

《操作系统》实验报告 (OS-Lab4)

中国人民大学 信息学院 崔冠宇 2018202147

一、实验题目： 进程通信.

二、实验目的:

- 掌握进程间通讯的编程方法;
- 加深对进程并发执行的理解.

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), 主要利用命名信号量、共享内存以及消息队列相关函数, 实现相应进程间通信功能.

编写完毕后, 使用 gcc 进行编译, 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

本次实验的要求与之前的实验相比略复杂, 除了 Linux 系统的默认数据结构(如 sem_t, shm, msg 等)之外, 还用到了以下的数据结构:

- 消息中表示进程状态的枚举:

```
1 typedef enum _msg_content
2 {
3     // User entered quit.
4     QUIT,
5     // Default message.
6     DEFAULT,
7     // Something's dropped.
8     DROP,
9 }msg_content;
```

- 消息结构体:

```
1 typedef struct _my_msg
```

```
2 {  
3     long msg_type;  
4     msg_content content;  
5     char msg_text[0];  
6 }my_msg;
```

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:

- 主进程首先初始化两个信号量, 创建一个共享内存和一个消息队列, 然后创建 Input, Dispatch 和 Display 三个进程, 等待它们结束后释放上述资源;
- Input 进程首先将共享内存链接到进程空间中, 等待用户输入内容后将字符串写入共享内存, 然后通过释放信号量(`sem_full`)来通知 Display 进程. 若用户输入的是“quit”, 则将共享内存断开, 进程退出;
- Dispatch 进程首先将共享内存链接至进程空间中, 等待 Input 进程释放信号量(`sem_full`)后, 从共享内存中拷贝数据. 如果数据是“quit”, 则向 Display 进程发送消息, 通知其退出, 然后断开共享内存自己退出; 否则就按照要求处理数据, 处理完毕后向 Display 进程发送消息, 通知其是否有非数字非字母的字符, 然后通过释放信号量(`sem_writedone`)通知 Display 进程操作完毕.
- Display 进程提示用户输入数据, 等待 Dispatch 进程释放信号量(`sem_writedone`)后, 从消息队列中接收一次信息, 如果信息枚举类型是“DROP”, 则输出字符串通知用户有内容被丢弃; 若信息枚举类型是“QUIT”, 则输出退出信息并退出.

程序的大致流程图如下:

