**实验6 进程通信（2）共享内存和消息队列通信**

实验指导

1. 消息队列通信原理
2. **消息队列原理**

目前主要有两种类型的消息队列：POSIX消息队列以及System V消息队列，System V消息队列目前被大量使用。本书主要介绍System V消息队列通信机制。

System V消息队列机制概括如下：

（1）System V消息队列是随内核持续的，只有在内核重启或者删除一个消息队列时，该消息队列才会真正被删除。

（2）消息队列就是一个消息的链表。每个消息队列都有一个队列头，用结构struct msg\_queue来描述。队列头中包含了该消息队列的大量信息，包括消息队列键值、用户ID、组ID、消息队列中消息数目等等，甚至记录了最近对消息队列读写的进程ID。用户可以访问这些信息，也可以设置其中的某些信息。

1. **消息队列数据结构**

（1）消息缓冲区

消息缓冲区用于存放消息，包括消息类型和消息正文，其数据结构定义如下：

Struct msgform

{

long mtype; /\*消息类型\*/

char mtext[ ]; /\*消息的文本\*/

};

（2）消息头结构

系统内核定义消息头结构描述和管理每个消息缓冲区，其数据结构定义如下：

struct msg

{

struct msg \*msgnext /\*指向消息队列中下一个消息的指针\*/

long msgtype; /\*消息类型\*/

short msgts; /\*消息正文的长度\*/

short msgspot; /\*消息正文的地址\*/

};

（3）消息队列头结构

为了描述和管理由消息构成的消息队列，设置消息队列头结构，描述队列的详细信息。其数据结构如下：

struct msqid\_ds

{

struct ipc\_perm msg\_perm; /\*消息队列访问控制结构\*/

struct msg \*msg\_first; /\*指向消息队列的第一个消息 \*/

struct msg \*msg\_last; /\*指向消息队列的最后一个消息\*/

\_\_kernel\_time\_t msg\_stime; /\*最后一次发送消息的时间 \*/

\_\_kernel\_time\_t msg\_rtime; /\*最后一次接收消息的时间 \*/

\_\_kernel\_time\_t msg\_ctime; /\*最后修改时间 \*/

unsigned short msg\_cbytes; /\* 队列中当前消息正文的总字节数 \*/

unsigned short msg\_qnum; /\*队列中消息个数\*/

unsigned short msg\_qbytes; /\* 队列允许容纳的最大字节数 \*/

\_\_kernel\_ipc\_pid\_t msg\_lspid; /\* 最后一次发送消息的进程ID \*/

\_\_kernel\_ipc\_pid\_t msg\_lrpid; /\* 最后一次接收消息的进程ID \*/

};

其中，struct kern\_ipc\_perm为消息队列的拥有者和访问方式，数据结构定义如下：

struct kern\_ipc\_perm

{

key\_t key; /\*该键值则唯一对应一个消息队列\*/

uid\_t uid; /\*当前用户ID\*/

gid\_t gid; /\*当前用户组ID\*/

uid\_t cuid; /\*队列创建用户ID\*/

gid\_t cgid; /\*队列创建用户组ID\*/

mode\_t mode; /\*队列的访问权限\*/

unsigned long seq; /\*访问队列的用户数\*/

}；

1. 消息队列系统函数

Linux系统为用户进程提供了用于建立消息队列、向消息队列发送或接收消息以及管理消息队列的系统调用，包括：msgget()，msgsnd()，msgrcv()以及msgctl()。这四个系统调用都需要如下三个头文件：

#include<sys/types.h>

#include<sys/ipc.h>

#include<sys/msg.h>

1.msgget函数

创建一个消息队列，获得一个与键值key相对应的消息队列描述符。搜索消息队列头表，确定是否有指定名字的消息队列。若无，核心将分配一新的消息队列头，并对它进行初始化，然后给用户返回一个消息队列描述符，否则它只是检查消息队列的许可权便返回。

1. 函数定义

系统调用格式：

msgqid=msgget(key,flag);

参数定义：

int msgget(key,flag);

key\_t key;

int flag;

参数说明：msgqid是该系统调用返回的消息队列描述字；key是用户指定的消息队列的键值；flag是用户设置的标志和访问方式。参数flag可以为IPC\_CREAT、IPC\_EXCL或IPC\_NOWAIT，通常取值IPC\_CREAT。

2.msgsnd函数

发送一个消息。向指定的msgqid代表的消息队列发送一个消息，并将该消息链接到该消息队列的尾部，即将发送的消息存储在msgp指向的msgbuf结构中。

系统调用格式：

msgsnd(msgqid,msgp,size,flag);

