第 6 章 多核划分方法

万物并育 《礼记·中庸》

智能嵌入式系统通常是实时系统：在限定时间内完成其任务并提交服务的系统，实时系统包括数字控制、指挥控制、信号处理系统等。

本章主要介绍实时系统的多核划分算法，在多核环境下，对实时系统的任务进行恰当地划分与分配，实现系统的多核划分。

本章主要内容可分成两个部分。第一部分是有关实时系统的实时调度问题，第二部分是介绍基于实时调度的多核划分算法。

第 6.1 节 实时系统

智能手机、飞行器控制系统（如火箭发射、飞机起降控制系统）、高速运动列车的运行控制系统、音乐播放系统、在线转账系统，围棋机器人 AlphaGO 都是典型的实时系统。为了给出实时系统较严格定义，先规范一些基本概念。这些概念大部分来自文献[30]，同时可以参考文献[31]。

文献[30] 将系统可调度和执行的工作单元定义为**工作** Job，而**任务** Task 是相关工作 Job 集合，为某个系统提供功能。文献[31]将任务定义为度量的执行线程(Thread)，并认为任务和进程(process)是大多数内核中具有可调度实体的例子，将任务和进程看作一体。本书将任务定义为系统可调度和可执行工作单位。

6.1.1 基本概念

任务 Job: 系统可调度和可执行的工作单位。

周期任务 period task：一个任务按照顺序周期性地执行。

处理器 Processor：可以在其上执行任务的系统， 如 CPU、FPGA、硬盘、网络等。

任务释放时刻 release time r: 任务时间的一个实例， 任务可以执行的时刻。

任务开始执行时刻 start execution time s： 任务时间的一个实例，任务开始执行的时刻。

任务结束时刻 deadline time d: 任务时间的一个实例， 任务须完成的时刻。

任务执行时间 execution time E: 任务从开始执行到结束的时间长度。

任务响应时间 response time R: 从释放时间到结束时间的时间长度。

执行周期 execution period p: 所有从释放时间到下个释放时间的时间长度中的最小者。

实时系统是由若干任务和处理器等资源组成的可完成某个或几个功能的实体。一般地，根据实时系统对任务结束时间要求是否严格，把实时系统分为硬实时系统和软实时系统。若一个系统，不满足时间约束会导致灾难性结果，这种时间约束称为**硬时间约束**，如火车停车控制系统、炸弹引爆系统、飞机降落系统等系统要求硬时间约束。若轻微的时间约束不满足不会导致严重伤害，这种时间约束称为**软时间约束，**如音乐播放系统、在线转账系统等系统通常会要求软时间约束。

定义 6.1 *具有硬时间约束的系统称为**硬实时系统**。*

例如，火车的制动系统、汽车的制动系统、航空器登陆控制系统等都是硬实时系统。

定义 6.2 *具有软时间约束的实时系统称为**软实时系统**。*

例如，音乐播放系统、电灯开关控制系统、在线转账系统、智能手机通话开关系统、电子游戏等都是软实时系统。

实时系统除了任务外，还有处理器和资源等相关概念:

处理器 P 人们会关注处理器上任务如何调度、不同处理器如何同步、处理器是否有效使用。其属性特征有速度和使用效率。

资源 通常是指被动的资源如寄存器、数据库等，其属性特征是容量。

给定实时系统的处理任务，需要明确开始执行时刻 s、任务执行时间 E、结束时刻 d 和执行周期 p，用 4 元组(s，E，d，p)来表示这些时间参数。一般线下设计并存储好任务的调度方案，供用户使用。调度的执行效率为 E/p，刻画了一个周期内任务执行的情况，通常越大越好，当然最大值为 1，即任务执行时间和执行周期相等。例如：(1，3，6，10)表示任务开始执行时间为 1，执行时间为 3，结束时间为 6，周期为 10，表示 10 个单位时间内任务执行 1 次，执行时间 3 个单位时间，因此执行效率为 3/10=0.3。

