

《操作系统》实验报告 (OS-Lab3)

中国人民大学 信息学院 崔冠宇 2018202147

一、实验题目： 线程通信.

二、实验目的:

- 加深对线程和多线程概念的理解;
- 掌握多线程程序设计的基本方法;
- 学习同一进程内线程间交换数据的方法.

三、实验方法:

在 Linux 兼容环境 (Mac 环境) 下, 按照实验要求, 编写 C 语言程序(详见末尾“程序清单”部分), 主要利用 POSIX 标准的线程相关函数, 实现相应线程间通信功能.

编写完毕后, 使用 gcc 进行编译(注意使用 -lpthread 参数), 运行可执行文件, 并观察程序输出. 如果和我们的预期相符合, 则该任务完成; 否则仔细分析原因, 查找资料, 修改程序, 直至正确为止.

四、程序结构

本次实验的要求相对简单, 除了 Linux 系统的默认数据结构(如 pthread_t, pthread_mutex_t 等)之外, 没有用到其他的数据结构, 所以下面主要介绍我的解决思路, 具体代码请参见末尾的“程序清单”部分.

本实验我的思路是:

- 主线程首先初始化互斥量及数个条件变量, 然后创建 Input, Display 和 Dispatch 三个线程, 等待它们结束后释放资源;
- Input 线程首先锁住缓冲区(m_buf), 等待用户输入内容后解锁, 然后通过条件变量(c_full)来通知 Display 线程;
- Display 线程提示用户输入数据, 等待 Input 线程使条件变量(c_full)成立, 输出“写文件中”, 再使用条件变量(c_write)通知 Dispatch 线程开始处理用户输入并写文件, 同时等待 Dispatch 线程完成任务后的通知(c_writedone), 如果此时全局变量 drop==1(表示有丢弃数据), 则输出提示, 最后输出“完成”;
- Dispatch 线程等待 Display 线程使条件变量(c_write)成立后, 按照要求处理数据, 如果有非数字非字母的字符则修改全局变量 drop=1, 然后通知 Display 线程操作完毕(c_writedone).

程序的大致流程图如下:

