

Lab4-进程通讯

张配天-2018202180

2020 年 6 月 9 日

目录

1 实验目的	1
2 实验思想和方法	1
3 程序结构和算法	2
4 运行结果	2
5 问题分析	3
5.1 消息队列没有权限访问	3
5.2 段错误	3
5.3 消息队列删除	3
5.4 无法输出信息/无法写入信息到文件	3
5.5 比较消息队列和共享内存两种进程间通讯方式的利弊	3
6 源代码	3

1 实验目的

创建三个进程，分别为输入进程、处理进程及显示进程；用户在输入进程输入信息，处理进程会处理输入的信息，显示进程输出相应显示；三个进程之间通过不同方式进行通信，交换输入数据等等。

2 实验思想和方法

设置三个进程，分别为客户端进程 (*Client*)，处理器进程 (*Deposit*)，显示器进程 (*Display*)。

- 客户端进程接收键盘输入，并利用消息队列将输入信息传递给显示器进程，利用共享内存将输入信息传递给处理器进程。
- 处理器进程读取共享内存中的输入数据，将数字和字母分别写入不同的文件中，丢弃别的字符；之后利用共享内存给显示器进程发送信号，提醒其输出相应信息。
- 显示器进程读取消息队列中的消息，将其显示在屏幕上；之后读取共享内存接受信号，输出信息提醒用户其输入已经被处理。
- 在用户输入“quit”后关闭客户端进程、处理器进程和显示器进程。

3 程序结构和算法

```
lab4
├── lab4_pre.c  封装函数、声明结构体
├── lab4_client.c 接受输入，发送信息
├── lab4_process.c 读入数据，处理数据
└── lab4_display.c 读入信息，显示信息
```

- 创建 4 个.c 文件，lab4_pre.c 用来封装函数，剩余三个对应三个进程，含有主函数。
- 在 lab4_pre.c 中封装所有创建获得消息队列、删除消息队列、发送接收消息、创建获得共享内存、写入读取共享内存、删除共享内存的函数。
- 在 lab4_pre.c 中定义消息的结构体，包含两个属性：消息类型和消息内容；
- 在 lab4_pre.c 中定义共享内存的结构体，包含两个属性：是否被处理过的标识符和消息内容；
- 消息类型仅有一种，定义为 INPUT 宏 =1；客户端进程向消息队列发送消息，显示器进程收取消息并显示；
- 写入共享内存时自动将标志符置 0，读取时置 1，如果处理器进程读取到输入为 quit，则将标识符置 2；客户端进程一旦发现此次输入的标识符已被置 1，则输出提醒用户的信息；发现为 2，则删除共享内存。

4 运行结果

运行结果如图 1 所示，从左到右分别是客户端进程、处理器进程和显示器进程，可以发现，显示器进程正确地输出了所有输入的信息，并且提醒用户其输入数据已经被处理。

图 1: 运行结果

numbers.txt 和 letters.txt 如图 2 所示，可见程序正确地将数字和字母分批写入到文件中。

图 2: 数字文件及字母文件

5 问题分析

5.1 消息队列没有权限访问

创建好的消息队列无法发送信息，通过 `ipcs -p` 检查发现其权限为 0，发现原来在创建消息队列和共享内存时必须在 `IPC_CREATE` 后或一个权限，否则无法访问。

5.2 段错误

最开始在消息队列的结构体中使用字符串指针而非字符数组，这时会出现段错误。我认为消息队列必须进行“深拷贝”，修改为字符数组后正常。

5.3 消息队列删除

在代码中调用 `msgctl(msg_id,IPC_RMID,NULL)` 删除消息队列后仍需要手动删除，即在终端中运行 `ipcrm -q msg_id` 将其彻底删除，否则再次使用同样的 key 值创建消息队列时会报错。

5.4 无法输出信息/无法写入信息到文件

曾遇到无法使用 `printf("%s",content)` 输出信息，亦无法使用 `fputc(char)` 写入字符到文件的问题，经查阅资料发现在写并发程序时需要即时使用 `fflush(stdout)/fflush(fp)` 清空缓存区，之后方可正常工作。

5.5 比较消息队列和共享内存两种进程间通讯方式的利弊

- I. 相比于消息队列，共享内存不用进行多次的拷贝，只需要各个进程访问同一个内存区就行；
- II. 相比于消息队列，共享内存不用在终端进行手动删除，可以完全由代码实现；
- III. 相比于消息队列，共享内存可以更方便地做到双方的交互，不用再次进行发收消息的动作；
- IV. 相比于共享内存，消息队列实现了自然的读端互斥，仅有一个进程可以读取队列中的一个消息，读完后消息会自然删除；

