操作系统课程设计

Xv6实验报告

Lab9 页表

|  |  |
| --- | --- |
| Name | 朱从周 |
| ID | 2351893 |
| Number | 42028701 |
| major | 软件工程 |



同济大学

Tongji University

1. **环境搭建**
2. 下载VMware虚拟机，并且下载 Ubuntu 镜像并安装 Ubuntu 系统。
3. 启动Ubuntu，安装本项目所需的所有软件，运行：

$ sudo apt-get update && sudo apt-get upgrade

$ sudo apt-get install git build-essential gdb-multiarch qemu-system-misc gcc-riscv64-linux-gnu binutils-riscv64-linux-gnu

1. 下载xv6内核源码

$ git clone git://github.com/mit-pdos/xv6-riscv.git

1. 更新镜像源

$ sudo nano /etc/apt/sources.list

$ sudo apt-get update

1. 获取源码

$ git clone git://g.csail.mit.edu/xv6-labs-2021

$ cd xv6-labs-2021

$ git checkout fs

**1. 实验目的**

本次实验的目的是扩展 xv6 文件系统，使其支持更大的文件大小。目前 xv6 文件大小受限于 268 个块，或 268 \* BSIZE 字节（在 xv6 中，BSIZE 为 1024）。这个限制是因为 xv6 inode 包含 12 个“直接”块号和一个“单间接”块号，它引用一个可以容纳多达 256 个块号的块，总共为 12+256=268 个块。

