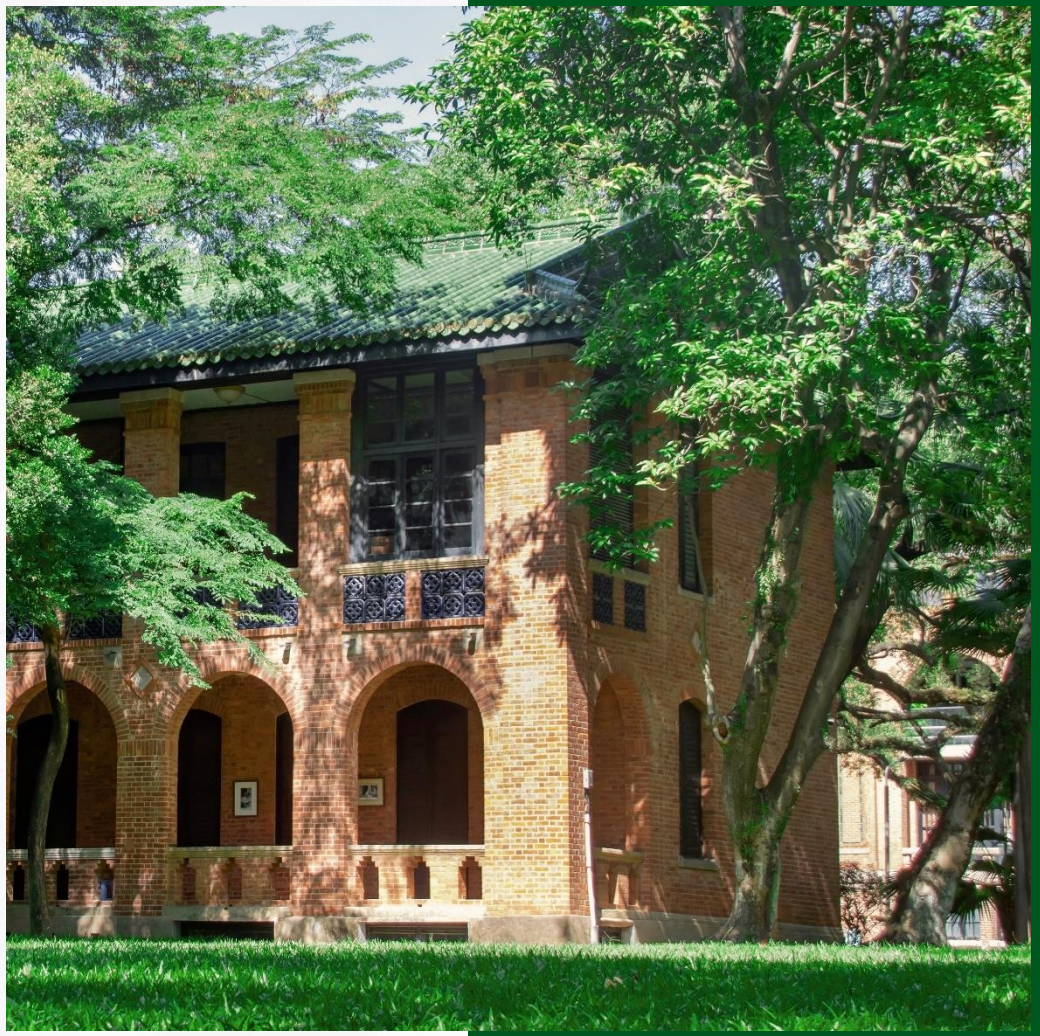
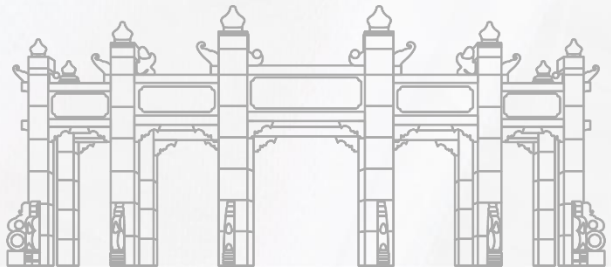
中山大學
SUN YAT-SEN UNIVERSITY

鸿蒙操作系统
进程与线程调度





HarmonyOS的进程调度与线程调度

1. 调度器设计原则
   - 要使得CPU尽量繁忙，不应浪费计算资源。
   - 要允许多任务有效共享系统资源。
   - 需要达到目标的QoS（Quality of Service）
   - 要避免整个系统的崩溃，例如调度器本身陷入死循环或者死锁。
2. 同时满足上述原则的调度器几乎不存在。
3. 特定的操作系统往往会根据自身的需要，选择合适的调度策略。
4. OpenHarmony LiteOS-A内核采用了高优先级优先 + 同优先级时间片轮转的抢占式调度机制。
5. OpenHarmony 的调度算法将tickless机制天然嵌入到调度算法中，一方面使得系统具有更低的功耗，另一方面也使得tick中断按需响应，减少无用的tick中断响应，进一步提高系统的实时性。
6. OpenHarmony的进程调度策略支持时间片轮转，线程调度策略支持时间片轮转和先进先出。

LiteOS-m调度策略

LiteOS-m采用优先级队列+FIFO的调度策略

- 优先级队列使用双向链表实现。
- 调度器维护一组不同优先级的Task队列。
- 每次都从这些Task中选择优先级最高的Task执行。
- 当有多个Task具备同样的优先级的时候，按照先入队先执行的方式来选择。