6.1.2 任务依赖关系

实时系统除了时间约束，通常任务间会有依赖关系，即一个任务执行依赖于

另一个任务的完成。

定义 6.3 称**任务 J 依赖于任务 I**，是指任务 I 完成后,任务 J 才能开始执行。此时称任务 I 为任务 J 的**前驱**(predecessor)，任务 J 为任务 I 的**后继**(successor)。如果 I 是任务 J 的前驱，并且不存在一个任务 K 使得 I 是任务 K 的前驱同时 K 是任务 J 的前驱，此时任务 I 称为任务 J 的**直接前驱**(immediate predecessor)，而任务 J 称为任务 I 的**直接后继**(immediate successor)。使用符号 I→J 表示 I 为 J 的直接前驱，J 是 I 直接后继的依赖关系。

依赖关系是一种偏序关系。可以用依赖关系图来描述这种任务之间的依赖关系。依赖关系是由任务和任务依赖关系组成的有向无环图。从依赖关系图中可以直接看出任务的依赖关系以及直接前驱和直接后继的关系。

例 6.1 设一系统有 8 个任务，任务之间的依赖关系如图 6-1。

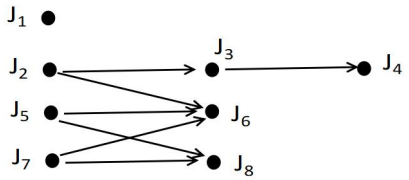

图 6-1 例 6.1 任务依赖关系图[30]

从图 6-1 中可以看到 J_1 是孤立任务，既没有前驱也没有后继。J_2 没有前驱但有后继 J_3、J_4 和 J_6，而 J_3 和 J_6 是其直接后继。J_8 有两个直接前驱 J_5 和 J_7，但没有后继。任务 J_3 是 J_2 的直接后继，J_4 是 J_2 的后继。J_6 是 J_2、J_5 和 J_7 的直接后继。

为了合理调度一个系统的任务，需要知道任务的执行时间。任务的执行是依赖于执行环境和资源的。因此，同一个任务的执行时间会有差异，一般会有最小执行时间、最大执行时间和平均执行时间。为了保证任务调度，一般把任务的最大执行时间规定为任务执行时间。通常把任务执行时间标注到依赖关系图中。

例 6.2 在例 6.1 基础上我们标注每个任务的执行时间，如图 6-2。

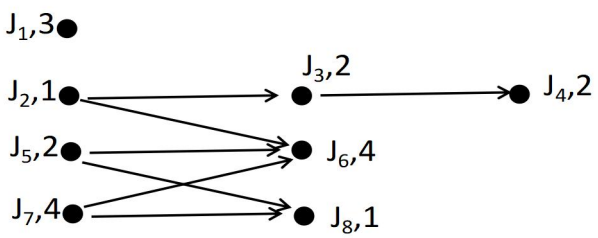

图 6-2 例 6.2 任务依赖关系图

任务后面的数字是指任务执行时间。如，任务 J_1 的执行时间为 3，而任务 J_6 的执行时间为 4。

6.1.3 任务抢占

实时系统中的任务常常是依次轮流执行的。如果一个正在执行中的任务被临时中断执行（被挂起），把处理器让给另一个更急迫的任务，等该紧迫任务完成后，处理器继续执行被挂起的任务，这种任务的中断称为**抢占**(preemption)。如果一个任务在其执行的任何时候都可以被抢占，则称之为**可抢占的**(preemptable)。若任务在执行过程中在任何时候是不可以被别的任务抢占，则称这个任务是**不可抢占的**(nonpreemptable)。

6.1.4 实时系统参考架构

实时系统参考架构是由任务依赖关系图、处理器关系图和资源架构图组成，见图 6-3。处理器关系图中关系表示了处理器间数据传输关系，资源架构图中关系表示了资源（如数据）间的逻辑关系。

