Unix 实验报告

实验： 实验 1 同步与异步 write 的效率比较

专业： 计算机科学与技术

班级： 1 班

姓名： 姚怀聿

学号： 22920202204632

2022 年 10 月 7 日

目　　录

一、 实验内容描述..... 2

二、 设计、实验构思..... 3

三、 实验结果 9

四、 体会和建议 11

五、 完成人姓名及完成时间..... 12

一、 实验内容描述

本实验的目的是掌握 UNIX 的文件 I/O 系统调用。

实验要求编写程序：`timewrite <outfile> [sync]`

其中，sync 是可选参数，若有该参数，表示输出文件用异步模式打开；反之，用异步模式。体现在程序中就是在使用`open`函数打开文件时，第二个参数是否或`O_SYNC`。

实验有以下几点要求：

1. 显示的时间要尽量接近 write 操作过程所花的时间，不要将从磁盘读文件的时间计入显示结果中。
2. 严格按照下表的结果格式输出(BUFFERSIZE 从 256 开始算到 128K)，抬头和分割线省略。

BUFFSIZE	用户 CPU（s）	系统 CPU（s）	时钟时间（s）	循环次数
1	20.03	117.50	138.73	516 581 760
2	9.69	58.76	68.6	258 290 880
4	4.60	36.47	41.27	129 145 440
8	2.47	15.44	18.38	64 572 720
16	1.07	7.93	9.38	32 286 360
32	0.56	4.51	8.82	16 143 180
64	0.34	2.72	8.66	8 071 590
128	0.34	1.84	8.69	4 035 795
256	0.15	1.30	8.69	2 017 898
512	0.09	0.95	8.63	1 008 949
1 024	0.02	0.78	8.58	504 475
2 048	0.04	0.66	8.68	252 238
4 096	0.03	0.58	8.62	126 119
8 192	0.00	0.54	8.52	63 060
16 384	0.01	0.56	8.69	31 530
32 768	0.00	0.56	8.51	15 765
65 536	0.01	0.56	9.12	7 883
131 072	0.00	0.58	9.08	3 942
262 144	0.00	0.60	8.70	1 971
524 288	0.01	0.58	8.58	986

二、 设计、实验构思

1. 实验要求执行程序得到一个输出文件，该文件的名称是指定的，而该文件是否以同步模式打开是可选的，故程序至少应传入一个输入参数，至多应传入两个输入参数，我们可以利用 argc 判断传入参数个数是否合法。由于 timewrite 作为程序的名称，故体现在程序中，即为

```
if(argc <= 1 || argc > 3) {
    err_quit("Usage: ./timewrite <outfile> [sync]\n");
}
```

2. 为了达到显示的时间尽量接近 write 操作所花的时间这一要求，我们可以充分利用缓冲区来读取，具体如下：我们先把从标准输入中读入的数据放在缓冲区，计算机再直接从缓冲区中取数据，这样就可以减少磁盘的读写次数，而计算机对缓冲区的操作速度远高于对磁盘的操作速度，故应用缓冲区可大大提高计算机的运行速度。缓冲区就是一块内存区，它用在输入输出设备和 CPU 之间，用来暂存数据。为了准确计算 write 耗费的时间，很重要的一点是要避免将 read 的时间计入，因为 I/O 操作的时间通常是毫秒级的，不可以忽略。一种有效的方法是，设置一个与输入文件长度相同的缓冲区，一次性地将输入文件读入缓冲区，而后就不必再读输入文件。这样就可以有效避免计入 read 的时间。在程序具体实现

中，我首先为缓冲区分配了一块很大的空间(1Gbit)，然后一次性从标准输入中将文件读入，程序代码如下：

```
7 #define BUFFERSIZE 1073741824
8 char buf[BUFFERSIZE];
```

```
33      long long length; // the size of input file
34      // read the input file into buffer
35      if((length = read(STDIN_FILENO, buf, BUFFERSIZE)) < 0) {
36          fprintf(stderr, "fille read error\n");
37          exit(1);
38      }
```

实验要求按照表格输出结果，且 BUFFERSIZE 要求从 256 开始算到 128K(即 131072)结束；对于这一要求，我们可以用一个循环来实现，BUFFERSIZE 从 256 开始，每计算一次时间，将 BUFFERSIZE 乘以 2，直到 BUFFERSIZE 计算到 131072 为止。在每一个循环中，我们在开始写文件之前调用 times()，记录下开始时间，然后在整个输入缓冲区都复制到输出文件之后，再调用 times()，两次调用 times()的时间间隔，就是在这个给定大小的输出缓冲区的限制下，复制整个输入文件所耗费的写时间。至于在每一次写的时候所执行的其他语句，它们相较于 I/O 操作，所花费的时间极小，可以忽略不计。

