**操作系统大作业2**

提交截止日：12月7日零时

**总体要求**

在github上创建os-assignment2项目，提供（1）虚存管理模拟程序源代码及结果（存成文本文件）；（2）实验报告（word/pdf），包含所有实验的基本过程描述。

1. **虚存管理模拟程序，40分**

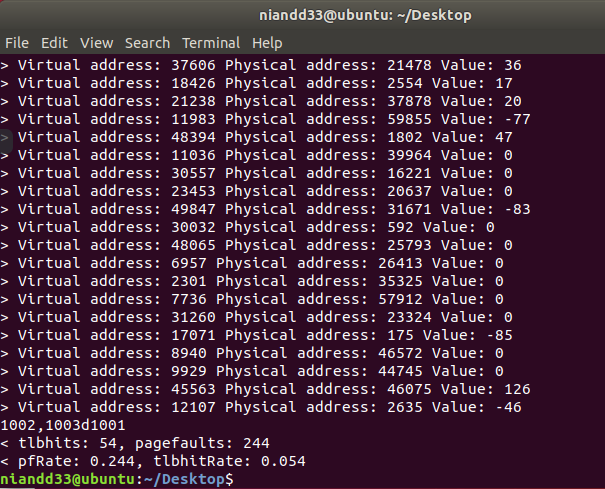
1.1 Chapter 10. Programming Projects: Designing a Virtual Memory Manager（OSC 10th ed.），30分。

1. 保持为vm.c，使用如下测试脚本test.sh，进行地址转换测试，并和correct.txt比较。

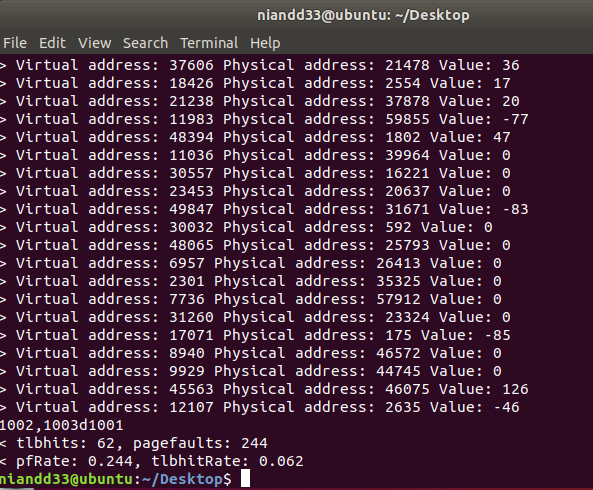
|  |
| --- |
| #!/bin/bash -e  echo "Compiling"  gcc vm.c -o vm  echo "Running vm"  ./vm BACKING\_STORE.bin addresses.txt > out.txt  echo "Comparing with correct.txt"  diff out.txt correct.txt |

