

Lab3-线程通讯

张配天-2018202180

2020 年 5 月 28 日

目录

1 实验目的	1
2 实验思想和方法	1
3 程序结构及算法	2
3.1 锁的声明	2
3.2 锁的设计	2
4 实验结果	2
5 问题探究	3
5.1 死锁	3
5.2 段错误	3
6 思考	3
7 源代码	3

1 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

2 实验思想和方法

- 自行学习线程间通讯的锁机制
- 实践互斥锁、条件变量和信号量
- 由于线程和父进程共享资源的特点，使用共享内存而非管道避免二次拷贝和二次读取
- 用 ascii 码来判断输入的是字母还是数字

3 程序结构及算法

3.1 锁的声明

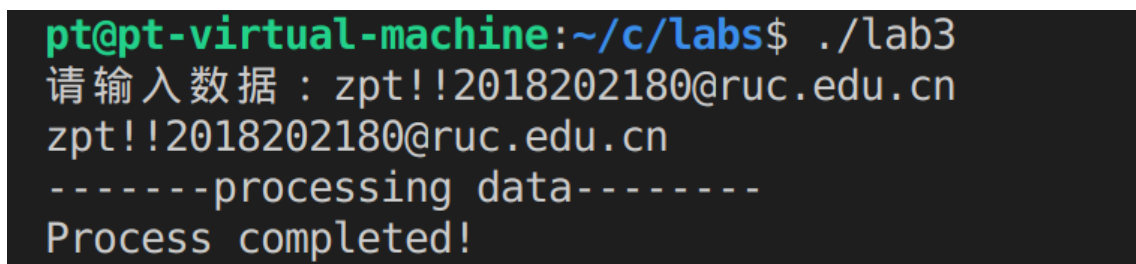
- 全局变量 `char buff_input[100] = ""`，作为共享的资源，存储输入字符串；
- 互斥锁 `mutex_buff`，实现输入和显示线程的互斥；
- 条件变量 `toDisplay`，实现输入后显示的同步；
- 信号量 `toProcess`，实现输入字符串后处理字符串的同步；
- 信号量 `toNotify`，实现处理好字符串后显示信息的同步；

3.2 锁的设计

- 输入和读取不能同时访问临界区资源 (即 `buff_input`)，因此使用互斥锁保证先写再读；
- 在输入完之后才能进行显示，因此在互斥锁内部使用条件变量实现同步，保证输入完成后再显示；
- 由于处理输入的线程和显示线程只有同步关系而非互斥关系，因此使用信号量完成同步 (条件变量必须在锁内部使用)，保证输入完成后再进行字符串处理；
- 字符串处理结束后唤醒显示线程输出信息，因此再次使用信号量完成同步。

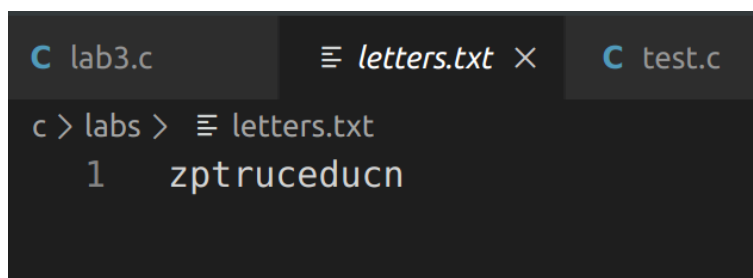
4 实验结果

运行结果如下：其中 `letters.txt,numbers.txt` 的结果如下：



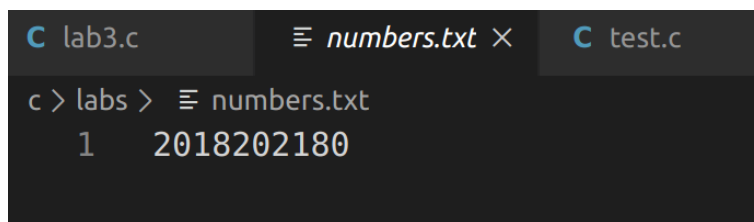
```
pt@pt-virtual-machine:~/c/labs$ ./lab3
请输入数据：zpt!!2018202180@ruc.edu.cn
zpt!!2018202180@ruc.edu.cn
-----processing data-----
Process completed!
```

