进程间通信

武汉众嵌

消息队列

消息队列概述

- 消息队列就是一个消息的列表。用户可以从消息队列种添加消息、读取消息等。从这点上看，消息队列具有一定的FIFO 的特性，但是它可以实现消息的随机查询，比FIFO 具有更大的优势。同时，这些消息又是存在于内核中的，由“队列ID”来标识。

key值和ID值

- Linux系统为每个IPC机制都分配了唯一的ID，所有针对该IPC机制的操作都使用对应的ID。因此，通信的双方都需要通过某个办法来获取ID值。显然，创建者根据创建函数的返回值可获取该值，但另一个进程如何实现呢？显然，Linux两个进程不能随意访问对方的空间（一个特殊是，子进程可以继承父亲进程的数据，实现父亲进程向子进程的单向传递），也就不能够直接获取这一ID值。

key值和ID值

- 为解决这一问题，IPC在实现时约定使用key值做为参数创建，如果在创建时使用相同的key值将得到同一个IPC对象的ID（即一方创建，另一方获取的是ID），这样就保证了双方可以获取用于传递数据的IPC机制ID值。

- System V 创建对象时假设进行IPC通讯双方都取了相同的key值.这样将双方关联起来
- 生成key的方法有三种
  - 双方直接设置为一个相同的整数为key值
  - 用IPC_PRIVA让系统自动产生一个key值，
  - 用ftok函数将一个路径转换为key值

key_t键和ftok函数

- 消息队列，共享内存和信号量它们都是从Unix继承而来，所以它们具有许多类似点。
- 三种类型的IPC使用key_t值作为它们的名字。
- key_t这个数据类型定义为整形，这些整数通常是ftok函数赋予的。

ftok函数

- 函数ftok把一个以存在的路径名和一个整数标识符转换为key_t值。成为IPC键。
- Key_t ftok(char* pathname,int id);
- 返回值：成功返回键值，否则返回-1。
- Pathname是一个已存在的文件名，id为一个非0值。

```
int main(int argc,char *argv[])
{
    key_t key;
    int i;
    key=ftok( “./XXX” ,1);
    printf( “key is %d\n” ,key);
}
```

消息队列

- 在早期进程间通信之一的管道它只能支持先写先读原则，而信号在传输过程中传输的信息量有限，这些不足给应用程序的开发带来了很多不便。
- 消息队列客服了上述的不足。即支持先写先读原则，也支持读取指定消息类型的数据。同时，消息数据可以是自己定义的类型。

- 消息队列消息通常要以一个long mtype放在消息开始，mtype成员代表消息类型，从消息队列中读取消息的一个重要依据就是消息的类型
  - struct msgbuf{ long mtype; char mtext[1]; };

- 消息队列与管道以及有名管道相比，具有更大的灵活性
  - 它提供有格式字节流，有利于减少开发人员的工作量
  - 消息具有类型，在实际应用中，可作为优先级使用。这两点是管道以及有名管道所不能比的
  - 消息队列可以在几个进程间复用，而不管这几个进程是否具有亲缘关系，这一点与有名管道很相似；但消息队列是随内核持续的，与有名管道（随进程持续）相比，生命力更强，应用空间更大。

