10215501422高宇菲 课程设计三

实验目的

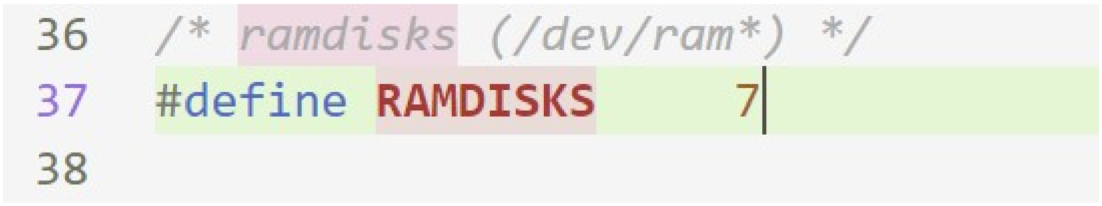
1. 熟悉类UNIX系统的I/O设备管理
2. 熟悉MINIX块设备驱动
3. 熟悉MINIX RAM盘

实验要求

- 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。

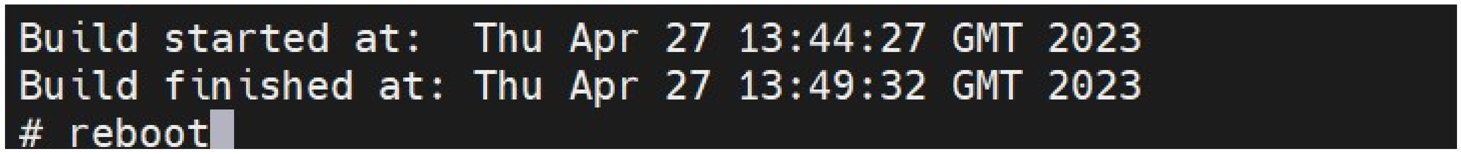
增加RAM盘

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



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

```

重新编译内核，重启reboot。



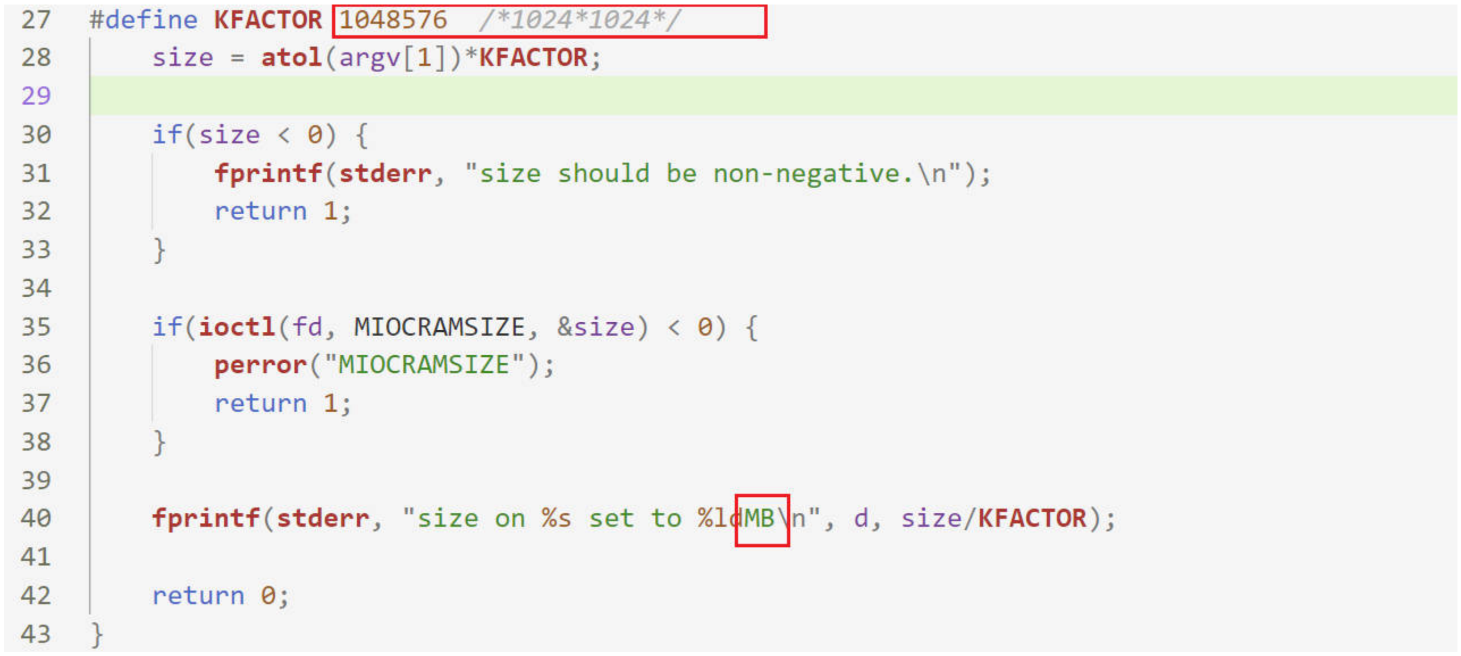
```
Build started at:  Thu Apr 27 13:44:27 GMT 2023
Build finished at: Thu Apr 27 13:49:32 GMT 2023
# reboot
```

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

