# 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：**2020 级 | **上机实践成绩：** |
| **指导教师：**翁楚良 | **姓名：**张熙翔 | |
| **上机实践名称：**I/O subsystem | **学号：**10205501427 | **上机实践日期：**2022/5/15 |

## 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

## 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（可用图表形式体现在实验报告中）。

## 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。

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

2. 重新编译内核，重启 reboot。

```
make build
rebooot
```

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
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。

```
#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。

```
make build
rebooot
```

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。

```
# 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。

6. 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 显示磁盘的文件系统与使用情形。

```
# 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/c0d0p0s0     262144     76568     185576  29% /
none                   0         0          0 100% /proc
/dev/c0d0p0s2   33566464   4567352   28999112  13% /usr
/dev/c0d0p0s1    8114176     84968    8029208   1% /home
none                   0         0          0 100% /sys
#
```

（注：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。）

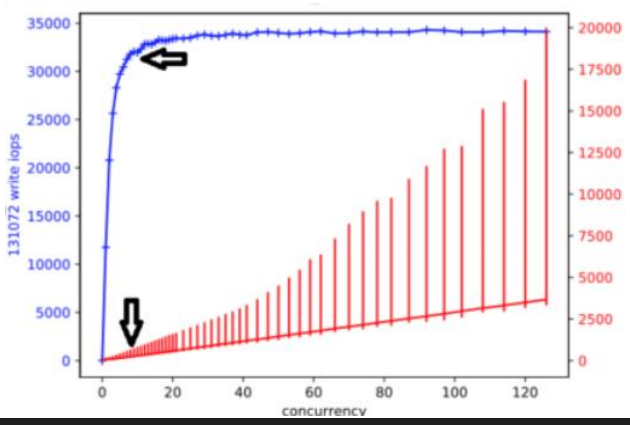
```
buildmyram 500 /dev/myram
// 吞吐量不会高于 700MB/s
mkfs.mfs /dev/myram
mount /dev/myram /root/myram
```

## 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。

计算公式：总吞吐量=总文件大小/执行时间

通常情况下，7-15 个进程达到饱和




```
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)blocks
ize/1024.0,blocksize,block/eachtime);
            printf("Concurrency=%d,speed=%fMB/s\n",Concurrency,block/eachtim
e);
        //}
    }
```

Disk 盘顺序写

```
blocksize=256,concurrency=1,speed=4.394531,iop
blocksize=256,concurrency=2,speed=8.789062,iop
blocksize=256,concurrency=3,speed=inf,iops=inf
blocksize=256,concurrency=4,speed=17.578126,io
blocksize=256,concurrency=5,speed=10.986328,io
blocksize=256,concurrency=6,speed=26.367189,io
blocksize=256,concurrency=7,speed=30.761718,io
blocksize=256,concurrency=8,speed=35.156249,io
blocksize=256,concurrency=9,speed=13.183594,io
blocksize=256,concurrency=10,speed=21.972656,i
blocksize=256,concurrency=11,speed=48.339843,i
blocksize=256,concurrency=12,speed=52.734377,i
blocksize=256,concurrency=13,speed=28.564453,i
blocksize=256,concurrency=14,speed=30.761719,i
blocksize=256,concurrency=15,speed=16.479492,i
blocksize=256,concurrency=16,speed=35.156250,i
blocksize=256,concurrency=17,speed=18.676758,i
blocksize=256,concurrency=18,speed=39.550782,i
blocksize=256,concurrency=19,speed=27.832031,i
blocksize=256,concurrency=20,speed=21.972656,i
```

*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。*

Ram 随机写

```
blocksize_KB=0.0625KB=64B,speed=0.854492MB/s
blocksize_KB=0.2500KB=256B,speed=12.849507MB/s
blocksize_KB=1.0000KB=1024B,speed=41.180346MB/s
blocksize_KB=4.0000KB=4096B,speed=48.310512MB/s
blocksize_KB=16.0000KB=16384B,speed=64.338235MB/s
blocksize_KB=64.0000KB=65536B,speed=57.565789MB/s
```