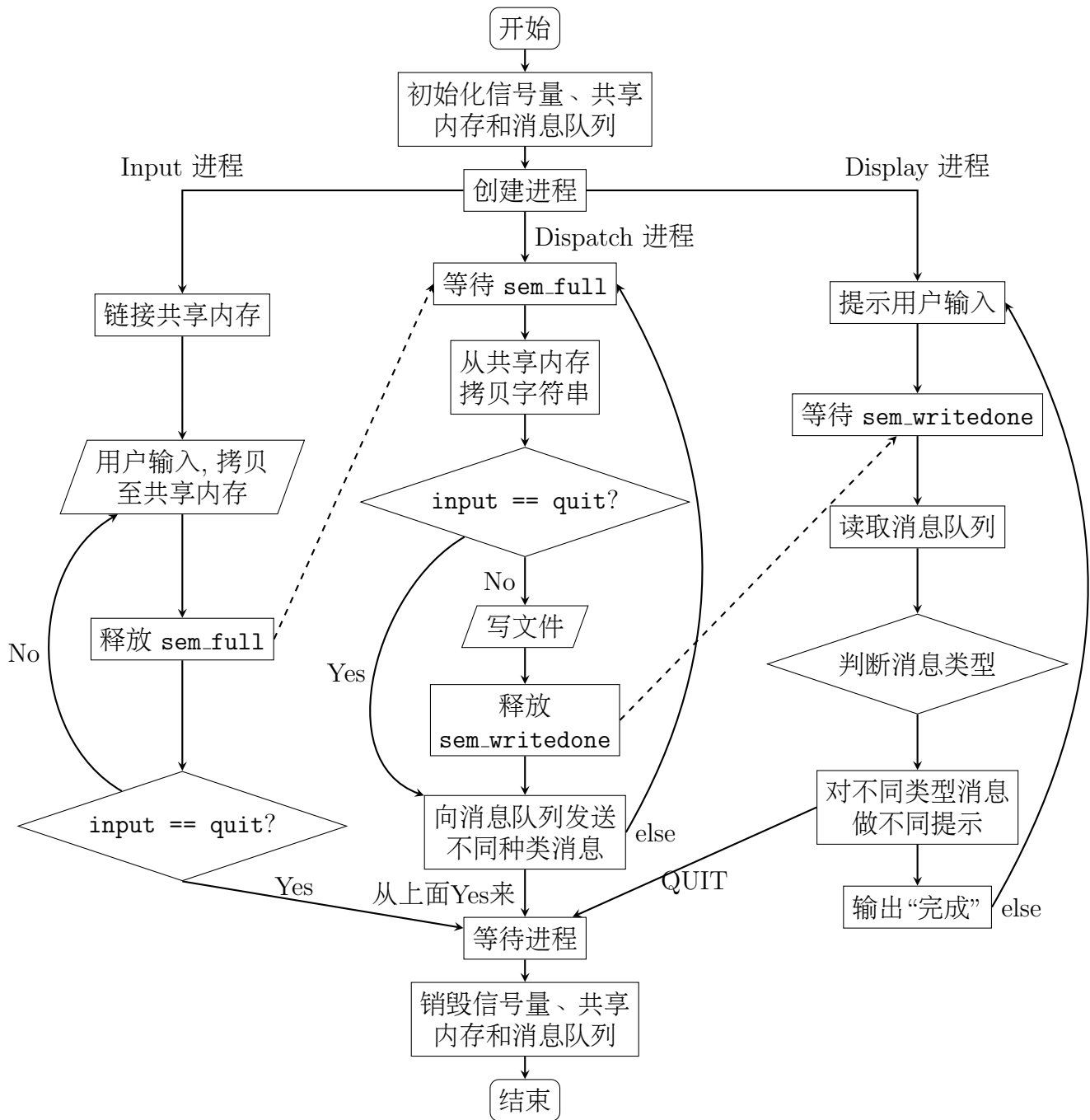
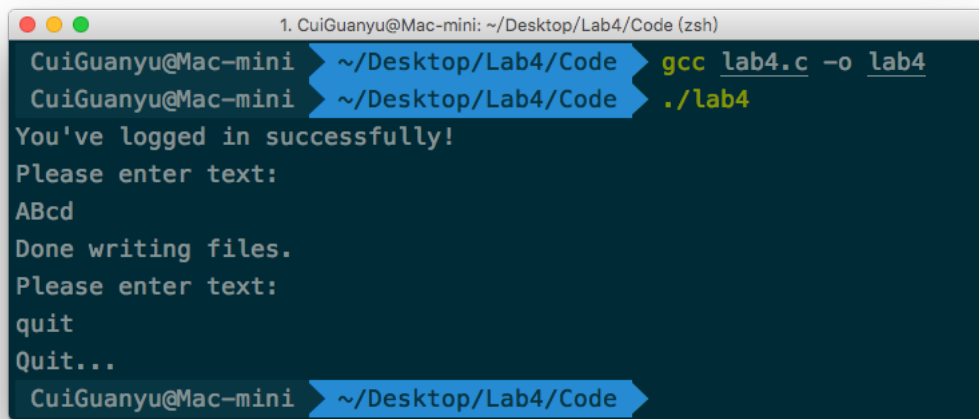


Figure 1: 主要步骤流程图.

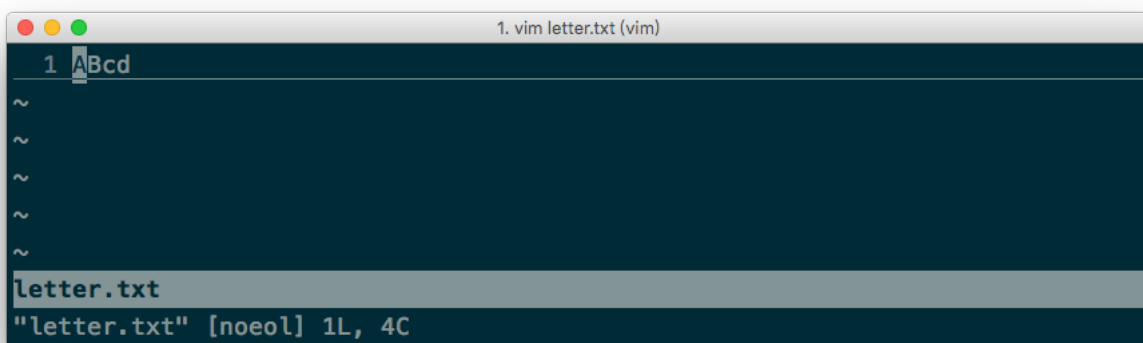
五、实验结果

编译运行程序, 输入下列测试数据, 运行结果分别如下所示:

