需求分析

1. 睡眠和唤醒

既有的线程挂起机制为通过在timer\_sleep()中执行while()循环来实现。当前时间若不满足挂起的时间要求，则调用thread\_yield()函数继续循环；如果满足则直接压进就绪队列。因此实际上只存在两个态， Running 和 Ready,并没有真正意义上的睡眠与唤醒。查看源代码中 thread\_yield()函数的注释：

*/\* Yields the CPU. The current thread is not put to sleep and*

*may be scheduled again immediately at the scheduler's whim. \*/*

可以发现线程并没有真正进入睡眠，且该线程有可能又一次立即被调度，这样的结果就是产生了忙等待。

我们需要实现的目标为，在timer\_sleep()函数中使线程进入Block态。系统运行一段时间后，睡眠时间到，再对该线程进行唤醒，从Block态转入Ready态。

1. 优先级调度

优先级调度要求实现一个基于优先级的调度策略。包括解决优先级授予和其他授予时的相关问题。

优先级翻转描述如下：

线程 A,B,C 分别具有 1,2,3 的优先级(数字越大说明优先级越高),线程 A,B 目前在就绪队列中等待调度,线程 A 对一个互斥资源拥有线程锁。而此时,高优先级的线程 C 也想要访问这个互斥资源,线程C只 好在这个资源上等待,不能进入就绪队列。当调度器开始调度时,它只能从 A 和 B 中进行选择,根据优先 级调度原理,线程 B 将会首先运行。 这时就产生了一个问题,即本来线程C的优先级比线程B高,但是线程B却先运行了,从而产生了优先级翻转问题。

1. 高级调度

普通线程的调度策略是基于时间片轮转的优先级调度，总体来说，实现简单方便，体现了严格按优先级运行的严谨。

这种调度策略有一定的优点，不过，在某些情况下，这种策略存在一定弊端。这种调度策略极易造成线程的饥饿，当有高优先级且所需运行时间片较长的线程被创建，其余的线程只有在它行完之后才能重新获得 CPU，这种调度策略没有考虑到系统的吞吐量，无法减少线程的平均等待时间。这种弊端在大用户 量的交互系统中会被放大，造成用户长时间的等待会降低用户对于该系统的评价。

因此，我们需要一种新的调度策略，该策略在考虑线程优先级的基础之上，能够按照线程运行情况来动 态地考虑修改线程的优先级，于是根据执行情况反馈信息而对线程队列进行组织并调度线程的调度器多 级反馈队列调度器应运而生。