1.1

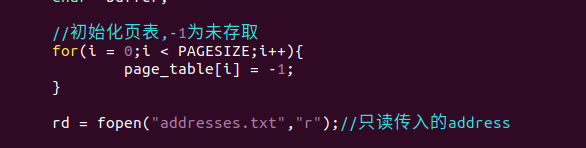
（1）

主要是写出一个程序能够读取虚拟地址并能够转换成虚拟地址输出

输入为addresses.txt的虚拟地址

输出为out.txt的物理地址，并且与correct对照无输出

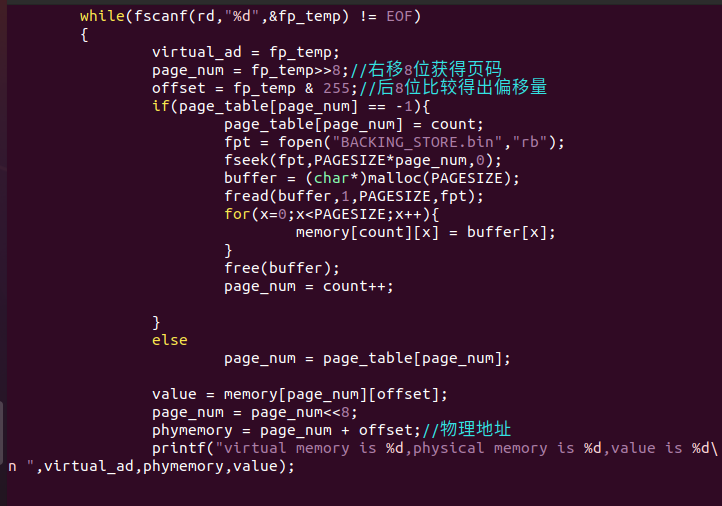
初始化与读入：



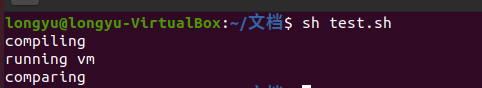
根据虚拟地址的定义，前8位遍历页表查表获得页码，后8位偏移量，

如果发现缺页，则运行Bin文件读取一个二进码补页

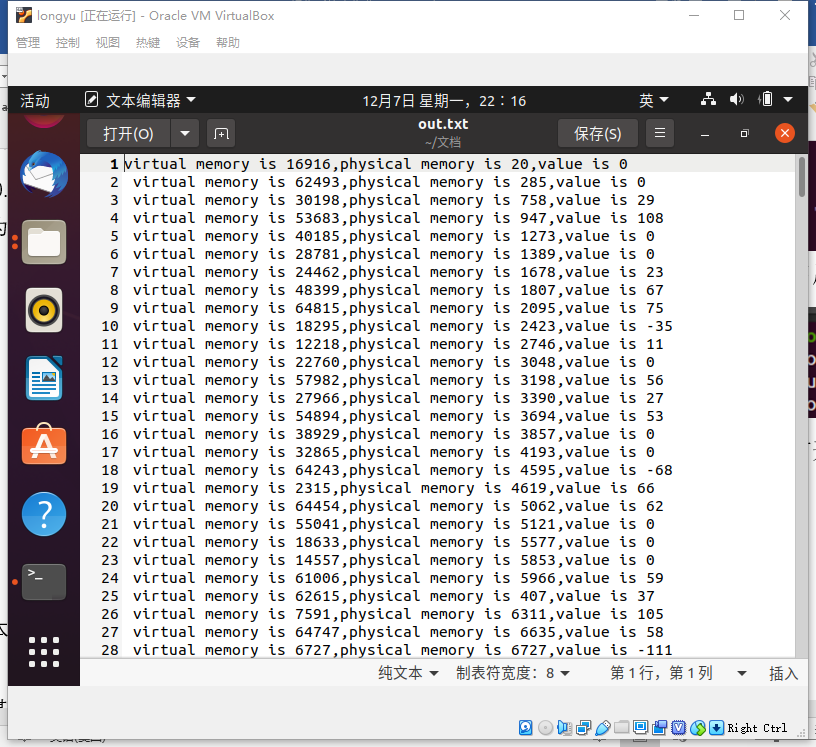
然后根据得到的页码和偏移量求出物理内存地址以及寄存在物理内存的值



启用test.sh测试得到



打开out.txt查看与correct基本无误

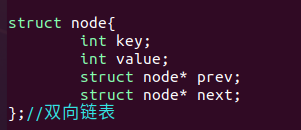


所以地址转换完成

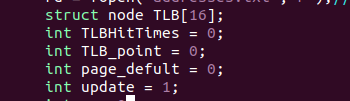
1.2要实现FIFO和LRU我们得先完成TLB

引入TLB能够极高的提高搜寻速度，可以节省出一部分遍历页表的时间

先定义一个节点结构作为TLB

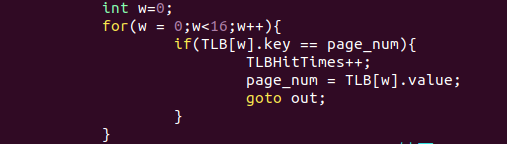


对其进行初始化