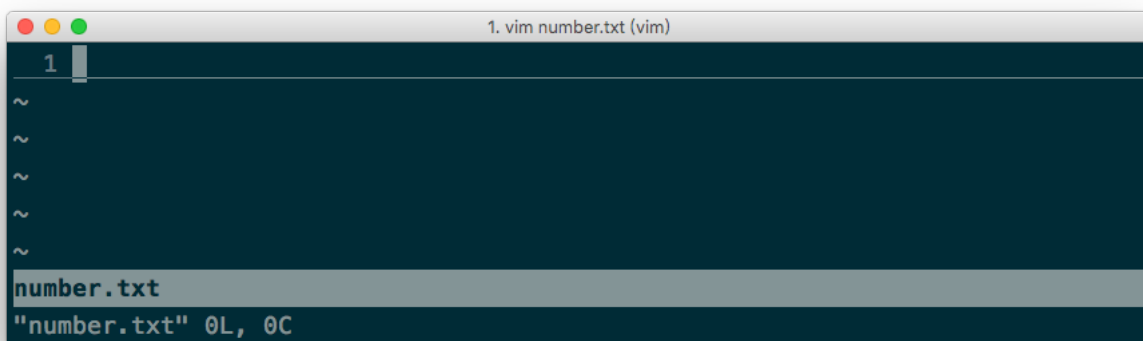
- 输入 ABcd (只含字母):



```
1. CuiGuanyu@Mac-mini: ~/Desktop/Lab4/Code (zsh)
CuiGuanyu@Mac-mini ~/Desktop/Lab4/Code gcc lab4.c -o lab4
CuiGuanyu@Mac-mini ~/Desktop/Lab4/Code ./lab4
You've logged in successfully!
Please enter text:
ABcd
Done writing files.
Please enter text:
quit
Quit...
CuiGuanyu@Mac-mini ~/Desktop/Lab4/Code
```



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

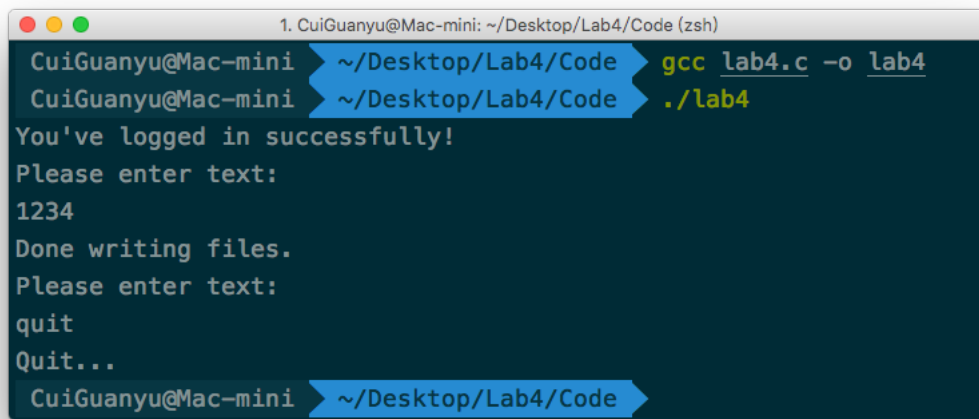


```
1. vim number.txt (vim)
1
~
~
~
~
~
number.txt
"number.txt" 0L, 0C
```

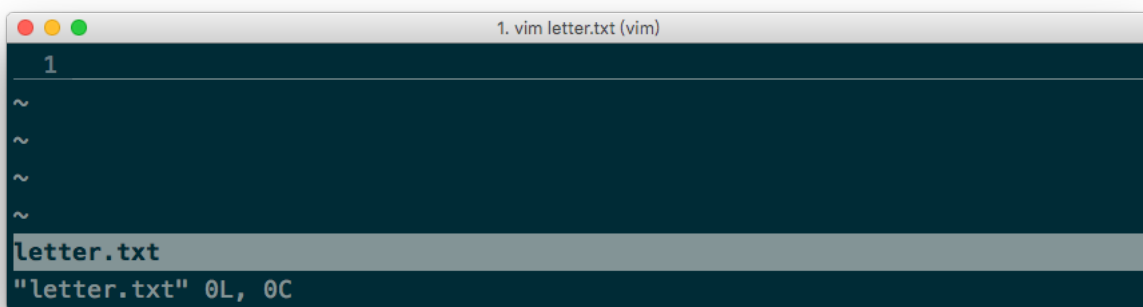
Figure 2: 运行结果.

