进程退出

进程退出函数没有返回值，他的参数是退出状态，会给到父进程，父进程可以获取到。

```
#include <stdlib.h>
void exit(int status);   //标准C库

#include <unistd.h>
void _exit(int status);  //Linux系统调用
```

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>

int main()
{
    //标准输出是行缓冲的，所以遇到换行符就会把用户区的缓冲刷到内核中。
    printf("Hello\n");
    printf("World");

    //_exit(int)该函数是Linux系统调用中的函数，用于终止进程。
    _exit(0);

    // exit(int)函数是标准C库的函数，内部也会调用_exit函数，但是在此之前会执行刷新缓冲IO、关闭文件描述符等等操作。
    //  exit(0);

    //如果第三个语句执行_exit函数，那么第二句是打印不出来的。而执行exit函数就可以打印出来。

    return 0;
}
```

孤儿进程

概述

■ 父进程运行结束，但子进程还在运行（未运行结束），这样的子进程就称为孤儿进程（Orphan Process）。
■ 每当出现一个孤儿进程的时候，内核就把孤儿进程的父进程设置为 init（1号进程），而 init 进程会循环地 wait() 它的已经退出的子进程。这样，当一个孤儿进程凄凉地结束了其生命周期的时候，init 进程就会代表党和政府出面处理它的一切善后工作。
■ 因此孤儿进程并不会有什么危害。

实例

```
#include <unistd.h>
#include <stdio.h>

int main()
{
    pid_t pid = fork();
```

```
    if (pid > 0)
    {
        //父进程
        printf("this is parent process \n");
    }
    else if (pid == 0)
    {
        //子进程
        for (int i = 0; i < 3; i++)
        {
            printf("你好 --> %d\n", i);
        }
        printf("子进程 %d  --> 的父进程 %d\n", getpid(), getppid());
    }
}
```

两种结果：

僵尸进程

概述

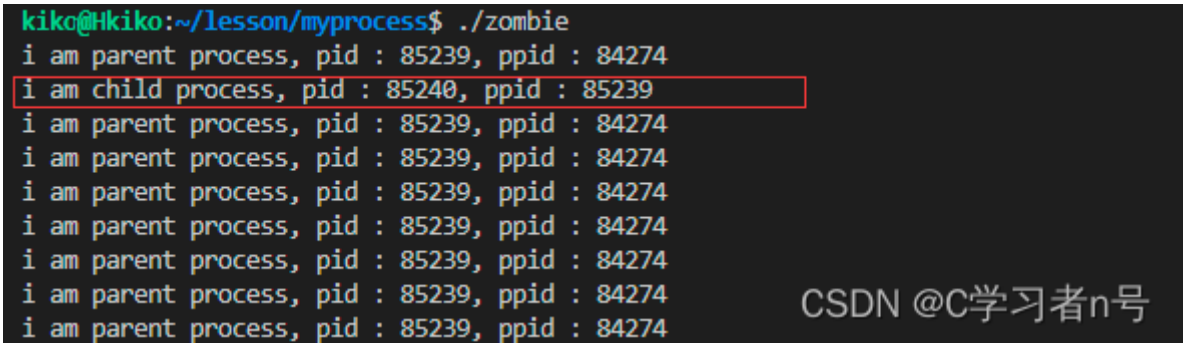
■ 每个进程结束之后, 都会释放自己地址空间中的用户区数据，内核区的 PCB 没有办法自己释放掉，需要父进程去释放。
■ 子进程终止时，父进程尚未回收，子进程残留资源（PCB）存放于内核中，变成僵尸（Zombie）进程。
■ 僵尸进程不能被 kill -9 杀死。这样就会导致一个问题，如果父进程不调用 wait() 或 waitpid() 的话，那么保留的那段信息就不会释放，其进程号就会一直被占用，但是系统所能使用的进程号是有限的，如果大量的产生僵尸进程，将因为没有可用的进程号而导致系统不能产生新的进程，此即为僵尸进程的危害，应当避免。
■ 僵尸进程的产生一般是：父进程一直在执行，基本不停止，但是内部并没有调用wait或者waitpid函数对终止的子进程回收，此时终止的子进程就是僵尸进程。
如果父进程不会执行很久，很快也结束了，但是里面也没有调用wait或者waitpid函数，那么即使产生了僵尸进程，最终init进程也会将僵尸进程清理。

实例

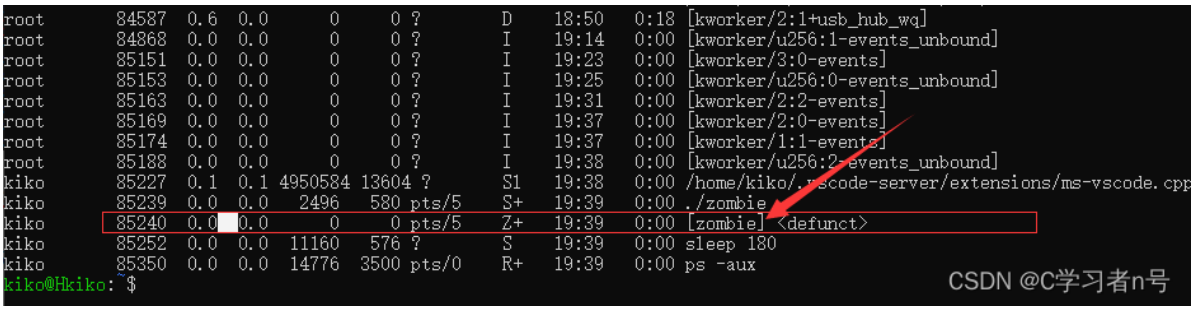
```
#include <unistd.h>
#include <stdio.h>

