# 实验六　进程控制

## 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：**进程控制

**项目类型：**验证 **项目学时：**2

## 实验预习

**实验目的和要求：**

进程的基本概念及进程的结构Linux环境下进程的相关函数的应用守护进程的概念、启动和建立进程控制程序的编写

**实验条件：**

1、装有Linux操作系统的微型计算机；

## 实验过程

**6.1进程简介**

* 进程是一个程序的一次执行的过程。
* 在Linux环境下，每个正在运行的程序都称为进程。
* 每个进程包含进程标识符及数据，这些数据包含进程变量、外部变量及进程堆栈等。

**1. 进程与程序**

* 由于一个进程对应一个程序的执行，但进程不等同于程序。因为程序是静态的概念，进程是动态的概念。
* 进程是程序执行的过程，包括了动态创建、调度和消亡的整个过程。进程是程序执行和资源管理的最小单位。
* 对系统而言，当用户在各级系统中键入命令执行一个程序的时候，它将启动一个进程，因此，一个程序可以对应多个进程。

**2. Linux环境下的进程管理**

* Linux环境下的进程管理包括启动进程和调度进程。
* 启动进程有两种主要途径：手工启动和调度启动。

（1）手工启动又可分为前台启动和后台启动。

* 前台启动：是手工启动一个进程的最常用方式。一般地，当用户输入一个命令时，就已经启动了一个进程，并且是一个前台的进程。
* 后台启动：往往是在该进程非常耗时，且用户也不急着需要结果的时候启动。一般地，当用户输入一个命令结尾加上一个“&”号，就是后台启动一个进程。

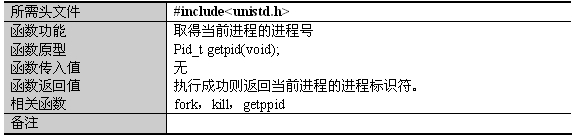
（2）调度启动

* 有时系统需要进行一些比较费时而且占用资源的维护工作，并且这些工作适合在深夜无人职守的时候进行，这时用户就可以事先进行调度安排，指定任务运行的时间或者场合，到时候系统就会自动完成这一切工作。
* 调度进程包括对进程的中断操作、改变优先级、查看进程状态等。
* Linux环境下常见的进程调用命令

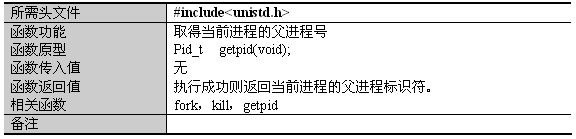


**6.2 Liunx进程控制**

* Linux环境下在进程启动时，系统会分配一个唯一的数值给每个进程，这个数值就称为进程标识符。
* 在Linux中最主要的进程标识有进程号（PID）和它的父进程号（PPID）。
  1. PID惟一地标识一个进程。
  2. PID和PPID都是非零的正整数。
  3. 在Linux中获得当前进程的PID和PPID的系统调用函数为getpid和getppid函数。
* getpid函数说明

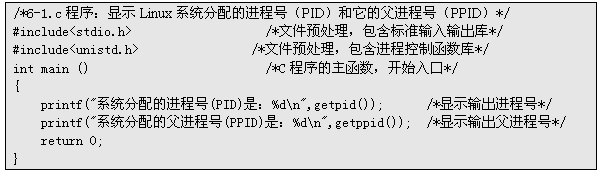


* getppid函数说明



**例6-1：**设计一个程序，要求显示Linux系统分配给此程序的进程号（PID）和它的父进程号（PPID）。

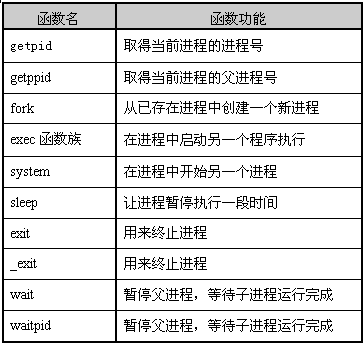
* 源程序代码:



