ch4实验报告

设计分析与实现

mmap

按照题目意思限制 `_start`，`_port` 的参数

把传入的虚拟地址空间用`MemorySet.translate`转换物理地址来判断是否冲突

依据`_port`的位来构造`MapPermission`传入`insert_framed_area`,并且加上`MapPermission::U`确保内存页是用户模式可访问的

问答题

SV39 页表页表项的组成

页表项的数据结构抽象与类型定义

63 54	53 28	27 19	18 10	9 8	7	6	5	4	3	2	1	0
Reserved	PPN[2]	PPN[1]	PPN[0]	RSW	D	A	G	U	X	W	R	V
10	26	9	9	2	1	1	1	1	1	1	1	1

ppn是物理页

[54:63] 保留字段 没用到

V: pte是否合法

RWX: 读写执行属性

U: 用户态可否访问

G:

A: A(Accessed) 记录自从页表项上的这一位被清零之后，页表项的对应虚拟页面是否被访问过

D: D(Dirty) 则记录自从页表项上的这一位被清零之后，页表项的对应虚拟页表是否被修改过。

缺页

- 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?

    答:

    Instruction page fault

    Store/AMO page fault

    Load page fault
  - 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。

    答:

    sval: 发生缺页时处理器访问的地址

    sepc: 发生缺页时的pc

- 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?

  答:

  app里有些代码是根本不会被执行到的,这样能减少内存使用,还能提升app的加载速度

- 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 10G 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)?

    答:

    2621440个4k页面
    5120个2m页面
    10个1g页面

    10737418240+26214408+51208+10*8 = 10758430800
  - 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。

    答:

    mmap函数执行的时候只是向地址空间预留一段内存,并且向进程结构体中记录这段内存
    当指令第一次访问这块内存的时候将会触发缺页异常,在缺页异常的时机里查找访问的虚拟地址是否记录在进程结构体中,如果存在,则对该页分配内存,中断返回之后指令应该不会触发缺页

- 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?

    答: V位清0,ppn描述换页的信息

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？

  修改`satp`寄存器

- 单页表情况下，如何控制用户态无法访问内核页面？（tips:看看上一题最后一问）

  pte不存在U位

- 单页表有何优势？（回答合理即可）

  异常环境切换速度比双页表快

- 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（回答合理即可）？

  1. 进程调度切换上下文时机,异常发生时机,异常返回时机
  2. 我会只在进程调度切换上下文时机更换页表