（1）exec函数说明  
fork函数是用于创建一个子进程，该子进程几乎是父进程的副本，而有时我们希望子进程去执行另外的程序，exec函数族就提供了一个在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数据段、代码段和堆栈段，在执行完之后，原调用进程的内容除了进程号外，其他全部被新程序的内容替换了。另外，这里的可执行文件既可以是二进制文件，也可以是Linux下任何可执行脚本文件。  
（2）在Linux中使用exec函数族主要有以下两种情况  
a. 当进程认为自己不能再为系统和用户做出任何贡献时，就可以调用任何exec 函数族让自己重生。  
b. 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，然后调用任何一个exec函数使子进程重生。  
（3）exec函数族语法  
实际上，在Linux中并没有exec函数，而是有6个以exec开头的函数族，下表列举了exec函数族的6个成员函数的语法。  
所需头文件： #include <unistd.h>  
函数说明： 执行文件  
函数原型：

1. int execl(const char \*path, const char \*arg, ...)
2. int execv(const char \*path, char \*const argv[])
3. int execle(const char \*path, const char \*arg, ..., char \*const envp[])
4. int execve(const char \*path, char \*const argv[], char \*const envp[])
5. int execlp(const char \*file, const char \*arg, ...)
6. int execvp(const char \*file, char \*const argv[])

函数返回值：成功 -> 函数不会返回，出错 -> 返回-1，失败原因记录在error中。  
这6 个函数在函数名和使用语法的规则上都有细微的区别，下面就可执行文件查找方式、参数表传递方式及环境变量这几个方面进行比较说明。  
①    查找方式：上表其中前4个函数的查找方式都是完整的文件目录路径，而最后2个函数（也就是以p结尾的两个函数）可以只给出文件名，系统就会自动从环境变量“$PATH”所指出的路径中进行查找。  
②    参数传递方式：exec函数族的参数传递有两种方式，一种是逐个列举的方式，而另一种则是将所有参数整体构造成指针数组进行传递。  
在这里参数传递方式是以函数名的第5位字母来区分的，字母为“l”（list）的表示逐个列举的方式，字母为“v”（vertor）的表示将所有参数整体构造成指针数组传递，然后将该数组的首地址当做参数传给它，数组中的最后一个指针要求是NULL。读者可以观察execl、execle、execlp的语法与execv、execve、execvp的区别。  
③    环境变量：exec函数族使用了系统默认的环境变量，也可以传入指定的环境变量。这里以“e”（environment）结尾的两个函数execle、execve就可以在envp[]中指定当前进程所使用的环境变量替换掉该进程继承的所以环境变量。  
（4）PATH环境变量说明  
PATH环境变量包含了一张目录表，系统通过PATH环境变量定义的路径搜索执行码，PATH环境变量定义时目录之间需用用“:”分隔，以“.”号表示结束。PATH环境变量定义在用户的.profile或.bash\_profile中，下面是PATH环境变量定义的样例，此PATH变量指定在“/bin”、“/usr/bin”和当前目录三个目录进行搜索执行码。  
PATH=/bin:/usr/bin:.  
export $PATH  
（5）进程中的环境变量说明  
    在Linux中，Shell进程是所有执行码的父进程。当一个执行码执行时，Shell进程会fork子进程然后调用exec函数去执行执行码。Shell进程堆栈中存放着该用户下的所有环境变量，使用execl、execv、execlp、execvp函数使执行码重生时，Shell进程会将所有环境变量复制给生成的新进程；而使用execle、execve时新进程不继承任何Shell进程的环境变量，而由envp[]数组自行设置环境变量。  
（6）exec函数族关系  
第4位统一为：exec  
第5位  
l：参数传递为逐个列举方式  
execl、execle、execlp  
v：参数传递为构造指针数组方式  
execv、execve、execvp  
第6位  
e：可传递新进程环境变量  
execle、execve  
p：可执行文件查找方式为文件名  
execlp、execvp

事实上，这6个函数中真正的系统调用只有execve，其他5个都是库函数，它们最终都会调用execve这个系统调用，调用关系如下图12-11所示：

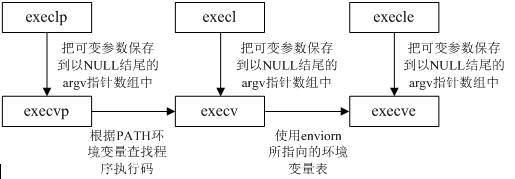


                                        图12-11 exec函数族关系图  
（7）exec调用举例如下：

1. char \*const ps\_argv[] ={"ps", "-o", "pid,ppid,pgrp,session,tpgid,comm", NULL};
2. char \*const ps\_envp[] ={"PATH=/bin:/usr/bin", "TERM=console", NULL};
3. execl("/bin/ps", "ps", "-o", "pid,ppid,pgrp,session,tpgid,comm", NULL);
4. execv("/bin/ps", ps\_argv);
5. execle("/bin/ps", "ps", "-o", "pid,ppid,pgrp,session,tpgid,comm", NULL, ps\_envp);
6. execve("/bin/ps", ps\_argv, ps\_envp);
7. execlp("ps", "ps", "-o", "pid,ppid,pgrp,session,tpgid,comm", NULL);
8. execvp("ps", ps\_argv);

请注意exec函数族形参展开时的前两个参数，第一个参数是带路径的执行码（execlp、execvp函数第一个参数是无路径的，系统会根据PATH自动查找然后合成带路径的执行码），第二个是不带路径的执行码，执行码可以是二进制执行码和Shell脚本。  
（8）exec函数族使用注意点  
在使用exec函数族时，一定要加上错误判断语句。因为exec很容易执行失败，其中最常见的原因有：  
①    找不到文件或路径，此时errno被设置为ENOENT。  
②    数组argv和envp忘记用NULL结束，此时errno被设置为EFAULT。  
③    没有对应可执行文件的运行权限，此时errno被设置为EACCES。