# 实验七　进程通信

## 基本信息

**实验课程：**Linux程序设计 **设课形式：**非独立

**课程学分：**2 **实验项目：**进程通信

**项目类型：**验证 **项目学时：**2

## 实验预习

**实验目的和要求：**

进程通信中信号概念及信号处理进程间的管道通信编程进程间的内存共享编程

**实验条件：**

1、装有Linux操作系统的微型计算机；

## 实验过程

7.1信号及处理

**7.1.1 信号及其使用**

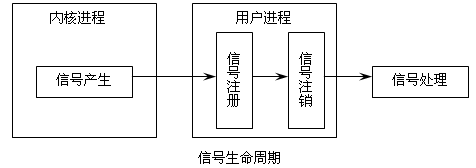
* 信号是在软件层次上对中断机制的一种模拟，是一种异步通信方式。信号可以直接进行用户空间进程和内核进程之间的交互，内核进程也可以利用它来通知用户空间进程发生了哪些系统事件。
* 信号事件的发生有两个来源：
  1. 硬件来源，如按下了键盘Delete键或者鼠标单击，通常产生中断信号（SIGINT）或者其它硬件故障。
  2. 软件来源，如使用系统调用或者是命令发出信号。最常用发送信号的系统函数是kill、raise、alarm、setitimer、sigation和sigqueue函数，软件来源还包括一些非法运算等操作。

**例7.1：**列出系统所支持的所有信号列表。

（1）使用系统命令：[root@localhost root]#**kill -l**

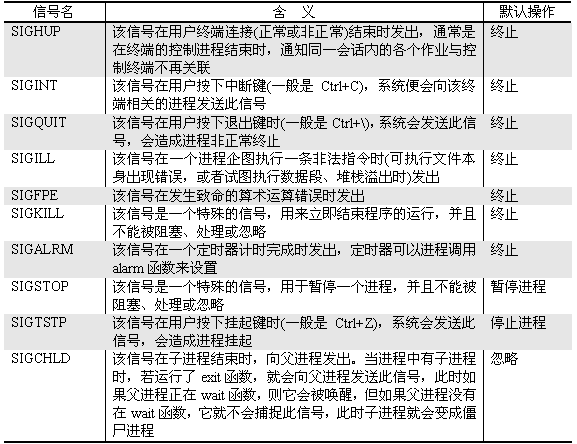
（2）分析：

* + SIG信号
  + SIGRTMIN信号是从UNIX系统中继承下来的称为不可靠信号(也称为非实时信号)。
  + SIGRTMAX是为了解决前面“不可靠信号”问题而进行更改和扩充的信号，称为可靠信号(也称为实时信号)。
  + 可靠信号(实时信号)：支持排队，发送用户进程一次就注册一次，发现相同信号已经在进程中注册，也要再注册。
  + 不可靠信号(非实时信号)：不支持排队，发送用户进程判断后注册，发现相同信号已经在进程中注册，就不再注册，忽略该信号。前面显示的31种“SIG”开头的，也属于非实时信号。



* 一旦有信号产生，用户进程对信号的响应有3种方式：
  1. 执行默认操作。Linux对每种信号都规定了默认操作。
  2. 捕捉信号。定义信号处理函数，当信号发生时，执行相应的处理函数。
  3. 忽略信号。不希望接收到的信号对进程的执行产生影响，而让进程继续进行时，可以忽略该信号，即不对信号进程任何处理。

7.1.2常见信号的含义及其默认操作



**7.1.3 信号操作的相关函数**

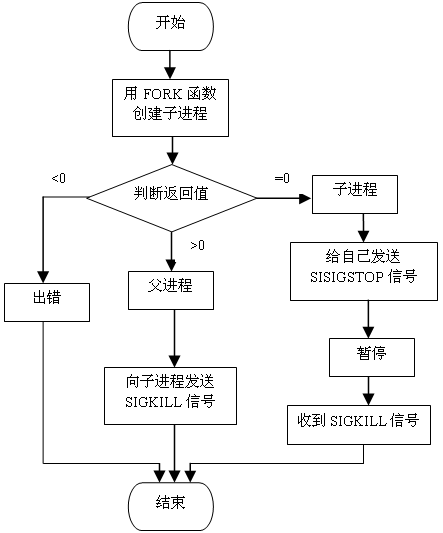
****

**1. 信号发送**

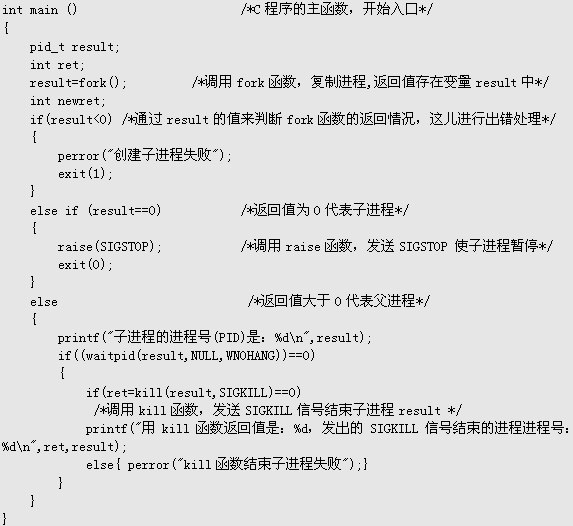
**信号发送的关键，是使系统知道向哪个进程发送以及发送什么信号。能否向某一进程发送某一特定信号是和用户的权限密切相关的。**

