04 共享内存

消息队列（MSG）

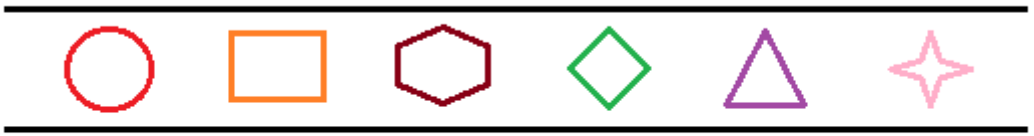
正文 B U A

一.概念

之前所学习的通信方式，它们都有一个弊端，无法在管道中读取到一个指定的数据.
因为这些数据没有做任何标记，读的进程只能按照次序挨个的读取，因此多进程之间的互相通信，
除非使用多条管道分别处理，否则无法使用一条管道来完成这样的通信。

PIPE或FIFO中的无标识消息数据

消息队列提供一种带有数据标识的特殊管道，使得每一段被写入的数据都变成带标识的消息，
读取该段消息的进程只要指定这个标识就可以正确地读取，而不会受到其他消息的干扰，
从运行效果来看，一个带标识的消息队列，就像多条并存的管道一样。



消息队列中的带标识的消息数据

在使用消息队列时，是可以选择性的接收消息队列中的信息，不一定要按照消息发送的先后顺序接收。可以根据消息标识符来读取信息。

那也就意味着，我们在发送数据或接收数据时，也是需要带上这个消息标识符。

- 相关Shell命令

```

查看消息队列 : ipcs -q
删除指定的消息队列 : ipcrm -q MSG_ID 或者 ipcrm -Q msg_key
```

二.相关API

- **申请消息队列**

```
================================================  msgget() 函数解析 ========================================
NAME
       msgget - get a System V message queue identifier

SYNOPSIS
```

```
       #include <sys/types.h>
       #include <sys/ipc.h>
       #include <sys/msg.h>

       int msgget(key_t key, int msgflg);

参数解析 :
       参数1 key_t key   : 消息队列的键值

       参数2 msgflg      :
              IPC_CREAT : 如果key对应的MSG不存在,则创建MSG.
              IPC_EXCL  : 如果key对应的MSG存在,则报错.
              mode      : MSG的访问权限(八进制权限,例如 0666).

返回值 :
        成 功 : 返回该消息队列的ID号.
        失 败 : 返回-1.

```

- **消息队列发送与接收**

```
NAME
       msgrcv, msgsnd - System V message queue operations

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

    int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
```

```
参数解析:
        参数1 int msqid        : 消息队列的ID
        参数2 const void *msgp : 存放需要发送的信息
               查看帮助手册, 手册告诉我们,消息队列在发送消息时,需要把消息打包成一个结构体进行发送
               struct msgbuf
               {
                   long mtype;       /* 消息标识符 */
                   char mtext[1];    /* 需要发送的数据 数据类型可自定义 */
               };

        参数3 size_t msgsz     : 发送数据的大小
        参数4 int msgflg       : 默认设置为0
 返回值 :
        成 功 : 返回  0
        失 败 : 返回 -1


    ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp,int msgflg);
参数解析:
        参数1 int msgsnd     : 消息队列的ID
        参数2 void *msgp     : 存放接收的信息
        参数3 size_t msgsz   : 接收数据的大小
        参数4 long msgtyp    : 消息标识符(重点)
        参数5 int msgflg     : 默认设置为0
返回值 :
        成 功 : 返回接收到的字节数
        失 败 : 返回 -1

