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. (1). do_fork函数会调用copy_process函数，复制当前进程的进程描述符，并创建一个新的进程。
(2).在copy_process函数中，会为新进程分配一个新的PID号，并将它的状态设置为TASK_RUNNING。会将进程描述符中的一些信息复制到新进程的进程描述符中。
(3). copy_process函数会调用alloc_thread_info函数为新进程分配一个新的进程线程信息结构(thread_info)。
(4).copy_process函数会调用copy_mm函数，为新进程分配一个新的虚拟地址空间，并将它的内存映射从父进程中复制到子进程中。如果父进程有共享内存或映射到共享库的区域，则这些区域也将被复制到子进程中

(5). copy_process函数会调用dup_task_struct函数，为新进程的内核栈分配空间，并复制父进程的内核栈中的内容到新进程的内核栈中。
(6).copy_process函数会为新进程设置CPU寄存器的值，包括设置新进程的堆栈指针和返回地址。
(7).copy_process函数会调用wake_up_new_task函数，唤醒新进程，使它可以开始执行。

(8).do_fork函数返回新进程的PID号，并在父进程中返回子进程的pid。

5. (1). 在schedule函数中，会根据调度策略选择下一个需要执行的进程，并将CPU的控制权交给该进程。
(2). 在切换进程之前，schedule函数会调用local_intr_save函数，将本地中断关闭。
(3).在switch_to函数中，会将当前进程的CPU寄存器的值保存到当前进程的进程控制块(task_struct)中，并将下一个进程的CPU寄存器的值从其进程控制块中读取出来，并写入到CPU寄存器中。
(4).当进程切换完成之后，会调用进程切换完成的回调函数，通过local_intr_restore函数将中断重新打开，以便CPU能够接收到中断信号。