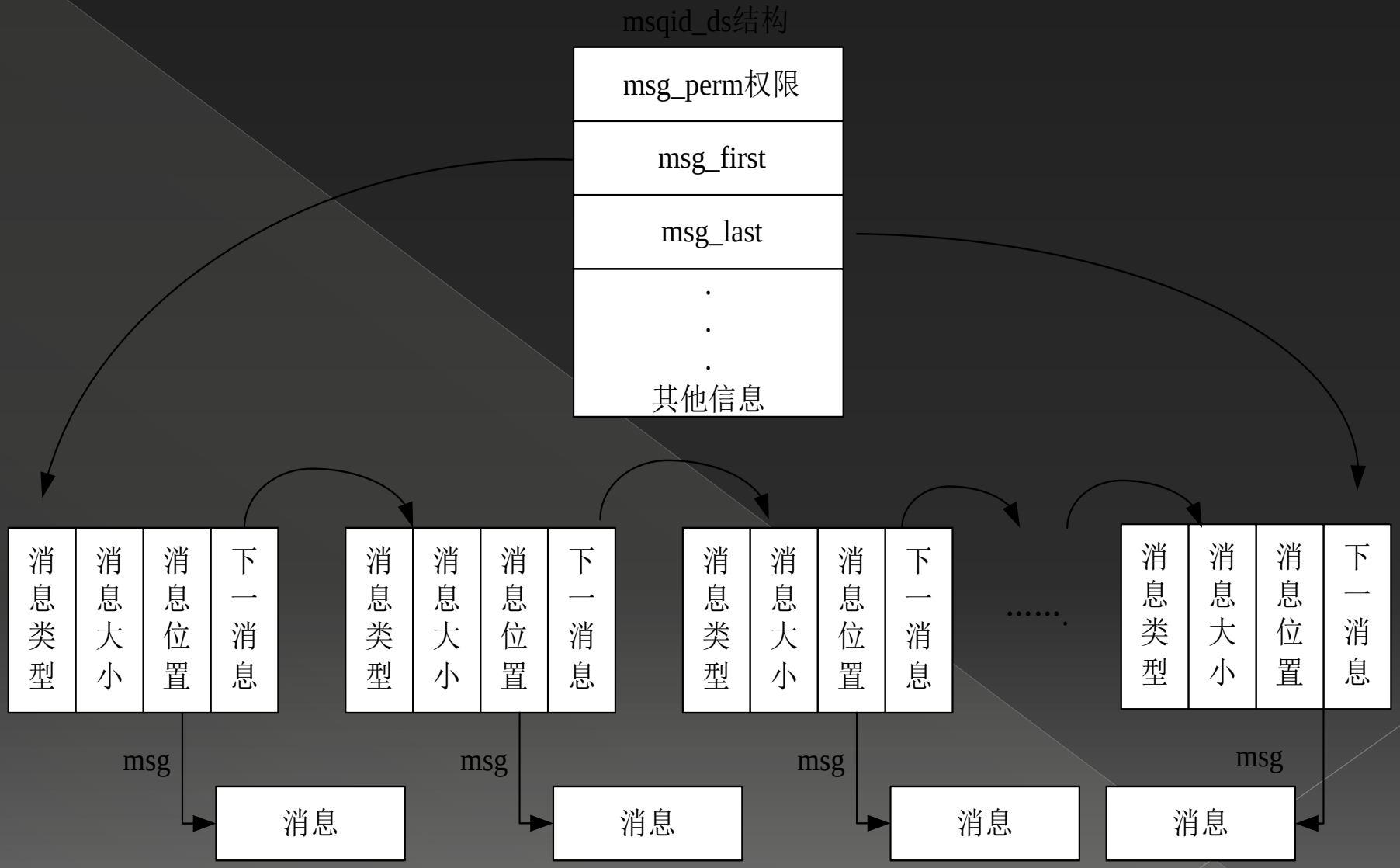
消息队列

- 消息队列是内核地址空间中的一个内部链表。可以把消息看作一个具有特点格式记录，消息可以按照顺序发送到队列中，也可以按照几种不同的方式从队列中读取。每一个消息队列用一个唯一的标识符表示。
- 消息队列中的内部链表是靠struct msqid_ds结构体维护的。创建每一个消息队列，内核就会创建一个对应的结构体进程维护。

消息队列

- 消息队列的msgid_sys控制结构：
- struct msqid_ds{
- struct ipd_perm msg_perm; /*操作权限结构*/
- struct msg * msg_first; /*指向消息队列的最后一个消息*/
- struct msg * msg_last; /*指向消息队列的最后一个消息*/
- msglen_t msg_cbytes; /*队列中当前的字节数*/
- msgqnum_t msg_qnum; /*队列中的消息数*/
- msglen_t msg_qbytes; /*队列中可容纳的最大字节数*/
- pid_t msg_lsid; /*最后发送消息的进程号*/
- pid_t msg_lrpid; /*最后接收消息的进程号*/
- time_t msg_stime; /*最后发送时间*/
- time_t msg_rtime; /*最后接受的时间*/
- time_t msg_ctime; /*消息队列最后修改时间*/
- }

```
struct msg_msg {
        struct list_head m_list;
        long    m_type;                         //消息类型
        int m_ts;               /* message text size */     //消息大小
        struct msg_msgseg* next;                //下一个消息位置
        void *security;                         //真正消息位置
        /* the actual message follows immediately */
};
```


msqid_ds结构
msg_perm权限
msg_first
msg_last
.
.
.
其他信息
消息类型
消息大小
消息位置
下一消息
消息类型
消息大小
消息位置
下一消息
消息类型
消息大小
消息位置
下一消息
.....
消息类型
消息大小
消息位置
下一消息
msg
msg
msg
msg
消息
消息
消息
消息