```
# mknod /dev/myram b 1 13
# ls /dev/ | grep ram
myram
ram
ram0
ram1
ram2
ram3
ram4
ram5
```

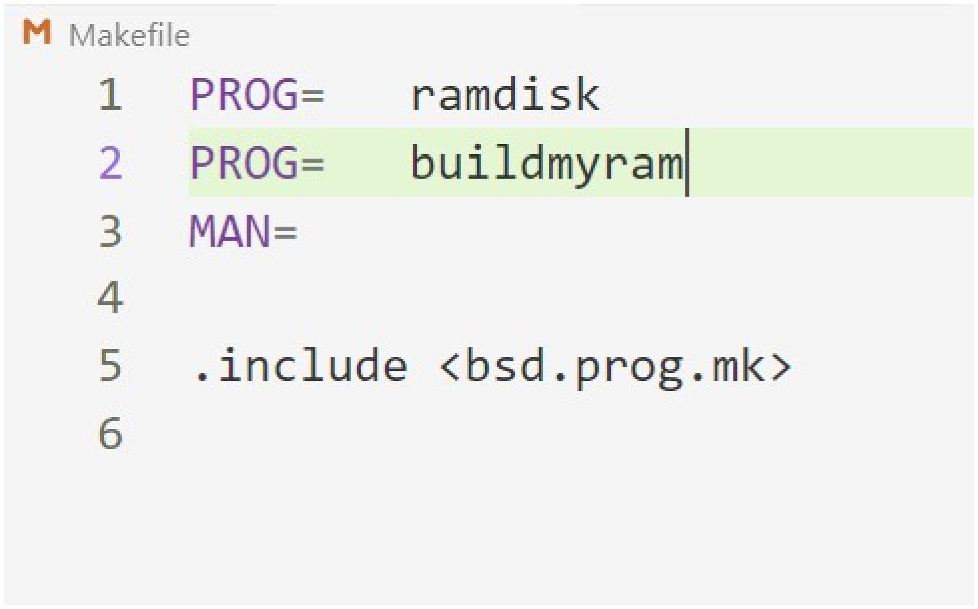
说明创建成功。

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。
修改两个位置：



```
#define KFACTOR 1048576  /*1024*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 %ldMB\n", d, size/KFACTOR);

    return 0;
}
```

要想把添加的文件编译进内核，就要将文件添加到MakeFile。仿照同目录下的文件格式添加：



```
PROG=    ramdisk
PROG=    buildmyram
MAN=

.include <bsd.prog.mk>

```

编译buildmyram.c文件，然后执行命令： buildmyram /dev/myram。创建一个RAM盘。这里设置为600MB。

```
Build started at:  Thu Apr 27 14:11:53 GMT 2023
Build finished at: Thu Apr 27 14:15:09 GMT 2023
# buildmyram 600 /dev/myram
size on /dev/myram set to 600MB
```

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
```

查看是否挂载成功：输入df

```
# mkfs.mfs /dev/myram
# mount /dev/myram /root/myram
mount: Can't mount /dev/myram on /root/myram: No such file or directory
# cd ../..
# cd root
# mkdir myram
# ls
.exrc                    myram                    test2t2
.lesshst                 test                     test_code.c
.profile                 test2                    test_code_nowversion.c
# cd ../usr/src
# mount /dev/myram /root/myram
/dev/myram is mounted on /root/myram
# df
Filesystem    512-blocks       Used      Avail %Cap Mounted on
/dev/myram        409600       6456     403144   1% /root/myram
/dev/c0d3p0s0     262144      76504     185640  29% /
none                   0          0          0 100% /proc
/dev/c0d3p0s2   33566464    4567672   28998792  13% /usr
/dev/c0d3p0s1    8114176      84976    8029200   1% /home
none                   0          0          0 100% /sys
```

编写性能测试代码

open函数：

- 头文件： #include <sys/types.h>
  #include <fcntl.h>
- 原型：int open(const char *path, int flags [, mode_t mode]);
- 打开一个文件（path）以读或写（由flags指定）并返回文件描述符。如果flags=O_CREAT,表示如果文件不存在则以mode（八进制权限码）方式创建。
- flags可取：
  O_RDONLY open for reading only
  O_WRONLY open for writing only
  O_RDWR open for reading and writing
  O_NONBLOCK do not block on open
  O_APPEND append on each write
  O_CREAT create file if it does not exist
  O_TRUNC truncate size to 0
  O_EXCL error if create and file exists

write函数：

- 头文件 <unistd.h>
- ssize_t write(int fd, const void *buf, size_t nbytes);
- 返回值：成功则返回已写的字节数（nbytes）；出错返回-1.

