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：

```
niandd33@ubuntu: ~/Desktop
File  Edit  View  Search  Terminal  Help
> Virtual address: 37606 Physical address: 21478 Value: 36
> Virtual address: 18426 Physical address: 2554 Value: 17
> Virtual address: 21238 Physical address: 37878 Value: 20
> Virtual address: 11983 Physical address: 59855 Value: -77
> Virtual address: 48394 Physical address: 1802 Value: 47
> Virtual address: 11036 Physical address: 39964 Value: 0
> Virtual address: 30557 Physical address: 16221 Value: 0
> Virtual address: 23453 Physical address: 20637 Value: 0
> Virtual address: 49847 Physical address: 31671 Value: -83
> Virtual address: 30032 Physical address: 592 Value: 0
> Virtual address: 48065 Physical address: 25793 Value: 0
> Virtual address: 6957 Physical address: 26413 Value: 0
> Virtual address: 2301 Physical address: 35325 Value: 0
> Virtual address: 7736 Physical address: 57912 Value: 0
> Virtual address: 31260 Physical address: 23324 Value: 0
> Virtual address: 17071 Physical address: 175 Value: -85
> Virtual address: 8940 Physical address: 46572 Value: 0
> Virtual address: 9929 Physical address: 44745 Value: 0
> Virtual address: 45563 Physical address: 46075 Value: 126
> Virtual address: 12107 Physical address: 2635 Value: -46
1002,1003d1001
< tlbhits: 54, pagefaults: 244
< pfRate: 0.244, tlbhitRate: 0.054
niandd33@ubuntu:~/Desktop$
```

TLB 在 LRU 策略下运行结果如图所示：

```
niandd33@ubuntu: ~/Desktop
File  Edit  View  Search  Terminal  Help
> Virtual address: 37606 Physical address: 21478 Value: 36
> Virtual address: 18426 Physical address: 2554 Value: 17
> Virtual address: 21238 Physical address: 37878 Value: 20
> Virtual address: 11983 Physical address: 59855 Value: -77
> Virtual address: 48394 Physical address: 1802 Value: 47
> Virtual address: 11036 Physical address: 39964 Value: 0
> Virtual address: 30557 Physical address: 16221 Value: 0
> Virtual address: 23453 Physical address: 20637 Value: 0
> Virtual address: 49847 Physical address: 31671 Value: -83
> Virtual address: 30032 Physical address: 592 Value: 0
> Virtual address: 48065 Physical address: 25793 Value: 0
> Virtual address: 6957 Physical address: 26413 Value: 0
> Virtual address: 2301 Physical address: 35325 Value: 0
> Virtual address: 7736 Physical address: 57912 Value: 0
> Virtual address: 31260 Physical address: 23324 Value: 0
> Virtual address: 17071 Physical address: 175 Value: -85
> Virtual address: 8940 Physical address: 46572 Value: 0
> Virtual address: 9929 Physical address: 44745 Value: 0
> Virtual address: 45563 Physical address: 46075 Value: 126
> Virtual address: 12107 Physical address: 2635 Value: -46
1002,1003d1001
< tlbhits: 62, pagefaults: 244
< pfRate: 0.244, tlbhitRate: 0.062
niandd33@ubuntu:~/Desktop$
```