参数定义：

int msgsnd(msgqid,msgp,size,flag);

int msgqid,size,flag;

struct msgbuf \*msgp;

参数说明：msgqid是消息队列描述符；msgp是指向用户消息缓冲区的一个结构体指针。

* flag规定当核心用尽内部缓冲空间时应执行的动作：进程是等待，还是立即返回。若在标志flag中未设置IPC\_NOWAIT位，则当该消息队列中的字节数超过最大值时，或系统范围的消息数超过某一最大值时，调用msgsnd进程睡眠。若是设置IPC\_NOWAIT，则在此情况下，msgsnd立即返回。

3.msgrcv函数

接收一消息。从指定的消息队列中读取一个消息，并把消息存储在msgp指向的msgbuf结构中。

系统调用格式：

msgrcv(msgqid,msgp,size,type,flag);

参数定义：

int msgrcv(msgqid,msgp,size,type,flag);

int msgqid,size,flag;

struct msgbuf \*msgp;

long type;

参数说明：msgqid,msgp,size,flag与msgsnd中的对应参数相似，type是规定要读的消息类型。参数flag可以为以下几个值：

* IPC\_NOWAIT：如果没有满足条件的消息，调用立即返回，此时，errno=ENOMSG。
* IPC\_EXCEPT：与type>0配合使用，返回队列中第一个类型不为type的消息。
* IPC\_NOERROR：如果队列中满足条件的消息内容大于所请求的size字节，则把该消息截断，截断部分将丢失。

返回值：调用成功返回读出消息的实际字节数，否则返回-1。

对于msgrcv()系统调用，核心须完成下述工作：

（1）对消息队列的描述符和许可权等进行检查。若合法就继续执行；否则返回；

（2）根据type的不同分成三种情况处理：

* type=0：接收该队列的第一个消息，并将它返回给调用者；
* type>0：接收类型为type的第一个消息；
* type<0：接收小于等于type绝对值的最低类型的第一个消息。

1. 当所返回消息大小等于或小于用户的请求时，核心便将消息正文拷贝到用户区，并从消息队列中删除此消息，然后唤醒睡眠的发送进程。但当消息长度比用户要求的大时，则出错返回。

4.msgctl函数

msgctl()用于读取消息队列的状态信息并进行修改，如查询消息队列描述符、修改它的许可权及删除该队列等。

系统调用格式：

msgctl(msgqid,cmd,buf);

参数定义：

int msgctl(msgqid,cmd,buf);

int msgqid,cmd;

struct msgqid\_ds \*buf;

参数说明：msgqid是消息队列描述字；buf是用户缓冲区地址，供用户存放控制参数和查询结果；cmd是规定的命令。命令可分三类：

* IPC\_RMID：删除消息队列。
* IPC\_STAT：获取消息队列信息，返回的信息存储在buf指向的msqid\_ds结构中；如查询消息队列中的消息数目、队列中的最大字节数、最后一个发送消息的进程标识符、发送时间等。
* IPC\_SET：用来设置消息队列的属性，按buf指向的结构中的值，设置和改变有关消息队列的属性。如改变消息队列的用户标识符、消息队列的许可权等。

返回值：调用成功时返回0，不成功则返回-1。

1. 使用举例
2. **发送进程send.c**

/\*send.c \*/

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

struct msgbuf

{

long mtype;

char mtext[1000];

};

int main()

{

int qid;

key\_t key=75;

int len;

struct msgbuf msg;

qid=msgget(key,IPC\_CREAT|0666); // IPC\_CREAT : 不存在创建, 存在打开 ; 0666 : 权限

printf("Input some text to send:\n");

fgets((&msg)->mtext,512,stdin);

msg.mtype=getpid();

len=strlen(msg.mtext);

msgsnd(qid,&msg,len,0);

printf("message has been sent to message queue\n");

return 0;

}

1. **接收进程receive.c**

/\*receive.c \*/

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

struct msgbuf

{

long mtype;

char mtext[1000];

};

int main()

{

int qid;

key\_t key=75;

int len;

struct msgbuf msg;

qid=msgget(key,0666);

msgrcv(qid,&msg,512,0,0);

printf("read from queue：%s\n",msg.mtext);

printf("The sending process id is:%ld\n",msg.mtype);

//msgctl(qid,IPC\_RMID,NULL)；

printf("message queue has been deleted\n");

return 0;

}