read函数：

- 头文件 <unistd.h>
- ssize_t read(int fd, const void *buf, size_t nbytes);
- 返回值：读到的字节数（<=nbytes）；若已读到结尾返回0；出错返回-1.

lseek函数:

- 头文件 <unistd.h>
- off_t currpos = lseek(int fd, off_t offset ,int whence);
- 修改偏移量, 影响下一次读写的操作
- 返回值：成功返回新偏移量；出错返回-1.
- whence=SEEK_SET, 偏移量设为距文件开始offset字节。
- whence=SEEK_CUR, 偏移量设为当前值加offset字节。
- whence=SEEK_CUR, 偏移量设为文件长度加offset字节。

测试代码：

```c
typedef int bool;

#define false 0
#define true 1
#define maxprocessN 15  /*最大进程数*/
#define maxblocksize  (64 * 1024) /*bytes*/

int filesize = (300*1024*1024);
struct timeval starttime, endtime;
char *Diskpath[maxprocessN] = {"/usr/test/file1.txt", "/usr/test/file2.txt", "/usr/test/file3.tx
char *RAMpath[maxprocessN] = {"/root/myram/test/file1.txt", "/root/myram/test/file2.txt", "/root
char writebuf[maxblocksize];

/*写文件:打开文件，判断返回值，如果正常打开文件就判断是否随机写，进行写操作*/
void write_file(int blocksize, bool isrand, char *filepath, int fs)
{
    int times = fs/blocksize;  /*计算出要写多少次*/
    // printf("times=%d\n", times);
    int fp = open(filepath, O_WRONLY|O_CREAT|O_SYNC, 755);
    if(fp<0){
        printf("open testfile failed.\n");
        return ;
    }
    // printf("open %s success!\n", filepath);
    lseek(fp, 0, SEEK_SET);
    for(int i=0; i<times; i++){
        int count = write(fp, writebuf, blocksize);
        if(count<0){
            printf("write error.\n");
            return;
        }
        if(isrand){
            lseek(fp, rand()%(fs-blocksize), SEEK_SET);  /*一开始是对fs取余，但还是有可能写出文件*/
        }
    }
}
/*读文件:打开文件，判断返回值，如果正常打开就判断是否随机读，进行读操作*/
void read_file(int blocksize, bool isrand, char *filepath, int fs)
{
    char buf[maxblocksize];
    int times = fs / blocksize; /*计算出要读多少次*/
    int fp = open(filepath, O_RDONLY|O_SYNC, 755);
    if (fp < 0){
        printf("open testfile failed.\n");
        return;
    }
    lseek(fp, 0, SEEK_SET);
    for (int i = 0; i < times; i++){
        int count = read(fp, buf, blocksize);
        if (count < 0){
            printf("read error.\n");
            return;
        }
        if (isrand){
            lseek(fp, rand() % (fs - blocksize), SEEK_SET); /*一开始是对filesize取余，但还是有可能
        }
    }
}

// 计算时间差，在读或写操作前后分别取系统时间，然后计算差值即为时间差，
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.
    return spendtime;
}

/*主函数：首先创建和命名文件，通过循环执行read file和write file函数测试读写差异，
  测试blocksize和concurrency对测试读写速度的影响，最后输出结果。*/
int main()
{
    for(int i=0; i<(maxblocksize/16); i++){
        strcat(writebuf, "abcd1abcd2abcd3a");
    }
    int concurrency = 7;   /*设置并发进程数*/
    srand((unsigned)time(NULL));
    for(int blocksize=64; blocksize<=(64*1024); blocksize*=4){
        printf("============== blocksize=%d =======================\n", blocksize);
        gettimeofday(&starttime, NULL);
        // printf("get starttime=%d\n", starttime.tv_usec);
        for (int i = 0; i < concurrency; i++){
            if(fork()==0){
                // 随机写
                // write_file(blocksize, true, RAMpath[i], filesize/concurrency);
                // write_file(blocksize, true, Diskpath[i], filesize/concurrency);

                // 顺序写
                write_file(blocksize, false, RAMpath[i], filesize / concurrency); /*filesize/cor
                // write_file(blocksize, false, Diskpath[i], filesize/concurrency);

                // 随机读
                // read_file(blocksize, true, RAMpath[i], filesize / concurrency);
                // read_file(blocksize, true, Diskpath[i], filesize / concurrency);

                // 顺序读
                // read_file(blocksize, false, RAMpath[i], filesize / concurrency);
                // read_file(blocksize, false, Diskpath[i], filesize/concurrency);

                exit(0);
            }
        }
        while ((wait(NULL)) >= 0); /*等待所有子进程结束*/
        gettimeofday(&endtime, NULL);
        // printf("get endtime=%d\n", endtime.tv_usec);
        long spendtime = get_time_left(starttime, endtime);
        printf("blocksize=%d B, concurrency=%d, speed=%f B/s \n", blocksize, concurrency, (doubl
    }
    return 0;
}
```