程序代码如下：

```
57      for(buffersize = 256; buffersize <= 131072; buffersize *= 2) {
58          // locate to the head
59          if(lseek(fd, 0, SEEK_SET) == -1) {
60              err_sys("lseek error\n");
61          }
62          clockStart = times(&tmsStart);
63          i = 0;
64          clockStart = times(&tmsStart);
65          int cnt = 0;
66          for(i = 0; i < length; i += buffersize) {
67              writesize = length - i;
68              if(buffersize < writesize) writesize = buffersize;
69              if(write(fd, buf + i, writesize) != writesize) {
70                  err_sys("write error\n");
71                  exit(1);
72              }
73              cnt++;
74          }
75          clockEnd = times(&tmsEnd);
76          fprintf(stdout, "%8d        %.2lf          %.2lf          %.2lf %10d\n", buffersize, (double)(tmsEnd.tms_utime - tmsStart.tms_utime) / ticks, (double)(tmsEnd.tms_stime - tms
Start.tms_stime) / ticks, (double)(clockEnd - clockStart) / ticks, cnt);
77      }
```

实验的设计如上，程序实现流程如下：

步骤 1.判断传入参数是否合法

```
if(argc <= 1 || argc > 3) {
    err_quit("Usage: ./timewrite <outfile> [sync]\n");
}

if(argc == 3 && strcmp(argv[2], "sync") != 0) { // 3 options but the third is not "sync"
    err_quit("Usage: ./timewrite <outfile> [sync]\n");
}
```

第一种情况，传入参数个数不合法；

第二种情况，传入参数中的第二个参数不为 “sync”

步骤 2.判断以哪种方式打开输出文件

```
int fd;

if(argc == 2) { // open file asynchronously
    if((fd = open(argv[1], O_WRONLY|O_CREAT, S_IRUSR|S_IWUSR)) < 0) {
        fprintf(stderr, "file: %s open failed\n", argv[1]);
        exit(1);
    }
} else if(argc == 3) { // open file synchronously
    if((fd = open(argv[1], O_WRONLY|O_CREAT|O_SYNC, S_IRUSR|S_IWUSR)) < 0) {
        fprintf(stderr, "file: %s open failed\n", argv[1]);
        exit(1);
    }
}
```

如果传入一个参数，就以异步模式打开；

如果传入两个参数，就以同步模式打开，即以 O_SYNC 模式打开；

若打开失败，输出失败信息，并退出(1)。

步骤 3.从标准输入中读取文件到缓冲区中

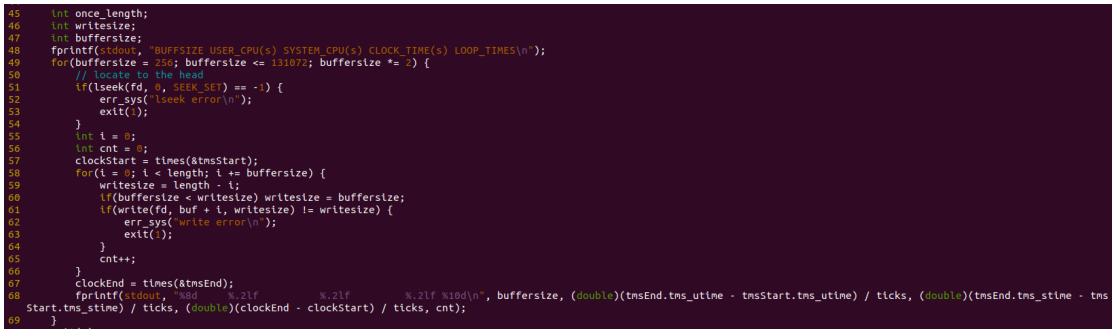
```
long long length; // the size of input file
// read the input file into buffer
if((length = read(STDIN_FILENO, buf, BUFFERSIZE)) < 0) {
    fprintf(stderr, "fille read error\n");
    exit(1);
}
```

length 记录了所读取的文件大小，如果失败，输出失败信息并退出(1)。

步骤 4.定义时钟用于记录进程运行的时间

```
int ticks = sysconf(_SC_CLK_TCK);
clock_t clockStart, clockEnd;
struct tms tmsStart, tmsEnd;
```

步骤 5.循环 buffersize，每次以固定的 buffersize 写入



每次循环开始时，先将文件偏移量定位到输出文件的开始

```
// locate to the head
if(lseek(fd, 0, SEEK_SET) == -1) {
    err_sys("lseek error\n");
    exit(1);
}
```

