# 浙大城市学院实验报告

* 课程名称：操作系统原理实验
* 实验项目名称：实验九 进程通信——消息队列
* 学生姓名：徐彬涵
* 专业班级：软件工程2003
* 学号：32001272
* 实验成绩：
* 指导老师：胡隽
* 日期：2022/05/04

## 实验目的

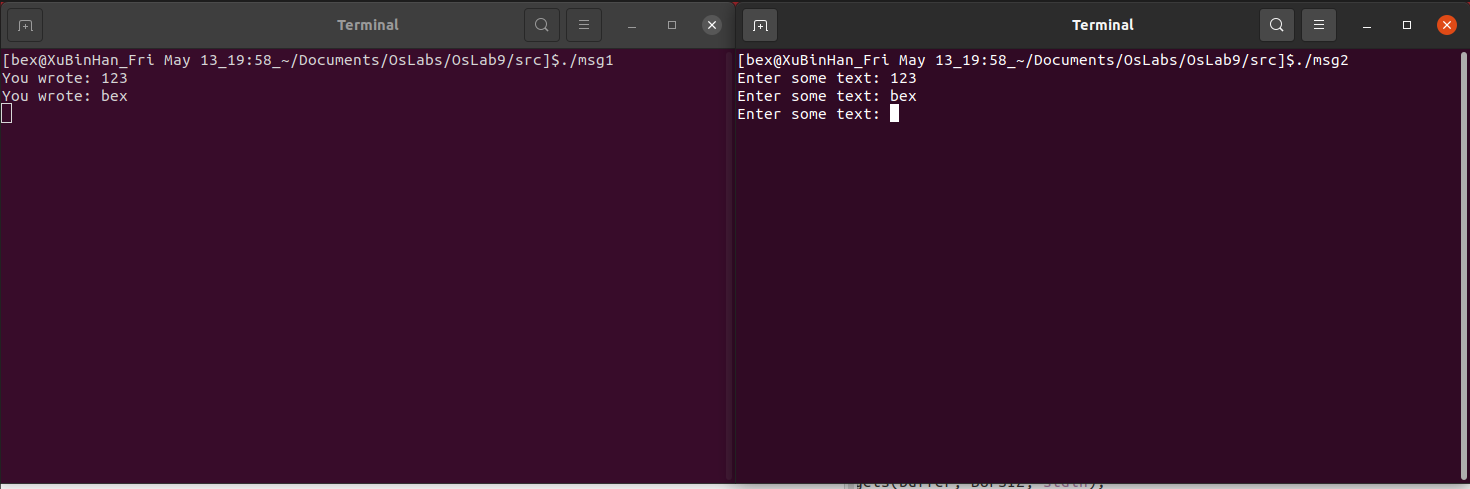
1. 了解Linux系统的进程间通信机构(IPC)
2. 理解Linux关于消息队列的概念
3. 掌握Linux支持消息队列的系统调用
4. 巩固进程同步概念

## 实验步骤

消息队列保存在内核中，是一个由消息结构组成的链表，每个消息队列（链表）有消息队列标识符，消息队列允许一个或多个进程写入或读取消息，因此消息队列提供了一种从一个进程向另一个进程发送一个数据块的方法，并可实现双向通信，消息队列解决了读写进程的同步和阻塞问题