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.

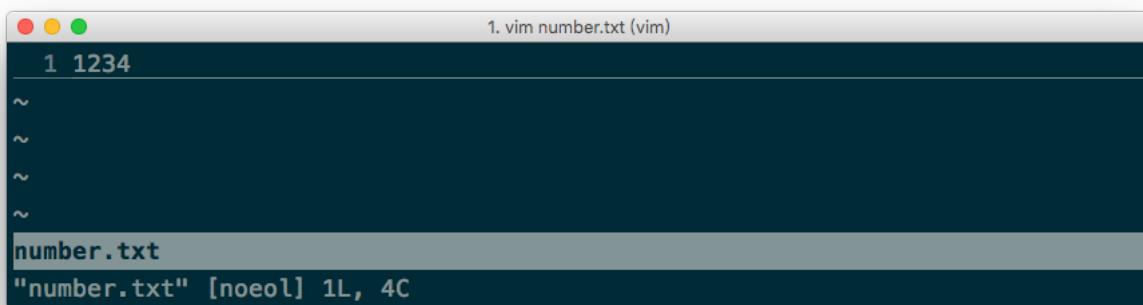
- 输入 1234 (只含数字):



```
1. CuiGuanyu@Mac-mini: ~/Desktop/Lab4/Code (zsh)
CuiGuanyu@Mac-mini ~/Desktop/Lab4/Code gcc lab4.c -o lab4
CuiGuanyu@Mac-mini ~/Desktop/Lab4/Code ./lab4
You've logged in successfully!
Please enter text:
1234
Done writing files.
Please enter text:
quit
Quit...
CuiGuanyu@Mac-mini ~/Desktop/Lab4/Code
```



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

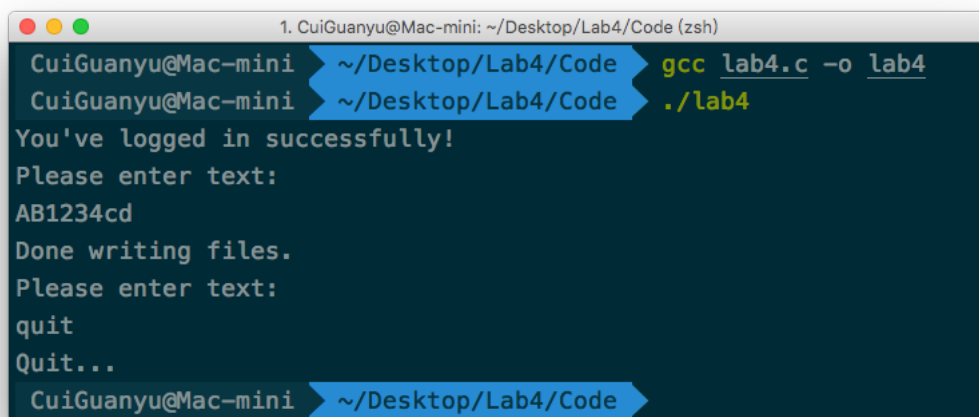


```
1. vim number.txt (vim)
1 1234
~
~
~
~
number.txt
"number.txt" [noeol] 1L, 4C
```

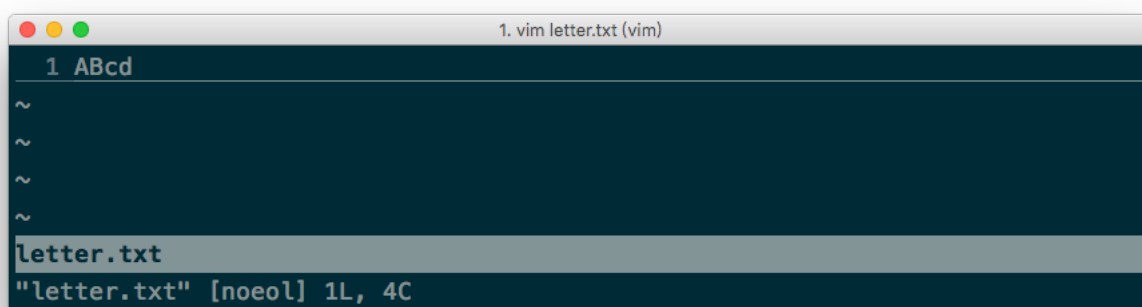
Figure 3: 运行结果.