**例7.2：设计一个程序，要求用户进程复制出一个子进程，父进程向子进程发出信号，子进程收到此信号，结束子进程。**

* **源程序代码：**

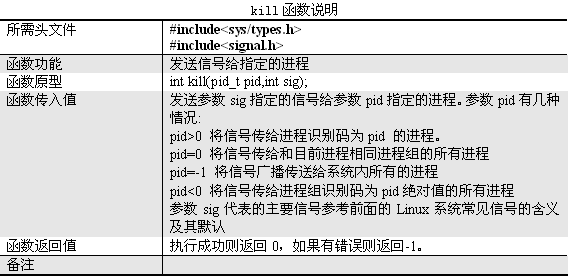
****

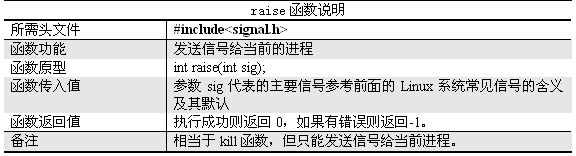
**程序7.2流程图**

****

**编译成功后，运行可执行文件，此时系统会显示子进程的进程号（PID）、kill 函数的返回值和SIGKILL信号所结束进程的进程号(PID)。**

**由此例可知，系统调用kill函数和raise函数，都是简单地向某一进程发送信号。kill函数用于给特定的进程或进程组发送信号，raise函数用于向一个进程自身发送信号。**

****

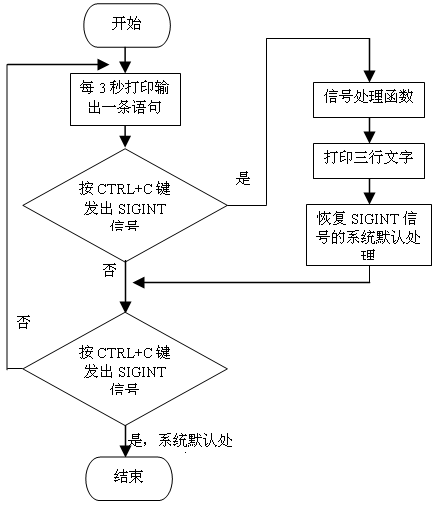
****

**2. 信号处理**

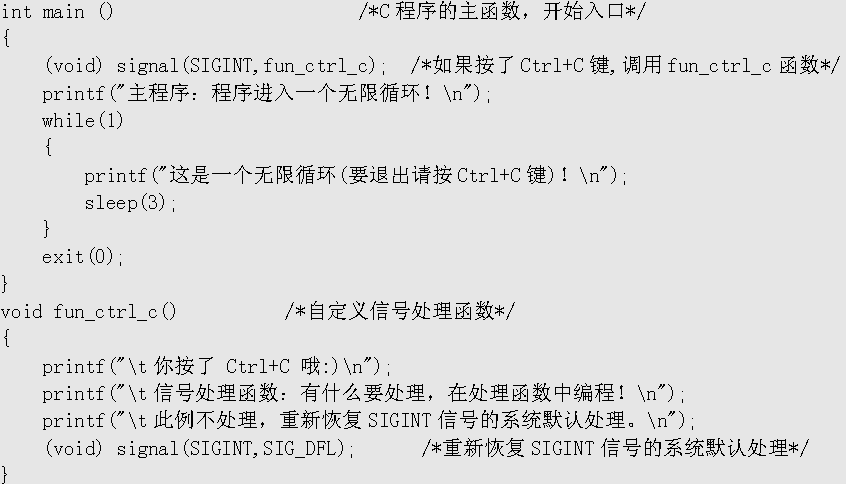
**当某个信号被发送到一个正在运行的进程时，该进程即对此特定信号注册相应的信号处理函数，以完成所需处理。**

