**华东师范大学数据学院上机实践报告**

|  |  |  |
| --- | --- | --- |
| **课程名称**： 操作系统 | **年级**：2019级 | **上机实践成绩**： |
| **指导教师**：翁楚良 | **姓名**：周辛娜 |  |
| **上机实践名称**： I/O subsystem | **学号**： | **上机实践日期**：6.8 |
| **上机实践编号**：3 |  |  |

1. **目的**
2. 熟悉类UNIX系统的I/O设备管理
3. 熟悉MINIX块设备驱动
4. 熟悉MINIX RAM盘
5. **内容与设计思想**

在MINIX3中安装一块X MB大小的RAM盘（minix中已有6 块用户可用RAM盘，7块系统保留RAM盘），可以挂载并且存取文件操作。

测试RAM盘和DISK盘的文件读写速度，分析其读写速度差异原因（可用图表形式体现在实验报告中）。

1. **使用环境**

VMware Workstation Pro

MobaXterm\_Personal\_20.6

1. **实验过程**
2. 增加RAM盘

修改/usr/src/minix/drivers/storage/memory/memory.c ，增加默认的用户RAM盘数：RAMDISKS=7。

重新编译内核，重启reboot。

创建设备mknod /dev/myram b 1 13，查看设备是否创建成功输入 ls /dev/ | grep ram。

实现buildmyram初始化工具（用于分配容量）。

参考/usr/src/minix/commands/ramdisk/ramdisk.c，实现buildmyram.c，但是需要将KB单位修改成MB。

编译buildmyram.c文件，然后执行命令： buildmyram /dev/myram。创建一个RAM盘。

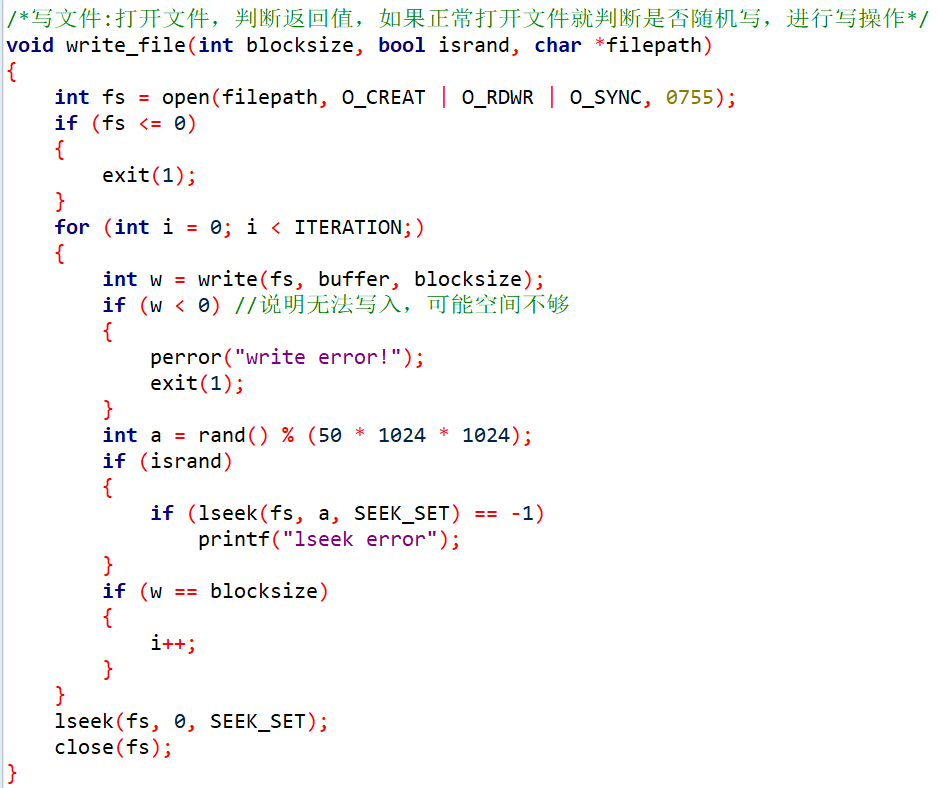
在ram盘上创建内存文件系统，mkfs.mfs /dev/myram。

将ram盘挂载到用户目录下，mount /dev/myram /root/myram,查看是否 挂在成功：输入df。

重启后用户自定义的ram盘内容会丢失，需要重新设置大小，创建文件系统，并挂载。

1. 性能测试

首先是代码的编写：

1)

