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**：2018 级	**上机实践成绩**：
指导教师：翁楚良	**姓名**：池欣宁	**学号**：10185501409
上机实践名称：I/O subsystem		**上机实践日期**：
上机实践编号：	**组号**：	**上机实践时间**：

一、实验目的

1. 熟悉 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

二、实验任务

1. 在 MINIX3 中安装一块 XMB 大小的 RAM 盘，可以挂载并且存取文件操作。
2. 测试 RAM 和 DISK 盘的文件读写速度，并分析读写速度有差异的原因。

三、使用环境

1. VMware Fusion 专业版 11.5.1
2. MINIX R3.3
3. 虚拟机搭载在 macos 上，物理机内存为 256GSSD 固态硬盘

四、实验过程

Task1. 安装盘---增加 RAM 盘

Step1. 修改/usr/src/minix/drivers/storage/memory/memory.c 增加 RAM 盘数
#define RAMDISK 7

Step2. 重新编译内核`make build MKUPDATE=yes`;重启 reboot 选择 latest kernel

Step3. 创建设备 mknod /dev/myram b 1 13;

/*mknod 的使用方式*/

mknod DEVNAME{b|c} MAJOR MINOR

DEVNAME 创建的设备文件名，这里为/dev/myram

b:块设备：系统从设备中读取数据的时候，直接从内存的 buffer 中读取数据，而不经过磁盘。

这里，我们的 RAM 盘，就是用内存来虚拟一个硬盘，所以这里选项应该是 b。

MAJOR 和 MINOR 表示主设备号和次设备号。

Step4. 检测设备是否创建成功：ls /dev/ | grep ram

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

Step5. 实现 buildmyram 初始化工具：

a. 在/usr/src/minix/commands/目录下新建 buildmyram 文件夹，参照 /usr/src/minix/commands/ramdisk 文件增加 buildmyram 也作为 minix 的内置命令。

b.编写 buildmyram.c 文件

类比 ramdisk.c，但要注意和修改`PATH`变量。

Buildmyram.c 代码如下

```c
#include <minix/paths.h>
#include <sys/ioc_memory.h>
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#define _PATH_MYRAMDISK "/dev/myram"
```

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

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

```
return 1;
}
```

```
d = argc == 2 ? _PATH_MYRAMDISK : argv[2];
if((fd=open(d, O_RDONLY)) < 0) {
perror(d);
return 1;
}
long int MFACTOR=1024*1024;
```

```
size = atol(argv[1])*MFACTOR;//size 是我输入的参数成 kb,mb 就再乘 1024 嘛
```

```
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;
}
```

c. 将 buildmyram.c 修改完毕以后，新建 buildmyram 文件下的 Makefile 文件（此新建也同样类比于 ramdisk），同时修改 commands 目录下的 Makefile 文件（此修改类比于 Project2）

当做完如此修改以后，即可将 buildmyram 作为新增的 commands 内置命令。

Step6. 在 ram 盘上创建内存文件系统：

mkfs.mfs /dev/myram

Step7. 将 ram 盘挂载到用户目录下：

`mount /dev/myram  /root/myram`

Step8. 检验挂载是否成功，输入 df 命令进行检测：

```
# df
Filesystem    512-blocks       Used      Avail %Cap Mounted on
/dev/myram         16384        552      15832   3% /root/myram
/dev/c0d0p0s0     262144      76552     185592  29% /
none                   0          0          0 100% /proc
/dev/c0d0p0s2   33566464    4571912   28994552  13% /usr
/dev/c0d0p0s1    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。

Task2. 性能测试：

编写性能测试文件：

1. 采用多进程并发的同步读写。

2. concurrency 的数目要增加到设备接近“饱和”**状态。**

3. 总吞吐量：fileSize(总文件大小)/执行时间(spend time)

4. 饱和：吞吐量难以继续提升，同时 I/O 延时恶化。

5. 性能测试的变量：“block size“和块扫描方式

P3-test 测试文件的编写：

测试文件框架：

```
void write_file(int blocksize,int isrand,char *filename,int fs){
   //todo
}
void write_file(int blocksize,int isrand,char *filename,int fs){
   //todo
}
long get_time_left(long starttime,long endtime){
   //todo
}
int main(){

   time_t starttime=time(NULL);//开始时间。
```

```
    for(int i=0;i<concurrency;i++){
        while(iter<10000){
            write_file();
            read_file();
        }//将读/写延续一定的时间。

    }
    wait();
    time_t endtime=time(NULL);
    printf("throughput:",throughput);
    return 0;
}
```

P3-test 具体的宏定义和全局变量如下：

```
#define iteration 100000//原来太小了就一直 INF
#define MBFRACTOR 1024*1024
#define KBFRACTOR 1024
#define fileSize 300*MBFRACTOR//总 fileSize 为 300MB
#define bufferSize 6*KBFRACTOR//buffersize 6KB
char buffer[bufferSize]="This is a test, and I dont know
what to use these for";
```

以写测试为例，写测试编写代码如下，读测试编写同理：

```
void write_file(int blocksize, int israND,char
*filepath){
int fd=0;

fd=open(filepath,O_CREAT | O_RDWR | O_SYNC);
/*成功*/
if(fd<0){
printf("Open error!");
return;
}
else{
/*ssize_t write(int fd,const void*buf,size_t count);
参数说明:
```