### msg1.c \ msg2.c

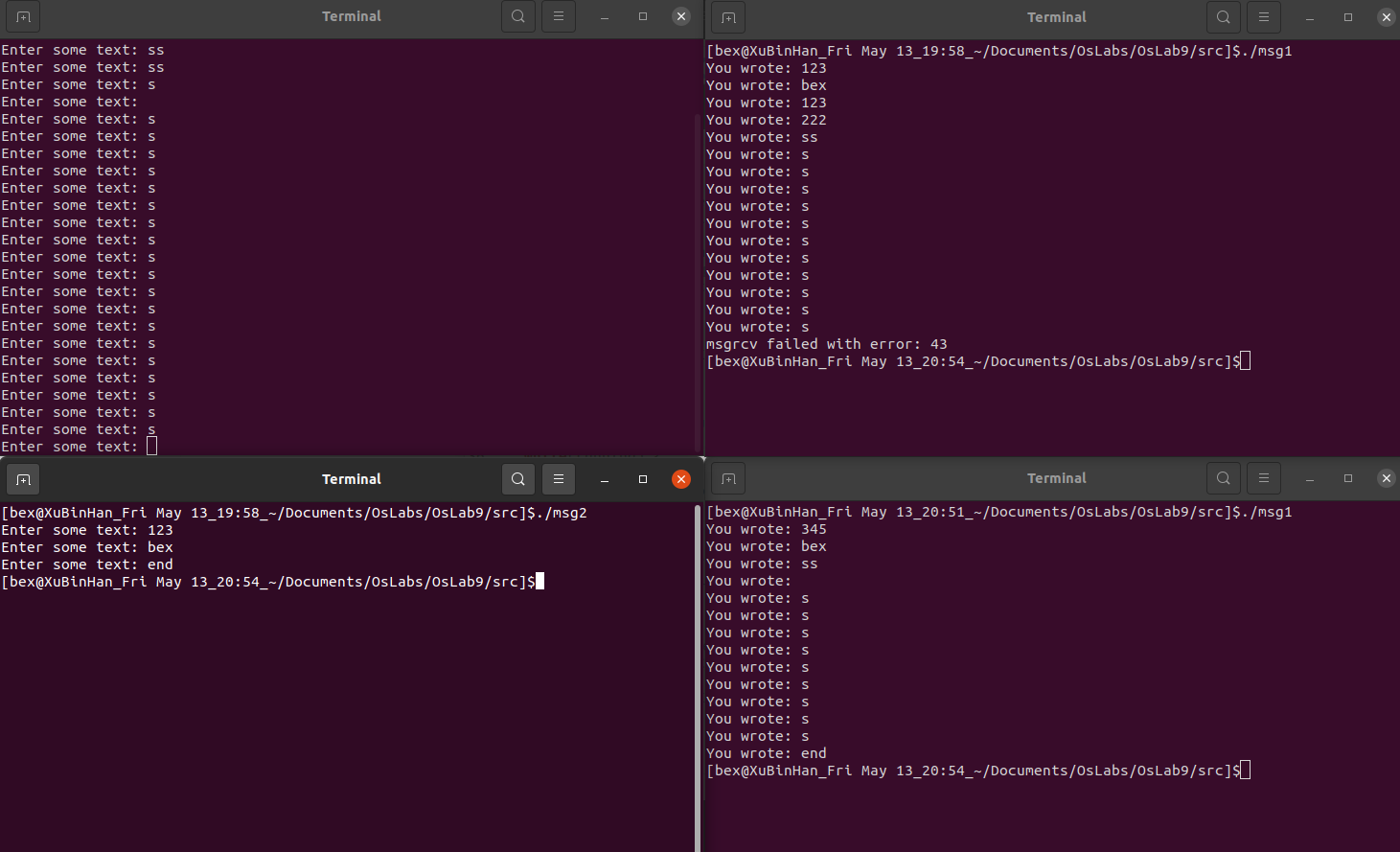
编译链接通过后，在两个终端分别运行发送进程和接收进程，观察进程并发执行结果，并思考下述问题：