头文件：#include <unistd.h>  
定义函数：size\_t write (int fd, const void \* buf, size\_t count);  
函数说明：write()会把参数buf 所指的内存写入count 个字节到参数fd 所指的文件内。 当然, 文件读写位置也会随之移动。返回值：如果顺利write()会返回实际写入的字节数。当有错误发生时则返回-1, 错误代码存入errno 中。

否则，随机写的话，由于每个file均为50MB，故每个file共50\*1024\*1024字节，便需要随机地选择文件写的位置，这里a=rand()%(50\*1024\*1024),代表应该从文件中的第a个字节处开始写，然后再利用lseek函数。

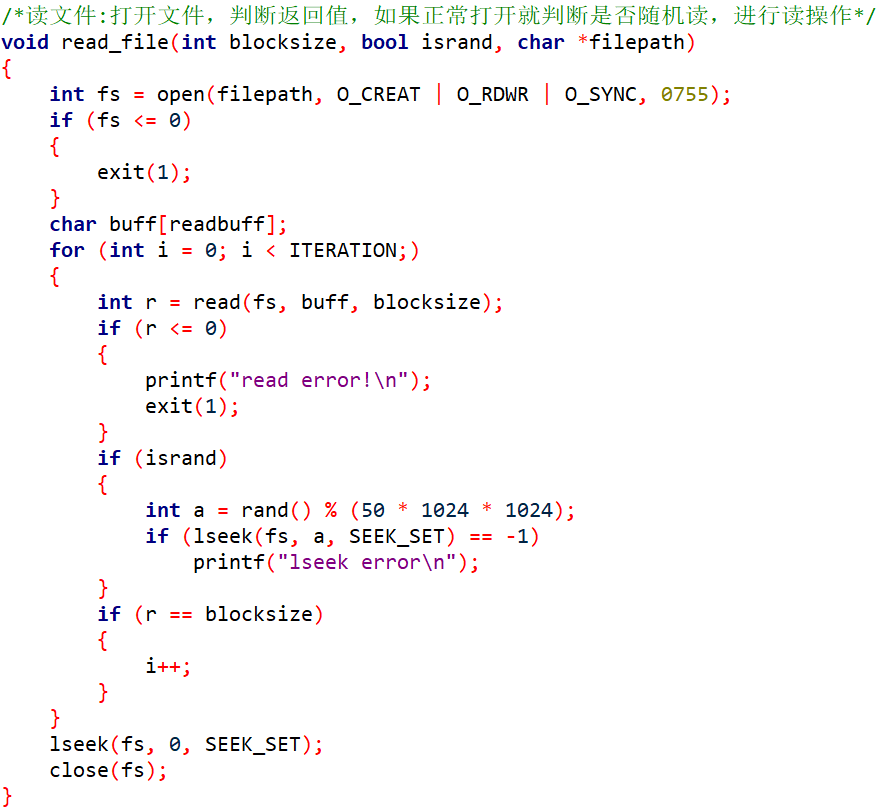
头文件：#include <sys/types.h>    #include <unistd.h>  
定义函数：off\_t lseek(int fildes, off\_t offset, int whence);

函数说明：每一个已打开的文件都有一个读写位置, 当打开文件时通常其读写位置是指向文件开头, lseek()用来控制该文件的读写位置. 参数fildes 为已打开的文件描述词, 参数offset 为根据参数whence来移动读写位置的位移数.  
参数whence为SEEK\_SET 参数offset 即为新的读写位置。  
返回值：当调用成功时则返回目前的读写位置, 也就是距离文件开头多少个字节. 若有错误则返回-1, errno 会存放错误代码.