```

- **使用示例**

```
1
2  ========================example1 写入数据到消息队列中.
3
4  #include <stdio.h>
5  #include <sys/types.h>
6  #include <sys/ipc.h>
7  #include <sys/msg.h>
8  #include <string.h>
9  #include <errno.h>
10
11 //自定义结构体,用于保存需要发送到消息队列的数据。
12 struct Send_msg
13 {
14     long msg_flag;                    //消息标识符
15     char msg_buf[50];         //消息的数据类型可根据自身的需求自定义。
16 };
17
18 int main(int argc, char const *argv[])
19 {
20     //申请消息队列
21     int msg_id = msgget(12343, IPC_CREAT | IPC_EXCL | 0777);
22     if (msg_id == -1)
23     {
24             if (errno == EEXIST)  //消息队列如果存在则直接打开
25             {
26                     msg_id = msgget(12343,0777);
27             }
```

```
28          else
29          {
30                  perror("打开消息队列失败\n");
31                  return -1;
32          }
33  }
34
35  //定义一个结构体变量存放要发送的数据
36  struct Send_msg data;
37  bzero(&data,sizeof(data)); //清空结构体
38
39  data.msg_flag = 111;
40  strcpy(data.msg_buf, "hello");
41
42  //定义一个结构体变量存放要发送的数据
43  struct Send_msg data1;
44  bzero(&data1,sizeof(data1)); //清空结构体
45
46  data1.msg_flag = 222;
47  strcpy(data1.msg_buf, "123");
48
49  //定义一个结构体变量存放要发送的数据
50  struct Send_msg data2;
51  bzero(&data2,sizeof(data2)); //清空结构体
52
53  data2.msg_flag = 333;
54  strcpy(data2.msg_buf, "gec");
55
56  //发送消息
57  int send_ret = msgsnd(msg_id, &data, sizeof(data) ,0);
```

```
    if (send_ret == -1)
    {
            printf("send data faile\n");
            return -1;
    }
    send_ret = msgsnd(msg_id, &data1, sizeof(data1) ,0);
    if (send_ret == -1)
    {
            printf("send data1 faile\n");
            return -2;
    }
    send_ret = msgsnd(msg_id, &data2, sizeof(data2) ,0);
    if (send_ret == -1)
    {
            printf("send data2 faile\n");
            return -3;
    }

    return 0;
}


==========example2 接收消息队列的数据

#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <string.h>
#include <errno.h>
```

```
88
89  //用于接收信息结构体
90  struct rcv_msg
91  {
92      long msg_flag;
93      char msg_buf[50];
94  };
95
96  int main(char argc, char*argv[])
97  {
98      //申请消息队列
99      int msg_id = msgget(12343, IPC_CREAT | IPC_EXCL | 0777);
100     if (msg_id == -1)
101     {
102             if (errno == EEXIST)  //消息队列如果存在则直接打开
103             {
104                     msg_id = msgget(12343,0777);
105             }
106             else
107             {
108                     perror("打开消息队列失败\n");
109                     return -1;
110             }
111     }
112     //存放信息结构体
113     struct rcv_msg read_data;
114     bzero(&read_data,sizeof(read_data));
115
116     //读取消息队列中的数据
117     msgrcv(msg_id, &read_data, sizeof(read_data), 333, 0);
```

```
118
119     printf("读取到的消息是 : %s\n", read_data.msg_buf);
120
121     return 0;
122 }
```

以上是消息队列的基本应用，我们在使用消息队列时，需注意以下特点：

1.数据从消息队列中读取出来以后，消息队列里的数据会消失，消息队列里的数据会被删除掉。
2.消息队列中，如果数据没有被读取完毕，则会残留在消息队列当中，累积起来。
3.在读取数据时，假如消息队列中没有指定的消息标识符，程序则会阻塞等待消息数据。
4.在消息队列中，多个消息标识符相同，需遵循先进先出的规则。

- 删除，设置，获取消息队列属性信息

```
1  NAME
2         msgctl - System V message control operations
3
4  SYNOPSIS
5         #include <sys/types.h>
6         #include <sys/ipc.h>
7         #include <sys/msg.h>
8
9         int msgctl(int msqid, int cmd, struct msqid_ds *buf);
10
11 参数解析:
12          参数1 int msqid : 消息队列ID
13          参数2 int cmd   :
```

```
                IPC_STAT   获取该MSG的信息，储存在结构体msqid_ds中
                IPC_SET    设置该MSG的信息，储存在结构体msqid_ds
                IPC_RMID   立即删除该MSG,并且唤醒所有阻塞在该MSG上的进程，
                           同时忽略第三个参数.直接填写NULL;

        参数3 struct msqid_ds *buf : 相关信息结构体缓冲区
              //消息队列的信息结构体
              struct msqid_ds
              {
                  struct ipc_perm msg_perm;     /* 权限相关信息 */
                  time_t          msg_stime;    /* 最后一次发送消息的时间 */
                  time_t          msg_rtime;    /* 最后一次接受消息的时间 */
                  time_t          msg_ctime;    /* 最后一次状态变更的时间 */
                  unsigned long   __msg_cbytes; /* 当前消息队列中的数据尺寸 */
                  msgqnum_t       msg_qnum;     /* 当前消息队列中的消息个数 */
                  msglen_t        msg_qbytes;   /* 消息队列的最大数据尺寸 */
                  pid_t           msg_lspid;    /* 最后一个发送消息进程的PID */
                  pid_t           msg_lrpid;    /* 最后一个接受消息进程的PID */
              };

              //消息队列权限相关信息
              struct ipc_perm
              {
                  key_t          __key;       /* 当前消息队列的键值 */
                  uid_t          uid;         /* 当前消息队列所有者的UID */
                  gid_t          gid;         /* 当前消息队列所有者的GID */
                  uid_t          cuid;        /* 当前消息队列创建者的UID */
                  gid_t          cgid;        /* 当前消息队列创建者的GID */
                  unsigned short mode;        /* 消息队列的读写权限 */
                  unsigned short __seq;       /* 序列号 */
```

```
44                  };
45
46  返回值  :
47          成 功 : 返回 0;
48          失 败 : 返回 -1;
```

练习：

用消息队列实现双向通信.