# 什么是守护进程？

[守护进程（Daemon Process）](http://baike.baidu.com/link?url=IcWqjC9d2Seh1YbNmbv4RdTbQ6PHDjQrqU1mxoTCalSOGlTm1ehoJIYnxVEgAqTDByUfSlH7rMdat00N1IKJva)，也就是通常说的 Daemon 进程（精灵进程），是 Linux 中的后台服务进程。通常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。

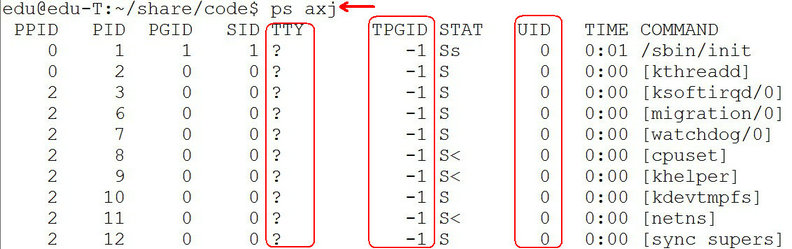
[守护进程](http://baike.baidu.com/link?url=IcWqjC9d2Seh1YbNmbv4RdTbQ6PHDjQrqU1mxoTCalSOGlTm1ehoJIYnxVEgAqTDByUfSlH7rMdat00N1IKJva)是个特殊的[孤儿进程](http://blog.csdn.net/tennysonsky/article/details/45969569)，这种进程脱离终端，其在执行过程中的信息也不在任何终端上显示。

**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。

# ****如何查看守护进程****

在终端敲：ps axj

* a 表示不仅列当前用户的进程，也列出所有其他用户的进程
* x 表示不仅列有控制终端的进程，也列出所有无控制终端的进程
* j 表示列出与作业控制相关的信息



从上图可以看出守护进程的一些特点：

* 守护进程基本上都是以超级用户启动（ UID 为 0 ）
* 没有控制终端（ TTY 为 ？）
* 终端进程组 ID 为 -1 （ TPGID 表示终端进程组 ID）

一般情况下，守护进程可以通过以下方式启动：

* 在系统启动时由启动脚本启动，这些启动脚本通常放在 /etc/rc.d 目录下；
* 利用 inetd 超级服务器启动，如 telnet 等；
* 由 cron 定时启动以及在终端用 nohup 启动的进程也是守护进程。

# 如何编写守护进程？

下面是编写守护进程的基本过程：

1）**屏蔽一些控制终端操作的信号**

这是为了防止守护进行在没有运行起来前，控制终端受到干扰退出或挂起。

signal(SIGTTOU,SIG\_IGN);

signal(SIGTTIN,SIG\_IGN);

signal(SIGTSTP,SIG\_IGN);

signal(SIGHUP,SIG\_IGN);

2）**在后台运行**

方法是在进程中调用 fork() 使父进程终止， 让守护进行在子进程中后台执行。

if( pid = fork() ){ // 父进程

    exit(0);        //结束父进程，子进程继续

}

3）**脱离控制终端、登录会话和进程组**

Linux 中的**进程与控制终端，登录会话和进程组之间的关系**：

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。

控制终端通常是创建进程的 shell 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**。因此需要调用 setsid() ，代码如下：

setsid();  
setsid() 调用成功后，子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。

4）**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**。禁止进程重新打开控制终端采用的方法是再次创建一个子进程，示例代码如下：

if( pid=fork() ){ // 父进程

    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）

}

5）**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。按如下方法关闭它们：

// NOFILE 为 <sys/param.h> 的宏定义

// NOFILE 为文件描述符最大个数，不同系统有不同限制

for(i=0; i< NOFILE; ++i){// 关闭打开的文件描述符

    close(i);

}

6）**改变当前工作目录**

示例代码如下：

chdir("/");

7）**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权限。为防止这一点，将文件创建掩模清除：

umask(0);

8）**处理 SIGCHLD 信号**

signal(SIGCHLD, SIG\_IGN);  
这样，内核在子进程结束时不会产生僵尸进程。

示例代码如下：

#include <unistd.h>

#include <signal.h>

#include <fcntl.h>

#include <sys/syslog.h>

#include <sys/param.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int init\_daemon(void)

{

    int pid;

int i;

    // 1）屏蔽一些控制终端操作的信号

    signal(SIGTTOU, SIG\_IGN);

    signal(SIGTTIN, SIG\_IGN);

    signal(SIGTSTP, SIG\_IGN);

signal(SIGHUP, SIG\_IGN);

    // 2）在后台运行

    if (pid = fork())

    {            // 父进程

        exit(0); //结束父进程，子进程继续

    }

    else if (pid < 0)

    { // 出错

        perror("fork");

        exit(EXIT\_FAILURE);

}

    // 3）脱离控制终端、登录会话和进程组

setsid();

    // 4）禁止进程重新打开控制终端

    if (pid = fork())

    {            // 父进程

        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）

    }

    else if (pid < 0)

    { // 出错

        perror("fork");

        exit(EXIT\_FAILURE);

}

    // 5）关闭打开的文件描述符

    // NOFILE 为 <sys/param.h> 的宏定义

    // NOFILE 为文件描述符最大个数，不同系统有不同限制

    for (i = 0; i < NOFILE; ++i)

    {

        close(i);

}

    // 6）改变当前工作目录

chdir("/tmp");

    // 7）重设文件创建掩模

umask(0);

    // 8）处理 SIGCHLD 信号

signal(SIGCHLD, SIG\_IGN);

    return 0;

}

int main(int argc, char \*argv[])

{

init\_daemon();

    while (1)

        ;

    return 0;

}

运行结果如下：