注：本小题不要求实现Page Replacement，TLB分别实现FIFO和LRU两种策略。

答：TLB在FIFO策略下运行结果如图所示：



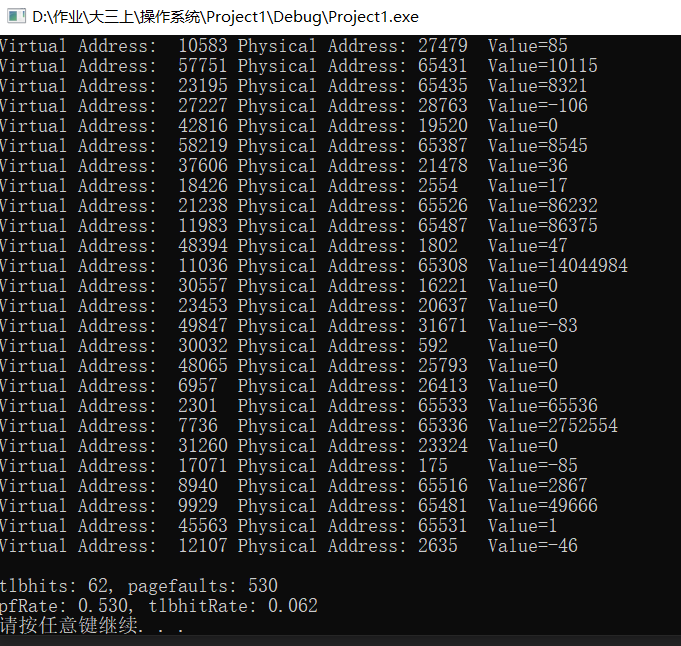
TLB在LRU策略下运行结果如图所示：



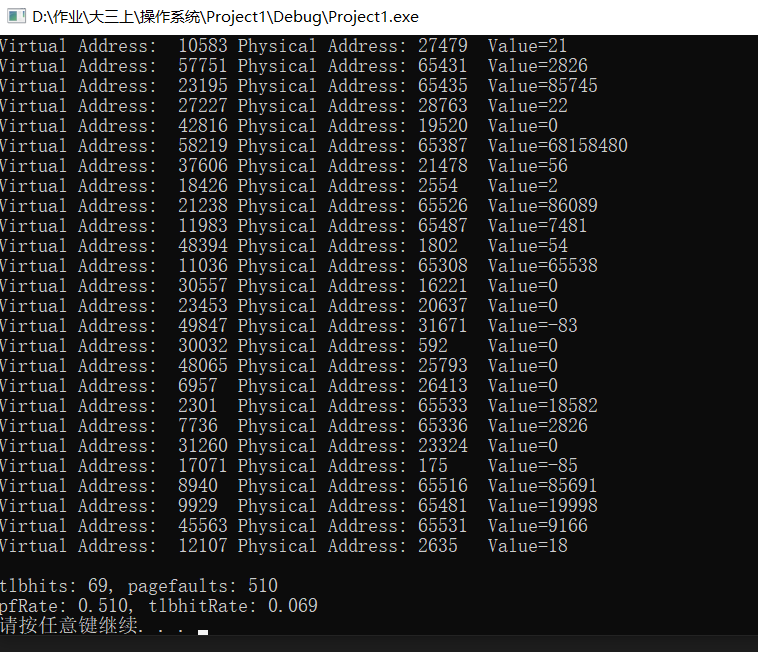
可以看出TLB在LRU置换策略下，命中率更高一些。

1. 实现基于LRU的Page Replacement；使用FIFO和LRU分别运行vm（TLB和页置换统一策略），打印比较Page-fault rate和TLB hit rate，给出运行的截屏。提示：通过getopt函数，程序运行时通过命令行指定参数。

Page Replacement 在LRU置换策略下运行结果如图：



Page Replacement 在FIFO置换策略下运行结果如图：



* 1. 编写一个简单trace生成器程序，可以用任意语言，报告里面作为附件提供。运行生成自己的addresses-locality.txt，包含10000条访问记录，体现内存访问的局部性（参考Figure 10.21, OSC 10th ed.），绘制类似图表（数据点太密的话可以采样后绘图），表现内存页的局部性访问轨迹。然后以该文件为参数运行vm，比较FIFO和LRU策略下的性能指标，最好用图对比。给出结果及分析，10分。

1. **xv6-lab-2020页表实验（**[**Lab:page tables**](https://pdos.csail.mit.edu/6.828/2020/labs/pgtbl.html)**），20分**

完成Print a page table任务。要求按图1格式打印页表内容；其中括号内表示页表项权限，R表示可读，W表示可写，X表示可执行，U表示用户可访问。物理页后的数字（pa 32618）表示第几个物理页帧。要求在报告中提供实现所需的源代码和运行截屏，代码要求有充分注释。然后，回答接下来的6个问题（分别对应代码注释行中的标签）。