**例7.3：设计一个程序，要求程序运行后进入无限循环，当用户按下中断键(Ctrl+C)时，进入程序的自定义信号处理函数，当用户再次按下中断键(Ctrl+C)后，结束程序运行。**

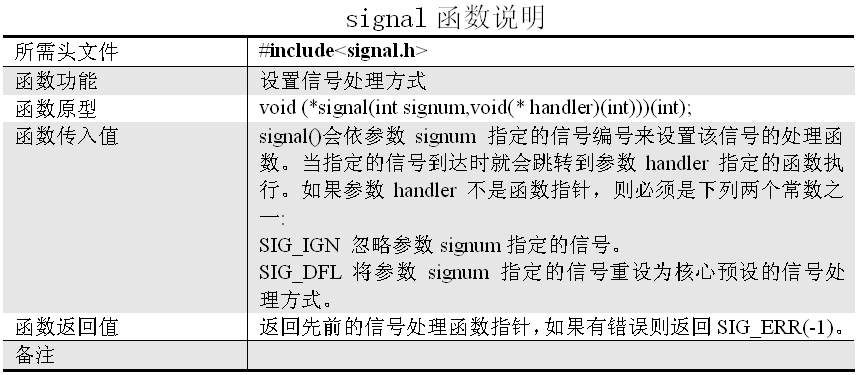
* **源程序代码 ：**

****

**程序7.3流程图**

****

**signal函数主要用于前31种非实时信号的处理，不支持信号传递信息(函数类型是void)，但使用简单、方便，只需把要处理的信号和处理函数列出即可，因此受到许多软件工程师欢迎。**

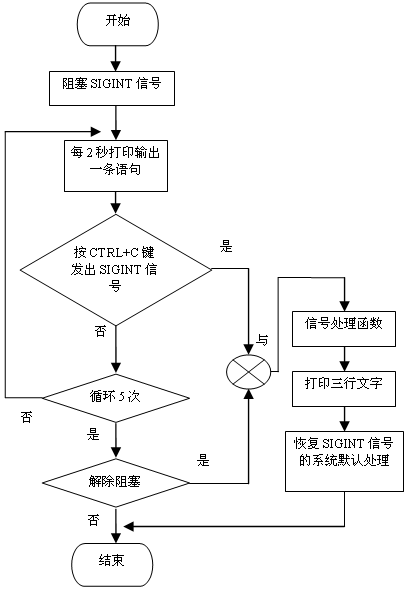
****

**3. 信号阻塞**

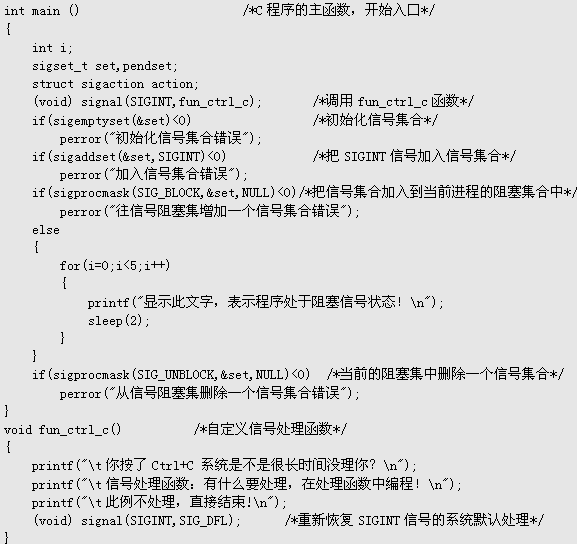
**有时既不希望进程在接收到信号时立刻中断进程的执行，也不希望此信号完全被忽略掉，而是延迟一段时间再去调用信号处理函数，这个时候就需要信号阻塞来完成。**

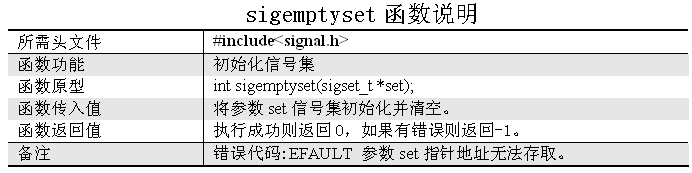
**例7.4：设计一个程序，要求程序主体运行时，即使用户按下的中断键(Ctrl+C)，也不能影响正在运行的程序，等程序主体运行完毕后才进入自定义信号处理函数**

* **源程序代码 ：**

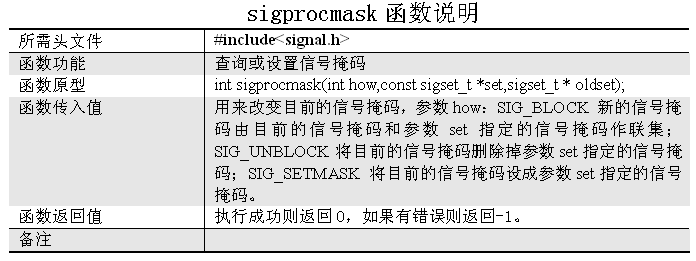
****

**程序7.4流程图**

****

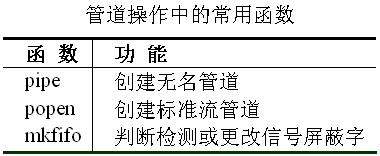
****

****

****

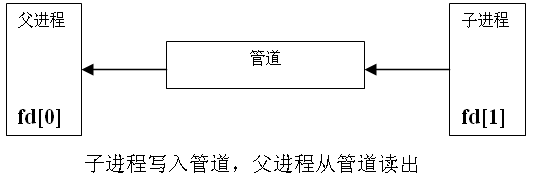
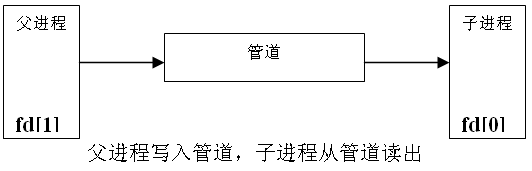
7.2 管道

* **在Linux中，管道是一种特殊的文件，对一个进程来说，管道的写入和读取与一个普通文件没有区别。**
* **在Linux系统中，管道用于两个进程间的通信，这两个进程要有同源性，即它们必须是最终由同一个进程所生成的进程。管道通信采用的是半双工方式，即同一时间只允许单方向传输数据。**
* **管道是Linux支持的最初[Unix](http://school.21tx.com/os/unix/) IPC形式之一，具有以下特点：** 
  1. **管道是半双工的，数据只能向一个方向流动；需要双方通信时，需要建立起两个管道；**
  2. **只能用于父子进程或者兄弟进程之间（具有亲缘关系的进程）；**
  3. **单独构成一种独立的文件系统：管道对于管道两端的进程而言，就是一个文件，但它不是普通的文件，它不属于某种文件系统，而是自立门户，单独构成一种文件系统，并且只存在与内存中。**