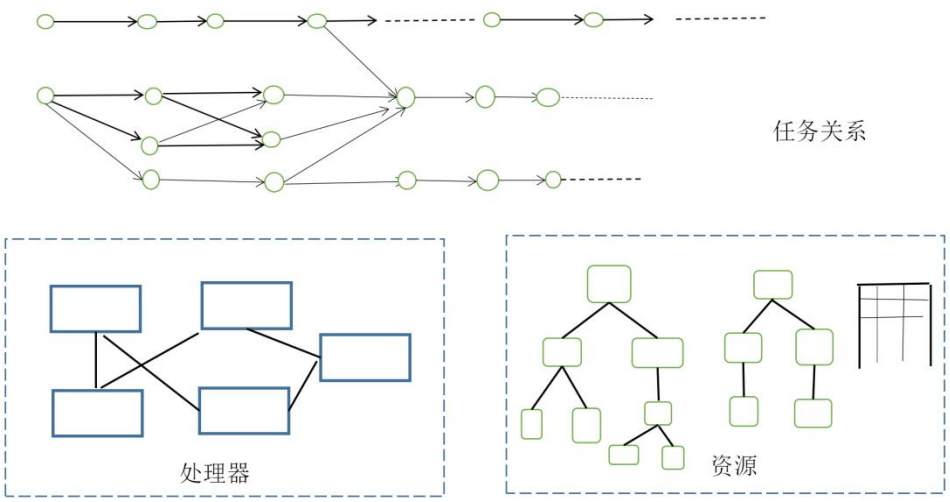

图 6-3 实时系统参考架构图

第 6.2 节 任务优先级

任务优先级是在任务依赖关系基础上，定义任务优先执行的级别，优先级别越高越优先执行。因而，在任务进行多核划分之前需要先构建好任务优先级表。

使用符号>表示任务优先级：I>J，表示任务 I 比任务 J 优先级高。

6.2.1 任务优先级值

任务优先级值是将一个数值（通常是正整数）指派给任务表示任务的优先级值，数值越大优先级别越高。

一个简单的任务优先级值定义是将任务执行时间指派给任务的优先级值。如**例 6.2** 中任务 J_1，…，J_8 的优先级值分别为 3、1、2、2、2、4、4、1。

下一段介绍一个稍微复杂的任务优先级值定义公式。

依据任务依赖关系图，按照节点的执行时间长短以及出度计算节点的优先级值，其结果是任务（节点）执行时间越长优先级越高，节点出度越大优先级越高。具体来说，优先级值按公式 6-1 计算。

设有 n 个节点 J_1，J_2，…，J_n，且每个节点 J_i 带权值 E_i(i=1…n)组成的有向无环图 G，符号 E 表示图 G 的边集合，$(J_i，J_j)\in E$ 表示节点 J_i 到 J_j 的有向边，J_i 称为此边的始端点，J_j 称为此边的末端点。即 J_i 是 J_j 的直接前驱，而 J_j 是 J_i 的直接后继。

对每个节点 J_i 使用符号 $Pre(J_i)$和 $Suc(J_i)$分别表示 J_i 的直接前驱节点集合和直接后继节点集合， 即：$Pre(J_i)=\{J_j \mid (J_j，J_i)\in E\}$，$Suc(J_i)=\{J_k \mid (J_i，J_k)\in E\}$。

对每个节点 J_i 定义其入度 $I(J_i)$为以 J_i 为末端点的边的个数，以及其出度 $O(J_i)$为以 J_i 为始端点的边的个数，即：$I(J_i)=|Pre(J_i)|$，$O(J_i)=|Pre(J_i)|$。

节点 J_i 的优先级值按照公式 6-1 定义：

$$OPri(J_i)=E_i+O(J_i)+Max\{OPri(J_k) \mid J_k\in Suc(J_i)\} \qquad \text{公式 6-1}$$

节点 J_i 的优先级值等于 J_i 的执行时间 E_i、出度 $O(J_i)$和直接后继节点优先级值最大者之和，体现了执行时间、直接后继的重要性，直接后继越多越需早点执行。