**多次运行例6.1的程序，每一次运行的结果PID值都是不一样的，所以说PID是惟一地标识一个进程。**

**6.2.1 进程的相关函数**

* Linux c与进程相关的主要函数



**6.2.2 进程创建**

**1. exec函数**

**例6-2：**设计一个程序，程序在运行时，能执行vim程序，即创建一个新的进程，并用ps命令查看程序的进程号与vim的进程号。

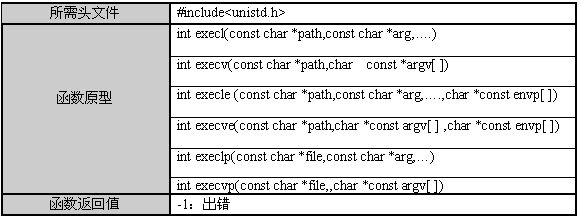
* 源程序代码：

****

* 编译、运行程序，系统会出现运行结果，先显示Linux系统分配的进程号（PID），接着运行vim程序，创建新的进程。
* 再打开一个终端，用ps查看原进程和新创建进程的进程号(PID)。
* 可以看到，在新进程创建后，原来的进程已经终止了。
* 在用execve函数创建新进程后，会以新的程序取代原来的进程，然后系统会从新进程运行，但是新进程的PID值会与原来进程的PID值相同。一般情况下，在运行execve函数后是不会返回原进程的，只有在错误时才会返回-1，所以在原进程中的execve函数下方，加入perror函数，输出错误信息，并返回1，表示有错误发生。

**注意：在使用exec族时，一定要加上错误判断语句，因为exec很容易执行失败。**

* 实际上，在Linux中并没有exec函数，而是有6个以exec开头的函数族。
* exec函数族的6个成员函数的语法

****

**事实上，这6个函数中真正的系统调用只有execve，其他5个都是库函数，它们最终都会调用execve这个系统调用。**

* **思考题1：execv函数的应用，要在程序中执行命令:ps -ef,命令ps在"/bin"目录下。在这一函数中，参数v表示参数传递(含命令)为构造指针数组方式：**

**char \*arg[]={"ps","-ef",NULL};**

**函数的使用为：**

**execv("/bin/ps",arg);**

**参考程序：**

**#include<stdio.h> /\*文件预处理，包含标准输入输出库\*/**

**#include<unistd.h> /\*文件预处理，包含getpid、getppid函数库\*/**

**int main () /\*C程序的主函数，开始入口\*/**

**{**

**char \*arg[]={"ls","-al",NULL};**

**execv("/bin/ls",arg);**

**return 1;**

**}**

* **思考题2：execlp 函数的应用，要在程序中执行命令:ps -ef,命令ps在"/bin"目录下。在这一函数中，参数l表示命令或参数逐个列举，参数p为文件查找方式（不需要给出路径）。因而此函数的调用形式为：**

**execlp("ps","ps","-ef",NULL);**

**请编写一程序进行调试。**

* **思考题3：execl 函数的应用，要在程序中执行命令:ps -ef,命令ps在"/bin"目录下。在这一函数中，参数l表示命令或参数逐个列举，文件需给定路径。因而此函数的调用形式为：execl("/bin/ps","ps","-ef",NULL);**

**请编写一程序进行调试。**

**2. system函数**

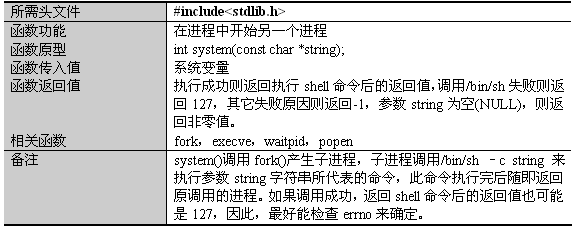
* **system函数是一个和操作系统紧密相关的函数。用户可以使用它在自己的程序中调用系统提供的各种命令。**
* **使用时不需要预处理头文件“unistd.h”。**

**例6-3：设计一个程序，要求测试到LUPA社区的网络连通状况。**

* **编辑源程序代码：**

****

* **编译、运行程序，系统会出现运行结果，先显示Linux系统分配的进程号（PID），接着运行ping程序，创建新的进程 。**
* **再打开一个终端，用ps查看原进程和新创建进程的进程号(PID) 。**
* **可以看到，原来6-3的进程 (PID)值和新进程的父进程号(PPID)值相同，在新进程创建后，原来的进程并没有终止。**
* **注意：在第二个终端的时候，第一个终端中的ping不能结束。**
* **system函数说明**

