**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

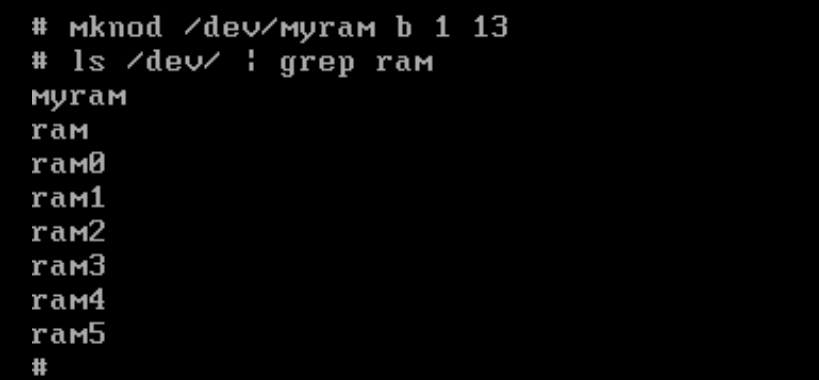
2.重新编译内核，重启reboot。

make build

rebooot

1. 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



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=

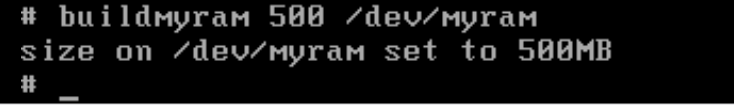
.include <bsd.prog.mk>

重新编译内核，重启虚拟机。

make build

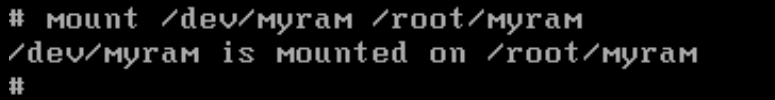
rebooot

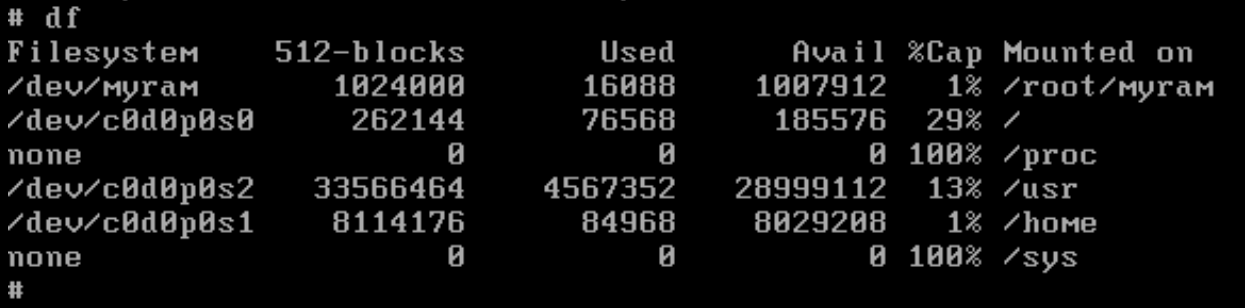
4.2编译buildmyram.c文件，然后执行命令： buildmyram <size in MB> /dev/myram。创建一个RAM盘。



5.在ram盘上创建内存文件系统，mkfs.mfs /dev/myram。

1. 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。





（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

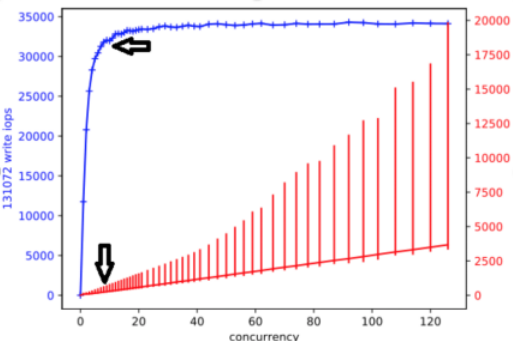
mount /dev/myram /root/myram

1. **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;