遇到的问题：

1. 一开始测试代码运行会卡住，发现在minix上会输出空间不足的提示，然后才改成600MB。
2. 对于RAMpath和Diskpath，一开始想用一个init_path函数来初始化，但是因为sprintf函数不是很好用，失败了，只好改为手动初始化。
3. 遇到一开始写成功，blocksize变大时失败：

```
# ./test
process0 is writing ...
process1 is writing ...
702171
process2 is writing ...
702171
process3 is writing ...
open /usr/test/file1.txt success!
702171
process4 is writing ...
702171
process5 is writing ...
702171
process6 is writing ...
702171
702171
open /usr/test/file2.txt success!
open /usr/test/file3.txt success!
open /usr/test/file4.txt success!
open /usr/test/file5.txt success!
open /usr/test/file6.txt success!
open /usr/test/file7.txt success!
blocksize=64, concurrency=7, spendtime=266666 us
process0 is writing ...
process1 is writing ...
175542
process2 is writing ...
175542
process3 is writing ...
175542
process4 is writing ...
175542
175542
process5 is writing ...
process6 is writing ...
175542
175542
open /usr/test/file1.txt success!
open /usr/test/file2.txt success!
open /usr/test/file3.txt success!
open /usr/test/file4.txt success!
open /usr/test/file5.txt success!
open /usr/test/file6.txt success!
open /usr/test/file7.txt success!
blocksize=256, concurrency=7, spendtime=-50000 us
process0 is writing ...
process1 is writing ...
43885
process2 is writing ...
43885
```

```
process3 is writing ...
43885
process4 is writing ...
43885
process5 is writing ...
43885
process6 is writing ...
43885
43885
open /usr/test/file1.txt success!
open /usr/test/file2.txt success!
open /usr/test/file3.txt success!
open /usr/test/file4.txt success!
open /usr/test/file5.txt success!
open /usr/test/file6.txt success!
open /usr/test/file7.txt success!
blocksize=1024, concurrency=7, spendtime=583334 us
process0 is writing ...
process1 is writing ...
10971
process2 is writing ...
10971
process3 is writing ...
10971
process4 is writing ...
10971
process5 is writing ...
10971
process6 is writing ...
10971
10971
open /usr/test/file1.txt success!
open /usr/test/file2.txt success!
write error.
open /usr/test/file3.txt success!
write error.
open /usr/test/file4.txt success!
write error.
open /usr/test/file5.txt success!
write error.
open /usr/test/file6.txt success!
write error.
open /usr/test/file7.txt success!
write error.
write error.
blocksize=4096, concurrency=7, spendtime=0 us
process0 is writing ...
process1 is writing ...
2742
process2 is writing ...
2742
```

```
process3 is writing ...
2742
process4 is writing ...
2742
process5 is writing ...
2742
process6 is writing ...
2742
2742
open /usr/test/file1.txt success!
open /usr/test/file2.txt success!
write error.
open /usr/test/file3.txt success!
write error.
open /usr/test/file4.txt success!
write error.
open /usr/test/file5.txt success!
write error.
open /usr/test/file6.txt success!
write error.
open /usr/test/file7.txt success!
write error.
write error.
blocksize=16384, concurrency=7, spendtime=0 us
process0 is writing ...
process1 is writing ...
685
process2 is writing ...
685
process3 is writing ...
685
process4 is writing ...
685
process5 is writing ...
685
process6 is writing ...
685
685
open /usr/test/file1.txt success!
open /usr/test/file2.txt success!
write error.
open /usr/test/file3.txt success!
write error.
open /usr/test/file4.txt success!
write error.
open /usr/test/file5.txt success!
write error.
open /usr/test/file6.txt success!
write error.
open /usr/test/file7.txt success!
write error.
write error.
blocksize=65536, concurrency=7, spendtime=-16667 us
```