可以看出 TLB 在 LRU 置换策略下，命中率更高一些。

(2) 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。

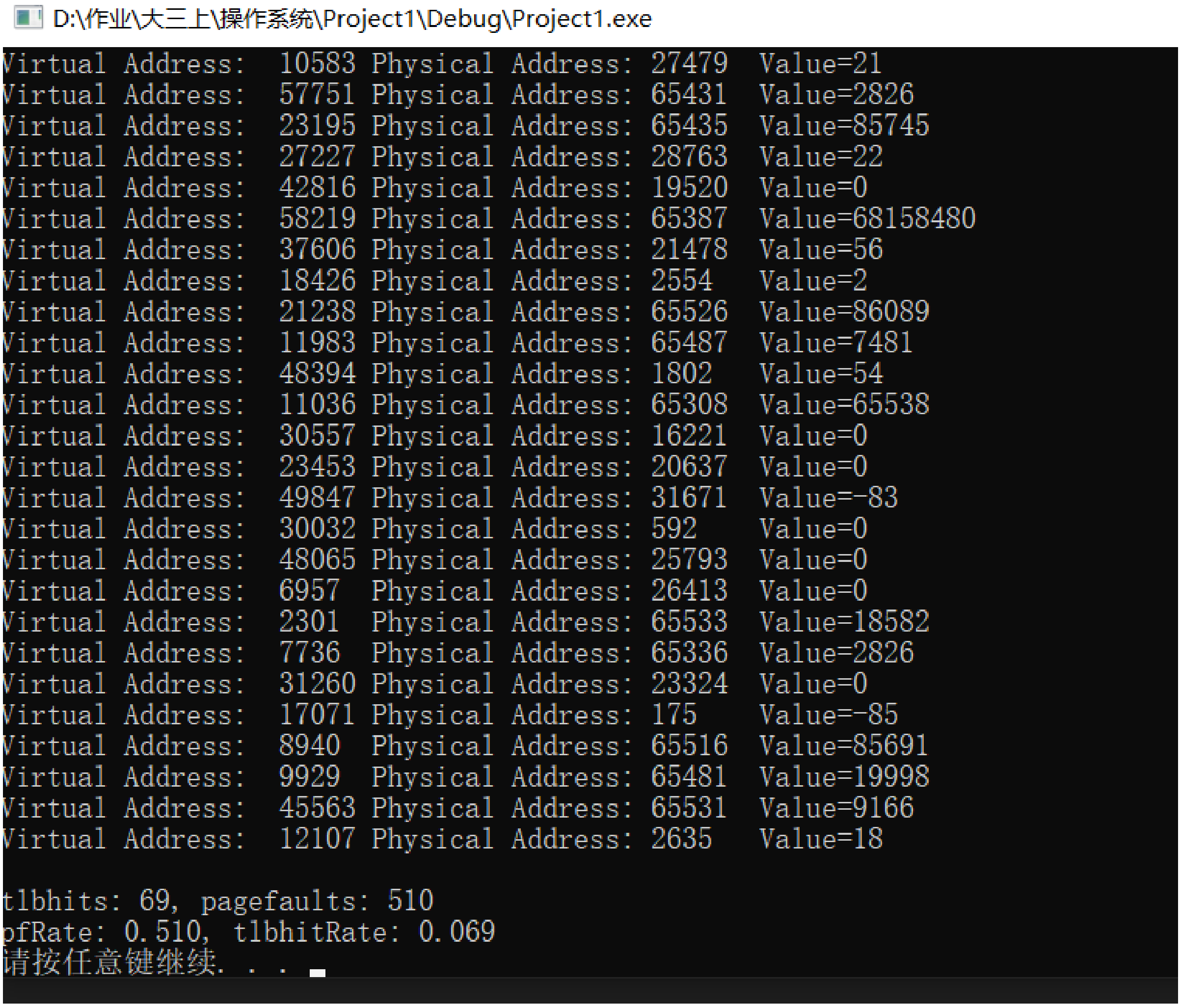
Page Replacement 在 LRU 置换策略下运行结果如图:

D:\作业\大三上\操作系统\Project1\Debug\Project1.exe

```
Virtual Address:  10583 Physical Address: 27479  Value=85
Virtual Address:  57751 Physical Address: 65431  Value=10115
Virtual Address:  23195 Physical Address: 65435  Value=8321
Virtual Address:  27227 Physical Address: 28763  Value=-106
Virtual Address:  42816 Physical Address: 19520  Value=0
Virtual Address:  58219 Physical Address: 65387  Value=8545
Virtual Address:  37606 Physical Address: 21478  Value=36
Virtual Address:  18426 Physical Address: 2554   Value=17
Virtual Address:  21238 Physical Address: 65526  Value=86232
Virtual Address:  11983 Physical Address: 65487  Value=86375
Virtual Address:  48394 Physical Address: 1802   Value=47
Virtual Address:  11036 Physical Address: 65308  Value=14044984
Virtual Address:  30557 Physical Address: 16221  Value=0
Virtual Address:  23453 Physical Address: 20637  Value=0
Virtual Address:  49847 Physical Address: 31671  Value=-83
Virtual Address:  30032 Physical Address: 592    Value=0
Virtual Address:  48065 Physical Address: 25793  Value=0
Virtual Address:  6957  Physical Address: 26413  Value=0
Virtual Address:  2301  Physical Address: 65533  Value=65536
Virtual Address:  7736  Physical Address: 65336  Value=2752554
Virtual Address:  31260 Physical Address: 23324  Value=0
Virtual Address:  17071 Physical Address: 175    Value=-85
Virtual Address:  8940  Physical Address: 65516  Value=2867
Virtual Address:  9929  Physical Address: 65481  Value=49666
Virtual Address:  45563 Physical Address: 65531  Value=1
Virtual Address:  12107 Physical Address: 2635   Value=-46

tlbhits: 62, pagefaults: 530
pfRate: 0.530, tlbhitRate: 0.062
请按任意键继续. . .
```