int main()
{

    // 创建子进程
```

```
    pid_t pid = fork();

    // 判断是父进程还是子进程
    if (pid > 0)
    {
        while (1)
        {
            printf("i am parent process, pid : %d, ppid : %d\n", getpid(),
getppid());
            sleep(2);
        }
    }
    else if (pid == 0)
    {
        // 当前是子进程
        printf("i am child process, pid : %d, ppid : %d\n", getpid(),
getppid());
    }

    return 0;
}
```

执行结果如下，父进程一直在执行，子进程执行完就结束了。



根据打印的结果，子进程的pid是85240。此时查看系统的进程`ps -aux`



进程回收

概述

进程回收主要是`wait`和`waitpid`函数。

所有这些系统调用都用于等待调用进程的子进程的状态变化，并获取有关其状态已更改的子进程的信息。 状态变化被认为是：孩子终止； 孩子被信号拦住； 或者孩子被一个信号恢复了。 在终止子进程的情况下，执行等待允许系统释放与子关联的资源。

如果一个孩子已经改变了状态，那么这些调用会立即返回。 否则，它们会阻塞直到子进程改变状态或信号处理程序中断调用。

注意：一次wait或waitpid调用只能清理一个子进程，清理多个子进程应使用循环。

wait函数

函数介绍

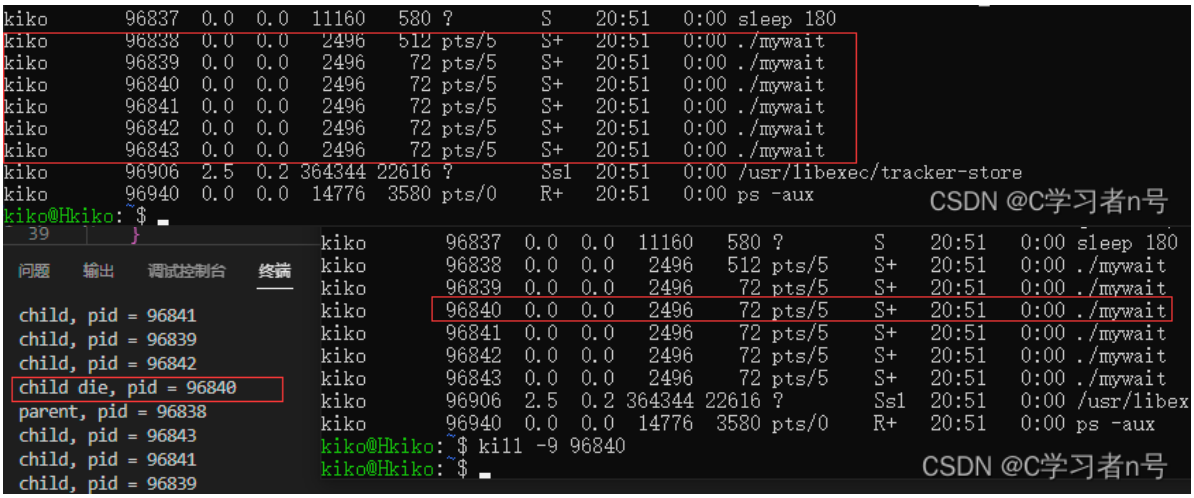
```
#include <sys/types.h>
#include <sys/wait.h>

