Linux进程、线程和调度(1)

讲解时间：9月13日、9月15日、9月19日、9月22日晚20点

宋宝华 <21cnbao@gmail.com>

报名直播或者录播：

http://edu.csdn.net/course/detail/5995

扫描二维码报名

麦当劳喜欢您来，喜欢您再来

扫描关注

Linuxer

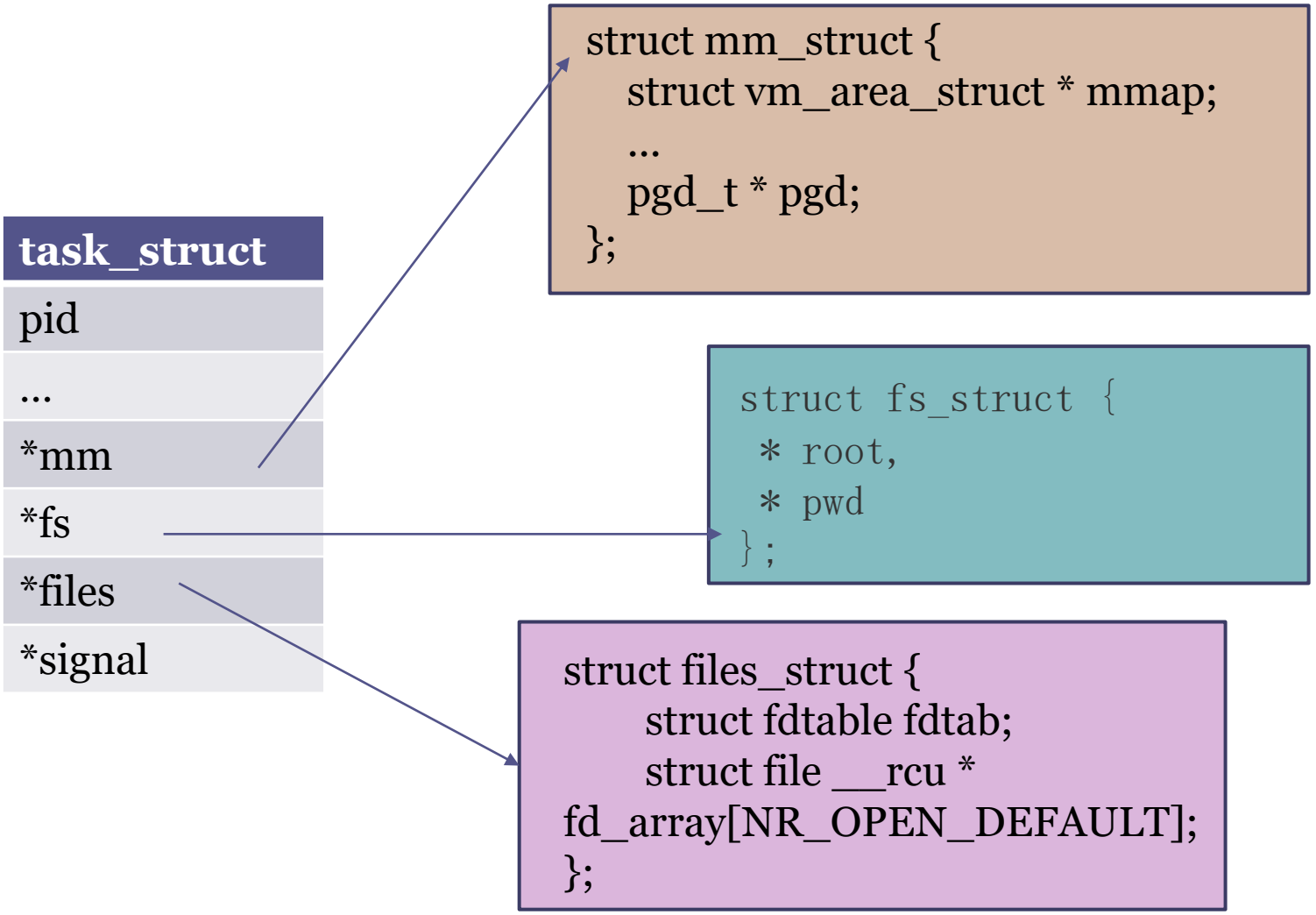
9.13第一次课大纲

* Linux进程生命周期(就绪、运行、睡眠、停止、僵死)
* 僵尸是个什么鬼？
* 停止状态与作业控制，cpulimit
* 内存泄漏的真实含义
* task_struct以及task_struct之间的关系
* 初见fork和僵尸