若可以写入blocksize字节的数据，且无论是顺序写还是随机写则i++，继续在上次的位置写。

随机读写时，可以采用lseek重新定位文件指针；顺序读写时，默认文件指针自动移动，当读到文件末尾时，可以用lseek返回文件头

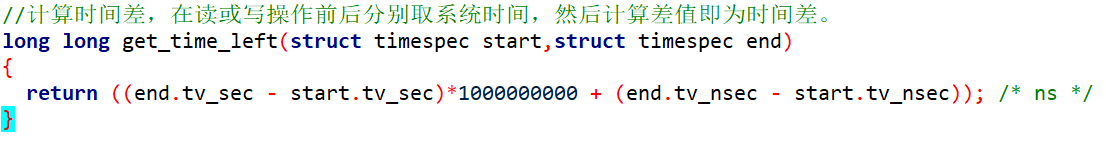
2)



头文件：#include <unistd.h>  
定义函数：ssize\_t read(int fd, void \* buf, size\_t count);  
函数说明：read()会把参数fd 所指的文件传送count 个字节到buf 指针所指的内存中. 若参数count 为0, 则read()不会有作用并返回0. 返回值为实际读取到的字节数, 如果返回0, 表示已到达文件尾或是无可读取的数据,此外文件读写位置会随读取到的字节移动。

Read\_file的逻辑同write\_file差不多，write函数换成read函数即可，再构造读的数组。

3)