****

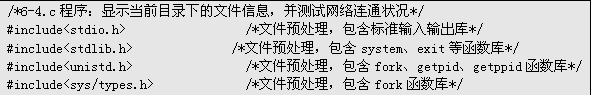
**思考题：如何使用前面的exec函数族中的函数，调用此例中的系统命令“ping [www.lupaworld.com](http://www.lupaworld.com)”？**

**3. fork函数**

* **使用fork函数创建进程时，新的进程叫子进程，原来调用fork函数的进程则称为父进程。**
* **子进程会复制父进程的数据和堆栈空间，并继承父进程的用户代码、组代码、环境变量、已经打开的文件代码、工作目录及资源限制等，但是子进程和父进程使用不同的内存空间。**

**例6-4：设计一个程序，要求先显示当前目录下的文件信息，然后测试到LUPA社区的网络连通状况。**

* **编辑源程序代码**

****

****

* **编译、运行程序，观察结果。**
* **可以看到，使用fork函数创建了一个子进程，子进程的返回值是0，父进程的返回值是子进程的进程号(PID) 。而子进程的父进程号(PPID)和父进程的进程号(PID)相同。**
* **可见，子进程由父进程派生出来。**

**注意：fork函数使用一次就创建一个进程，所以若把fork函数放在if else判断语句或for循环语句中则要小心，不能多次使用fork函数。**

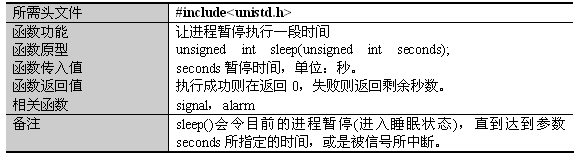
**如： void main()**

**{**

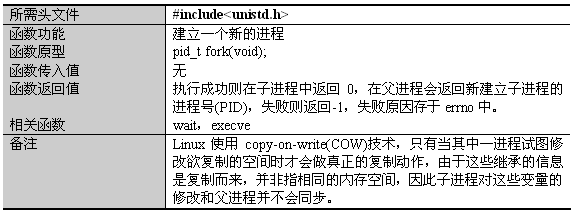
**for(;;)fork();**

**}**

* **sleep函数说明**

****

* **fork函数说明**

****

**思考题：此例中，为什么用sleep等待10秒钟？**

**思考题：设计一个程序，在子进程中调用函数execl("/bin/ps","ps","-ef",NULL)，而在父进程中调用函数execle("/bin/env","env",NULL,envp)，其中有定义：char \*envp[]={"PATH=/tmp","USER=liu",NULL};**

**请编写并进行调试。**

**6.2.3 进程终止**

* **滥用fork函数会占满系统进程，而且子进程与父进程使用不同的内存空间，不断产生子进程，也可能让系统资源消耗殆尽。**
* **Linux环境下c终止进程主要用exit和\_exit函数。**

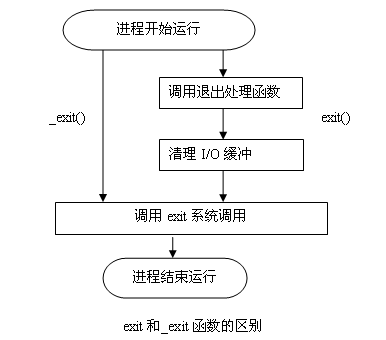
**例6-5：设计一个程序，要求子进程和父进程都在显示输出一些文字后分别用exit和\_exit函数终止进程。**

* **编辑源程序代码：**

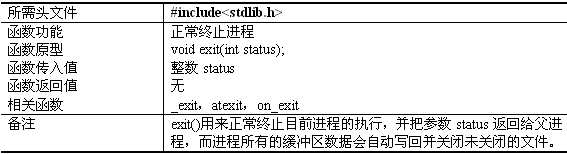
****

**观察结果可以看出，调用exit函数时，缓冲区中的记录能正常输出；而调用\_exit时，缓冲区中的记录无法输出。**

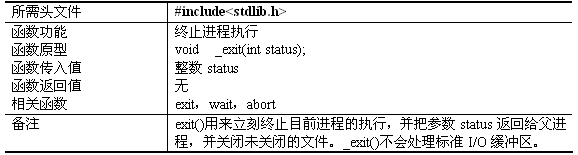
* **\_exit()函数作用：直接使进程停止运行,清除其使用的内存空间,并清除其在内核中的各种数据结构；**
* **exit()函数则在执行退出之前加了若干道工序，exit函数在调用exit系统之前要查看文件的打开情况，把文件缓冲区中的内容写回文件。**