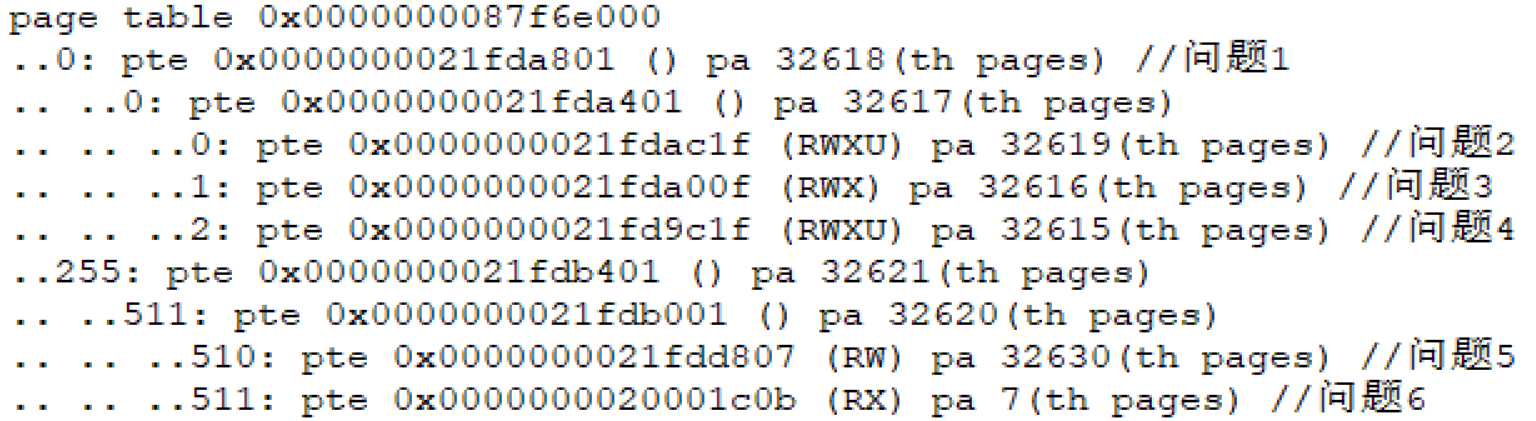


图1. init进程的页表内容

问题1：为什么第一对括号为空？32618在物理内存的什么位置，为什么不从低地址开始？结合源代码内容进行解释。

答：由运行结果可知，32618的物理地址为0x0000000087f6b000。

问题2：这是什么页？装载的什么内容？结合源代码内容进行解释。

问题3：这是什么页，有何功能？为什么没有U标志位？

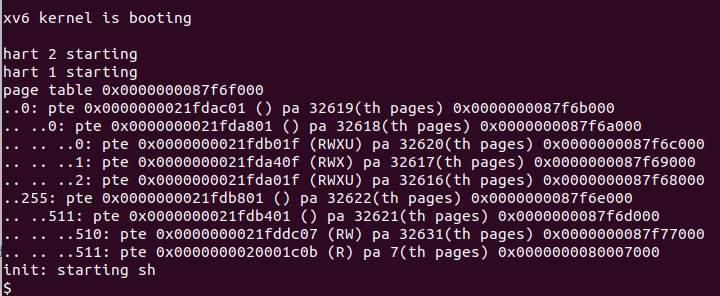
问题4：这是什么页？装载的什么内容？指出源代码初始化该页的位置。

问题5：这是什么页，为何没有X标志位？

问题6：这是什么页，为何没有W标志位？装载的内容是什么？为何这里的物理页号处于低地址区域（第7页）？结合源代码对应的操作进行解释。

1. **static** **void** traversal\_pt(pagetable\_t pagetable, **int** level){
2. **for**(**int** i=0; i<512; i++){
3. pte\_t pte = pagetable[i];
4. **char** signal[4] = {'\0'};//存储页面权限的数组
5. **if** (pte & PTE\_R) signal[0] = 'R';
6. **if** (pte & PTE\_W) signal[1] = 'W';
7. **if** (pte & PTE\_X) signal[2] = 'X';
8. **if** (pte & PTE\_U) signal[3] = 'U';
9. **if**(pte & PTE\_V){
10. uint64 child = PTE2PA(pte);
11. uint64 phyaddr = (pte>>10)&0x7FFF;
12. **if**(level==0){//如果深度为0，需要继续寻找下一层树
13. printf("..%d: pte %p (%s) pa %d(th pages) %p\n", i, pte,signal, phyaddr);
14. traversal\_pt((pagetable\_t)child, level + 1);
15. }
16. **else** **if**(level==1){//如果深度为1，仍需要继续寻找下一层树
17. printf(".. ..%d: pte %p (%s) pa %d(th pages) %p\n", i, pte,signal, phyaddr);
18. traversal\_pt((pagetable\_t)child, level + 1);
19. }
20. **else**{// 如果深度为2，直接打印
21. printf(".. .. ..%d: pte %p (%s) pa %d(th pages) %p\n", i, pte,signal, phyaddr);
22. }
24. }
26. }
28. }
30. **void** vmprint(pagetable\_t pagetable){
32. printf("page table %p\n", pagetable);
33. traversal\_pt(pagetable, 0);
35. }

运行结果如图所示：



1. **xv6-lab-2020内存分配实验 (**[**Lab: xv6 lazy page allocation**](https://pdos.csail.mit.edu/6.828/2020/labs/lazy.html)**)，40分**

3.1 完成Lazy allocation子任务，要求echo hi正常运行，报告中可以描述自己的尝试过程，以及一些中间变量。

3.2 完成Lazytests and Usertests子任务。对于Lazytests，要求屏幕输出如下图所示；对于usertests任务，要求通过所有除sbrkarg之外的测试。给出运行截屏。