发现写入的buf没有进行初始化，打开写入的文件，发现确实都是乱码。对writebuf进行初始化，使它包含maxblocksize个字符。再次测试，这次没有出现写错误了，但是spendtime有正有负。

```
# ./test
process0 is writing ...
process1 is writing ...
times=702171
process2 is writing ...
times=702171
process3 is writing ...
open /root/myram/test/file1.txt success!
times=702171
process4 is writing ...
open /root/myram/test/file2.txt success!
times=702171
process5 is writing ...
open /root/myram/test/file3.txt success!
times=702171
process6 is writing ...
open /root/myram/test/file4.txt success!
times=702171
open /root/myram/test/file5.txt success!
times=702171
open /root/myram/test/file6.txt success!
open /root/myram/test/file7.txt success!
blocksize=64, concurrency=7, spendtime=-250000 us
process0 is writing ...
process1 is writing ...
times=175542
process2 is writing ...
times=175542
process3 is writing ...
times=175542
process4 is writing ...
times=175542
process5 is writing ...
times=175542
process6 is writing ...
times=175542
times=175542
open /root/myram/test/file1.txt success!
open /root/myram/test/file2.txt success!
open /root/myram/test/file3.txt success!
open /root/myram/test/file4.txt success!
open /root/myram/test/file5.txt success!
open /root/myram/test/file6.txt success!
open /root/myram/test/file7.txt success!
blocksize=256, concurrency=7, spendtime=400000 us
process0 is writing ...
process1 is writing ...
times=43885
process2 is writing ...
times=43885
```

```
process3 is writing ...
times=43885
process4 is writing ...
times=43885
process5 is writing ...
times=43885
process6 is writing ...
times=43885
times=43885
open /root/myram/test/file1.txt success!
open /root/myram/test/file2.txt success!
open /root/myram/test/file3.txt success!
open /root/myram/test/file4.txt success!
open /root/myram/test/file5.txt success!
open /root/myram/test/file6.txt success!
open /root/myram/test/file7.txt success!
blocksize=1024, concurrency=7, spendtime=-866666 us
process0 is writing ...
process1 is writing ...
times=10971
process2 is writing ...
times=10971
process3 is writing ...
times=10971
process4 is writing ...
times=10971
process5 is writing ...
times=10971
process6 is writing ...
times=10971
times=10971
open /root/myram/test/file1.txt success!
open /root/myram/test/file2.txt success!
open /root/myram/test/file3.txt success!
open /root/myram/test/file4.txt success!
open /root/myram/test/file5.txt success!
open /root/myram/test/file6.txt success!
open /root/myram/test/file7.txt success!
blocksize=4096, concurrency=7, spendtime=666666 us
process0 is writing ...
process1 is writing ...
times=2742
process2 is writing ...
times=2742
process3 is writing ...
times=2742
process4 is writing ...
times=2742
process5 is writing ...
times=2742
process6 is writing ...
```

```
times=2742
times=2742
open /root/myram/test/file1.txt success!
open /root/myram/test/file2.txt success!
open /root/myram/test/file3.txt success!
open /root/myram/test/file4.txt success!
open /root/myram/test/file5.txt success!
open /root/myram/test/file6.txt success!
open /root/myram/test/file7.txt success!
blocksize=16384, concurrency=7, spendtime=166667 us
process0 is writing ...
process1 is writing ...
times=685
process2 is writing ...
times=685
process3 is writing ...
times=685
process4 is writing ...
times=685
process5 is writing ...
times=685
process6 is writing ...
times=685
times=685
open /root/myram/test/file1.txt success!
open /root/myram/test/file2.txt success!
open /root/myram/test/file3.txt success!
open /root/myram/test/file4.txt success!
open /root/myram/test/file5.txt success!
open /root/myram/test/file6.txt success!
open /root/myram/test/file7.txt success!
blocksize=65536, concurrency=7, spendtime=-683333 us
```