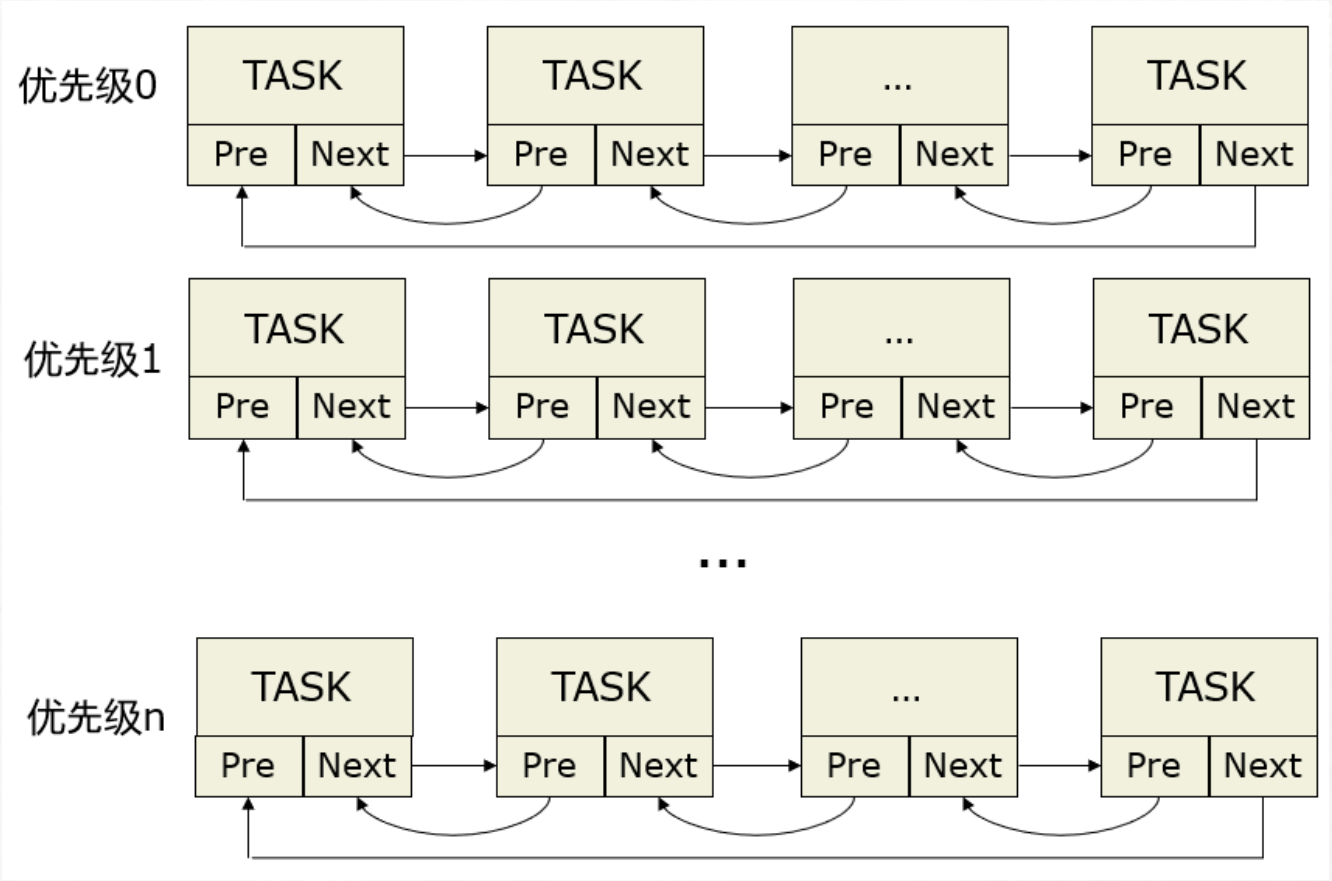


图4-1 LiteOS-m的线程调度模型

LiteOS-a调度策略

1. LiteOS-a采用进程和线程（Task）两级调度
   - 调度器先在进程优先级队列中找到优先级最高的进程，然后再在此进程的Task优先级队列中找到优先级最高的Task进行执行。
2. 支持更复杂的调度策略
   - FIFO：先进先出
   - RR（Round-Robin）：时间片轮转，LiteOS-a的RR机制采用的时间片粒度为10个Tick