1. 熟悉和消息队列相关的系统调用

* //创建消息队列  
  int msgget(  
   key\_t key, //消息对列的键值  
   int msgflag//消息队列的建立标志和存取权限  
  );  
  //例  
  msgget((key\_t)1234, 0666 | IPC\_CREAT);  
  /\*会返回键值为1234的消息队列的消息队列标识符，如果消息队列已存在则返回已存在消息队列的标识符  
  0666指的是消息队列权限  
  从左向右:  
  第一位:表示这是个八进制数 000  
  第二位:当前用户的经权限:6=110(二进制),每一位分别对就 可读,可写,可执行,,6说明当前用户可读可写不可执行  
  第三位:group组用户,6的意义同上  
  第四位:其它用户,每一位的意义同上  
  这里表示所有用户都对这个消息队列有读写执行权限\*/
* //向消息队列中发送消息  
  int msgsnd(  
   int msqid, //消息队列的标识符  
   const void \*msgp,  
   /\*指向消息缓冲区的指针,此位置用来暂时存储发送和接收的消息，是一个用户可定义的通用结果，形态如下  
   struct msgbyf{  
   long mtype; //消息类型，必须>0  
   char mtext[size]; //消息文本  
   }\*/  
   size\_t msgsz,   
   int msgflg  
  );  
  //返回值：成功：0、识别：-1
* //从消息队列中接收消息  
  ssize\_t msgrcv(  
   int msqid,//消息队列的标识符  
   void \*msgp,//指向消息缓冲区的指针  
   size\_t msgsz,//消息正文的字节数，不包括消息类型指针变量  
   long msgtyp,  
   /\*值为0表示接收消息队列的第一个消息  
   值大于0表示接收消息队列中第一个类型为msgtyp的消息  
   值小于0表示接收消息队列中第一个类型值不小于msgtyp绝对值且最小的\*/  
   int msgflg  
   /\*值为MSG\_NOERROR表示是返回字节比msgzsz多，消息截短  
   值为IPC\_NOWAIT，在队列空时，msgsnd()不会阻塞，立即返回-1  
   值为0，在队列空时，采取阻塞等待的处理方式\*/  
  );  
  //返回值：成功：0、识别：-1
* //在消息队列上执行指定的操作  
  int msgctl(  
   int msgid,//消息队列的标识符  
   int cmd,  
   /\*值为IPC\_STAT，读取消息队列的数据结构msqid\_ds，并将其存储在buf指定的地址中  
   值为IPC\_SET，设置消息队列的数据结构msqid\_ds中的ipc\_perm域值，值取自buf参数  
   值为IPC\_RMID，从系统内核中删除消息队列\*/  
   struct msqid\_ds \*buf//消息队列的msqid\_ds结构类型变量  
  );  
  //返回值：成功：0、识别：-1

