linux消息队列-应用编程-专题讲座

written by 王保明

# 1消息队列基本概念

|  |
| --- |
| **消息队列**   * 消息队列提供了一个从一个进程向另外一个进程发送一块数据的方法 * 每个数据块都被认为是有一个类型，接收者进程接收的数据块可以有不同的类型值 * 消息队列也有管道一样的不足，就是每个消息的最大长度是有上限的（MSGMAX），每个消息队列的总的字节数是有上限的（MSGMNB），系统上消息队列的总数也有一个上限（MSGMNI）   对比：  管道：流管道 消息：有边界  先进先出 可以后进入、先出来  消息大小三大限制  cat /proc/sys/kernel/msgmax 最大消息长度 限制,一般 64k  cat /proc/sys/kernel/msgmnb 消息队列总的字节数,一般 64k  cat /proc/sys/kernel/msgmni 消息队列条目数 |
| **IPC对象数据结构**   * 内核为每个IPC对象维护一个数据结构   struct ipc\_perm {  key\_t \_\_key; /\* Key supplied to xxxget(2) \*/  uid\_t uid; /\* Effective UID of owner \*/  gid\_t gid; /\* Effective GID of owner \*/  uid\_t cuid; /\* Effective UID of creator \*/  gid\_t cgid; /\* Effective GID of creator \*/  unsigned short mode; /\* Permissions \*/  unsigned short \_\_seq; /\* Sequence number \*/  };  struct msqid\_ds {  struct ipc\_perm msg\_perm; /\* Ownership and permissions \*/  time\_t msg\_stime; /\* Time of last msgsnd(2) \*/  time\_t msg\_rtime; /\* Time of last msgrcv(2) \*/  time\_t msg\_ctime; /\* Time of last change \*/  unsigned long \_\_msg\_cbytes; /\* Current number of bytes in  queue (nonstandard) \*/  msgqnum\_t msg\_qnum; /\* Current number of messages  in queue \*/  msglen\_t msg\_qbytes; /\* Maximum number of bytes  allowed in queue \*/  pid\_t msg\_lspid; /\* PID of last msgsnd(2) \*/  pid\_t msg\_lrpid; /\* PID of last msgrcv(2) \*/  }; |
| **消息队列在内核中的表示** |
| **消息队列函数**   * #include <sys/types.h> * #include <sys/ipc.h> * #include <sys/msg.h> * int msgget(key\_t key, int msgflg); * int msgctl(int msqid, int cmd, struct msqid\_ds \*buf); * int msgsnd(int msqid, const void \*msgp, size\_t msgsz, int msgflg); * ssize\_t msgrcv(int msqid, void \*msgp, size\_t msgsz, long msgtyp, int msgflg); |
| **msgget函数**   * 功能：用来创建和访问一个消息队列 * 原型   + int msgget(key\_t key, int msgflg); * 参数   + key: 某个消息队列的名字   + msgflg:由九个权限标志构成，它们的用法和创建文件时使用的mode模式标志是一样的，状态字有九个位组成。 * 返回值：成功返回一个非负整数，即该消息队列的标识码；失败返回-1 |
| **msgget函数参数关系图** |
| **msgctl函数**   * 功能：消息队列的控制函数 * 原型   + int msgctl(int msqid, int cmd, struct msqid\_ds \*buf); * 参数   + msqid: 由msgget函数返回的消息队列标识码   + cmd:是将要采取的动作,（有三个可取值） * 返回值：成功返回0，失败返回-1   **cmd:将要采取的动作（有三个可取值），分别如下：** |

