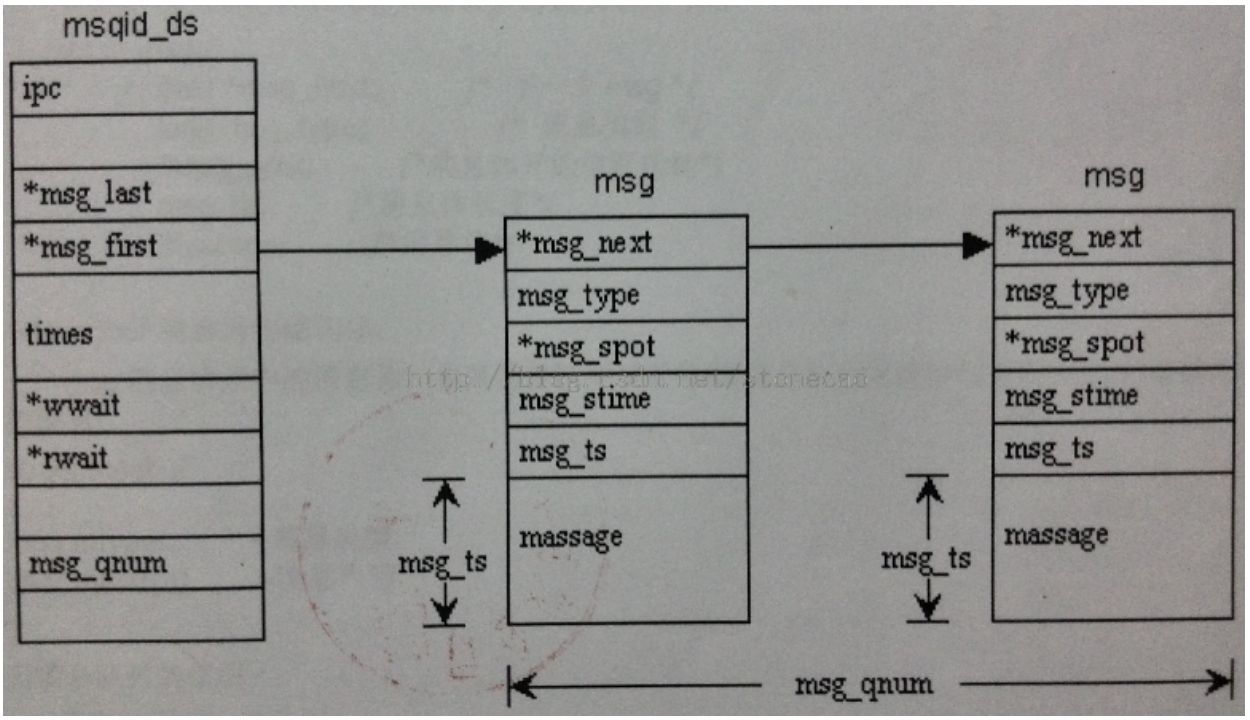
--信号（Singal）
--管道（Pipe）
--消息队列（Message）
--信号量（Semaphore）

在本地通信中(同一台机器上的进程间通讯)，socket的网络特性却成了累赘，组装解析网络报头、报文确认、CRC校验等都是针对网络的，本地通信没有必要，反而会影响传输效率。本地通信的一些传统技术，如管道、FIFO、消息队列等，没有网络功能的负担，传输速度应该高于socket

1：消息队列（Message）

消息队列是链表队列，它通过内核提供一个struct msqid_ds *msgque[MSGMNI]向量维护内核的一个消息队列列表，因此linux系统支持的最大消息队列数由msgque数组大小来决定，每一个msqid_ds表示一个消息队列，并通过msqid_ds.msg_first、msg_last维护一个先进先出的msg链表队列，当发送一个消息到该消息队列时，把发送的消息构造成一个msg结构对象，并添加到msqid_ds.msg_first、msg_last维护的链表队列，同样，接收消息的时候也是从msg链表队列尾部查找到一个msg_type匹配的msg节点，从链表队列中删除该msg节点，并修改msqid_ds结构对象的数据。



2.消息队列的数据结构
--1.struct msqid_ds *msgque[MSGMNI]向量：

msgque[MSGMNI]是一个msqid_ds结构的指针数组，每个msqid_ds结构指针代表一个系统消息队列，msgque[MSGMNI]的大小为MSGMNI=128，也就是说系统最多有MSGMNI=128个消息队列