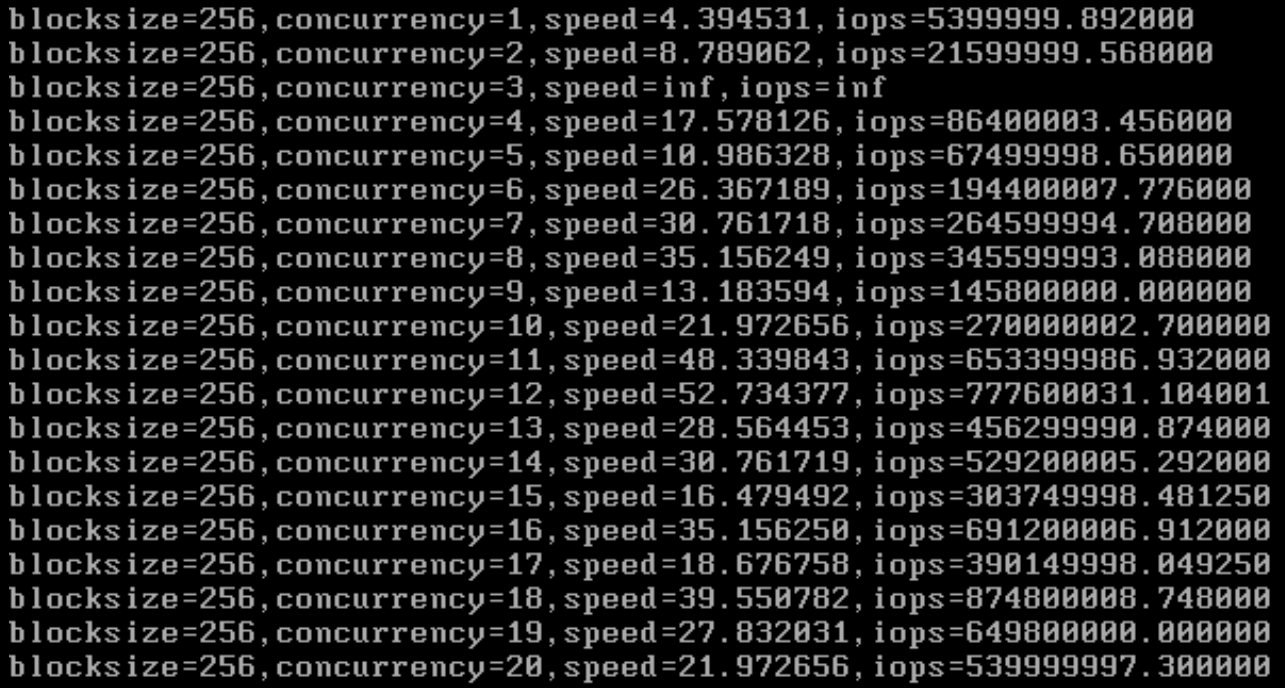
            //printf("blocksize\_KB=%.4fKB=%dB,speed=%fMB/s\n",(double)blocksize/1024.0,blocksize,block/eachtime);

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

        //}

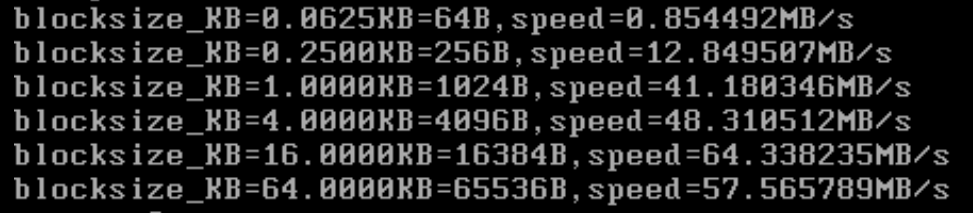
    }

Disk盘顺序写

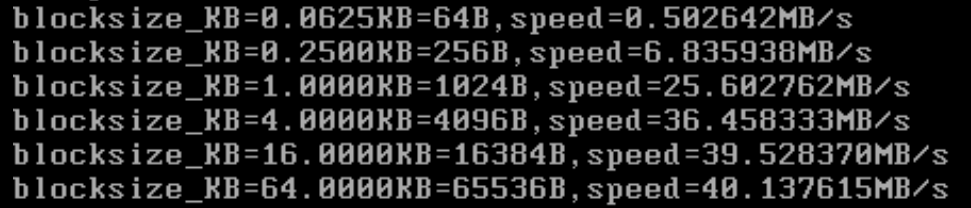


*性能测试的二个变量为“块大小”（推荐 64B/256B/1KB/4KB/16KB/64KB）和“块扫描方式”（顺序/随机）。 可以画四张曲线图对比RAM盘和Disk盘性能（随机读，随机写，顺序读，顺序写）。实验结果预计为RAM盘性能高于DISK盘，特别是随机读写性能。*

