****



**MIT xv6实验报告**

**——lab3 : pgtbl**

**学生姓名 胡轶然**

**学 号 3019244355**

1. **实验目的**
2. 实现页表的打印、为每个进程分配一个内核页表和对copyin等函数的简化。
3. 熟悉xv6系统中页表的工作原理。
4. **前期准备**
5. 切换git分支。
6. 阅读指导书第3章的知识。
7. 阅读memlayout.h、vm.c和kalloc.c等文件中的相关代码。
8. **实验内容及实现步骤**

**任务一：print a page table**

1. **问题描述**

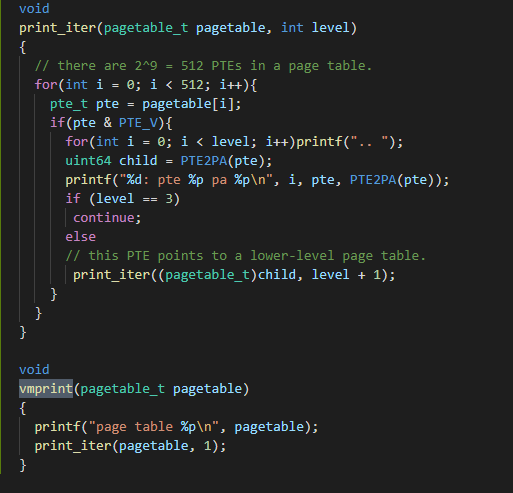
要求在系统boot时打印page table，要求用”..”标识出页表的分级情况，“**深度优先**”地打印出某页表在高级页表中的**序号**、**PTE**和**pa**（物理地址）。

1. **思路与实现步骤**

xv6系统使用三级页表，题目要求从第一级页表开始，“深度优先“地打印各级页表。可以实现一个递归函数，若某页表存储的是下一级页表的PTE，则递归地打印下一级页表；若某页表中直接存储数据，则终止此分支的打印。

具体实现方法如下：

1. 要求在boot时打印page table，首先要定义一个打印page table的函数vmprint，内容包括：
   1. 打印页表地址。
   2. 调用递归函数，打印各级页表。
2. 实现一个递归函数print\_iter，递归地打印各级页表的序号、PTE和PA。具体内容如下：
   1. 遍历页表中的512个PTE (1个页表4096字节，1个PTE 8字节，故**一个页表中有4096/8=512个PTE**)。
   2. 获取页表中某个（以下称为第i个）下一级页表的PTE。
   3. 检验PTE是否持有PTE\_V权限，即第i个下一级页表的内容是否有效（valid）。
   4. 使用PTE2PA，由PTE获取第i个下一级页表的地址。
   5. 打印第i个下一级页表的序号、PTE和pa（物理地址），注意**格式化输出64位整数时需使用”%p”**。



1. 在exec函数调用vmprint函数，打印条件是” p->pid==1”（进程pid=1），即**只在系统boot时打印第一个进程的页表**。
2. **问题与解决方法**

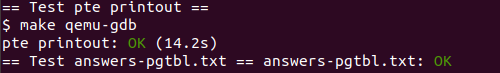
**问题一**：不理解PTE的含义。

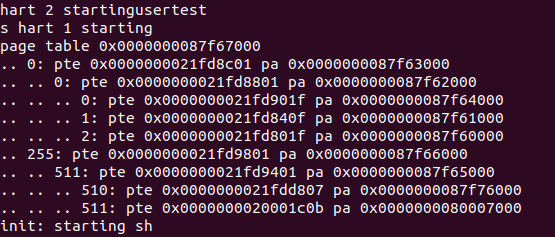
**解决方法**：阅读指导书、相关资料后了解到，**PTE由某页表的物理地址(pa)和权限标记(flag)构成，共8个字节，64位，低10位存储了读、写、有效位等权限，高54位经PTE2PA操作后转化为页表的物理地址**（**右移10位去掉标记位，然后左移12位，对应一个页表4096字节**）。