```
fd:是文件描述符（输出到 command line，就是 1）
buf:通常是一个字符串，需要写入的字符串
count：是每次写入 blocksize 的字节数*/
for(int i=0;i<iteration;i++){
```

```
write(fd,buffer,blocksize);
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
}
lseek(fd,0,SEEK_SET);//重置文件指针，文件开头开始读。
}
//顺序读写时：默认文件指针自由移动
}
```

1. 这里，O_SYNC 表示读写同步；
2. 对于 open/write/read 函数的返回值需进行判断。
3. 因为要测试顺序读写与随机读写，故需要使用 lseek 函数定位文件指针。
4. 读写测试时，需要反复持续一定的时间，这里的时间长度我选择为了 100000，因为在测试中我发现，如果时间长度太短，会很容易的出现 INF 的情况。
5. 这里，所花费时间为开始时间点，以 Linux time(NULL)函数获得，结束时间点同样以相同函数获得，然后相减获得时间长度，单位为 ms.
6. 测试时，concurrency 选择为 7.

Linux time(NULL)函数被包括在 time.h 头文件中：

```
/* time -  获取计算机系统当前的日历时间(Calender Time)
 *        处理日期时间的函数都是以本函数的返回值为基础进行运算
 *
 * 函数原型：
 *    #include <time.h>
 *
 *    time_t time(time_t *calptr);
 *
 * 返回值：
 *    成功：秒数，从 1970-1-1,00:00:00
 *
 * 使用：
 *    time_t now;
 *
```

```
 *      time(&now); // == now = time(NULL);
 */
```

为了使时间获取正确，编写的 timeTest 文件如下：

```
#include<stdio.h>
#include<time.h>
int main(){
time_t starttime=time(NULL);
printf("%Ld\n",starttime);
int status=0;
if(fork()==0){
sleep(30);
exit(1);
}
else{
wait(&status);
time_t endtime=time(NULL);
printf("%Ld\n",endtime);
long spendtime=endtime-starttime;
printf("spendtime = %ld",spendtime);
exit(0);
}
}
```

```
1591502428
1591502458
spendtime = 30
```

P3-test 的测试文件 main 函数如下：

```
int main(){
srand((unsigned)time(NULL));//随机种子
int concurrency=7;//并发的进程的个数
char *filepathDISK="/usr/mytest.txt";
char *filepathRAM="/root/myram/mytest.txt";
```

```
char
*filepathDISKNUM[10]={"/usr/mytest1.txt","/usr/mytest2.tx
t","/usr/mytest3.txt","/usr/mytest4.txt","/usr/mytest5.tx
t","/usr/mytest6.txt","/usr/mytest7.txt","/usr/mytest8.tx
t","/usr/mytest9.txt","/usr/mytest10.txt"};
char
*filepathRAMNUM[10]={"/root/myram/mytest1.txt","/root/myr
am/mytest2.txt","/root/myram/mytest3.txt","/root/myram/my
test4.txt","/root/myram/mytest5.txt","/root/myram/mytest6.
txt","/root/myram/mytest7.txt","/root/myram/mytest8.txt",
"/root/myram/mytest9.txt","/root/myram/mytest10.txt"};
time_t starttime,endtime;
int status = 0;
long spendtime;
int blocksize=64;//64B
int nameIter=0;
char *nameStr;
for(int j=0;j<bufferSize; j=j+6){
strncat(buffer,"haha haha!\n",6);
}
do{
starttime=time(NULL);
// printf("starttime:%lld",starttime);
```

```
for(int i=0;i<concurrency;i++){
if(fork()==0){
//随机写
int iter=0;
// write_file(blocksize,1,filepathDISKNUM[i]);
// write_file(blocksize,0,filepathDISKNUM[i]);
// write_file(blocksize,0,filepathRAMNUM[i]);
write_file(blocksize,1,filepathRAMNUM[i]);
```

```
/*write*/
```

```
// read_file(blocksize,1,filepathDISKNUM[i]);
// read_file(blocksize,0,filepathDISKNUM[i]);
// read_file(blocksize,0,filepathRAMNUM[i]);
// read_file(blocksize,1,filepathRAMNUM[i]);
```

