1. 在fork()系统调用成功后，父进程会得到子进程的PID，而子进程会得到0。因此，可以通过判断PID是否为0来确定当前进程是父进程还是子进程。关于进程的执行顺序，是不确定的。
2. 子进程会变成孤儿进程，被init进程收养。
3.

```
dgy@dgy-ubuntu ~/g/o/lab5 (master)> ps -la
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000   14402   14378  0  80   0 - 41509 do_pol pts/2    00:00:00 fish
0 S  1000   30628   24588  0  80   0 -   655 hrtime pts/3    00:00:00 a.out
1 Z  1000   30629   30628  0  80   0 -     0 -      pts/3    00:00:00 a.out <defunct>
4 R  1000   30636   30584  0  80   0 -  3895 -      pts/4    00:00:00 ps
```

test.c > ...

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

int main() {
    pid_t pid = fork();
    if (pid < 0) {
        printf(format: "Fork error\n");
        exit(status: 1);
    } else if (pid == 0) {
        // 子进程
        printf(format: "I'm the child process\n");
        exit(status: 0);
    } else {
        // 父进程
        printf(format: "I'm the parent process\n");
        sleep(seconds: 10);
    }
    return 0;
}

```

4.

```c
current = idleproc;

int pid = kernel_thread(init_main, "Hello world!!", 0);
if (pid <= 0) {
    panic("create init_main failed.\n");
}
```

You, 42分钟前 • add lab5

这条指令的作用是创建一个内核线程，该线程的入口函数是init_main，它的参数为"Hello world!!"和0。

在这条指令中，init_main函数被作为创建线程的入口函数，"Hello world!!"作为参数传递给init_main函数，0则表示该线程不共享进程空间，即是一个独立的线程。

5. 当一个进程调用fork系统调用时，会进入内核空间执行do_fork函数。在do_fork函数中，会创建新的进程并设置它的context。当do_fork函数成功创建新进程后，会调用forkret函数，将新进程的返回值放到eax寄存器中，并将栈顶指针esp设置为新进程的内核栈顶。forkrets函数会从栈中读取父进程的状态，包括寄存器的值和栈指针的值，然后将它们装入对应的寄存器和内存地址中。最后，它会调用iret指令，将控制权切换回父进程，继续执行父进程的代码。