****



**MIT xv6实验报告**

**——lab10 : mmap**

**学生姓名 胡轶然**

**学 号 3019244355**

1. **实验目的**
2. 实现mmap和munmap功能，即文件到内存空间的映射和释放。
3. 了解xv6系统中读写文件时，文件到内存空间映射的工作原理。
4. **前期准备**
5. 切换git分支。
6. 阅读指导书文件系统和内存映射相关的知识。
7. 阅读相关代码。
8. **实验内容及实现步骤**

**任务1mmap**

1. **问题描述**

本实验要求实现文件到内存空间的映射和释放，虽然没有明确划分任务，但整体可分为mmap（映射）与munmap（释放）两个部分。

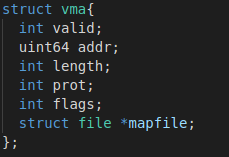
1. **思路与实现步骤**
   1. **Mmap**

根据题意，要实现文件到内存的映射，首先要调用mmap函数为文件关联一块虚拟地址空间(VMA)，当程序访问这块虚拟内存空间中的某一页时，再使用lazy alloc将文件对应内容读入该页中。

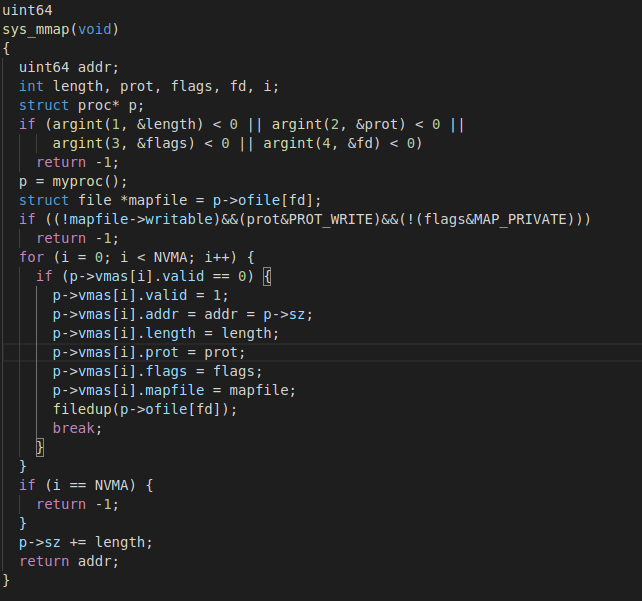
同时，需要注意权限的管理，包括读写权限和共享权限。还要修改fork函数，使得子进程与父进程拥有相同的虚拟地址空间。

具体实现方法如下：

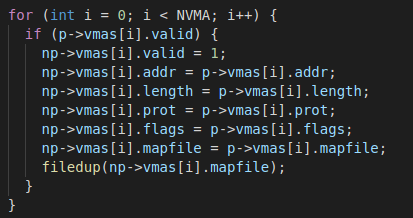
1. 添加系统调用，修改MAKEFILE、user.h、usys.pl、syscall.h等文件，具体内容不再赘述。
2. 添加vma结构体，包括空间使用情况、起始地址、长度、权限、共享标记（回写标记）和文件指针等元素。