消息队列使用方法的简介

- 头文件
  - #include <sys/types.h>
  - #include <sys/ipc.h>
  - #include <sys/msg.h>
- msgget打开或创建消息队列
  - int msgget(key_t key, int msgflg) ;返回线队列ID
- msgrcv从队列接收消息
  - int msgrcv(int msqid, struct msgbuf *msgp, int msgsz, long msgtyp, int msgflg);
- msgsnd 向队列发送消息
  - int msgsnd(int msqid, struct msgbuf *msgp, int msgsz, int msgflg);
- msgctl 发送队列控制命令
  - **msgctl(int msqid, int cmd, struct msqid_ds *buf);**
  - 共有三种cmd操作： IPC_STAT、IPC_SET 、IPC_RMID。

消息队列

- 创建或打开一个新消息队列:
- int msgget(key_t key, int msgflg)
- 返回值：成功返回消息队列标识符，否则返回-1。
- 参数：参数key是一个键值，由ftok获得或设置成常数IPC_PRIVATE； msgflg参数是一些标志位。该调用返回与健值key相对应的消息队列描述字。

消息队列

- 在以下两种情况下，该调用将创建一个新的消息队列：
  - 如果没有消息队列与健值key相对应，并且msgflg中包含了IPC_CREAT标志位；
  - key参数为IPC_PRIVATE；
- 参数msgflg可以为以下：IPC_CREAT、IPC_EXCL、IPC_NOWAIT(接收时为非阻塞)。
- 注意：参数key设置成常数IPC_PRIVATE并不意味着其他进程不能访问该消息队列，只意味着即将创建新的消息队列。

```
int main(int argc,char** argv){
        key_t key;
        int msgid;
        key = ftok(".",1);
        if(key < 0 ){
                perror("ftok err");
                exit(1);
        }
        printf("key: %x\n",key);
        msgid = msgget(key,0666|IPC_CREAT);
        if(msgid < 0){
                perror("msgid err.");
        }
        printf("msgid : %d\n",msgid);
}
```

消息队列

- 消息队列发送函数：
- int msgsnd(int msqid, void*ptr, int msgsz, int msgflg);
- 返回值：成功返回0，否则返回-1；
- Msqid是消息队列的标识符。ptr是要发送的消息，该结构具有如下模板：

```
struct msgbuf{
    long mtype;
    char mtext[1];
};
```

其中消息类型(mtype)必须为大于0的值。

注意：消息的数据部分(mtext)并不局限于文本。任何形式的数据都是允许的。

消息队列

- Msgsz是以字节为单位指定待发消息的长度。这是位于长整形消息类型之后的用户自定义数据长度。
- Msgflg参数既可以是0，也可以是IPC_NOWAIT。IPC_NOWAIT使得msgsnd调用非阻塞：如果没有存放新消息的空间，该函数就立即返回。

```
typedef struct MsgBuf
{
    long mtype;
    char mtext[128];
}msgbuf;
int main(){
    key_t key;
    int msgid;
    if((key = ftok(".",1)) < 0){
        perror("ftok err.");
        return -1;
    }
```

```
    if((msgid = msgget(key,0666|IPC_CREAT)) < 0){
        perror("msgid err.");
        return -1;
    }
    printf("key:%x,msgid:%d\n",key,msgid);
    msgbuf buf;
    scanf("%d %s",&buf.mtype,buf.mtext);
    if(msgsnd(msgid,&buf,sizeof(msgbuf)-sizeof(long),0) < 0){
        perror("msg snd err.");
        exit(1);
    }
}
```

消息队列

- 接收消息函数：
- int msgrcv(int msqid, void *msgp, int msgsz, long msgtyp, int msgflg);
- 返回值：出错返回-1，否则返回接收到数据的字节数。
- Msgrcv的前三个参数和msgsnd意义相同。
- Msgtyp指定希望从所给队列中读取什么样的消息：
  - 如果为0，则返回第一个消息。
  - 如果大于0，则返回其类型值为msgtyp的第一个消息。
  - 如果小于0，则返回其类型值小于或等于msgtyp绝对值的消息中最小的第一个消息。
- Msgflg参数指定所请求类型的消息不在所指定的队列中该做何处理。可以设置为：IPC_NOWAIT/MSG_NOERROR(只能忽略E2BIG)

```
typedef struct MsgBuf
{
    long mtype;
    char mtext[128];
}msgbuf;
int main(){
    key_t key;
    int msgid;
    if((key = ftok(".",1)) < 0){
        perror("ftok err.");
        return -1;
    }
```

```
    if((msgid = msgget(key,0666|IPC_CREAT)) < 0){
        perror("msgid err.");
        return -1;
    }
    printf("key:%x,msgid:%d\n",key,msgid);
    msgbuf buf;
    if((n = msgrcv(msgid,&buf,sizeof(buf)-sizeof(long),0,0)) < 0){
        perror("MSG RECV ERR");
        exit(0);
    }
    printf("n : %d\n",n);
    printf("type:%ld,text:%s\n",buf.mtype,buf.mtext);
}
```