**问题二**：使用%d打印64位整数（指针），导致打印出的PTE和物理地址不完整。

**解决方法**：查阅资料后得知，**应使用%p(pointer)打印64位整数**。

1. **结果**





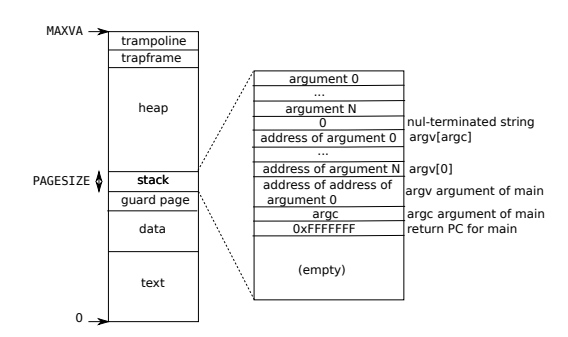
**任务二：A kernel page table per process**

1. **问题描述**

要求为每个用户维护一个内核页表，**使得系统在执行系统调用等内核态操作时，也可以使用用户进程的虚拟地址指针**。

1. **回答问题**

**问题一**：Explain the output of vmprint in terms of Fig 3-4 from the text. What does page 0 contain? What is in page 2?



答：**page0是代码段和数据段，page2是stack（栈）**。（page1是guard page，用于保护和分隔数据和栈）

**问题二**：When running in user mode, could the process read/write the memory mapped by page 1?

答：**page1是guard page**，用于保护栈和数据，该页没有PTE\_U等权限，读写等**访问会报错(page fault)**。

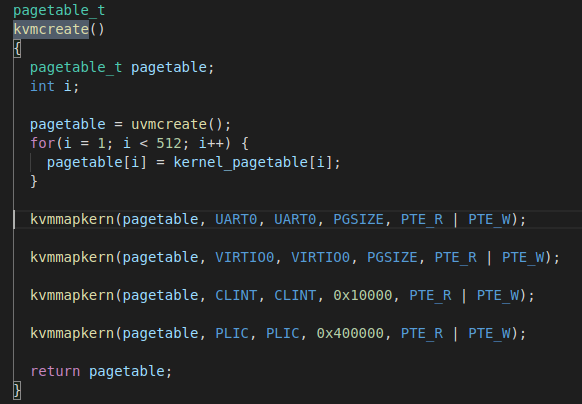
1. **思路与实现步骤**

可以仿照原代码中创建、映射和释放内核页表的相关代码，实现对某进程的内核页表的创建、映射和释放。

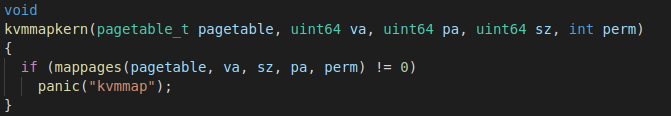
具体实现方法如下：

1. 在proc结构体中添加“内核页表“属性：”kpagetable”。
2. 模仿kvminit函数，编写创建某进程的内核页表的函数kvmcreate，具体内容如下：
   1. 调用uvmcreate函数创建一个页表。
   2. 复制共享内核页表的所有PTE。
   3. 仿照kvminit函数，实现对UART0(uart registers)、VIRTIO0(virtio mmio disk interface)、CLINT和PLIC等参数的映射。其中，kvmmapkern是仿照kvmmap实现的映射函数。

注：经测试，kvminit函数中**对KERNBASE、(uint64)etext和TRAMPOLINE等参数的映射是可以在kvmcreate函数中删去的**。



1. 仿照kvmmap函数，实现kvmmapkern函数，调用mappages函数实现内核页表中虚拟地址和物理地址的映射（**二者内容完全一致**，实现此函数只是为了在逻辑上做区分）。



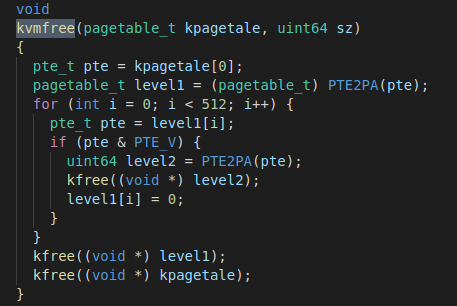
1. 接下来在proc.c的scheduler函数中，仿照kvminithart函数，实现进程切换时的内核页表切换，并在完成切换后**调用sfence\_vma函数刷新TLB**。



