实验二：同步与异步write的效率比较

* 实验目的

掌握unix的文件I/O系统调用。

* 实验要求

1、编写程序

timewrite <outfile> [sync]

不得变更程序的名字和使用方法。sync参数为可选，若有，则输出文件用O\_SYNC打开。

例：

timewrite <f1 f2 表示输出文件f2不用O\_SYNC 打开。

timewrite f1 sync <f2 表示输出文件f1用O\_SYNC 打开。

2、显示的时间应当尽量接近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。

3、严格按p55表3-2的结果格式输出（BUFFSIZE从1024开始起算直至128K），抬头和分割线可以省略。

* 实验过程

1. 实验要求程序必须指定输出的文件名，而该文件是否按同步方式打开，则是可以选择的。因此程序至少带一个，至多两个输入参数。程序默认从标准输入STDIN\_FILENO读取输入文件，可以利用shell的输入定向功能选择具体的输入文件。

2. 系统调用times()的说明

#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 \*/

};

times函数的返回值是进程迄今为止的存活时间。所有时间都是以“滴答”为单位的，函数sysconf(\_SC\_CLK\_TCK)可获得所运行系统每秒的滴答数(参考课本P33)。

3、计算write耗费的时间

为了准确计算write耗费的时间，很重要的就是要避免将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。

设置输入缓冲区时需要知道输入文件的长度。除了使用系统调用stat外，更简单的方法是利用lseek的返回值来获取文件的长度。

在按每一个给定大小的输出缓冲区计算写文件时间时，应当在开始写之前调用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。

**注意，在开始按一个给定大小的输出缓冲区复制输入文件时，应当先将输出文件的写位置复位到输出文件的开头（这意味着不能以O\_APPEND方式打开输出文件）。可以使用lseek做到这一点，以避免多次打开、关闭输出文件**。

* 实验结果

本程序由：error.c, timewrite.c, apue.h, ourhdr.h这四个文件构成。

编译过程：

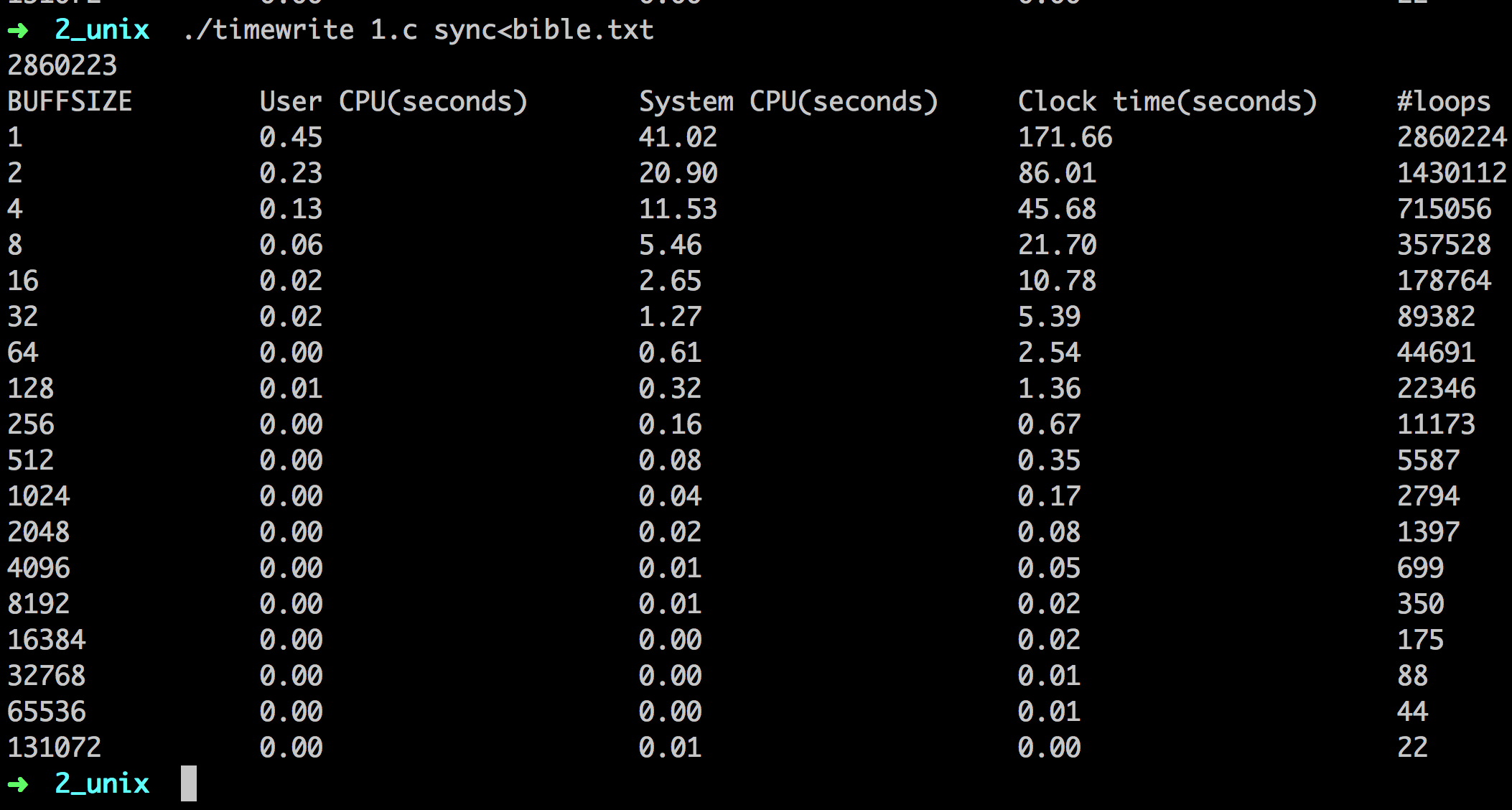
1 .执行语句：gcc -o timewrite error.c timewrite.进行编译;