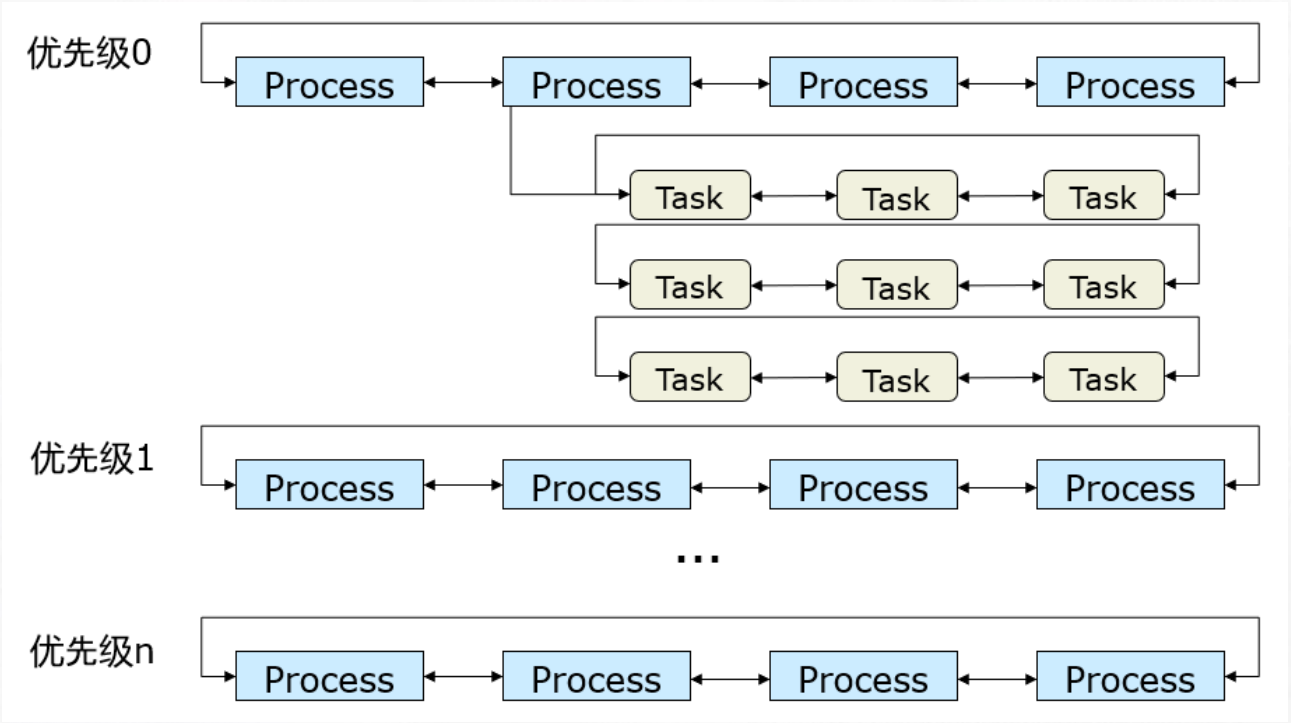


图4-2 LiteOS-a的调度模型

LiteOS-a调度器结构

LiteOS-a调度器主要使用到了位图Bitmap、队列和双向链表三种基本的数据结构。

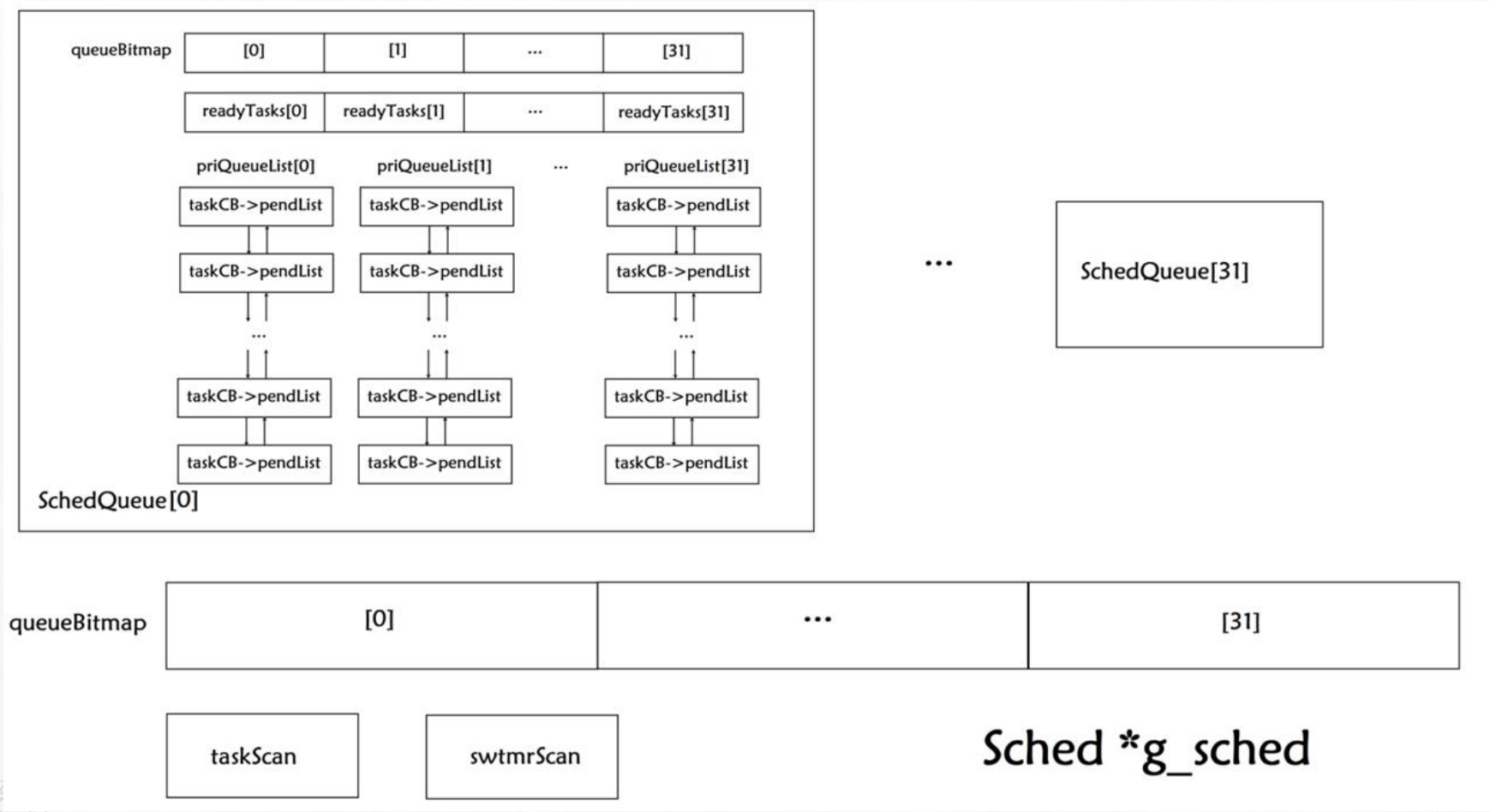


图4-3 LiteOS-a的调度器结构

LiteOS-a调度策略

- g_sched：全局调度器。
- taskScan：线程（任务）扫描器。
- swtmrScan：定时扫描器。
- queueBitmap（图中红色框）：队列位图。每一位对应一个进程优先级队列SchedQueue，用于标识对应优先级队列中是否有就绪进程。LiteOS一共设置了32个优先级队列。
- SchedQueue[i]：同一优先级的进程队列。队列的进程中又会包含多个线程Task。因此每一个SchedQueue[i]中还需要再对线程Task进行调度。

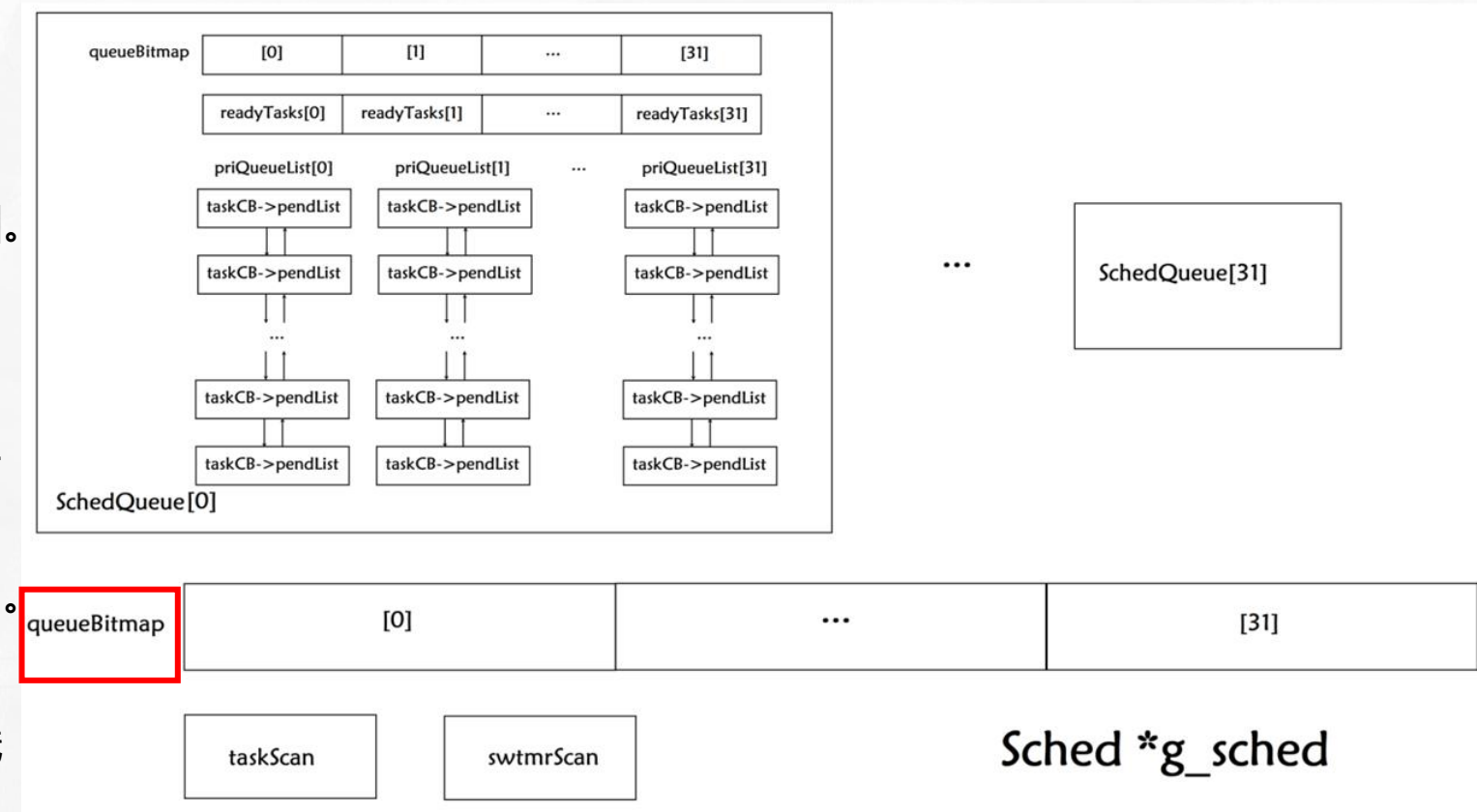


图4-3 LiteOS-a的调度器结构

LiteOS-a调度策略

- queueBitmap（图中蓝色框）：队列位图。每一位对应一个线程优先级队列，用于标识对应优先级队列中是否有就绪线程。
- readyTask：线程记录数组。表示对应的优先级线程队列中就绪线程的数量。
- priQueueList：各优先级线程就绪队列，默认32级。调度时会先调度优先级高的就绪线程。
- taskCB->pendlist：线程阻塞队列。这些线程可能在等待某些事件而被挂起。