1. 尝试运行多个发送进程和多个接收进程，观察进程的并发执行，并解释原因

* 
* 两个输出进程交替输出内容，原因在于进程并发，相互争夺资源，当一个接收进程接收后另一个接收进程就不会再接收，导致了两进程的交替输出

### msgthread.c

编译链接通过后，运行程序，通过系统提示，观察多线程的并发执行，并思考下述问题：

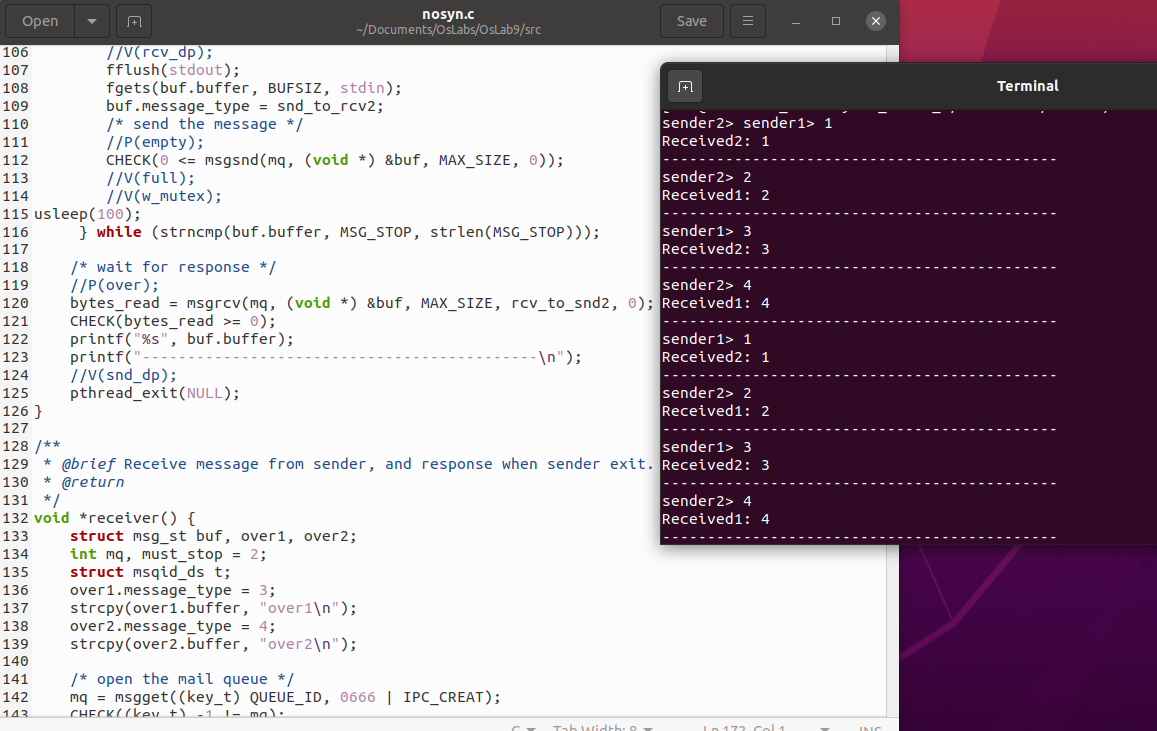
1. 熟悉和消息队列相关的系统调用

* //创建消息队列  
  int msgget(  
   key\_t key, //消息对列的键值  
   int msgflag//消息队列的建立标志和存取权限  
  );  
  //例  
  msgget((key\_t)1234, 0666 | IPC\_CREAT);  
  /\*会返回键值为1234的消息队列的消息队列标识符，如果消息队列已存在则返回已存在消息队列的标识符  
  0666指的是消息队列权限  
  从左向右:  
  第一位:表示这是个八进制数 000  
  第二位:当前用户的经权限:6=110(二进制),每一位分别对就 可读,可写,可执行,,6说明当前用户可读可写不可执行  
  第三位:group组用户,6的意义同上  
  第四位:其它用户,每一位的意义同上  
  这里表示所有用户都对这个消息队列有读写执行权限\*/
* //向消息队列中发送消息  
  int msgsnd(  
   int msqid, //消息队列的标识符  
   const void \*msgp,  
   /\*指向消息缓冲区的指针,此位置用来暂时存储发送和接收的消息，是一个用户可定义的通用结果，形态如下  
   struct msgbyf{  
   long mtype; //消息类型，必须>0  
   char mtext[size]; //消息文本  
   }\*/  
   size\_t msgsz,   
   int msgflg  
  );  
  //返回值：成功：0、识别：-1
* //从消息队列中接收消息  
  ssize\_t msgrcv(  
   int msqid,//消息队列的标识符  
   void \*msgp,//指向消息缓冲区的指针  
   size\_t msgsz,//消息正文的字节数，不包括消息类型指针变量  
   long msgtyp,  
   /\*值为0表示接收消息队列的第一个消息  
   值大于0表示接收消息队列中第一个类型为msgtyp的消息  
   值小于0表示接收消息队列中第一个类型值不小于msgtyp绝对值且最小的\*/  
   int msgflg  
   /\*值为MSG\_NOERROR表示是返回字节比msgzsz多，消息截短  
   值为IPC\_NOWAIT，在队列空时，msgsnd()不会阻塞，立即返回-1  
   值为0，在队列空时，采取阻塞等待的处理方式\*/  
  );  
  //返回值：成功：0、识别：-1
* //在消息队列上执行指定的操作  
  int msgctl(  
   int msgid,//消息队列的标识符  
   int cmd,  
   /\*值为IPC\_STAT，读取消息队列的数据结构msqid\_ds，并将其存储在buf指定的地址中  
   值为IPC\_SET，设置消息队列的数据结构msqid\_ds中的ipc\_perm域值，值取自buf参数  
   值为IPC\_RMID，从系统内核中删除消息队列\*/  
   struct msqid\_ds \*buf//消息队列的msqid\_ds结构类型变量  
  );  
  //返回值：成功：0、识别：-1

1. 回顾于POSIX线程控制的信号量相关的函数

* 信号量初始化：sem\_init(sem\_t \*sem, int pshared, unsigned int value);
* 参数说明：
* sem\_t \*sem：信号量变量
* int pshared： 指明信号量的类型。不为0时此信号量在相关进程间共享，否则只 能为当前进程的所以线程共享
* unsigned int value：该参数指定信号量的初始值
* 返回值：成功时返回0：错误时，返回-1，并将errno设置为合适的值
* P操作（信号量的-1操作）：int sem\_wait(sem\_t \*sem);
* 参数说明：
* sem\_t \*sem：信号量变量
* 函数说明：
* 等待信号量，如果信号量的值大于0，将信号量的值减1，立即返回。如果信号量 的值为0，则线程阻塞。
* 返回值：成功返回0，失败返回-1。
* V操作（信号量的+1操作）：int sem\_post(sem\_t \*sem);
* 参数说明：
* sem\_t \*sem：信号量变量
* 函数说明：
* 释放信号量，让信号量的值加1。若此时有sem\_wait正在阻塞则唤醒。

