死磕代码的毅力了吧，很多次 pa 写到心态爆炸，debug 没有效果，最后也坚持下来了。看到一些小程序能在 nemu 上跑出来的时候，还是有满满的成就感的。