很明显，若 $O(J_i)=0$，即没有直接后继，则 $OPri(J_i)=E_i$。

推论： 若节点 J_i 是节点 J_j 的直接前驱，则 $OPri(J_i) > Pri(J_j)$。

公式 6-1 是依据图节点进行回溯给出，即首先给出出度为 0 的节点优先级值，然后沿着有向边的方向进行回溯计算节点的优先级值。

例 6.3 图 6-4 是一个含有 8 个任务的任务依赖关系图，边表示依赖约束，每个任务名后面的数字是它的执行时间。

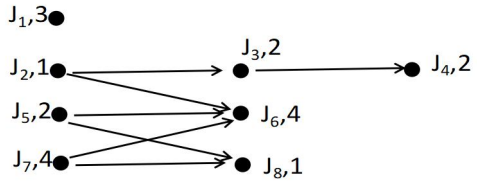

图 6-4 例 6.3 任务依赖关系图

解：按照公式 6-1 计算例 6.3 中 8 个节点的优先级值分别为：$OPri(J_1)=3$，$OPri(J_4)=2$，$OPri(J_6)=4$，$OPri(J_8)=1$，$OPri(J_3)=5$，$OPri(J_2)=8$，$OPri(J_5)=8$，$OPri(J_7)=10$。

6.2.2 任务优先级表

按照任务优先级值，用下面的算法构建出任务调度优先级表：按照优先级值、任务依赖关系、任务释放时间以及执行时间进行排序，构成任务优先级静态表。

算法 6-1 任务优先级排序算法 TaPSA

Input: 含有 n 个任务 J1，…，Jn 的有向无环图 G、G 的节点依赖关系矩阵、任务释放时间表、任务执行时间表、任务优先级值表

Output: 任务优先级表

第 1 步：按照优先级值从高到低进行排序，若没有优先级值相等，则排序过程结束，输出排序表转入到第 3 步，否则转入第 2 步。

第 2 步：将优先级值相等的任务进行组合，形成若干个子表。在每个子表中，先按照释放时间先后顺序进行排序：释放时间早优先级高；若释放时间一致，则将按照执行时间从大到小进行排序；若执行时间也相等则按照任务编号从小到大排序。将排序后各个子表代替第 1 步按照优先级值排序表中相应子表，排序过程结束，输出排序表并转入到第 3 步。

第 3 步：算法结束。

例 6.4 图 6-4 是一个含有 8 个任务的任务依赖关系图，边表示依赖约束，每个任务名后面的数字是它的执行时间。J_5 在时刻 4 释放，其它任务释放时间都是 0。按照任务优先级排序算法 TaPSA 求出节点的优先级表。

解： 按照公式 6.1 计算出的优先级值，得到优先级表为 $J_7>J_2>J_5>J_3>J_6>J_1>J_4>J_8$。需要注意的是：此例中 J_2 与 J_5 的优先级值相等（=8），由于 J_2 的释放时间为 0，而 J_5 的释放时间为 4，因而 J_2 的优先级高于 J_5 的优先级。

注：依据任务执行时间定义的优先级值得到任务优先级表可以有两种方式：

(1) **最长执行时间：**执行时间越长优先级值越大，同时直接前驱优先级高于直接后继，释放时间早任务的优先级高于释放时间晚的任务，在这三个都一样的情况下，任务编号小的优先级高。依据这个优先级排序要求，得到例 6.4 任务优先级表为 $J_7>J_6>J_1>J_3>J_4>J_5>J_2>J_8$，其中 J_6 和 J_7 优先级值相等（=4），释放时间都是时刻 0，但 J_7 是 J_6 的直接前驱，因而 J_7 优先级别高于 J_6。任务 J_2 与 J_8 的优先级关系是因为任务 J_2 编号小于任务 J_8 的编号，而 J_3、J_4、J_5 任务优先级相等（=2），J_3 与 J_4 的释放时间都是时刻 0，而 J_5 的释放时间是时刻 4，因而 J_3 与 J_4 优先级高于 J_5，J_3 高于 J_4 是因为 J_3 是 J_4 的直接前驱。

(2) **最短执行时间：**执行时间越短优先级别越大，其他情况相同于最长执行时间，其优先级表为 $J_2>J_8>J_3>J_4>J_5>J_1>J_7>J_6$。

第 6.3 节 实时调度

一个实时系统通常会有多个任务，需要把它们安排给处理器去执行。这个安排过程通常称为调度。

6.3.1 实时调度问题

一个实时调度问题是：依据任务的依赖关系图，将任务合理地调度（划分）到多个处理器（包括 CPU,、GPU、ARM 和 FPGA），使得全体任务都完成且所需要的时间最少，即完工时间最小。具体而言，可以定义如下：

实时调度问题

已知：一个实时系统含有 n 个任务 J_1，J_2，...，J_n，每个任务 J_i 都关联执行时间 E_i，i=1...n，由 n 个节点带权的有向无环图表示这个 n 个任务间的依赖关系。每个任务 J_j 定义一个开始时间 s_j 和结束时间 d_j(j=1...n)，任务 J_j 占用时间段为 s_j 到 d_j。

问题：将这个 n 个任务分配到 k 个处理器 P1，...，Pk 使得 d_j 最大者最小，即 Min $Max_{j=1...n}$ $\{d_j\}$.

