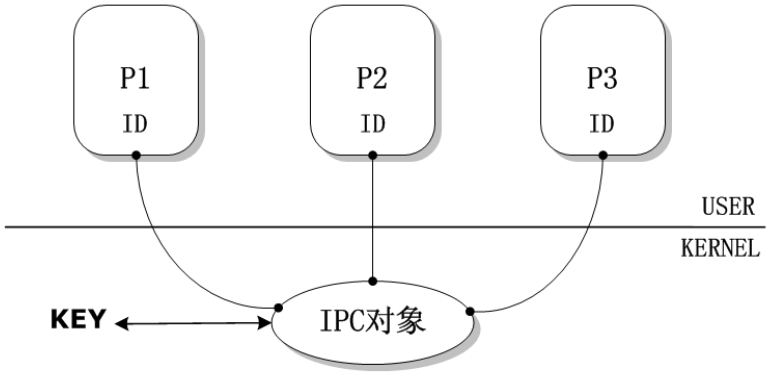
作者：曾任斯

##

1. IPC对象概述

各种不同的IPC其实是在不同时期逐步引入的，在UNIX伯克利版本system-V（念作系统五，V是罗马数字，是Unix伯克利分支的版本号）中引入的三种通信方式（消息队列、共享内存和信号量组）被称为IPC对象，它们有较多共同的特性：

- key：键值，不同的进程，如果需要通信，必须获得相同的key。
  - IPC对象：消息队列、共享内存、信号量统称为IPC对象。
- 在系统中使用所谓键值（KEY）来唯一确定，类似于文件系统中的文件路径。
- 当某个进程创建（或打开）一个IPC对象时，将会获得一个整型ID，类似于文件描述符。
- IPC对象属于系统，而不是进程，因此在没有明确删除操作的情况下，IPC对象不会因为进程的退出而消失。
- 不同进程通过相同键值，获得相同IPC对象的操作权限，从而实现通信。



系统IPC对象

2. IPC对象相关命令

以下命令可以帮助更好了解系统IPC。

2.1 查看IPC对象

```
ipcs        # 查看所有IPC对象
ipcs -a     # 同上
ipcs -q     # 查看消息队列对象
ipcs -m     # 查看共享内存对象
ipcs -s     # 查看信号量对象
```

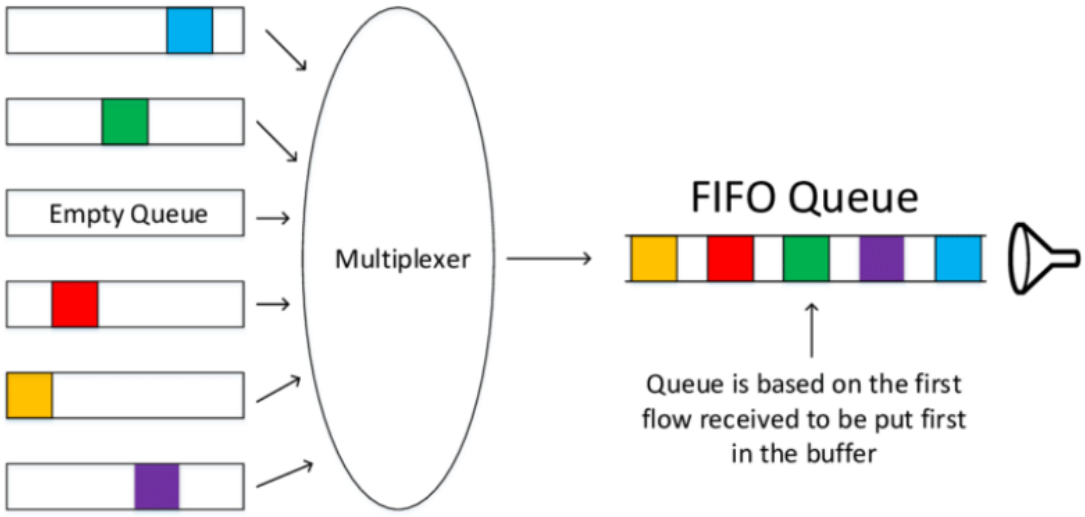
2.2 删除IPC对象

```
ipcrm -Q key : 根据键值key，删除指定的消息队列
ipcrm -q id : 根据ID，删除指定的消息队列

ipcrm -M key : 根据键值key，删除指定的共享内存
ipcrm -m id:  根据ID，删除指定的共享内存

ipcrm -S key : 根据键值key，删除指定的信号量
ipcrm -s id:  根据ID，删除指定的信号量
```

3. 消息队列

3.1 基本逻辑

消息队列是system-V三种IPC对象之一，其最主要的特征是允许发送的数据携带类型，具有相同类型的数据在消息队列内部排队，读取的时候也要指定类型，然后依次读出数据。这使得消息队列用起来就像一个多管道集合，如下图所示：



消息队列基本逻辑

由于每个消息都携带有类型，相同的类型自成一队，因此读取方可以根据类型来“挑选”不同的队列，也因此MSG适用于所谓“多对一”的场景，经典案例是系统日志：多个不同的、不相关的进程向同一管道输入数据。

3.2 函数接口

3.2.1 创建或打开MSG对象

对消息队列的使用非常简单，由如下接口提供：

```
// 创建（或打开）消息队列
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgget(key_t key, int msgflg);
```

- 接口说明
  - 返回值：消息队列MSG对象ID
  - 参数key：键值，全局唯一标识，可由ftok()产生
  - 参数msgflg：操作模式与读写权限，与文件操作函数open类似。

示例代码：

```
int main()
{
    // 以当前目录和序号1为系数产生一个对应的键值
    key_t key = ftok(".", 1);

    // 创建（若存在则报错）key对应的MSG对象
    int msgid = msgget(key, IPC_CREAT|IPC_EXCL|0666);
}
```