--2.struct msqid_ds

一个消息队列结构

struct msqid_ds 中主要数据成员介绍如下：

```
struct msqid_ds
{
        struct ipc_perm msg_perm;
        struct msg *msg_first;                          /*消息队列头指针*/
        struct msg *msg_last;                           /*消息队列尾指针*/
        __kernel_time_t msg_stime;                      /*最后一次插入消息队列消息的时间*/
        __kernel_time_t msg_rtime;                      /*最后一次接收消息即删除队列中一个消息的时间*/
        __kernel_time_t msg_ctime;
        struct wait_queue *wwait;                       /*发送消息等待进程队列*/
        struct wait_queue *rwait;
        unsigned short msg_cbytes;
        unsigned short msg_qnum;                        /*消息队列中的消息个数*/
        unsigned short msg_qbytes;
        __kernel_ipc_pid_t msg_lspid;                   /*最后一次消息发送进程的pid*/
        __kernel_ipc_pid_t msg_lrpid;                   /*最后一次消息发送进程的pid*/
};
```

--3.struct msg 消息节点结构：

msqid_ds.msg_first，msg_last维护的链表队列中的一个链表节点

```
struct msg
{
       msg *msg_next;           /*下一个msg*/
       long msg_type;           /*消息类型*/
       *msg_spot;               /*消息体开始位置指针*/
        msg_ts;                 /*消息体长度*/
        message;                /*消息体*/
}
```

--4.msgbuf消息内容结构：

msg 消息节点中的消息体，也是消息队列使用进程（消息队列发送接收进程）发送或者接收的消息

```
struct msgbuf
{
    long mtype;       --消息类型
    char mtext[n];    --消息内容
}
```

3.消息队列的使用

1.消息队列Key的获取：

在程序中若要使用消息队列，必须要能知道消息队列key,因为应用进程无法直接访问内核消息队列中的数据结构，因此需要一个消息队列的标识，让应用进程知道当前操作的是哪个消息队列，同时也要保证每个消息队列key值的唯一性

----a.通过ftok函数获取

```
key_t key;
key=ftok(".","a")
```

该函数通过一个路径名称映射出一个消息队列key(我的理解是使用路径映射的方式比较容易获取一个唯一的消息队列key)

----b.直接定义key:

```
#define MSG_KEY    123456
```

自定义key的方式要注意避免消息队列的重复。

2.获取或者打开一个消息队列

```
qid=msgget(key_t key, int msgflag)
```

--key: 消息队列key

--msgflag:

IPC_PRIVATE:创建一个该进程独占的消息队列，其它进程不能访问该消息队列

IPC_CREAT:若消息队列不存在，创建一个新的消息队列，若消息队列存在，返回存在的消息队列

IPC_CREAT | IPC_EXCL: IPC_EXCL标志本身没有多大意义，与IPC_CREAT一起使用，保证只创建新的消息队列，若对应key的消息队列已经存在，则返回错误

IPC_NOWAIT:小队列以非阻塞的方式获取（若不能获取，立即返回错误）

0660：存取权限控制符

函数原理

------1）如果key==IPC_PRIVATE，则申请一块内存，创建一个新的消息队列（数据结构msqid_ds），将其初始化后加入到msgque向量表中的某个空位置处，返回标示符。

------2）在msgque向量表中找键值为key的消息队列，如果没有找到，结果有二：

msgflag表示不创建新的队列，则错误返回。

msgflag表示要创建新的队列（IPC_CREAT），则创建新消息队列，创建过程如1）。

-------3）如果在msgque向量表中找到了键值为key的消息队列，则有以下情况：

如果msgflg表示一定要创建新的消息队列而且不允许有相同键值的队列存在，则错误返回。

如果找到的队列是不能用的或已经损坏的队列，则错误返回。

认证和存取权限检查，如果该队列不允许msgflg要求的存取，则错误返回。

正常，返回队列的标识符。

3.发送一个消息到消息对列

int msgsnd (int msqid, struct msgbuf *msgp, size_t msgsz, int msgflg)

------msqid为消息队列的qid

------msgp对应消息内容结构体指针

------msgsz消息的大小即msgp指针指向的消息结构体的大小

------msgflg消息标志

0：忽略该标志位，以阻塞的方式发送消息到消息队列

IPC_NOWAIT：以非阻塞的方式发送消息，若消息队列满，函数立即返回。

------返回：

0: 成功

-1:非阻塞方式访问满消息队列返回

EACCES：没有该消息队列写权限

EFAULT:消息队列地址无法获取

EIDRM:消息队列已经被删除

EINTR:消息队列等待写入的时候被中断

ENOMEM:内存不够

函数原理：

1)计算id = (unsigned int) msqid % MSGMNI,然后根据id在linux系统消息队列向量msgque[MSGMNI]中查找对应的消息队列，并进行认证检查，合法性检查