在阅读报告中提供代码修改片段，说明针对哪些文件，哪些函数进行了修改，新代码加上充分注释；可以写一些体会。

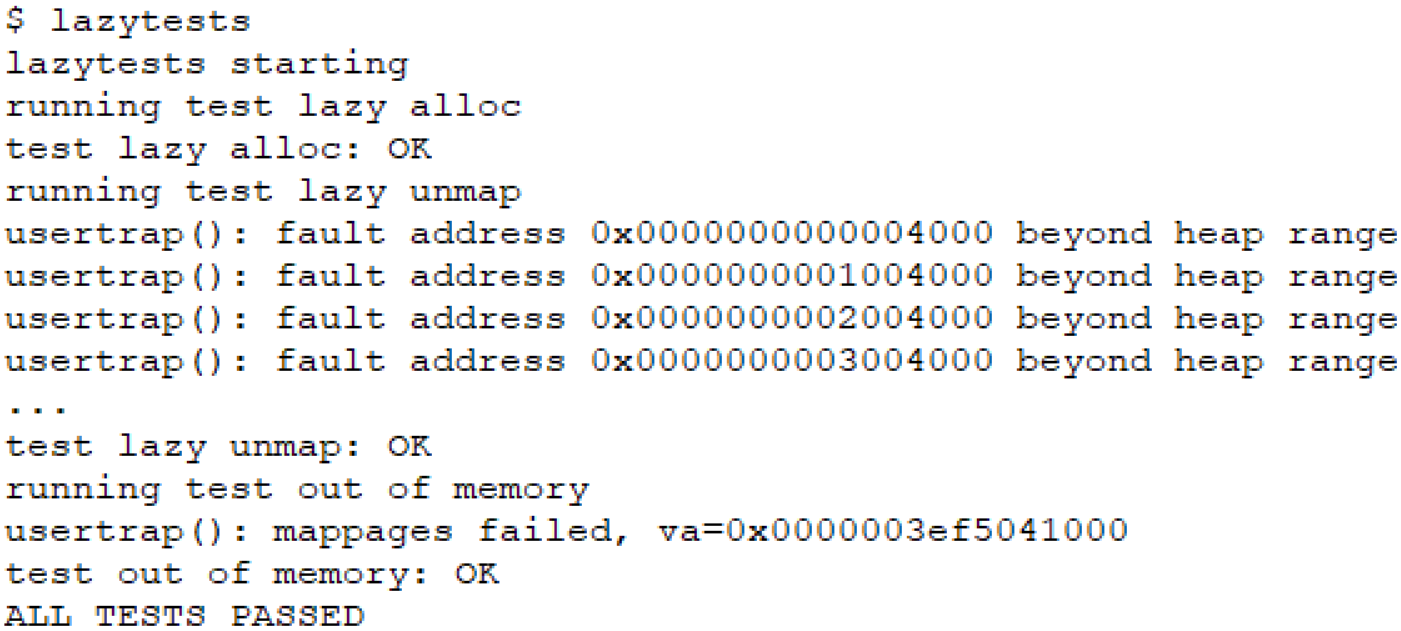
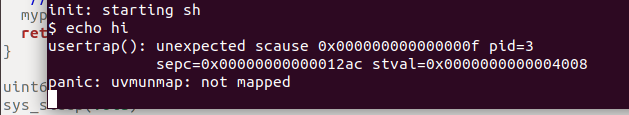


图2. Lazytests运行输出示例

3.1:题目要求删除sbrk(n)系统调用中分配页面内存的代码，sbrk(n)会根据参数n增加分配给进程的内存,然后返回新分配内存区域的起始地址。新sbrk(n)函数只根据参数n增加myproc()->sz,不实际分配页面内存。修改代码如下：

1. uint64
2. sys\_sbrk(**void**)
3. {
4. **int** addr;
5. **int** n;
7. **if**(argint(0, &n) < 0)
8. **return** -1;
9. addr = myproc()->sz;
10. //if(growproc(n) < 0)//注释掉原来的分配页面内存
11. // return -1;
12. myproc()->sz += n;//增加myproc()->sz N个字节
13. **return** addr;
14. }

运行结果如图：



发生了了错误，"usertrap(): ..."信息来自于trap.c用户中断处理程序，发生了一个他自己不知道怎么处理的中断，"stval=0x0..04008"表明发生页面错误的虚拟地址是 0x4008。