****

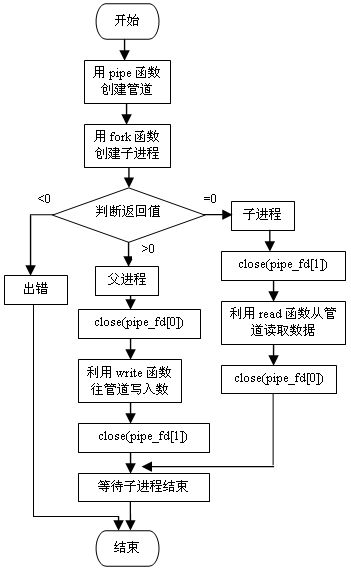
**7.2.1 低级管道操作**

* **低级管道操作时，建立管道用pipe函数，建立管道后Linux系统会同时为该进程建立2个文件描述符pipe\_fd[0]和pipe\_fd[1]。pipe\_fd[0]用来从管道读取数据，pipe\_fd[1]用来把数据写入管道。**

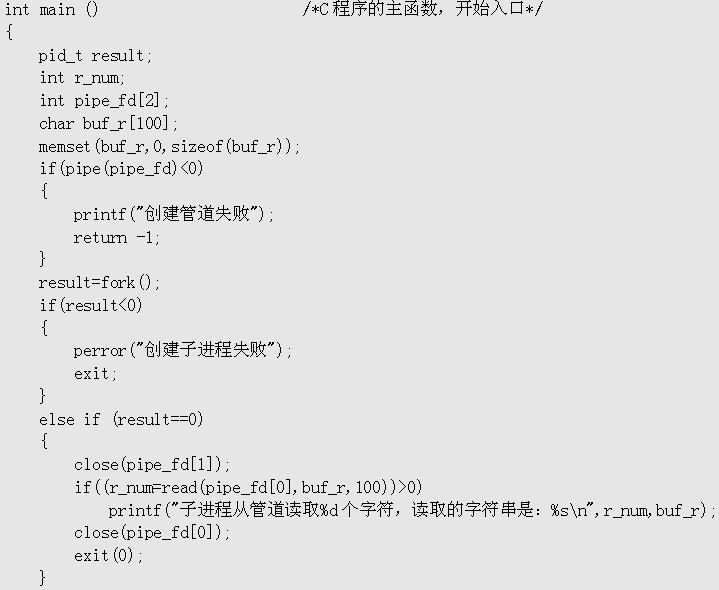
****

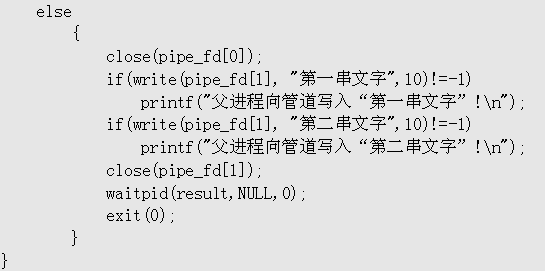
**例7.5：设计一个程序，要求创建一个管道，复制进程，父进程往管道中写入字符串，子进程从管道中读取前输出**