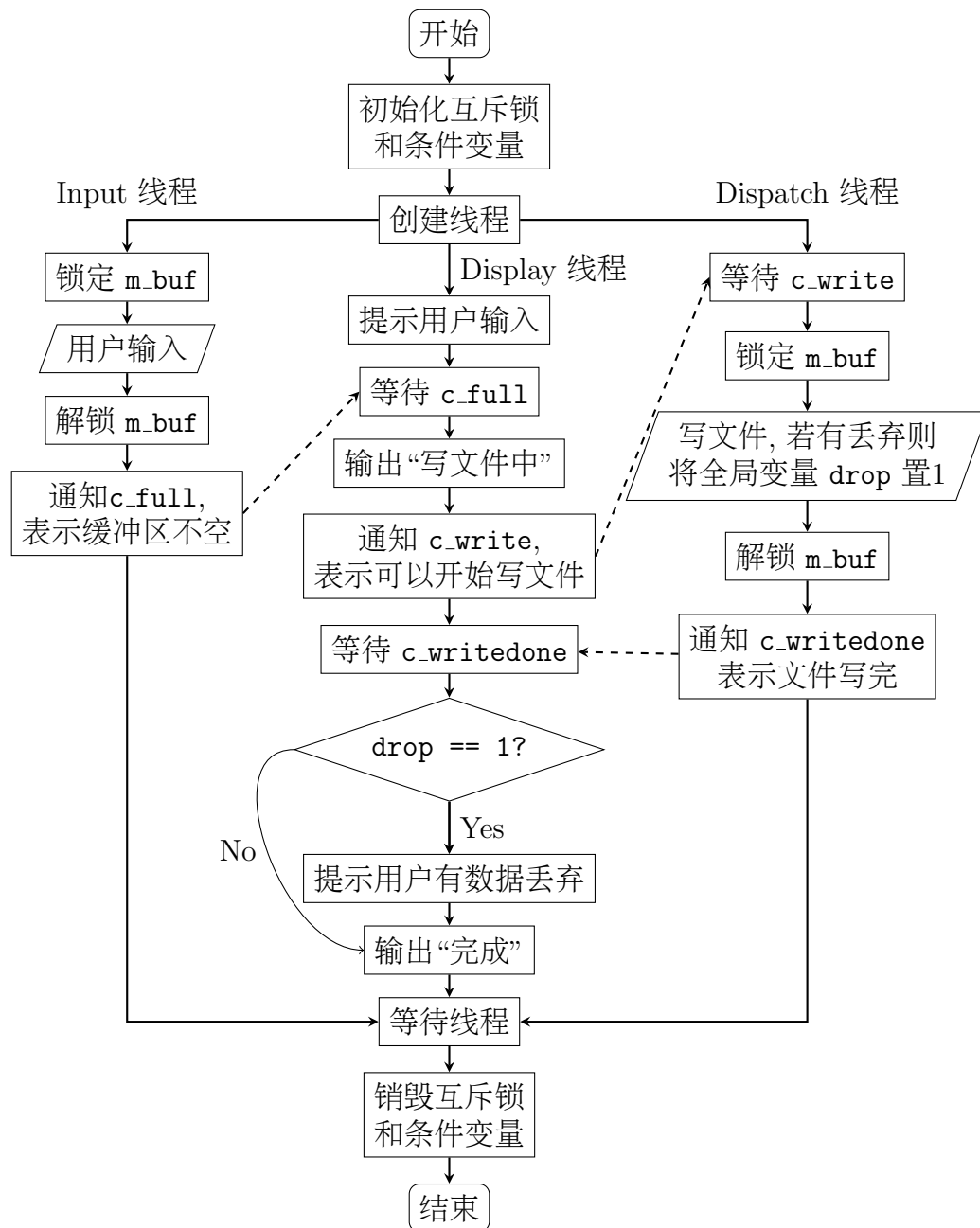


Figure 1: 主要步骤流程图.

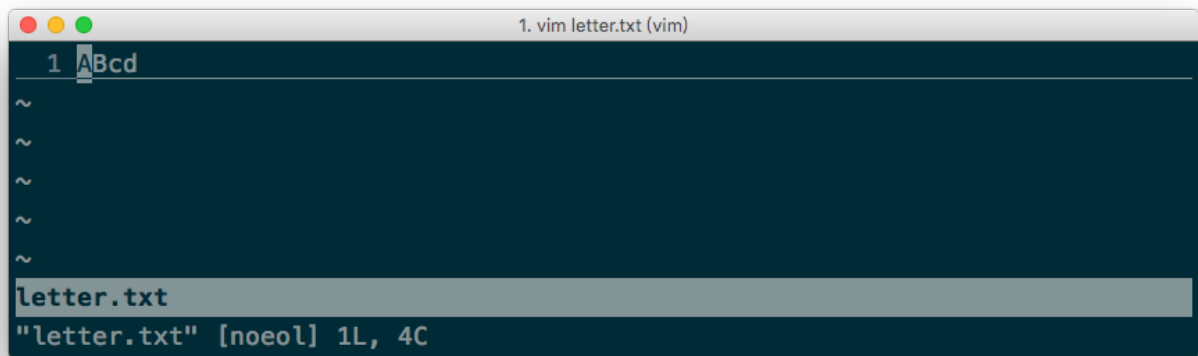
五、实验结果

编译运行程序, 输入下列测试数据, 运行结果分别如下所示:

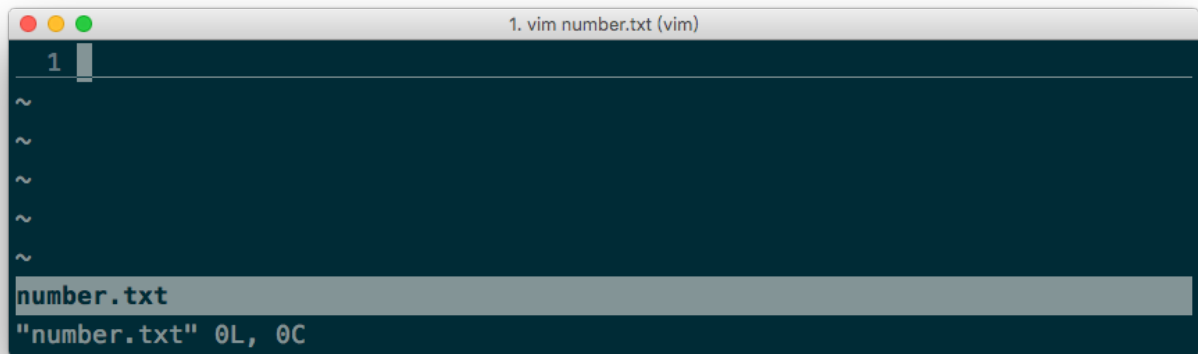
- 输入 ABcd (只含字母):



```
1. CuiGuanyu@Mac-mini: ~/Desktop/Lab3/Code (zsh)
CuiGuanyu@Mac-mini > ~/Desktop/Lab3/Code gcc -o lab3 lab3.c -lpthread
CuiGuanyu@Mac-mini > ~/Desktop/Lab3/Code ./lab3
You've logged in successfully!
Please enter text:
ABcd
Writing files.....
Done!
CuiGuanyu@Mac-mini > ~/Desktop/Lab3/Code
```



```
1. vim letter.txt (vim)
1 ABcd
~
~
~
~
~
letter.txt
"letter.txt" [noeol] 1L, 4C
```