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.

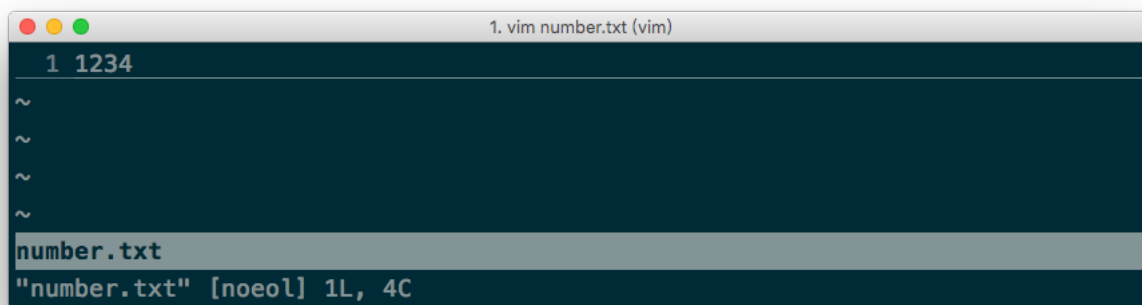
- 输入 AB1234cd (含字母及数字):



```
1. CuiGuanyu@Mac-mini: ~/Desktop/Lab4/Code (zsh)
CuiGuanyu@Mac-mini ~/Desktop/Lab4/Code gcc lab4.c -o lab4
CuiGuanyu@Mac-mini ~/Desktop/Lab4/Code ./lab4
You've logged in successfully!
Please enter text:
AB1234cd
Done writing files.
Please enter text:
quit
Quit...
CuiGuanyu@Mac-mini ~/Desktop/Lab4/Code
```



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

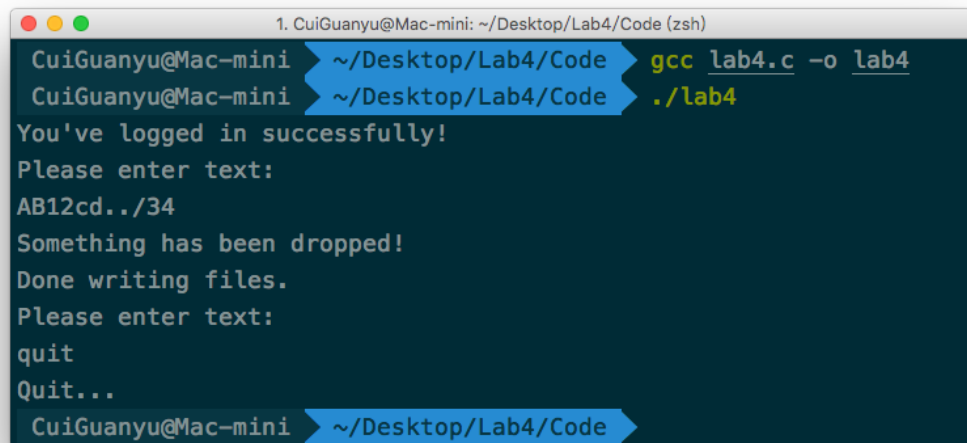


```
1. vim number.txt (vim)
1 1234
~
~
~
~
number.txt
"number.txt" [noeol] 1L, 4C
```

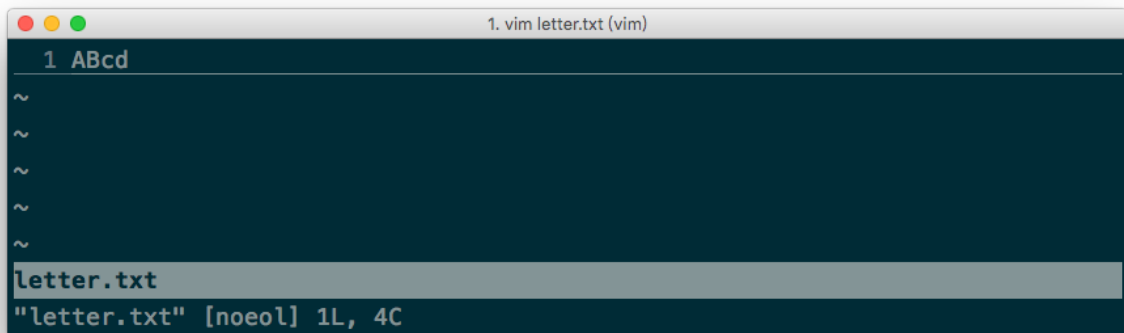
Figure 4: 运行结果.