实时调度问题的有效算法需要满足如下几条实时调度/划分原则：

1、每个处理器在任何时刻最多指定一个任务；

2、每个任务在任何时刻最多指定给一个处理器；

3、任务在释放时间之前是不可以调度的；

4、所有依赖关系和资源使用的约束都要满足，只有一个任务的前驱都执行

后，该任务才能执行；

5、若允许抢占则优先级高的任务可以抢占优先级低的任务。

6.3.2 系统完工时间与处理器使用率

在设计调度算法时，处理器使用率也是需要考虑的一个重要指标。**处理器完工时间**是指处理器最后一个任务完成的时刻。 如图 6-5 处理器 1 的完工时间为 13， 而处理器 2 的完工时间为 10。**处理器使用率**是指一个处理器完成所有任务执行时间之和与处理器完工时间之比。如图 6-5 处理器 1 的使用率为 9/13，处理器 2 的使用率为 10/10。

当在多处理器协同工作完成一个系统时，**系统完工时间**规定为所有处理器完工时间的最大者，而一**处理器使用率**定义为该处理器实际使用时间（在完成任务中所有任务的执行时间之和）与系统完工时间之比。例如：图 6-5 的系统完工时间为 13，处理器 P1 使用率为 9/13，处理器 P2 的使用率为 10/13。

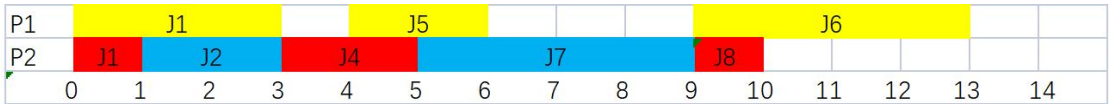

图 6-5 处理器使用率

6.3.3 优先级驱动算法

常用的实时调度算法有：时间驱动（time-driven）调度算法、优先级驱动（priority-driven）调度算法。

顾名思义，**时间驱动方法**是时间驱动的调度方法，一般要明确任务执行时间和开始执行时刻。线下设计并存储好任务的调度方案，供用户使用。

本节接下来重点介绍优先级驱动算法。任务按照优先级顺序进行排队，在调度的任何时刻，具有最大优先级的任务优先安排到处理器上执行。比较常见的优先级别策略有：

- 最短执行时间优先 SETF(Shortest-Executive-Time-First)和最长执行时间优先 LETF(Longest-Executive-Time-First)：这两种策略将执行时间作为（唯一）基础来确定优先级。
- 最早结束时间优先 EDF（Earliest-Deadline-First）：这种算法适用于单个处理器且允许抢占的调度。

优先级驱动方法是指一大类绝不故意让资源空闲的实时调度算法，只有当没有任务需要资源准备时资源才空闲。优先级驱动方法采用贪心策略，试图寻找局部最优的决定。当资源空闲，某个任务准备使用资源并且开始处理时，这个算法绝不会让这个任务等下去。优先级驱动方法通常会使用一张表表示任务优先级别，因此有时也称其为表调度（List Scheduling）算法。 任务可以按照优先级顺序进行排队，在调度的任何时刻，具有最大优先级的任务优先安排到处理器上执行。优先级表以及其他规则，如何时允许抢占，完全定义了这个调度算法。

第 6.4 节 多核划分算法

本节介绍基于实时调度的多处理器系统划分算法 **MuPPA**，依据任务优先执行列表进行划分，将任务分配给空闲的处理器。在实时划分时除了依据优先级列表，还需注意到一个任务可以执行的前提是它的所有前驱任务都已经执行完毕。