****

* **exit函数说明**

****

* **\_exit函数说明**

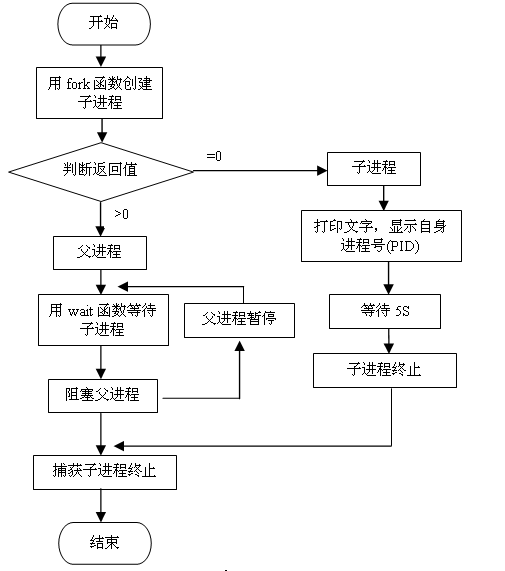
****

**6.2.4 僵尸进程**

* **一个僵尸进程(zombie)，是指已终止运行，但尚末被清除的进程，又称为过渡进程。**
* **当使用fork函数创建子进程时，由于子进程有可能比父进程晚终止，父进程终止后，子进程还没终止，子进程就成了僵尸进程。为避免这种情况，可以在父进程中调用wait或waitpid函数。**
* **wait函数是用于使父进程阻塞，直到一个子进程终止或者该进程接到了一个指定的信号为止。**
* **waitpid的作用和wait一样，但它并不一定要等待第一个终止的子进程，它还有若干选项，也能支持作业控制。**
* **实际上wait函数只是waitpid函数的一个特例，在Linux内部实现wait函数时直接调用的就是waitpid函数。**

