华东师范大学数据科学与工程学院实验报告

课程名称：操作系统	**年级**：21 级	**上机实践成绩**：
指导教师：翁楚良	**姓名**：杨茜雅	**学号**：10215501435
上机实践名称： Lab03		**上机实践日期**：2023.5
上机实践编号：3	**组号**：	**上机实践时间**：2023.5

一、 实验目的

- 熟悉类 UNIX 系统的 I/O 设备管理
- 熟悉 MINIX 块设备驱动
- 熟悉 MINIX RAM 盘

二、 实验任务

- 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）。

三、 使用环境

物理机：Windows10
虚拟机：Minix3
虚拟机软件：Vmware
代码编辑：VScode
物理机与虚拟机文件传输：FileZilla

四、 注意事项

- 使用 posix 函数 open 打开文件，利用 O_SYNC 参数使得 write/read 操作为同步模式。
- 一定要检查 write/read 函数的返回值，以及写入的字节数目，确定是否成功。
- 为了简化实验，可以为每个进程分配一个独立的文件。为了减小主机操作系
- 统的缓存机制造成的误差，文件总大小越大越好（例如 300MB）。
- 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。
- 每组的读写需要反复持续一段时间，过短的时间会造成误差较大。
- 通常情况下，7~15 个进程达到饱和，吞吐量不会高于 700MB/s（ram 盘顺序读写）。
- 如果 minix 虚拟机建在 SSD 下，会导致随机和顺序的差距减小，所以最好把虚拟机放在机械硬盘上，实验效果更明显。

五、 实验过程

一、 增加 RAM 盘

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

修改前：

```
/* ramdisks (/dev/ram*) */
#define RAMDISKS     6
```

修改后：

```
/* ramdisks (/dev/ram*) */
#define RAMDISKS     7
```

2、 重修编译内核，重启 reboot

```
# cd /usr/src
# make build_
configinstall ===> etc/mtree
    install  //etc/mtree/NetBSD.dist
    install  //etc/mtree/special
    install  //usr/lib/fonts
do-hdboot ===> releasetools
install -N /usr/src/etc -c -r ../minix/servers/ds/ds /boot/minix/.temp/mod01_ds
install -N /usr/src/etc -c -r ../minix/servers/rs/rs /boot/minix/.temp/mod02_rs
install -N /usr/src/etc -c -r ../minix/servers/pm/pm /boot/minix/.temp/mod03_pm
install -N /usr/src/etc -c -r ../minix/servers/sched/sched /boot/minix/.temp/mod
04_sched
install -N /usr/src/etc -c -r ../minix/servers/vfs/vfs /boot/minix/.temp/mod05_v
fs
install -N /usr/src/etc -c -r ../minix/drivers/storage/memory/memory /boot/minix
/.temp/mod06_memory
install -N /usr/src/etc -c -r ../minix/drivers/tty/tty/tty /boot/minix/.temp/mod
07_tty
install -N /usr/src/etc -c -r ../minix/fs/mfs/mfs /boot/minix/.temp/mod08_mfs
install -N /usr/src/etc -c -r ../minix/servers/vm/vm /boot/minix/.temp/mod09_vm
install -N /usr/src/etc -c -r ../minix/fs/pfs/pfs /boot/minix/.temp/mod10_pfs
install -N /usr/src/etc -c -r ../sbin/init/init /boot/minix/.temp/mod11_init
rm /dev/c0d3p0s0:/boot/minix/3.3.0r5
Done.
Build started at:  Sat May  6 19:20:42 GMT 2023
Build finished at: Sat May  6 19:34:19 GMT 2023
# reboot
```

3、 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。创建块设备/dev/myram,主设备号为 1,次设备号为 13

```
# mknod /dev/myram b 1 13
# ls / dev/ | grep ram
ls: dev/: No such file or directory
# ls /dev/ | grep ram
myram
ram
ram0
ram1
ram2
ram3
ram4
ram5
```