消息队列

- 设置消息队列属性函数：
- int msgctl(int msqid, int cmd, struct msqid_ds *buf);
- Msgctl函数提供了三个命令：
  - IPC_RMID:从系统中删除由msgid指定的消息队列。对于该命令，第三个参数被忽略。
  - IPC_SET：给所指定的消息队列设置msqid_ds结构的4个成员(msg_perm.uid,msg_perm.gid,msg_perm.mode和msg_qbytes)。
  - IPC_STAT：给调用者返回所指定消息队列对应的当前msgid_ds结构。

```
typedef struct MsgBuf
{
    long mtype;
    char mtext[1];
}msgbuf;
int main(int argc,char** argv)
{
    key_t key;
    int msgid;
    struct msqid_ds info;
    msgbuf buf;
    if((key = ftok(".",1)) < 0){
        perror("ftok err.");
        exit(1);
    }
```

```
    if((msgid = msgget(key,0666|IPC_CREAT)) < 0){
            perror("msgget err.");
            exit(1);
    }
    buf.mtype = 1;
    buf.mtext[0] =1;
    msgsnd(msgid,&buf,1,0);
    msgctl(msgid,IPC_STAT,&info);
    printf("read-write:%03o, cbytes = %lu,qnum = %lu,qbytes = %lu\n",info.msg_perm.mode & 0777,info.msg_cbytes,info.msg_qnum,info.msg_qbytes);
    exit(0);
}
```

消息队列与管道的耗时比较

操作	用户(s)	系统(s)	时钟(s)
消息队列	0.57	3.63	4.22
管道	0.50	3.21	3.71

- 有数据可见，消息队列与管道相比较在速度方面基本没很大的差别(管道略快)。所以消息队列主要用于传输可靠的控制流的记录信息，并且带有消息类型。

消息队列——实例

- 用消息队列编写一个客户端服务器通信的程序
- 【实验目的】
- 通过此实验，学员可以熟悉消息队列的概念，并能够用消息队列编写一个客户端服务器通信的程序。
-
- 【实验原理】
- 本实验需要用消息队列设计一个简易的双人聊天程序（一个服务器，两个客户端）。消息队列重点在于消息类型的匹配，客户端和服务端的“通信协议”的设计。设计思想如下:
- 服务器端：接受客户端发来的任何信息，并根据其消息类型，转发给对应的客户端。同时，检测是否有退出标志，有则给所有的客户端发送退出标志，等待 10s 后，确定客户端都退出后，删除消息队列，释放空间，并退出。
- 客户端:A 和 B,A 给 B 发送信息,先发给服务器,由服务器根据自定义协议转发该消息给B。同时 B 接受到消息后,经由服务器给 A 一个回执信息,以此形成简易的聊天模式。

消息队列——小结

- 消息队列与管道以及有名管道相比，具有更大的灵活性：
  - 首先，它提供有格式字节流，有利于减少开发人员的工作量；
  - 其次，消息具有类型，在实际应用中，可作为优先级使用。
  - 消息队列可以在几个进程间复用，而不管这几个进程是否具有亲缘关系，这一点与有名管道很相似；
  - 消息队列是随内核持续的，与有名管道（随进程持续）相比，生命力更强，应用空间更大。

例题

- 利用消息队列设计一个简易的双人聊天程序（一个服务器，两个客户端）。
- 消息队列重点在于消息类型的匹配，客户端和服务端的“通信协议”的设计。设计思想如下:
- 服务器端：接受客户端发来的任何信息，并根据其消息类型，转发给对应的客户端。同时，检测是否有退出标志，有则给所有的客户端发送退出标志，等待10s后，确定客户端都退出后，删除消息队列，释放空间，并退出。
- 客户端:A和B,A给B发送信息,先发给服务器,由服务器根据自定义协议转发该消息给B。同时B接受到消息后,经由服务器给A一个回执信息,以此形成简易的聊天模式。