**例6-6：**设计一个程序，要求复制进程，子进程显示自己的进程号(PID)后暂停一段时间，父进程等待子进程正常结束，打印显示等待的进程号(PID)和等待的进程退出状态。

* 流程图：

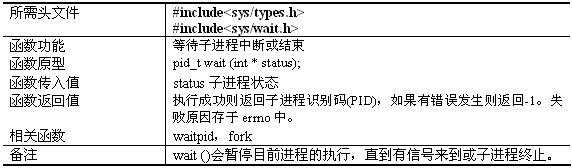


* **编辑源程序代码：**

****

**此例中的子进程运行时间，明显比父进程时间长。为了避免子进程成为僵尸进程，父进程调用wait，阻塞父进程的运行，等待子进程正常结束，父进程才继续运行，直到正常结束。**

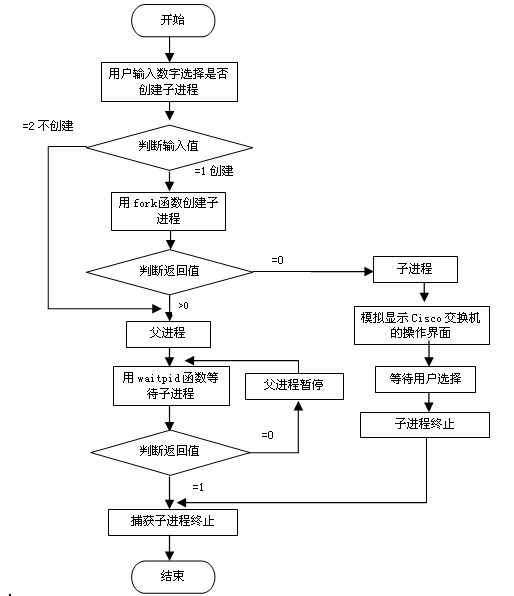
* **wait函数说明**

****

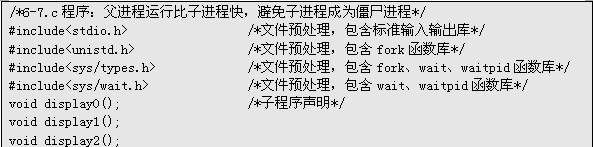
****

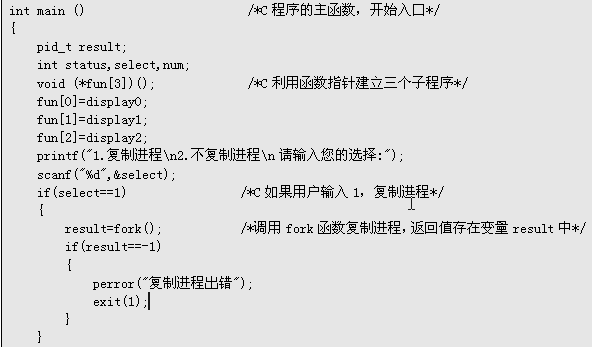
**例6-7：设计一个程序，要求用户可以选择是否复制进程，子进程模仿思科(Cisco)1912交换机的开机界面，以命令行的方式让用户选择进入，父进程判断子进程是否正常终止。**

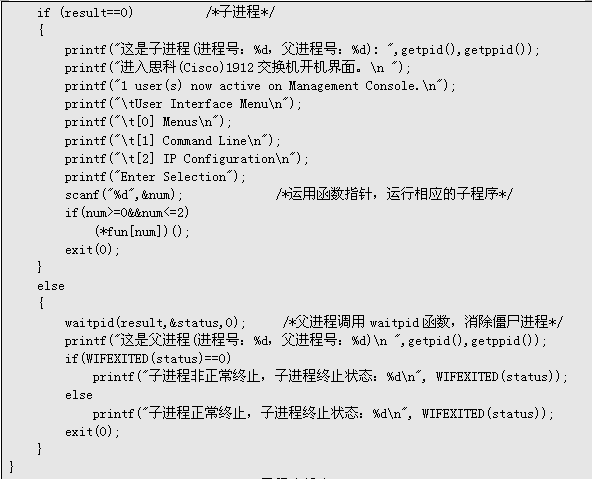
* **流程图：**

****

* **编辑源程序代码：**

****

****

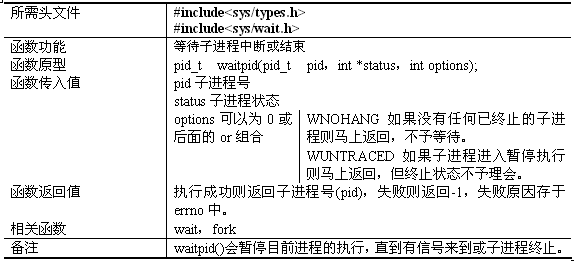
****

****

* **编译、运行程序，提示是否复制进程，先选择“2.不复制进程”，此时没有产生子进程，返回值为“0” 。**
* **再次运行程序后，选择“1.复制进程”，此时产生子进程，子进程的功能是模拟交换机的开机界面，提示选择画面，这儿选择0，进入子程序display0，等待子程序运行终止后，返回值为“1”，父进程才终止。**
* **修改程序：不用waitpid函数。**
* **再次运行程序后，选择“1.复制进程”，这时候父进程没有等待子进程，也就是在模拟显示完交换机的开机界面后，根本没来得及输入选择，父进程就终止了，子进程就变成了僵尸进程。**

**此例可以看出，在没有语法、语义等错误的情况下，程序还是没有完成设计要求。可见，在多进程程序设计时，除了养成使用完后就终止的良好习惯，还要让子进程工作完成后再终止，这个时候父进程就得灵活使用wait函数和waitpid函数。**

* **waitpid函数说明**

****

**6.3 Linux守护进程**

* **守护进程（Daemon）是运行在后台的一种特殊进程。**
* **守护进程独立于控制终端并且周期性地执行某种任务或等待处理某些发生的事件。**
* **守护进程是一种很有用的进程。Linux的大多数服务器就是用守护进程实现的。**
* **同时，守护进程完成许多系统任务。**

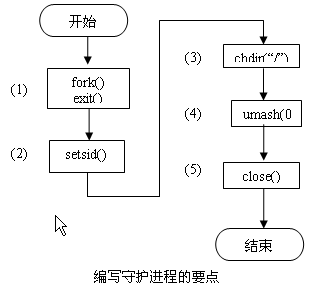
**6.3.1 守护进程及其特性**

* **守护进程最重要的特性是后台运行。**
* **其次，守护进程必须与其运行前的环境隔离开来。这些环境包括未关闭的文件描述符，控制终端，会话和进程组，工作目录以及文件创建掩码等。这些环境通常是守护进程从执行它的父进程（特别是shell）中继承下来的。**
* **最后，守护进程的启动方式有其特殊之处。它可以在Linux系统启动时从启动脚本/etc/rc.d中启动，也可以由作业规划进程crond启动，还可以由用户终端（通常是shell）执行。**
* **通过ps –aux命令可查看Linux环境下的守护进程：**