这样我们拥有一个大小为16的TLB了

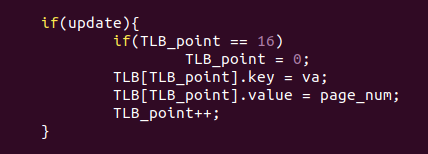
在查询时，我们优先遍历TLB



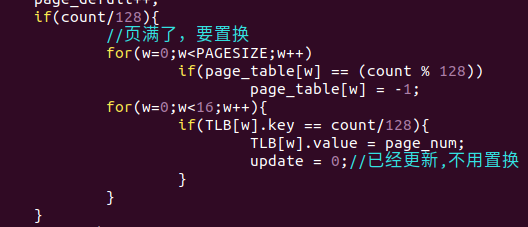
如果有结果，则直接跳转至输出

无结果则继续遍历页表，并对TLB做更新

FIFO更新的机制是先进先出



在这一题中，我们的物理内存缩小为128，所以会存在满的情况，我们需要对其进行置换，腾出新的空间给新的进程



换页时也顺便更新TLB。

最后FIFO的命中率以及缺页率

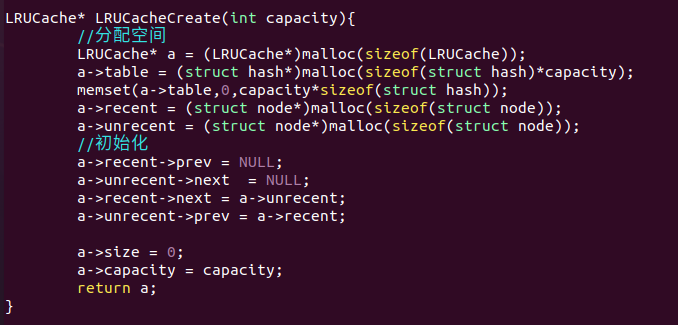
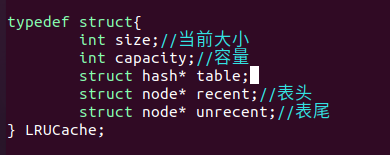


LRU：LRU策略使用的是少用先出的策略

我们构建一个链表，每有新的被命中，或者更新新的数据组，我们就将节点放在表头，如此一来，很少被调用的或者近期没被调用的节点就会沉在表尾

每次置换时只需要将表尾的节点换出即可

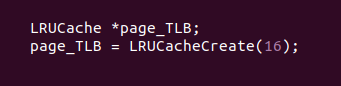
链表的节点结构



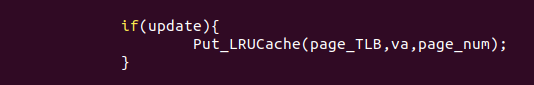
我们编出get函数输入key返回value（输入逻辑地址输出页码）

Put函数输入新的节点添加或置换进链表内

初始化：



更新TLB:



