**Lazy page allocation**

一、实验目的

操作系统使用页表的硬件来完成一些很巧妙的技巧之一就是对堆内存来懒分配(lazy applocation)。Xv6程序使用系统调用sbrk()来申请堆内存。在现有的xv6源码当中,sbrk()分配了物理内存并且将新分配的物理内存同进程的虚拟内存映射起来。有一些程序申请了内存，但是并没有使用这些内存，比如说一个很大的稀疏矩阵。复杂的内核会延迟每一个内存页的分配知道程序真正需要这个页--会触发page fault（因为没有分配页，所以会触发page fault）。本次实验的目的就是在xv6中实现lazy page allocation。

二、实验内容

[Homework: xv6 lazy page allocation (mit.edu)](https://pdos.csail.mit.edu/6.828/2018/homework/xv6-zero-fill.html)

三、实验步骤与实验结论

第一部分：消除 sbrk（）中的分配

在sbrk()移除page allocation的代码。sbrk()会掉用sys\_sbrk()来实现内存的分配，所以要去修改sys\_sbrk()的代码（sys\_sbrk在sysproc.c中）。

新的sbrk(n)这个系统调用将进程的内存范围增长n字节，然后返回旧的内存大小（新分配的内存块的地址）即只增加进程内存大小，但不分配内存。

echo的命令类型是execcmd,因此我们修改execcmd函数代码和morecore函数代码，而malloc函数中仅引用了一次morecore,且后续代码未访问分配的内存区域



第二部分：惰性分配

第二部分要求我们修改trap.c中的代码通过新分配一个物理页并映射到故障地址的方式来解决这个页故障,之后返回到用户空间来使进程继续执行,应当在打印错误消息之前添加代码,代码无需涵盖所有极端情况和错误情况,能够满足让sh运行echo,ls之类的简单命令就可以了。

vm.c中的mappages 函数static声明去除，在trap.c中，当发现是page fault错误的时候，可以按照当前进程的proc->sz来实际的分配内存，注意这个时候的sz的大小不是实际的大小，而是希望的大小值

所以首先应该获取发生page fault时刻的虚地址，那个地址之后的部分就应该是本来应该分配但是实际没有分配的，而实际需要分配多少，应该根据proc->sz的大小来定。

因为在发生page fault的时候的地址，是我们在malloc之后返回给进程的地址，而这个地址又是原来的proc->sz，所以该虚地址的大小就应该是实际的内存的大小

具体实现：

查看打印上面错误信息的那行cprintf

可以参考vm.c中allocuvm函数的代码,该函数由sbrk()的子函数growproc()调用

使用PGROUNDDOWN(va)来保证发生故障的虚拟地址边界对齐

使用break或return 避免打印错误信息和设置myproc()->killed位

修改代码过程中需要调用mappages(),为此,需要删除vm.c中mappages()的static修饰符,并且需要在trap.c中对mappages()进行任何调用之前,声明此函数

检查tf->trapno是否等于T\_PPGFLT来判断是否发生页故障。

修改trap.c中的代码，以通过在故障地址处映射新分配的物理内存页面，然后返回用户空间以使进程继续执行，来响应用户空间中的页面错误



四、反思总结

在shell中运行echo时，需要调用malloc函数。运行malloc的时候，虽然返回是成功了的，但是当程序试图操作cmd指向的内存区域的时候，发现该内存区域不是当前进程所有的，因为在sys\_sbrk中根本没分配。