Page Replacement 在 FIFO 置换策略下运行结果如图:



D:\作业\大三上\操作系统\Project1\Debug\Project1.exe

```
Virtual Address:  10583 Physical Address: 27479  Value=21
Virtual Address:  57751 Physical Address: 65431  Value=2826
Virtual Address:  23195 Physical Address: 65435  Value=85745
Virtual Address:  27227 Physical Address: 28763  Value=22
Virtual Address:  42816 Physical Address: 19520  Value=0
Virtual Address:  58219 Physical Address: 65387  Value=68158480
Virtual Address:  37606 Physical Address: 21478  Value=56
Virtual Address:  18426 Physical Address: 2554   Value=2
Virtual Address:  21238 Physical Address: 65526  Value=86089
Virtual Address:  11983 Physical Address: 65487  Value=7481
Virtual Address:  48394 Physical Address: 1802   Value=54
Virtual Address:  11036 Physical Address: 65308  Value=65538
Virtual Address:  30557 Physical Address: 16221  Value=0
Virtual Address:  23453 Physical Address: 20637  Value=0
Virtual Address:  49847 Physical Address: 31671  Value=-83
Virtual Address:  30032 Physical Address: 592    Value=0
Virtual Address:  48065 Physical Address: 25793  Value=0
Virtual Address:  6957  Physical Address: 26413  Value=0
Virtual Address:  2301  Physical Address: 65533  Value=18582
Virtual Address:  7736  Physical Address: 65336  Value=2826
Virtual Address:  31260 Physical Address: 23324  Value=0
Virtual Address:  17071 Physical Address: 175    Value=-85
Virtual Address:  8940  Physical Address: 65516  Value=85691
Virtual Address:  9929  Physical Address: 65481  Value=19998
Virtual Address:  45563 Physical Address: 65531  Value=9166
Virtual Address:  12107 Physical Address: 2635   Value=18

tlbhits: 69, pagefaults: 510
pfRate: 0.510, tlbhitRate: 0.069
请按任意键继续. . .
```

1.2 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。

2. xv6-lab-2020 页表实验（Lab:page tables），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）。

```
page table 0x0000000087f6e000
..0: pte 0x0000000021fda801 () pa 32618(th pages) //问题1
.. ..0: pte 0x0000000021fda401 () pa 32617(th pages)
.. .. ..0: pte 0x0000000021fdac1f (RWXU) pa 32619(th pages) //问题2
.. .. ..1: pte 0x0000000021fda00f (RWX) pa 32616(th pages) //问题3
.. .. ..2: pte 0x0000000021fd9c1f (RWXU) pa 32615(th pages) //问题4
..255: pte 0x0000000021fdb401 () pa 32621(th pages)
.. ..511: pte 0x0000000021fdb001 () pa 32620(th pages)
.. .. ..510: pte 0x0000000021fdd807 (RW) pa 32630(th pages) //问题5
.. .. ..511: pte 0x0000000020001c0b (RX) pa 7(th pages) //问题6
```