* **源程序代码：**

****

**程序7.5流程图**

****

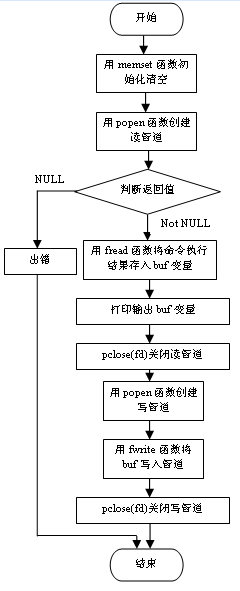
****

****

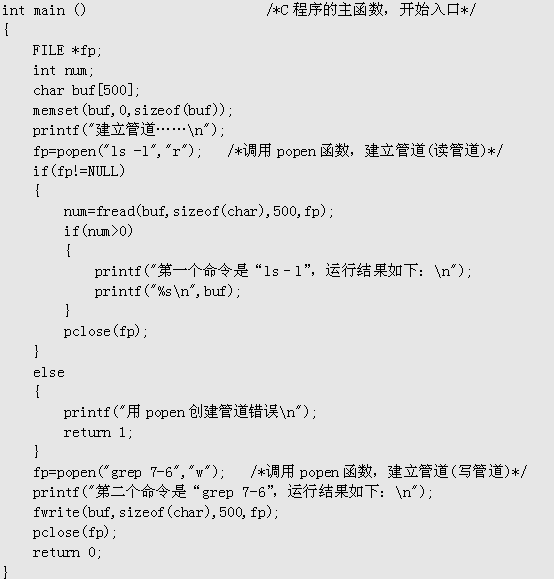
* + 1. **高级管道操作**

**例7.6：设计一个程序，要求用popen创建管道，实现“ls -l|grep 7-6”的功能。**

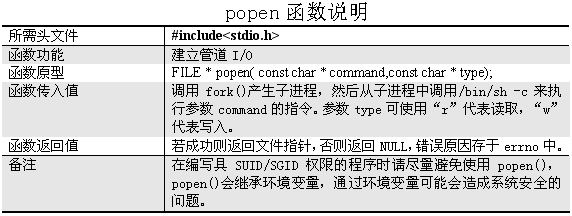
* **源程序代码 :**

****

**程序7.6流程图**

****

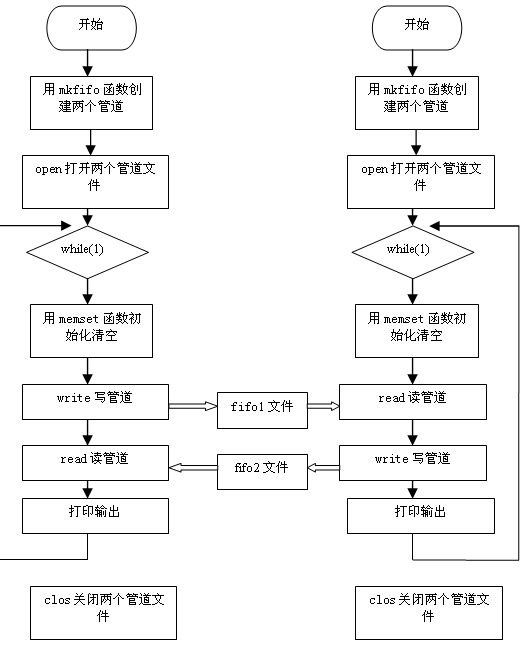
**使用popen函数读写管道，实际上也是调用pipe函数建立一个管道，再调用fork函数建立子进程，接着会建立一个shell环境，并在这个shell环境中执行参数指定的进程。**

****

**7.2.3 命名管道**

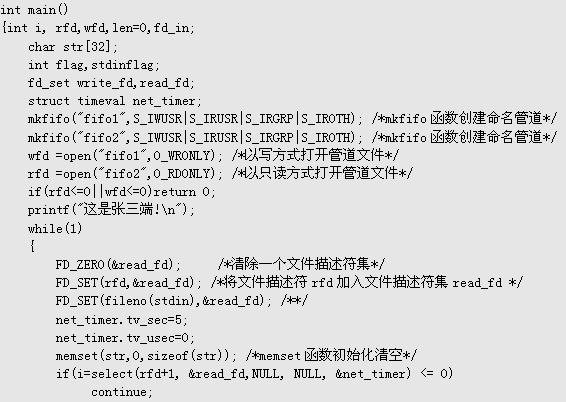
* **若要在两个不相关的进程之间用管道通信，需要用到命名管道FIFO。**
* **命名管道FIFO是通过Linux系统中的文件进行通信。命名管道的创建一般用mkfifo函数，创建成功后，就使用open、read、write等函数传输数据。**

