进程间通信基本背景:

1. 进程之间具有独立性,进程和进程相互沟通配合就不太方便
2. 要想完成进程间通信就需要有一个公共的资源让多个进程都能访问到

进程间通信分类:

1. 管道 是操作系统内核中的一块内存,使用时当作函数,**匿名管道**

pipe() 创建匿名管道

**匿名管道 特点:**

1. 只能用于具有共同祖先(有血缘关系)的进程之间的通信

eg; ps aux | grep hehe ps grep都是bash进程 是兄弟关系

2) 管道是单向读取数据的(半双工的)

3) 进程退出,管道释放,所以管道周期随进程

4) 内核对管道操作进行同步与互斥(没有就阻塞(等待),写满了就不写了,读完了就不读了)

5) 管道提供流式服务(面向字节流,即:读写灵活,不需完全匹配)

eg;

int fd[2] ={0};

int ret=pipe(fd);// 调用成功就会返回一对文件描述符

//其中fd[0]用来读 fd[1]用来写

//加上linux中一切皆文件思想

//就通过这一对文件描述符来操作管道对应的内存

if(ret<0){

perror(“pipe”);

return 1;

}

**命名管道**

创建命名管道： mkfifo [文件名]

EOF：结束标记符

ctrl+z 把进程从前台变成后台（在windows系统中为结束标记符）

fg 把进程从后台变为前台

ctrl+d 输出结束标记符

命名管道特点：

a)单项通信

b)命名管道可以用于随意进程

c)生命周期随进程

1. **system V(Unix中的某个版本,一套标准)进程间通信**

**消息队列**

消息队列本质上也是一个队列,队列中的元素具备数据类型(业务类型),并且不是严格的先进先出(是指定类型的先进先出),消息队列的生命周期不随进程,而是随内核(手动删掉或关闭操作系统,消息队列才会退出),消息队列是双向通行的

通过相同的IPC\_key才能访问相同的消息队列

**IPC对象数据结构**

struct ipc\_perm{

key\_t //身份标示符

}

**消息队列结构**

struct msqid\_ds{

}

客户端进程:主动读取数据

1. 从标准输入读入数据
2. 把数据写到消息队列中
3. 从消息队列中华读取数据
4. 把结果打印到显示器上

服务器进程:被动的一方

1. 从消息队列中读取数据
2. 根据读到的数据进行计算,生成响应的数据
3. 把响应的数据写回到消息队列

IPC\_CREAT 如果消息队列不存在就创建 如果存在就打开

IPC\_EXCL 必须配合IPC\_CREAT使用,不能单独使用 含义就是如果消息队列存在,打开就会失败

eg: key\_t key= ftok(FILEPATH,PROJ\_ID);// 构建消息队列

if (key < 0){

perror (“ftok”);

}

eg:销毁消息队列 -->IPC\_RMID

int Destorymsg(int msgid){

int ret=msgctrl(msgid,IPC\_RMID,NULL)

if(ret<0){

perror (“RMID”);

}

}

将msgid看作消息队列的句柄.句柄:操控消息队列的”遥控器”

**命令:ipcs \_q** 查看当前操作系统的消息队列

1. **POSIX(也是另一套标准)进程间通信**