# 僵尸进程（Zombie Process）

进程已运行结束，但进程的占用的资源未被回收，这样的进程称为僵尸进程。

在每个进程退出的时候，内核释放该进程所有的资源、包括打开的文件、占用的内存等。 但是仍然为其保留一定的信息，这些信息主要主要指进程控制块的信息（包括进程号、退出状态、运行时间等）。直到父进程通过 wait() 或 waitpid() 来获取其状态并释放。

子进程已运行结束，父进程未调用 wait() 或 waitpid() 函数回收子进程的资源是子进程变为僵尸进程的原因。

僵尸进程测试程序如下：

#include <stdio.h>

#include <unistd.h>

#include <errno.h>

#include <stdlib.h>

int main(int argc, char \*argv[])

{

    pid\_t pid;

    pid = fork();   //创建进程

    if( pid < 0 ){ // 出错

        perror("fork error:");

        exit(1);

    }else if( 0 == pid ){ // 子进程

        printf("I am child process.I am exiting.\n");

        printf("[son id]: %d\n", getpid() );

        exit(0);

    }else if( pid > 0){ // 父进程

        // 父进程没有调用 wati() 或 watipid()

        sleep(1); // 保证子进程先运行

        printf("I am father process.I will sleep two seconds\n");

        printf("[father id]: %d\n", getpid() );

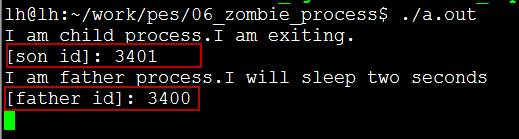
        while(1); // 不让父进程退出

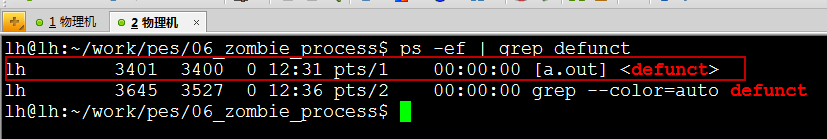
    }

    return 0;

}

运行结果：



在终端敲：ps -ef | grep defunct，后面尖括号里是 defunct 的都是僵尸进程。  


# 如何避免僵尸进程？

1）最简单的方法，父进程通过 wait() 和 waitpid() 等函数等待子进程结束  
  
2）如果父进程要处理的事情很多，不能够挂起，通过 signal()  函数人为处理信号 SIGCHLD，只要有子进程退出自动调用指定好的回调函数，可以在其回调函数里调用 wait() 或 waitpid() 回收。。  
  
测试代码如下：

#include <stdio.h>

#include <unistd.h>

#include <errno.h>

#include <stdlib.h>

#include <signal.h>

void sig\_child(int signo)

{

    pid\_t  pid;

    //处理僵尸进程, -1 代表等待任意一个子进程, WNOHANG代表不阻塞

    while( (pid = waitpid(-1, NULL, WNOHANG)) > 0 ){

        printf("child %d terminated.\n", pid);

    }

}

int main()

{

    pid\_t pid;

    // 创建捕捉子进程退出信号

    // 只要子进程退出，触发SIGCHLD，自动调用sig\_child()

    signal(SIGCHLD, sig\_child);

    pid = fork();   // 创建进程

    if (pid < 0){ // 出错

        perror("fork error:");

        exit(1);

    }else if(pid == 0){ // 子进程

        printf("I am child process,pid id %d.I am exiting.\n",getpid());

        exit(0);

    }else if(pid > 0){ // 父进程

        sleep(2);   // 保证子进程先运行

        printf("I am father, i am exited\n\n");

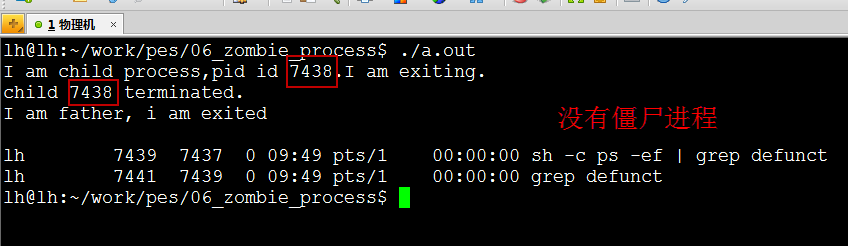
        system("ps -ef | grep defunct"); // 查看有没有僵尸进程

    }

    return 0;

}

运行结果：



3）如果父进程不关心子进程什么时候结束，那么可以用signal（SIGCHLD, SIG\_IGN）通知内核，忽略信号SIGCHLD，那么子进程结束后，内核自动会回收

#include <stdio.h>

#include <unistd.h>

#include <errno.h>

#include <stdlib.h>

#include <signal.h>

int main()

{

    pid\_t pid;

    // 忽略子进程退出信号的信号

    // 那么子进程结束后，内核会回收， 并不再给父进程发送信号

    signal(SIGCHLD, SIG\_IGN);

    pid = fork();   // 创建进程

    if (pid < 0){ // 出错

        perror("fork error:");

        exit(1);

    }else if(pid == 0){ // 子进程

        printf("I am child process,pid id %d.I am exiting.\n",getpid());

        exit(0);

    }else if(pid > 0){ // 父进程

        sleep(2);   // 保证子进程先运行

        printf("I am father, i am exited\n\n");

        system("ps -ef | grep defunct"); // 查看有没有僵尸进程

    }

    return 0;

}

运行结果：