4、 实现buildmyram 初始化工具（用于分配容量）。

4.1 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。

参考的ramdisk.c代码：

```c
#include <minix/paths.h>

#include <sys/ioc_memory.h>
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>

int
main(int argc, char *argv[])
{
    int fd;
    signed long size;
    char *d;

    if(argc < 2 || argc > 3) {
        fprintf(stderr, "usage: %s <size in kB> [device]\n",
            argv[0]);
        return 1;
    }

    d = argc == 2 ? _PATH_RAMDISK : argv[2];
    if((fd=open(d, O_RDONLY)) < 0) {
        perror(d);
        return 1;
    }

#define KFACTOR 1024
    size = atol(argv[1])*KFACTOR;

    if(size < 0) {
        fprintf(stderr, "size should be non-negative.\n");
        return 1;
    }

    if(ioctl(fd, MIOCRAMSIZE, &size) < 0) {
        perror("MIOCRAMSIZE");
        return 1;
    }

    fprintf(stderr, "size on %s set to %ldkB\n", d, size/KFACTOR);

    return 0;
}
```

Buildmyram.c代码：

```c
#include <minix/paths.h>
#include <sys/ioc_memory.h>
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    int fd;
    signed long size;
    char *d;
    if(argc < 2 || argc > 3) {
        fprintf(stderr, "usage: %s <size in MB> [device]\n",
            argv[0]);
        return 1;
    }
    d = argc == 2 ? _PATH_RAMDISK : argv[2];
    if((fd=open(d, O_RDONLY)) < 0) {
        perror(d);
        return 1;
    }
// 需要把宏从1024改为1024*1024
#define MFACTOR 1048576
    size = atol(argv[1])*MFACTOR;
    if(size < 0) {
        fprintf(stderr, "size should be non-negative.\n");
        return 1;
    }
    if(ioctl(fd, MIOCRAMSIZE, &size) < 0) {
        perror("MIOCRAMSIZE");
        return 1;
    }
    fprintf(stderr, "size on %s set to %ldMB\n", d, size/MFACTOR);
    return 0;
}
```

在同一个目录下的Makefile文件中添加相应条目

```
PROG= ramdisk
PROG= buildmyram
MAN=

.include <bsd.prog.mk>
```

重新编译内核，重启虚拟机

```
# ls
bin         boot_monitor home        proc        service     usr
boot        dev          lib         root        sys         var
boot.cfg    etc          mnt         sbin        tmp
# cd /usr/src
# make build
```

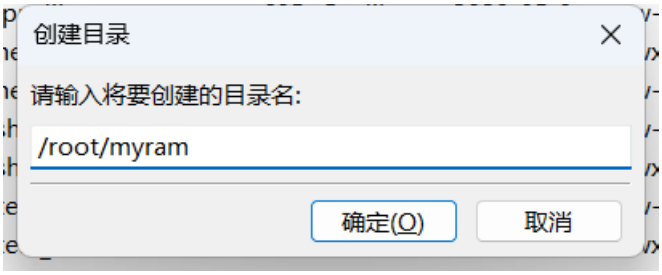
4.2 编译buildmyram.c 文件，然后执行命令： buildmyram <size in MB> /dev/myram。创建一个RAM 盘。

```
# cd /usr/src
# buildmyram 500 /dev/myram
size on /dev/myram set to 500MB
# _
```

5、 在ram 盘上创建内存文件系统，mkfs.mfs /dev/myram

```
# buildmyram 500 /dev/myram
size on /dev/myram set to 500MB
# mkfs.mfs /dev/myram
```

发现提示“No such file or directory”,说明没有创建该目录，则需要创建/root/myram文件目录



问题解决，将ram 盘挂载到用户目录下，mount /dev/myram /root/myram

```
# mkfs.mfs /dev/myram
# mount /dev/myram /root/myram
mount: Can't mount /dev/myram on /root/myram: No such file or directory
# mount /dev/myram /root/myram
/dev/myram is mounted on /root/myram
# _
```