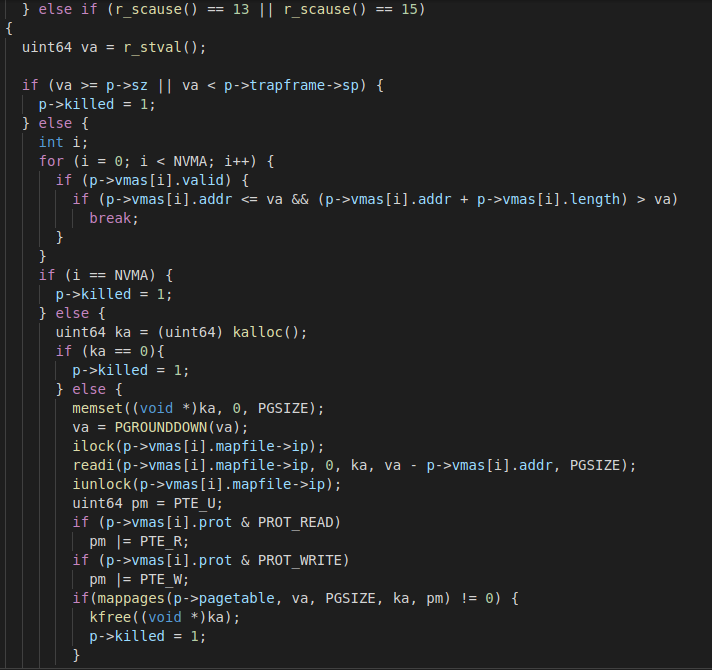
1. 为每个进程预分配一定数量的vma，具体内容包括：
2. 添加NVMA宏定义，规定每个进程最大vma数量。
3. 在proc结构体中添加vma数组，长度为NVMA。
4. 在procalloc函数中，初始化进程的所有vma。
5. 添加sys\_mmap系统调用，具体内容包括
6. 接收文件标识、权限、共享标记、长度等参数。
7. 获取文件指针，若文件不提供写权限，而调用进程又允许向文件回写内容的话，发生权限冲突，异常返回”-1”。
8. 在进程的vma数组中顺序找到第一个可用的vma，将文件信息存入vma中，同时增大文件的引用次数。
9. 增大进程内存空间(p->sz)，但不分配页面(lazy allocation)。



1. 为保证父、子进程的vma内容相同，执行以下改动：
   1. 向proc.c文件中添加头文件fcntl.h。
   2. 在fork函数中添加代码，将父进程vma数组的内容复制到子进程中。
   3. 增大文件引用数(filedup)。



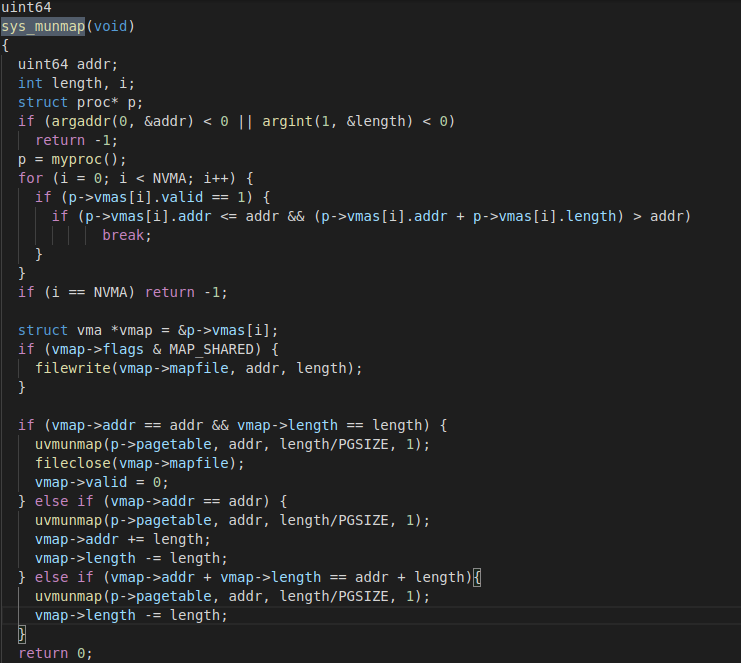
1. 向trap.c文件中添加sleeplock.h、fs.h、file.h和fcntl.h等头文件，然后调整usertrap函数中的缺页逻辑，允许lazy allocation，具体内容为：
2. 当发生内存加载或回写异常，且异常位置位于某个vma中，则使用kalloc分配一个物理内存页ka。
3. 使用宏定义PGROUNDDOWN定位到虚拟页顶部，并从文件相应位置读取一页内容，存入ka。
4. 调用mappages函数，将物理页ka映射到虚拟页表中。
5. 根据vma的权限为虚拟页分配相应的读写权限。
6. 在vm.c文件的uvmunmap和uvmcopy函数中，取消对缺页异常的panic。