**2. 实验内容**

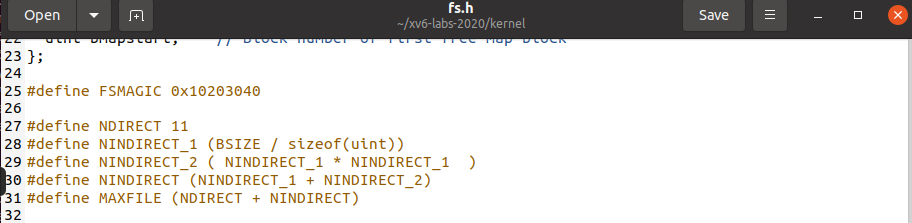
得到仓库源码分支：

git clone git://g.csail.mit.edu/xv6-labs-2020

git checkout fs

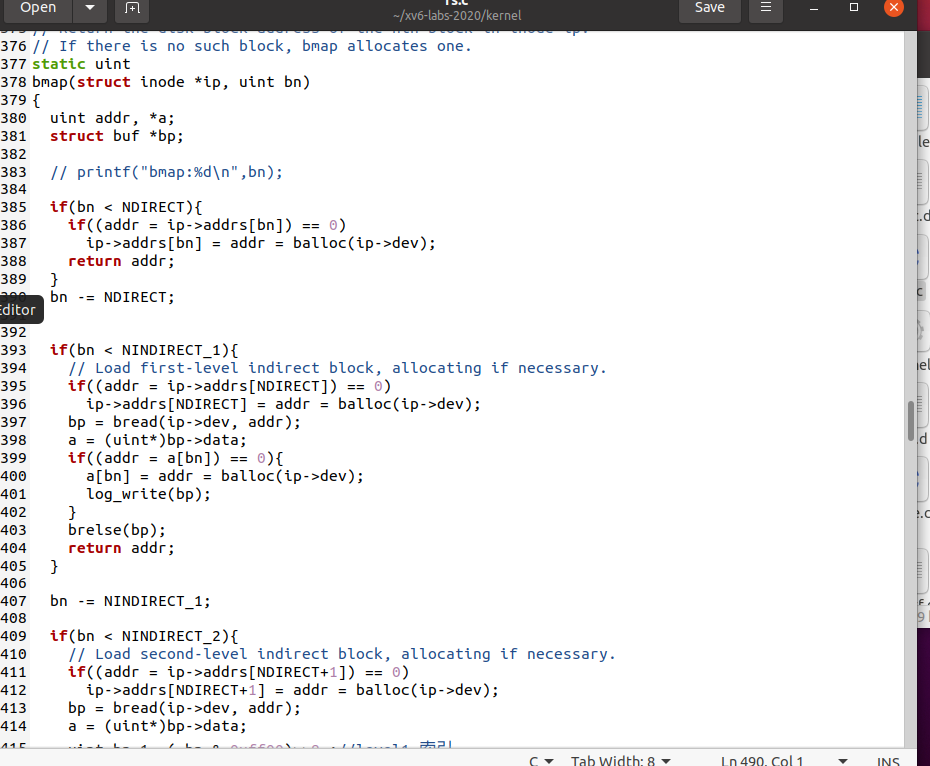
**large file：**

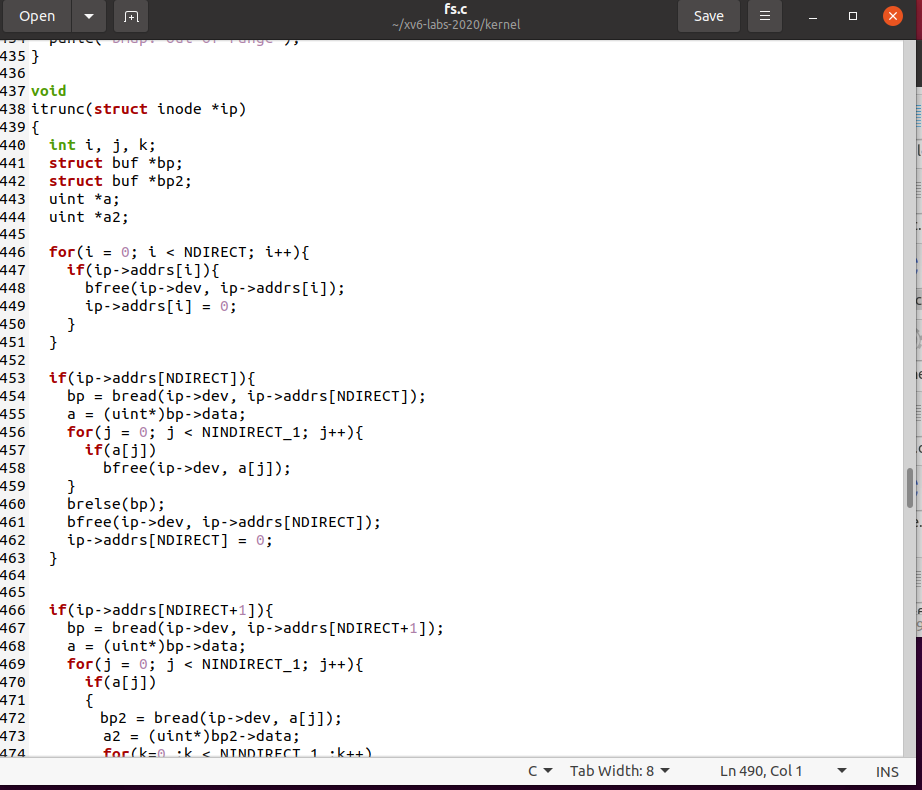
首先修改fs.h中的宏定义



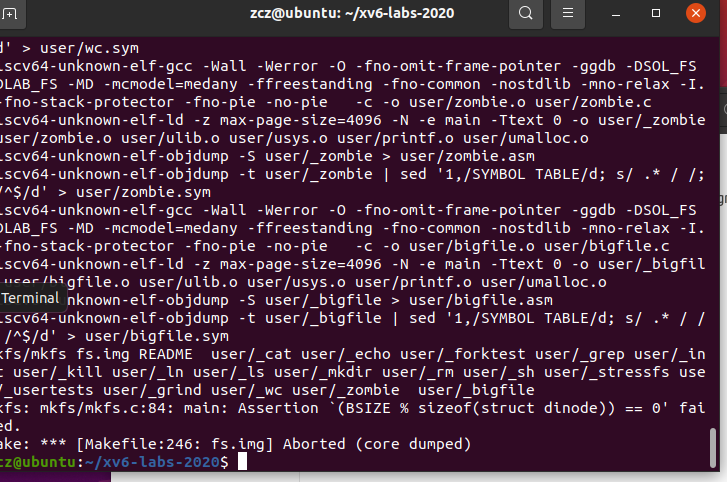
NINDIRECT\_1是第一级间接访问的block数目，NINDIRECT\_2是第二季间接访问的block数目

bmap和itrunc代码





**测试结果：**



报错的原因是内核空间不足，是因为该虚拟机的分配的内存不足导致的，之后我会进行分配更大的虚拟机进行结果测试。

**Symbolic links (**[**moderate**](https://link.zhihu.com/?target=https%3A//pdos.csail.mit.edu/6.828/2020/labs/guidance.html)**)**