查看是否挂载成功：输入df显示磁盘的文件系统与使用情形

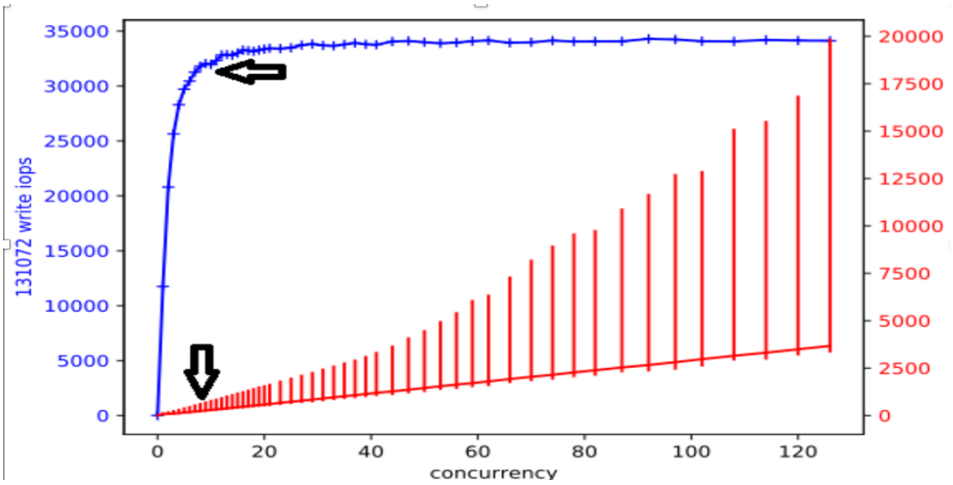
```
# df
Filesystem    512-blocks      Used      Avail %Cap Mounted on
/dev/myram       1024000     16088    1007912   1% /root/myram
/dev/c0d3p0s0     262144     76576     185568  29% /
none                   0         0          0 100% /proc
/dev/c0d3p0s2   33566464   4568120   28998344  13% /usr
/dev/c0d3p0s1    8114176     84968    8029208   1% /home
none                   0         0          0 100% /sys
# _
```

注意：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。

```
# cd root
# ls
.exrc         hello         myram         shell.o       test_code.o   your
.profile      hello.c       shell.c       test_code.c   text.txt
# buildmyram 500 /dev/myram
size on /dev/myram set to 500MB
# mkfs.mfs /dev/myram
# mount /dev/myram /root/myram
/dev/myram is mounted on /root/myram
# df
Filesystem    512-blocks      Used      Avail %Cap Mounted on
/dev/myram       1024000     16088    1007912   1% /root/myram
/dev/c0d3p0s0     262144     76576     185568  29% /
none                   0         0          0 100% /proc
/dev/c0d3p0s2   33566464   4568128   28998336  13% /usr
/dev/c0d3p0s1    8114176     84968    8029208   1% /home
none                   0         0          0 100% /sys
```

二、 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。通常情况下，7-15 个进程达到饱和



计算公式：总吞吐量=总文件大小/执行时间

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）。可以画四张曲线图对比RAM 盘和Disk 盘性能（随机读，随机写，顺序读，顺序写）。实验结果预计为RAM 盘性能高于DISK 盘，特别是随机读写性能。

