fork()：

|  |  |
| --- | --- |
| 所需头文件 | #include <sys/types.h>  #include <unistd.h> |
| 函数原型 | pid\_t fork(void); |
| 返回值(调用一次，会得到两个返回值) | -1：创建子进程失败(设置errno) |
| 0：创建子进程成功，0被返回至子进程 |
| >0的整数：创建子进程成功，将子进程的pid返回给父进程 |

1. fork()函数用于创建一个子进程，该子进程几乎拷贝了父进程的所有内容(正文段、用户数据段、系统数据段)；
2. 父子进程各有各的内存空间；
3. 父子进程可以同时运行；
4. 父子进程谁先结束是不确定的。

若父进程先结束，则子进程的父进程变为1号进程init，即子进程被init进程收养，这种情况下，子进程被称为孤儿进程。

若子进程先结束，而父进程还在运行。则子进程称为僵尸进程，等待父进程调用wait()或waitpid()收尸。

一般来说，在fork之后是父进程先执行还是子进程先执行是不确定的。这取决于内核所使用的调度算法。

vfork()：

由于fork()函数在创建子进程时，完整地拷贝了父进程的整个地址空间，因此执行速度比较慢。

vfork()函数也可用于创建子进程，但它不产生父进程的副本。它通过允许父子进程访问相同物理内存伪装了对进程地址空间的真实拷贝，当子进程需要改变内存中数据时才拷贝父进程，这就是著名的“写操作时拷贝”(copy-on-write)技术。

vfork()与fork()的区别：

(1) 结束进程，必须用exit函数；

(2) 子进程一定先运行，子进程结束后，父进程才开始运行；

(3) 父子进程共用内存空间；

(4) 在子进程结束前，父进程处于等待不可中断状态(D状态)。(不可中断，是指不可被信号中断)

exec函数族：

fork()函数用于创建一个子进程，该子进程几乎复制了父进程的全部内容，但是，这个新创建的进程如何执行呢？exec函数族就提供了在一个进程中启动另一个程序执行的方法。它可以根据指定的文件名或目录名找到可执行文件，并用它来取代原调用进程的代码段、数据段和堆栈段，在执行完之后，原调用进程的内容除进程号外，其他内容全部被新的进程替换了。另外，这里的可执行文件既可以是二进制文件，也可以是Linux下任何可执行的脚本文件。

在Linux中使用exec函数族主要有两种情况：

1. 当进程认为自己不能再为系统和用户作出任何贡献时，就可以调用exec函数族中的任意一个函数让自己重生；
2. 如果一个进程想执行另一个程序，那么它就可以调用fork()函数新建一个进程，然后在子进程中调用exec函数族中的任一个函数，这样看起来就像通过执行应用程序而产生了一个新进程。

|  |  |
| --- | --- |
| 所需头文件 | #include <unistd.h> |
| 函数原型 | int execl(const char \*path, const char \*arg, …); |
| int execlp(const char \*file, const char \*arg, …); |
| int execle(const char \*path, const char \*arg, …, char \*const envp[]); |
| int execv(const char \*path, char \*const argv[]); |
| int execvp(const char \*file, char \*const argv[]); |
| int execve(const char \*pathname, char \* const argv[], char \*const \* envp[]); |
| 返回值 | -1：出错 |

注意：

1. 可执行文件的查找方式：以p结尾的两个函数可以只给出文件名，系统会自动按照PATH环境变量指定的路径进行查找；其他四个函数必须指定完整的文件路径名；
2. 参数传递方式：字母为’l’表示逐个列举参数，语法为const char \*arg；字母为’v’表示将所有参数构造成指针数组进行传递，语法为char \* const argv[]；这两中形式的传参，必须以任一参数开始，以NULL结束，即真正的参数为中间部分；
3. 环境变量：exec函数族可以默认系统的环境变量，也可以传入指定的环境变量；以’e’结尾的两个函数就可以在envp[]中指定当前进程所使用的环境变量；且envp[]也要以NULL结束。
4. 在使用exec函数族时，一定要加上错误判断语句。exec很容易执行失败，常见原因有：找不到文件或路径；数组argv和envp忘记用NULL结束；没有对应可执行文件的运行权限。
5. 可以使用getenv()函数(头文件stdlib.h)获取某一指定环境变量的内容，例如：getenv(“PATH”);其返回值为环境变量内容的首地址。

exit()和\_exit()：

exit()和\_exit()函数都是用来终止进程的。但二者还是有着一些明显的不同。

|  |  |
| --- | --- |
| 所需头文件 | exit(): #include <stdlib.h> (库函数) |
| \_exit():#include <unistd.h> (系统调用) |
| 函数原型 | exit(): void exit(int status); |
| \_exit(): void \_exit(int status); |
| 函数传入值 | status：可以利用这个参数传递进程结束时的状态。通常0表示正常结束，其他数值表示出现了错误，进程非正常结束。在实际编程中，可以利用wait系统调用接受子进程的返回值，进行相应的处理。 |