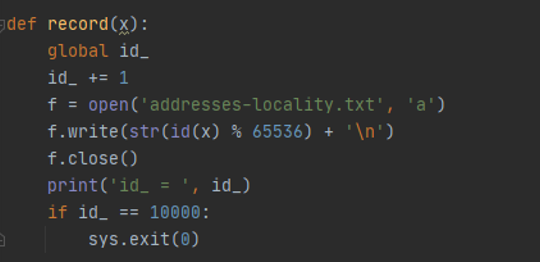
最后命中率如图：



可以看出LRU策略和FIFO策略的命中率与缺页没有太大区别

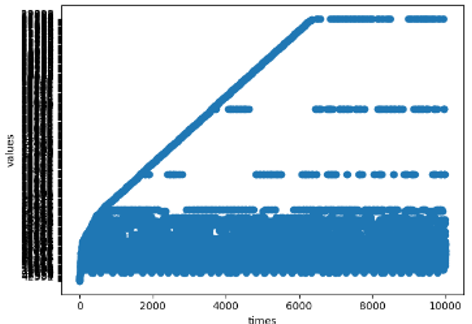
1.2trace

利用python我们可以很轻松的写出一个绘图函数



X为内存地址，坐标轴设为65536，y为访问记录，坐标轴设为10000

由address\_locality.txt生成的绘图如下：



运行FIFO运行得到



LRU运行得到



可以看出，比起随机生成的数据，有规律的数据TLB的命中率更高，且LRU的效果远远好于FIFO

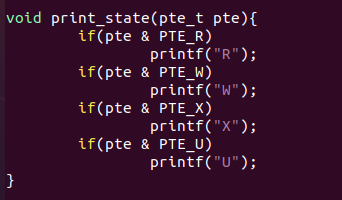
2.page table

程序修改：

2.1我们需要在vm.c文件下写一个print函数，使得我们可以打印页表

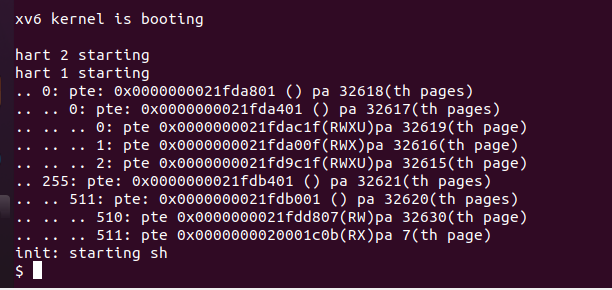


Print\_state函数如图：



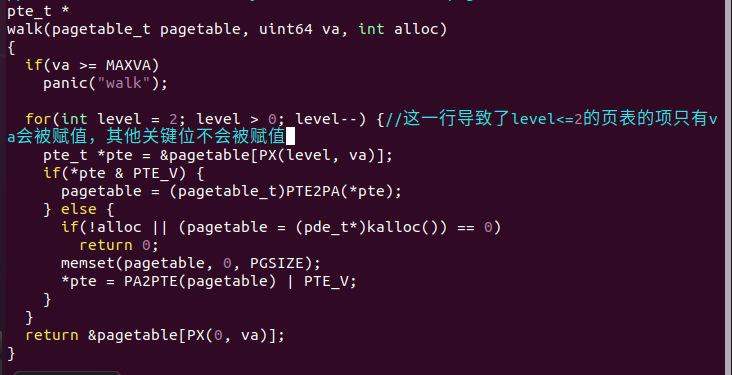
写完函数后，去exec.c下加上extern vmprint(参数);即可

打印结果如下：



2.2

（1）空括号的问题：



物理内存32618转换为87F6 A000,对应的是RAM区域

为什么从高地址开始是因为低地址要留给各种寄存器来使用

（2）

装载的是程序的数据段

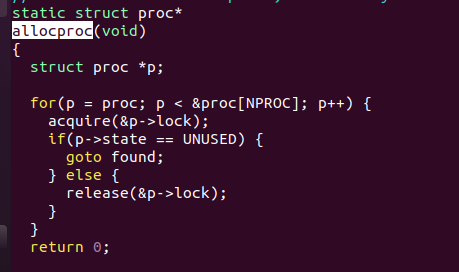


Uvmalloc函数调用当前的虚拟地址并存到PTE中

（3）这是guard page 页，主要防止用户申请的stack空间溢出，没有U是因为这是受保护的页面，不能被用户修改

（4）这是用户申请栈的位置

（5）由打印的表可知，该页物理帧数最大，分配时间最早，是我们第一个分配的内存，相关源码：

