## exec 函数族

背景：fork创建进程之后，子进程和父进程执行相同的代码，但是在实际开发当中，我们希望父子进程执行不同的代码。

作用：执行指定的程序

#include <unistd.h>

int execl(const char \*path, const char \*arg, …);

int execlp(const char \*file, const char \*arg, …);

成功时执行指定的程序；失败时返回EOF

path 执行的程序名称，包含路径

arg… 传递给执行的程序的参数列表

file 执行的程序的名称，在PATH中查找

注意：两个函数区别execlp不需要写文件名全路径，在PATH查找

最后一个参数必须用空指针(NULL)作结束

进程当前内容被指定的程序替换，但进程号不变

第0个参数必须要写，虽然它没有使用

## 守护进程（Daemon Process）

概念：

守护进程又叫精灵进程（Daemon Process），它是一个生存期较长的进程，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。

特点：

始终在后台运行，独立于任何终端，周期性的执行某种任务或等待处理特定事件。

它是个特殊的孤儿进程，这种进程脱离终端，为什么要脱离终端呢？之所以脱离于终端是为了避免进程被任何终端所产生的信息所打断，其在执行过程中的信息也不在任何终端上显示。由于在 Linux 中，每一个系统与用户进行交流的界面称为终端，每一个从此终端开始运行的进程都会依附于这个终端，这个终端就称为这些进程的控制终端，当控制终端被关闭时，相应的进程都会自动关闭。

举例：

http 服务的守护进程叫 httpd，mysql 服务的守护进程叫 mysqld。

更简便地创建守护进程： nohup 命令

nohup xxxx &

setsid函数：

pid\_t setsid(void);

成功：返回调用进程的会话ID；失败：-1，设置errno。  
调用了setsid函数的进程，既是新的会长，也是新的组长

getsid函数

pid\_t getsid(pid\_t pid)

成功：返回调用进程的会话ID；失败：-1，设置errno

1.pid为0表示察看当前进程session ID

2.ps ajx命令查看系统中的进程。参数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,参数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,参数j表示列出与作业控制相关的信息。

3.组长进程不能成为新会话首进程,新会话首进程必定会成为组长进程。

getpid：pid\_t getpid(void); 获取进程id

getpgid：pid\_t getpgid(pid\_t pid); 获取进程组id

## GDB调试多进程程序

set follow-fork-mode child 设置GDB调试子进程

set follow-fork-mode parent 设置GDB调试父进程

set detach-on-fork on/off 设置GDB跟踪调试单个进程或多个

on: 只调试父进程或子进程的其中一个，(根据follow-fork-mode来决定)，这是默认的模式

off：父子进程都在gdb的控制之下，其中一个进程正常调试(根据follow-fork-mode来决定),另一个进程会被设置为暂停状态。

info inferiors 显示GDB调试的进程

inferiors 进程序号（1,2,3....） 切换GDB调试的进程