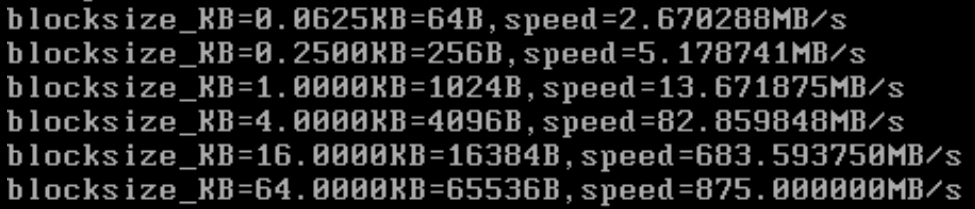
Ram随机写



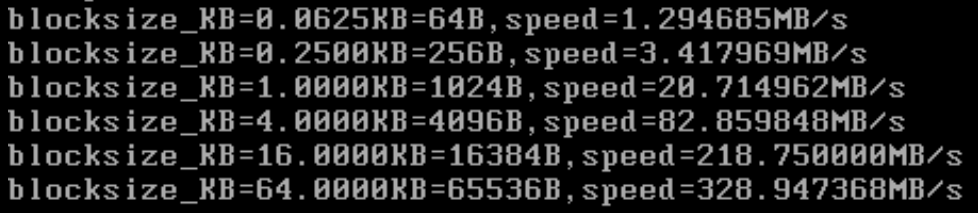
Disk随机写



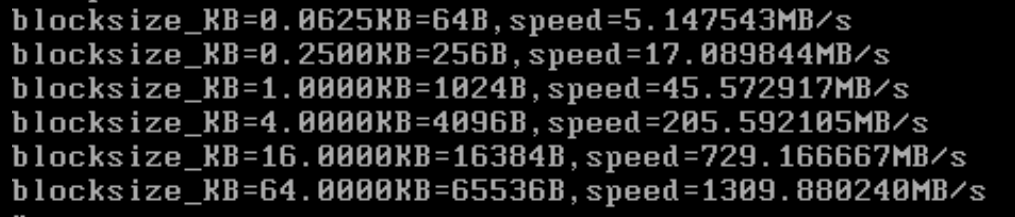
Ram顺序写

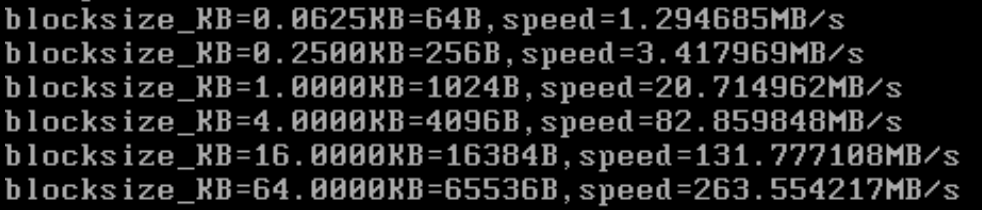


Disk顺序写

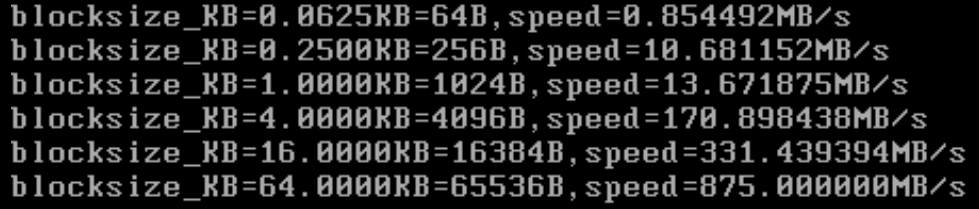


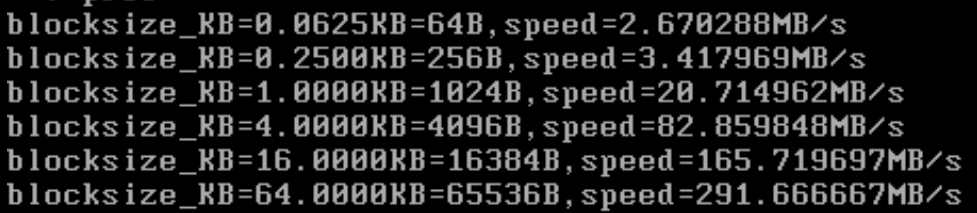
Ram随机读

Disk随机读



Ram顺序读

Disk顺序读



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。

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","/usr/file17.txt"};

char \*filepathRam[17]={"/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","/root/myram/file8.txt","/root/myram/file9.txt","/root/myram/file10.txt","/root/myram/file11.txt","/root/myram/file12.txt","/root/myram/file13.txt","/root/myram/file14.txt","/root/myram/file15.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)blocksize/1024.0,blocksize,block/eachtime);

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

        //}

    }

    return 0;

}