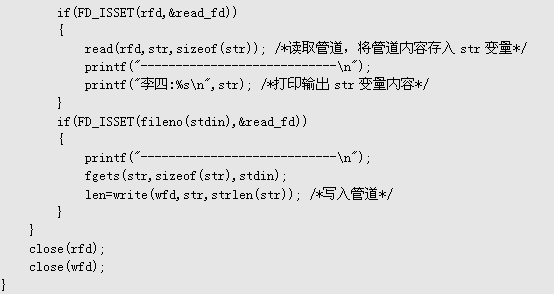
**例7.7：设计两个程序，要求用命名管道FIFO，实现简单的聊天功能。**

****

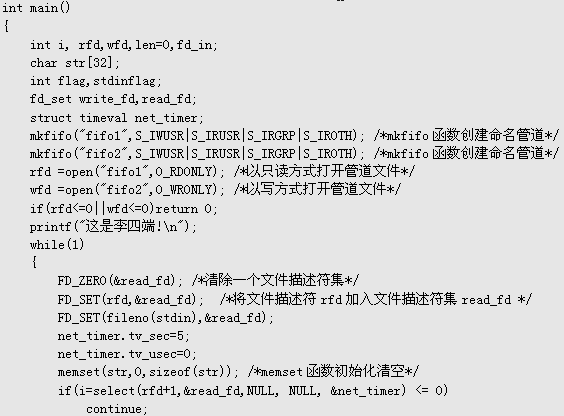
**程序7.7流程图**

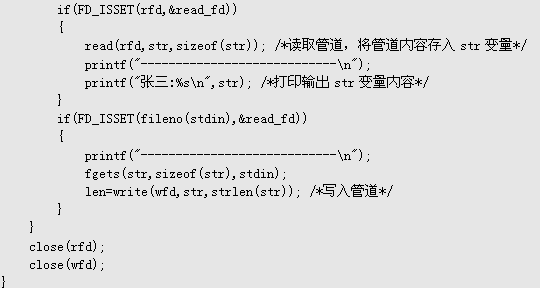
* **源程序7-7zhang.c代码:**

****

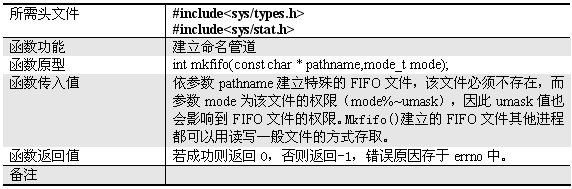
****

* **7-7li.c程序代码如下：**

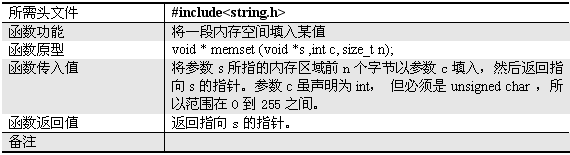
****

****

* **mkfifo函数说明**

****

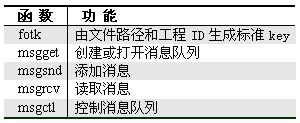
* **memset函数说明**

****

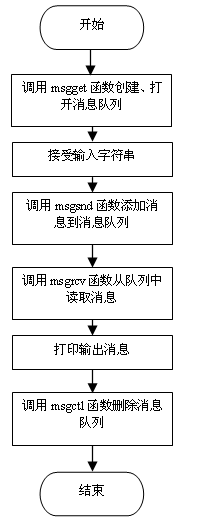
7.3 消息队列

* **消息队列，就是一个消息的链表，是一系列保存在内核中的消息的列表。用**
* **消息队列的优势：对每个消息指定特定消息类型，接收的时候不需要按队列次序，而是可以根据自定义条件接收特定类型的消息。**
* **可以把消息看作一个记录，具有特定的格式以及特定的优先级。对消息队列有写权限的进程可以按照一定的规则添加新消息；对消息队列有读权限的进程则可以从消息队列中读取消息。**

**消息队列的常用函数**

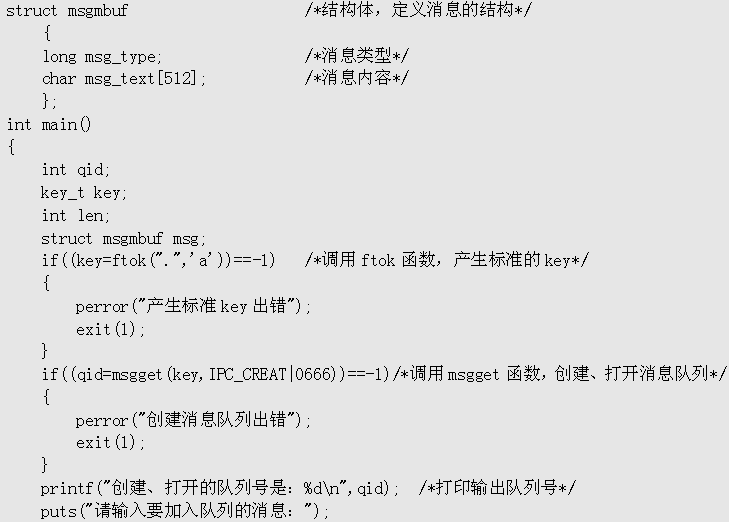
****

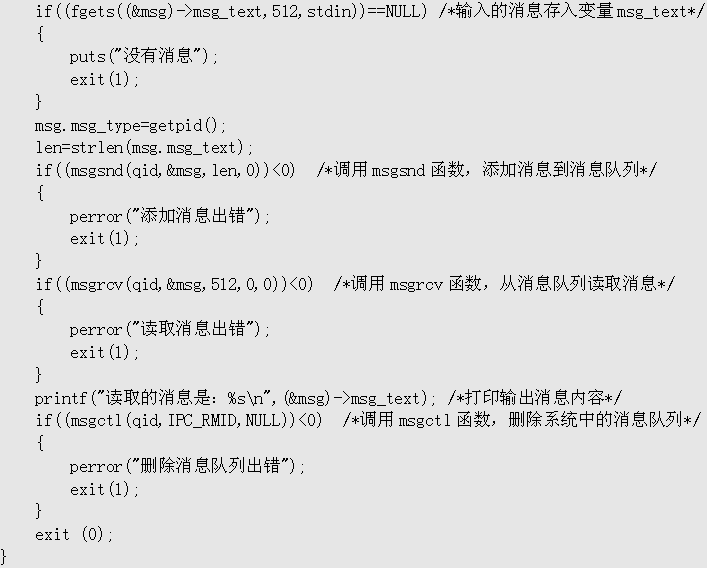
**例7.8：设计一个程序，要求创建消息队列，输入的文字添加到消息队列后，读取队列中的消息输出。**

