Introduction to Thread, Process, and Parallel

1. 我们知道计算机的核心是CPU，它就像是一座工厂。

2. 但工厂的电力有限，每次只能给一个车间使用。也就是说当，一个车间开工，其他车间就必须停工。这就对应着“每个CPU每次只能支持一个任务运行”。

3. Process（进程）就像是工厂的车间，它代表了CPU所能处理的单个任务。任意时刻，CPU总是运行一个进程，其他进程处于停工状态。

4. 每个车间（Process）里可以有多个工人，线程（Thread）就像是车间里的工人，一个进程（Process）中可以包含多个线程（Thread），多个线程之间可以相互协作，共同完成一个任务。

5. 车间（Process）的空间是工人们（Thread）共享的，就好比车间内有很多房间都是允许工人们随意进出的。这里提到的房间就好比计算机的内存空间，意味着一个进程（Process）的内存空间共享的，每个线程（Thread）都可以共享这些内存。

6. 但是每个内存的大小有所不同，有些内存可能只能同时容纳一个线程（Thread）。当有多个线程都想要使用这样的内存时，其他线程（Thread）必须等到当前线程使用完该内存后，才能占用这一块内存。 7. 从上面一步能发现，很有可能出现多个线程（Thread）想要同时占用同一个内存空间的情况。为了发生混乱，我们可以在门（内存空间）上加一把锁。先占用的Thread对内存空间上锁，从而让之后的Thread进行排队。这个锁就叫“互斥锁”（Mutual Exclusion, i.e. Mutex）。

8. 但并不是每个房间（内存空间）都是只能容纳一个Thread的。有些内存空间可以同时容纳 n 个Thread。同理，为了防止超过 n 个Thread同时占用这个内存空间，我们同样可以加 n 个Mutex（互斥锁）。

9. 这时候，除了加 n 个Mutex外，我们还要在门口加 n 个钥匙。每进去一个人（Thread）就取一把钥匙，出来时再把钥匙挂回原处。如果门口的钥匙架空了，就需要排队。这种方法叫做“信号量”（Semaphore），用来保证多个线程（Thread）之间的工作不会相互冲突。

10. 很容易发现，Mutex 其实是 Semaphore的一种特殊情况（n=1）。也就是说，理论上可以用后者来代替前者。但由于Mutex较为简单，且实现起来效率高，所

以在必须保证资源独占的情况下，还是采用Mutex的设计。
11. 综上，操作系统的设计可以归结为三点：
(1). 以多进程（Process）形式，允许多个任务同时运行；
(2). 以多线程（Thread）形式，允许单个任务分成不同的部分运行；
(3). 提供协调机制，一方面防止进程（Process）之间和线程（Thread）之间产生冲突，另一方面允许进程之间和线程之间贡献资源。是能够被scheduler进行独立管理的最小序列

How to handle topicalization

I'll just assume a tree structure like (1).

(1) Structure of A′ Projections:

CP

Spec C′

C SAgrP

Mood

Mood changes when there is a topic, as well as when there is WH-movement. *Irrealis* is the mood when there is a non-subject topic or WH-phrase in Comp. *Realis* is the mood when there is a subject topic or WH-phrase.

参考文献

https://www.ruanyifeng.com/blog/2013/04/processes_and_threads.html