算法 6-2 多核系统划分算法 MuPPA

Input: 含有 n 个任务 J1，...，Jn 的有向无环图 G、G 的节点依赖关系矩阵、任务释放时间表、任务执行时间表、k 个处理器 m1,...,mk
Output: n 个任务在 k 个处理器划分方案、k 个处理器最大执行时间、每个处理器使用率

第 1 步：使用公式 6-1 计算这 n 个任务的优先级值表；

第 2 步：使用任务优先级排序算法 TaPSA 构建这 n 个任务优先级表；

第 3 步：按照任务优先级表级别高低检查其所有前驱任务是否已执行完毕，将所有前驱执行完毕且优先级别最高且释放时间已到的任务分配到空闲的处理器，同时将这个任务从优先级表中删除；
第 4 步：如果任务优先级表为空则算法结束，否则转到第 3 步。

定理 6.4 将 n 个任务划分到 k 个处理器的多处理器划分算法 MuPPA 时间复杂度是 $O(n^2k)$。

证明：该算法计算优先级值时，所用时间为 $O(n^2)$。确定优先级，实际上是按照优先级值进行排序，因此所用时间为 $O(n^2)$。划分所用时间比较难以确定，因为可能需要对 O(n)数量级别的任务进行来回确定，是否其所有的直接前驱都已经完成就绪。为了减少这个确认遍数，可以每当有一个新的任务完成时，去进行下一步划分分配。采用这种策略，则时间复杂度为 $O(n^2k)$，其中 k 为处理器数量，

因为每一次分配一个任务，都要考察哪台处理器空闲。综上，整个算法的时间复杂度是 $O(n^2k)$。

例 6.5（例 6.4 继续）：图 6-4 是一个任务依赖图，边表示依赖约束，每个任务名后面的数字是它的执行时间。J_5 在时刻 4 释放，其它任务释放时间都是 0。 调度这些任务在两个处理器 P1 和 P2 上执行，并计算每个处理器的使用率。

解：依据公式 6-1 计算的任务优先级，并由任务优先级排序算法 TaPSA 得到任务优先级表：$J_7>J_2>J_5>J_3>J_6>J_1>J_4>J_8$。

使用任务划分算法 MuPPA，得到如下划分过程：

时刻 0：J_7 放在 P1 中，把 J_2 放在 P2 中，更新后任务优先级表：$J_5>J_3>J_6>J_1>J_4>J_8$；

时刻 1：J_2 执行完毕，P2 空闲。由于 J_5 还没有释放，因此 J_5 不能调度，把 J_3 放在 P2 中，更新后任务优先级表：$J_5>J_6>J_1>J_4>J_8$；

时刻 2：两个处理器都不空闲， 因此不能安排新的任务进来；

时刻 3：J_3 执行完毕，P2 空闲，应该安排 J_6 但 J_6 依赖于 J_5，因此不能安排 J_6，把 J_1 放在 P2 中，更新后任务优先级表：$J_5>J_6>J_4>J_8$；

时刻 4：J_7 实现完毕，P1 可以安排任务，此时 J_5 释放，因此把 J_5 放在 P1 中，更新后任务优先级表：$J_6>J_4>J_8$；

时刻 5：J_1 和 J_5 都没有执行完毕，不能安排新任务进来；

时刻 6：J_1 和 J_5 都执行完毕，安排 J_6 到 P1，J_4 到 P2，更新后任务优先级表：J_8；

时刻 7：J_4 和 J_6 都没有完成，不能安排新的任务；

时刻 8：J_4 完成，安排 J_8 在 P2 中执行，更新后任务优先级表为空表，调度完成。

时刻 9：J_8 完成， J_6 继续执行；

时刻 10：J_6 完成，整个系统执行完毕。

任务在处理器上划分结果：

P1 处理器： J_7，J_5，J_6；

P2 处理器： J_2，J_3，J_1，J_4，J_8。

本题划分结果见图 6-6。

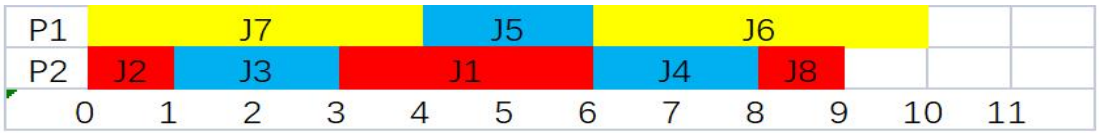

图 6-6 例 6.5 划分结果图（公式 6-1 计算优先级值、最长执行时间为优先级值）

整个任务完工时间是 10，即 J_6 完成时间。处理器 P1 使用率=1，P2 使用率=0.9.