若定位失败，输出失败信息，并退出(1)；

每次写入开始前，先调用 times()，获得写文件开始的时间，每次写文件结束后，再次调用 times()，获得写文件结

束的时间，最后将两时间相减，得到写文件所用的时间。具体来说有三类时间，分别是用户调用时间、系统调用时间以及时钟时间。

```
#include <sys/times.h>
clock_t times(struct tms *buf);
struct tms {
    clock_t tms_utime;   /* 记录进程除系统调用外所使用的 CPU 时间 */
    clock_t tms_stime;   /* 记录进程的系统调用所使用的 CPU 时间 */
    clock_t tms_cutime;  /* 记录子进程除系统调用外所使用的 CPU 时间 */
    clock_t tms_cstime;  /* 记录子进程的系统调用所使用的 CPU 时间 */
};
```

调用函数 times()时，向 times()传入一个结构体 tms 的地址，函数 times()会修改地址的内容，修改后得到的 tms_utime 便是用户使用时间，tms_stime 便是系统使用时间，在本次实验中，我们只需要用到这两个，另外两个无需使用；至于时钟时间，该 times()函数的返回值 clock_t 即为时钟时间。具体程序如下：

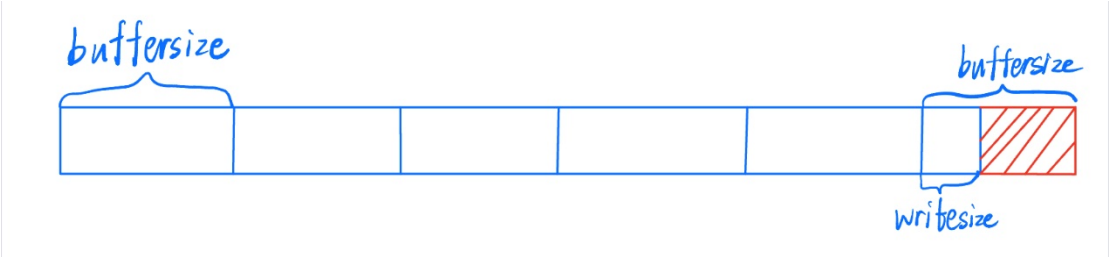
```
int ticks = sysconf(_SC_CLK_TCK);
clock_t clockStart, clockEnd;
struct tms tmsStart, tmsEnd;

clockStart = times(&tmsStart);

clockEnd = times(&tmsEnd);
```

这里还有一个小问题，times()函数所有的值均是以”滴答”为单位的，我们需要用到函数 sysconf(_SC_CLK_TCK)

来获得所运行系统每秒的滴答数，最后将两者相除，便可得到程序运行的秒数。

```
68          fprintf(stdout, "%8d    %.2lf          %.2lf          %.2lf %10d\n", buffersize, (double)(tmsEnd.tms_utime - tmsStart.tms_utime) / ticks, (double)(tmsEnd.tms_stime - tms
    Start.tms_stime) / ticks, (double)(clockEnd - clockStart) / ticks, cnt);
69      }
```

每次往文件中写入程序时，我们需要比较一下还未写入的长度与 buffersize 的大小，我们将写入的大小是较小的那个。




```
for(i = 0; i < length; i += buffersize) {
    writesize = length - i;
    if(buffersize < writesize) writesize = buffersize;
    if(write(fd, buf + i, writesize) != writesize) {
        err_sys("write error\n");
        exit(1);
    }
    cnt++;
}
```

每写一个 writesize 长度，都需要将 i 加上 buffersize，在最后不够一个 buffersize 大小的时候，将剩余的长度全部写入；每成功写入一次，将 cnt 加 1，即为循环次数加 1。

最后，在每次循环结束时，将 buffersize 变为原来的两倍，继续在新的 buffersize 下重新写入文件。

至此，程序实现流程描述完毕。

为了看出异步和同步 write 的差异，我另写了一个程序 gen_ran()用于生成一个指定大小的随机文件(默认是 100MByte)。

该程序的编译方式为`gcc gen_ran.c`

该程序的使用方式为`./a.out <input_file> [size(MB)]`

```
gcc gen_ran.c
```

```
rongrong@rongrong-virtual-machine:~/Desktop/unix/homework/homework1$ ./a.out input_file
random input file input_file was generated successfully
```