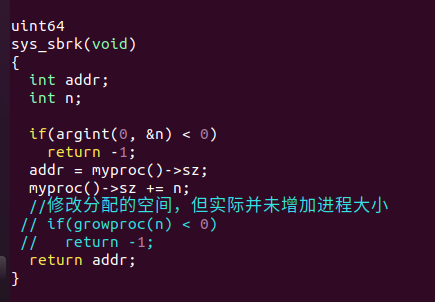


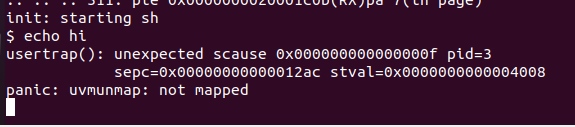
该函数内没有指令操作，故没有X标记

（6）最后这页是trampoline 是用来专门切换模式的，只有内核才能使用，所以没有UX标记

* 1. **lazy page allocation**

懒加载我们需要先对sys\_sbrk()函数进行修改，增加内存空间的大小，但不进行实际的分配

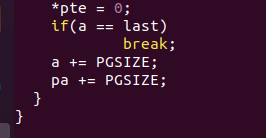
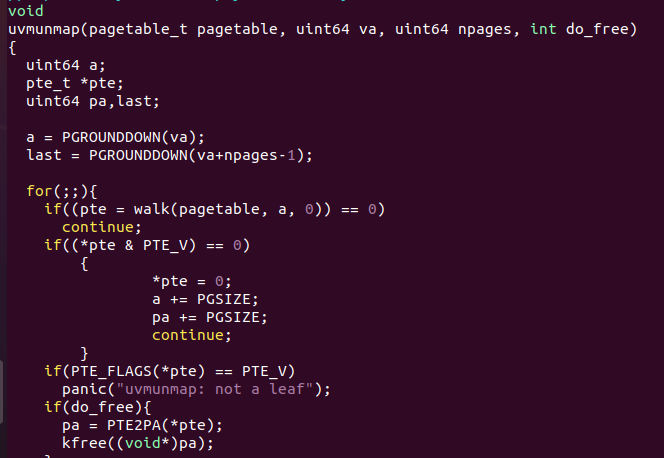


修改后，我们运行make qemu：

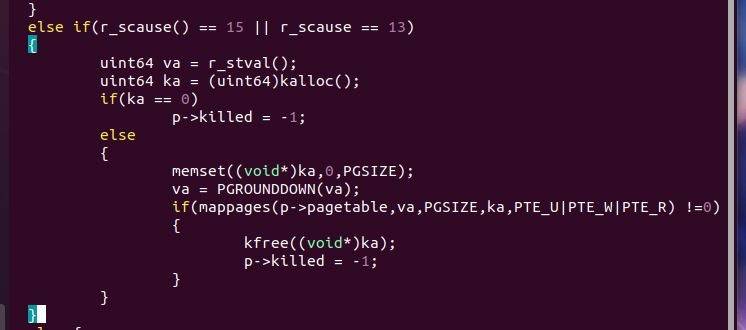
可以看出提示为 not mapped

于是我们去vm.c下的uvmunmap（）函数进行修改，。让他结束流程而不是报错

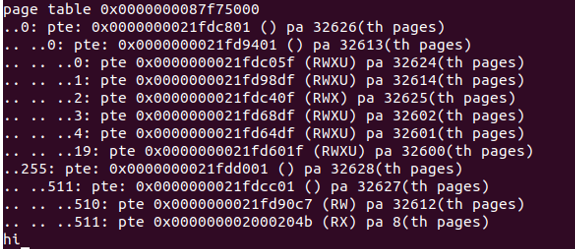
主要是删除两个panic，并将其改为continue



然后对trap进行修改即可

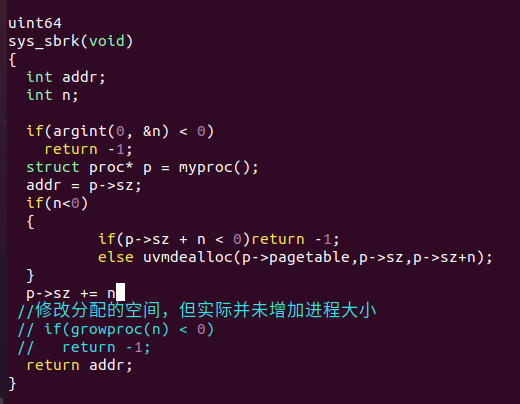


打印结果：



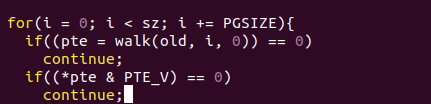
* 1. lazytest and usertest

首先我们得先将sbrk的函数完善，给他添加处理参数为负数的情况



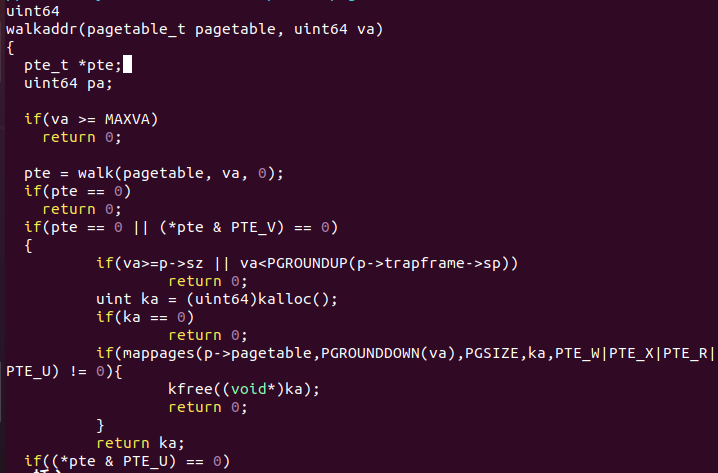
然后去修改复制的代码，为vm.c的uvmcopy（）

依旧是删掉panic换位continue



最后当系统调用时，得到的地址需要找到相应的物理地址，添加到pagetable中

我们需要对walkaddr进行修改



这样我们就完成了

打印结果如下：