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.

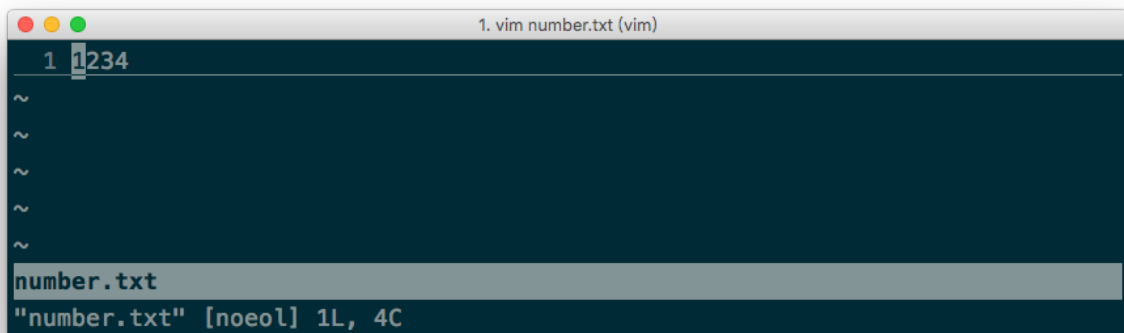
- 输入 AB12cd../34 (含字母、数字及其它符号):



```
1. CuiGuanyu@Mac-mini: ~/Desktop/Lab4/Code (zsh)
CuiGuanyu@Mac-mini ~/Desktop/Lab4/Code $ gcc lab4.c -o lab4
CuiGuanyu@Mac-mini ~/Desktop/Lab4/Code $ ./lab4
You've logged in successfully!
Please enter text:
AB12cd../34
Something has been dropped!
Done writing files.
Please enter text:
quit
Quit...
CuiGuanyu@Mac-mini ~/Desktop/Lab4/Code
```



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

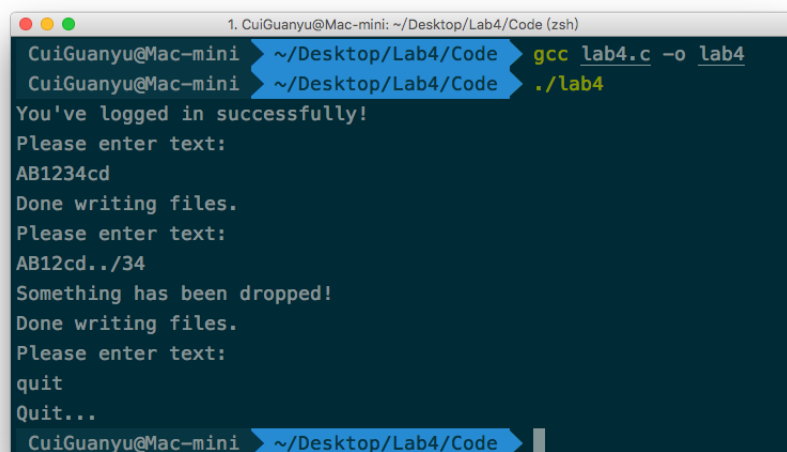


```
1. vim number.txt (vim)
1 1234
~
~
~
~
~
number.txt
"number.txt" [noeol] 1L, 4C
```

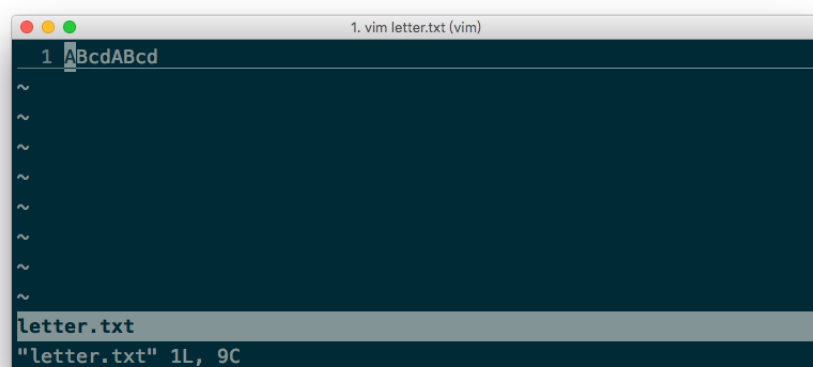
Figure 5: 运行结果.