图 1: 运行结果



```
lab3.c  letters.txt ×  test.c
c > labs > letters.txt
1      zptruceducn
```

图 2: letters.txt



```
C lab3.c  numbers.txt X  test.c
c > labs > numbers.txt
1 2018202180
```

图 3: numbers.txt

5 问题探究

5.1 死锁

- 如果把 `pthread_cond_wait()` 放在互斥锁外执行则会引起死锁，因为该函数为了保证即使条件变量得到满足，仍然仅能有 1 个线程访问临界区，会在运行结束后自动将互斥锁闭合。
- `pthread_cond_signal()` 的位置便比较灵活

5.2 段错误

如果将互斥锁指针参数传为 `NULL`，则会引起段错误

6 思考

线程通讯和进程通讯的不同之处在于线程可以访问父进程的资源，使得通讯比进程更为方便。

7 源代码

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4 #include <string.h>
5 #include <semaphore.h>
6
7
8 char inputBuff[100] = "";
9 pthread_mutex_t mutex_buff;
10 pthread_cond_t toDisplay;//toProcess;
11 sem_t toProcess,toNotify;//用条件变量的时候必须满足其和output进程的互斥，但
    是process进程和output进程没有互斥关系
12 pthread_t tid1,tid2,tid3;
13
14 void * KCP_input(){
15     pthread_mutex_lock(&mutex_buff);
16     printf("请输入数据: ");
17     scanf("%s",inputBuff);
```

```
18     //pthread_cond_signal(&toDisplay);
19
20     pthread_mutex_unlock(&mutex_buff);
21     //条件变量的signal的位置要求不严格，因为其不涉及开锁再关锁的行为
22     pthread_cond_signal(&toDisplay);
23     sem_post(&toProcess);
24     //pthread_cond_signal(&toProcess);
25     return (void*)0;
26 }
27 void * DCP_output(){
28     //cond_wait必须放在互斥锁内部，否则由于其机制是收到相应条件变量后会再次
        把该锁锁上，会造成死锁
29     //pthread_cond_wait(&toDisplay,&mutex_buff);
30     pthread_mutex_lock(&mutex_buff);
31     pthread_cond_wait(&toDisplay,&mutex_buff);
32     printf("%s\n",inputBuff);
33     pthread_mutex_unlock(&mutex_buff);
34     printf("————processing data————\n");
35     sem_wait(&toNotify);
36     printf("Process completed!\n");
37     return (void*)0;
38 }
39 void * CCP_dispose(){
40     sem_wait(&toProcess);
41     FILE * fp_num = fopen("numbers.txt","w");
42     FILE * fp_let = fopen("letters.txt","w");
43     //pthread_mutex_lock(&mutex_buff);
44     //pthread_cond_wait(&toProcess,&mutex_buff);
45     for(int i = 0;i < strlen(inputBuff);i++){
46         int c = (int)inputBuff[i];
47         if(c<= 57 && c>=48){
48             fputc(inputBuff[i],fp_num);
49         }
50         else if(c<=122 && c>=65){
51             fputc(inputBuff[i],fp_let);
52         }
53     }
54     sem_post(&toNotify);
55     //关闭文件
56     fclose(fp_let);
57     fclose(fp_num);
58     return (void*)0;
59 }
```

```
60
61 int main(){
62
63     pthread_mutex_init(&mutex_buff, NULL);
64     //pthread_cond_init(&toProcess, NULL);
65     pthread_cond_init(&toDisplay, NULL);
66     sem_init(&toProcess, 1, 0); // 同步
67     sem_init(&toNotify, 1, 0); // 同步
68
69
70     //默认先创建的线程会先执行
71     pthread_create(&tid1, NULL, DCP_output, NULL);
72     pthread_create(&tid2, NULL, CCP_dispose, NULL);
73     pthread_create(&tid3, NULL, KCP_input, NULL);
74
75     pthread_join(tid1, NULL);
76     pthread_join(tid2, NULL);
77     pthread_join(tid3, NULL);
78     //等待所有线程完成任务后释放内存
79     pthread_cond_destroy(&toDisplay);
80     pthread_mutex_destroy(&mutex_buff);
81     sem_destroy(&toProcess);
82     sem_destroy(&toNotify);
83
84     return 0;
85 }
```