# 2消息队列基本API

## 消息队列的创建及控制

|  |
| --- |
| **消息队列函数**   * #include <sys/types.h> * #include <sys/ipc.h> * #include <sys/msg.h> * int msgget(key\_t key, int msgflg); * int msgctl(int msqid, int cmd, struct msqid\_ds \*buf); * int msgsnd(int msqid, const void \*msgp, size\_t msgsz, int msgflg); * ssize\_t msgrcv(int msqid, void \*msgp, size\_t msgsz, long msgtyp, int msgflg); |
| **msgget函数**   * 功能：用来创建和访问一个消息队列 * 原型   + int msgget(key\_t key, int msgflg); * 参数   + key: 某个消息队列的名字   + msgflg:由九个权限标志构成，它们的用法和创建文件时使用的mode模式标志是一样的 * 返回值：成功返回一个非负整数，即该消息队列的标识码；失败返回-1   man 2 msgget  有两个功能，获取或者创建,key用来指示消息队列，可以通过ftok来获取，也可以指定一个。 |
| **msgget函数参数关系图** |
| **msgctl函数**   * 功能：消息队列的控制函数 * 原型   + int msgctl(int msqid, int cmd, struct msqid\_ds \*buf); * 参数   + msqid: 由msgget函数返回的消息队列标识码   + cmd:是将要采取的动作,（有三个可取值） * 返回值：成功返回0，失败返回-1   **cmd:将要采取的动作（有三个可取值），分别如下：** |

## 消息队列的发送和接受

|  |
| --- |
| **msgsnd函数**   * 功能：把一条消息添加到消息队列中 * 原型   + int msgsnd(int msqid, const void \*msgp, size\_t msgsz, int msgflg); * 参数   + msgid: 由msgget函数返回的消息队列标识码   + msgp:是一个指针，指针指向准备发送的消息，   + msgsz:是msgp指向的消息长度，这个长度不含保存消息类型的那个long int长整型   + msgflg:控制着当前消息队列满或到达系统上限时将要发生的事情 * 返回值：成功返回0；失败返回-1 * msgflg=IPC\_NOWAIT表示队列满不等待，返回EAGAIN错误。 * 消息结构在两方面受到制约。首先，它必须小于系统规定的上限值；其次，它必须以一个long int长整数开始，接收者函数将利用这个长整数确定消息的类型 * 消息结构参考形式如下：   struct msgbuf {  long mtype;  char mtext[100];  } |
| **msgrcv函数**   * 功能：是从一个消息队列接收消息 * 原型   + ssize\_t msgrcv(int msqid, void \*msgp, size\_t msgsz, long msgtyp, int msgflg); * 参数   + msgid: 由msgget函数返回的消息队列标识码   + msgp:是一个指针，指针指向准备接收的消息，   + msgsz:是msgp指向的消息长度，这个长度不含保存消息类型的那个long int长整型   + msgtype:它可以实现接收优先级的简单形式   + msgflg:控制着队列中没有相应类型的消息可供接收时将要发生的事 * 返回值：成功返回实际放到接收缓冲区里去的字符个数，失败返回-1 |
| * msgtype=0返回队列第一条信息 * msgtype>0返回队列第一条类型等于msgtype的消息 * msgtype<0返回队列第一条类型小于等于msgtype绝对值的消息，并且是满足条件的消息类型最小的消息 * msgflg=IPC\_NOWAIT，队列没有可读消息不等待，返回ENOMSG错误。 * msgflg=MSG\_NOERROR，消息大小超过msgsz时被截断 * msgtype>0且msgflg=MSG\_EXCEPT，接收类型不等于msgtype的第一条消息。 |
|  |