综上，在本实验的环境下，我认为共享内存优于消息队列。

此外，可以进一步探究消息队列实现多个读取段，以及共享内存设置互斥的办法，另外可以考虑用信号实现处理消息的反馈，更加符合逻辑。

6 源代码

- lab4_pre.c

```
1 #include<sys/types.h>
2 #include<sys/ipc.h>
3 #include<sys/msg.h>
4 #include<sys/shm.h>
5 #include<stdio.h>
6 #include<string.h>
7 #include<stdlib.h>
8 #define INPUT 1
```

```
9
10 struct msgItem
11 {
12     long type;
13     char text[1024];
14 }msgItem = {0,{0}};
15
16 struct shmItem
17 {
18     int processed;
19     char text[1024];
20 }shmItem = {0,{0}};
21
22 int initMsgQueue(int status){
23     key_t key = ftok("./",0);
24     if(key < 0){
25         perror("failure in creating key");
26     }
27     int msg_id = msgget(key,status);
28     if(msg_id < 0){
29         perror("failure in creating message queue");
30     }
31     return msg_id;
32 }
33 int createMsgQueue(){
34     return initMsgQueue(IPC_CREAT|0666);
35 }
36 int getMsgQueue(){
37     return initMsgQueue(IPC_CREAT);
38 }
39 int destroyMsgQueue(int msg_id){
40     if(msgctl(msg_id,IPC_RMID,NULL) < 0){
41         perror("failure in destrpying message queue");
42     }
43     return 0;
44 }
45 int sendMsg(int msg_id,int type,char* message){
46     struct msgItem msgitem;
47     msgitem.type = type;
48     strcpy(msgitem.text,message);
49     if(msgsnd(msg_id,(void*)&msgitem,sizeof(msgitem.text),0) < 0){
50         perror("error in sending message");
51         return -1;
```

```
52     }
53     return 0;
54 }
55
56 int recvMsg(int msg_id,int type,char * out){
57     struct msgItem recvitem;
58     int size = sizeof(recvitem.text);
59     if(msgrcv(msg_id,(void*)&recvitem,size,type,0) < 0){
60         perror("error in receiving message");
61         return -1;
62     }
63
64     strcpy(out,recvitem.text);
65     return 0;
66 }
67
68 int initShm(int status){
69     key_t key = ftok("./",0);
70     if(key < 0){
71         perror("failure in creating key");
72     }
73     int msg_id = shmget(key,1024,status);
74     if(msg_id < 0){
75         perror("failure in creating shared memory");
76     }
77     //printf("%d",msg_id);
78     return msg_id;
79 }
80
81 struct shmItem * createShm(){
82     int shm_id = initShm(IPC_CREAT|0666);
83     return (struct shmItem * )shmat(shm_id,0,0);
84 }
85
86 struct shmItem * getShm(){
87     int shm_id = initShm(IPC_CREAT);
88     return (struct shmItem * )shmat(shm_id,0,0);
89 }
90
91 int destroyShm(int shm_id,struct shmItem * shareMem){
92     if(shmdt(shareMem) == -1){
93         perror("Error in Unbounding Shared Memory");
94     }
```

```

95     if(shmctl(shm_id, IPC_RMID, 0) == -1){
96         perror("Error in Destroying Shared Memory");
97         return -1;
98     }
99 };
100
101 int writeToShm(struct shmItem * shareMem, char * text){
102     shareMem->processed = 0;
103     strcat(shareMem->text, text);
104     return 0;
105 }
106
107 int readFromShm(struct shmItem * shareMem, char out[]){
108     shareMem->processed = 1;
109     strcpy(out, shareMem->text);
110     //puts(out);
111     memset(shareMem->text, 0, sizeof(out));
112     return 0;
113 }

```

- lab4_client.c

```

1  #include "lab4_pre.c"
2  int main(){
3      int msg_id = createMsgQueue();
4      struct shmItem * shared = createShm();
5      //printf("%p", shared);
6      printf("Process Client\n");
7      char buf[100] = {0};
8      while (1)
9      {
10         printf("type in message:\n");
11         gets(buf);
12         sendMsg(msg_id, (long)INPUT, buf);
13         writeToShm(shared, buf);
14         if(!strcmp("quit", buf)){
15             return 0;
16         }
17     }
18 }

```

- lsb4_process.c

```

1  #include "lab4_pre.c"
2  int main(){

```

```

3      struct shmItem * shared = getShm();
4
5      FILE * fp_num = fopen("numbers.txt", "w+");
6      FILE * fp_let = fopen("letters.txt", "w+");
7      printf("Process Deposit\n");
8      while (1)
9      {
10         if(shared->processed){
11             continue;
12         }
13         char out[100] = {};
14         readFromShm(shared, out);
15         if(!strcmp(out, "quit")){
16             shared->processed = 2;
17             return 0;
18         }
19
20         //printf("%s", out);
21         //fflush(stdout);
22
23
24         for(int i = 0; i < strlen(out); i++){
25             int c = (int)out[i];
26             //printf("%d", c);
27             if((c <= 57 && c >= 48) || c == 10){
28                 fputc(out[i], fp_num);
29                 fflush(fp_num);
30             }
31             else if((c <= 122 && c >= 65) || c == 10){
32                 fputc(out[i], fp_let);
33                 fflush(fp_let);
34             }
35         }
36         fputc('\n', fp_num);
37         fputc('\n', fp_let);
38         fflush(fp_num);
39         fflush(fp_let);
40     }
41 }

```

- lab4_display.c

```

1 #include "lab4_pre.c"
2 int main(){

```

```
3      int msg_id = getMsgQueue();
4      int shm_id = initShm(IPC_CREAT);
5      struct shmItem * shared = getShm();
6      printf("Process Display\n");
7      while (1)
8      {
9          char out[100] = {};
10         recvMsg(msg_id, (long) INPUT, out);
11         if (!strcmp("quit", out)) {
12             printf("message queue deleted (need to be deleted manually in\n\nshell)");
13             destroyMsgQueue(msg_id);
14             return 0;
15         }
16         printf("output:%s\n", out);
17         while (!shared->processed) {}
18         if (shared->processed == 1) {
19             printf("Inputting Data has been Processed\n");
20         }
21         else
22         {
23             printf("Shared Memory Destroyed");
24             destroyShm(shm_id, shared);
25             return 0;
26         }
27     }
28 }
```