struct timespec {

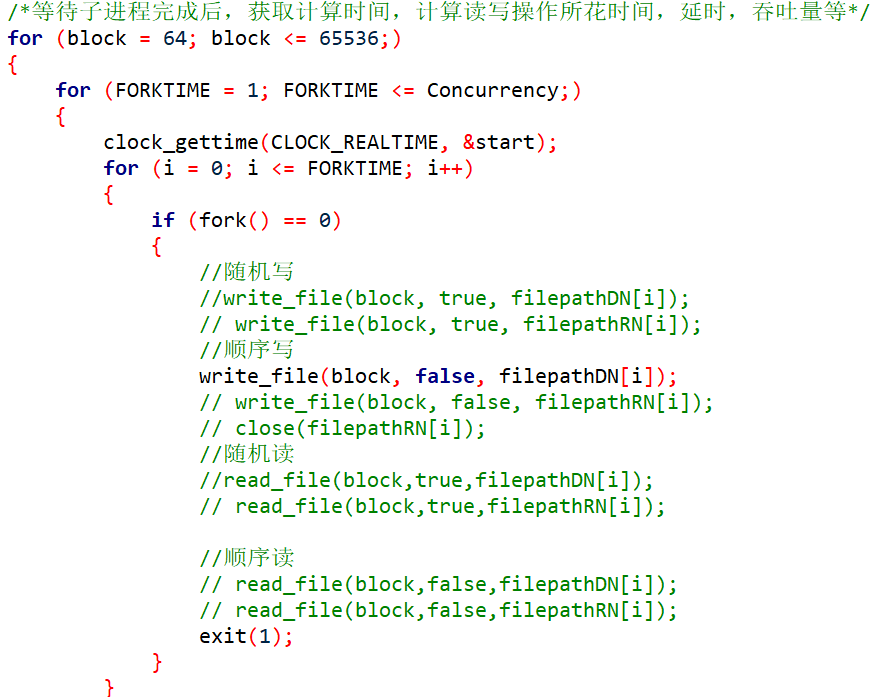
time\_t tv\_sec; // seconds

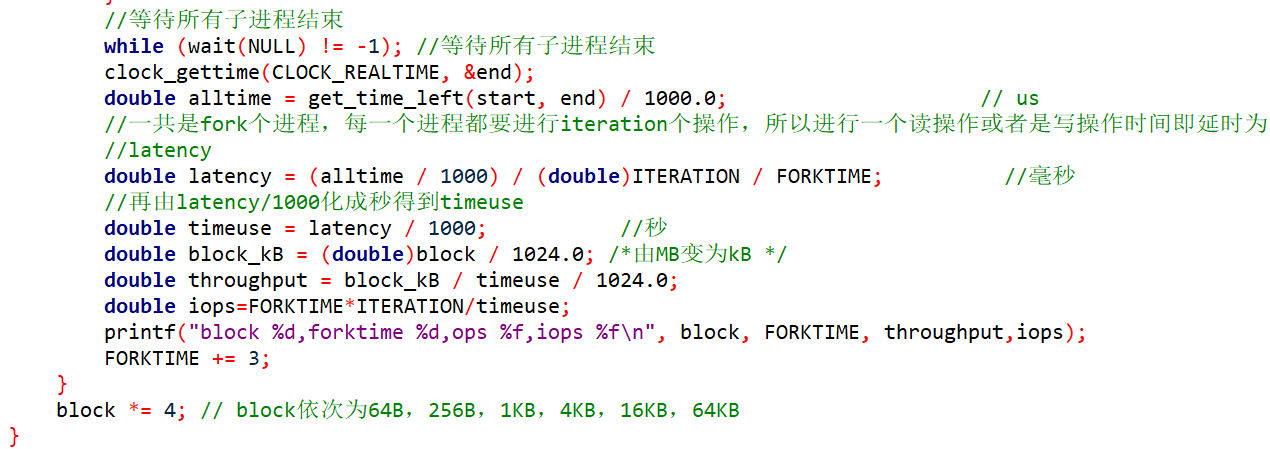
long tv\_nsec; // nanoseconds

};

struct timespec被定义在#include <time.h>中，有两个成员，一个是秒，一个是纳秒。get\_time\_left即让timespec的两个成员变量相减即可，最后单位为ns

1. main函数主体：

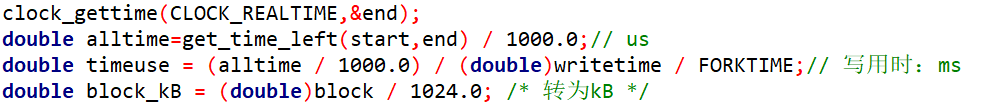




1. **第一个性能测试：**

RAM盘和Disk盘的性能测试中，需要采用多进程并发的同步读写，并发数要增加到设备接近“饱和”状态（吞吐量难以继续提升，但是I/O延时恶化）。在出现饱和前，总吞吐量随着并发数线性增长。

取定blocksize=256，concurrency取1、4、7、10、13、16、19，主要是throughput和iops的计算问题，通过get\_time\_left得到总的时间，除以fork的进程数，除以读或写操作循环的次数，得到进行一次读或写操作的时间，即延时，它就是一个中间量, 再把时间化成秒，得到timeuse



再通过下面两个式子可以分别求得throughput和iops

ac463d7d3b12f5d3ed40b9844ee46a8

实验结果：

分析：在不同的blocksize和ram盘disk盘顺序读、随机读的情况下基本上都是throughput先到达瓶颈，iops线性增加，这是因为I/O延时恶化，吞吐量难以增大，在达到饱和之前，总吞吐量随着并发数线性增长。

1. **第二个性能测试**

性能测试的二个变量为“块大小”（推荐 64B/256B/1KB/4KB/16KB/64KB）和“块扫描方式”（顺序/随机）。这里main函数主要有三层循环，block变化,fork的进程个数变化，最后得到的throughput再取平均。

实验结果：

分析：可以看到，在这四种情况下，均是ram盘的吞吐量高于disk盘吞吐量。ram盘使用预先分配的主存来存储数据块。ram盘具有快速存取的优点，没有寻道和旋转延迟，而disk盘有寻道和旋转延迟。