pid_t wait(int *wstatus);
```

wait() 系统调用暂停调用线程的执行，直到其子线程之一终止。
参数：用于获取进程结束的状态码，是int值的地址。一个输出参数。
返回值：`成功时，返回终止子进程的进程ID`； `所有的子进程都结束或者调用函数失败，返回 -1`。

实例1

```
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main()
{
    //创建5个进程
    pid_t pid;
    for (size_t i = 0; i < 5; i++)
    {
        pid = fork();
        if (pid == 0)
            break;
    }

    if (pid > 0)
    {
        //父进程
        while (1)
        {
            printf("parent, pid = %d\n", getpid());

            //会阻塞，等待回收子进程
            int ret = wait(NULL); // NULL参数表示不需要获取进程结束的状态码

            printf("child die, pid = %d\n", ret);
        }
    }
    else if (pid == 0)
    {
        //子进程
        while (1)
        {
            printf("child, pid = %d\n", getpid());
            sleep(1);
        }
    }
```

```
    return 0; // exit(0)
}
```



实例2

```
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main()
{
    //创建5个进程
    pid_t pid;
    for (size_t i = 0; i < 5; i++)
    {
        pid = fork();
        if (pid == 0)
            break;
    }

    if (pid > 0)
    {
        //父进程
        while (1)
        {
            printf("parent, pid = %d\n", getpid());

            //会阻塞，等待回收子进程
            int status;
            int ret = wait(&status); // NULL参数表示不需要获取进程结束的状态码

            printf("child die, pid = %d  --- status = %d\n", ret, status);