1. 尝试删除信号量的同步控制，观察并发线程运行的混乱状况

* 

1. 理清例程中并发线程同步和互斥关系

* w\_mutex信号用于实现互斥，控制了两个sender线程不同时进行，snd\_dp信号量用于实现同步，保证在一个sender线程向消息队列输入内容后reader线程必须输出内容之后下一个sender线程才可以再次输入内容，full信号量表示消息队列中的信息数量，empty信号量表示我们自己设定的消息队列中还可放入的信息数量，over信号量用于控制sender线程的关闭

### 编程题

实现双向通话

#### dialog1.c

#include <stdio.h>  
#include <sys/types.h>  
#include <sys/msg.h>  
#include <sys/ipc.h>  
#include <sys/wait.h>  
#include <stdlib.h>  
#include <string.h>  
#include <unistd.h>  
#include <errno.h>  
#include<signal.h>  
  
#define MSGKEY1 66  
#define MSGKEY2 68  
  
  
struct msgform  
{  
 long mtype;  
 char mtext[1000];  
}msg;  
  
int msgqid;  
  
void server( ) {  
 msgqid=msgget(MSGKEY1,0777|IPC\_CREAT);  
 while (1){  
 sleep(1);  
 msgrcv(msgqid,&msg,1024,0,0);  
 if (msg.mtype == 2) {  
 printf("(server)received:%s\n", msg.mtext);  
 strcpy(msg.mtext, "");  
 msg.mtype = 1;  
 msgsnd(msgqid, &msg, 1024, 0);  
 }else  
 continue;  
 }  
}  
void client() {  
 /\*打开消息队列\*/  
 msgqid=msgget(MSGKEY2,0777|IPC\_CREAT);  
 while (1) {  
 sleep(1);  
 scanf("%s", msg.mtext);  
 msg.mtype = 2;  
 msgsnd(msgqid, &msg, 1024, 0);  
 if (strncmp(msg.mtext, "bye", 3) == 0) break;  
 }  
}  
int main( )  
{  
 int pid;  
 if(fork()!=0)  
 server();  
 else  
 client();  
 kill(pid,2);  
 wait(0);  
}

#### dialog2.c

#include <stdio.h>  
#include <sys/types.h>  
#include <sys/msg.h>  
#include <sys/ipc.h>  
#include <sys/wait.h>  
#include <stdlib.h>  
#include <string.h>  
#include <unistd.h>  
#include <errno.h>  
#include<signal.h>  
  
#define MSGKEY1 66  
#define MSGKEY2 68  
  
  
struct msgform  
{  
 long mtype;  
 char mtext[1000];  
}msg;  
  
int msgqid;  
  
void server( ) {  
 msgqid=msgget(MSGKEY2,0777|IPC\_CREAT);  
 while (1){  
 sleep(1);  
 msgrcv(msgqid,&msg,1024,0,0);  
 if (msg.mtype == 2) {  
 printf("(server)received:%s\n", msg.mtext);  
 strcpy(msg.mtext, "");  
 msg.mtype = 1;  
 msgsnd(msgqid, &msg, 1024, 0);  
 }else  
 continue;  
 }  
}  
void client() {  
 /\*打开消息队列\*/  
 msgqid=msgget(MSGKEY1,0777|IPC\_CREAT);  
 while (1) {  
 sleep(1);  
 scanf("%s", msg.mtext);  
 msg.mtype = 2;  
 msgsnd(msgqid, &msg, 1024, 0);  
 if (strncmp(msg.mtext, "bye", 3) == 0) break;  
 }  
}  
int main( )  
{  
 int pid;  
 if(pid = fork()!=0)  
 server();  
 else  
 client();  
 kill(pid,2);  
 wait(0);  
}