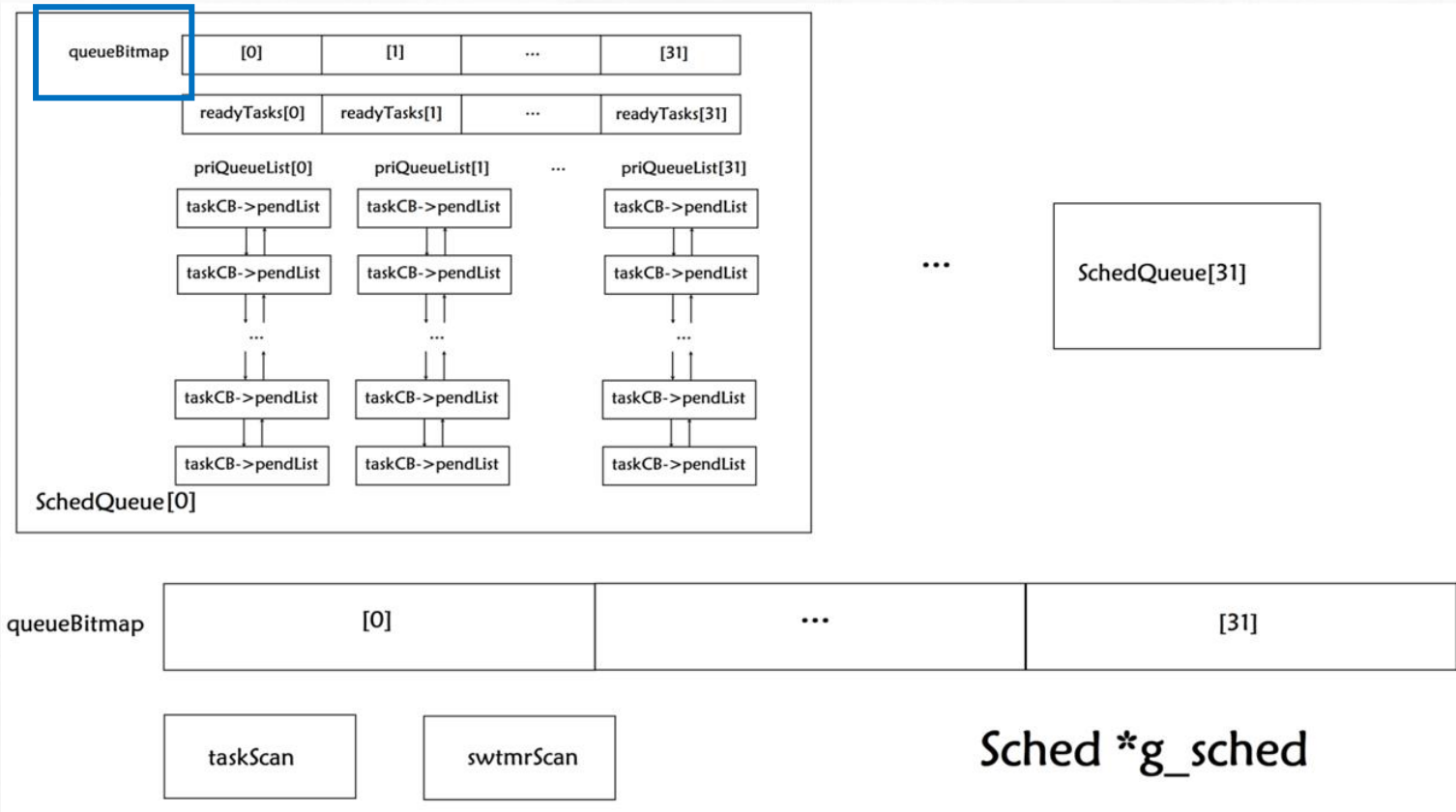


图4-3 LiteOS-a的调度器结构

LiteOS-a调度策略

LiteOS-a调度时机

- 事件处理(event)：在事件写的过程中会进行调度
- 快速锁(futex)：在锁的获取和释放阶段会进行调度
- 互斥量(mutex)：互斥量释放过程会进行调度。
- 队列(queue)：队列为空时进行调度。
- 线程相关事件：Task创建、Task暂停、Task处理等。
- 时间片结束。
- 等待线程超时。

