《进程》章节练习

1. 在Linux中，若在一个多线程进程的某个线程中使用fork创建一个子进程，则在子进程中会存在几个线程？创建出的子进程可能会导致什么问题？Linux为何要如此设计Fork的语义？

- 仅会将发起调用的线程复制到子进程中，其他线程均在子进程中立即停止并消失，并且不会为这些线程调用清理函数以及针对线程局部存储变量的析构函数
- 一个线程在fork()被调用前锁定了某个互斥量，且对某个全局变量的更新也做到了一半，此时fork()被调用，所有数据及状态被拷贝到子进程中，那么子进程中对该互斥量就无法解锁（因为其并非该互斥量的属主），如果再试图锁定该互斥量就会导致死锁。同时，全局变量的状态也可能处于不一致的状态，因为对其更新的操作只做到了一半对应的线程就消失了
- 因为长期以来程序都是单线程的，fork()运转正常。且复制所有进程很容易导致错误，比如有的线程因为执行系统调用而被挂起

  参考：https://zhuanlan.zhihu.com/p/130873706

2. vfork是Linux中一个和fork十分相似的系统调用，关于vfork的设计详情，可以参考man vfork命令的输出内容。请回答，对于分配大量内存的应用，为何及时采用了写时拷贝（Copy-on-write，COW）优化后，fork的性能仍然比vfork要差？请尝试利用vfork的思路对fork进行优化，从而在不改变fork语义的前提下，提升fork的性能。

因为fork虽然不拷贝实际内存，但是要拷贝内存映射，而vfork直接让父子进程共享同一地址空间，不需要拷贝映射，在应用程序消耗大量内存时，拷贝内存映射也是一个很大的开销。对于这种应用，可以采用写时拷贝映射的方式，在fork时仅复制内核的数据结构，不进行页表项的复制。页表项的复制推迟到子进程修改数据时进行

3. 在像ChCore一样的单进程多线程的操作系统中，操作系统通常以线程为粒度进行调度。在一些调度的实现中，在从属于同一个进程的两个不同线程之间进行上下文切换所消耗的时间比在从属于不同进程的两个不同线程之间进行切换耗时更低，试解释其原因。

前者的虚拟地址空间还是一样的，只需要修改一下寄存器，PC等。而后者要修改内存映射、页表等，这是一个很大的开销

4. 在ChCore中，内核会在上下文切换的过程中切换属于不同进程的页表。于此同时，ChCore内核中同样使用虚拟地址进行内存访问，需要根据页表进行地址翻

译。在这一设计下，如何确保上下文切换过程中对页表的切换不会影响到内核的正常执行?

因为ARM的内核态和用户态不共享页表，在上下文切换的时候只会改变TTBR0，而不改变TTBR1

5. 对于像协程一样的用户态线程而言，由于所有的协程均从属于同一个内核态线程，因此同一时间仅能同时运行一个协程。在这一情况下，为何协程仍然能够提升应用性能？对于哪类应用协程能够尽可能高的提升系统性能?

- 线程很多的时候，线程切换的开销很大，使用协程可以避免这种开销
- 需要管理长时间运行的任务时，使用协程可以避免任务阻塞主线程并导致应用冻结