发现是自己对gettimeofday函数的理解有误。所花费时间应该是timeval结构体的秒和微秒相加。

修改之后输出没有问题了：

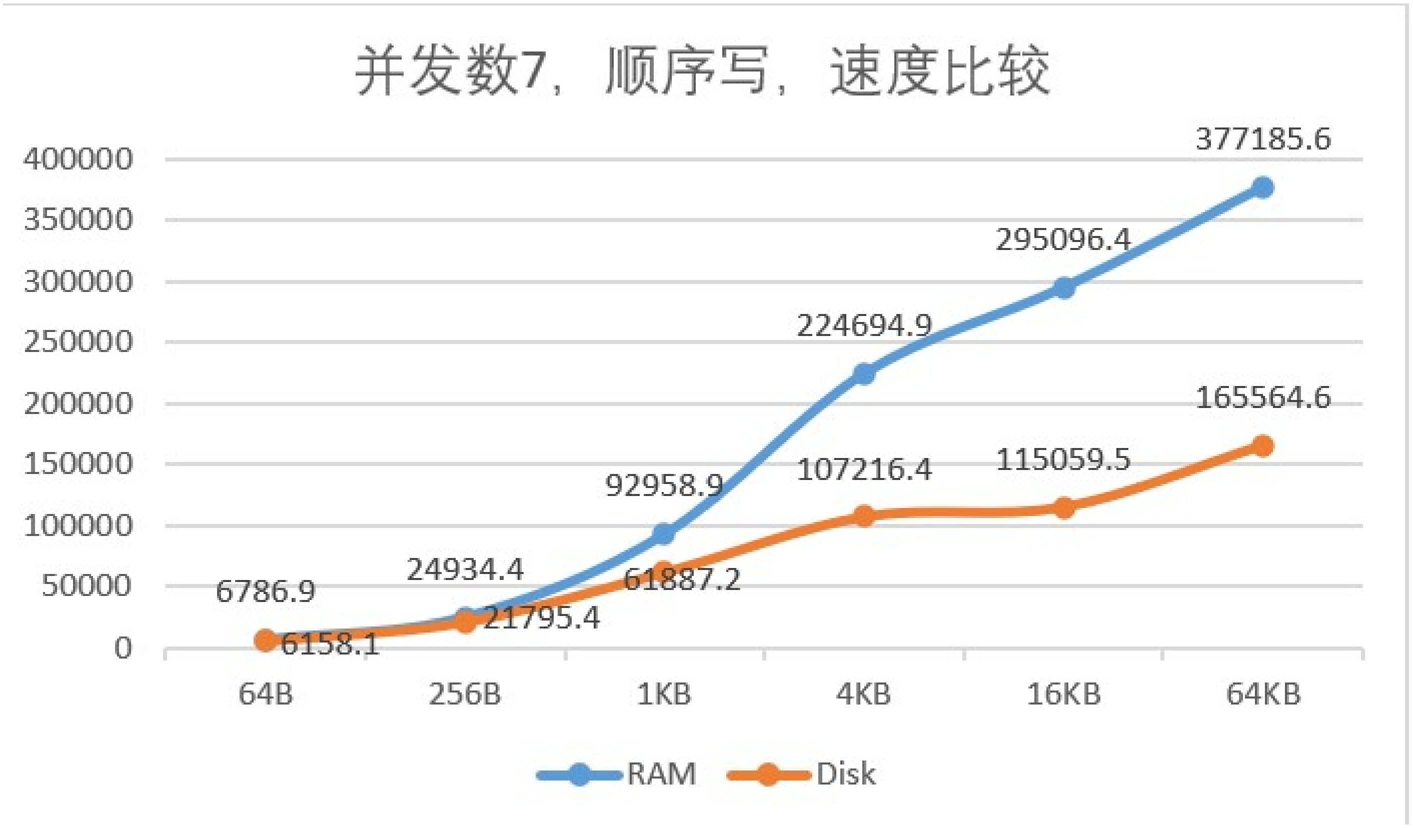
```
# ./test
# ./test
============== blocksize=64 =======================
process0 is writing ...
process1 is writing ...
process2 is writing ...
process3 is writing ...
process4 is writing ...
process5 is writing ...
process6 is writing ...
blocksize=64 B, concurrency=7, speed=6786.899676 B/s
============== blocksize=256 =======================
process0 is writing ...
process1 is writing ...
process2 is writing ...
process3 is writing ...
process4 is writing ...
process5 is writing ...
process6 is writing ...
blocksize=256 B, concurrency=7, speed=24934.432467 B/s
============== blocksize=1024 =======================
process0 is writing ...
process1 is writing ...
process2 is writing ...
process3 is writing ...
process4 is writing ...
process5 is writing ...
process6 is writing ...
blocksize=1024 B, concurrency=7, speed=92958.865248 B/s
============== blocksize=4096 =======================
process0 is writing ...
process1 is writing ...
process2 is writing ...
process3 is writing ...
process4 is writing ...
process5 is writing ...
process6 is writing ...
blocksize=4096 B, concurrency=7, speed=224694.857143 B/s
============== blocksize=16384 =======================
process0 is writing ...
process1 is writing ...
process2 is writing ...
process3 is writing ...
process4 is writing ...
process5 is writing ...
process6 is writing ...
blocksize=16384 B, concurrency=7, speed=295096.435272 B/s
============== blocksize=65536 =======================
process0 is writing ...
process1 is writing ...
```

```
process2 is writing ...
process3 is writing ...
process4 is writing ...
process5 is writing ...
process6 is writing ...
blocksize=65536 B, concurrency=7, speed=377185.611511 B/s
```

