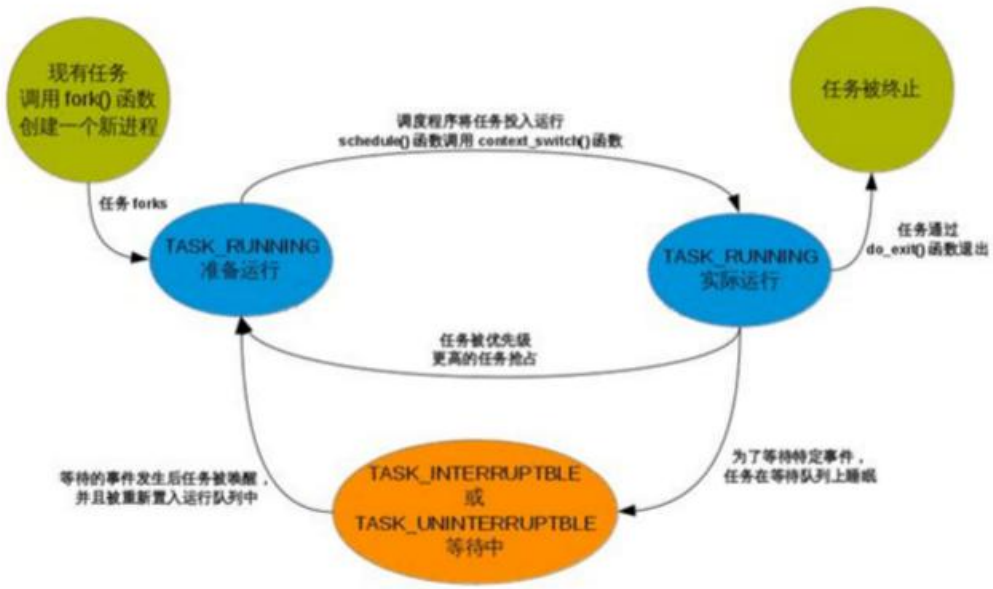
困难一：对 fork/exec/wait/exit 的理解与系统调用

- fork：在执行了 fork 系统调用之后，会执行正常的中断处理流程，最终将控制权转移给 syscall，之后根据系统调用号执行 sys_fork 函数，进一步执行了上文中的 do_fork 函数，完成新的进程的进程控制块的初始化、设置、以及将父进程内存中的内容到子进程的内存的复制工作，然后将新创建的进程放入可执行队列（runnable），这样的话在之后就有可能由调度器将子进程运行起来了；
- exec：在执行了 exec 系统调用之后，会执行正常的中断处理流程，最终将控制权转移给 syscall，之后根据系统调用号执行 sys_exec 函数，进一步执行了上文中的 do_execve 函数，在该函数中，会对内存空间进行清空，然后将新的要执行的程序加载到内存中，然后设置好中断帧，使得最终中断返回之后可以跳转到指定的应用程序的入口处，就可以正确执行了；
- wait：在执行了 wait 系统调用之后，会执行正常的中断处理流程，最终将控制权转移给 syscall，之后根据系统调用号执行 sys_wait 函数，进一步执行了的 do_wait 函数，在这个函数中，将搜索是否指定进程存在着处于 ZOMBIE 态的子进程，如果有的话直接将其占用的资源释放掉即可；如果找不到这种子进程，则将当前进程的状态改成 SLEEPING 态，并且标记为等待 ZOMBIE 态的子进程，然后调用 schedule 函数将其当前线程从 CPU 占用中切换出去，直到有对应的子进程结束来唤醒这个进程为止；

• exit：在执行了 exit 系统调用之后，会执行正常的中断处理流程，最终将控制权转移给 syscall，之后根据系统调用号执行 sys_exit 函数，进一步执行了的 do_exit 函数，首先将释放当前进程的大多数资源，然后将其标记为 ZOMBIE 态，然后调用 wakeup_proc 函数将其父进程唤醒（如果父进程执行了 wait 进入 SLEEPING 态的话），然后调用 schedule 函数，让出 CPU 资源，等待父进程进一步完成其所有资源的回收；

困难二：ucore 中一个用户态进程的执行状态生命周期图。



吐槽一：用户进程的状态切换的函数调用与切换不好理解。

吐槽二：这个实验需要分析的代码较多，调用关系也更复杂。