Disk 随机写

```
blocksize_KB=0.0625KB=64B,speed=0.502642MB/s
blocksize_KB=0.2500KB=256B,speed=6.835938MB/s
blocksize_KB=1.0000KB=1024B,speed=25.602762MB/s
blocksize_KB=4.0000KB=4096B,speed=36.458333MB/s
blocksize_KB=16.0000KB=16384B,speed=39.528370MB/s
blocksize_KB=64.0000KB=65536B,speed=40.137615MB/s
```

Ram 顺序写

```
blocksize_KB=0.0625KB=64B,speed=2.670288MB/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=82.859848MB/s
blocksize_KB=16.0000KB=16384B,speed=683.593750MB/s
blocksize_KB=64.0000KB=65536B,speed=875.000000MB/s
```

Disk 顺序写

```
blocksize_KB=0.0625KB=64B,speed=1.294685MB/s
blocksize_KB=0.2500KB=256B,speed=3.417969MB/s
blocksize_KB=1.0000KB=1024B,speed=20.714962MB/s
blocksize_KB=4.0000KB=4096B,speed=82.859848MB/s
blocksize_KB=16.0000KB=16384B,speed=218.750000MB/s
blocksize_KB=64.0000KB=65536B,speed=328.947368MB/s
```

Ram 随机读

```
blocksize_KB=0.0625KB=64B,speed=5.147543MB/s
blocksize_KB=0.2500KB=256B,speed=17.089844MB/s
blocksize_KB=1.0000KB=1024B,speed=45.572917MB/s
blocksize_KB=4.0000KB=4096B,speed=205.592105MB/s
blocksize_KB=16.0000KB=16384B,speed=729.166667MB/s
blocksize_KB=64.0000KB=65536B,speed=1309.880240MB/s
```

Disk 随机读

```
blocksize_KB=0.0625KB=64B,speed=1.294685MB/s
blocksize_KB=0.2500KB=256B,speed=3.417969MB/s
blocksize_KB=1.0000KB=1024B,speed=20.714962MB/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=263.554217MB/s
```