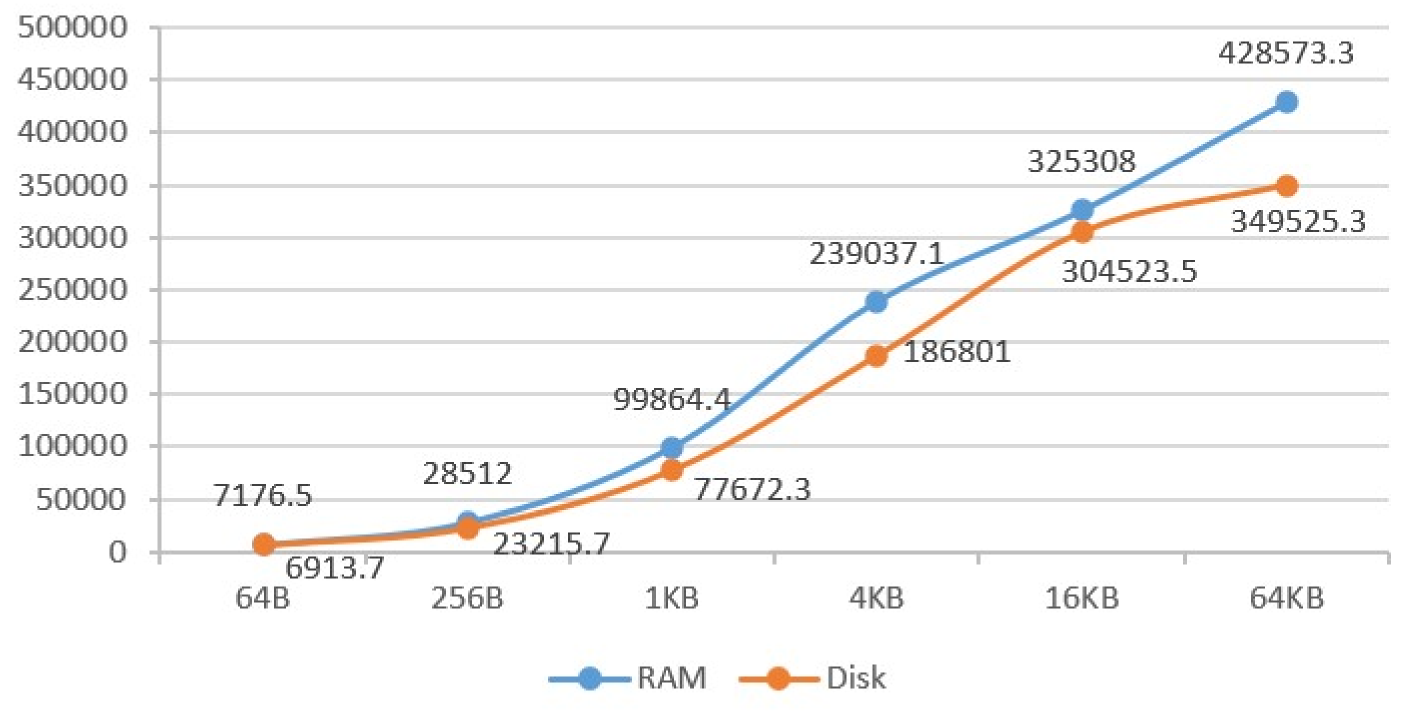
随机写时，写了很久都没有跑出结果，后来发现是顺序写的时候没有吧lseek归位。修改之后仍然跑了很久，但最后出了结果。

