@[TOC](Mit6.S081-实验5-xv6 lazy page allocation)

# 一、Eliminate allocation from sbrk()

## 1，实验准备

O/S可以跟page table hardware表演的众多小花招之一是用户空间堆内存的懒分配。

xv6应用使用system call sbrk()向kernel索取堆内存。

在kernel中，我们已经给出了，sbrk()分配物理内存并且匹配它到进程的虚拟地址空间。

对一个大的分配请求，这对kernel来说将花费大量时间来分配和匹配内存，例如，1GB包含262144个4096字节的页。

这是大量分配，即使每个是少量的。

另外，一些程序分配比实际使用更多的内存（例如稀疏数组 sparse arrays），或者在使用前分配内存。

为了让sbrk()在这些场景下完成的更快，精密的kernel懒惰地分配内存 。

就是说，sbrk()不分配物理内存，而是仅仅记录哪个用户地址被分配了，并且标记那些未分配的地址是无效的。

当进程首次尝试使用任意给定的懒分配内存页，cpu生成一个page fault，kernel通过分配物理内存、清零、映射来处理这个page fault。

在此实验中，你将添加这个懒分配特征到xv6。

1) 写代码前，阅读xv6 book第四章（特别是4.6），以及可能更改的相关文件：

kernel/trap.c、kernel/vm.c、kernel/sysproc.c

2) 为了开始此实验，切换到lazy分支

git fetch、git checkout lazy、make clean

## 2，实验要求

你的第一个任务是在sbrk(n) system call实现（函数sys\_sbrk()在sysproc.c中）中删除page分配。

sbrk(n) system call通过n bytes增长进程的内存尺寸，然后返回最新分配区域的起始位置。

你的新sbrk(n)应该只是增加进程尺寸（myproc()->sz）并且返回旧尺寸。

它应该不分配内存——所以你该删除对growproc()的调用（但你仍然要增加进程尺寸）。

尝试猜测更改的结果是什么：什么会break？

做此更改，启动xv6，并且在shell输入echo hi。你该看到像下面：

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201223113138306.png)

"usertrap():..."信息来自trap.c中的user trap handler；

它已经捕获一个异常（不知道如何解决）。确保你理解为什么这个page fault发生。

stval=0x0..04008表明导致page fault的虚拟地址是0x4008。

## 3，具体实现

1）在kernel/sysproc.c中去除growproc()

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201223113442306.png)

# 二、Lazy allocation

## 1，实验要求

更改trap.c中的代码对来自用户空间page fault（在faulting address映射一个新分配的物理内存页）做出响应。

然后返回到用户空间来让进程继续执行。

你该在printf（生成”usertrap():...”信息）之前添加你的代码。

更改可以使echo hi正常需要的任何其他xv6 kernel代码。

这是一些提示：

1. 在usertrap()通过看r\_scause()是13或15，你可以判断fault是否为page fault。

2. r\_stval()返回了risc-v stval寄存器的值，包含了导致page fault的虚拟地址

3. 从vm.c中的uvmalloc()剽窃代码，sbrk()通过growproc()调用uvmalloc。你将需要调用kalloc()和mappages()。

4. 使用 PGROUNDDOWN(va)让faulting虚拟地址低到页边界

5. uvmunmap()将出错；更改它让其在页不映射时，不报错

6. 如果kernel挂掉，在kernel/kernel.asm中查看sepc

7. 使用pgtbl lab中的vmprint函数来打印page table内容

8. 如果你看到错误：”incomplete type proc”，include "spinlock.h" then "proc.h"

如果所有执行正常，你的lazy allocation代码应该导致echo hi正常。你应该至少得到一个page fault（因此懒分配），也可能是两个。

## 2，具体实现

1）修改kernel/trap.c的usertrap()，使得在发生fault page时，给虚拟地址分配物理页

![在这里插入图片描述](https://img-blog.csdnimg.cn/2020122311421177.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70)

2）修改kernel/vm.c的uvmunmap()，让进程销毁时，对于尚未分配实际物理页的虚拟地址，不做处理

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201223114401758.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70)

## 3，执行效果

![](https://img-blog.csdnimg.cn/20201223114817221.png)

# 二、Lazytests and Usertests

## 1，实验要求

我们已经给你提供了lazytests，一个xv6用户程序，可以测试一些特殊场景（对lazy memory allocator造成压力）。更改kernel code让lazytests和usertests都通过测试。

1. 处理sbrk()负参数

2. 如果page-faults的虚拟内存地址比sbrk()分配的大，则杀掉此进程

3. 正确处理fork() parent-to-child内存拷贝

4. 处理这么一种情况：进程传递一个来自sbrk()的有效地址给system call例如read or write，但是那些地址的内存尚未分配

5. 正确处理超出内存：如果kalloc()在page fault handler中失败，杀掉当前进程

6. 处理user stack下的invalid page fault

## 2，具体实现

1）修改kernel/sysproc.c的sys\_sbrk()

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201223171320635.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70)

2）修改kernel/vm.c的uvmunmap()，让进程销毁时，对于尚未分配实际物理页的虚拟地址，不做处理

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201223115945954.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70)

3）修改kernel/vm.c的uvmcopy()，让进程fork时，对于parent进程中尚未分配实际物理页的虚拟地址，不做处理

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201223115856545.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70)

4）修改kernel/vm.c

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201223135744909.png)

5）修改kernel/vm.c中的copyout，write方法将会用到尚未分配物理页的虚拟地址，在copyout中分配物理页。

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201223144621694.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70)

6）修改kernel/vm.c中的copyin，read方法将会用到尚未分配物理页的虚拟地址，在copyin中分配物理页。

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201223144737168.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70)

7）修改kernel/trap.c中的usertrap()

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201223145309395.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70)

## 3，测试效果

lazytests测试

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201223172436409.png)

usertests测试

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201223172535312.png)