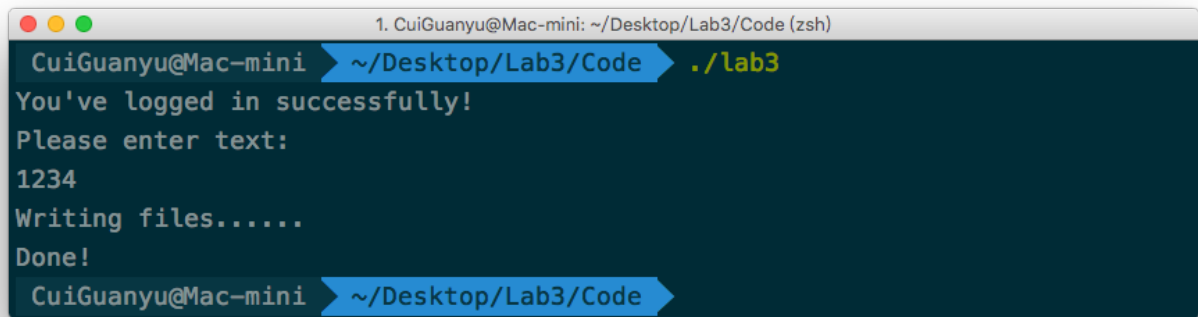


```
1. vim number.txt (vim)
1
~
~
~
~
~
number.txt
"number.txt" 0L, 0C
```

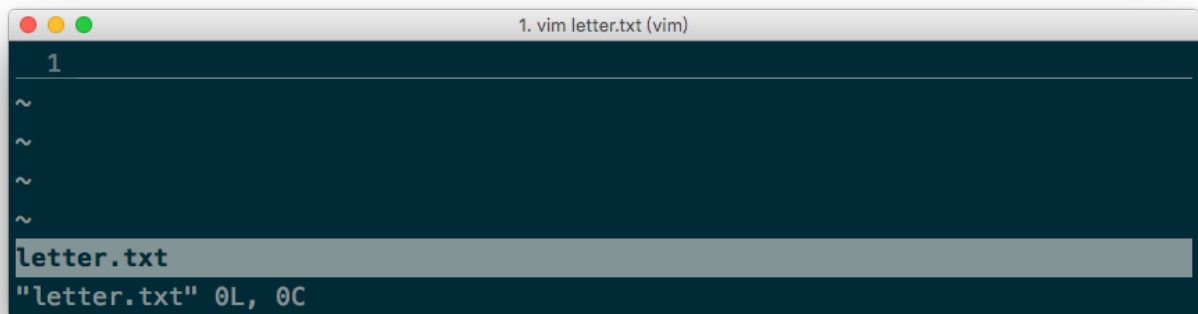
Figure 2: 运行结果.

可见程序运行正确, letter.txt 文件内有“ABcd”, 而 number.txt 则是空文件, 符合我们的预期.