1. **Init系统守护进程：它是进程1，负责启动各运行层次特定的系统服务。**
2. **Keventd守护进程：为在内核中运行计划执行的函数提供进程上下文。**
3. **Kswapd守护进程：也称为页面调出守护进程。它通过将脏页面以低速写到磁盘上从而使这些页面在需要时仍可回收使用，这种方式支持虚存子系统。**
4. **bdflush和kupdated守护进程：Linux内核使用两个守护进程bdflush和kupdated将调整缓存中的数据冲洗到磁盘上。当可用内存达到下限时，bdflush守护进程将脏缓冲区从缓冲池中冲洗到磁盘上，每隔一定时间间隔，kupdated守护进程将脏页面冲洗到磁盘上，以便在系统失效时减少丢失的数据。**
5. **portmap端口映射守护进程：提供将RPC（远程过程调用）程序号映射为网络端口号的服务。**
6. **syslogd守护进程：可由帮助操作人员把系统消息记入日志的任何程序使用。**
7. **inetd守护进程（xinetd）：它侦听系统网络接口，以便取得来自网络的对各种网络服务进程的请求。**
8. **nfsd、lockd、rpciod守护进程：提供对网络文件系统（NFS） 的支持。**
9. **cron守护进程：在指定的日期和时间执行指定的命令。许多系统管理任务是由cron定期地执行相关程序而实现的。**
10. **cupsd守护进程：是打印假脱机进程，它处理对系统提出的所有打印请求。**

**注意：大多数守护进程都以超级用户（用户ID为0）特权运行。没有一个守护进程具有控制终端，其终端名设置为问号（？）。**

**6.3.2 编写守护进程的要点**

****

**（1）创建子进程，终止父进程**

**pid=fork();**

**if(pid>0)**

**{exit(0);} /\*终止父进程\*/**

**（2）在子进程中创建新会话**

* + **setsid函数用于创建一个新的会话，并担任该会话组的组长，其作用：**
    1. **让进程摆脱原会话的控制；**
    2. **让进程摆脱原进程组的控制；**
    3. **让进程摆脱原控制终端的控制。**
  + **而setsid函数能够使进程完全独立出来，从而脱离所有其他进程的控制。**

**（3）改变工作目录**

* + **改变工作目录的常见函数是chdir。**

**（4）重设文件创建掩码**

* + **文件创建掩码是指屏蔽掉文件创建时的对应位。**
  + **把文件创建掩码设置为0，可以大大增强该守护进程的灵活性。**
  + **设置文件创建掩码的函数是umask。**

