PA4 实验报告

王卫东　221900332

2024 年 1 月 21 日

一　展示内容

完成了 pa4 的所有内容，在进程中加载了 3 个 nterm 和一个 hello 程序，支持通过 F1,F2,F3 进行用户程序不同 nterm 的切换。在 nterm 中可以键入”pal –skip”,”bird”(可能要等一会),”nslider” 和”echo xxxx” 来进入相应进程并打印信息。

二　必做题

1　分时多任务的具体过程

1.1　请结合代码, 解释分页机制和硬件中断是如何支撑仙剑奇侠传和 hello 程序在我们的计算机系统 (Nanos-lite, AM, NEMU) 中分时运行的.

从 hello 切换到 pal 的过程：

1. 每一条指令执行结束时,nemu 的 cpu_exec() 中都会调用 timer_interrupt() 函数来检查 CPU 的 INTR 引脚有没有被拉高。如果被拉高了说明有设备发出了中断请求（在 PA 中只有时钟中断请求），此时 CPU 会将 mstatus 寄存器的 MIE 位置零（防止中断嵌套），然后进入 nanos-lite 的中断/异常处理程序 trap.S。

2. 在 trap.S 中，nanos-lite 首先会将 sp 指针切换到 hello 用户进程的内核栈，在内核栈上保存 hello 进程的上下文，将 mscratch 和 c→np 等信息设置好之后，进入 ___am_irq_handle() 函数。

3. 在 ___am_irq_handle() 函数中，nanos-lite 发现 mcause 寄存器中的事件编号是 0x80000007(riscv)，于是标记该事件为时钟中断，调用 do_event() 函数。do_event() 函数中调用了 shedule() 函数，schedule() 函数将 current→ cp 指向刚才保存在 hello 内核栈上的 hello 上下文，将 current 指针指向 pal 进程的 pcb，并返回 pal 进程内核栈上的上下文结构体。

4. ___am_irq_handle() 函数最终会返回 pal 进程的上下文结构体。回到 trap.S 中以后，nanos-lite 会恢复 pal 进程的上下文，并且根据 c→np 将栈指针从 pal 进程的内核栈移到 pal 进程的用户栈。执行完 mret 指令之后，nanos-lite 就切换到了 pal 的代码继续执行。

2 理解计算机系统

2.1 尝试在 Linux 中编写并运行以下程序，你会看到程序因为往只读字符串进行写入而触发了段错误. 请你根据学习的知识和工具，从程序，编译器，链接器，运行时环境，操作系统和硬件等视角分析" 字符串的写保护机制是如何实现的". 换句话说，上述程序在执行 p[0] = 'A' 的时候，计算机系统究竟发生了什么而引发段错误？计算机系统又是如何保证段错误会发生？如何使用合适的工具来证明你的想法？

字符串 "abc" 在编译的时候会被放入 ELF 文件的只读数据段，并在链接运行的时候位于只读代码段。对应的汇编代码为：

```
0000000000000000 <main>:
 0: f3 0f 1e fa endbr64
 4: 55 push %rbp
 5: 48 89 e5 mov %rsp,%rbp
 8: 48 8d 05 00 00 00 00 lea 0x0(%rip),%rax # f <main+0xf>
 f: 48 89 45 f8 mov %rax,-0x8(%rbp)
13: 48 8b 45 f8 mov -0x8(%rbp),%rax
17: c6 00 41 movb $0x41,(%rax) # p[0] = 'A'
1a: b8 00 00 00 00 mov $0x0,%eax
1f: 5d pop %rbp
20: c3 ret
```

执行到 <main+0x17> 这条指令时，硬件中的 MMU 会检查当前进程的权限是否可以访问这一内存区域。由于 "abc" 是只读数据，所以发生了越权行为。访问非法地址后 MMU 会修改一些特权寄存器的值记录信息，然后

转到操作系统的异常处理程序。操作系统会向用户进程发送一个 SIGSEGV 信号，表示发生了段错误。

我们可以通过 gdb 来检测 SIGSEGV 信号：在 gdb 中使用 `layout asm` 打开汇编代码，一条一条执行，在执行到 movb 语句时会显示 SIGSEGV 信号。

为了看到操作系统执行该程序时经历的过程，我们可以用 strace 工具来查看执行./a.out 时使用的所有系统调用。

部分内容：

```
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0
    x7fe701032000
arch_prctl(ARCH_SET_FS, 0x7fe701221580) = 0
mprotect(0x7fe701211000, 12288, PROT_READ) = 0
mprotect(0x555ca947c000, 4096, PROT_READ) = 0
mprotect(0x7fe701267000, 8192, PROT_READ) = 0
munmap(0x7fe701222000, 76978) = 0
rt_sigaction(SIGHUP, {sa_handler=0x555ca947a189, sa_mask=[HUP], sa_flags=
    SA_RESTORER|SA_RESTART, sa_restorer=0x7fe701075040}, {sa_handler=SIG_DFL,
    sa_mask=[], sa_flags=0}, 8) = 0
--- SIGSEGV {si_signo=SIGSEGV, si_code=SEGV_ACCERR, si_addr=0x555ca947b01c} ---
+++ killed by SIGSEGV (core dumped) +++
Segmentation fault (core dumped)
```

这里系统调用先用 execve 执行 a.out, 中间是 mmap 系统调用，最后发出了 SIGSEGV 信号，并给出了触发该信号的指令地址。

三　实验心得

pa4 应该算是做的时间跨度最久的一个实验（从考完期末开始做），中间很不幸的生病了半周没写（于是就发现快到 ddl 了赶紧 push 进度）。

在做一阶段的时候发现在 native 和 nemu 上运行 nterm 没法正常切 pal/传入 “–skip” 的参数，发现了两个 bug：一个是非常离谱的 strcpy 没有加 “\0” 传不了参数，另一个是在 context_uload 中应该先保存参数信息再加载新进程，否则会将这一段覆盖掉。

做二阶段的时候处理最久的 bug 是 sys_brk 没有改为 mm_brk，（真有点无语），打印了很久的测试信息发现 malloc 出错误了，最后想到堆区申请根本没有进入... 写完分页的 loader 之后只能正常运行 dummy 和 hello（可能和用户栈有关系），但是急着拿分就先去写时钟中断，写完之后就发现

所有的用户程序都可以正常运行了。(why...?)

三阶段的用户栈切换感觉是整个 pa 里面为数不多的很坑的地方，容易翻车的地方还蛮多的，这里就不一一列举了。

总而言之，pa4 的调试确实和之前的不是一个量级的，需要对错误进行定位再慢慢打断点调试，要照着 ABI 实现每个函数的功能（照着自己想很容易 G），不过最终实现的多进程还是蛮有成就感的（(

纵观整个 pa 进程，学到最多的就是读手册，看要求，和不断地学习，至少有学到变得耐心而且出 bug 不慌（?），还是有激发自己的学习热情的(不多也不少，但是真的薄纱拔尖班的破烂问题求解)。我觉得 pa 确实是很厉害的一个学习项目，ics 可以算是目前来说上过最好的课了 qaq。希望能继续保持前进，开年的 os 课继续加油啊 ~