练习题

* fork的例子
* life-period例子，观察僵尸
* 用cpulimit控制CPU利用率

进程控制块PCB


task_struct
pid
...
*mm
*fs
*files
*signal
struct mm_struct {
struct vm_area_struct * mmap;
...
pgd_t * pgd;
};
struct fs_struct {
* root,
* pwd
};
struct files_struct {
struct fdtable fdtab;
struct file __rcu *
fd_array[NR_OPEN_DEFAULT];
};

pid

pid的数量是有限的

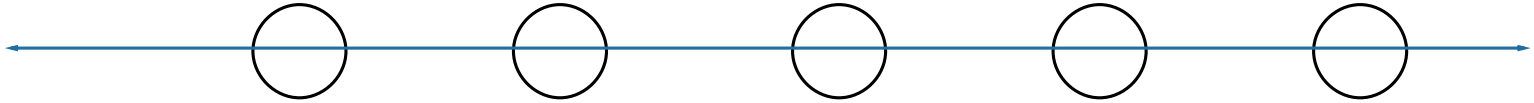
```
$ cat /proc/sys/kernel/pid_max
32768
```

Fork炸弹

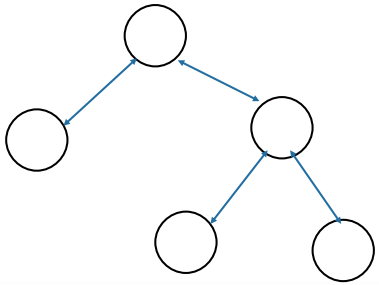
```
