一、预备知识

1、父子进程：

在 Linux 中，当一个进程创建了另一个进程后，它们就成为了父子进程，而进程和它的所有子女以及后裔共同组成一个进程组。

2、init 进程：

是 Linux 内核启动的第一个用户级进程（PID=1），用于系统的一些初始化工作，并且维护系统的整个运行过程。在整个系统中，所有的进程都属于以 init 为根的一棵树。

3、孤儿进程：

当一个父进程结束时，它还在运行的子进程便成了孤儿进程。此时，孤儿进程会 init 进程所收养（即 init 进程成为了它们的父进程），init 进程负责收集它们的结束状态值以及完成对它们的彻底销毁。

4、僵尸进程：

子进程退出后会释放自己的资源，但会保留一些信息（如 PID、结束状态、运行时间等）存储在结束状态值中，此时若父进程未能及时调用 wait 或 waitpid 来获取子进程的这些状态信息，那么这些信息会一直保存到系统中。这种子进程称作僵尸进程。

二、原理分析

1、 危害性：

1）僵尸进程会一直占用进程号，但进程号是有限的，若产生了大量僵尸进程，则会造成进程号缺少而无法产生新的进程。

2）孤儿进程是没有危害的，因为它会被 init 进程收养，并完成它们的善后工作。

2、采取措施：

1）本质上讲，问题的根源并不是僵尸进程，而是产生它们的父进程，所以，要么让父进程在合适的时间调用 wait 或 waitpid 对僵尸进程进行处理，要么直接杀死父进程让子进程成为孤儿进程被 init 进程收养。

2）子进程退出时向父进程发送 SIGCHILD 信号，所以当父进程需要持续运行无法结束时，可以接受并处理 SIGCHILD 信号来销毁子进程。

3、函数介绍：

1）pid_t fork(void);

（pid_t 是一个宏定义，其实质是 int 被定义在#include<sys/types.h>中）

返回值： 若成功调用一次则返回两个值，子进程返回 0，父进程返回子进程 ID；否则，出错返回-1

作用：创建一个子进程，子进程是父进程的完整复制，将获得父进程数据空间、堆、栈等资源的副本。

2）void (*signal(int signum,void(* handler)(int)))(int);

或者：typedef void(*sig_t) (int); sig_t signal(int signum,sig_t handler);

第一个参数 signum 指明了所要处理的信号类型，它可以是取除了 SIGKILL 和 SIGSTOP 外的任何一种信号，如子进程退出时向父进程发送的 SIGCHILD 信号。

第二个参数 handler 描述了与信号关联的动作，它可以取以下三种值：

（1）一个无返回值的函数地址

此函数必须在 signal()被调用前申明，handler 中为这个函数的名字。当接收到一个类型为 sig 的信号时，就执行 handler 所指定的函数。这个函数应有如下形式的定义：

void func(int sig);

sig 是传递给它的唯一参数。执行了 signal()调用后，进程只要接收到类型为 sig 的信号，不管其正在执行程序的哪一部分，就立即执行 func()函数。当 func()函数执行结束后，控制权

返回进程被中断的那一点继续执行。

（2）SIG_IGN

这个符号表示忽略该信号，执行了相应的 signal()调用后，进程会忽略类型为 sig 的信号。

（3）SIGDFL

这个符号表示恢复系统对信号的默认处理。

三、具体实现

1、孤儿进程测试

1）程序设计：

首先调用 fork()创建一个子进程，在子进程中输出 pid 和 ppid（父进程 id），然后让父进程结束，再让子进程输出 pid 和 ppid。

2）代码如下：

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

int main()
{
    pid_t pid = fork();
    //创建失败退出程序
    if (pid < 0)
    {
        printf("fork error.");
        exit(1);
    }
    //子进程入口
    if (pid == 0)
    {
        printf("I am the child process.\n");
        //输出进程 ID 和父进程 ID
        printf("pid: %d\tppid:%d\n",getpid(),getppid());
        //让出 CPU，保证父进程先退出
        sleep(1);
        printf("pid: %d\tppid:%d\n",getpid(),getppid());
        printf("child process is exited.\n");
    }
    //父进程入口
    else
    {
        printf("I am the father process.\n");
        //让出 CPU，先让子进程输出 pid 和 ppid
        sleep(1);
        printf("father process is exited.\n");
    }
    return 0;
```

}

3）测试结果：

2、僵尸进程测试

1）程序设计：

调用 fork()创建一个子进程后，在子进程中输出 pid 和 ppid 后迅速结束，然后在父进程中查看进程信息，看是否存在僵尸进程，然后调用 wait，再次查看进程信息。

2）代码如下：

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

int main()
{
    pid_t pid = fork();
    //创建失败退出程序
    if (pid < 0)
    {
        printf("fork error.");
        exit(1);
    }
    //子进程入口
    if (pid == 0)
    {
        printf("I am the child process.\n");
        printf("pid: %d\tppid:%d\n",getpid(),getppid());
        printf("child process is exited.\n");
        exit(0);
    }
    //等待子进程先退出
    sleep(1);
    printf("I am the father process.\n");
    //输出进程信息
```

```
    system("ps -o pid,ppid,state,command");
    //处理僵尸进程
    wait(NULL);
    printf("wait finished.\n");
    //再次输出进程信息
    system("ps -o pid,ppid,state,command");
    printf("father process is exited.\n");
    return 0;
}
```

3）测试结果：

3、通过信号机制处理僵尸进程

1）程序设计：

首先用 signal()捕获 SIGCHILD 信号，并在处理函数中调用 wait()；之后调用 fork()创建一个子进程，在子进程中输出 pid 和 ppid 后迅速结束，然后在父进程中查看进程信息，看是否存在僵尸进程。

2）代码如下：

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

void handler(int sig);

int main()
{
    //捕捉子进程退出信号
    signal(SIGCHLD,handler);

    pid_t pid = fork();
```

```
    if (pid < 0)
    {
        printf("fork error.");
        exit(1);
    }
    //子进程入口
    if (pid == 0)
    {
        printf("I am the child process.\n");
        printf("pid: %d\tppid:%d\n",getpid(),getppid());
        printf("child process is exited.\n");
        exit(0);
    }
    //等待子进程先退出
    sleep(1);
    printf("I am the father process.\n");
    //输出进程信息
    system("ps -o pid,ppid,state,command");
    printf("father process is exiting.\n");
    return 0;
}

void handler(int sig)
{
    pid_t pid = wait(NULL);
    printf("wait finished.\n");
}
```

3）测试结果：