**1、**[**系统调用**](https://zhida.zhihu.com/search?content_id=223651999&content_type=Article&match_order=1&q=%E7%B3%BB%E7%BB%9F%E8%B0%83%E7%94%A8&zhida_source=entity)**的方式实现的，所以我们需要实现一个名为 symlink() 的系统调用，供 Caller 使用。声明在 user/user.h 中，**

**int** **symlink**(**const** **char\***, **const** **char\***);

symlink() 接受两个参数，前者为 target ，意为符号链接要指向文件路径名为 target 的 inode ；后者为 path ，也就是新符号链接的路径名。在 xv6 中文件路径名存放在 inode 文件地址空间的起始位置，即第一块 block

**2、并且在 user/usys.pl 中追加 symlink entry ，使 xv6 能够找到 symlink() 系统调用，**

entry("symlink");

**3、之后，在 kernel/syscall.h 中定义宏 SYS\_symlink ，**

**#define SYS\_symlink 22**

**4、在 kernel/syscall.c 中追加 extern 声明，并将 sys\_symlink() 加入序列中，**

**extern** uint64 **sys\_symlink**(**void**);

**static** **uint64** (**\***syscalls[])(**void**) **=** {

...

[SYS\_symlink] sys\_symlink,

};

**5、最后，还需在 kernel/stat.h 中定义宏 T\_SYMLINK ，**

**#define T\_SYMLINK 4** */\*\* 符号链接标记位 \*/*

**6、在 kernel/fcntl.h 中定义宏 O\_NOFOLLOW ，**

**#define O\_NOFOLLOW 0x800**

**7、在 Makefile 中找到 UPROGS 选项，在其后追加，**

UPROGS=\

$U/\_cat\

...