:(){ :|:& };:
```

task_struct被管理

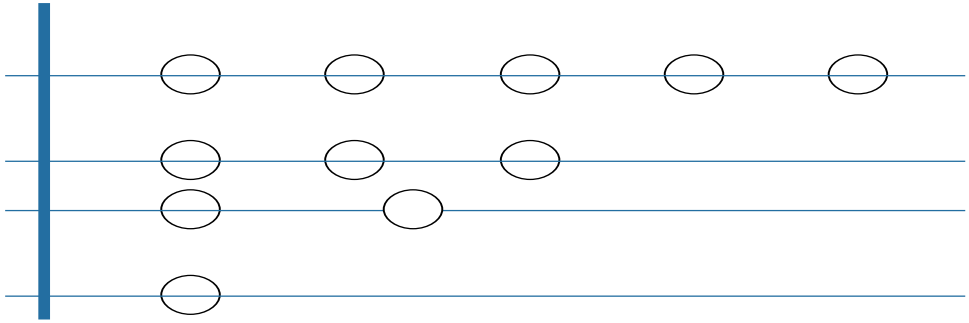
形成链表



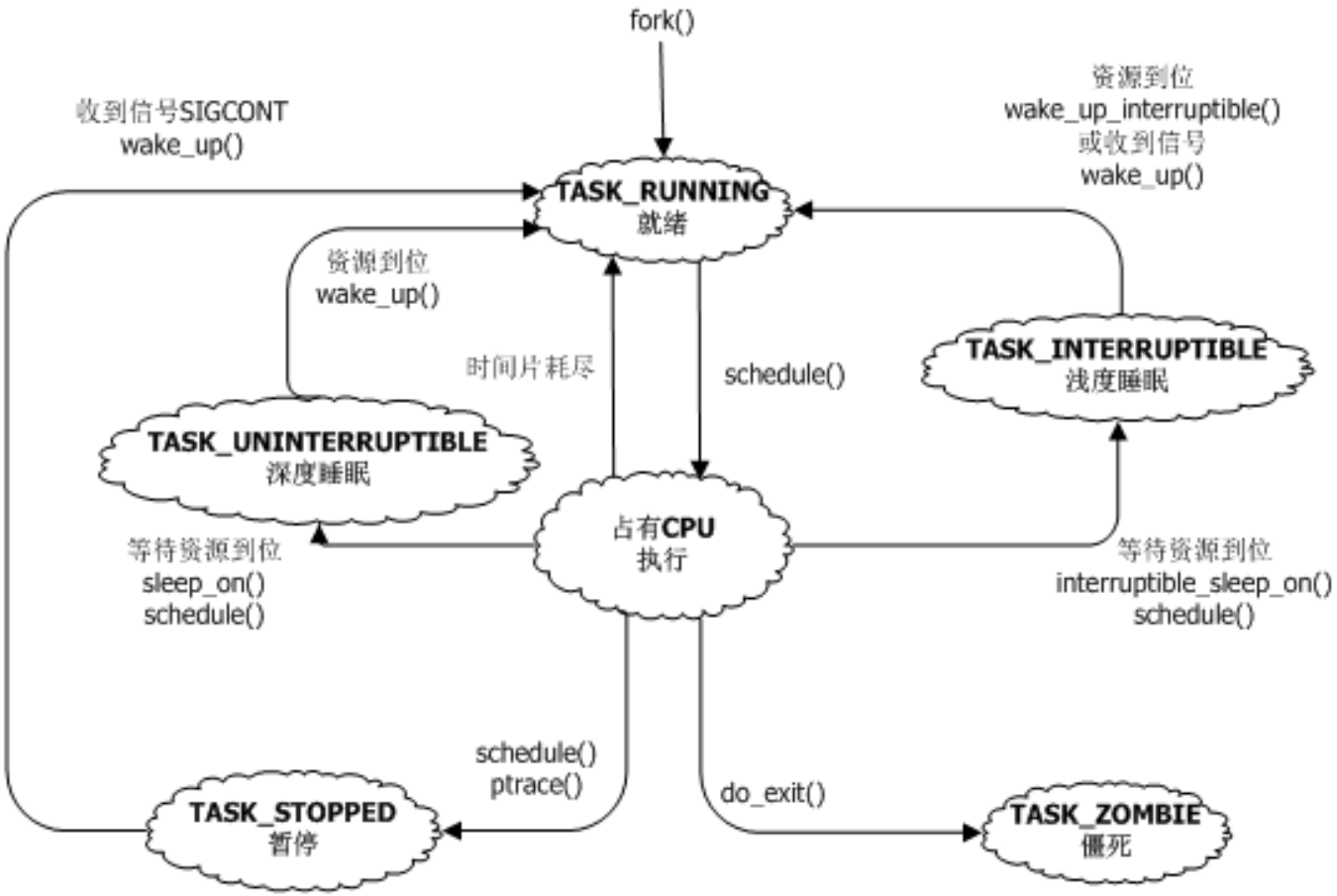
形成树



形成哈希：pid-> task_struct



进程生命周期


fork()
收到信号SIGCONT
wake_up()
资源到位
wake_up_interruptible()
或收到信号
wake_up()
TASK_RUNNING
就绪
资源到位
wake_up()
时间片耗尽
schedule()
TASK_INTERRUPTIBLE
浅度睡眠
TASK_UNINTERRUPTIBLE
深度睡眠
占有CPU
执行
等待资源到位
sleep_on()
schedule()
等待资源到位
interruptible_sleep_on()
schedule()
schedule()
ptrace()
do_exit()
TASK_STOPPED
暂停
TASK_ZOMBIE
僵死