若使用最长执行时间优先级表进行划分，得到的划分结果同 6-6 一致。若使用最短执行时间优先级表进行划分，得到的划分结果如图 6-7：

图 6-7 例 6.5 划分结果图（最短执行时间为优先级值）

例 6.6（本例子来自文献[32]）

将图 6-7 中 11 个任务划分到 3 个处理器 P1，P2 和 P3，并求出系统完工时间和每个处理器使用率，其中所有任务释放时间均为 0，圆内数字为任务编号，圆外数字为执行时间。

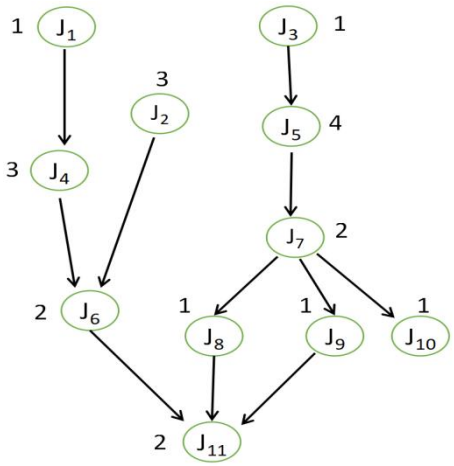

图 6-8 例 6.6 任务依赖关系图

解：使用优先级值计算公式 6-1 计算每个节点的优先级值，任务优先级排序算法 TaPSA 得到任务优先级表，使用任务调度算法 MuPPA，完成任务多核划分。

第一步：使用优先级值计算公式 6-1 计算节点优先级值

$OPri(J_{11})=2$， $OPri(J_{10})=1$， $OPri(J_9)=4$， $OPri(J_8)=4$， $OPri(J_7)=9$， $OPri(J_6)=5$，$OPri(J_5)=14$，$OPri(J_4)=9$，$OPri(J_3)=16$，$OPri(J_2)=9$，$OPri(J_1)=11$。

第二步：启用任务优先排序算法 TaPSA 得到优先关系表

使用任务优先排序算法 TaPSA 进行排序，得到任务优先级表：$J_3>J_5>J_1>J_2>J_4>J_7>J_6>J_8>J_9>J_{11}>J_{10}$。优先值等于 9 的节点有 J_2，J_4 与 J_7，优先级值等于 4 的节点有 J_8 与 J_9。释放时间均为 0，因而需要按执行时间大小排序，J_2 与 J_4 的执行时间等于 3 大于 J_7 的执行时间(=2)，J_8 与 J_9 的执行时间都是 1。再按编号小级别高原则则有：J_2 优先级高于 J_4，J_4 优先级高于 J_7，J_8 优先级高于 J_9。

第三步：在 3 个处理器 P1,P2 和 P3 上使用划分算法 MuPPA 进行划分：

时刻 0: 由于 J_5 与 J_3 有依赖关系，因此 J_5 暂时还不能安排，将 J_3, J_1 和 J_2 分别放到处理器 P1，P2 和 P3。此时，任务优先级表为 $J_5>J_4>J_7>J_6>J_8>J_9>J_{11}>J_{10}$。

时刻 1: J_3 与 J_1 都完工，安排 J_5 在 P1 中执行，J_4 在 P2 中，J_2 在 P3 中继续执行。此时，任务优先级表为：$J_7>J_6>J_8>J_9>J_{11}>J_{10}$。

