**[linux C守护进程编写](http://www.cnblogs.com/ringwang/p/3528093.html)**

linux编程-守护进程编写

守护进程（Daemon）是运行在后台的一种特殊进程。它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。比如，Internet服务器inetd，Web服务器httpd等。

同时，守护进程完成许多系统任务。比如，作业规划进程crond，打印进程lpd等。

　　守护进程的编程本身并不复杂，复杂的是各种版本的Unix的实现机制不尽相同，

造成不同 Unix环境下守护进程的编程规则并不一致。

需要注意，照搬某些书上的规则（特别是BSD4.3和低版本的System V）到Linux会出现错误的。

下面结合一些前辈的文档和自己的例子说说守护进程的编程。

.基本概念

.进程

.每个进程都有一个父进程

.当子进程终止时，父进程会得到通知并能取得子进程的退出状态。

.进程组

.每个进程也属于一个进程组

.每个进程主都有一个进程组号，该号等于该进程组组长的PID号

.一个进程只能为它自己或子进程设置进程组ID号

.会话期

.对话期(session)是一个或多个进程组的集合。

.setsid()函数可以建立一个对话期：

如果，调用setsid的进程不是一个进程组的组长，此函数创建一个新的会话期。

(1)此进程变成该对话期的首进程

(2)此进程变成一个新进程组的组长进程。

(3)此进程没有控制终端，如果在调用setsid前，该进程有控制终端，那么与该终端的联系被解除。

如果该进程是一个进程组的组长，此函数返回错误。

(4)为了保证这一点，我们先调用fork()然后exit()，此时只有子进程在运行，

子进程继承了父进程的进程组ID，但是进程PID却是新分配的，所以不可能是新会话的进程组的PID。

从而保证了这一点。

if((pid=fork())>0) //parent

exit(0);

else if(pid==0){ //th1 child

setsid(); //th1是成为会话期组长

if(fork() ==0){ //th2不会是会话期组长(变成孤儿进程组)

...

}

}

一． 守护进程及其特性

(1)守护进程最重要的特性是后台运行。在这一点上DOS下的常驻内存程序TSR与之相似。

(2)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模等。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。

(3)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，

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。

　　总之，除开这些特殊性以外，守护进程与普通进程基本上没有什么区别。

因此，编写守护进程实际上是把一个普通进程按照上述的守护进程的特性改造成为守护进程。

二． 守护进程的编程要点 (来自UEAP)

　　前面讲过，不同Unix环境下守护进程的编程规则并不一致。所幸的是守护进程的编程原则其实都一样，

区别在于具体的实现细节不同。这个原则就是要满足守护进程的特性。

同时，Linux是基于Syetem V的SVR4并遵循Posix标准，实现起来与BSD4相比更方便。编程要点如下；

1. 在后台运行。

　　为避免挂起控制终端将Daemon放入后台执行。方法是在进程中调用fork使父进程终止，

让Daemon在子进程中后台执行。

if(pid=fork())

exit(0); //是父进程，结束父进程，子进程继续

2. 脱离控制终端，登录会话和进程组

　　进程属于一个进程组，进程组号（GID）就是进程组长的进程号（PID）。登录会话可以包含多个进程组。

这些进程组共享一个控制终端。这个控制终端通常是创建进程的登录终端。

控制终端，登录会话和进程组通常是从父进程继承下来的。

我们的目的就是要摆脱它们，使之不受它们的影响。

方法是在第1点的基础上，调用setsid()使进程成为会话组长：

setsid();

　　说明：当进程是会话组长时setsid()调用失败。但第一点已经保证进程不是会话组长。

setsid()调用成功后，进程成为新的会话组长和新的进程组长，并与原来的登录会话和进程组脱离。

由于会话过程对控制终端的独占性，进程同时与控制终端脱离。

3. 禁止进程重新打开控制终端

　　现在，进程已经成为无终端的会话组长。但它可以重新申请打开一个控制终端。

可以通过使进程不再成为会话组长来禁止进程重新打开控制终端：

if(pid=fork())

exit(0); //结束第一子进程，第二子进程继续（第二子进程不再是会话组长）

4. 关闭打开的文件描述符

　　进程从创建它的父进程那里继承了打开的文件描述符。如不关闭，将会浪费系统资源，

造成进程所在的文件系统无法卸下以及引起无法预料的错误。按如下方法关闭它们：

for(i=0;i 关闭打开的文件描述符close(i);>

5. 改变当前工作目录

　　进程活动时，其工作目录所在的文件系统不能卸下。一般需要将工作目录改变到根目录。

对于需要转储核心，写运行日志的进程将工作目录改变到特定目录如 /tmpchdir("/")

6. 重设文件创建掩模

　　进程从创建它的父进程那里继承了文件创建掩模。它可能修改守护进程所创建的文件的存取位。

为防止这一点，将文件创建掩模清除：umask(0);

7. 处理SIGCHLD信号

　　处理SIGCHLD信号并不是必须的。

但对于某些进程，特别是服务器进程往往在请求到来时生成子进程处理请求。

如果父进程不等待子进程结束，子进程将成为僵尸进程（zombie）从而占用系统资源。

如果父进程等待子进程结束，将增加父进程的负担，影响服务器进程的并发性能。

在Linux下可以简单地将 SIGCHLD信号的操作设为SIG\_IGN。

signal(SIGCHLD,SIG\_IGN);

　　这样，内核在子进程结束时不会产生僵尸进程。

这一点与BSD4不同，BSD4下必须显式等待子进程结束才能释放僵尸进程。

三． 守护进程实例

　　守护进程实例包括两部分：主程序test.c和初始化程序init.c。

主程序每隔一分钟向/tmp目录中的日志test.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。读者可以利用init\_daemon函数生成自己的守护进程。