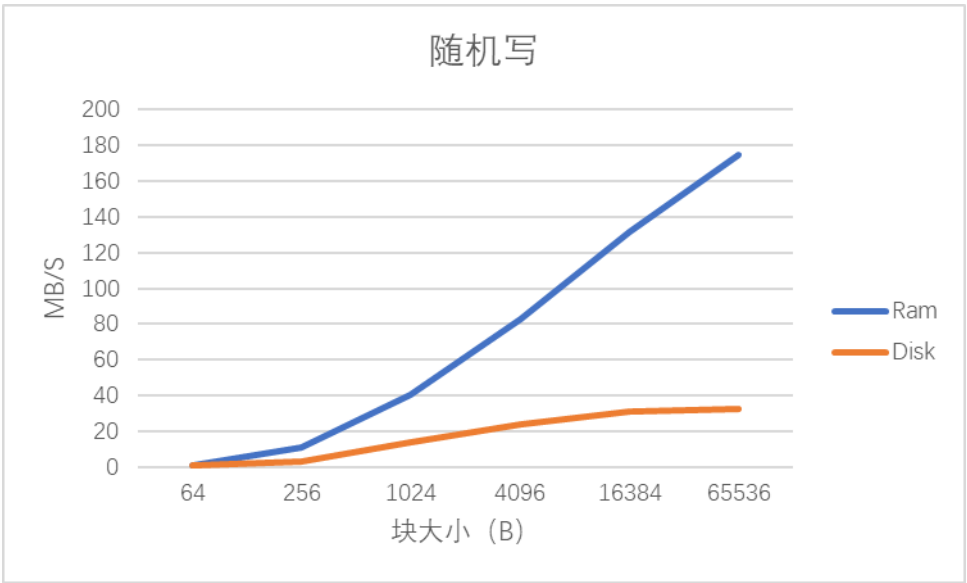
随机写：

Ram:

```
# ./mytest.o
blocksize_KB=0.0625KB=64B,speed=1.294685MB/s
blocksize_KB=0.2500KB=256B,speed=10.681152MB/s
blocksize_KB=1.0000KB=1024B,speed=40.211397MB/s
blocksize_KB=4.0000KB=4096B,speed=82.859848MB/s
blocksize_KB=16.0000KB=16384B,speed=131.777108MB/s
blocksize_KB=64.0000KB=65536B,speed=175.000000MB/s
```

Disk:

```
# ./mytest.o
blocksize_KB=0.0625KB=64B,speed=0.647343MB/s
blocksize_KB=0.2500KB=256B,speed=3.417969MB/s
blocksize_KB=1.0000KB=1024B,speed=13.671875MB/s
blocksize_KB=4.0000KB=4096B,speed=23.572198MB/s
blocksize_KB=16.0000KB=16384B,speed=31.250000MB/s
blocksize_KB=64.0000KB=65536B,speed=32.796102MB/s
#
```



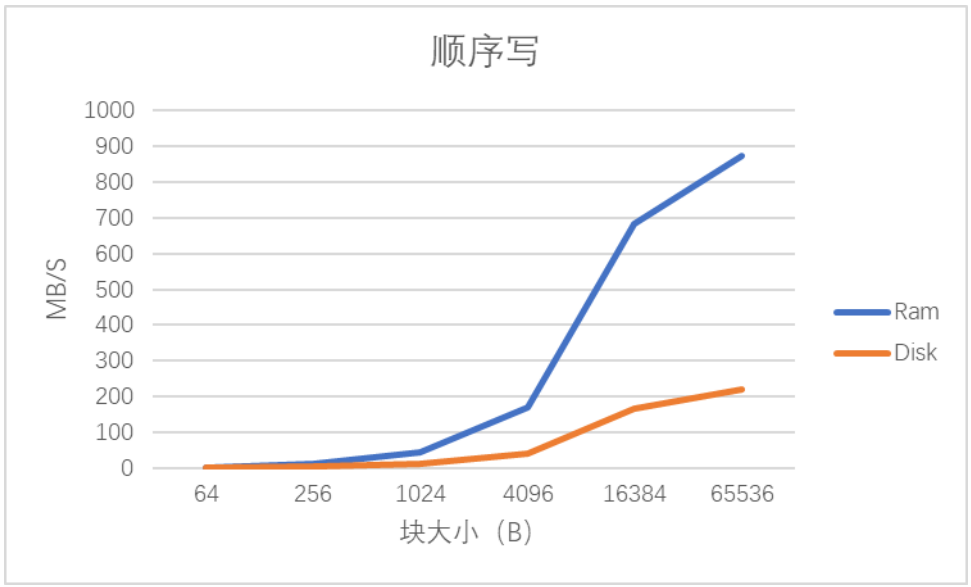
顺序写：

Ram:

```
# ./mytest.o
blocksize_KB=0.0625KB=64B,speed=2.670288MB/s
blocksize_KB=0.2500KB=256B,speed=10.681152MB/s
blocksize_KB=1.0000KB=1024B,speed=42.724609MB/s
blocksize_KB=4.0000KB=4096B,speed=170.898438MB/s
blocksize_KB=16.0000KB=16384B,speed=683.593750MB/s
blocksize_KB=64.0000KB=65536B,speed=875.000000MB/s
```

Disk:

```
# ./mytest.o
blocksize_KB=0.0625KB=64B,speed=1.294685MB/s
blocksize_KB=0.2500KB=256B,speed=5.178741MB/s
blocksize_KB=1.0000KB=1024B,speed=13.671875MB/s
blocksize_KB=4.0000KB=4096B,speed=41.429924MB/s
blocksize_KB=16.0000KB=16384B,speed=165.719697MB/s
blocksize_KB=64.0000KB=65536B,speed=218.750000MB/s
```