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。

```c
static void traversal_pt(pagetable_t pagetable, int level){
    for(int i=0; i<512; i++){
        pte_t pte = pagetable[i];
        char signal[4] = {'\0'};//存储页面权限的数组
        if (pte & PTE_R) signal[0] = 'R';
        if (pte & PTE_W) signal[1] = 'W';
        if (pte & PTE_X) signal[2] = 'X';
        if (pte & PTE_U) signal[3] = 'U';
        if(pte & PTE_V){
            uint64 child = PTE2PA(pte);
            uint64 phyaddr = (pte>>10)&0x7FFF;
            if(level==0){//如果深度为 0，需要继续寻找下一层树
                printf("..%d: pte %p (%s) pa %d(th pages) %p\n", i, pte,signal, phyaddr);
                traversal_pt((pagetable_t)child, level + 1);
            }
```

```c
            else if(level==1){//如果深度为 1，仍需要继续寻找下一层树
               printf(".. ..%d: pte %p (%s) pa %d(th pages) %p\n", i, pte,signal, phyaddr);
               traversal_pt((pagetable_t)child, level + 1);
            }
            else{// 如果深度为 2，直接打印
               printf(".. .. ..%d: pte %p (%s) pa %d(th pages) %p\n", i, pte,signal, phyaddr);
            }

        }

    }

}

void vmprint(pagetable_t pagetable){

    printf("page table %p\n", pagetable);
    traversal_pt(pagetable, 0);

}
```

运行结果如图所示:

```
xv6 kernel is booting

hart 2 starting
hart 1 starting
page table 0x0000000087f6f000
..0: pte 0x0000000021fdac01 () pa 32619(th pages) 0x0000000087f6b000
.. ..0: pte 0x0000000021fda801 () pa 32618(th pages) 0x0000000087f6a000
.. .. ..0: pte 0x0000000021fdb01f (RWXU) pa 32620(th pages) 0x0000000087f6c000
.. .. ..1: pte 0x0000000021fda40f (RWX) pa 32617(th pages) 0x0000000087f69000
.. .. ..2: pte 0x0000000021fda01f (RWXU) pa 32616(th pages) 0x0000000087f68000
..255: pte 0x0000000021fdb801 () pa 32622(th pages) 0x0000000087f6e000
.. ..511: pte 0x0000000021fdb401 () pa 32621(th pages) 0x0000000087f6d000
.. .. ..510: pte 0x0000000021fddc07 (RW) pa 32631(th pages) 0x0000000087f77000
.. .. ..511: pte 0x0000000020001c0b (R) pa 7(th pages) 0x0000000080007000
init: starting sh
$
```

3. xv6-lab-2020 内存分配实验（[Lab: xv6 lazy page allocation]），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。

```
$ lazytests
lazytests starting
running test lazy alloc
test lazy alloc: OK
running test lazy unmap
usertrap(): fault address 0x0000000000004000 beyond heap range
usertrap(): fault address 0x0000000001004000 beyond heap range
usertrap(): fault address 0x0000000002004000 beyond heap range
usertrap(): fault address 0x0000000003004000 beyond heap range
...
test lazy unmap: OK
running test out of memory
usertrap(): mappages failed, va=0x0000003ef5041000
test out of memory: OK
ALL TESTS PASSED
```

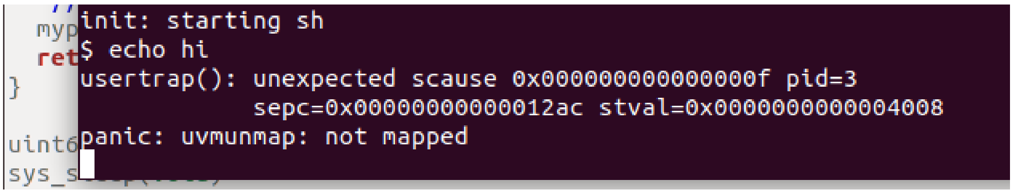
图 2. Lazytests 运行输出示例

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:

```
uint64
sys_sbrk(void)
{
  int addr;
  int n;

  if(argint(0, &n) < 0)
    return -1;
  addr = myproc()->sz;
  //if(growproc(n) < 0)//注释掉原来的分配页面内存
   // return -1;
  myproc()->sz += n;//增加 myproc()->sz N 个字节
  return addr;
}
```