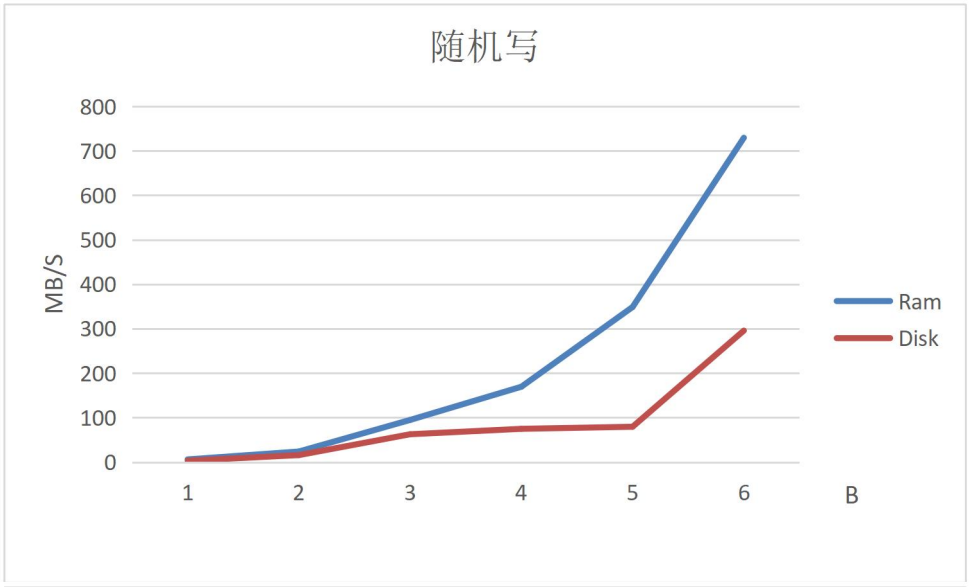
Ram 顺序读

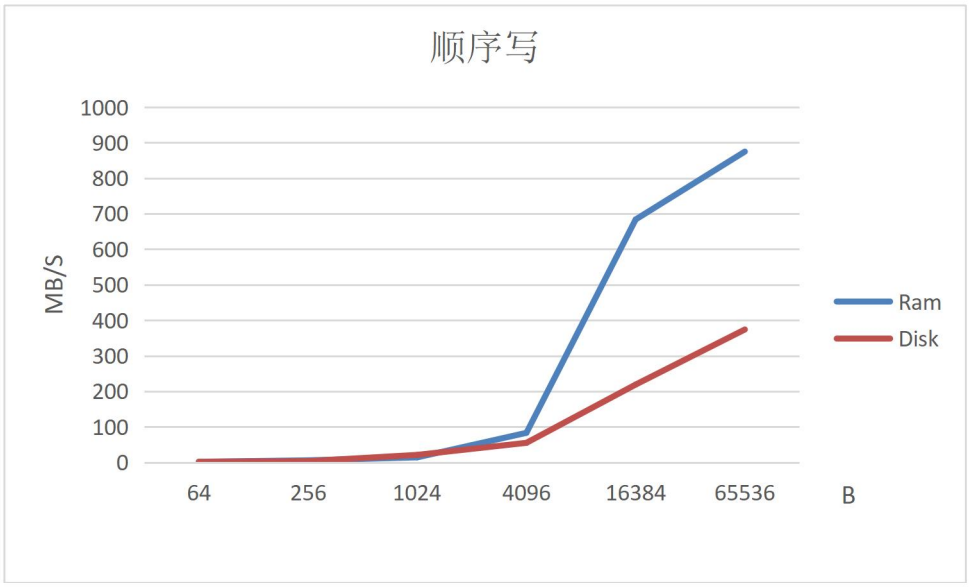
```
blocksize_KB=0.0625KB=64B,speed=0.854492MB/s
blocksize_KB=0.2500KB=256B,speed=10.681152MB/s
blocksize_KB=1.0000KB=1024B,speed=13.671875MB/s
blocksize_KB=4.0000KB=4096B,speed=170.898438MB/s
blocksize_KB=16.0000KB=16384B,speed=331.439394MB/s
blocksize_KB=64.0000KB=65536B,speed=875.000000MB/s
```