****

**程序7.8流程图**

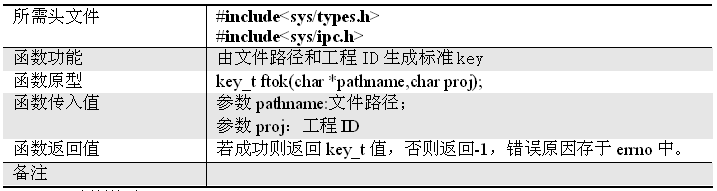
* **源程序代码：**

****

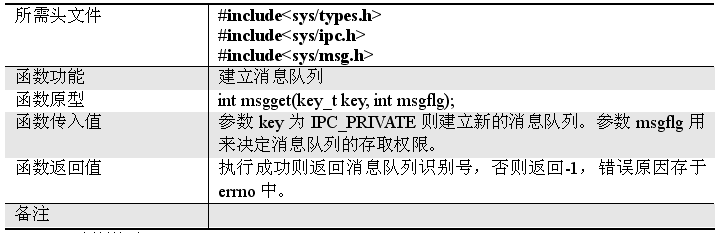
****

**由此例可知，进程间通过消息队列通信，主要是创建或打开消息队列、添加消息、读取消息和控制消息队列这四种操作。**

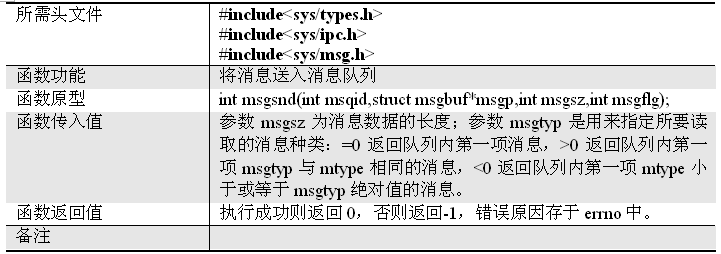
* **ftok函数说明**

****

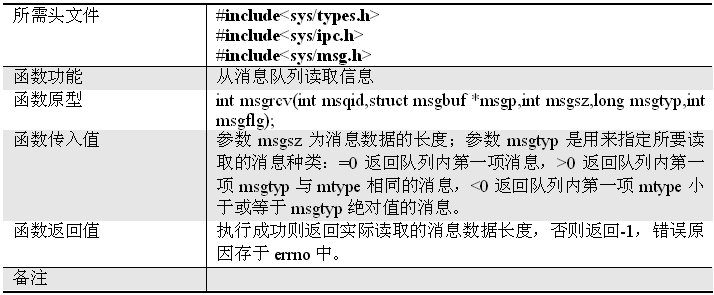
* **msgget函数说明**

****

* **msgsnd函数说明**

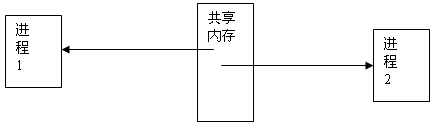
****

* **msgrcv函数说明**

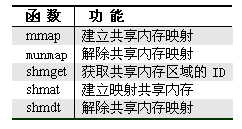
****

7.4 共享内存

* **共享内存允许两个或多个进程共享一给定的存储区，因为数据不需要来回复制，所以是最快的一种进程间通信机制。**
* **共享内存原理：**

****

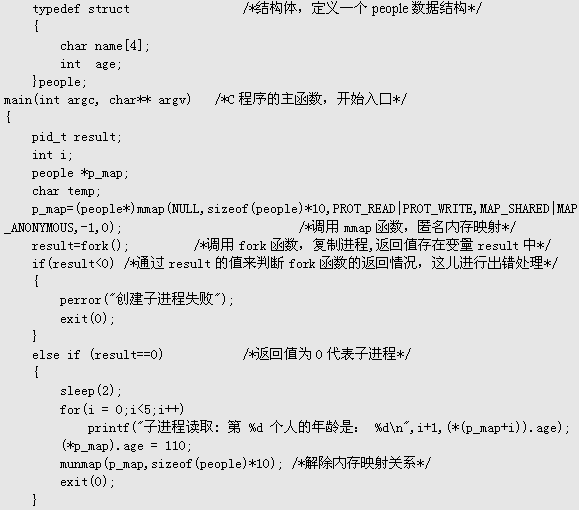
* **共享内存可以通过mmap()系统调用（特殊情况下还可以采用匿名映射）机制实现，也可以通过系统V共享内存机制实现。应用接口和原理很简单，内部机制复杂。为了实现更安全通信，往往还与信号灯等同步机制共同使用。**
* **常用函数 ：**