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：

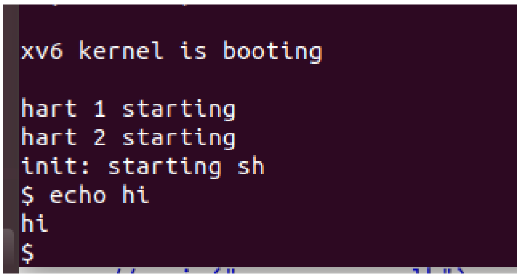
```
} else if((which_dev = devintr()) != 0){
    // ok
  } else if(r_scause()==13||r_scause()==15){
    ////等于 13 或 15 都说明发生了 page fault
    uint64 va = r_stval();//发生 page fault 的地址
    uint64 ka = (uint64) kalloc();
    if(ka==0) p->killed =-1 ;
    else{
        memset((void*)ka,0,PGSIZE);//新分配一个物理页
        va = PGROUNDDOWN(va);
        if(mappages(p->pagetable,va,PGSIZE,ka,PTE_U|PTE_W|PTE_R)!=0){//映射到发生错误的地址
           kfree((void*)ka);
           p->killed=-1;
        }
    }
  } else {
    printf("usertrap(): unexpected scause %p pid=%d\n", r_scause(), p->pid);
    printf("            sepc=%p stval=%p\n", r_sepc(), r_stval());
    p->killed = 1;
  }
```

同时修改 vm.c 函数 uvmunmap() 中的代码，如下所示：

```
for(a = va; a < va + npages*PGSIZE; a += PGSIZE){
    if((pte = walk(pagetable, a, 0)) == 0)
      //panic("uvmunmap: walk");
      continue;
    if((*pte & PTE_V) == 0)
      //panic("uvmunmap: not mapped");
      continue;
    if(PTE_FLAGS(*pte) == PTE_V)
      panic("uvmunmap: not a leaf");
```

```
    if(do_free){
      uint64 pa = PTE2PA(*pte);
      kfree((void*)pa);
    }
    *pte = 0;
  }
```

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：

```
uint64
sys_sbrk(void)
{
  int addr;
  int n;

  if(argint(0, &n) < 0)
    return -1;
  addr = myproc()->sz;
  if(n<0){
     if(myproc()->sz<0) return -1;//n 不能大于 p->sz
     else uvmdealloc(myproc()->pagetable,myproc()->sz,myproc()->sz+n);//dealloc 相应的内存

  }
  //if(growproc(n) < 0)
    //return -1;
  myproc()->sz += n;
  return addr;
}
```

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：

```
} else if((which_dev = devintr()) != 0){
    // ok
```

```c
    } else if(r_scause()==13||r_scause()==15){

      uint64 va = r_stval();
      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
         uint64 ka = (uint64) kalloc();              //或者虚拟地址比用户进程栈小
         if(ka==0) p->killed=-1;//申请空间不够
         else{
             memset((void*)ka,0,PGSIZE);
             va = PGROUNDDOWN(va);
             if(mappages(p->pagetable,va,PGSIZE,ka,PTE_U|PTE_W|PTE_R)!=0){
                kfree((void*)ka);
                p->killed=-1;
             }
         }

      }
    } 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.

```c
for(i = 0; i < sz; i += PGSIZE){
    if((pte = walk(old, i, 0)) == 0)//子进程复制父进程地址空间
      //panic("uvmcopy: pte should exist");
      continue;
    if((*pte & PTE_V) == 0)//地址空间不存在
      //panic("uvmcopy: page not present");
      continue;
```

最终运行结果如图：

```
xv6 kernel is booting

hart 1 starting
hart 2 starting
init: starting sh
$ lazytests
lazytests starting
running test lazy alloc
test lazy alloc: OK
running test lazy unmap
memory not unmapped
test lazy unmap: FAILED
running test out of memory
panic: walk
```

```
xv6 kernel is booting

hart 1 starting
hart 2 starting
init: starting sh
$ usertests
usertests starting
test execout: OK
test copyin: OK
test copyout: OK
test copyinstr1: OK
test copyinstr2: OK
test copyinstr3: OK
test rwsbrk: OK
test truncate1: OK
test truncate2: OK
test truncate3: OK
test reparent2: OK
test pgbug: OK
test sbrkbugs: OK
test badarg: OK
```