三、 实验结果

源程序名	可执行程序名
timewrite.c	timewrite
gen_ran.c	a.out

编译生成可执行文件：

对于 timewrite.c，使用如下指令：

`make -f Makefile`

```
rongrong@rongrong-virtual-machine:~/Desktop/unix/homework/homework1$ ls
apue.h  error.c  gen_ran.c  Makefile  timewrite.c
rongrong@rongrong-virtual-machine:~/Desktop/unix/homework/homework1$ make -f Makefile
gcc -c timewrite.c
gcc -c error.c
gcc timewrite.o error.o -o timewrite
rongrong@rongrong-virtual-machine:~/Desktop/unix/homework/homework1$ ls
apue.h  error.c  error.o  gen_ran.c  Makefile  timewrite  timewrite.c  timewrite.o
rongrong@rongrong-virtual-machine:~/Desktop/unix/homework/homework1$
```

对于 gen_ran.c，使用如下指令：

`gcc gen_ran.c`

```
rongrong@rongrong-virtual-machine:~/Desktop/unix/homework/homework1$ gcc gen_ran.c
rongrong@rongrong-virtual-machine:~/Desktop/unix/homework/homework1$ ls
a.out  apue.h  error.c  error.o  gen_ran.c  Makefile  timewrite  timewrite.c  timewrite.o
```

运行程序：

首先生成指定大小的随机文件：

`./a.out input_file 50`表示生成一个 50MB 大小的名称为 “input_file”的文件。

```
rongrong@rongrong-virtual-machine:~/Desktop/unix/homework/homework1$ ./a.out input_file 50
random input file input_file was generated successfully
rongrong@rongrong-virtual-machine:~/Desktop/unix/homework/homework1$ ls
a.out  apue.h  error.c  error.o  gen_ran.c  input_file  Makefile  timewrite  timewrite.c  timewrite.o
rongrong@rongrong-virtual-machine:~/Desktop/unix/homework/homework1$ ll input_file
-rw-rw-r-- 1 rongrong rongrong 52428800 10月  8 11:23 input_file
```

然后执行 timewrite 程序查看写入程序的时间：

`./timewrite output_file <input_file`

表示以异步模式打开 output_file 并将 input_file 写入。

```
rongrong@rongrong-virtual-machine:~/Desktop/unix/homework/homework1$ ./timewrite output_file <input_file
BUFFSIZE USER_CPU(s) SYSTEM_CPU(s) CLOCK_TIME(s) LOOP_TIMES
     256      0.02          0.92         0.95     204800
     512      0.04          0.11         0.15     102400
    1024      0.03          0.06         0.09      51200
    2048      0.01          0.05         0.07      25600
    4096      0.00          0.04         0.03      12800
    8192      0.01          0.02         0.03       6400
   16384      0.00          0.02         0.02       3200
   32768      0.00          0.03         0.02       1600
   65536      0.00          0.02         0.02        800
  131072      0.00          0.02         0.03        400
rongrong@rongrong-virtual-machine:~/Desktop/unix/homework/homework1$ diff output_file input_file
```

`diff output_file input_file`查看两文件是否有不同之处，未输出任何信息，表示两文件内容相同。

`./timewrite output_file sync <input_file`

表示以同步模式打开 output_file 并将 input_file 写入。

```
rongrong@rongrong-virtual-machine:~/Desktop/unix/homework/homework1$ ./timewrite output_file sync <input_file
BUFFSIZE USER_CPU(s) SYSTEM_CPU(s) CLOCK_TIME(s) LOOP_TIMES
     256      0.02         23.56         78.71     204800
     512      0.02          8.14         13.20     102400
    1024      0.01          4.05          6.41      51200
    2048      0.00          2.11          3.32      25600
    4096      0.00          1.07          1.68      12800
    8192      0.00          0.57          0.86       6400
   16384      0.00          0.32          0.48       3200
   32768      0.00          0.19          0.28       1600
   65536      0.00          0.10          0.13        800
  131072      0.00          0.09          0.12        400
rongrong@rongrong-virtual-machine:~/Desktop/unix/homework/homework1$ diff input_file output_file
```

`diff output_file input_file`查看两文件是否有不同之处，未输出任何信息，表示两文件内容相同。

通过对比，我们可以看到，异步相较于同步，在写入文件比较大的情况下，异步模式打开写入所花费的时间远小于同步模式打开写入所花费的时间。

四、 体会和建议

体会：本次实验可以说是真正意义上的第一次实验，通过本次实验，我体会到了操作系统的奥妙。以前写程序，都只是在写函数，然后调用，这次是第一次感受到了系统更深层的内容。以前写程序时，main 函数中基本不会传入什么参数，在本次实验中，我体会到原理 main 函数中所传入的参数有

这么大的作用。

建议：希望以后上机课之前，老师能提前说明实验课需要做什么，比如要用到哪些知识点，以及书上哪些程序对本次实验是比较有帮助的，我们可以通过看书本上的例程去更好的实现实验要求。

五、 完成人姓名及完成时间

完成人姓名	完成时间
姚怀聿	2022 年 10 月 8 日