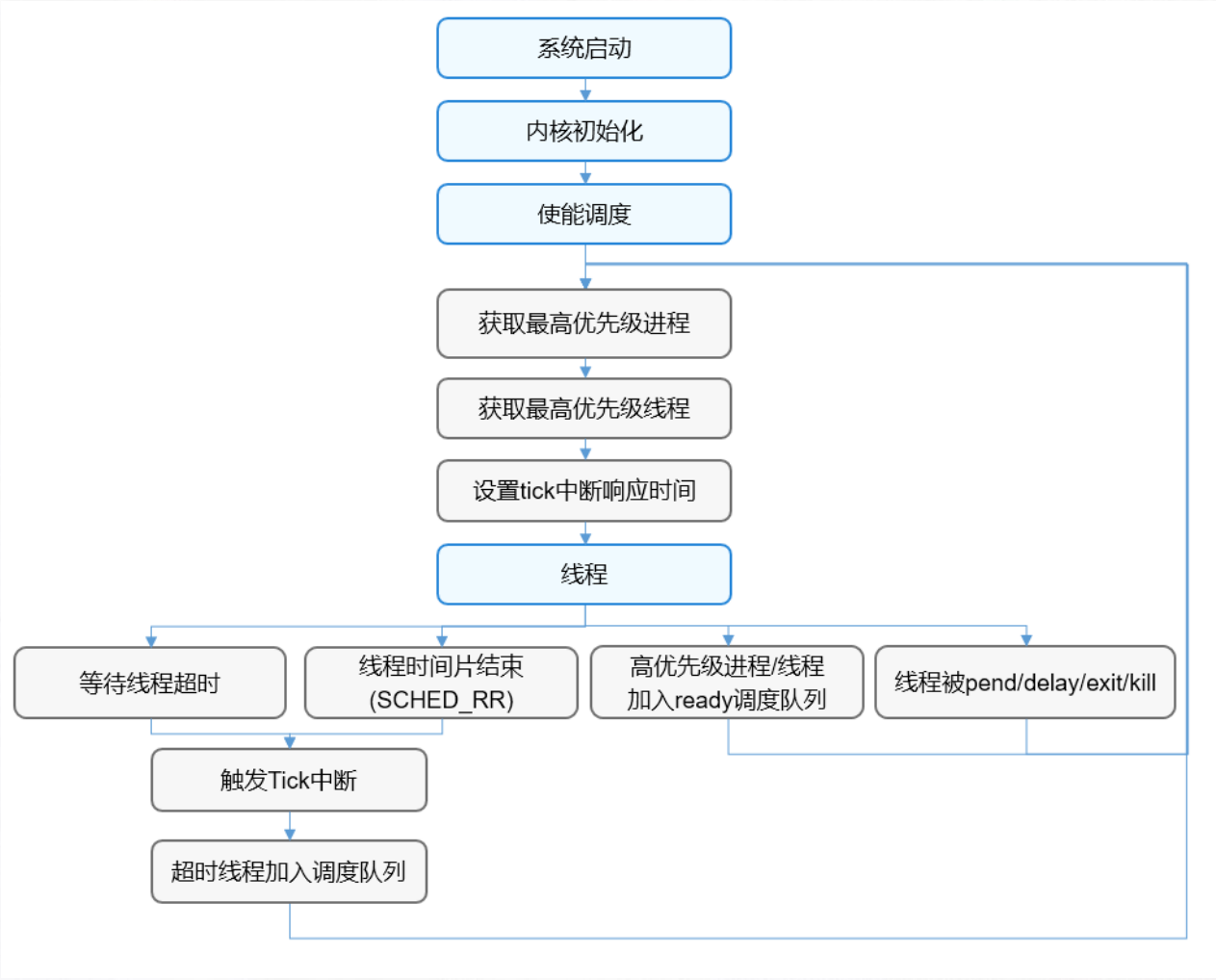


图4-4 LiteOS-a的调度流程

进程调度与线程调度的区别

1. 进程调度的单位是进程；线程调度的单位是线程。
2. 进程拥有资源的独立副本，每个进程都有自己的地址空间，因此进程切换需要更大的开销。而线程不拥有系统资源，同一个进程中的所有线程共享该进程的资源，线程的切换只需要保存线程的相关寄存器和计数器等信息。
3. 进程与进程之间是相互独立的，一个进程的崩溃不会直接影响到其他进程。而线程之间共享进程资源，一个线程出错会引起多个线程的运行。
4. 由于线程调度开销更小，因此线程的调度更为频繁。
5. 进程调度需要考虑到系统资源分配和所拥有的线程的状态，因此进程的调度策略更为复杂。

时钟系统

1. 时钟系统：分时或实时操作系统通过处理时钟中断，即处理时间事件，来获取特定的时间信息，并根据时间信息做出相应的动作。
2. Tick是操作系统调度的基本时间单位。
3. 时间系统主要为操作系统提供以下几项服务

   (1) 系统时间维护;

   (2) 进程/任务调度的相关服务;

   (3) 提供定时器;

   (4) 触发统计或调试相关操作;

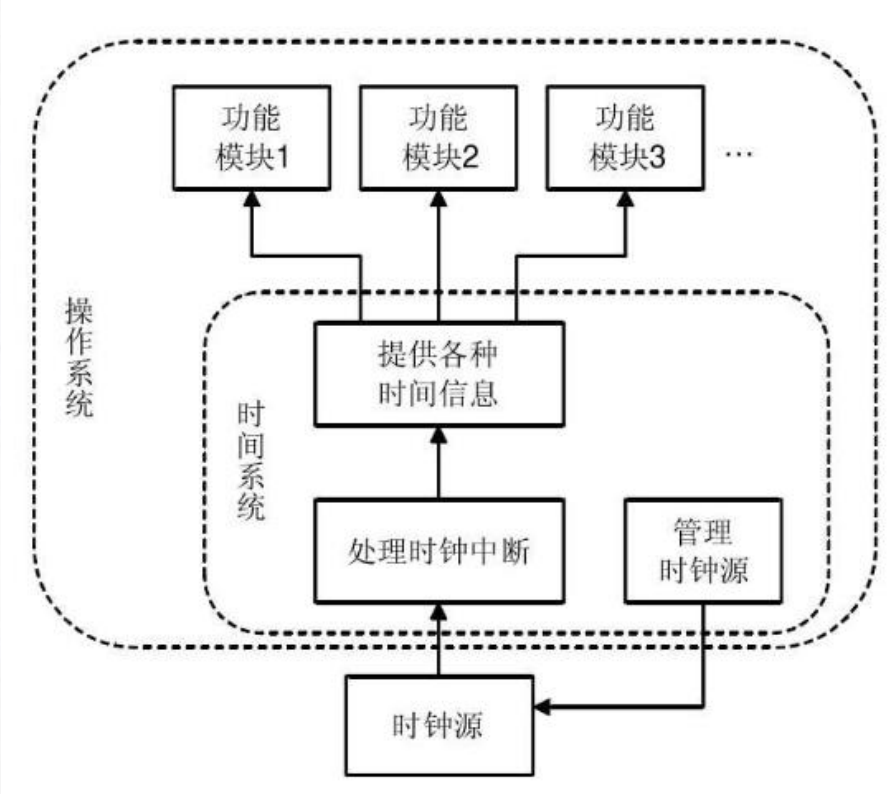


图4-5 时钟系统示意图

时钟管理

1. 时间管理以系统时钟为基础。时间管理提供给应用程序所有和时间有关的服务。系统时钟是由定时/计数器产生的输出脉冲触发中断而产生的，一般定义为整数或长整数。输出脉冲的周期叫做一个“时钟滴答”。
2. 系统时钟也称为时标或者Tick。一个Tick的时长可以静态配置。用户是以秒、毫秒为单位计时，而操作系统时钟计时是以Tick为单位的，当用户需要对系统操作时，例如任务挂起、延时等，输入秒为单位的数值，此时需要时间管理模块对二者进行转换。
3. Tick与秒之间的对应关系可以配置。
   - Cycle 系统最小的计时单位。Cycle的时长由系统主频决定，系统主频就是每秒钟的Cycle数。
   - Tick Tick是操作系统的基本时间单位，对应的时长由系统主频及每秒Tick数决定，由用户配置。
4. OpenHarmony系统的时间管理模块提供时间转换、统计、延迟功能以满足用户对时间相关需求的实现。

软件定时器

1. 软件定时器，是基于系统Tick时钟中断且由软件来模拟的定时器，当经过设定的Tick时钟计数值后会触发用户定义的回调函数。定时精度与系统Tick时钟的周期有关。
2. 硬件定时器受硬件的限制，数量上不足以满足用户的实际需求，因此为了满足用户需求，提供更多的定时器，LiteOS-A内核提供软件定时器功能。
3. 软件定时器的运行机制
   - 软件定时器是系统资源，在模块初始化的时候已经分配了一块连续的内存。
   - 软件定时器使用了系统的一个队列和一个任务资源，软件定时器的触发遵循队列规则，先进先出。同一时刻设置的定时时间短的定时器总是比定时时间长的靠近队列头，满足优先被触发的准则。
   - 软件定时器以Tick为基本计时单位，当用户创建并启动一个软件定时器时，OpenHarmony会根据当前系统Tick时间及用户设置的定时间隔确定该定时器的到期Tick时间，并将该定时器控制结构挂入计时全局链表。
   - 当Tick中断到来时，在Tick中断处理函数中扫描软件定时器的计时全局链表，看是否有定时器超时，若有则将超时的定时器记录下来。
   - Tick中断处理函数结束后，软件定时器任务（优先级为最高）被唤醒，在该任务中调用之前记录下来的定时器的超时回调函数。