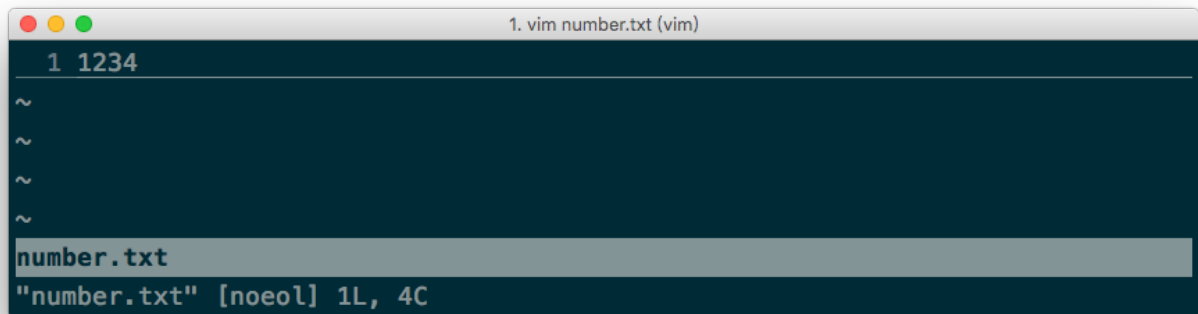
- 输入 1234 (只含数字):



```
1. CuiGuanyu@Mac-mini: ~/Desktop/Lab3/Code (zsh)
CuiGuanyu@Mac-mini > ~/Desktop/Lab3/Code > ./lab3
You've logged in successfully!
Please enter text:
1234
Writing files.....
Done!
CuiGuanyu@Mac-mini > ~/Desktop/Lab3/Code >
```



```
1. vim letter.txt (vim)
1
~
~
~
~
~
letter.txt
"letter.txt" 0L, 0C
```

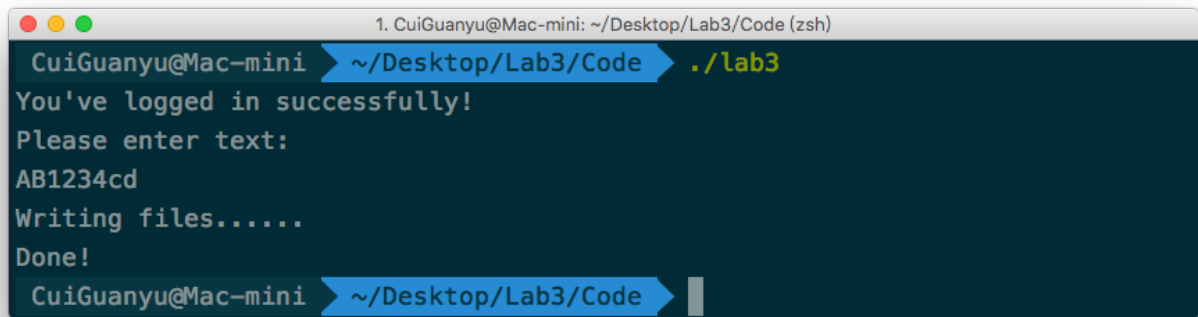


```
1. vim number.txt (vim)
1 1234
~
~
~
~
~
number.txt
"number.txt" [noeol] 1L, 4C
```

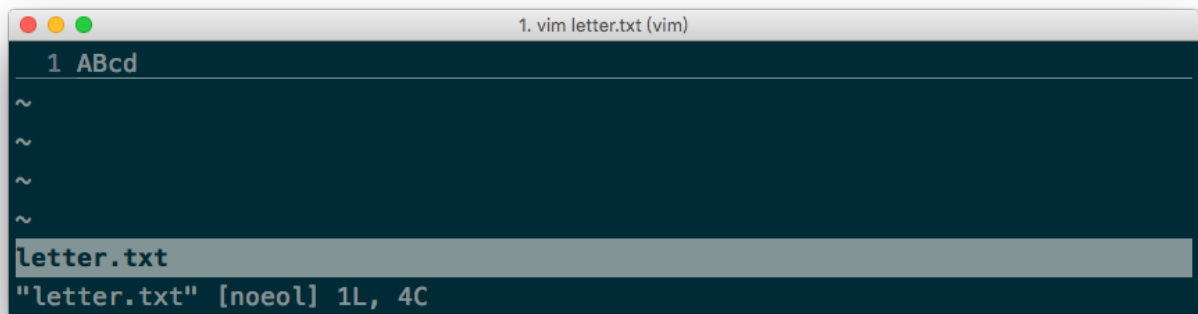
Figure 3: 运行结果.

可见程序运行正确, letter.txt 是空文件, 而 number.txt 内容为“1234”, 符合我们的预期.

