hw2

1. 不能，因为就绪态的进程没有执行，因此不会运行其程序导致其变为阻塞状态
2. 运行进程小于等于4，就绪态不超过n-4个，阻塞态可以取[0,n]范围内整数
3. ULT的创建、切换、销毁不需要内核干预，减少开销。同时也能自定义调度算法与优先级，更灵活高效
4. 每个线程都有自己的局部变量、函数调用、返回地址等信息，这些信息都需要保存在栈中。如果多个线程共享栈，会造成栈空间的混乱和冲突。

   对于内核级线程，线程由内核进行管理，每个线程应当独立地被内核调度，当一个线程进行系统调用等操作或是发生异常时，其他线程应当不受影响，因此每个线程都应当存在一个核心栈。

   对于用户级线程，线程栈是否也要分为用户栈和核心栈，取决于用户级线程的实现方式。如果用户级线程使用了操作系统提供的系统调用或信号处理机制，那么它也需要有一个核心栈来存储内核态下的执行信息。如果用户级线程完全在用户空间实现，并不依赖操作系统的服务，那么它只需要有一个用户栈即可。
5. 若是单线程，每个请求处理时间期望为15*2/3+90*1/3=40ms，平均每秒处理1000/40=25个请求

   若是多线程，磁盘平均处理时间为75*1/3=25ms，CPU平均处理时间为15ms，二者可以并行，平均每秒处理1000/25=40个请求
6. 对于linux，其线程由内核直接管理调度，线程和进程共享大部分资源，故线程的创建和销毁开销比较小。同时linux使用抢占式的调度方案，线程可以被中断和恢复，可以支持更加细颗粒度的开发。linux也提供大量的线程同步和通信机制，使得线程编程更加方便。但也有一些缺点，如线程与进程之间的资源共享会导致资源竞争，需要更复杂的锁设计。而且linux线程由内核直接管理，其切换开销比用户线程更大。

   对于windows，可以根据实现方式自由选择使用线程还是进程，从而避免在内核空间和用户空间之间频繁切换，提高程序效率和安全性。但这也导致创建和销毁线程的开销较大和线程的优先级问题。

   总的来说，在需要高度并发和轻量级线程的场景下，Linux线程机制更适合；而在需要更丰富的线程同步和通信机制以及更好的开发体验的场景下，Windows线程机制更适合。