注意：
key实质上就是一个整数，但该整数一般应由 ftok() 产生而不应手写，因为key作为键值是 IPC 对象在系统中的唯一标识，万一误撞就会导致错乱。

ftok()的接口有时也容易使人糊涂：

```
#include <sys/types.h>
#include <sys/ipc.h>
key_t ftok(const char *pathname/*路径*/, int proj_id/*序号*/);
```

- 对于ftok()函数参数，首先需要说明的一点是，路径和序号一样的情况下，产生的键值key也是一样的。那么，由于项目开发中，需要互联互通的进程一般会放在同一目录下，而其他无关的进程则不会放在一起，一起使用路径来产生键值是有效避免键值误撞的手段，序号是为了以防在某路径下需要产生多个IPC对象的情况。
- 最后需要再重申一点的是，ftok()函数参数中的路径仅仅是产生键值key的参数，与实际文件系统并无关系。
- 若 msgget() 中的key写成 IPC_PRIVATE，那意味着新建一个私有的IPC对象，该对象只在本进程内部可见，与外部的系统MSG对象不会冲突。

3.2.2 向MSG对象发送消息

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
```

接口说明：

- msgqid：MSG对象的ID，由msgget()获取。
- msgp：一个指向等待被发送的消息的指针，由于MSG中的消息最大的特点是必须有一个整数标识，用以区分MSG中的不同的消息，因此MSG的消息会使用一个特别的结构体来表达，具体如下所示：

```
struct msgbuf
{
    // 消息类型（固定）
    long mtype;

    // 消息正文（可变）
    // ...
};
```

因此一般而言，msgp就是一个指向上述结构体的指针。

- msgsz：消息正文的长度（单位字节），注意不含类型长度。
- msgflg：发送选项，一般有：
    - 0：默认发送模式，在MSG缓冲区已满的情形下阻塞，直到缓冲区变为可用状态。
    - IPC_NOWAIT：非阻塞发送模式，在MSG缓冲区已满的情形下直接退出函数并设置错误码为EAGAIN.

示例代码：

```
struct message
{
    long mtype;
    char mtext[80];
};

int main(void)
{
    int msgid;
    msgid = msgget(ftok(".", 1), IPC_CREAT | 0666);

    struct message msg;
    bzero(&msg, sizeof(msg));

    // 消息类型
    msg.mtype = 1;
    // 消息内容
    fgets(msg.text, 80, stdin);

    // 发送消息
    msgsnd(msgid, &msg, strlen(msg.mtext), 0);
}
```

3.2.3 从MSG对象接收消息

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp, int msgflg);
```

接口说明：

- msgqid：MSG对象的ID，由msgget()获取。

- msgp：存放消息的内存入口。
- msgsz：存放消息的内存大小。
- msgtyp：欲接收消息的类型：
  - 0：不区分类型，直接读取MSG中的第一个消息。
  - 大于0：读取类型为指定msgtyp的第一个消息（若msgflg被配置了MSG_EXCEPT则读取除了类型为msgtyp的第一个消息）。
  - 小于0：读取类型小于等于msgtyp绝对值的第一个具有最小类型的消息。例如当MSG对象中有类型为3、1、5类型消息若干条，当msgtyp为-3时，类型为1的第一个消息将被读取。
- msgflg：接收选项：
  - 0：默认接收模式，在MSG中无指定类型消息时阻塞。
  - IPC_NOWAIT：非阻塞接收模式，在MSG中无指定类型消息时直接退出函数并设置错误码为ENOMSG.
  - MSG_EXCEPT：读取除msgtyp之外的第一个消息。
  - MSG_NOERROR：如果待读取的消息尺寸比msgsz大，直接切割消息并返回msgsz部分，读不下的部分直接丢弃。若没有设置该项，则函数将出错返回并设置错误码为E2BIG。

示例代码：

```
struct message
{
    long mtype;
    char mtext[80];
};

int main(void)
{
    int msgid;
    msgid = msgget(ftok(".", 1), IPC_CREAT | 0666);

    struct message msgbuf;
    bzero(&msgbuf, sizeof(msgbuf));

    printf("等待消息...\n");
    int m = msgrcv(msgid, &msgbuf, sizeof(msgbuf)-sizeof(long), 1, 0);

    if(m < 0)
        perror("msgrcv()");
    else
        printf("%s\n", msgbuf.text);

    msgctl(msgid, IPC_RMID, NULL);
    return 0;
}
```

3.2.4 对MSG对象其余操作

IPC对象是一种持久性资源，如果没有明确的删除掉他们，他们是不会自动从内存中消失的，除了可以使用命令的方式删除，可以使用函数来删除。比如，要想显式地删除掉MSG对象，可以使用如下接口：

```c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

接口说明：

- msqid：MSG对象ID
- cmd：控制命令字
  - IPC_STAT：获取该MSG的信息，储存在结构体msqid_ds中
  - IPC_SET：设置该MSG的信息，储存在结构体msqid_ds
  - IPC_RMID：立即删除该MSG，并且唤醒所有阻塞在该MSG上的进程，同时忽略第三个参数

在程序中如果不再使用MSG对象，为了节省系统资源，应用如下代码删除：

```c
msgctl(id, IPC_RMID, NULL);
```

注意：
管道打开时，必须同时有读者和写者，否则 open 也会阻塞。

「课堂练习」

- 编写发送代码 send.c，使其能够循环发送数据。
- 编写接收代码 recv.c ，使其能够循环接收。
  - 当接收到 “exit” 数据时，使用 msgctl函数将消息队列删除。
    （ipcs可以查看是否已删除。）