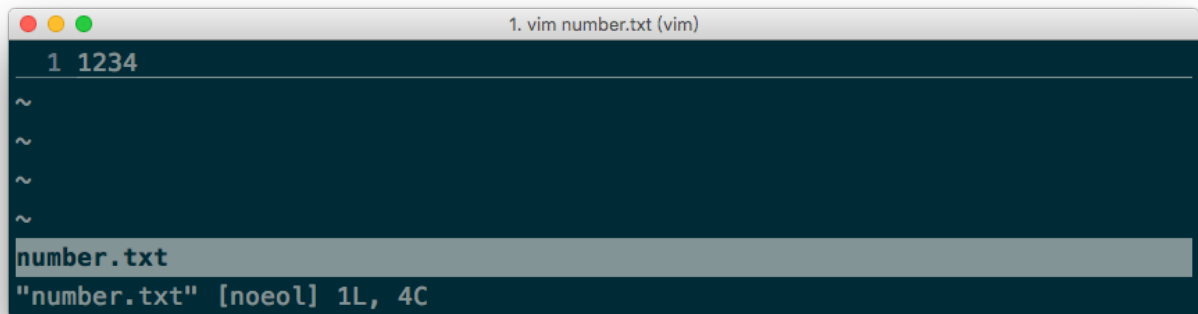
- 输入 AB1234cd (含字母及数字):



```
1. CuiGuanyu@Mac-mini: ~/Desktop/Lab3/Code (zsh)
CuiGuanyu@Mac-mini > ~/Desktop/Lab3/Code > ./lab3
You've logged in successfully!
Please enter text:
AB1234cd
Writing files.....
Done!
CuiGuanyu@Mac-mini > ~/Desktop/Lab3/Code > |
```



```
1. vim letter.txt (vim)
1 ABcd
~
~
~
~
letter.txt
"letter.txt" [noeol] 1L, 4C
```

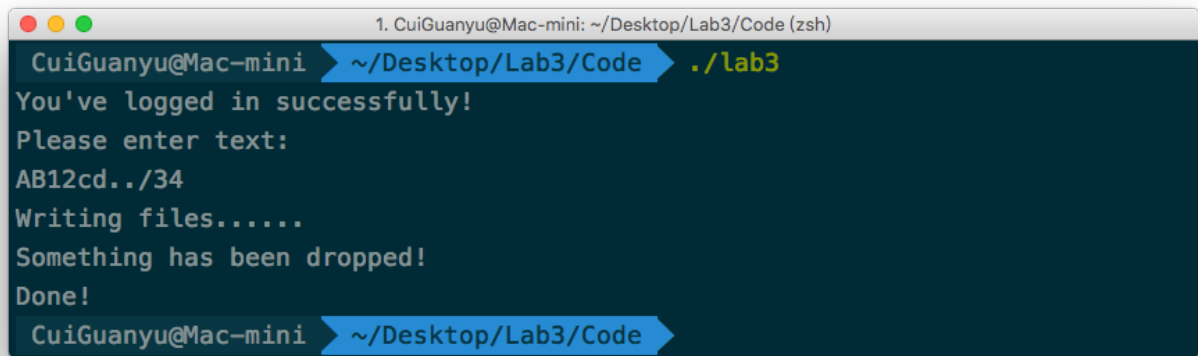


```
1. vim number.txt (vim)
1 1234
~
~
~
~
number.txt
"number.txt" [noeol] 1L, 4C
```

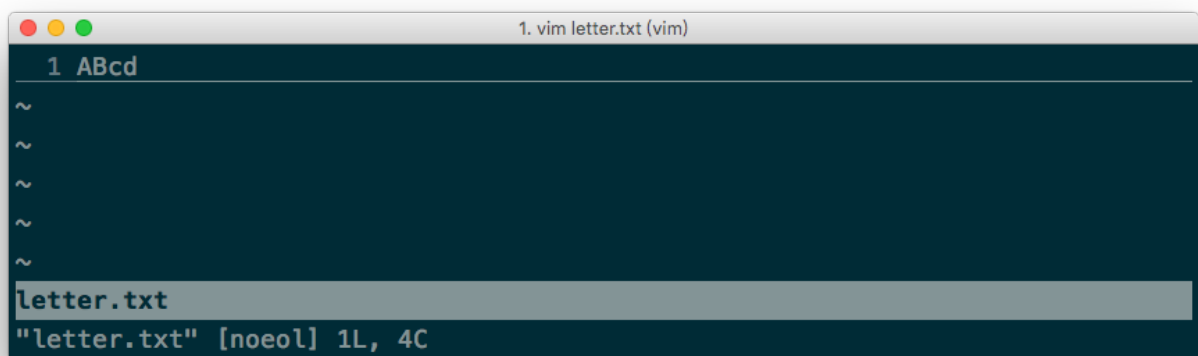
Figure 4: 运行结果.

