[**守护进程的创建方法和步骤**](http://blog.chinaunix.net/uid-25365622-id-3055635.html)

概念：

     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。（这里的结尾字母d就是Daemon的意思）

创建步骤：

①使进程在后台运行

  创建子进程父进程退出

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

   exit(0);

else if(pid<0)

{

    perror("fail to fork");

    exit(-1);

}

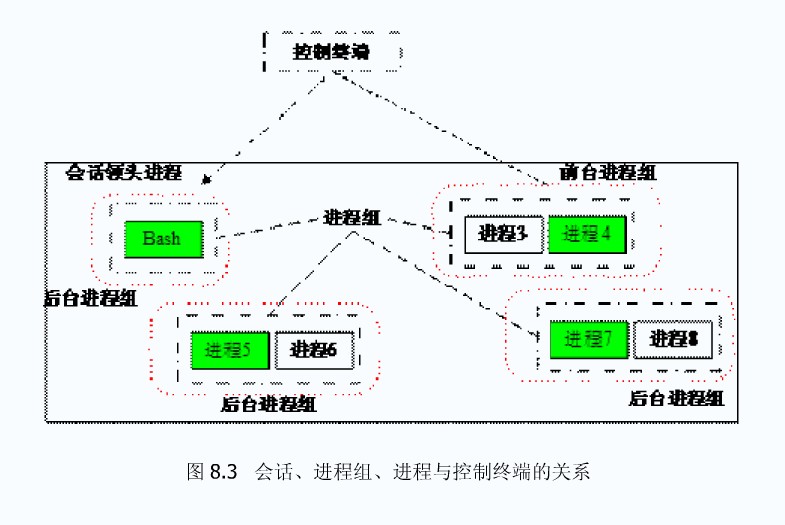
②脱离控制终端，登录会话和进程组（创建新会话）

     有必要先介绍一下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。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。

[](http://blog.chinaunix.net/attachment/201201/9/25365622_1326091221XGVy.jpg)

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关闭所有文件描述符

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:

for(i=0;i<=getdtablesize();i++)

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/tmp:

chdir("/tmp") ;

⑥重设权限掩码

    进程从创建它的父进程那里继承了文件创建掩码。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。

无代码无真相

代码说明：

init\_deamon.c :按照送上面步骤创建守护进程

test.c :调用创建守护进程函数，每隔一秒向/tmp目录下的print\_time文件打印系统时间

1. /\*name: init\_deamon.c
2. \*function:创建一个守护进程
3. \*/
4. #include <stdio.h>
5. #include <stdlib.h>
6. #include <unistd.h>
7. #include <signal.h>
8. #include <sys/param.h>
9. #include <sys/types.h>
10. #include <sys/stat.h>
11. void init\_deamon(void)
12. {
13. int pid;
14. int i;
15. /\* 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。\*/
16. if(signal(SIGCHLD,SIG\_IGN) == SIG\_ERR){
17. printf("Cant signal in init\_daemon.");
18. exit(1);
19. }
20. if(pid=fork())
21. exit(0);//是父进程，结束父进程
22. else if(pid< 0){
23. perror("fail to fork1");
24. exit(1);//fork失败，退出
25. }
26. //是第一子进程，后台继续执行
27. setsid();//第一子进程成为新的会话组长和进程组长
28. //并与控制终端分离
29. if(pid=fork())
30. exit(0);//是第一子进程，结束第一子进程
31. else if(pid< 0)
32. exit(1);//fork失败，退出
33. //是第二子进程，继续
34. //第二子进程不再是会话组长
35. for(i=0;i< getdtablesize();++i)//关闭打开的文件描述符
36. close(i);
37. chdir("/tmp");//改变工作目录到/tmp
38. umask(0);//重设文件创建掩模
39. return;
40. }

1. /\* name     :test.c
2. \* function    :调用init\_deamon函数使进程变成守护进程，然后每个一秒向/tmp目录下的print\_time文件打印当前时间
3. \* \*/
4. #include <stdio.h>
5. #include <time.h>
6. void init\_deamon(void);//守护进程初始化函数
7. void main()
8. {
9. FILE \*fp;
10. time\_t t;
11. init\_deamon();//初始化为Daemon
12. while(1)//每隔一分钟向test.log报告运行状态
13. {
14. sleep(1);//睡眠一秒钟
15. if((fp=fopen("print\_time","a")) >=0)
16. {
17. t=time(0);
18. fprintf(fp,"The time right now is : %s",asctime(localtime(&t)));
19. fclose(fp);
20. }
21. }
22. return;
23. }