```
exit(1);
}
}
while(wait(NULL)!=-1){}
```

```
endtime=time(NULL);
printf("endtime:%lld\n",endtime);
spendtime=get_time_left(starttime,endtime);//操作所花费的时
间
printf("spendtime:%ld\n",spendtime);
double throughput=1.0*blocksize/spendtime;
printf("blocksize = %d,throughput
= %lf\n",blocksize,throughput);//printf("plz input
another blocksize\n");
blocksize=blocksize*2;
//printf("the blocksize = %d\n",blocksize);
// concurrency++;
}while( blocksize < 9092);
```

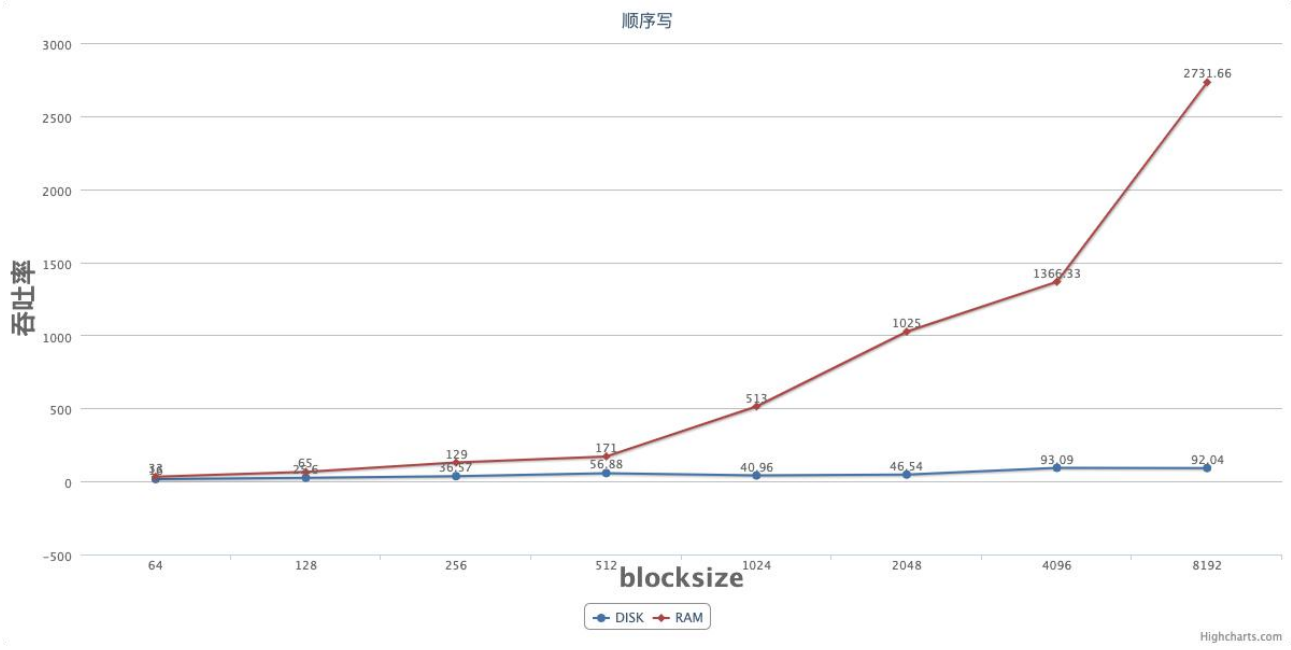
```
/*等待子进程完成后，获取计算时间,计算读写操作所花时间，延时，吞吐量
*/
return 0;
}
```

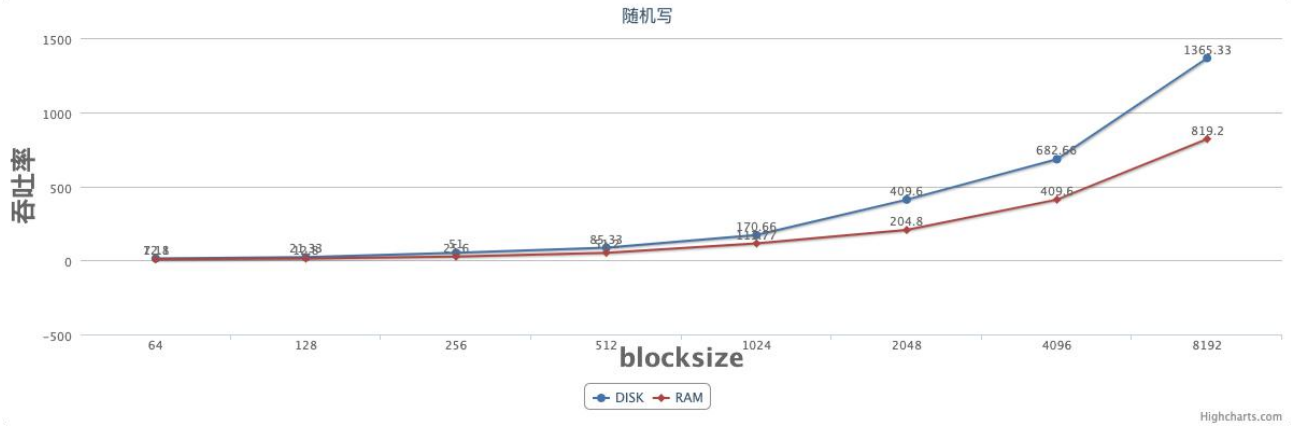