接下来修改trap.c中的代码来响应用户空间中的页面错误，它会新分配一个物理页并映射到故障地址，然后返回到用户空间来使进程继续执行。

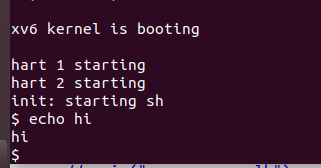
在打印"usertrap(): ..." 信息的代码前添加代码如下：

1. } **else** **if**((which\_dev = devintr()) != 0){
2. // ok
3. } **else** **if**(r\_scause()==13||r\_scause()==15){
4. ////等于13或15都说明发生了page fault
5. uint64 va = r\_stval();//发生page fault的地址
6. uint64 ka = (uint64) kalloc();
7. **if**(ka==0) p->killed =-1 ;
8. **else**{
9. memset((**void**\*)ka,0,PGSIZE);//新分配一个物理页
10. va = PGROUNDDOWN(va);
11. **if**(mappages(p->pagetable,va,PGSIZE,ka,PTE\_U|PTE\_W|PTE\_R)!=0){//映射到发生错误的地址
12. kfree((**void**\*)ka);
13. p->killed=-1;
14. }
15. }
16. } **else** {
17. printf("usertrap(): unexpected scause %p pid=%d\n", r\_scause(), p->pid);
18. printf("            sepc=%p stval=%p\n", r\_sepc(), r\_stval());
19. p->killed = 1;
20. }

同时修改vm.c函数uvmunmap()中的代码，如下所示：

1. **for**(a = va; a < va + npages\*PGSIZE; a += PGSIZE){
2. **if**((pte = walk(pagetable, a, 0)) == 0)
3. //panic("uvmunmap: walk");
4. **continue**;
5. **if**((\*pte & PTE\_V) == 0)
6. //panic("uvmunmap: not mapped");
7. **continue**;
8. **if**(PTE\_FLAGS(\*pte) == PTE\_V)
9. panic("uvmunmap: not a leaf");
10. **if**(do\_free){
11. uint64 pa = PTE2PA(\*pte);
12. kfree((**void**\*)pa);
13. }
14. \*pte = 0;
15. }

最终运行结果如图：



3.2:

首先给sbrk添加处理参数为负数的情况，就是dealloc相应的内存n，注意n不能大于p->sz。修改sysproc.c中的sys\_sbrk(void)如下：

1. uint64
2. sys\_sbrk(**void**)
3. {
4. **int** addr;
5. **int** n;
7. **if**(argint(0, &n) < 0)
8. **return** -1;
9. addr = myproc()->sz;
10. **if**(n<0){
11. **if**(myproc()->sz<0) **return** -1;//n不能大于p->sz
12. **else** uvmdealloc(myproc()->pagetable,myproc()->sz,myproc()->sz+n);//dealloc相应的内存
14. }
15. //if(growproc(n) < 0)
16. //return -1;
17. myproc()->sz += n;
18. **return** addr;
19. }

当发现缺页异常时，如果发生异常的虚拟地址比p->sz大，或者当虚拟地址比进程的用户栈还小，或者申请空间不够的时候终止进程。修改trap.c中的uesrtrap()如下:

1. } **else** **if**((which\_dev = devintr()) != 0){
2. // ok
3. } **else** **if**(r\_scause()==13||r\_scause()==15){
5. uint64 va = r\_stval();
6. **if**(va<p->sz&&va>PGROUNDDOWN(p->trapframe->sp)){//发生异常的虚拟地址比任何分配的都大
7. uint64 ka = (uint64) kalloc();              //或者虚拟地址比用户进程栈小
8. **if**(ka==0) p->killed=-1;//申请空间不够
9. **else**{
10. memset((**void**\*)ka,0,PGSIZE);
11. va = PGROUNDDOWN(va);
12. **if**(mappages(p->pagetable,va,PGSIZE,ka,PTE\_U|PTE\_W|PTE\_R)!=0){
13. kfree((**void**\*)ka);
14. p->killed=-1;
15. }
16. }
18. }
19. } **else** p->killed=-1;

然后是对vm.c 中的uvmcopy()函数进行修改，子进程复制父进程地址空间的时候，发现地址空间不存在时需要忽略.

1. **for**(i = 0; i < sz; i += PGSIZE){
2. **if**((pte = walk(old, i, 0)) == 0)//子进程复制父进程地址空间
3. //panic("uvmcopy: pte should exist");
4. **continue**;
5. **if**((\*pte & PTE\_V) == 0)//地址空间不存在
6. //panic("uvmcopy: page not present");
7. **continue**;

最终运行结果如图：

