实验二报告

实现的功能

- 实现了在任务的地址空间中分配和释放指定大小的虚存,并完成了fn sys mmap和fn sys munmap
- · 修改 Ch3 中的相关方法以及 fn sys get time, 能对内核空间中直接访问的字节数组切片操作。
- 完善系统调用方法

问答题

1. 请列举 SV39 页表页表项的组成,描述其中的标志位有何作用?

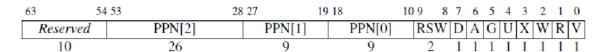


Figure 1: 来自 RISC-V-Reader-Chinese

- V 位决定了该页表项的其余部分是否**有效** (V = 1) 时有效)。若 V = 0,则任何遍历到此页表项的虚址转换操作都会导致页错误。
- R、W 和 X 位分别表示此页是否可以**读取、写入和执行**。如果这三个位都是 0, 那么这个页表项是指向下一级页表的指针,否则它是页表树的一个叶节点。
- U 位表示该页是否是**用户页面**。若 U = 0,则 U 模式不能访问此页面,但 S 模式可以。若 U = 1,则 U 模式下能访问这个页面,而 S 模式不能。
- G 位表示这个映射是否对所有虚址空间有效,硬件可以用这个信息来提高地址转换的性能。这一位通常只用于属于操作系统的页面。
- A 位表示自从上次 A 位被清除以来,该页面是否被访问过。
- D 位表示自从上次清除 D 位以来页面是否被弄脏(例如被写入)。
- 2. 缺页指的是进程访问页面时页面不在页表中或在页表中无效的现象,此时 MMU 将会返回一个中断,告知 os 进程内存访问出了问题。os 选择填补页表并重新执行异常指令或者杀死进程。
 - 请问哪些异常可能是缺页导致的?

Load|Store|Instruction PageFault

• 发生缺页时,描述相关重要寄存器的值,上次实验描述过的可以简略。

缺页有两个常见的原因,其一是 Lazy 策略,也就是直到内存页面被访问才实际进行页表操作。比如,一个程序被执行时,进程的代码段理论上需要从磁盘加载到内存。但是 os 并不会马上这样做, 而是会保存 .text 段在磁盘的位置信息,在这些代码第一次被执行时才完成从磁盘的加载操作。

・ 这样做有哪些好处?

延迟页面的加载, 节省 I0 操作和物理内存

其实,我们的 mmap 也可以采取 Lazy 策略,比如:一个用户进程先后申请了 106 的内存空间, 然后用了其中 1M 就直接退出了。按照现在的做法,我们显然亏大了,进行了很多没有意义的页表操作。

・ 处理 10G 连续的内存页面,对应的 SV39 页表大致占用多少内存 (估算数量级即可)?

GB÷KB (页大小)*B (页表)≈MB

- 请简单思考如何才能实现 Lazy 策略,缺页时又如何处理? 描述合理即可,不需要考虑实现。
 - ▶ 在程序申请内存时,只分配虚拟内存页表,不分配物理页面
 - ▶ 当访问到未分配的页面时, 出发缺页
 - · 在缺页处理中分配物理页面并将需要的数据加载到内存中

缺页的另一个常见原因是 swap 策略,也就是内存页面可能被换到磁盘上了,导致对应页面失效。

• 此时页面失效如何表现在页表项(PTE)上?

V 位为 0

3. 双页表与单页表

为了防范侧信道攻击,我们的 os 使用了双页表。但是传统的设计一直是单页表的,也就是说, 用户线程和对应的内核线程共用同一张页表,只不过内核对应的地址只允许在内核态访问。 (备注: 这里的单/双的说法仅为自创的通俗说法,并无这个名词概念,详情见 KPTI)

・ 在单页表情况下, 如何更换页表?

不需要更换页表?

• 单页表情况下,如何控制用户态无法访问内核页面?

U 位为 0

- ・ 单页表有何优势?
- 双页表实现下,何时需要更换页表?假设你写一个单页表操作系统,你会选择何时更换页表? 上下文切换时

荣誉准则

1. 在完成本次实验的过程(含此前学习的过程)中,我曾分别与 以下各位 就(与本次实验相关的)以下方面做过交流,还在代码中对应的位置以注释形式记录了具体的交流对象及内容:

无

- 2. 此外,我也参考了 以下资料 ,还在代码中对应的位置以注释形式记录了具体的参考来源及内容: 无
- **3.** 我独立完成了本次实验除以上方面之外的所有工作,包括代码与文档。 我清楚地知道,从以上方面获得的信息在一定程度上降低了实验难度,可能会影响起评分。
- 4. 我从未使用过他人的代码,不管是原封不动地复制,还是经过了某些等价转换。 我未曾也不会向他人(含此后各届同学)复制或公开我的实验代码,我有义务妥善保管好它们。 我提交至本实验的评测系统的代码,均无意于破坏或妨碍任何计算机系统的正常运转。 我清楚地知道,以上情况均为本课程纪律所禁止,若违反,对应的实验成绩将按"-100"分计。