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.

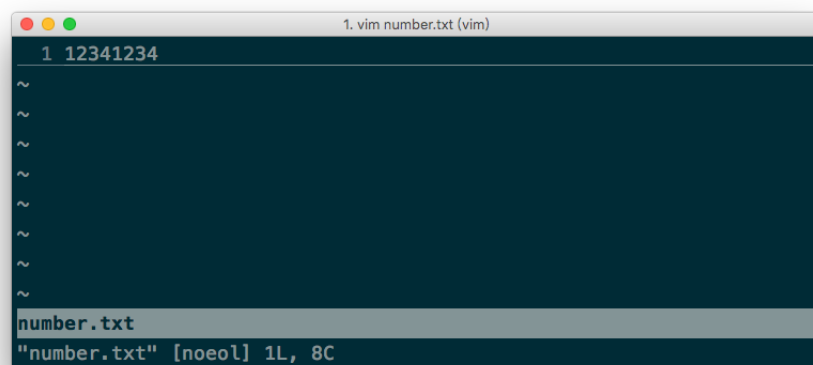
- 分两次输入 AB1234cd 和 AB12cd../34:



```
1. CuiGuanyu@Mac-mini: ~/Desktop/Lab4/Code (zsh)
CuiGuanyu@Mac-mini ~/Desktop/Lab4/Code gcc lab4.c -o lab4
CuiGuanyu@Mac-mini ~/Desktop/Lab4/Code ./lab4
You've logged in successfully!
Please enter text:
AB1234cd
Done writing files.
Please enter text:
AB12cd../34
Something has been dropped!
Done writing files.
Please enter text:
quit
Quit...
CuiGuanyu@Mac-mini ~/Desktop/Lab4/Code
```



```
1. vim letter.txt (vim)
1 ABcdABcd
~
~
~
~
~
~
~
~
~
~
letter.txt
"letter.txt" 1L, 9C
```



```
1. vim number.txt (vim)
1 12341234
~
~
~
~
~
~
~
~
~
~
number.txt
"number.txt" [noeol] 1L, 8C
```

Figure 6: 运行结果.

可见程序运行正确, letter.txt 内容为“ABcdABcd”, 而 number.txt 内容为“12341234”, 符合我们的预期.

六、问题分析

思考:比较共享存储和消息队列的利弊?

- 共享存储: 好处是可以直接传输数据, 可传输的数据量较大; 不足是系统不提供互斥访问, 需要用户自行设计, 比较麻烦.
- 消息队列: 好处是比较直观, 而且系统实现了互斥访问, 不需要用户手动编程; 弊端是传输数据时系统需要多次拷贝, 时间慢, 而且消息的大小相对固定, 不够灵活.

实验中遇到的问题及解决方法:

在实验过程中我遇到了与消息队列有关的问题, 最终都通过查找资料调试通过了.

1. 实验时, 我发现程序运行没有按照我的预期进行, 经过排查, 发现创建消息队列时有“Permission Denied”的错误出现. 经过查找资料, 发现是创建消息队列时权限设置不足, 调整后解决了问题.
2. 实验时, 在传输信息时程序再一次没有按照我的预期进行, 经过排查, 我发现发送消息时有“Invalid Argument”的错误出现, 经过查询才知道我将msg的type设置为0, 而系统要求消息类型应为正数, 修改后解决了问题.

七、程序清单

代码也可以在随附的 Code 目录中查看.

1. 代码 —— lab4.c:

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <stdlib.h>
4 #include <sys/msg.h>
5 #include <sys/shm.h>
6 #include <semaphore.h>
7 #include <string.h>
8 #include <ctype.h>
9
10 #define N 2048
11
12 // Shared memory id.
13 int shm_id;
```

```

14 // Message queue id.
15 int msg_id;
16 // Semaphores.
17 sem_t * sem_full, * sem_writedone;
18
19 typedef enum _msg_content
20 {
21     // User entered quit.
22     QUIT,
23     // Default message.
24     DEFAULT,
25     // Something's dropped.
26     DROP,
27 }msg_content;
28
29 typedef struct _my_msg
30 {
31     long msg_type;
32     msg_content content;
33     char msg_text[0];
34 }my_msg;
35
36 // Input process.
37 int proc_input()
38 {
39     // Link shared memory.
40     char * dest = (char *)shmat(shm_id, NULL, 0);
41     while(1)
42     {
43         char buf[N] = {0};
44         scanf("%s", buf);
45         strcpy(dest, buf);
46         sem_post(sem_full);
47         // If entered "quit".
48         if(strcmp(buf, "quit") == 0)
49         {

```