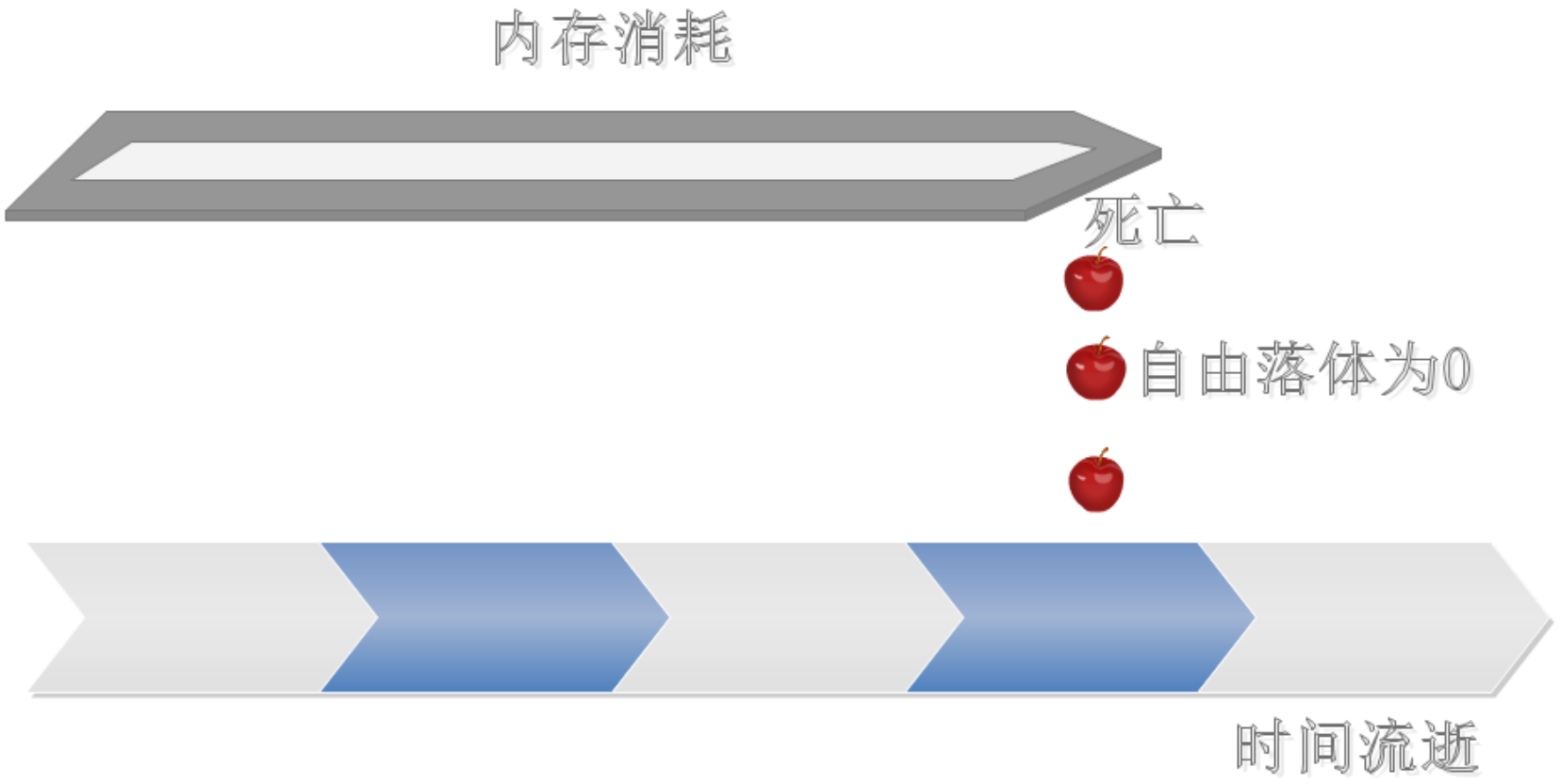
僵尸是什么

资源已经释放：无内存泄漏等
task_struct还在：父进程可以查到子进程死因

```
static int wait_task_zombie(struct wait_opts *wo, struct task_struct *p)
{
        int state, retval, status;
        pid_t pid = task_pid_vnr(p);
        uid_t uid = from_kuid_munged(current_user_ns(), task_uid(p));
        struct siginfo __user *infop;

        if (!likely(wo->wo_flags & WEXITED))
                return 0;

        if (unlikely(wo->wo_flags & WNOWAIT)) {
                int exit_code = p->exit_code;
                int why;

                get_task_struct(p);
                read_unlock(&tasklist_lock);
                sched_annotate_sleep();

                if ((exit_code & 0x7f) == 0) {
                        why = CLD_EXITED;
                        status = exit_code >> 8;
                } else {
                        why = (exit_code & 0x80) ? CLD_DUMPED : CLD_KILLED;
                        status = exit_code & 0x7f;
                }
```