可见程序运行正确, letter.txt 内容为“ABcd”, 而 number.txt 内容为“1234”, 符合我们的预期.

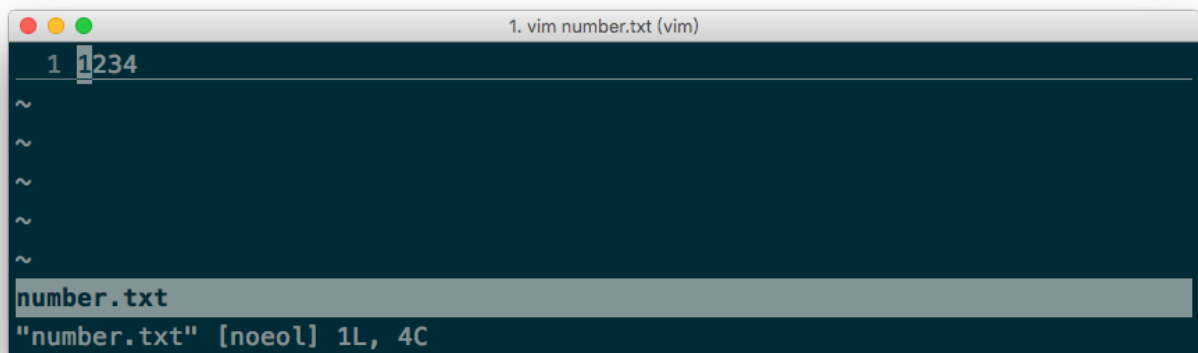
- 输入 AB12cd../34 (含字母、数字及其它符号):



```
CuiGuanyu@Mac-mini: ~/Desktop/Lab3/Code (zsh)
CuiGuanyu@Mac-mini: ~/Desktop/Lab3/Code ./lab3
You've logged in successfully!
Please enter text:
AB12cd../34
Writing files.....
Something has been dropped!
Done!
CuiGuanyu@Mac-mini: ~/Desktop/Lab3/Code
```



```
1 ABcd
~
~
~
~
~
letter.txt
"letter.txt" [noeol] 1L, 4C
```



```
1 1234
~
~
~
~
~
number.txt
"number.txt" [noeol] 1L, 4C
```

Figure 5: 运行结果.

可见程序运行正确, 输出了提示有数据丢弃的信息, letter.txt 内容为“ABcd”, 而 number.txt 内容为“1234”, 符合我们的预期.

六、问题分析

思考:比较线程间通信方式与进程间通信方式特点?

多线程环境中, 由于各个线程处于同一个进程空间内, “隔离”没有进程之间的那么明显, 所以线程间通信相对简单. 比如线程间通信可以使用全局变量(但要注意互斥)、互斥锁、条件变量、信号与信号量等进行通信. 而进程间不能使用全局变量通信, 但是相比线程通信而言, 还可以使用管道等方式通信.

实验中出现的問題及解决方法:

本实验较为简单, 故实验过程中我没有遇到问题. 但是实验完成后, 仔细审视代码, 我有以下的一些疑惑:

1. 本次实验一开始我全部使用了互斥锁, 发现也能实现相应功能, 后来在老师的提示之下尝试使用了条件变量替代. 似乎互斥锁一定程度上也能代替信号量等实现一些同步功能, 那么二者在使用时, 到底有什么共性与区别呢? 什么情况下使用互斥锁, 什么条件下使用信号量呢?
2. 我还曾试图使用信号量替代用于实现同步的互斥锁, 却发现了一个有意思的事. 在 macOS 中无名信号量相关的函数已经被标记为弃用(deprecated), 导致不能创建无名信号量, 这是处于何种考虑呢? 是否是出于安全机制?

七、程序清单

代码也可以在随附的 Code 目录中查看.