实验结果

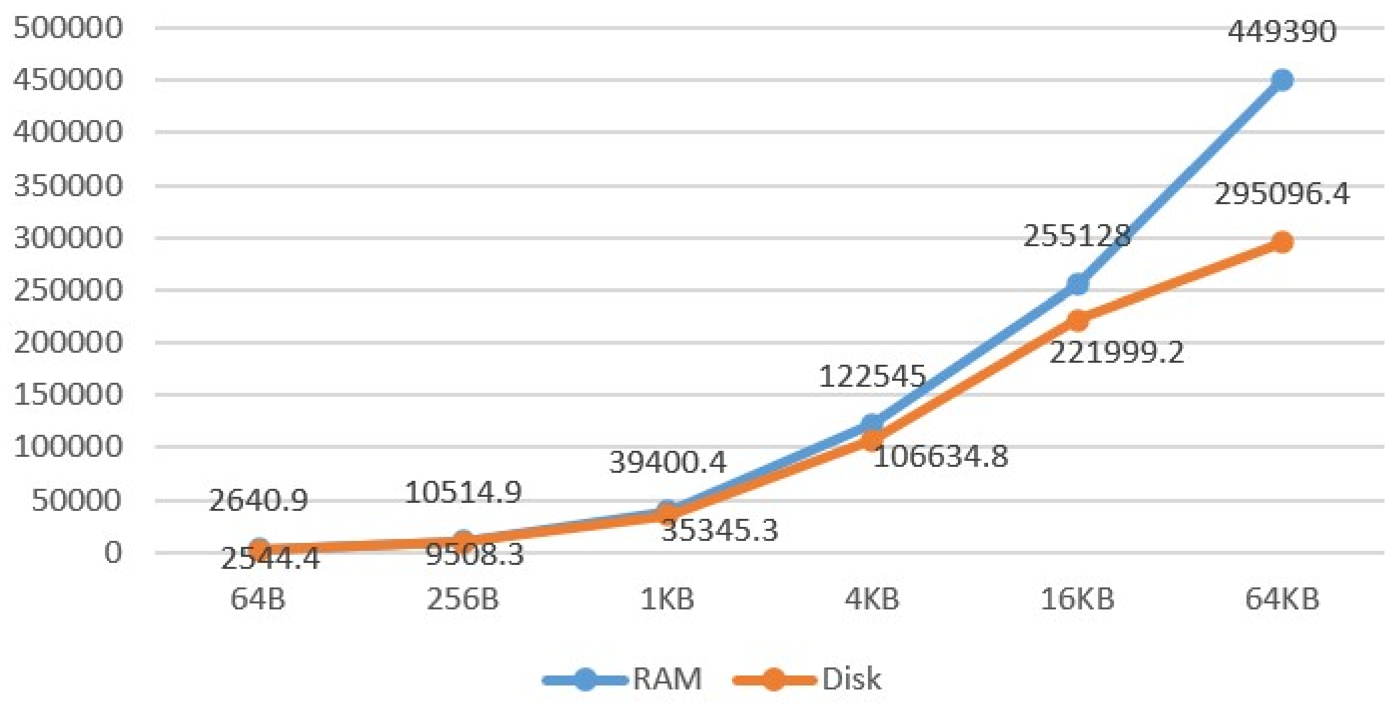
为了实验方便，先顺序写，再顺序读，再随机读，再随机写。

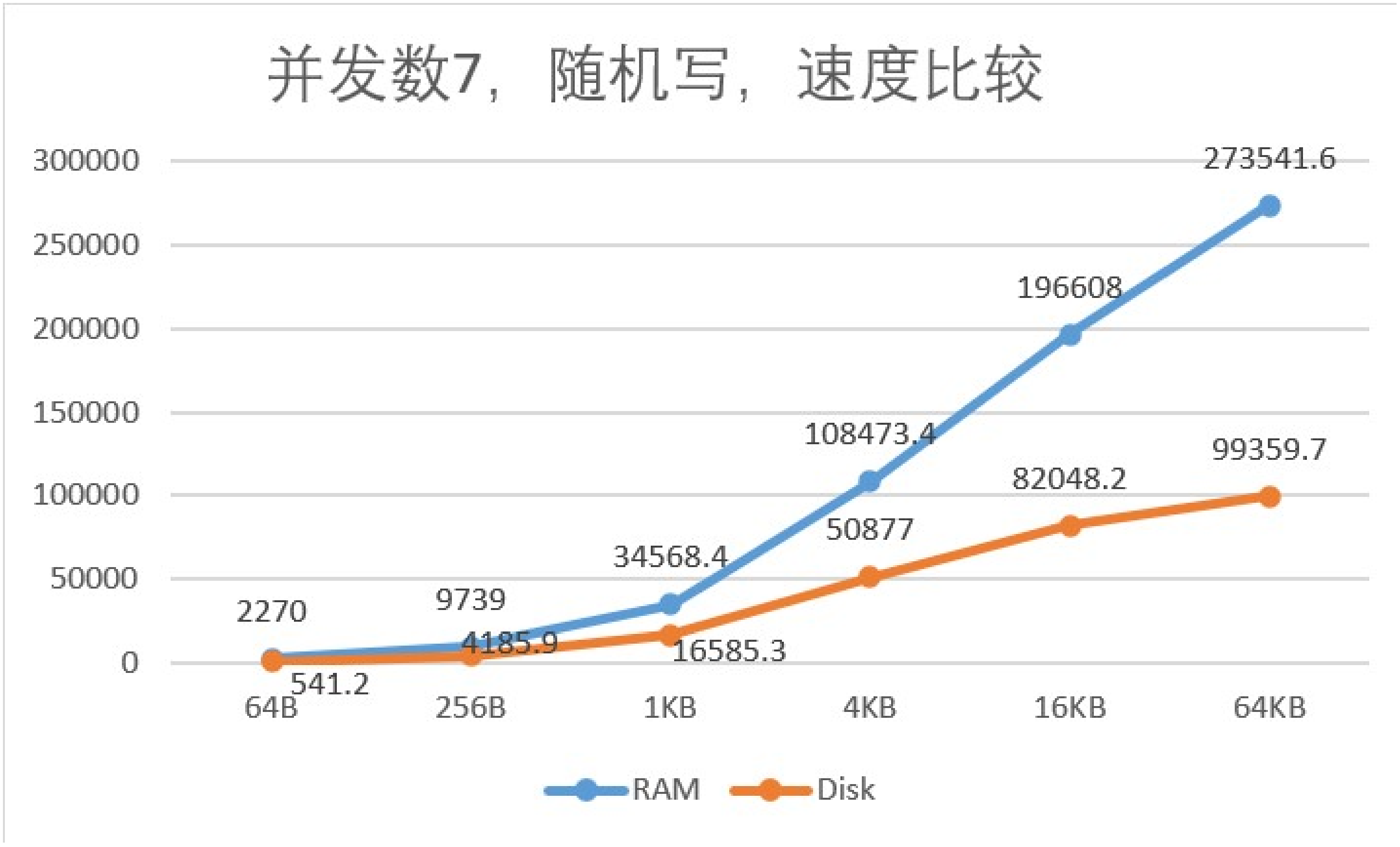


并发数7， 顺序读，速度比较



并发数7，随机读，速度比较





分析：
总体上，读比写快，顺序读写比随机读写快。原因可能是随机读写没有保证良好的局部性。
随着blocksize变大，RAM盘和disk盘的读写速度都有所上升，但是它们之间读写速度差很多，blocksize越大，RAM盘优势更明显。原因是RAM盘使用预先分配的主存来存储数据快，没有寻道和旋转延迟。

实验总结

本次实验比较了RAM盘和disk盘的读写速度差异。本次实验总体较为顺利，遇到的问题大多都予以解决。为今后学习打下基础。