**（5）关闭文件描述符**

* + **通常按如下方式关闭文件描述符：**

**for(i=0;i<NOFILE;i++)**

**close(i);**

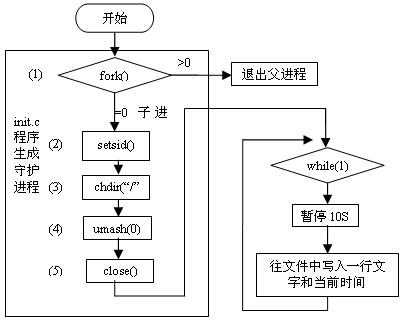
* + **或者也可以用如下方式：**

**for(i=0;i<MAXFILE;i++)**

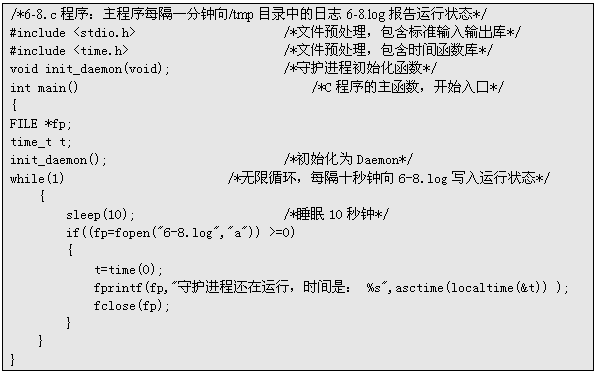
**close(i);**

**6.3.3 守护进程的编写**

**例6-8：设计两个程序，主程序6-8.c和初始化程序init.c。要求主程序每隔十秒钟向/tmp目录中的日志6-8.log报告运行状态。初始化程序中的init\_daemon函数负责生成守护进程。**

****

* **编辑源程序代码：**

****

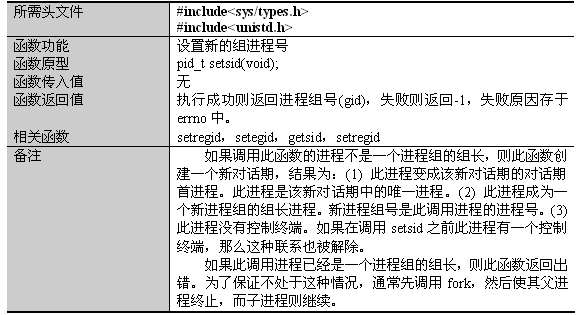
****

**注意：这里的fopen函数必须具有root权限。如果没有root权限，可以看到守护进程的运行，但不会在文件里写入任何字符。**

* **编译、运行文件后，没有任何提示，等待一段时间后，查看一下 6-8.log文件中有没有文字写入，输入“tail -f /tmp/6-8.log”，显示多条文字，从时间上看，说明守护进程在暗地里每隔10秒写入一串字符。**
* **用ps命令查看进程。**
* **可见，6-8确实一直在运行，而且看到“?”结合Linux环境下进程的知识，知道确实有了一个守护进程。**

**注意：父进程创建了子进程，而父进程又退出之后，此时该子进程就变成了“孤儿进程”。在Linux中，每当系统发现一个孤儿进程，就会自动由1号进程（也就是init进程）收养它，原先的子进程就会变成init进程的子进程了。**

* **setsid函数说明:**

****

**思考题：**

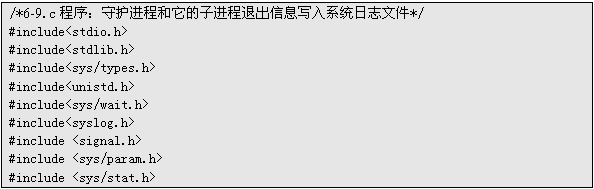
1. **此例中如果不先终止父进程，就调用setsid函数会发生什么？守护进程运行后，注消当前用户后再登录进去，守护进程还在运行吗？**
2. **编写一程序，要求运行后成为守护进程，每隔5分钟修改一次本机的IP地址。**

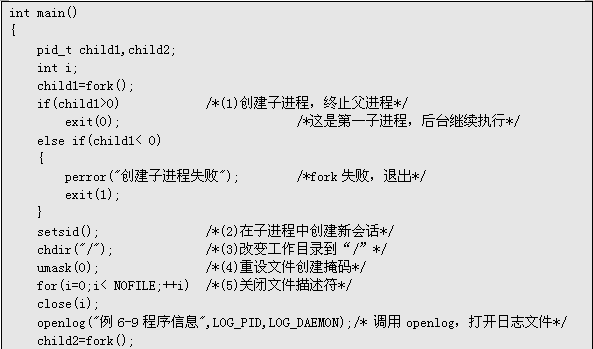
**例6-9：设计两个程序，要求运行后成为守护进程，守护进程又复制出一个子进程，守护进程和它的子进程都调用syslog函数，把结束前的状态写入系统日志文件。**

