CS334 Lab5 report

代码中如何区分父子进程？父子进程的执行顺序是否是固定的?

通过进程标识符（Process ID）来区分父子进程

执行顺序不固定

请回答第四步僵尸进程中列举的第4种情况的结果会是什么。

父进程不执行 wait() 并且父进程比子进程先结束会导致子进程会被 reparent 过继给 init 进程或者注册过的祖父进程（孤儿院）以确保每个进程都一定有父进程。

请编写一段c语言代码（截图），用于产生僵尸进程，并截图僵尸进程的状态(ps)。

c语言代码：

```c
#include<stdio.h>
#include<stdlib.h>
int main(){
    int rc = fork();
    if (rc < 0){
        printf("Fork() error\n");
    }
    else if (rc == 0){ // child process
        printf("In child Process, its pid = %d \n", getpid());
        while(1){

        }
    }else{// parent process

        printf("In parent process, its pid = %d\n", getpid());
    }
    return 0;

}
```

僵尸进程状态：

```
lmq@lmq-virtual-machine:~/Desktop/lab5$ In child Process, its pid = 7163
ps -al
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S  1000    1631    1624  1  80   0 - 81222 ep_pol tty2     00:00:38 Xorg
0 S  1000    1660    1624  0  80   0 - 48418 poll_s tty2     00:00:00 gnome-session-b
1 R  1000    6784    1556 94  80   0 -   624 -      pts/0    00:02:31 a.out
1 R  1000    6964    1556 85  80   0 -   624 -      pts/0    00:00:52 a.out
1 R  1000    7000    1556 80  80   0 -   624 -      pts/0    00:00:39 a.out
1 R  1000    7034    1556 76  80   0 -   624 -      pts/0    00:00:32 a.out
1 R  1000    7163    1556 62  80   0 -   624 -      pts/0    00:00:04 a.out
4 R  1000    7180    4789  0  80   0 -  3622 -      pts/0    00:00:00 ps
```

详细描述do_fork的函数调用过程，具体请包括do_fork内部调用（跳转至）了哪些函数，调用过程是怎样的，每个函数（包括do_fork）的作用是什么。请不要直接粘贴代码。

首先，调用`alloc_proc()`函数来分配并初始化进程的控制块

之后使用 `setup_stack()` 函数来分配并初始化内核栈

之后调用 `copy_mm()` 函数，根据 `clone_flags` 来决定是复制还是共享内存管理系统

再调用 `copy_thread()` 函数来设置进程的中断帧和上下文

之后设置好的进程加入链表

再将新建的进程设置为就绪态

最后将返回值设置为线程的 id

详细描述schedule的函数调用过程，具体请包括 schedule内部调用（跳转至）了哪些函数，调用过程是怎样的，每个函数（包括schedule）的作用是什么。请不要直接粘贴代码。

schedule函数的作用是跳转执行新的进程

首先，在根据调度算法找到下一个需要执行的 `next` 进程之后，调用 `proc_run()` 函数作用：将当前运行的进程设置为要切换过去的 `next` 进程，再将页表换成新进程的页表，之后使用 `switch_to` 切换到新进程，进行上下文的切换

`switch_to` 函数作用：保存和调换需要保存的寄存器

切换完成后执行 forkret，也就是 ra 寄存器指向的内容

之后 forkret 会把 sp 寄存器的值当做参数，sp 寄存器指向我们的 tf 跳转到 trapret

trapret 会把 tf 中的值当做要恢复现场的内容，恢复完成后会调到 epc 寄存器指向的位置，epc 指向 kernel_thread_entry

kernel_thread_entry 中会在 a0 寄存器，设置我们执行函数的参数，然后跳转到我们要执行的函数，指向结束之后，进入 do_exit 函数