这里 维护了两个 filename 的字符串数组，方便使读写文件时为每个进程分配一个杜立德文件。

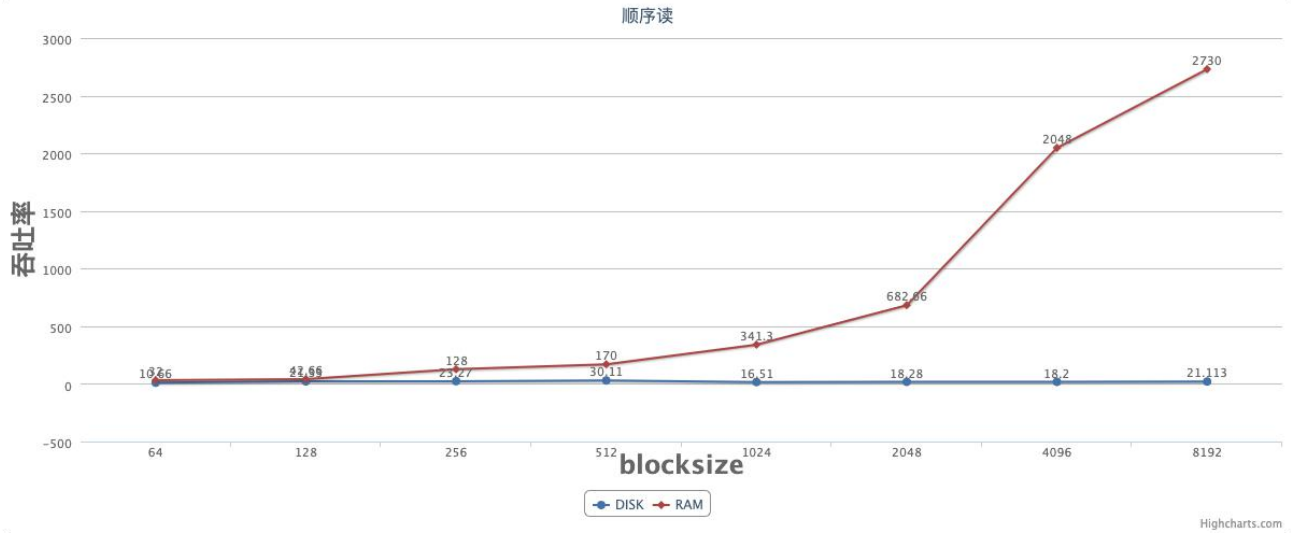
为了减少主机操作系统的缓存机制造成的误差，文件总大小越大越好，这里文件总大小为 fileSize.

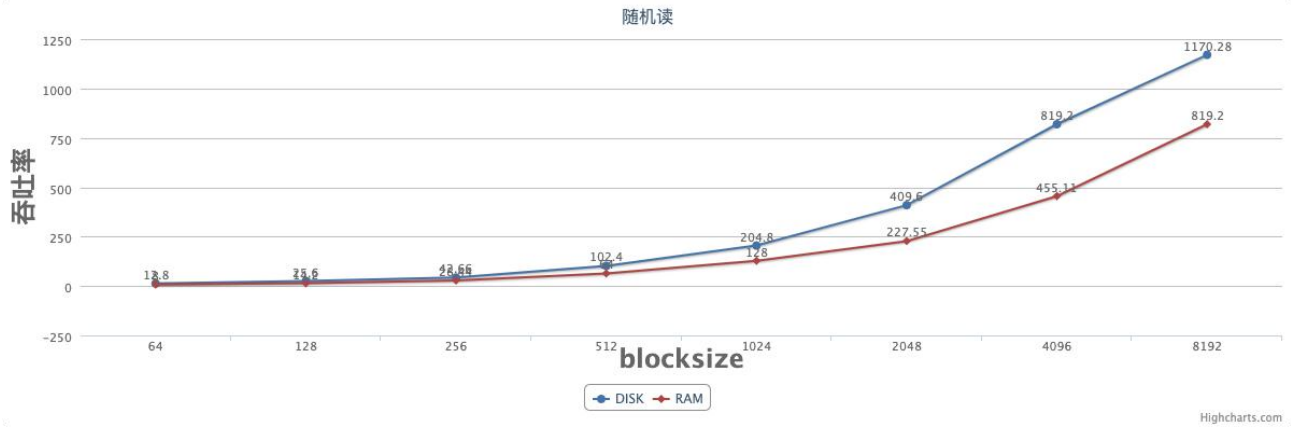
Task 3:分析 顺序/随机 读/写的性能测试的结果。

随机写			顺序写			随机读			顺序读		
blockSize	DISK	RAM	blocksize	DISK	RAM	blocksize	DISK	RAM	blocksize	DISK	RAM
64	12.8	7.11	64	16	33	64	12.8	8	64	10.66	32
128	21.33	12.8	128	25.6	65	128	25.6	14.2	128	21.33	42.66
256	51	25.6	256	36.57	129	256	42.66	28.44	256	23.27	128
512	85.33	51.2	512	56.88	171	512	102.4	64	512	30.11	170
1024	170.66	113.77	1024	40.96	513	1024	204.8	128	1024	16.51	341.3
