仓库链接：https://github.com/zhaoyihui233/mit-6.S081-lab-2021/tree/main/lab10%20Mmap

# 1 mmap

## 1.1实验内容

本实验需要我们实现两个个Unix操作系统中常见的系统调用，mmap和munmap，它们的功能是把文件映射到用户空间中，便于用户可以直接通过内存来修改和访问文件。

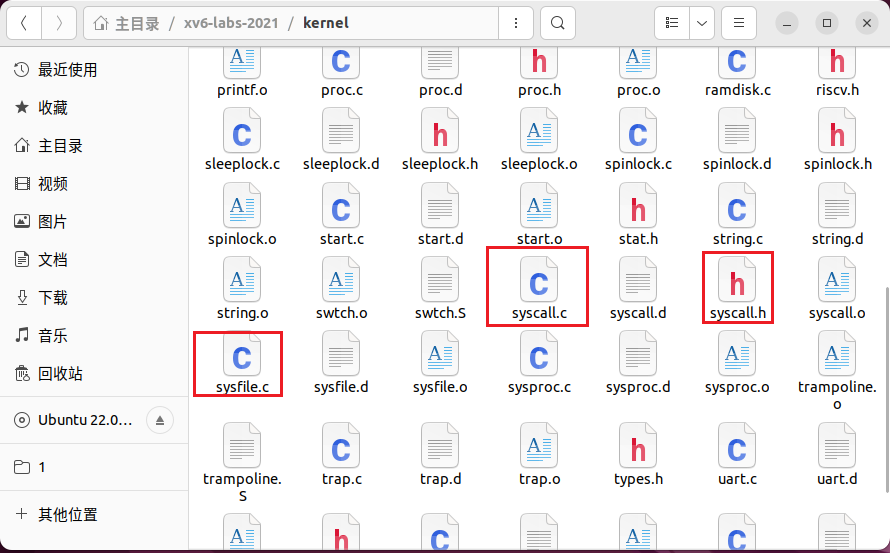
void \*mmap(void \*addr, size\_t length, int prot, int flags, int fd, off\_t offset);

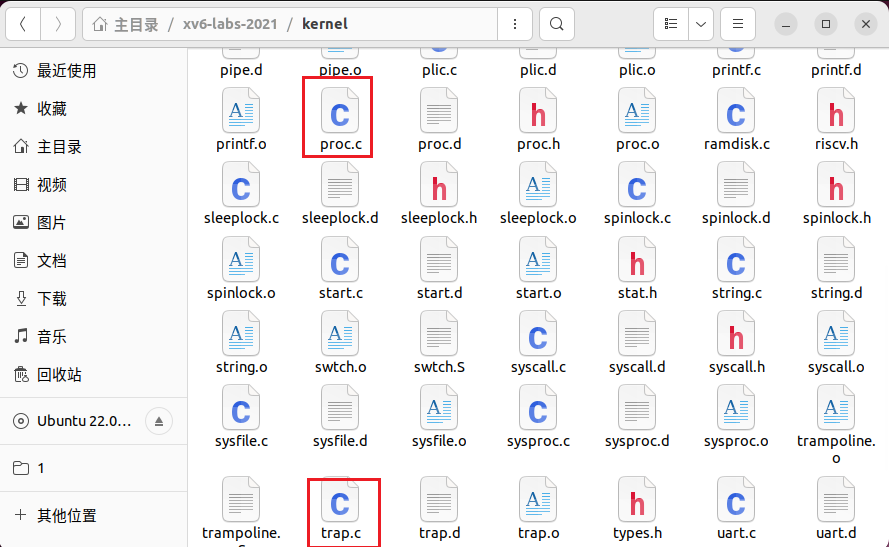
addr是用户地址空间中映射区的起始地址，由操作系统指定，length是用户地址空间映射区的长度，也是文件中被映射部分的长度，prot可以为PROT\_READ, PROT\_WRITE和PROT\_EXEC，表示当前进程可以对映射区操作的权限。flags可以为MAP\_SHARED、MAP\_PRIVATE，表示再取消映射时，是否将修改更新到文件，fd是被映射文件的文件描述符，offset是被映射文件的偏移，当前实验中的测试默认addr和offset都为0。

int munmap(void \*addr, size\_t length);

