# 一、结束进程

## void exit(int value);

功能：

结束调用此函数的进程。

参数：

status：返回给父进程的参数（低 8 位有效）

返回值：

无

所需头文件：

#include <unistd.h>

## void \_exit(int value);

功能：

结束调用此函数的进程。

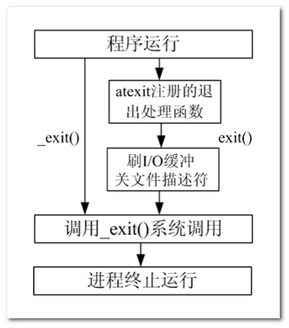
参数：

status：返回给父进程的参数（低 8 位有效）

返回值：

无

exit() 和 \_exit() 函数功能和用法是一样的，exit()属于标准库函数，\_exit()属于系统调用函数。



下面的例子验证调用 exit() 函数，会刷新 I/O 缓冲区：

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

int main(int argc, char \*argv[])

{

    printf("hi, lh, you are so good"); // 打印，没有换行符"\n"

    exit(0);      // 结束进程，标准库函数，刷新缓冲区，printf()的内容能打印出来

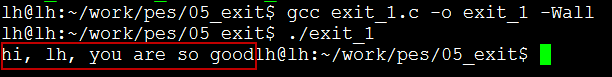
    // \_exit(0);  // 结束进程，系统调用函数，printf()的内容不会显示到屏幕

while(1);   // 不让程序结束

    return 0;

}

运行结果：



# 二、等待进程结束

当一个进程正常或异常终止时，内核就向其父进程发送 SIGCHLD 信号，相当于告诉父亲他哪个儿子挂了，**而父进程可以通过 wait() 或 waitpid() 函数等待子进程结束，获取子进程结束时的状态，同时回收他们的资源**

wait() 和 waitpid() 函数的功能一样，区别在于，wait() 函数会阻塞，waitpid() 可以设置不阻塞

所需头文件：

#include <sys/types.h>

#include <sys/wait.h>

## pid\_t wait(int \*status);

功能：

等待任意一个子进程结束，如果任意一个子进程结束了，此函数会回收该子进程的资源。调用 wait() 函数的进程会挂起（阻塞）**，wait()函数的主要功能为回收已经结束子进程的资源。**

参数：

status: 进程退出时的状态信息。

如下两个函数用于获取子进程的退出信息：

WIFEXITED(status)

如果子进程是正常终止的，取出的字段值非零。

WEXITSTATUS(status)

返回子进程的退出状态，退出状态保存在 status 变量的 8~16 位。

返回值：

成功：已经结束子进程的进程号

失败：-1

## pid\_t waitpid(pid\_t pid, int \* status, int options);

功能：

等待子进程终止，如果子进程终止了，此函数会回收子进程的资源。

参数：

pid: 参数 pid 的值有以下几种类型：

pid > 0

等待进程 ID 等于 pid 的子进程。

pid = 0

等待同一个进程组中的任何子进程

pid = -1

等待任一子进程，此时 waitpid 和 wait 作用一样。

pid < -1

等待指定进程组中的任何子进程，这个进程组的 ID 等于 pid 的绝对值。

status: 进程退出时的状态信息。和 wait() 用法一样。

options: options 提供了一些额外的选项来控制 waitpid()。

0：

同 wait()，阻塞父进程，等待子进程退出。

WNOHANG；

没有任何已经结束的子进程，则立即返回。

WUNTRACED：

如果子进程暂停了则此函数马上返回，并且不予以理会子进程的结束状态。

返回值：

waitpid() 的返回值比 wait() 稍微复杂一些，一共有 3 种情况：

当正常返回的时候，waitpid() 返回收集到的已经子进程的进程号；

如果设置了选项 WNOHANG，而调用中 waitpid() 发现没有已退出的子进程可等待，则返回 0；

如果调用中出错，则返回 -1，这时 errno 会被设置成相应的值以指示错误所在，如：当 pid 所对应的子进程不存在，或此进程存在，但不是调用进程的子进程，waitpid() 就会出错返回，这时 errno 被设置为 ECHILD；

测试代码：

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

int main(int argc, char \*argv[])

{

    pid\_t pid;

    pid = fork(); // 创建进程

    if( pid < 0 ){ // 出错

        perror("fork");

        exit(0);

    }

    if( pid == 0 ){// 子进程

        int i = 0;

        for(i=0;i<5;i++)

        {

            printf("this is son process\n");

            sleep(1);

        }

        \_exit(2); // 子进程退出，数字 2 为子进程退出的状态

    }else if( pid > 0){ // 父进程

        int status = 0;

        // 等待子进程结束，回收子进程的资源

        // 此函数会阻塞

        // status 某个字段保存子进程调用 \_exit(2) 的 2，需要用宏定义取出

        wait(&status);

        // waitpid(-1, &status, 0); // 和 wait() 没区别，0：阻塞

        // waitpid(pid, &status, 0); // 指定等待进程号为 pid 的子进程， 0 阻塞

        // waitpid(pid, &status, WNOHANG); // WNOHANG：不阻塞

        if(WIFEXITED(status) != 0){ // 子进程是否正常终止

            printf("son process return %d\n", WEXITSTATUS(status));

        }

        printf("this is father process\n");

    }

    return 0;

}

运行结果：