2048	409.6	204.8	2048	46.54	1025	2048	409.6	227.55	2048	18.28	682.66
4096	682.66	409.6	4096	93.09	1366.33	4096	819.2	455.11	4096	18.2	2048
8192	1365.33	819.2	8192	92.04	2731.66	8192	1170.28	819.2	8192	21.113	2730







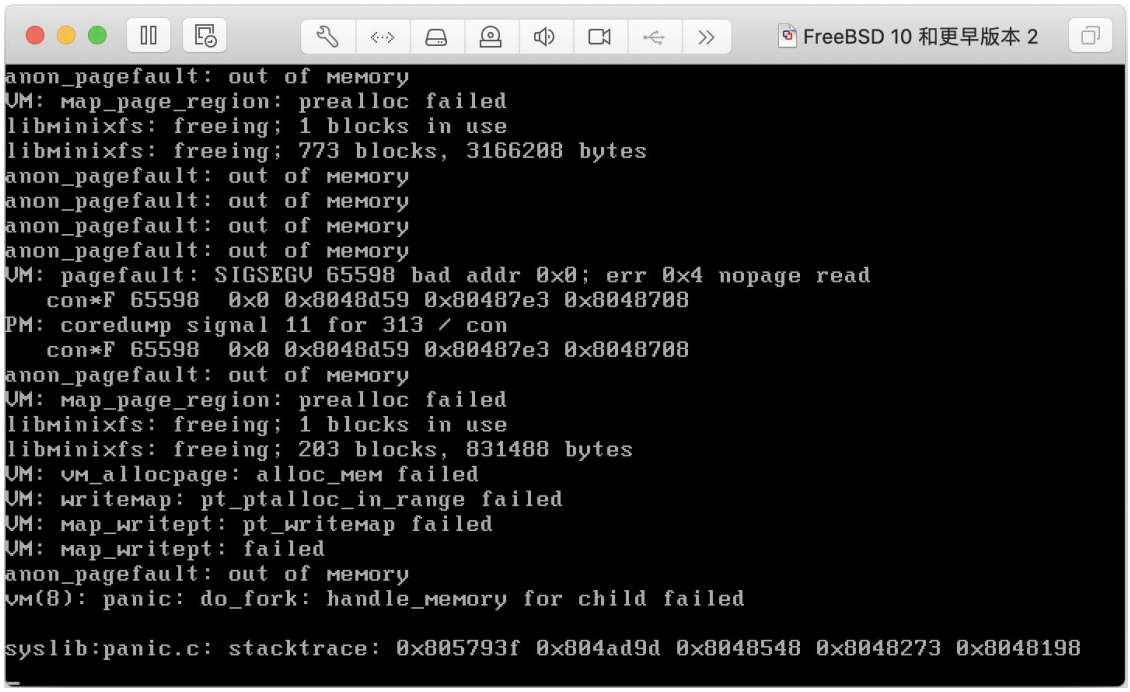


【注】这里折线图由 http://charts.udpwork.com/ 网站生成。
这里同时检验 concurrency 的饱和 threshold.

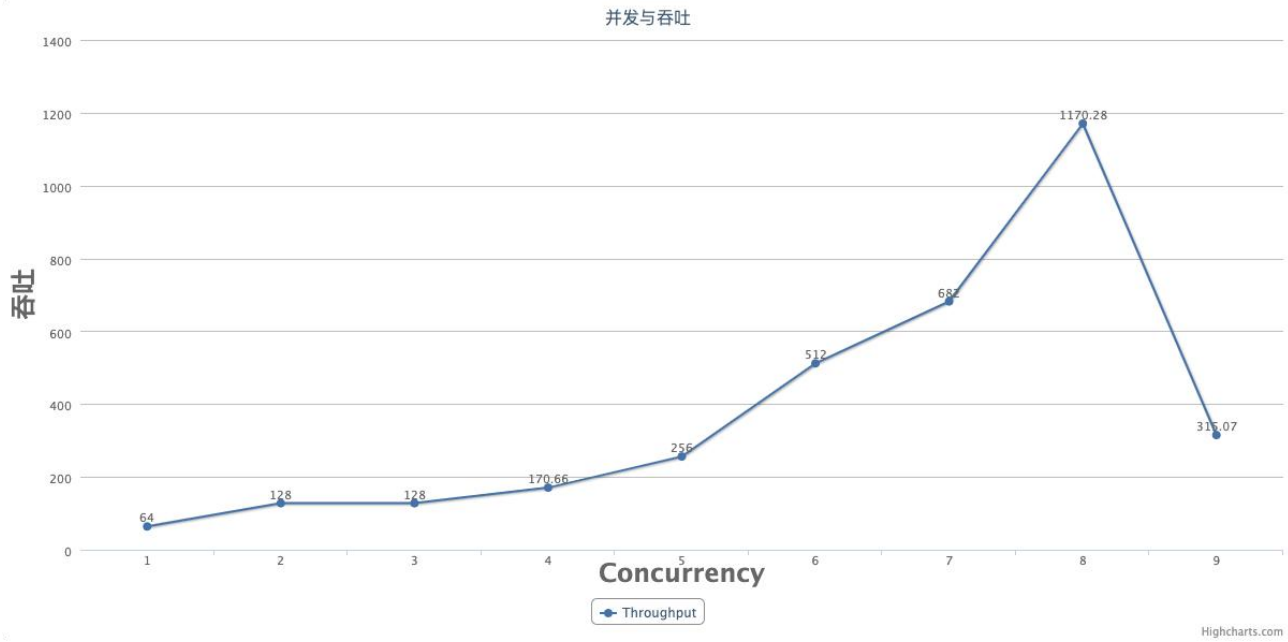
```
printf("blocksize = %d,throughput = %lf\n",blocksize,through
//printf("plz input another blocksize\n");
//blocksize=blocksize*2;
//printf("the blocksize = %d\n",blocksize);
concurrency++;
}while( concurrency<=17);
```

【注意】当 concurrency 增加 1 时，blocksize 也要相应的变大。