注意：

1. \_exit()函数的作用最为简单：直接使进程终止运行，清除其使用的内存空间，并销毁其在内核中的各种数据结构；
2. exit()函数在终止当前进程之前要检查文件的打开情况，把文件缓冲区中的内容写回文件，即“清理I/O缓存”；
3. exit()函数会将status&0377(八进制------二进制011 111 111)返回给父进程。即不论status是多么大的一个数值，只取其最低的一个字节值返回。

wait()：

|  |  |
| --- | --- |
| 所需头文件 | #include <sys/types.h>  #include <sys/wait.h> |
| 函数原型 | pid\_t wait(int \*status); |
| 参数 | status是一个整型指针，指向的对象用来保存子进程退出时的状态 |
| 返回值 | 成功：已结束运行的子进程的pid； |
| 失败：-1 |

注意：

1. wait()函数是父进程用来处理僵尸进程的。
2. wait()函数用于使父进程阻塞(等待状态)，直到一个子进程结束或者该进程接收到一个指定的信号为止。如果该父进程没有子进程或者其子进程已经结束，则wait()就会立即返回。
3. status指向的对象，只有最低两个字节为有效字节：若\*status & 0xff==0，则子进程为正常结束，\*status的倒数第二个字节存放exit()的退出值；否则，子进程是被信号杀死的，倒数第一个字节存放杀死子进程的信号编号。
4. wait(& status)等价于waitpid(-1, & status, 0)。

waitpid()：

|  |  |  |
| --- | --- | --- |
| 所需头文件 | | #include <sys/types.h> #include <sys/wait.h> |
| 函数原型 | | pid\_t waitpid(pid\_t pid, int \*status, int options); |
| 参数 | pid | pid>0：只等待进程号等于pid的子进程结束，不管是否有其他子进程结束，只要指定的子进程还未结束，就一直等下去。 |
| pid=-1：等待任一个进程结束，和wait()一样 |
| status | 同wait() |
| options | WNOHANG：若有pid指定的子进程并不立即可用，则waitpid不阻塞，此时返回值是0 |
| WUNTRACED：作业控制 |
| 0：同wait，阻塞父进程，等待子进程退出 |
| 返回值 | | 成功：已结束运行的子进程的pid； |
| 使用WNOHANG选项且无子进程结束时：0 |
| 失败：-1 |

守护进程daemon：

守护进程，也就是通常所说的daemon进程，是Linux中的后台服务进程。它是一个生命周期较长的进程，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。守护进程常常在系统引导载入时启动，在系统关闭时终止。

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的进程都依附于这个终端，这个终端称为这些进程的控制终端，当控制终端关闭时，相应的进程都会自动关闭，但是守护进程却能突破这种限制。如果想让某个进程不因用户、终端或其他变化而受到影响，就必须将这个进程变为守护进程。

Linux中守护进程的编写步骤：

1. 创建子进程，父进程退出；pid = fork(); if(pid > 0) exit(0);
2. 在子进程中创建新会话；------setsid();
3. 更改工作目录为根目录；------chdir(“/”);
4. 重设文件权限掩码；------umask(0);
5. 关闭文件描述符。------close(fd);

注意：

1. 第一步完成后：子进程变为孤儿进程，被1号进程init收养，运行于后台(R+)。子进程就在形式上做到了与控制终端的脱离；
2. 第二步是创建守护进程最重要的一步：调用fork()函数创建子进程时，子进程复制了父进程的会话期、进程组、控制终端等，虽然父进程退出了，但子进程的会话期、进程组、控制终端并没有改变，因此，这还不是真正意义上的独立。调用setsid()函数，可以使子进程完全独立出来。
3. 进程组：进程组是一个或多个进程的集合。进程组由进程组ID(PGID)来唯一标示。每个进程组都有一个组长进程(父进程)，其组长进程的PID即为PGID。
4. 会话组：会话组是一个或多个进程组的集合。会话组由SID来标示。通常，一个会话开始于用户登录，终止于用户退出，在此期间该用户运行的所有进程都属于这个会话期。会话组的组长即为bash。