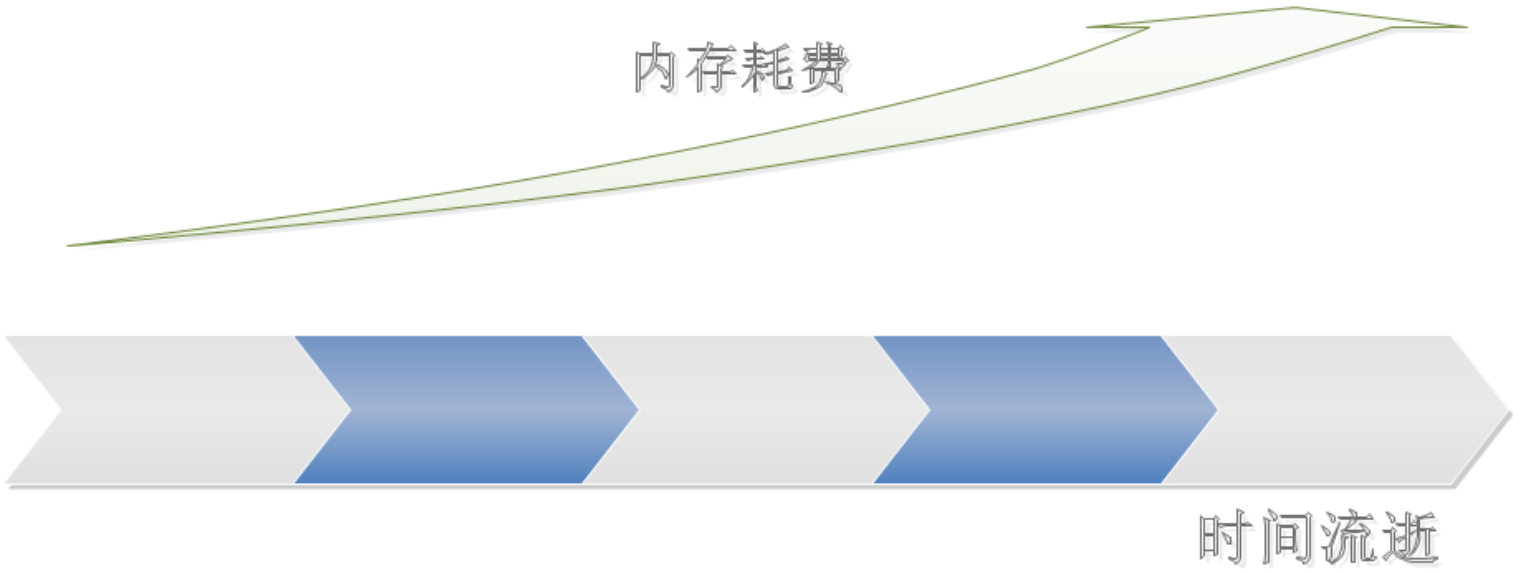
内存泄漏到底是什么?

不是：进程死了，内存没释放



内存泄漏到底是什么(cont.)?