```
# ./con
endtime:1591537746
spendtime:1
blocksize = 64,throughput = 64.000000,concurrency = 1
endtime:1591537747
spendtime:1
blocksize = 128,throughput = 128.000000,concurrency = 2
endtime:1591537749
spendtime:2
blocksize = 256,throughput = 128.000000,concurrency = 3
endtime:1591537752
spendtime:3
blocksize = 512,throughput = 170.666667,concurrency = 4
endtime:1591537756
spendtime:4
blocksize = 1024,throughput = 256.000000,concurrency = 5
endtime:1591537760
spendtime:4
blocksize = 2048,throughput = 512.000000,concurrency = 6
endtime:1591537766
spendtime:6
blocksize = 4096,throughput = 682.666667,concurrency = 7
endtime:1591537773
spendtime:7
blocksize = 8192,throughput = 1170.285714,concurrency = 8
endtime:1591537825
spendtime:52
blocksize = 16384,throughput = 315.076923,concurrency = 9
endtime:1591537829
spendtime:4
```



这里出现 coredump 是因为一开始预留的 memory 太小了。



在 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，当并发数为增加到饱和状态时，此饱和状态不仅与 concurrency 有关，也与 blocksize 有关。

在出现饱和前，总吞吐随着并发数增长。

Task 4. 性能测试结果分析

RAM: RAM 本质上是将一部人的内存(RAM)当作 DISK 来使用，RAM 有固定的大小，可以像正常硬盘区分那样去使用。

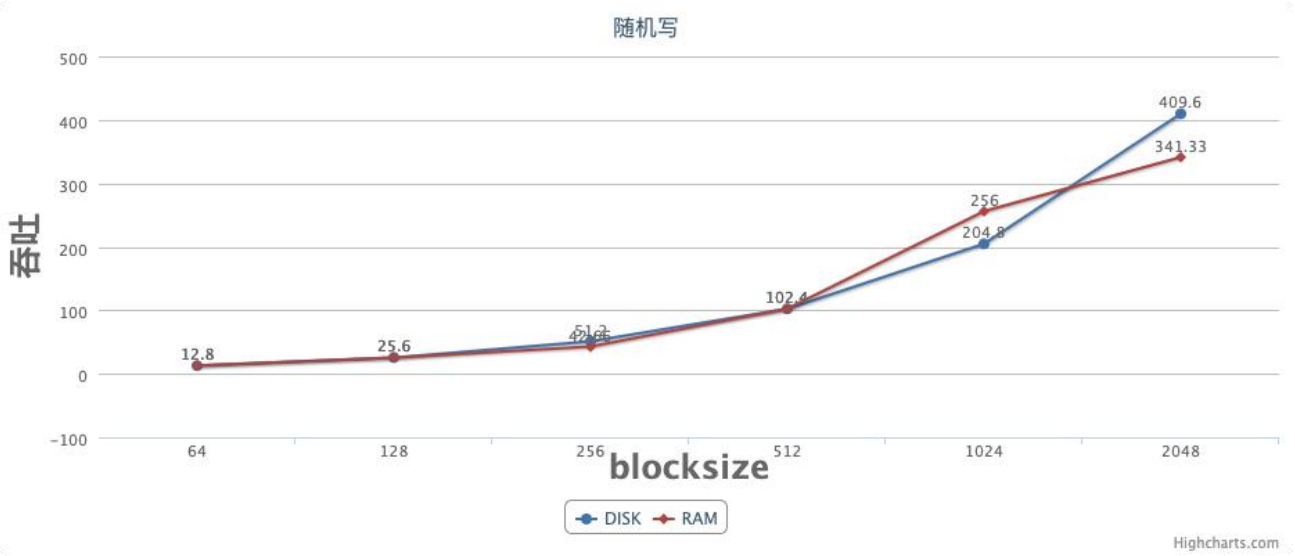
RAM 具有存取速度快，读写方便的特点。当然，数据不能长久保持，所以再关机以后我们需要对 RAM 盘重新挂载。

而具有机械结构的硬盘 DISK，在读取时，需要磁头运动到相应的位置，然后再进行磁信号的读取。即使是硬盘的另一个种类是固态硬盘，使用闪存，但也不能和真正的处于内存的

RAM 进行比较。

在随机读写测试中，write/read 的随机读写我在本机内存(256G 固态硬盘）测试中，DISK 和 RAM 的性能相近，随机写测试如下：

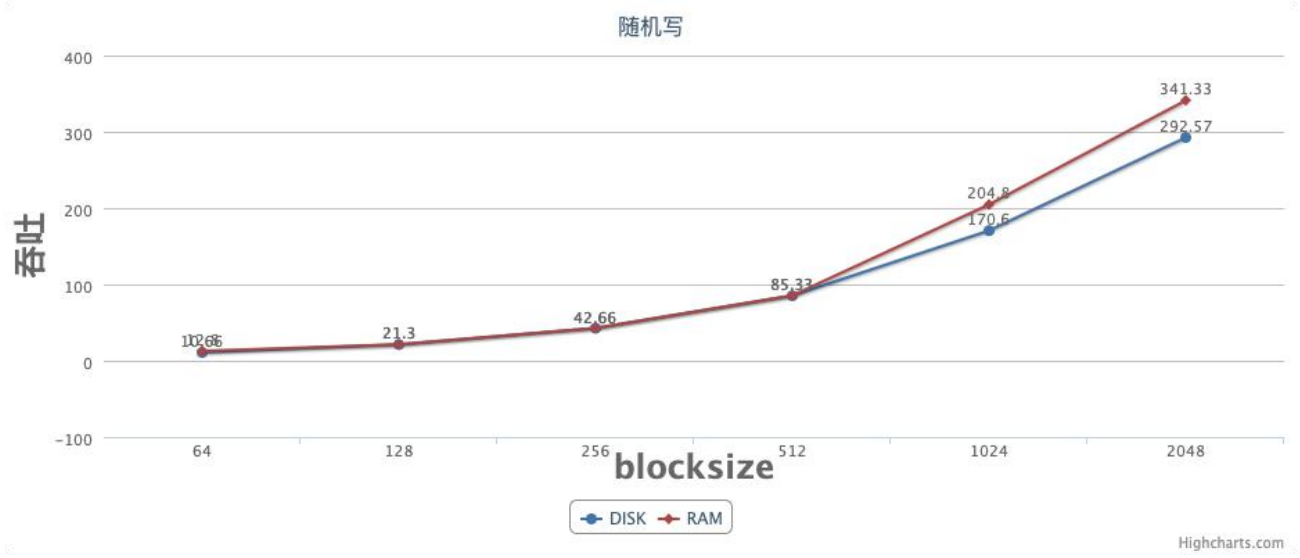
blocksize	DISK	RAM	Concurrency
64	12.8	12.8	14
128	25.6	25.6	14
256	51.2	42.66	14
512	102.4	102.4	14
1024	204.8	256	14
2048	409.6	341.33	14



blocksize	DISK	RAM	Concurrency
64	10.66	10.66	15
128	18.28	21.3	15
256	42.66	51.2	15
512	73.13	85.33	15
1024	146.28	170.66	15
2048	341.33	341.33	15

blocksize	DISK	RAM	Concurrency
64	10.66	12.8	20
128	21.3	21.3	20
256	42.66	42.66	20
512	85.33	85.33	20

1024	170.6	204.8	20
2048	292.57	341.33	20