            if (ret == -1)
                break;
        }
    }
    else if (pid == 0)
    {
        //子进程
```

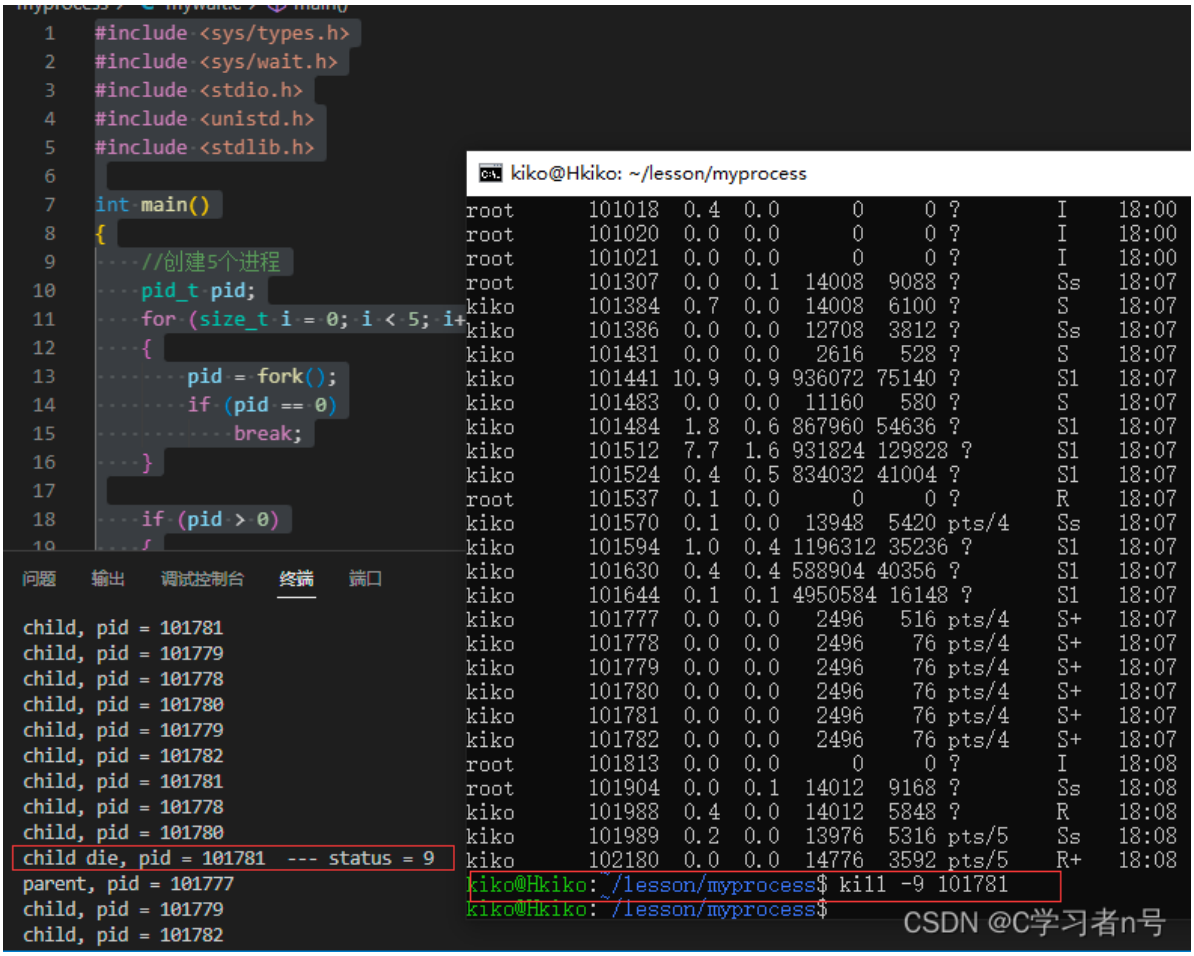
```
        // while (1)
        // {
        printf("child, pid = %d\n", getpid());
        // sleep(3);
        exit(1);
        // }
    }

    return 0; // exit(0)
}
```

```
kiko@Hkiko:~/lesson/myprocess$ ./mywait
child, pid = 99562
parent, pid = 99561
child, pid = 99563
child die, pid = 99562  --- status = 256
parent, pid = 99561
child, pid = 99566
child die, pid = 99563  --- status = 256
parent, pid = 99561
child die, pid = 99566  --- status = 256
parent, pid = 99561
child, pid = 99565
child die, pid = 99565  --- status = 256
parent, pid = 99561
child, pid = 99564
child die, pid = 99564  --- status = 256
parent, pid = 99561
child die, pid = -1  --- status = 256
```


实例3

```
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main()
{
    //创建5个进程
    pid_t pid;
    for (size_t i = 0; i < 5; i++)
    {
        pid = fork();
        if (pid == 0)
            break;
    }

    if (pid > 0)
    {
        //父进程
        while (1)
        {
            printf("parent, pid = %d\n", getpid());

            //会阻塞，等待回收子进程
            int status;
```

```c
            int ret = wait(&status); // NULL参数表示不需要获取进程结束的状态码

            printf("child die, pid = %d  --- status = %d\n", ret, status);

            if (ret == -1)
                break;
        }
    }
    else if (pid == 0)
    {
        //子进程
        while (1)
        {
            printf("child, pid = %d\n", getpid());
            sleep(3);
            // exit(1);
        }
    }

    return 0; // exit(0)
}
```



waitpid函数

概述

函数原型

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t waitpid(pid_t pid, int *wstatus, int options);
```

