1. 为什么要有上下文切换？

（1）问题：

系统不可能只有一个程序在运行，可能多个程序（任务）在运行，而一个CPU只能同时执行一个任务。

那如何让用户感觉多个任务在执行呢？

（2）解决：

使用时间片轮转方法，CPU给每个任务都服务一定时间。

此时，就涉及到任务状态切换与保存，所以在切换的时候需要将上一个任务的状态保存下来，然后再加载下一个任务状态。即上下文切换。

1. 什么会导致上下文切换？

中断处理：在中断处理中，其他程序”打断”了当前正在运行的程序。当CPU接收到中断请求时，会在正在运行的程序和发起中断请求的程序之间进行一次上下文切换。中断分为硬件中断和软件中断，软件中断包括因为IO阻塞、未抢到资源（锁）或者用户代码等原因，线程被挂起。

多任务处理：在多任务处理中，CPU会在不同程序之间来回切换，每个程序都有相应的处理时间片，CPU在两个时间片的间隔中进行上下文切换。

用户态切换：对于一些操作系统，当进行用户态切换时也会进行一次上下文切换，虽然这不是必须的。

当前执行任务的时间片用完之后，系统CPU正常调度下一个任务；

当前执行任务碰到IO阻塞，调度器将此任务挂起，继续下一任务；

多个任务抢占锁资源，当前任务没有抢到锁资源，被调度器挂起，继续下一任务；

用户代码挂起当前任务，让出CPU时间；

硬件中断；

1. 上下文切换
2. 上下文含义：某个时间点，PC和CPU寄存器的信息
3. 上下文切换：操作系统对进程（或者线程）的动作，包括以下活动

挂起进程（线程）：将当前进程（线程）上下文存储在PCB（TCB）中

恢复进程（线程）：检索下一个进程（线程）PCB（TCB）的上下文，将上下文恢复

执行进程（线程）：跳到PC位置执行（即跳转到进程被中断代码处）

1. 细分上下文切换步骤

进程A -> 进程B

1. 保存进程A上下文（寄存器、PC）
2. 更新进程A的PCB状态（“阻塞态”、“就绪态”等）
3. 将进程A的PCB加入相应状态的队列（就绪队列、阻塞队列）
4. 将进程B的PCB改为“运行态”，并恢复进程B的上下文，执行B进程
5. 细分保存进程（线程）上下文

进程：

1. 切换页目录以使用新的地址空间（页表）
2. 切换内核栈与硬件上下文（PC与寄存器）

线程：

1. 切换内核栈与硬件上下文（PC与寄存器）
2. 切换所带来的损耗：

直接消耗：指的是CPU寄存器需要保存和加载, 系统调度器的代码需要执行, TLB实例需要重新加载, CPU 的pipeline需要刷掉；

间接消耗：指的是多核的cache之间得共享数据, 间接消耗对于程序的影响要看线程工作区操作数据的大小；

1. 减少切换
2. 减少锁：

无锁并发：多线程竞争时，会引起上下文切换，所以多线程处理数据时，可以用一些办法来避免使用锁，如将数据的ID按照Hash取模分段，不同的线程处理不同段的数据；

CAS算法：Java的Atomic包使用CAS算法来更新数据，而不需要加锁；

（2）减少线程：

最少线程：避免创建不需要的线程，比如任务很少，但是创建了很多线程来处理，这样会造成大量线程都处于等待状态；

使用协程：在单线程里实现多任务的调度，并在单线程里维持多个任务间的切换；

1. TCB线程控制块（每线程一个PC）

线程标识符

一组寄存器

通用寄存器

程序计数器PC（每线程一个PC，代码执行位置，若遇到return就停止；不同线程其PC肯定不同，因为执行的业务不同）

状态寄存器

线程运行状态

优先级

线程专有存储区

信号屏蔽

堆栈指针

1. PCB进程控制块

进程状态：可以是new、ready、running、waiting或halted等。

程序计数器：接着要运行的指令地址。

CPU寄存器：如累加器、索引寄存器（Index Register）、堆栈指针以及一般用途寄存器、状况代码等，主要用途在于中断时暂时存储数据，以便稍后继续利用；其数量及类因计算机架构有所差异。

CPU排班法：优先级、排班队列等指针以及其他参数。

存储器管理：如分页表（PageTable）等。

会计信息：如CPU与实际时间之使用数量、时限、帐号、工作或进程号码。

输入输出状态：配置进程使用I/O设备，如磁带机。

第二种描述[2]

（1）进程标识符（内部，外部）

（2）处理机的信息：通用寄存器，指令计数器，PSW，用户的栈指针。

（3）进程调度信息：进程状态，进程的优先级，进程调度所需的其它信息，事件。

（4）进程控制信息：程序的数据的地址，资源清单，进程同步和通信机制，链接指针。