```
# df
Filesystem    512-blocks       Used      Avail %Cap Mounted on
/dev/myram         49152       1576      47576   3% /root/myram
/dev/c0d0p0s0     262144      91984     170160  35% /
none                   0          0          0 100% /proc
/dev/c0d0p0s2   33566464    5308128   28258336  15% /usr
/dev/c0d0p0s1    8114176      84968    8029208   1% /home
none                   0          0          0 100% /sys
# df
Filesystem    512-blocks       Used      Avail %Cap Mounted on
/dev/myram         49152       2656      46496   5% /root/myram
/dev/c0d0p0s0     262144      91984     170160  35% /
none                   0          0          0 100% /proc
/dev/c0d0p0s2   33566464    5309216   28257248  15% /usr
/dev/c0d0p0s1    8114176      84968    8029208   1% /home
none                   0          0          0 100% /sys
# df
Filesystem    512-blocks       Used      Avail %Cap Mounted on
/dev/myram         49152       3016      46136   6% /root/myram
/dev/c0d0p0s0     262144      91984     170160  35% /
none                   0          0          0 100% /proc
/dev/c0d0p0s2   33566464    5309584   28256880  15% /usr
/dev/c0d0p0s1    8114176      84968    8029208   1% /home
none                   0          0          0 100% /sys
```

截屏

三次读写过程中的示意图，比较 root/myram 的使用情况。

为什么 RAM 的读写性能在理论上要远好于机械硬盘(理论)?

在顺序读写过程中，利用程序的局部性，(可以理解成磁盘的在寻道时间上花费的少一些)，但 RAM 盘作为内存，读写性能肯定显著优于作为硬盘(机械/固态)的外存。

在随机读写过程中，程序的局部性无法利用，所以 RAM 盘的性能肯定显著优于 DISK 盘。

机械硬盘与 SSD：

当我在本机进行测试时，由于电脑的外部存储均采用 SSD 固态硬盘，所以测试效果并不如运行在机械硬盘上的效果好。