功能：回收指定进程号的子进程，可以设置是否阻塞，默认是阻塞的。
参数：

```
pid
    pid > 0 : 某个子进程的pid
    pid = 0 : 回收当前进程组的所有子进程
    pid = -1 : 回收所有的子进程，相当于 wait()   （最常用）
    pid < -1 : 某个进程组的组id的绝对值，回收指定进程组中的子进程
options
    0 : 阻塞
    WNOHANG : 非阻塞
wstatus
    输出参数，用于获取终止进程的状态。和wait函数一样。
```

返回值

```
> 0 : 返回被终止的子进程的id
= 0 : 如果 options=WNOHANG, 表示还有子进程未终止
= -1 : 错误，或者没有子进程了
```

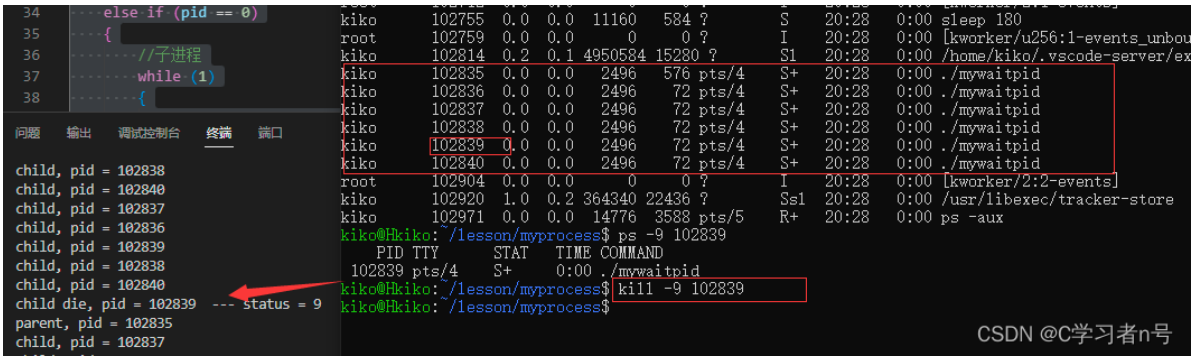
wait和waitpid函数

```
wait(&wstatus)
等价于:
waitpid(-1, &wstatus, 0);
```

实例1

```
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main()
{
    //先创建5个进程
    pid_t pid;
    for (size_t i = 0; i < 5; i++)
    {
        pid = fork();
        if (pid == 0)
            break;
    }
```

```
    if (pid > 0)
    {
        //父进程
        while (1)
        {
            printf("parent, pid = %d\n", getpid());

            int status;
            int ret = waitpid(-1, &status, 0); // 0，阻塞; -1,清理所有子进程

            printf("child die, pid = %d  --- status = %d\n", ret, status);

            if (ret == -1)
                break;
        }
    }
    else if (pid == 0)
    {
        //子进程
        while (1)
        {
            printf("child, pid = %d\n", getpid());
            sleep(3);
            // exit(1);
        }
    }

    return 0; // exit(0)
}
```



实例2

```
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main()
{
    //先创建5个进程
    pid_t pid;
    for (size_t i = 0; i < 5; i++)
    {
        pid = fork();
        if (pid == 0)
```

```
            break;
    }

    if (pid > 0)
    {
        //父进程
        while (1)
        {
            printf("parent, pid = %d\n", getpid());
            sleep(2);

            int status;
            int ret = waitpid(-1, &status, WNOHANG); // WNOHANG，阻塞; -1,清理所有
子进程

            printf("child die, pid = %d  --- status = %d\n", ret, status);

            if (ret == -1)
                break;
        }
    }
    else if (pid == 0)
    {
        //子进程
        while (1)
        {
            printf("child, pid = %d\n", getpid());
            sleep(3);
            // exit(1);
        }
    }

    return 0; // exit(0)
}
```

父进程不会被阻塞。如果没有被杀死的进程，waiptpid返回0，获取的状态也是0.