1. 代码 —— **lab3.c**:

```
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <signal.h>
4 #include <unistd.h>
5 #include <string.h>
6 #include <ctype.h>
7
8 // Buffer
9 #define N 2048
10 char inbuf[N] = {0};
11 // Thread ID
12 pthread_t t_input, t_display, t_dispatch;
13 // Mutexes for buffer and cv.
14 pthread_mutex_t m_buf, m_cond;
15 // Cond variables for input, write and writedone.
16 pthread_cond_t c_full, c_write, c_writedone;
```

```

17
18 int drop = 0;
19
20 // Input thread.
21 void * thread_input()
22 {
23     // So it will start here.
24     // Lock the buffer.
25     pthread_mutex_lock(&m_buf);
26     // Get input.
27     scanf("%s", inbuf);
28     // Unlock buffer.
29     pthread_mutex_unlock(&m_buf);
30     // Unlock empty mutex for display thread.
31     pthread_cond_signal(&c_full);
32     // Done.
33     return NULL;
34 }
35 // Display thread.
36 void * thread_display()
37 {
38     // Print something...
39     puts("You've logged in successfully!");
40     puts("Please enter text:");
41     // Wait for input thread.
42     pthread_mutex_lock(&m_cond);
43     pthread_cond_wait(&c_full, &m_cond);
44     pthread_mutex_unlock(&m_cond);
45     puts("Writing files.....");
46     // Unlock write cv for disptach thread.
47     pthread_cond_signal(&c_write);
48     // Wait for dispatch thread.
49     pthread_mutex_lock(&m_cond);
50     pthread_cond_wait(&c_writedone, &m_cond);
51     pthread_mutex_unlock(&m_cond);
52     // If something's dropped.

```



```

53     if(drop)
54     {
55         puts("Something has been dropped!");
56     }
57     // Done.
58     puts("Done!");
59     return NULL;
60 }
61
62 // Dispatch thread
63 void * thread_dispatch()
64 {
65     // Buffer for writing file.
66     char charbuf[N] = {0};
67     char numbuf[N] = {0};
68     // Wait for display thread.
69     pthread_mutex_lock(&m_cond);
70     pthread_cond_wait(&c_write, &m_cond);
71     pthread_mutex_unlock(&m_cond);
72     // Lock buffer.
73     pthread_mutex_lock(&m_buf);
74     // Handle input.
75     int len = strlen(inbuf);
76     int charlen = 0;
77     int numlen = 0;
78     for(int i = 0; i < len; i++)
79     {
80         if(isdigit(inbuf[i]))
81         {
82             numbuf[numlen] = inbuf[i];
83             numlen++;
84         }
85         else if(isalpha(inbuf[i]))
86         {
87             charbuf[charlen] = inbuf[i];
88             charlen++;

```

```

89         }
90         else
91         {
92             drop = 1;
93         }
94     }
95     // Unlock.
96     pthread_mutex_unlock(&m_buf);
97     // Write file.
98     FILE * fChars = fopen("letter.txt", "w");
99     FILE * fNums = fopen("number.txt", "w");
100     fprintf(fChars, "%s", charbuf);
101     fprintf(fNums, "%s", numbuf);
102     fclose(fChars);
103     fclose(fNums);
104     // Unlock for display thread.
105     pthread_cond_signal(&c_writedone);
106     return NULL;
107 }
108
109 int main(int argc, char *argv[])
110 {
111     // Init mutex.
112     pthread_mutex_init(&m_buf, NULL);
113     pthread_mutex_init(&m_cond, NULL);
114     // Init cv.
115     pthread_cond_init(&c_full, NULL);
116     pthread_cond_init(&c_write, NULL);
117     pthread_cond_init(&c_writedone, NULL);
118     // Create 3 threads.
119     pthread_create(&t_input, NULL, thread_input, NULL);
120     pthread_create(&t_display, NULL, thread_display, NULL);
121     pthread_create(&t_dispatch, NULL, thread_dispatch, NULL);
122     // Wait for threads.
123     pthread_join(t_input, NULL);
124     pthread_join(t_display, NULL);

```

```
125     pthread_join(t_dispatch, NULL);
126     // Destroy mutexes.
127     pthread_mutex_destroy(&m_buf);
128     pthread_mutex_destroy(&m_cond);
129     // Destroy cv.
130     pthread_cond_destroy(&c_full);
131     pthread_cond_destroy(&c_write);
132     pthread_cond_destroy(&c_writedone);
133     return 0;
134 }
```