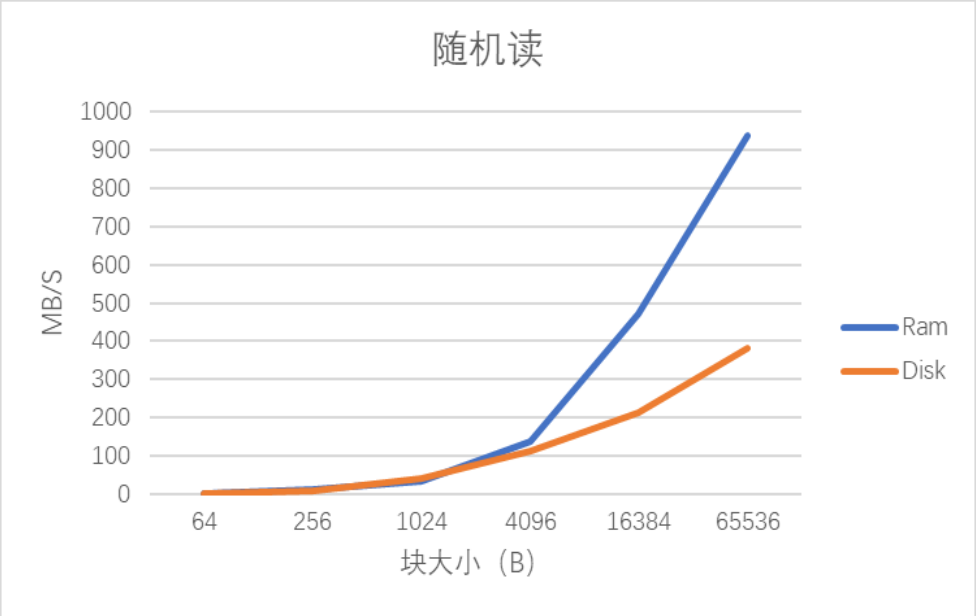
随机读：

Ram:

```
# ./mytest.o
blocksize_KB=0.0625KB=64B,speed=2.543132MB/s
blocksize_KB=0.2500KB=256B,speed=10.295087MB/s
blocksize_KB=1.0000KB=1024B,speed=34.179688MB/s
blocksize_KB=4.0000KB=4096B,speed=136.718750MB/s
blocksize_KB=16.0000KB=16384B,speed=471.443966MB/s
blocksize_KB=64.0000KB=65536B,speed=938.841202MB/s
```

Disk:

```
# ./mytest.o
blocksize_KB=0.0625KB=64B,speed=1.485728MB/s
blocksize_KB=0.2500KB=256B,speed=9.485628MB/s
blocksize_KB=1.0000KB=1024B,speed=40.475283MB/s
blocksize_KB=4.0000KB=4096B,speed=113.348724MB/s
blocksize_KB=16.0000KB=16384B,speed=214.387528MB/s
blocksize_KB=64.0000KB=65536B,speed=382.384838MB/s
```