2)如果队列已满，以可中断等待状态（TASK_INTERRUPTIBLE）将当前进程挂起在wwait等待队列(发送消息等待队列)上（msgflag==0）。

3)否则 根据msgbuf的大小申请一块空间，并在其上创建一个消息数据结构struct msg(内核空间)，将消息缓冲区中的消息内容拷贝到该内存块中消息头的后面（从用户空间拷贝到内

核空间）。
4)将消息数据结构加入到消息队列的队尾，修改队列msqid_ds的相应参数。
5)唤醒在该消息队列的rwait进程队列(读等待进程队列)上等待读的进程,并返回。

4.从消息队列接收一个消息到msgbuf*

int msgrcv (int msqid, struct msgbuf *msgp, size_t msgsz,long msgtyp, int msgflg)
----msqid为消息队列的qid
----msgp是接收到的消息将要存放的缓冲区
----msgsz是消息的大小
----msgtyp是期望接收的消息类型
----msgflg是标志
0:表示忽略
IPC_NOWAIT:如果消息队列为空，不阻塞等待，返回一个ENOMSG
----返回
0：成功
-1:消息长度大于msgsz
EACCES：没有该消息队列读权限
EFAULT:消息队列地址无法获取
EIDRM:消息队列已经被删除
EINTR:消息队列等待写入的时候被中断
ENOMEM:内存不够
.函数原理：
1)计算id = (unsigned int) msqid % MSGMNI,然后根据id在msgque[MSGMNI]中查找对应的消息队列，并进行认证检查，合法性检查
2)根据msgtyp搜索消息队列，情况有二：
----如果找不到所要的消息，则以可中断等待状态（TASK_INTERRUPTIBLE）将当前进程挂起在rwait等待队列上
----如果找到所要的消息，则将消息从队列中摘下，调整队列msqid_ds参数，唤醒该消息队列的wwait进程队列上等待写的进程，将消息内容拷贝到用户空间的消息缓冲区msgp中，释放内核中该消息所占用的空间，返回

5.消息的控制

int msgctl (int msqid, int cmd, struct msqid_ds *buf)
------msqid：为消息队列的qid
------cmd：为该函数要对消息队列执行的操作

IPC_STAT：取出消息队列的msqid_ds结构体并将参数存入buf所指向的msqid_ds结构对象中

IPC_SET：设定消息队列的msqid_ds 数据中的msg_perm 成员。设定的值由buf 指向的msqid_ds结构给出。

IPC_EMID：将队列从系统内核中删除。

----buf：消息队列msqid_ds结构体指针

.函数作用

对消息队列进行设置以及相关操作，具体操作由cmd指定。

例子：

```
1. #include <sys/types.h>
2. #include <sys/ipc.h>
3. #include <sys/msg.h>
4. #include <stdio.h>
5. #include <string.h>
6.
7. int main()
8. {
9.     key_t unique_key;
10.    int msgid;
11.
12.    int status;
13.    char str1[]={"test message:hello muge0913"};
14.    char str2[]={"test message:goodbye muge0913"};
15.
16.    struct msgbuf
17.    {
18.    long msgtype;
19.    char msgtext[1024];
20.    }sndmsg,rcvmsg;
21.
22.    if((msgid = msgget(IPC_PRIVATE,0666))==-1)
23.    {
24.    printf("msgget error!\n");
25.    exit(1);
26.    }
27.
28.    sndmsg.msgtype =111;
```

```
29.     sprintf(sndmsg.msgtext,str1);
30.
31.     if(msgsnd(msgid,(struct msgbuf *)&sndmsg,sizeof(str1)+1,0)==-1)
32.     {
33.     printf("msgsnd error!\n");
34.     exit(1);
35.     }
36.
37.     sndmsg.msgtype =222;
38.     sprintf(sndmsg.msgtext,str2);
39.     if(msgsnd(msgid,(struct msgbuf *)&sndmsg,sizeof(str2)+1,0)==-1)
40.     {
41.     printf("msgsnd error\n");
42.     exit(1);
43.     }
44.
45.     if((status = msgrcv(msgid,
(struct msgbuf *)&rcvmsg,80,222,IPC_NOWAIT))==-1)
46.     {
47.     printf("msgrcv error\n");
48.     exit(1);
49.     }
50.
51.     printf("The receved message:%s\n",rcvmsg.msgtext);
52.     msgctl(msgid,IPC_RMID,0);
53.     exit(0);
54. }
```