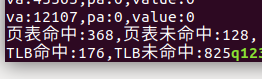


操作系统大作业

姓名：晋军

学号：18364037

**第一大题运行截图**：

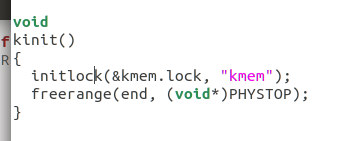


**第一题：**

因为该页面存储的是页表，不需要（RWXU）保护。

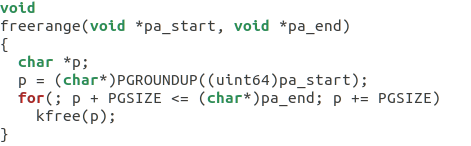
323618在内核末尾和PHYSTOP之间。系统page由freelist保存，并通过kalloc()获取。freelist有高地址到低地址发送给kalloc()。

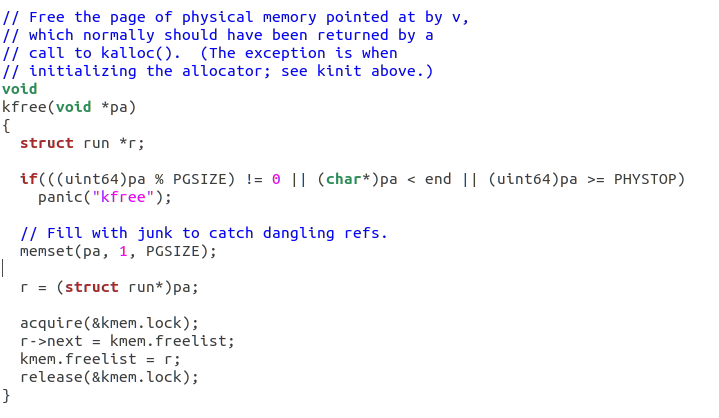
代码：kinit的初始化是由内核末尾到PHYSTOP



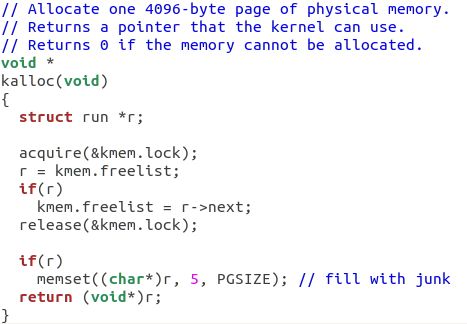


由内核末尾一个个添加给freelist，所以最后freelist首先指向了PHYSTOP

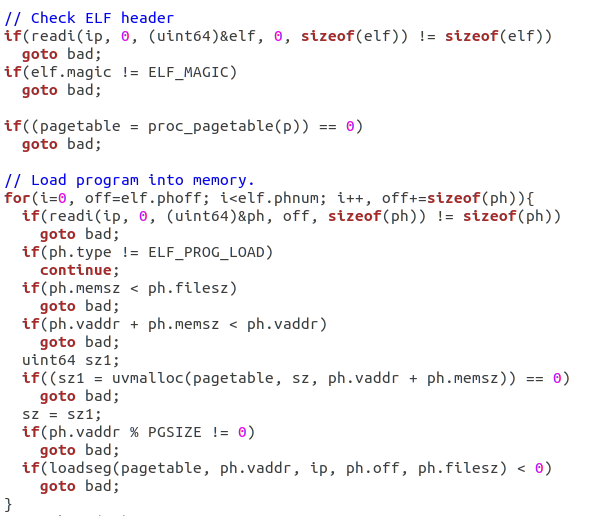




当调用kalloc时，首先得到的时freelist的表头地址，然后freelist更新至下一个地址。一开始freelist指向PHYSTOP

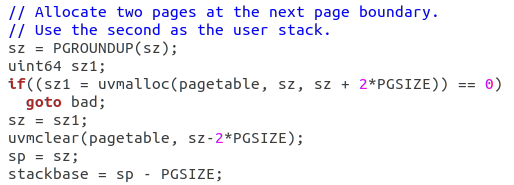


**第二题：**

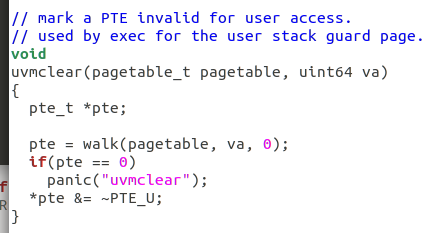
这是用户空间的text页，用于保存二进制代码。可以通过EDL Header找到，