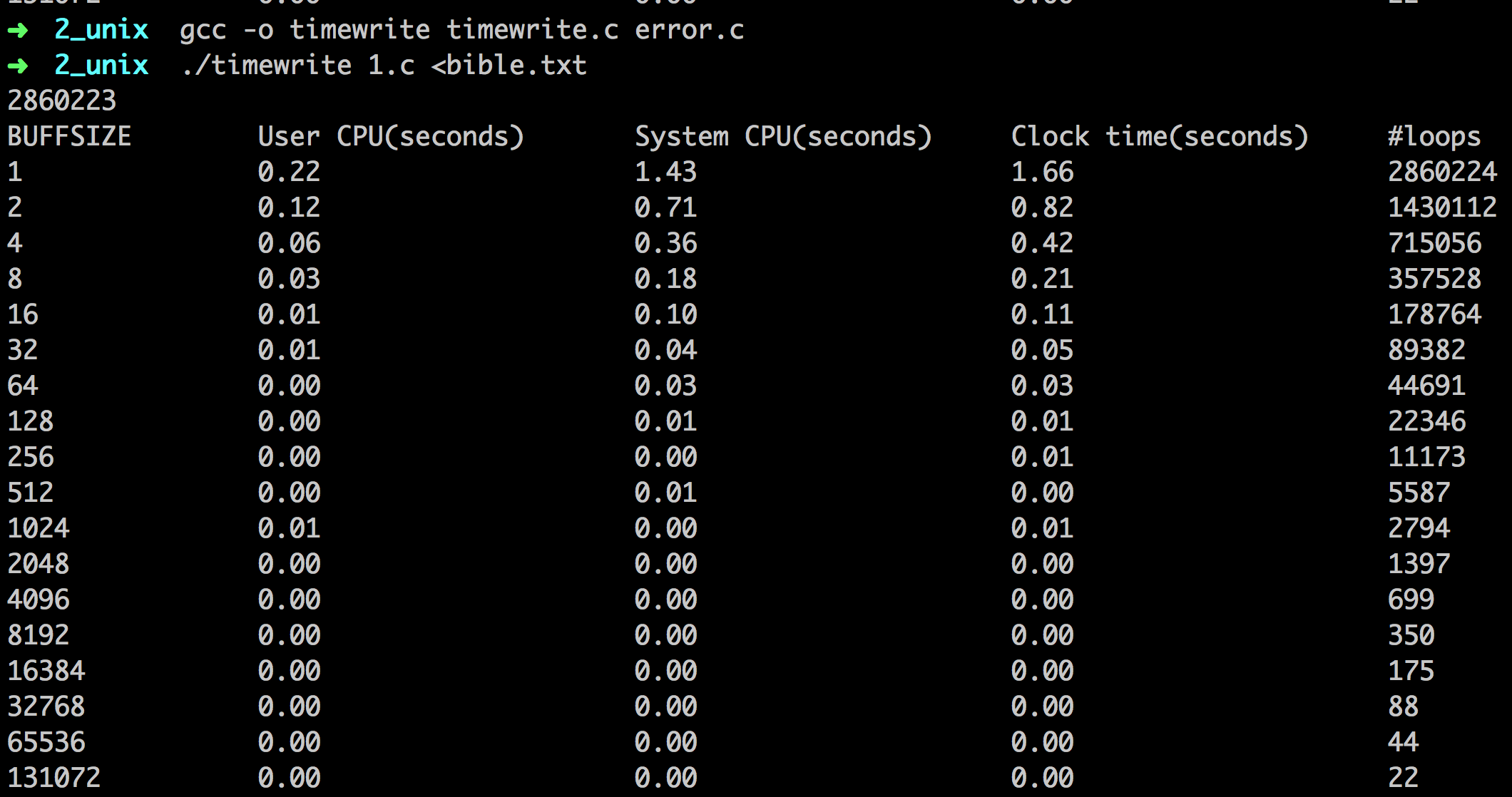
2. 编译通过后会生成一个名字为timewrite 的可执行程序；

3. 如执行timewrite f1 < f2, 进行异步wirte操作

4. 如执行timewrite f1 sync <f2， 进行同步write操作

5. 将两次结果进行比较如下图所示， 是将名字为bible.txt的文件，分别同步和异步写入文件1.c中的比较结果。





* 个人体会

通过自己编程比较同步与异步write的效率，对unix下的文件读写操作函数有了更熟练的掌握，通过对不同的缓冲区大小以及同步与异步写的效率比较，发现不同读写策略所耗费的时间存在很大的差距：缓冲区越大，写入效率越高，但随着缓冲区的增大，对写入效率的提高作用越小；同步写比异步写效率要高，缓冲区越小，两者差距越大，随着缓冲区的增大，两者的差距逐渐减小。实验结果提醒我们今后在写入数据操作时要兼顾效率与安全性的均衡，选择合适的写入策略。