调度器的Tick定时机制

1. 目前几乎所有分时操作系统以及大部分实时操作系统，它们的时间系统都使用传统的tick定时机制。
2. 该机制最突出的特点就是通过设置时钟源产生周期性的时钟中断来驱动操作系统的运转。
3. tick定时机制的缺陷
   - 周期性的时钟中断造成不必要的系统能耗：在操作系统运行的很多时段内，除了系统时间维护，时钟中断对系统的运转没有实际意义;
   - 周期性定时机制使系统定时精度难以提高：操作系统的定时以及其它时间信息处理的精度最高只能以ticks为单位。

- **调度器的Tick定时机制（例子）**

  周期任务：Task A，Task C。（经过固定一段时间就会发生的事件）

  突发任务：Task B。

  横轴为时间轴，T1、T2、T3是时间片。

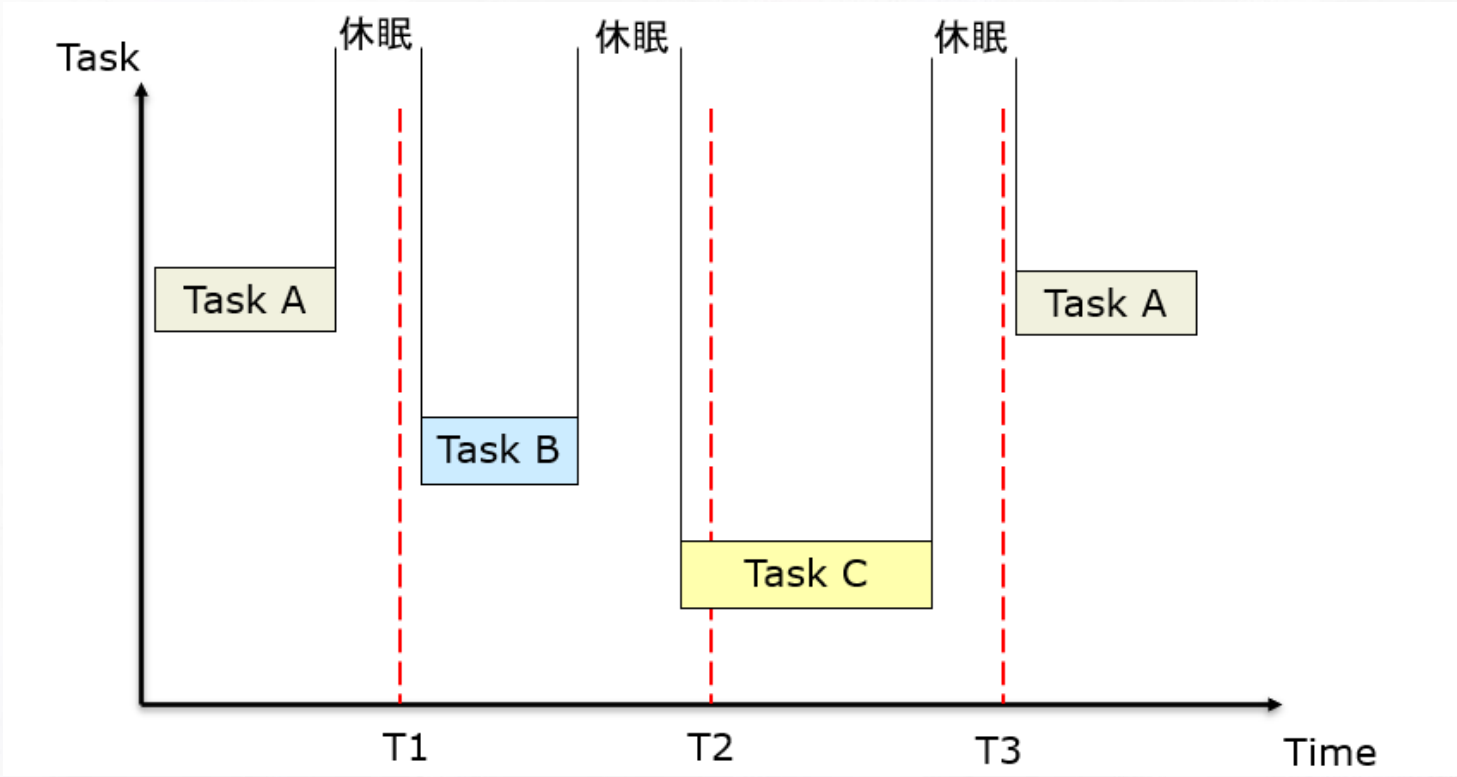


图4-6 Tick机制示意图

- **调度器的Tick定时机制（例子）**

  当使用Tick定时机制，在T1、T2和T3时刻都会出现时钟中断，导致处于休眠状态的处理器重新被唤醒，频繁地唤醒处理器将带来更多的开销。

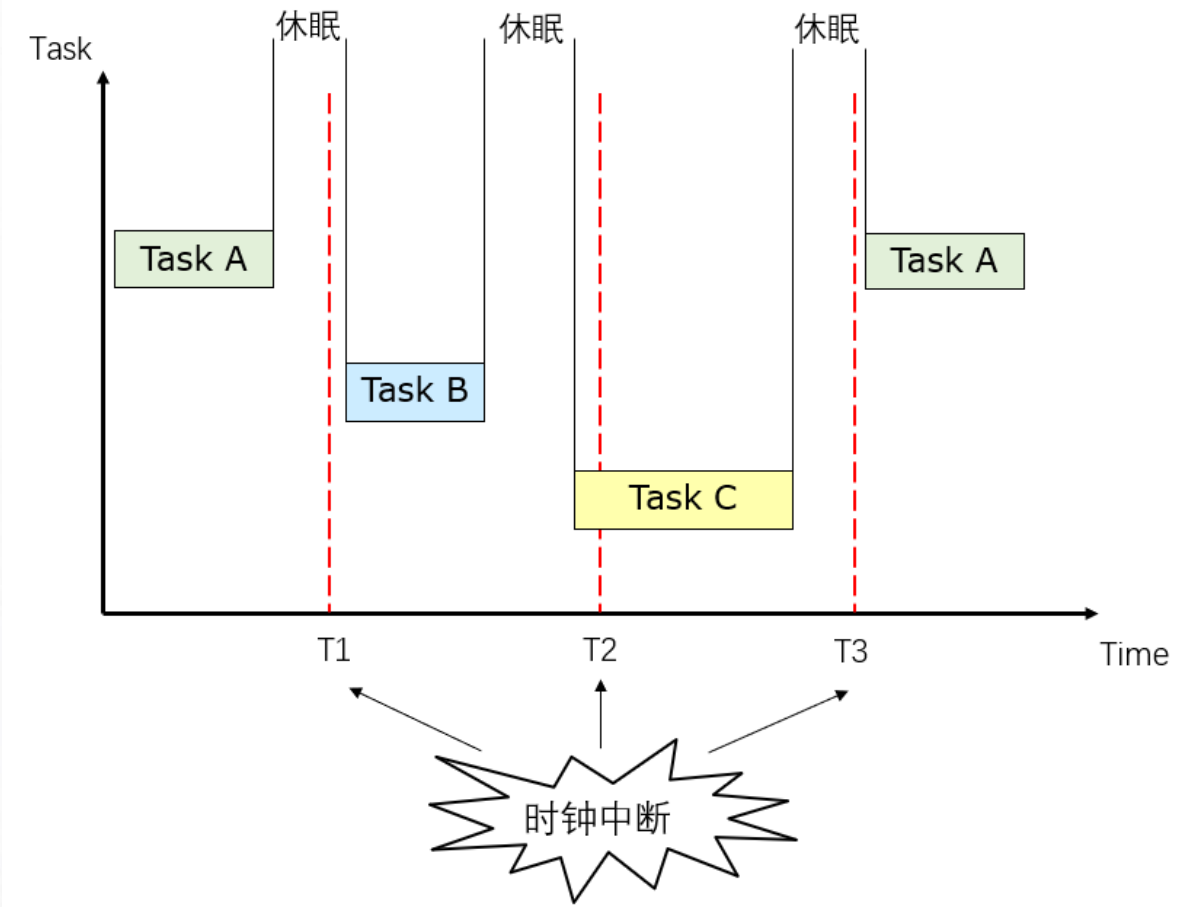


图4-6 Tick机制示意图