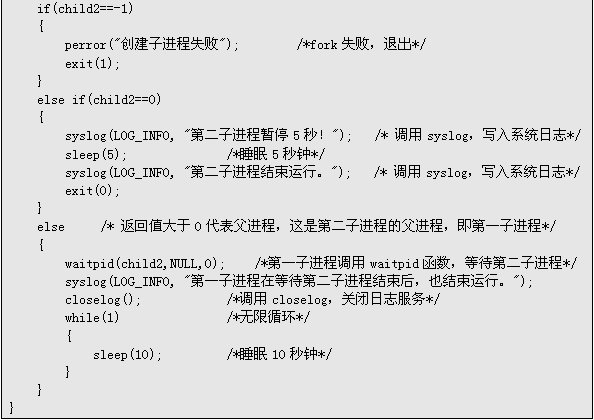
**流程图：**

****

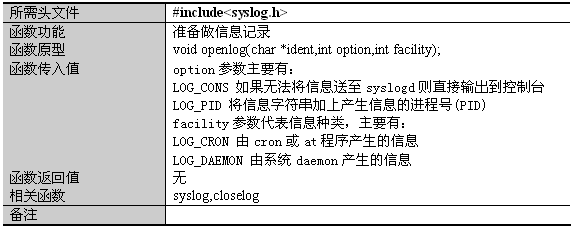
* **编辑源程序代码：**

****

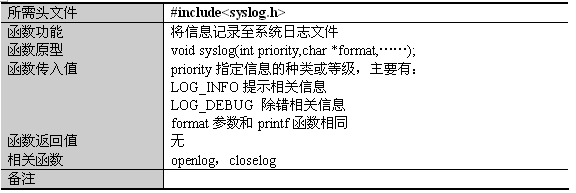
****

****

* **编译、运行程序后，没有任何提示，等待一段时间后，查看一下/var/log/messages文件中有没有文字写入，输入“tail -f /var/log/messages”， 此时，显示多条文字，说明守护进程通过系统日志管理服务，在暗地里写入一串字符，而且从时间上看出，第二子进程确实是在暂停5秒钟后退出的。**
* **用ps命令查看进程 。**
* **可见，6-9确实一直在运行，而且看到“?”结合Linux环境下进程的知识，知道确实有了一个守护进程。**
* **openlog函数说明**

****

* **syslog函数说明**

****

**思考题：编写一程序，要求运行后成为守护进程，复制守护进程的子进程，子进程往某个文件里写入字符串“测试守护进程”，守护进程的错误信息输出到系统日志文件“/var/log/messages”，程序以普通用户权限编译后运行调试会有什么结果？请把产生守护进程的部分分割成独立的程序文件。**

**思考与实验**

**什么是进程？进程与作业有何区别？**

**答：（1）**进程是一个程序的一次执行的过程。在Linux环境下，每个正在运行的程序都称为进程。每个进程包含进程标识符及数据，这些数据包含进程变量、外部变量及进程堆栈等。

1. **进程启动的方式有哪几种？**
2. **用exec函数创建一个进程，显示当前目录下的文件信息。**
3. **execle函数的应用，要在程序执行时设定环境变量，路径为tmp，用户为liu，执行命令env时把这些环境变量传递给系统，在这一函数中，参数e表示可传递新进程环境变量，参数l表示命令或参数逐个列举，文件查找需给出路径。命令env在"/bin"目录下。把环境变量设定为：**

**char \*envp[]={"PATH=/tmp","USER=liu",NULL};**

**因而此函数的调用形式为：**

**execle("usr/bin/env","env",NULL,envp);**

**请编写一程序进行调试。**

1. **execve函数的应用，要在程序执行时设定环境变量，路径为tmp，用户为liu，执行命令env时把这些环境变量传递给系统，在这一函数中，参数e表示可传递新进程环境变量，参数v表示传递的参数(含命令)为构造指针数组，文件查找需给出路径。命令env在"/bin"目录下。把环境变量设定为：**

**char \*envp[]={"PATH=/tmp","USER=liu",NULL};**

**参数的构造指针数组为：**

**char \*arg[]={"env",NULL};**

**因而此函数的调用形式为：**

**execve("usr/bin/env", arg ,envp);**

**请编写一程序进行调试。**

1. **execvp 函数的应用，要在程序中执行命令:ps -ef,命令ps在“/bin”目录下。在这一函数中，参数v为构造指针数组，参数p为文件查找方式（不需要给出路径）。因而构造的指针数组为：**

**char \*arg[]={"ps","-ef",NULL};**

**此函数的调用形式为：**

**execvp("ps",arg);**

**请编写一程序进行调试。**

1. **思考例6-5中的子进程有没有变成僵尸进程，为什么？**
2. **编写一个后台检查邮件的程序，这个程序每隔一个指定的时间会去检查邮箱，如果发现有邮件了，会不断的通过机箱上的小喇叭来发出声音报警 (Linux 的默任个人的邮箱地址是 /var/spool/mail/用户的登录名) 。**

## 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。**