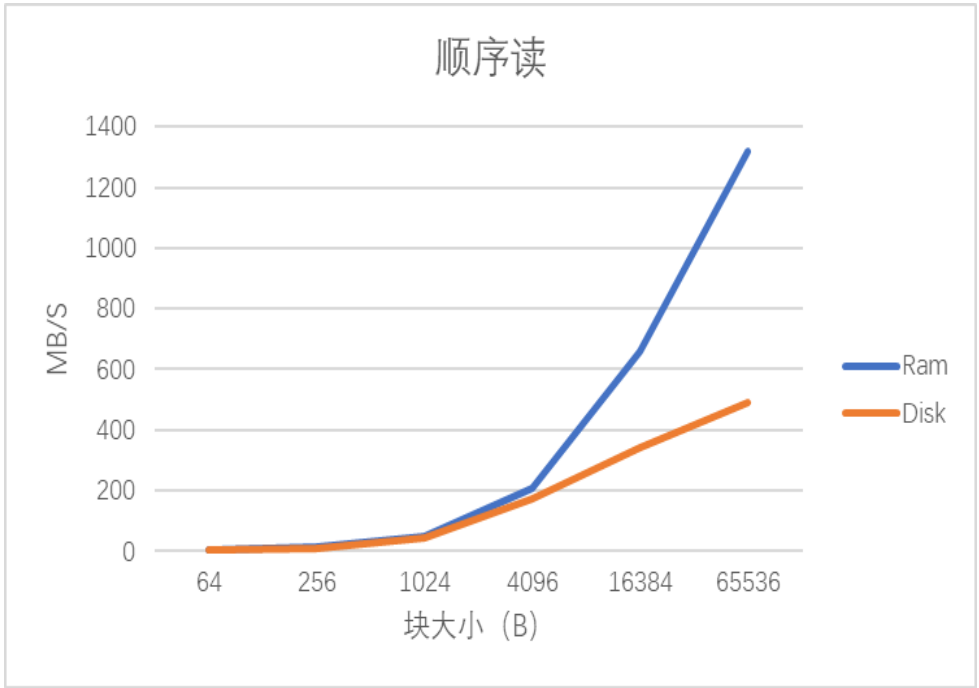
顺序读：

Ram:

```
# ./mytest.o
blocksize_KB=0.0625KB=64B,speed=4.272461MB/s
blocksize_KB=0.2500KB=256B,speed=12.946851MB/s
blocksize_KB=1.0000KB=1024B,speed=51.787405MB/s
blocksize_KB=4.0000KB=4096B,speed=207.149621MB/s
blocksize_KB=16.0000KB=16384B,speed=658.885542MB/s
blocksize_KB=64.0000KB=65536B,speed=1317.771084MB/s
#
```

Disk:

```
# ./mytest.o
blocksize_KB=0.0625KB=64B,speed=2.735621MB/s
blocksize_KB=0.2500KB=256B,speed=10.372834MB/s
blocksize_KB=1.0000KB=1024B,speed=43.237437MB/s
blocksize_KB=4.0000KB=4096B,speed=173.123769MB/s
blocksize_KB=16.0000KB=16384B,speed=342.648365MB/s
blocksize_KB=64.0000KB=65536B,speed=492.487285MB/s
#
```



三、 实验结果说明：

总体来看，写的速率比读的速率慢，随机读写的速率比顺序读写的速率慢，尤其是磁盘两种快扫描方式的速率差异更加明显。

顺序VS随机

究其原因在于磁盘读写时，机械寻道是影响磁盘读写速率的主要因素，随机读写每次都需要重新寻道，而顺序读写只需要在第一次进行寻道，因此两种方式的读写速率差别很大。

Disk VS RAM

对比Disk盘和RAM盘，RAM盘的读写速率普遍快于Disk盘，特别是随机读写性能。由于RAM盘为内存分配的一块区域，使用预先分配的主存来存储数据块，因此没有寻道和旋转延迟，具有快速存取的优点，而disk 盘有寻道和旋转延迟，所以RAM的读写速率优于Disk。但对Disk进行一次读后，其内容会被缓存，之后再次读Disk上的数据，其效率有所提高。

