PA4 实验报告

211240001 潘昕田

2023 年 1 月 13 日

1 实验进度

1. PA4 阶段 1 完成
2. PA4 阶段 2 完成
3. PA4 阶段 3 完成

2 必答题

2.1 分时多任务的具体过程

分页机制通过 vme 的 map 函数对虚拟地址和具体物理地址的映射，同时 cte 返回时会设置 satp 寄存器的值，而不同进程 map 的物理地址不同，页目录基地址也不相同，因此 nemu 根据 satp 寄存器的值查询页目录，保证不同进程的地址空间访问的正确性，从而保证 hello 和仙剑奇侠传程序的地址空间在有重合的情况下依然能够正常运行。

而硬件中断则是通过每 10ms 强制触发异常，am 的 cte 保存上下文后转至 nanos-lite 处理，nanos-lite 的 schedule 函数决定下一时间片调度的进程，从而保证了 hello 和仙剑奇侠传两个进程都能得以进行，并且由于时间间隔短，可以制造出两个进程同时进行的假象。

2.2 理解计算机系统

首先，操作系统在装载图示进程时，会将字符串常量所在的节 (.rodata) 设置为不可写，而在执行对其的写操作时，硬件 mmu 会翻译出该段不可写，进而触发 mmu 异常，mmu 异常被操作系统识别后，将通过 SIGSEGV 信

号 kill 进程，而 SIGSEGV 信号是段错误信号，从而使得进程触发段错误后退出。对于 Linux 而言，段错误的触发通过硬件的相关机制以及 Linux 操作系统的信号进制来保证段错误的触发。可以通过 gdb 或者 strace 观察到 SIGSEGV 信号发送至程序，而 man 2 signal 则可以查询该信号触发的具体原因。

3 思考题

3.1 为什么需要使用不同的栈空间

如果使用相同的栈空间，会导致不同进程在栈上保存的数据相会覆盖，会导致后续进程的行为成为 UB。

3.2 一山不能藏二虎?

共享了部分的物理地址空间，而现在进程依然是直接访问真实物理地址。

3.3 理解分页细节

首先，低 12 位是页内偏移量，不需要存储。其次，表项必须是物理地址，因为虚拟地址空间必须映射到确定的物理空间中，这一转换过程是通过真实存储在物理地址空间中的页表所完成的，如果页表使用虚拟地址，则根本无法完成页表的任务。最后，一级页表缺陷在于页表过大，可能根本无法在内存中存储。

3.4 灾难性的后果

会导致上下文被不断覆盖，导致无法正常从中断中恢复第一次进入中断的状态。

4 实验心得

这次实验的 debug 是我整个 pa 过程中最痛苦的一次，为了查出 vme test 无法通过的原因，我连续查了大约一周的 bug。

第一个查出的 bug 是 pa4 关于 vme 关于时钟中断状态的设置，由于直接设置在 MIE 位，导致一旦 mstatus 被恢复就会立即进入开中断状态，进而导致中断嵌套。后将其在初始化时设置到 MPIE 解决。

第二个查出的 bug 是传参规则不符合 ABI 的约定，而是使用了自己的规约，自觉没有可移植性，于是重新修改了传参的设置。

第三个查出的 bug 是加载程序的 bug，会错误将部分页面加载到先前的页面中，导致在跨越边界时出现缺页错误，于是修改了 loader 程序。

第四个查出的 bug 是 pa3 的文件读程序的边界问题（会多读一位），后来通过修订边界附近的读写解决了问题。

然而，修改了上述错误乃至其它一些细节问题均无法通过 OJ 评测，几近绝望之下，重读了 pa4.2 的讲义，突然发现，讲义关于地址空间的描述是 $[\text{start}, \text{end})$，而我在 ucontext 以及 kcontext 中约定的都是 $[\text{start}, \text{end}]$，于是修改了相关边界，结果便通过了测试。

在设计时，由于 RISCV 将 0 寄存器设置为 hardwired zero，即其恒为 0, 不必在 cte 中保存和恢复其值，于是我把 c->np 映射到了 c->gpr[0]（0 代表 Kernel，1 代表 User），从而未在 Context 结构体中增加 np 变量。

总的来说，PA4 虽然最后的 debug 流程极为折磨人，但是整个完成实验的过程以及从中发现的错误还是让我对于整个计算机系统有了十分深入的理解，回顾整个 PA 的流程，自己从硬件的模拟，逐步完善一个简单计算机系统的基础设施乃至可以运行一个简单的操作系统并运行仙剑奇侠传这样较为大型的游戏，不仅收获了大量的知识与能力，同时也使得自己的计算机系统知识更加充实与扎实，为后续操作系统的学习打下了一个好基础。

P.S. 我觉得 PA4.3 关于内核栈切换的部分比 PA4.2 要难。