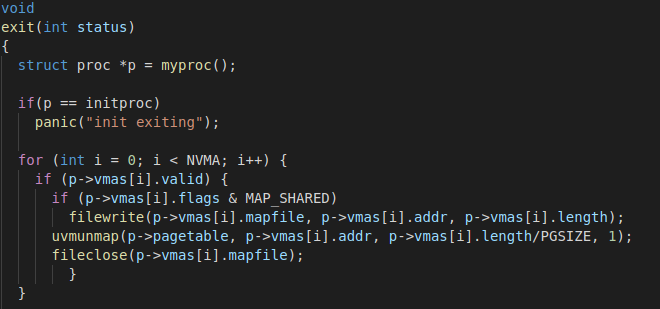
* 1. Munmap

本部分要实现vma的释放逻辑。从进程周期角度，Vma的释放可分为两类:在进程运行过程中释放单个vma的部分或全部内存，和在进程结束时释放所有vma的全部内容，而释放单个vma又分为释放头部片段、尾部片段和全部内容。

1. 添加系统调用” sys\_munmap” ，修改MAKEFILE、user.h、usys.pl、syscall.h等文件，具体内容不再赘述。
2. 编辑sys\_munmap函数，实现对单个vma的部分或全部释放。具体内容如下
3. 接收传入的起始地址和释放空间长度等参数。
4. 遍历进程的vma数组，找出要释放的内存片段所在的vma。
5. 若该vma要求回写（持有回写权限和共享标记），则将vma中的数据写入响应文件中。
6. 调用uvmunmap函数释放指定内存片段，本实验将释放单个vma分为三种类型：
   * 1. 释放头部片段，调用uvmunmap函数释放内存，并将vma的起始地址后移，同时减小vma长度。
     2. 释放尾部片段，调用uvmunmap函数释放内存，减小vma长度。
     3. 释放某个vma的全部内容，修改vma有效位”valid”，将vma设为一个空闲的虚拟内存片段，还要调用fileclose函数，关闭对应文件。



1. 编辑exit函数，释放进程vma数组中的所有vma。具体内容包括：
   1. 若vma持有MAP\_SHARED标记，则将文件内容回写到磁盘中。
   2. 调用uvmunmap函数，关闭文件。此部分操作与上一小节相近，不再赘述。



1. **问题与解决方法**

**问题一**：没有理解vma结构体中，权限(prot)和标记(flags)的区别。

**解决方法**：

仔细阅读指导书并查阅相关资料后明白，**prot类似内存页pte的权限位，存储着虚拟内存的读、写权限情况；而flags则存储着”MAP\_SHARED”标记，记录了在释放虚拟内存时，是否要将文件的改动回写到磁盘中。**

**只有持有写权限的Vma才能拥有MAP\_SHARED标记**，因为只读文件不会被修改，也就没有将改动回写到磁盘的需求。

**困难二**：忘记删去uvmcopy函数中对PTE\_V权限的检查代码，触发了panic：page not present

**解决方法**：

注释了uvmcopy和uvmunmap中对pte vaild权限的检查，因为**lazy allocation允许扩大进程内存空间而暂时不分配内存页。**

**困难三**：在最初编写sys\_munmap函数时，不清楚如何编写释放vma的逻辑。包括**如何构建vma结构体，vma结构体的虚拟地址范围如何确定，如何区分和处理三种释放vma的逻辑（头、尾和全部）。**

**解决方案**：

首先，阅读指导书，查阅资料了解到vma的大小是可变的，因此在要**释放vma的部分内存空间时，只需要在原有vma结构体上修改虚拟地址范围即可。**

阅读指导书，发现munmap通过起始地址(address)和地址长度(length)两个参数来指定要释放的空间。且该**函数只会释放vma头部或尾部的内存，而不会从中间将vma切分成两个vma。**

综上所述，在vma结构体中加入起始地址(addr)和空间大小(length)两个属性，释放vma时，**计算vma的起始地址和末尾地址（初末地址），与要释放空间的初末地址比较，即可确定要释放vma的头部、尾部还是全部内容。**计算出vma新的初末地址，并相应地修改addr和length属性即可。

1. **结果**



1. **实验结果**

通过make grade测试。