# 3消息队列综合api使用

|  |
| --- |
| 同一个进程，使用消息队列 |
| struct msg\_buf  {  long mtype;  char data[255];  };    /\* 注意long 和 int 在32bit 和 64bit系统之下是不一样的  struct msg\_buf  {  long mtype;  char data[255];  };  \*/    int main()  {  key\_t key;  int msgid;  int ret;  struct msg\_buf msgbuf;  int msgtype = getpid();  //通过路径获取键值key，各个应用app统一的获取key，也可以通过一个种子文件  // key=ftok("./tmp/",'a');  //export SEEDFILE=”/home/ltf/etc/seedfile”,所有的app读环境变量，getenv(“SEEDFILE”)  key=ftok("./msgfile",'a');  printf("key =[%x]\n",key);    printf("sizeof(long):%ld, sizeof(int):%d \n", sizeof(long), sizeof(int));    msgid=msgget(key, IPC\_CREAT |IPC\_EXCL|0666); //通过文件对应  if(msgid==-1)  {  if (errno == EEXIST)  {  printf("EEXIST:.....\n");  key=ftok("./msgfile",'a');  msgid=msgget(key, IPC\_CREAT|0666); //通过文件对应  }  else  {  printf("create error\n");  perror("msget: \n");  return -1;  }    }  printf("msgid:%d \n", msgid);  msgbuf.mtype = msgtype; // getpid();  printf("getpid(): %d \n", getpid());  strcpy(msgbuf.data,"test haha");  ret = msgsnd(msgid,&msgbuf, sizeof(msgbuf.data), IPC\_NOWAIT);  if(ret==-1)  {  printf("send message err\n");  perror("senderr");  return -1;  }  sleep(1);  memset(&msgbuf,0,sizeof(msgbuf));    ret=msgrcv(msgid, &msgbuf, sizeof(msgbuf.data), msgtype, IPC\_NOWAIT);  if(ret==-1)  {  printf("recv message err\n");  perror("dd");  return -1;  }  printf("recv msg =[%s]\n",msgbuf.data);    } |
|  |

# 4 消息队列项目开发案例

|  |
| --- |
| 消息队列实现回射客户/服务器 |
|  |
|  |

消息队列由内核持久化

删除ipc消息队列命令：

ipcrm msg msqid

linux下一切文件皆设备

eg:

0666：消息队列存在则返回，不存在返货ENOENT

0666|IPC\_CREATE:消息队列存在使用旧的，消息队列不存在则创建。

IPC\_CREATE|IPC\_EXCL : 若消息队列存在，返回EEXIST错误，反之，创建。单独使用IPC\_EXCL无意义。

msgget(IPC\_PRIVATE, 0666);

创建一个key为0的消息队列，IPC\_PRIVATE(0)表示每次都创建新的共享内存。这样两个进程就不能共享消息队列了，IPC\_PRIVATE表示自己使用,此时IPC\_CREATE和IPC\_EXCL将被忽律。

创建的消息队列 自在自己的家族中使用，不在没有血缘关系的进程间用,fork的可以。每一次创建的消息队列的msqid不一样，msqid即使传给其他进程，也不能使用。

msgget(key, 0400|IPC\_CREATE) -> msgget(key, 0600)

权限位：创建的权限只有0400， 如果按照0600打开消息队列，则会失败。

421 rwx

消息队列这些IPC对象，systemv是让linux内核持久化(保存和管理)

IPC使用同一套函数。查看IPC 使用 ipcs tool

Ipcs 和 ipcs -l

消息队列的控制

man 2 msgctl 先获取在设置

数据结构：msqid\_ds -> ipc\_perm， linux通过这两个数据结构管理ipc对象。

msgsend

1. 消息是有类型的
2. 发送消息的结构 msgbuf
3. flag = flag|MSG\_NOWAIT 设置非阻塞

msgrcv

1. 接收消息类型

msgtype > 0 指定消息 = 0 从队列里取第一条消息 < 0 若 -3 就取 1 2 3 这样的消息

1. flag
2. 内容的长度 msg

案例：

客户端发送服务器消息类型总是1

服务器端回客户端type是对方进程号，客户端接收类型和自己进程号一样的消息

相当于服务器端从消息队列中收消息，然后服务器端分类型回复客户端(通过消息队列)

n个消息队列进行交换有没有产生死锁的可能？

n个客户端向服务端发送消息(本质上是向内核消息队列发送消息)，若消息队列满了服务端回射时会阻塞，造成程序死锁，即使使用非阻塞接口也会阻塞。

管道

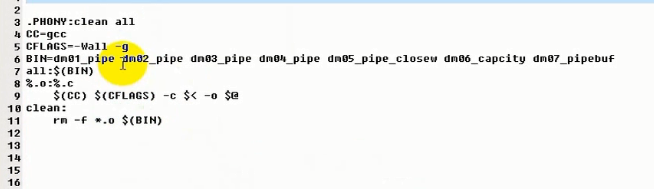
fd0表示读，fd1表示写，子进程写需要关闭读端，父进程读需要关闭写端。

pipe

write

read

makefile例子：



cleanpic 脚本