1. 在销毁进程时，需要释放内核页表。在freeproc函数中，模仿释放用户态页表的代码，实现对内核页表的释放。具体内容为：
   1. 调用自定义的kvmfree函数，释放内核页表的空间。
   2. 将内核页表的指针置空。



其中，用到了自定义的kvmfree函数，该函数释放了内核页表所占用的空间。



1. **问题与解决方法**

**问题一**：刚接触本任务时不理解内核页表的概念和功能，感到无从下手。

**解决方法**：阅读了指导书相关章节和相关资料，对内核页表的功能有了一定了解，经讨论发现，只需模仿原代码创建、映射和释放共享内核页表的逻辑，重新实现上述功能即可。

1. **结果**



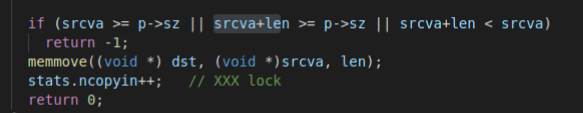
**任务3：Simplify copyin/copyinstr**

1. **任务描述**

本任务要求使用内核页表简化copyin和copyinstr操作，使得系统在执行copyin操作时，不再需要根据进程页表进行地址转换。

1. **回答问题**

**问题一**：Explain why the third test srcva + len < srcva is necessary in copyin\_new(): give values for srcva and len for which the first two test fail (i.e., they will not cause to return -1) but for which the third one is true (resulting in returning -1).



**答**：这些判断条件的目的是使拷贝的起始、终止范围不超出页表的范围，前两个条件要求拷贝的起始、终止地址不超过进程空间上限p->sz，第三个条件防止len过大时，导致srcva+len**因过大而导致整型数据溢出**。

比如，当srcva=0x05000000，p->sz=0x0A000000，len=0xFFFFFFFFFB000000时会发生只触发第三个判断条件的溢出错误。

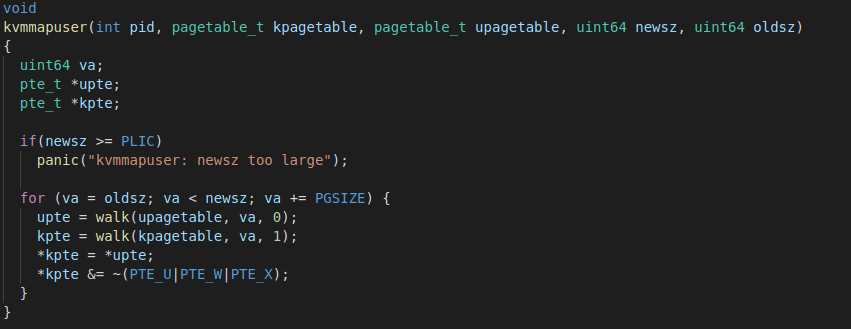
1. **思路与实现步骤**

原代码中已经提供了使用内核页表进行copyin、copyinstr操作的函数：copyin\_new和copyinstr\_new。将copyin/copyinstr函数替换为新的函数即可。

此外，每当用户页表修改时，内核页表也要做相应地修改，因此要调整fork、exec和sbrk等函数的逻辑。

具体的实现如下：

1. 首先要将copyin/copyinstr替换为copyin\_new/copyinstr\_new函数，这里使**用了一种取巧的方法，在copyin/copyinstr函数内部调用copyin\_new/copyinstr\_new函数，并return后者的返回值。**
2. 还需要将进程用户页表的改动复制到该进程的内核页表中，定义了kvmmapuser函数，将用户页表的改动复制到内核页表中。并在fork、exec和sbrk等涉及修改用户页表的函数中，调用kvmmapuser函数：
   1. 判断newsz是否在合理范围内。
   2. 遍历新增的所有虚拟页，调用walk函数，通过用户页表获取其物理地址(pte)。
   3. 将pte存入内核页表中。
   4. 取消内核页表的PTE\_U、PTE\_W和PTE\_X权限，将内核页表的新增页设置为只读的。



1. **问题和解决方法**

**问题一**：最初打算将所有copyin、copyinstr的调用逐个换成新函数。

**解决方法**：讨论后发现，只需在旧函数中调用新函数即可。

1. **结果**

见make grade测试。

1. **实验结果**

通过了make grade测试。