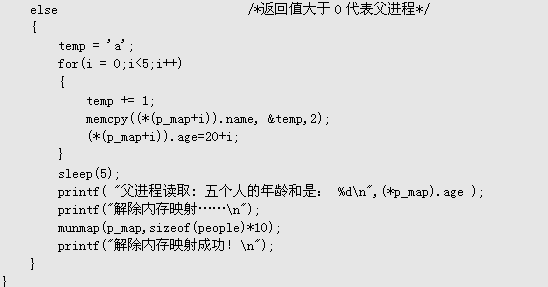
****

* + 1. **mmap系统调用**

**例7.9：设计一个程序，要求复制进程，父子进程通过匿名映射实现共享内存。**

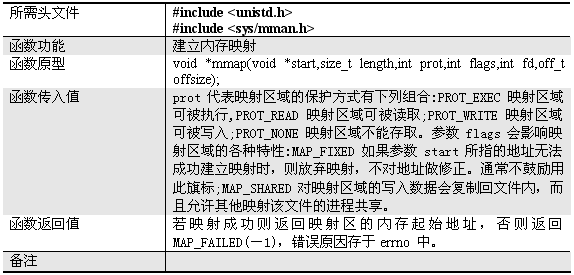
* **源程序代码：**

****

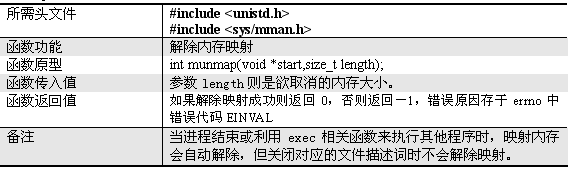
****

**使用特殊文件提供匿名内存映射，适用于具有亲缘关系的进程之间。一般而言，子进程单独维护从父进程继承下来的一些变量。而mmap函数的返回地址，由父子进程共同维护。**

* **mmap函数说明**

****

* **munmap函数说明**

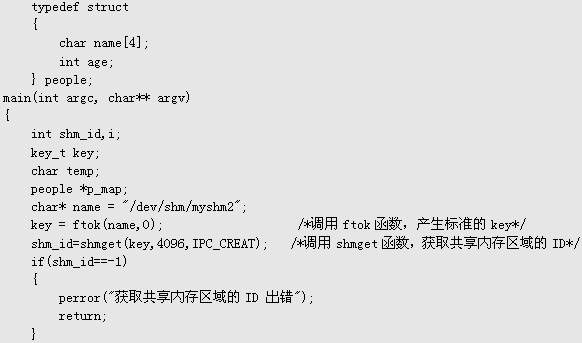
****

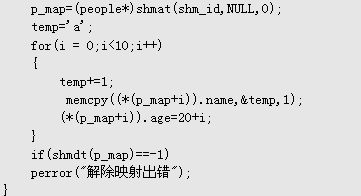
**7.4.2 系统V共享内存**

* **系统V共享内存指的是把所有共享数据放在共享内存区域（IPC shared memory region），任何想要访问该数据的进程都必须在本进程的地址空间新增一块内存区域，用来映射存放共享数据的物理内存页面。**
* **系统V共享内存是通过映射特殊文件系统shm中的文件实现进程间的共享内存通信。**

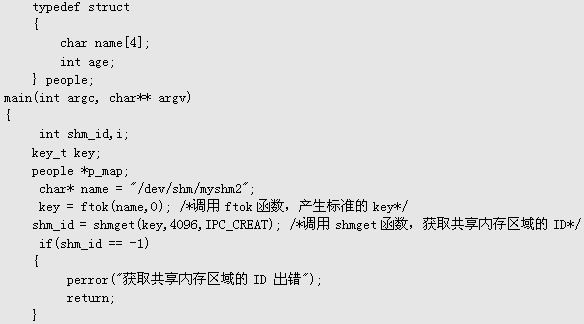
**例7.10：设计两个程序，要求通过系统V共享内存通信，一个程序写入系统V共享区域，另一个程序读取系统V共享区域。**

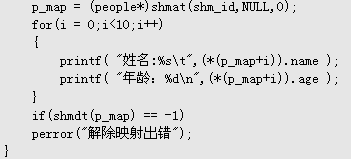
* **7-10write.c程序代码：**

****

****

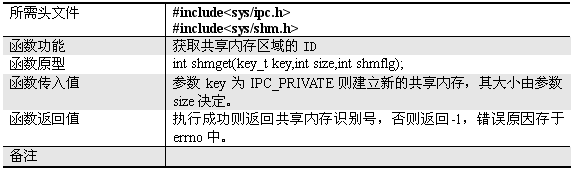
* **7-10read.c程序代码：**

****

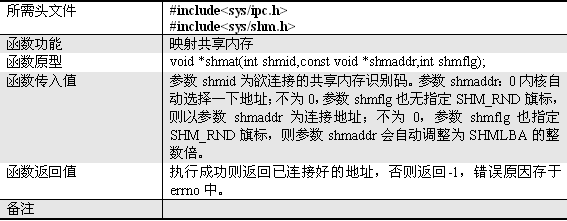
****

**结论：**

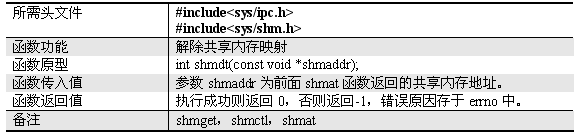
* 1. **系统V共享内存中的数据，从来不写入到实际磁盘文件中去；而通过mmap()映射普通文件实现的共享内存通信可以指定何时将数据写入磁盘文件中。**
  2. **系统V共享内存是随内核持续的，即使所有访问共享内存的进程都已经正常终止，共享内存区仍然存在（除非显式删除共享内存），在内核重新引导之前，对该共享内存区域的任何改写操作都将一直保留。**
  3. **通过调用mmap()映射普通文件进行进程间通信时，一定要注意考虑进程何时终止对通信的影响。而通过系统V共享内存实现通信的进程则不然。**
* **shmget函数说明**

****

* **shmat函数说明**

****

* **shmdt函数说明**

****

## 思考与实验

1. 设计一个程序，要求程序运行后进入一个无限循环，当用户按下中断键(Ctrl+Z)时，进入程序的自定义信号处理函数，当用户再次按下中断键(Ctrl+Z)后，结束程序运行。
2. 设计一个程序，要求程序主体运行时，即使用户按下的中断键(Ctrl+C)，也不能影响正在运行的程序，等程序主体运行完毕后才进入自定义信号处理函数。
3. 设计一个程序，要求创建一个管道PIPE，复制进程，父进程运行命令“ls -l”，把运行结果写入管道，子进程从管道中读取“ls -l”的结果，把读出的作为输入接着运行“grep 7-5”。
4. 设计两个程序，要求用命名管道FIFO，实现简单的文本文件或图片文件的传输功能。
5. 设计两个程序，要求用消息队列，实现聊天程序，每次发言后自动在后面增加当前系统时间。增加结束字符，比如最后输入“88”后结束进程。
6. 设计两个程序，要求用mmap系统，实现简单的聊天程序。

## 实验结果分析

1、熟悉并记录程序执行结果。

2、写出自己的心得体会。