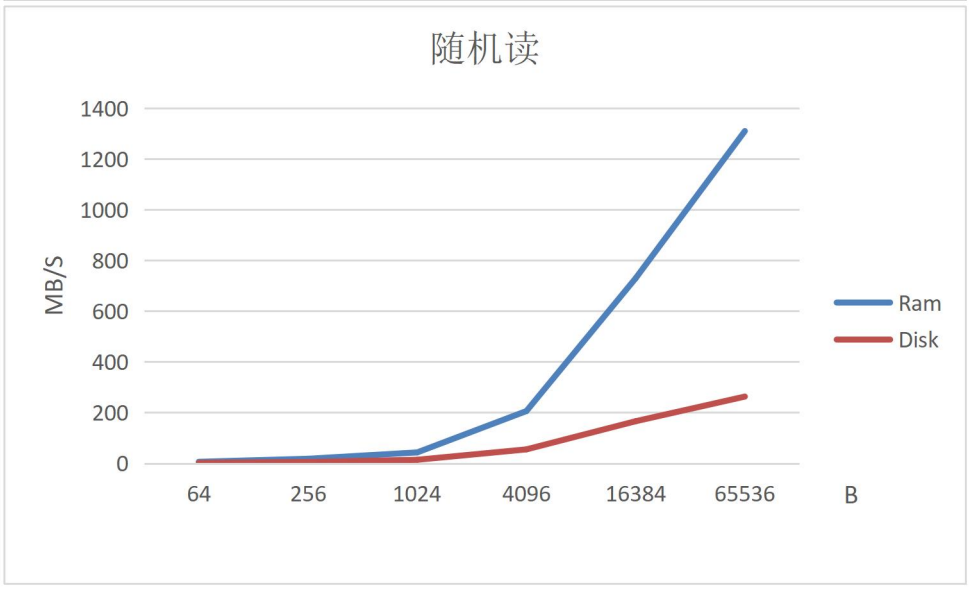
Disk 顺序读

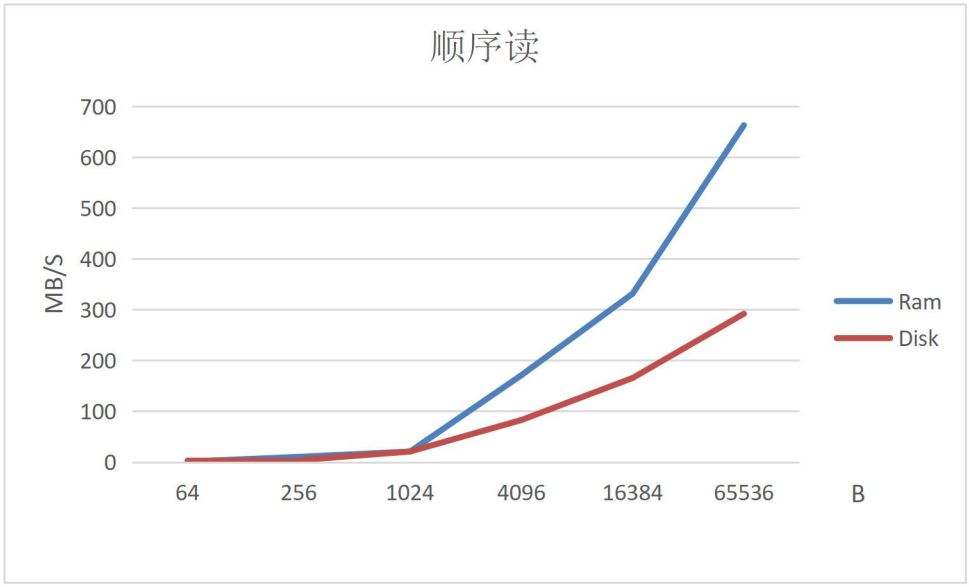
```
blocksize_KB=0.0625KB=64B,speed=2.670288MB/s
blocksize_KB=0.2500KB=256B,speed=3.417969MB/s
blocksize_KB=1.0000KB=1024B,speed=20.714962MB/s
blocksize_KB=4.0000KB=4096B,speed=82.859848MB/s
blocksize_KB=16.0000KB=16384B,speed=165.719697MB/s
blocksize_KB=64.0000KB=65536B,speed=291.666667MB/s
```

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。


随机写
800
700
600
500
400
300
200
100
0
MB/S
1
2
3
4
5
6
B
Ram
Disk


顺序写
1000
900
800
700
600
500
400
300
200
100
0
MB/S
64
256
1024
4096
16384
65536
B
Ram
Disk


随机读
1400
1200
1000
800
600
400
200
0
MB/S
64
256
1024
4096
16384
65536
B
Ram
Disk




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。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);
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。

```
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。

*测试文件：*

```
#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 1000//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[17]={"/usr/file1.txt","/usr/file2.txt","/usr/file3.txt","/usr
/file4.txt","/usr/file5.txt","/usr/file6.txt","/usr/file7.txt","/usr/file8.
txt","/usr/file9.txt","/usr/file10.txt","/usr/file11.txt","/usr/file12.txt",
"/usr/file13.txt","/usr/file14.txt","/usr/file15.txt","/usr/file16.txt","/u
sr/file17.txt"};
char
*filepathRam[17]={"/root/myram/file1.txt","/root/myram/file2.txt","/root/my
ram/file3.txt","/root/myram/file4.txt","/root/myram/file5.txt","/root/myram
/file6.txt","/root/myram/file7.txt","/root/myram/file8.txt","/root/myram/fi
le9.txt","/root/myram/file10.txt","/root/myram/file11.txt","/root/myram/fil
e12.txt","/root/myram/file13.txt","/root/myram/file14.txt","/root/myram/fil
e15.txt","/root/myram/file16.txt","/root/myram/file17.txt"};
//维护了两个 filename 的字符串数组，方便使读写文件时为每个进程分配一个独立的文件

char buff[maxline]="Xixiang";


void write_file(int blocksize, bool israND, 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(israND)
                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 isrand, 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(isrand)
                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,"aaaaaaaaaaaaaaaa");}
    //int blocksize=256;
```

```
    for(int blocksize=64;blocksize<=1024*64;blocksize=blocksize*4){
        //for(int Concurrency=7;Concurrency<=15;Concurrency++){
            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]);
                //rite_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)blocksiz
e/1024.0,blocksize,block/eachtime);
      //printf("Concurrency=%d,speed=%fMB/s\n",Concurrency,block/eachtime);
        //}
    }
    return 0;
}
```