```

50         // Quit.
51         shmdt(dest);
52         return 0;
53     }
54 }
55 return 0;
56 }
57
58 // Dispatch process.
59 int proc_dispatch()
60 {
61     // Link shared memory.
62     char * src = (char *)shmat(shm_id, NULL, 0);
63     // Clear files.
64     FILE * fChars = fopen("letter.txt", "w");
65     FILE * fNums = fopen("number.txt", "w");
66     while(1)
67     {
68         // Buffer for writing file.
69         char inbuf[N] = {0};
70         char charbuf[N] = {0};
71         char numbuf[N] = {0};
72         // Waiting for input.
73         sem_wait(sem_full);
74         // Read in content.
75         strcpy(inbuf, src);
76         // If entered quit.
77         if(strcmp(inbuf, "quit") == 0)
78         {
79             // Send quit message to display.
80             my_msg msg = {1, QUIT};
81             msgsnd(msg_id, (void *)&msg, sizeof(msg.content), 0);
82             sem_post(sem_writedone);
83             shmdt(src);
84             return 0;
85         }

```

```

86     // Handle input.
87     int len = strlen(inbuf);
88     int charlen = 0;
89     int numlen = 0;
90     int drop = 0;
91     for(int i = 0; i < len; i++)
92     {
93         if(isdigit(inbuf[i]))
94         {
95             numbuf[numlen] = inbuf[i];
96             numlen++;
97         }
98         else if(isalpha(inbuf[i]))
99         {
100             charbuf[charlen] = inbuf[i];
101             charlen++;
102         }
103         else
104         {
105             // Drop something.
106             drop = 1;
107         }
108     }
109     // Write file.
110     fChars = fopen("letter.txt", "a");
111     fNums = fopen("number.txt", "a");
112     fprintf(fChars, "%s", charbuf);
113     fprintf(fNums, "%s", numbuf);
114     fclose(fChars);
115     fclose(fNums);
116     // Send message to display.
117     my_msg msg = {1, (drop ? DROP : DEFAULT)};
118     msgsnd(msg_id, (void *)&msg, sizeof(msg), 0);
119     sem_post(sem_writedone);
120 }
121 return 0;

```

```

122 }
123
124 // Display process.
125 int proc_display()
126 {
127     puts("You've logged in successfully!");
128     while(1)
129     {
130         puts("Please enter text:");
131         // Waiting for dispatch.
132         sem_wait(sem_writedone);
133         // Receive message.
134         my_msg msg;
135         msgrcv(msg_id, (void *)&msg, sizeof(msg), 01, 0);
136         // Drop something.
137         if(msg.content == DROP)
138         {
139             puts("Something has been dropped!");
140         }
141         else if(msg.content == QUIT)
142         {
143             puts("Quit...");
144             return 0;
145         }
146         puts("Done writing files.");
147     }
148     return 0;
149 }
150
151 int main(int argc, char *argv[])
152 {
153     // Open 3 semaphores.
154     sem_full = sem_open("sem_full", O_CREAT, 0777, 0);
155     sem_writedone = sem_open("sem_writedone", O_CREAT, 0777, 0);
156     // Create shared memeory.
157     key_t shm_key = ftok("./tmp/shm", 0666);

```

```

158     shm_id = shmget(shm_key, N, IPC_CREAT | 0666);
159     // Create message queue.
160     key_t msg_key = ftok("./tmp/msg_queue", 0666);
161     msg_id = msgget(msg_key, IPC_CREAT | 0666);
162
163     // Fork 3 processes.
164     pid_t input = fork();
165     if(input == 0)
166     {
167         proc_input();
168         return 0;
169     }
170     pid_t dispatch = fork();
171     if(dispatch == 0)
172     {
173         proc_dispatch();
174         return 0;
175     }
176     pid_t display = fork();
177     if(display == 0)
178     {
179         proc_display();
180         return 0;
181     }
182
183     // Wait child processes.
184     while(wait(NULL) != -1);
185     // Remove message queue.
186     msgctl(msg_id, IPC_RMID, NULL);
187     // Remove shared memory.
188     shmctl(shm_id, IPC_RMID, NULL);
189     // Close semaphores.
190     sem_close(sem_full);
191     sem_close(sem_writedone);
192     return 0;
193 }

```