僵尸进程：一个进程使用fork创建子进程，如果子进程退出，而父进程并没有调用wait或waitpid获取子进程的状态信息，那么子进程的进程描述符仍然保存在系统中。这种进程称之为僵死进程。

说明：

1：父进程没有调用 wait 和 waitpid函数，子进程结束后，父进程依然在执行，这时子进程就会成为 僵尸进程；

什么是僵尸进程?

首先内核会释放终止进程(调用了exit系统调用)所使用的所有存储区，关闭所有打开的文件等，但内核为每一个终止子进程保存了一定量的信息。这些信息至少包括进程ID，进程的终止状态，以及该进程使用的CPU时间，所以当终止子进程的父进程调用wait或waitpid时就可以得到这些信息。

而僵尸进程就是指：一个进程执行了exit系统调用退出，而其父进程并没有为它收尸(调用wait或waitpid来获得它的结束状态)的进程。

任何一个子进程(init除外)在exit后并非马上就消失，而是留下一个称外僵尸进程的数据结构，等待父进程处理。这是每个子进程都必需经历的阶段。另外子进程退出的时候会向其父进程发送一个SIGCHLD信号。

僵尸进程的目的?

设置僵死状态的目的是维护子进程的信息，以便父进程在以后某个时候获取。这些信息至少包括进程ID，进程的终止状态，以及该进程使用的CPU时间，所以当终止子进程的父进程调用wait或waitpid时就可以得到这些信息。如果一个进程终止，而该进程有子进程处于僵尸状态，那么它的所有僵尸子进程的父进程ID将被重置为1（init进程）。继承这些子进程的init进程将清理它们（也就是说init进程将wait它们，从而去除它们的僵尸状态）。

1：产生僵尸进程

现在想要通过一个进程创建一个进程，然后让这两个进程各自独立地完成各自的任务（通常我们都是这样做的），如果现在只调用 fork() 一次，让父子进程分别完成各自的任务，假设父进程处理程序的时间要比子进程处理程序的时间要长，那么 子进程 必然先于 父进程 退出，子进程必然有一段时间会处于僵尸状态。

为了解决这个问题（也就是说不让进程出现僵尸状态），就可以调用 fork() 两次。让子进程再去创建一个孙子进程，然后子进程立即退出，在父进程中等待子进程的退出，由于子进程马上就退出了，所以父进程不会阻塞很长时间就会立即返回，接着指向父进程该执行

的任务；接着说孙子进程，由于子进程已经退出，所以孙子进程此时已经变成孤儿进程，会由 init 进程领养，不可能会变成 僵尸进程。

代码如下：

```
main(void)
{
    pid_t   pid;

    if ((pid = fork()) < 0) {
        err_sys("fork error");
    } else if (pid == 0) {      /* first child */
        if ((pid = fork()) < 0)
            err_sys("fork error");
        else if (pid > 0)
            exit(0);    /* parent from second fork == first child */

        /*
         * We're the second child; our parent becomes init as soon
         * as our real parent calls exit() in the statement above.
         * Here's where we'd continue executing, knowing that when
         * we're done, init will reap our status.
         */
        sleep(2);
        printf("second child, parent pid = %d\n", getppid());
        exit(0);
    }

    if (waitpid(pid, NULL, 0) != pid)   /* wait for first child */
        err_sys("waitpid error");

    /*
     * We're the parent (the original process); we continue executing,
     * knowing that we're not the parent of the second child.
     */
    exit(0);
}
```

==========

如何避免僵尸进程?

1. 通过signal(SIGCHLD, SIG_IGN)通知内核对子进程的结束不关心，由内核回收。如果**不想让父进程挂起**，可以在父进程中加入一条语句：signal(SIGCHLD,SIG_IGN);表示父进程忽略SIGCHLD信号，该信号是子进程退出的时候向父进程发送的。
2. 父进程调用wait/waitpid等函数等待子进程结束，如果尚无子进程退出**wait会导致父进程阻塞。waitpid可以通过传递WNOHANG使父进程不阻塞立即返回。**
3. 如果父进程很忙可以用signal注册信号处理函数，在信号处理函数调用wait/waitpid等待子进程退出。

4. 通过两次调用fork。父进程首先调用fork创建一个子进程然后waitpid等待子进程退出，子进程再fork一个孙进程后退出。这样子进程退出后会被父进程等待回收，而对于孙子进程其父进程已经退出所以孙进程成为一个孤儿进程，孤儿进程由init进程接管，孙进程结束后，init会等待回收。

第一种方法忽略SIGCHLD信号，这常用于并发服务器的性能的一个技巧因为并发服务器常常fork很多子进程，子进程终结之后需要服务器进程去wait清理资源。如果将此信号的处理方式设为忽略，可让内核把僵尸子进程转交给init进程去处理，省去了大量僵尸进程占用系统资源。测试是可行的

僵尸进程的处理：

示例：

如以下代码会创建100个子进程，但是父进程并未等待它们结束，所以在父进程退出前会有100个僵尸进程。

复制代码

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

int main() {

  int i;
  pid_t pid;

  for(i=0; i<100; i++) {
    pid = fork();
    if(pid == 0)
      break;
  }

  if(pid>0) {
    printf("press Enter to exit...");
    getchar();
  }

  return 0;
}
```

复制代码

其中一个解决方法即是编写一个SIGCHLD信号处理程序来调用wait/waitpid来等待子进程返回。

复制代码

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

void wait4children(int signo) {

  int status;
  wait(&status);

}

int main() {

  int i;
  pid_t pid;

  signal(SIGCHLD, wait4children);

  for(i=0; i<100; i++) {
    pid = fork();
    if(pid == 0)
      break;
  }

  if(pid>0) {
    printf("press Enter to exit...");
    getchar();
  }

  return 0;
}
```

复制代码

但是通过运行程序发现还是会有僵尸进程，而且每次僵尸进程的数量都不定。这是为什么呢？其实主要是因为**Linux的信号机制是不排队的**，假如在某一时间段多个子进程退出后都会发出SIGCHLD信号，但父进程来不及一个一个地响应，所以最后父进程实际上只执行了一次信号处理函数。但执行一次信号处理函数只等待一个子进程退出，所以最后会有一些子进程依然是僵尸进程。
虽然这样但是有一点是明了的，就是收到SIGCHLD必然有子进程退出，而我们可以在信号处理函数里循环调用waitpid函数来等待所有的退出的子进程。至于为什么不用wait，主要原因是在wait在清理完所有僵尸进程后再次等待会阻塞。

所以最佳方案如下：

复制代码

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

void wait4children(int signo) {
  int status;
  while(waitpid(-1, &status, WNOHANG) > 0);
}

int main() {

  int i;
  pid_t pid;

  signal(SIGCHLD, wait4children);

  for(i=0; i<100; i++) {
    pid = fork();
    if(pid == 0)
      break;
  }

  if(pid>0) {
    printf("press Enter to exit...");
```

```
    getchar();
  }

  return 0;
}
```

复制代码

这里使用waitpid而不是使用wait的原因在于：我们在一个循环内调用waitpid，以获取所有已终止子进程的状态。我们必须指定WNOHANG选项，它告诉waitpid在有尚未终止的子进程在运行时不要阻塞。我们不能在循环内调用wait，因为没有办法防止wait在正运行的子进程尚有未终止时阻塞。