<https://blog.csdn.net/drdairen/article/details/51896141>

exit()就是退出，传入的参数是程序退出时的状态码，0表示正常退出，其他表示非正常退出，一般都用-1或者1，标准C里有EXIT\_SUCCESS和EXIT\_FAILURE两个宏，用exit(EXIT\_SUCCESS);可读性比较好一点。   
**1、异同：**   
作为系统调用而言，\_exit和exit是一对孪生兄弟，它们究竟相似到什么程度，我们可以从Linux的源码中找到答案： 

“\_NR“是在Linux的源码中为每个系统调用加上的前缀，请注意第一个exit前有2条下划线，第二个exit前只有1条下划线。   
这时随便一个懂得C语言并且头脑清醒的人都会说，\_exit和exit没有任何区别，但我们还要讲一下这两者之间的区别，这种区别主要体现在它们在函数库中的定义。\_exit在Linux函数库中的原型是： 

exit()函数定义在stdlib.h中，而\_exit()定义在unistd.h中   
\_exit()函数的作用最为简单：直接使进程停止运行，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；exit() 函数则在这些基础上作了一些包装，在执行退出之前加了若干道工序，也是因为这个原因，有些人认为exit已经不能算是纯粹的系统调用。   
exit()函数与\_exit()函数最大的区别就在于exit()函数在调用exit系统调用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，就是”清理I/O缓冲”。

**2、exit()在结束调用它的进程之前，要进行如下步骤：**   
1.调用atexit()注册的函数（出口函数）；按ATEXIT注册时相反的顺序调用所有由它注册的函数,这使得我们可以指定在程序终止时执行自己的清理动作.例如,保存程序状态信息于某个文件,解开对共享数据库上的锁等.

2.cleanup()；关闭所有打开的流，这将导致写所有被缓冲的输出，删除用TMPFILE函数建立的所有临时文件.

3.最后调用\_exit()函数终止进程。

**2、\_exit做3件事（man）：**   
1，Any open file descriptors belonging to the process are closed   
2，any children of the process are inherited by process 1, init   
3，the process’s parent is sent a SIGCHLD signal

exit执行完清理工作后就调用\_exit来终止进程。

**在Linux的标准函数库中，有一套称作”高级I/O”的函数，我们熟知的printf()、fopen()、fread()、fwrite()都在此 列，它们也被称作”缓冲I/O（buffered I/O）”，其特征是对应每一个打开的文件，在内存中都有一片缓冲区，每次读文件时，会多读出若干条记录，这样下次读文件时就可以直接从内存的缓冲区中读取，每次写文件的时候，也仅仅是写入内存中的缓冲区，等满足了一定的条件（达到一定数量，或遇到特定字符，如换行符和文件结束符EOF）， 再将缓冲区中的 内容一次性写入文件，这样就大大增加了文件读写的速度，但也为我们编程带来了一点点麻烦。如果有一些数据，我们认为已经写入了文件，实际上因为没有满足特 定的条件，它们还只是保存在缓冲区内，这时我们用\_exit()函数直接将进程关闭，缓冲区中的数据就会丢失，反之，如果想保证数据的完整性，就一定要使用exit()函数。**

简单的说，exit函数将终止调用进程。在退出程序之前，所有文件关闭，缓冲输出内容将刷新定义，并调用所有已刷新的“出口函数”（由atexit定义）。   
\_exit:该函数是由Posix定义的，不会运行exit handler和signal handler，在UNIX系统中不会flush标准I/O流。   
简单的说，\_exit终止调用进程，但不关闭文件，不清除输出缓存，也不调用出口函数。   
共同：   
不管进程是如何终止的，内核都会关闭进程打开的所有file descriptors,释放进程使用的memory!

**更详细的介绍：**   
**Calling exit()**   
The exit() function causes normal program termination.   
The exit() function performs the following functions:   
1. All functions registered by the Standard C atexit() function are called in the reverse   
order of registration. If any of these functions calls exit(), the results are not portable.   
2. All open output streams are flushed (data written out) and the streams are closed.   
3. All files created by tmpfile() are deleted.   
4. The \_exit() function is called.

**Calling \_exit()**   
The \_exit() function performs operating system-specific program termination functions.   
These include:   
1. All open file descriptors and directory streams are closed.   
2. If the parent process is executing a wait() or waitpid(), the parent wakes up and   
status is made available.   
3. If the parent is not executing a wait() or waitpid(), the status is saved for return to   
the parent on a subsequent wait() or waitpid().   
4. Children of the terminated process are assigned a new parent process ID. Note: the   
termination of a parent does not directly terminate its children.   
5. If the implementation supports the SIGCHLD signal, a SIGCHLD is sent to the parent.   
6. Several job control signals are sent.

**为何在一个fork的子进程分支中使用\_exit函数而不使用exit函数？**   
‘exit()’与‘\_exit()’有不少区别在使用‘fork()’，特别是‘vfork()’时变得很   
突出。   
‘exit()’与‘\_exit()’的基本区别在于前一个调用实施与调用库里用户状态结构(user-mode constructs)有关的清除工作(clean-up)，而且调用用户自定义的清除程序 (自定义清除程序由atexit函数定义，可定义多次，并以倒序执行)，相对应，\_exit函数只为进程实施内核清除工作。

在由‘fork()’创建的子进程分支里，正常情况下使用‘exit()’是不正确的，这是 因为使用它会导致标准输入输出(stdio: Standard Input Output)的缓冲区被清空两次，而且临时文件被出乎意料的删除(临时文件由tmpfile函数创建在系统临时目录下，文件名由系统随机生成)。在C++程序中情况会更糟，因为静态目标(static objects)的析构函数(destructors)可以被错误地执行。(还有一些特殊情况，比如守护程序，它们的父进程需要调用‘\_exit()’而不是子进程；适用于绝大多数情况的基本规则是，‘exit()’在每一次进入‘main’函数后只调用一次。)