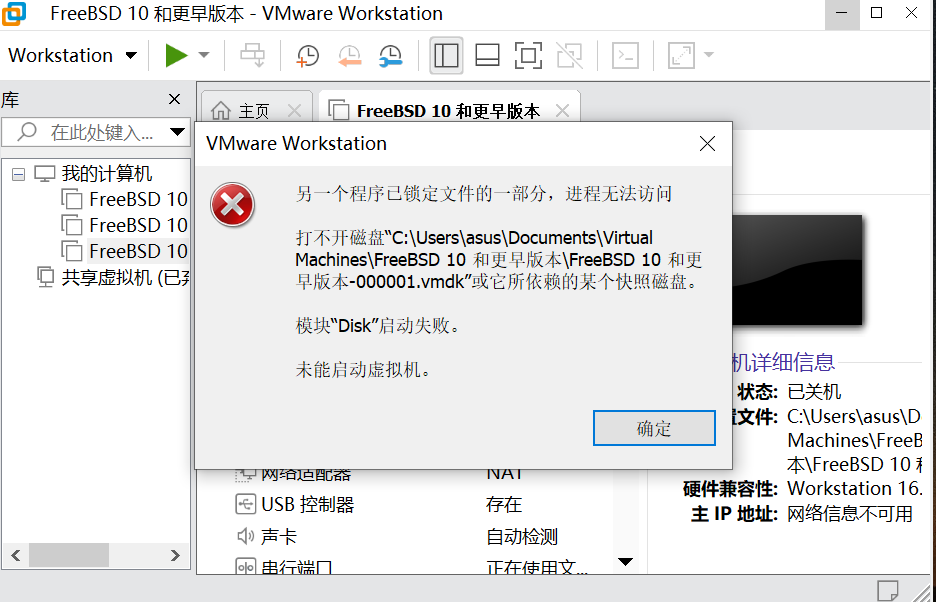
1. **总结**

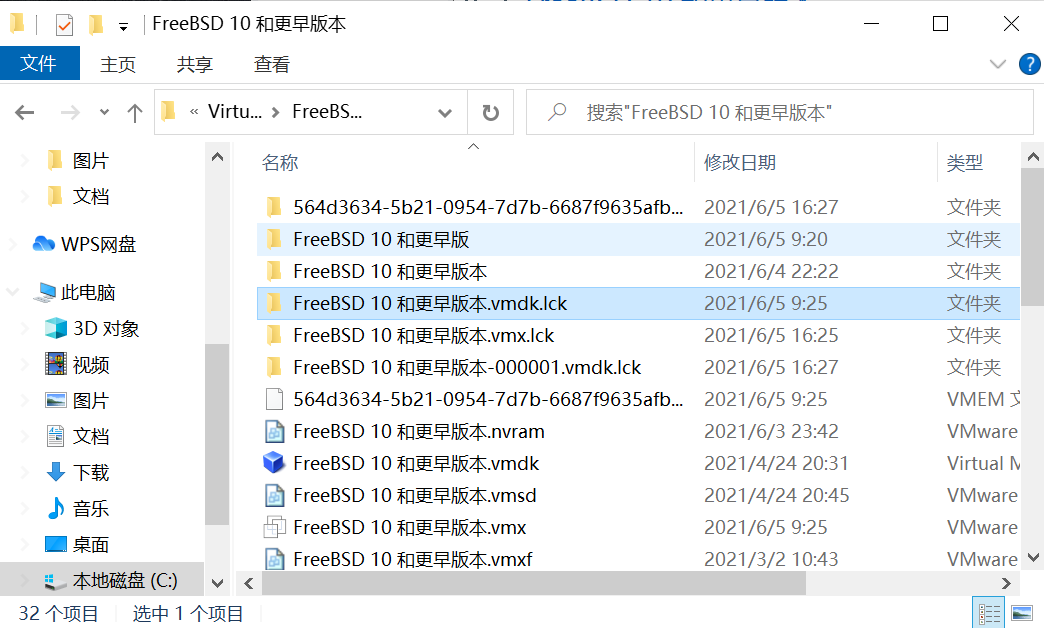
在进行实验的过程中，也出现了一些问题：

1.在增加RAM盘修改/usr/src/minix/drivers/storage/memory/memory.c ，增加默认的用户 RAM盘数：RAMDISKS=7后，直接reboot，没有make build编译，后面便出现了device not configue的问题。

2.解决了make build的问题后，发现出现了segmentation fault，原因在于虚拟机设置的内存太小，后面设置成了2GB。

3.由于电脑没电，虚拟机未正常关机，便出现了以下问题





只需要把vmdk.lck文件夹删了或者改名