Tickless定时机制的原理

Tickless 机制（无节拍机制）Linux2.6.21 内核中引入的新定时机制，它改变了以往Linux 通过周期性时钟中断定时的状况，取而代之的是更精确的动态时钟中断（dynamic ticks）。该机制省去了CPU 空闲时期的无意义的时钟中断，这使得系统开销大幅降低，同时定时的精确性对系统的实时性提高有很大帮助。

Tickless定时机制的原理

1. 通过预期省略不必要的时钟中断

- 由于大部分周期性的时钟中断不会触发进程调度和定时器事件，系统就可根据周期事件来预测对下一次需要执行调度的时刻。
- 如T1、T3的时钟中断是不必要的，由于Task C是周期事件，Task A执行后休眠计时器可提前设置为Task C发生的时间。

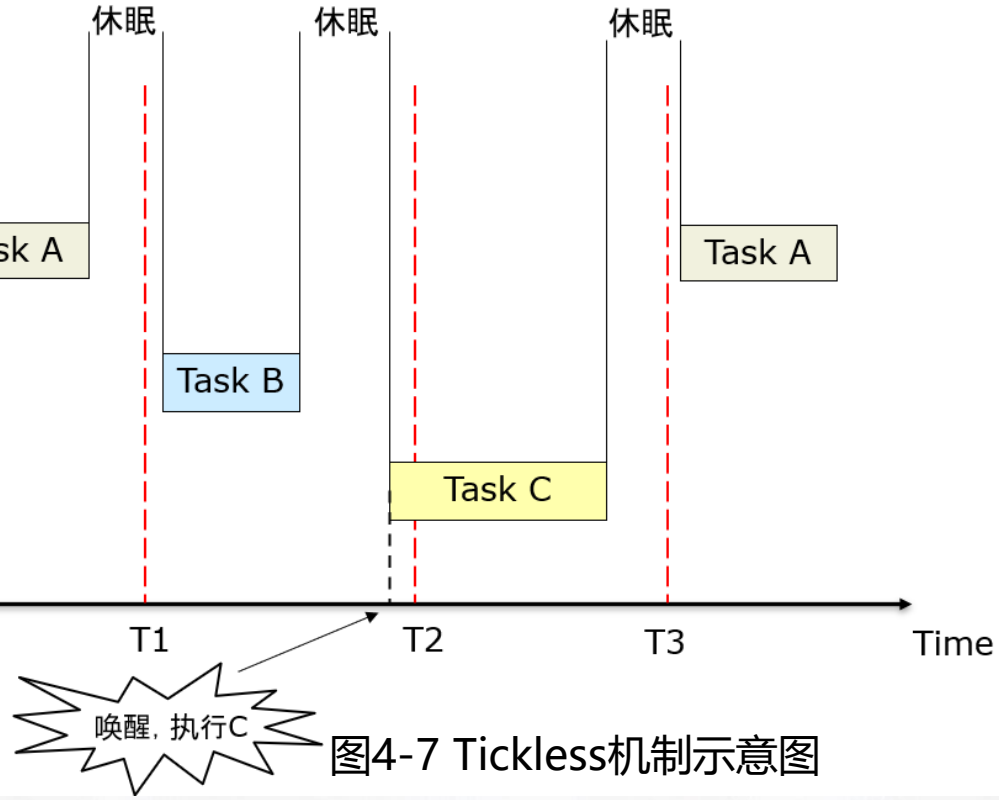


图4-7 Tickless机制示意图

T1,T2,T3时刻的不产生时钟中断，而是通过周期时间的周期时间来预测下一次需要唤醒的时刻。

Tickless： Tickless机制通过计算下一次有意义的时钟中断的时间，来减少不必要的时钟中断，从而降低系统功耗。打开Tickless功能后，系统会在CPU空闲时启动Tickless机制。