$U/\_zombie\

$U/\_symlinktest\

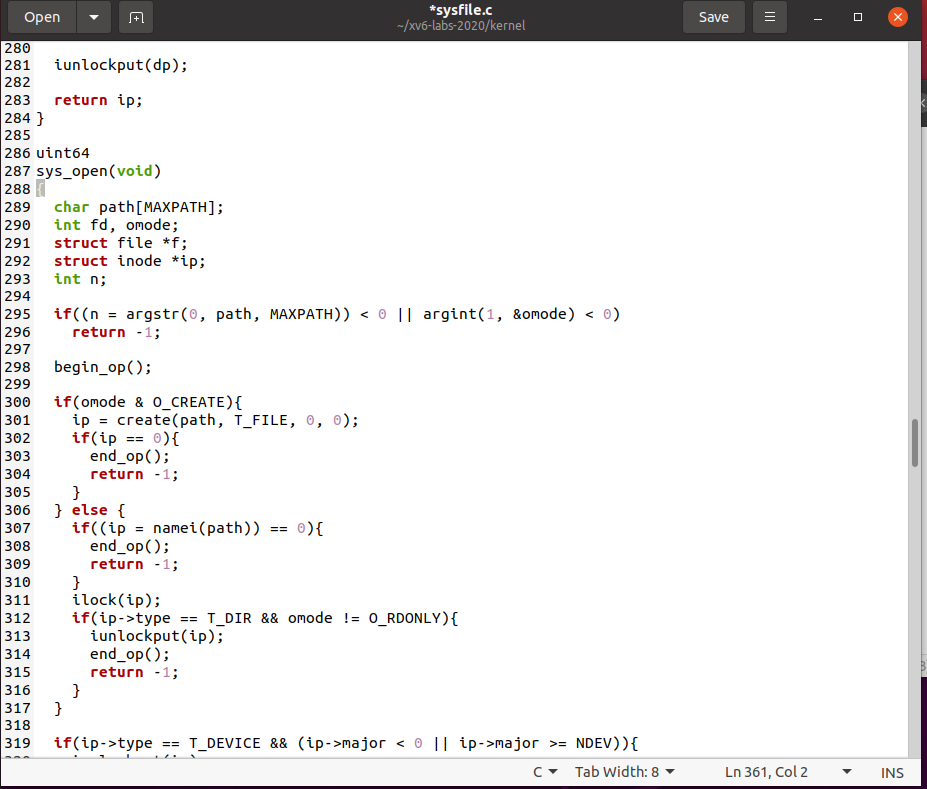
**完成 symlinktest 的测试单元编译工作**

**8、根据要求，在kernel/sysfile.c中，建立链接：**

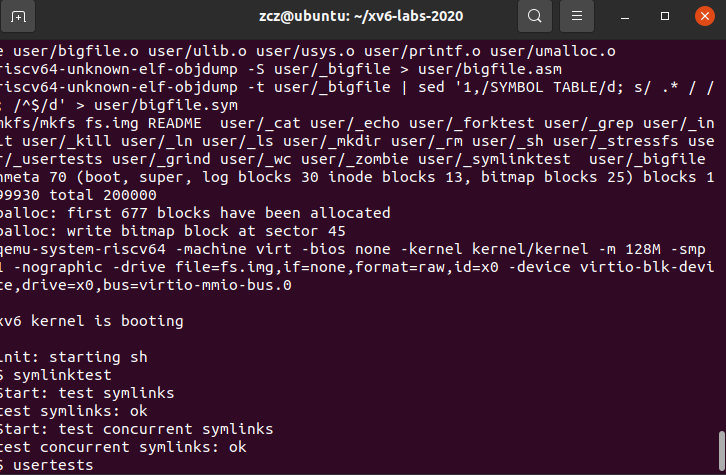
**在 sys\_symlink() 中定义了 target 和 path 字符串，用来接收从 symlink() 中传来的 target 和 path**

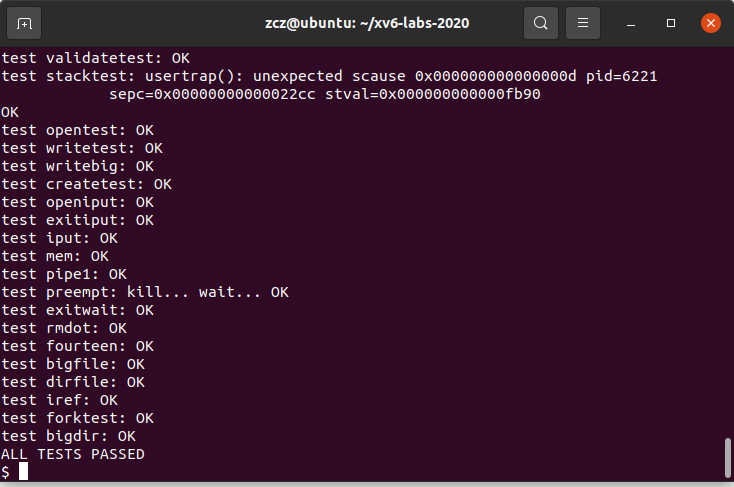


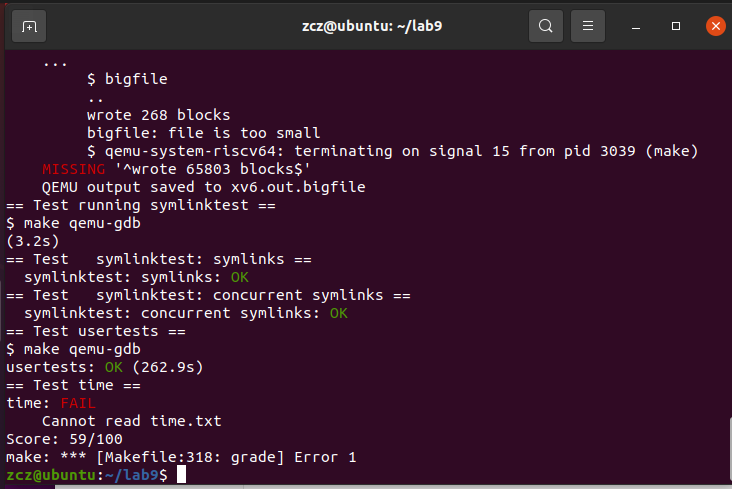
kernel/sysfile.c中sys\_oepn()来追根溯源打开文件：



**测试结果：**







测试分数低是因为大文件要求没有满足，后续会再次新的虚拟机中进行实验。

**3. 实验心得**

在本次实验中，我成功地向 xv6 操作系统添加了符号链接（软链接）的支持。符号链接是一种特殊类型的文件，可以跨越磁盘设备引用其他文件。在这个实验中，我实现了 symlink(char \*target, char \*path) 系统调用，该调用可以创建一个新的符号链接文件，将其指向目标文件。此外，为了防止符号链接文件溯源的过程中陷入死循环，我们还额外考虑了循环的深度限制和成环检测，这提高了我们的效率，并且保障了程序的安全性。