意思是取消从addr开始，长度为length的文件映射，取消映射可能只取消某个映射区的部分映射。

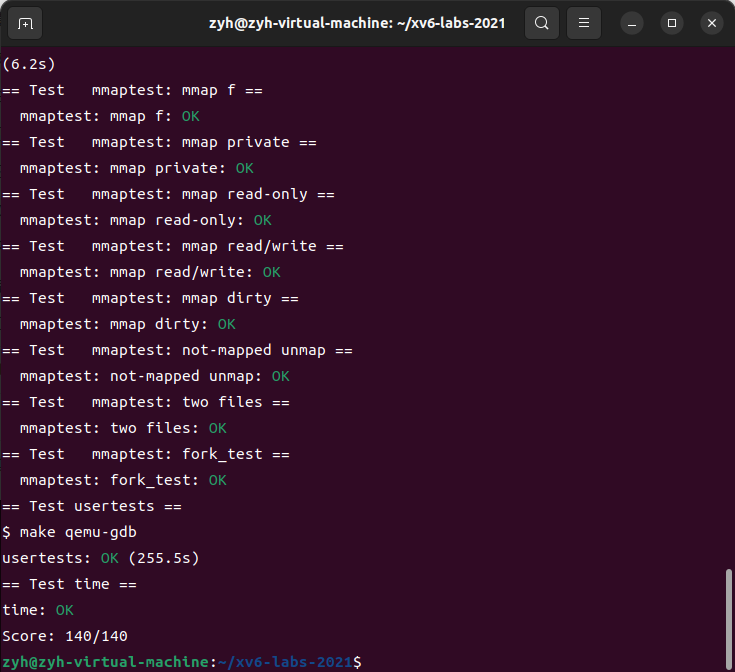
## 1.2代码位置与截图



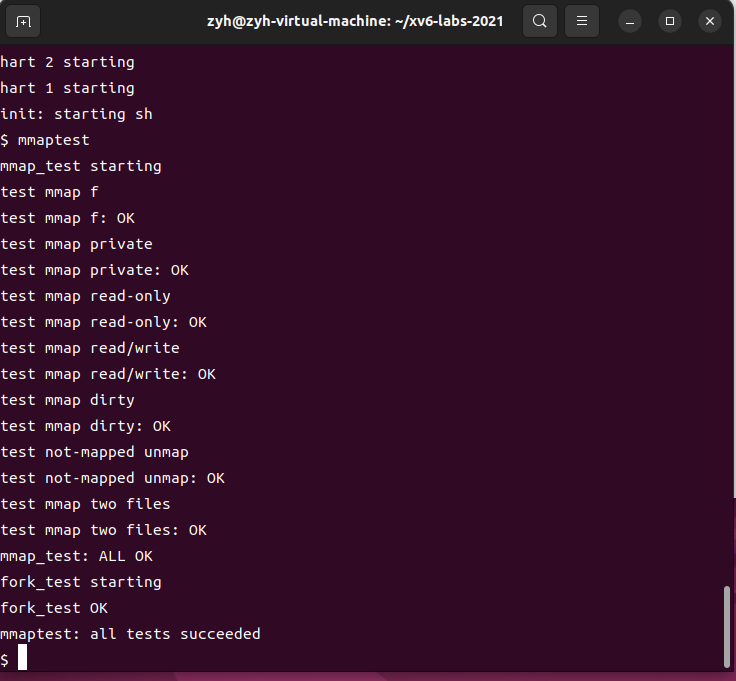


## 1.3实验结果

make grade



make qemu -> mmaptest



./grade-lab-mmap mmap



## 1.4代码实现

### 1.4.1实现思路

首先是实验前的准备工作：在Makefile中的UPROGS下添加\_mmaptest，添加mmap和munmap系统调用。在kernel/syscall.c，kernel/syscall.h，kernel/sysfile.c，user/usys.pl，user/user.h各个文件中添加内容，并在user/user.h中声明系统调用。

然后，对于缺页错误(page faults)，使用懒分配的思想，进程需要时才分配物理页。这样能够快速建立大文件的mmap，或者大于物理内存的文件的mmap才有可能建立。每个进程应该记录建立mmap映射的信息，于是我们定义了一个VMA结构体，记录映射空间的起始地址，映射空间的长度，以及权限，被映射的文件，VMA的长度固定为16。

实现mmap（kernel/sysfile.c）：在进程的地址空间中找一块未使用的内存用于映射文件，并将VMA添加到进程中。mmap应该增加文件的引用计数，保证文件关闭之后，结构体仍然存在。mmap中不建立虚拟地址与物理地址的映射关系，只是将进程结构体proc的结构中vma结构体进行初始化，载入一些信息，真正建立映射关系在trap.c中断处理程序中，按需分配(lazy allocation) page。mmap需要返回映射空间的起始地址。

修改usertrap函数（kernel/trap.c）：在缺页错误处理中添加代码，为映射区域添加分配一页物理地址(4KB)，读取被映射文件4KB的大小作为一页映射到用户地址空间。读取文件使用readi()，它采用一个偏移量来读取文件，记得分配的一个page设置对应的权限。

实现munmap（kernel/sysfile.c）：找到地址范围内的VMA，并取消指定页面的映射(hints: uvmunmap())，如果munmap取消了一个mmap所有映射的页面，那么修改相应文件的引用计数。如果未映射的页面已经被修改，且该文件的映射模式为MAP\_SHARED，那么需要将修改的页面写回文件，使用filewrite中进行操作。意思是取消从逻辑地址addr开始，长度为length的文件映射，这个函数保证不支持在映射范围的中间挖洞，只能在开始或结尾部分取消全部映射。假设我们有一个 [1, 100]的映射范围，那么如果我们想要取消 [l, r] 范围的映射，需要符合l = 1 && r <= 100或者l >= 100 && r = 100。

修改exit函数（kernel/proc.c），就好像调用了munmap一样。

修改fork（kernel/proc.c）以确保子进程具有与父进程相同的映射区域。同时增加VMA结构中文件的引用计数。在子进程的缺页错误处理程序中，可以分配一个新的物理页面，而不是与父进程共享页面。

实际情况，文件映射区的长度(length)不一定是是页(4K)的整数倍，为了避免回收物理内存时出现的边界的问题，所以每次分配物理内存都是整数页分配，且分配PGROUNDUP(length)的页数，回收映射区的物理内存时也是按页进行回收，这样虽然比较方便分配与回收操作，但是会导致内存的页内空间的浪费。文件1映射区使用了page 1和page 2，但是page 2的内存并没有使用，也无法分配给其他文件映射区使用，所以造成了一定的浪费。lab中的测试代码解除映射时addr和length都是页对齐的，所以比较好操作。