**第三题：**

这页用于防止用户栈溢出。当用户申请访问直至溢出的时候，将会申请到该页，由于PTE\_U=0,所以将会出发错误，防止用户栈溢出。



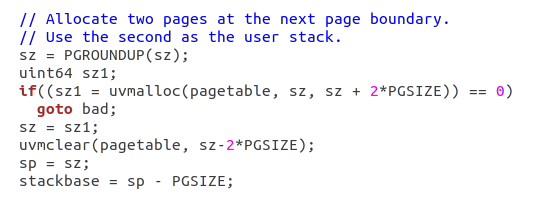
该函数将这页的PTE\_U位设置为了0



**第四题：**

这是用户栈，用于存储argument和address of argument

这里一开始为上述存储内容申请了两页page



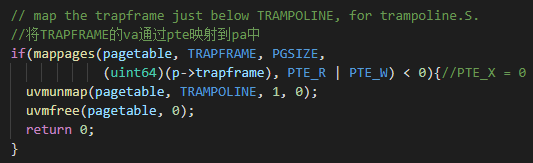
**第五题：**

这是TRAPFRAME页

代码部分：在（exec.c/exec）中，通过调用（proc.c/proc\_pagetable）映射TRAMPOLINE和TRAPFRAME两页



在（proc.c/proc\_pagetable）中可以看到具体的实现过程：





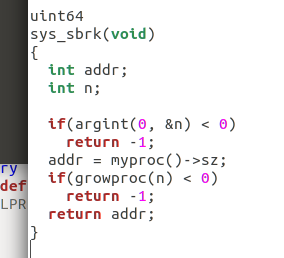
可以看到，代码只对PTE\_R和PTE\_W取1，PTE\_X并未取1

因为TRAPFRAME中存储的不是可执行代码，而是某些寄存器中的内容，所以PTE\_X未取一

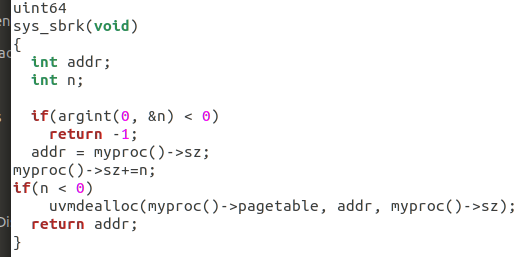
**第三大题：**

首先更改sysproc.c函数，使得申请内存的进程无法实际上获得内存空间。

更改前：



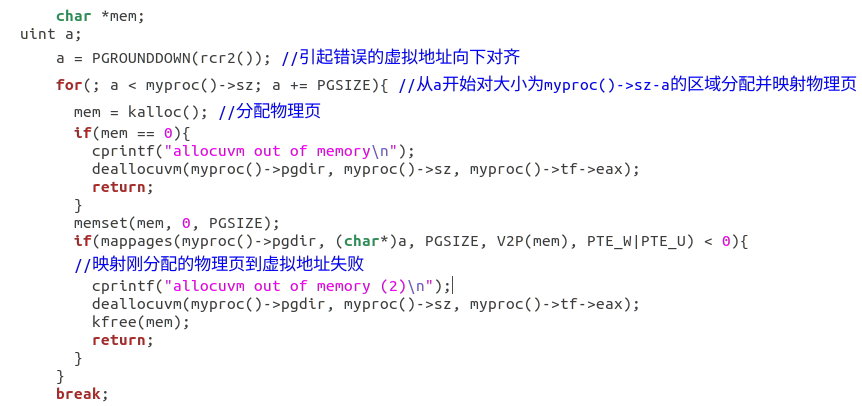
更改后：



然后运行echo hi的结果如下：



这是因为当系统去查看进程对应的内存，但是实际上我们并没有分配内存给他。那么我们就需要在此之前，赶紧把需要的内存分配了。

通过更改trap.c里面的trap()的default的部分，可以实现我们的目标  


然后结果如下