测试代码：

```c
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<sys/wait.h>
#include<fcntl.h>
#include<time.h>
#include<string.h>

#define TIMES 700//读写次数
//每组读写要反复持续一段时间 过短的时间会造成误差较大
#define maxline (1024*1024)
#define filesize (300*1024*1024)//文件总大小300MB
#define buffsize (1024*1024*1024)
char rbuff[buffsize];

char *filepathDisk[7]={"/usr/file1.txt","/usr/file2.txt","/usr/file3.txt","/usr/file4.txt","/usr/file5.txt","/usr/file6.txt","/usr/file7.txt"};
char *filepathRam[7]={"/root/myram/file1.txt","/root/myram/file2.txt","/root/myram/file3.txt","/root/myram/file4.txt","/root/myram/file5.txt",
"/root/myram/file6.txt","/root/myram/file7.txt"};

char buff[maxline]="10215501435";
void write_file(int blocksize, bool israrand, char *filepath){
    int fd=open(filepath,O_RDWR|O_CREAT|O_SYNC,0755);
    if(fd<0){
        printf("Open file error!");
        //return;
    }
    int temp;//记录实际写入
    //多次重复写入计算时间
    for(int i=0;i<TIMES;i++){
        if((temp=write(fd,buff,blocksize))!=blocksize){
            printf("%d\n",temp);
            printf("Write file error!\n");
        }
        if(israrand)
                lseek(fd,rand() % filesize,SEEK_SET);//利用随机函数写到文件的任意一个位置
        //如果是随机
       }
       //如果读到末尾则从文件开头开始读。
    lseek(fd, 0, SEEK_SET);//重设文件指针
        //顺序读写时默认文件指针自由移动
}
```

```
void read_file(int blocksize, bool israrand, char *filepath){
    int fd=open(filepath,O_RDWR|O_CREAT|O_SYNC,0755);
    if(fd<0){
        printf("Open file error!");
        //return;
    }
    int temp;//记录实际写入
    //多次重复写入计算时间
    for(int i=0;i<TIMES;i++){
        if((temp=read(fd,rbuff,blocksize))!=blocksize){
            printf("%d\n",temp);
            printf("Read file error!\n");
        }
        if(israrand)
                lseek(fd,rand() % filesize,SEEK_SET);//利用随机函数写到文件的任意一个位置
        //如果是随机
       }
       //如果读到末尾则从文件开头开始读。
    lseek(fd, 0, SEEK_SET);//重设文件指针
        //顺序读写时默认文件指针自由移动
}

long get_time_left(struct timeval starttime,struct timeval endtime){
    long spendtime;
    spendtime=(long)(endtime.tv_sec-starttime.tv_sec)*1000+(endtime.tv_usec-starttime.tv_usec)/1000;
        //换算成秒
    //spendtime=spendtime/1000;
    return spendtime;
}

int main(){
    srand((unsigned)time(NULL));
    struct timeval starttime, endtime;
    double spendtime;
    for(int i=0;i<maxline;i+=16){
        strcat(buff,"abcdefghijklmnop");
    }
    int blocksize=64*4*4*4;
    for(int blocksize=64;blocksize<=1024*64;blocksize=blocksize*4){
        //for(int Concurrency=1;Concurrency<=15;Concurrency=Concurrency+1){
            int Concurrency=7;
            gettimeofday(&starttime, NULL);
            for(int i=0;i<Concurrency;i++){
                if(fork()==0){
                //随机写
                //write_file(blocksize,true,filepathDisk[i]);
                //write_file(blocksize,true,filepathRam[i]);

                //顺序写
                //write_file(blocksize,false,filepathDisk[i]);
                //write_file(blocksize,false,filepathRam[i]);

                //随机读
                //read_file(blocksize,true,filepathDisk[i]);
                //read_file(blocksize,true,filepathRam[i]);

                //顺序读
                //read_file(blocksize,false,filepathDisk[i]);
                //read_file(blocksize,false,filepathRam[i]);
                exit(0);
                }
            }
            //等待所有子进程结束
            while(wait(NULL)!=-1);
            gettimeofday(&endtime, NULL);
            spendtime=get_time_left(starttime,endtime)/1000.0;
            double eachtime=spendtime/TIMES;
            double block=blocksize*Concurrency/1024.0/1024.0;
            printf("blocksize_KB=%.4fKB=%dB,speed=%fMB/s\n",(double)blocksize/1024.0,blocksize,block/eachtime);

            //printf("Concurrency=%d,speed=%fMB/s\n",Concurrency,block/eachtime);
        //}
    }
    return 0;
}
```

四、 函数说明：

定义函数：size_t write (int fd, const void * buf, size_t count);
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。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。
Read()函数与之类似。
定义函数：off_t lseek(int fildes,off_t offset,int whence);
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。

```
struct timeval
{
time_t tv_sec; /* Seconds. */
suseconds_t tv_usec; /* Microseconds. */
};
```

定义在#include <time.h>中，有两个成员，一个是秒，一个是微秒。

六、 总结

通过本次实验，我熟悉了类UNIX系统的I/O设备管理、MINIX块设备驱动以及MINIX RAM盘的相关知识。动手编写代码对比了在顺序读写和随机读写四种情况下，RAM盘和DISK盘的速率差异