Tickless定时机制的原理

2. 将定时器任务与周期定时脱钩，提高定时精度

- 将定时时间单位的粒度缩小，让时钟中断更加逼近其代表的时钟事件的发生时间。

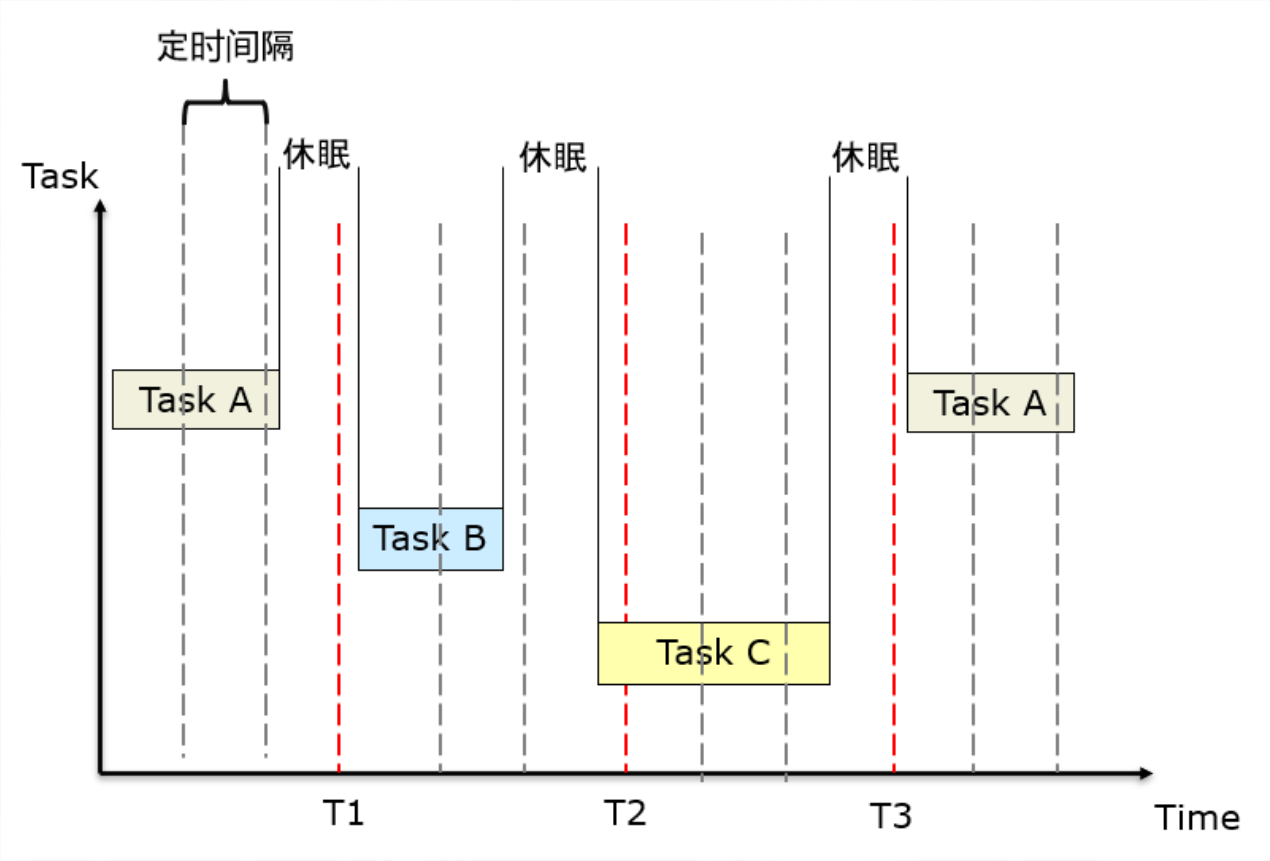


图4-7 Tickless机制示意图

因为时钟中断会唤醒系统带来开销，因此传统的时钟系统无法将tick设置的太小，否则会带来大量开销；但tick太大会导致定时精度较低。
而现在不需要每一个tick都产生中断来唤醒系统，因此可以将定时时间单位的粒度缩小，提高定时精度。

Tickless定时机制的原理

3. 原有周期性时钟中断所携带的时间信息需要恢复

- 虽然周期性的时钟中断已被抑制，但仍可以以tick为单位度量空闲时间段。在实际中断产生后，启动休眠计时器恢复时间信息。

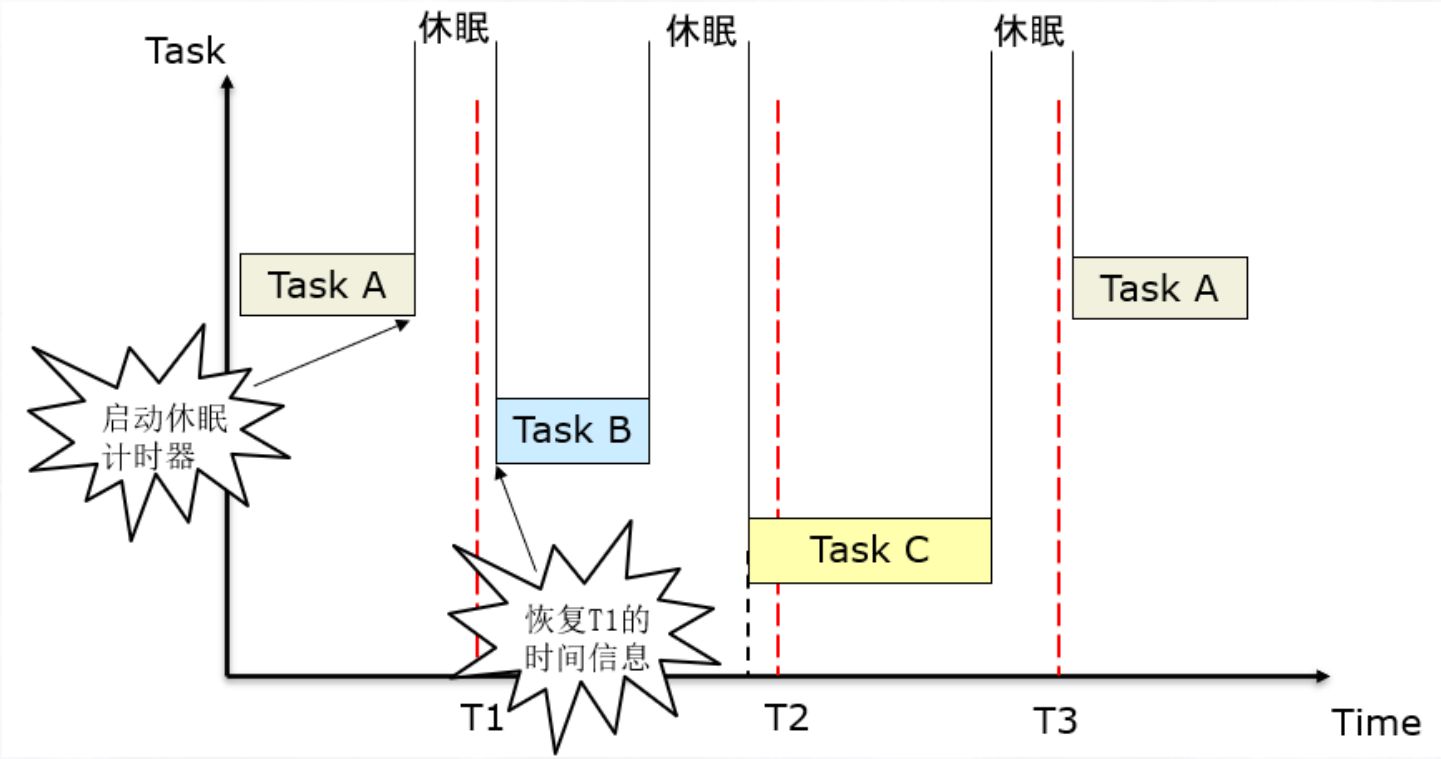


图4-7 Tickless机制示意图

周期性的时钟中断被抑制，而通过预测的方式来唤醒周期事件。但该周期事件被唤醒的时候可能需要使用到时间信息，因此需要对时间信息进行恢复。

方法是在处理器要休眠时启动一个休眠计时器，下一次处理器被唤醒时就能通过休眠计时器来恢复时间信息。

中山大學
SUN YAT-SEN UNIVERSITY

谢谢观看

SUN YAT-SEN UNIVERSITY