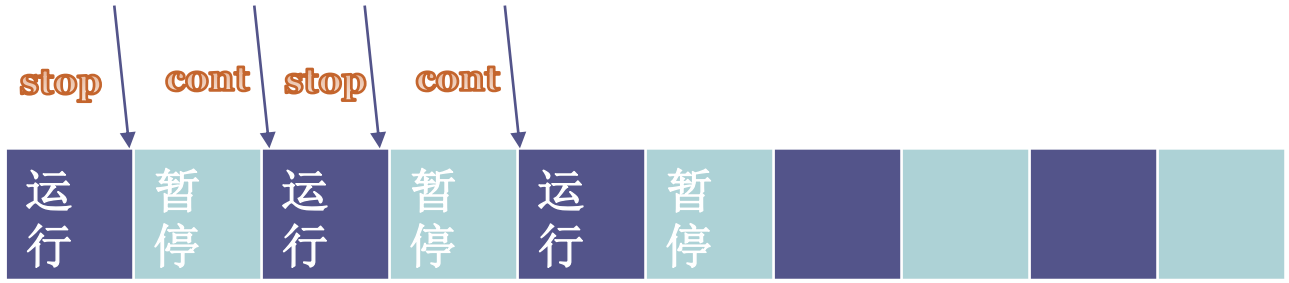
而是：进程活着，运行越久，耗费内存越多



作业控制

ctrl+ z, fg/bg
cpulimit

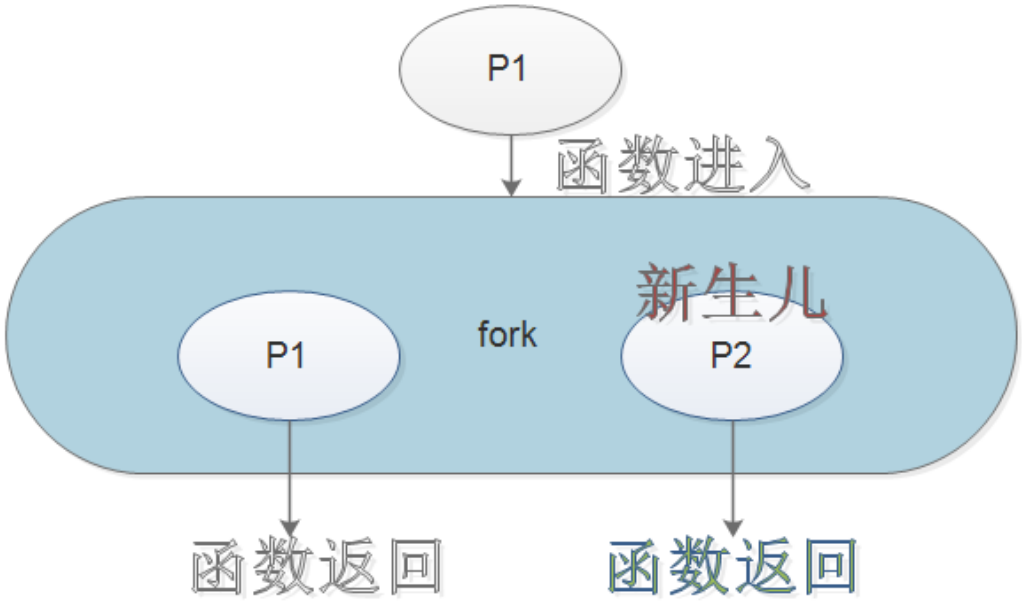
cpulimit -l 20 -p 10111
限制pid 为10111程序 的 cpu使用率不超过 10%



fork

打印几个hello？

```
1 main()
2 {
3         fork();
4         printf("hello\n");
5         fork();
6         printf("hello\n");
7         while(1);
8 }
```



fork(cont.)

怎么打印？

```
 6 int main(void)
 7 {
 8         pid_t pid,wait_pid;
 9         int status;
10
11         pid = fork();
12
13         if (pid==-1)    {
14                 perror("Cannot create new process");
15                 exit(1);
16         } else  if (pid==0) {
17                 printf("a\n");
18         } else {
19                 printf("b\n");
20         }
21
22         printf("c\n");
23         while(1);
24 }
```

子死父清场

```
pid = fork();

if (pid==-1)    {
        perror("Cannot create new process");
        exit(1);
} else  if (pid==0) {
        printf("child process id: %ld\n", (long) getpid());
        pause();
        _exit(0);
} else {
        wait_pid=waitpid(pid, &status, WUNTRACED | WCONTINUED);

        if (wait_pid == -1) {
                perror("cannot using waitpid function");
                exit(1);
        }

        if(WIFSIGNALED(status))
                printf("child process is killed by signal %d\n", WTERMSIG(status));
```

课程练习源码

https://github.com/21cnbao/process-courses

更早课程

- 《Linux总线、设备、驱动模型》录播：
http://edu.csdn.net/course/detail/5329

- 深入探究Linux的设备树
http://edu.csdn.net/course/detail/5627

- Linux进程、线程和调度
http://edu.csdn.net/course/detail/5995

谢谢！