机械硬盘存在机械结构，读取和写入时由磁头在转动的盘上转动寻找文件所在扇区。

SSD 固态硬盘由主控缓存和闪存组成，不存在机械结构，读取和写入时无需寻道时间。

影响机械硬盘速度的因素有寻道时间/旋转延迟/数据传输时间，其中，寻道时间和旋转延迟是影响硬盘速度的根本因素。

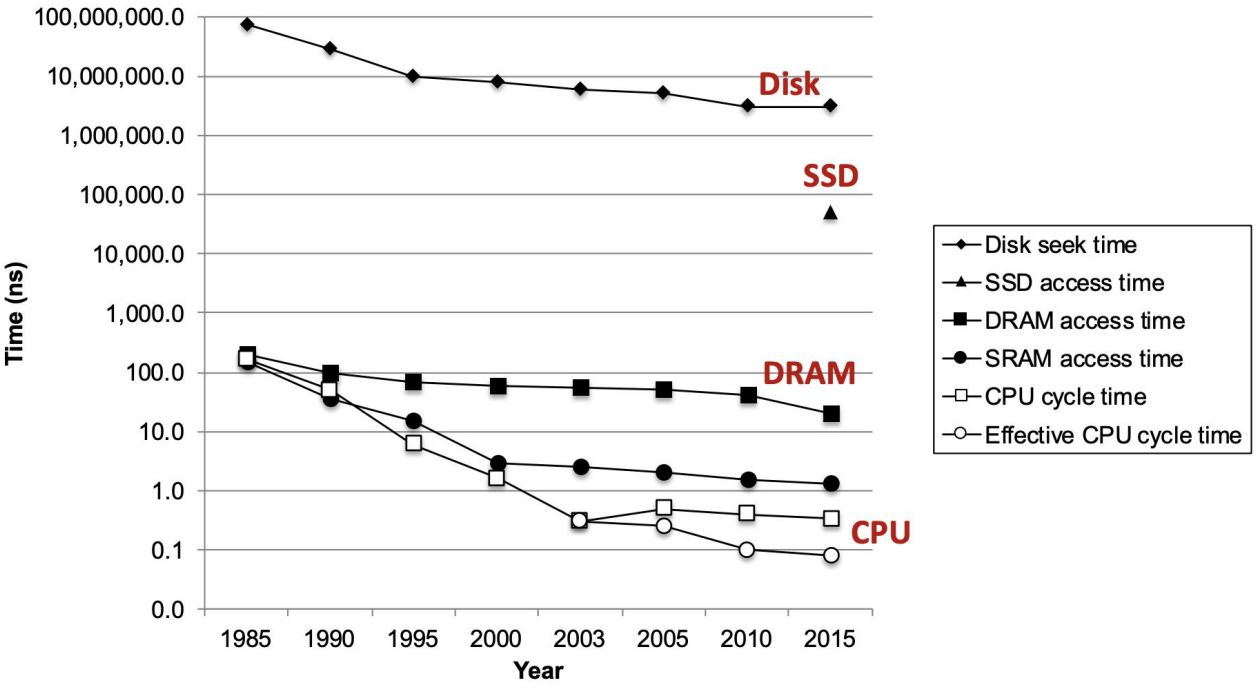
所以此时，操作系统磁盘驱动程序通过合理调度，如此题的顺序/随机读写过程中，可以减少机械硬盘平均服务时间的目的。

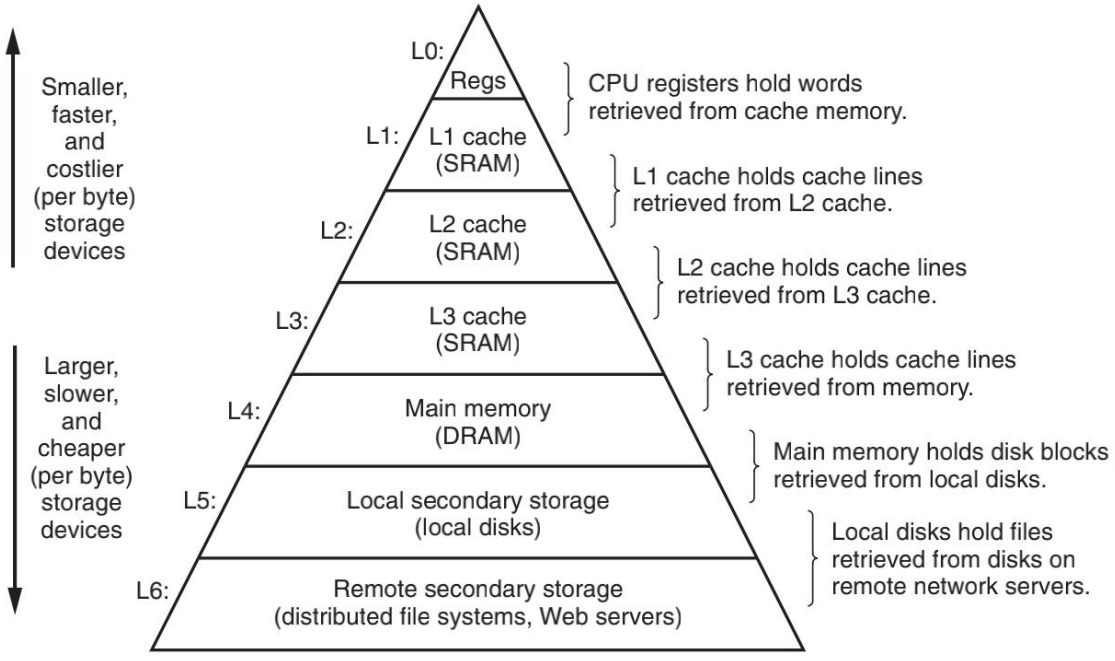
五、总结

这一次通过 I/O 读写测试，首次真正感觉到了内外存储设备对于性能的影响，同时也从多并发的角度，体会到了多线程对于软件性能的重要性。

同时进一步明确了 minix 系统如何增加 shell commands 和操作系统的编译，以及修改类 linux 系统设置的相关代码(配置)文件的修改，更进一步的接触到了操作系统，而不是像以前使用电脑操作系统的时候感觉操作系统的透明。

最后也回顾了理论上计算机系统的组成和 memory hierarchy 和 上学期学习 CSAPP 时关于存储金字塔和 CPU memory gap 的理解。


100,000,000.0
10,000,000.0
1,000,000.0
100,000.0
10,000.0
1,000.0
100.0
10.0
1.0
0.1
0.0
Time (ns)
1985
1990
1995
2000
2003
2005
2010
2015
Year
Disk
SSD
DRAM
CPU
Disk seek time
SSD access time
DRAM access time
SRAM access time
CPU cycle time
Effective CPU cycle time


Smaller, faster, and costlier (per byte) storage devices
Larger, slower, and cheaper (per byte) storage devices
L0:
Regs
CPU registers hold words retrieved from cache memory.
L1:
L1 cache (SRAM)
L1 cache holds cache lines retrieved from L2 cache.
L2:
L2 cache (SRAM)
L2 cache holds cache lines retrieved from L3 cache.
L3:
L3 cache (SRAM)
L3 cache holds cache lines retrieved from memory.
L4:
Main memory (DRAM)
Main memory holds disk blocks retrieved from local disks.
L5:
Local secondary storage (local disks)
Local disks hold files retrieved from disks on remote network servers.
L6:
Remote secondary storage (distributed file systems, Web servers)