时刻 2: 3 个处理器都不空闲。

时刻 3：J_2 完工，J_4 和 J_5 都没完工，处理器 P3 空闲， P1 和 P2 都不空闲。J_7 依赖于 J_5，J_6 依赖于 J_4，后面的 J_8，J_9，J_{11} 和 J_{12} 前驱都没有完成，因此不能进行安排。此时 P3 空闲，优先级表为：$J_7>J_6>J_8>J_9>J_{11}>J_{10}$。

时刻 4：J_4 完工，J_5 在 P1 中继续执行，P2 和 P3 处理器空闲。J_7 在等待 J_5 完工，因此把 J_6 安排在 P3 中，其余任务的前驱都没有完成，P2 仍空闲。优先级表为：$J_7>J_8>J_9>J_{11}>J_{10}$。

时刻 5：J_5 完工，J_7 安排在 P1 中。P2 空闲，因为 J_8 到 J_{11} 都依赖于 J_7 完工。任务优先级表为：$J_8>J_9>J_{11}>J_{10}$。

时刻 6：J_6 完工，J_7 在 P1 中继续执行，因此 P3 和 P2 空闲。优先级表为：$J_8>J_9>J_{11}>J_{10}$。

时刻 7：J_7 完工，3 个处理器都空闲，安排 J_8，J_9，J_{10} 分别到 P2，P3，P1 中，因为 J_{11} 依赖于 J_8 和 J_9，任务优先级表为：J_{11}。

时刻 8：J_7，J_8，J_9 都完工了，把 J_{11} 安排在 P2（因而 P2 空闲时间较长），P1 和 P3 空闲，优先级表为空表。

时刻 9：J_{11} 在 P2 中继续执行，P1 和 P3 空闲。

时刻 10： J_{11} 完工。整个系统执行完毕。

系统完工时间为任务 J_{11} 的结束完工时间 10。

处理器划分结果：

P1 处理器：J_3，J_5，J_7，J_{10}，使用率=0.8;

P2 处理器：J_1，J_4，J_8，J_{11}，使用率=0.7;

P3 处理器：J_2，J_6，J_9，使用率=0.6。

本题划分结果见图 6-9。

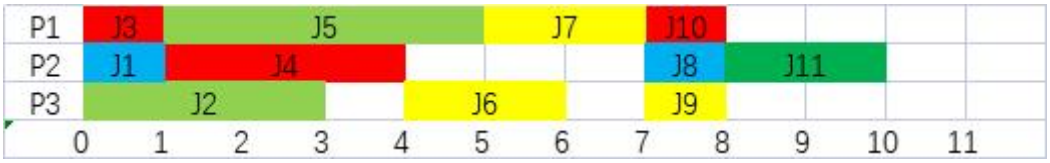

图 6-9 例 6.6 划分结果图

第 6.5 节 带抢占的多核划分

本节考虑允许抢占的多核划分方法。

规定：1. 优先级别高的任务总可以抢占优先级别低的任务，除非规定该优先级别低的任务不可以被抢占。

2. 每个任务至多被抢占一次。

3. 被抢占任务回到优先级表等待再次安排划分，此时该任务执行时间是剩下待执行时间，即该任务原执行时间减去已执行的时间所得到差。

4. 被抢占任务回到优先级表有两种返回方式：

(1) 回到优先级表原来的位置，

(2) 回到优先级表的表头，即优先级表中级别最高的待分配任务。

5. 被抢占任务再次划分时处理器安排有两种可能：

(1) 只安排在上次安排的处理器，

(2) 随意安排给任何处理器。

例 6.7 图 6-4 是一个任务依赖关系图，边表示优先约束，每个任务名后面的数字是它的执行时间。J_2 在时刻 1 释放，J_5 在时刻 5 释放，其它任务释放时间都是 0。

划分这些任务在两个处理器 P1 和 P2 上执行。规定优先级高的任务可以抢占优先级低的任务先执行，只允许被抢占一次。。

解：分两种情况进行划分： 情况 1. 被抢占任务回到优先级表原来位置(4(1))以及随意安排给任何处理器(5(2))；情况 2. 被抢占任务回到优先级表原来位置(4(1))以及只安排给上次安排的处理器(5(1))。其他情况作为练习。

情况 1. 被抢占任务回到优先级表原来位置(4(1))以及随意安排给任何处理器(5(2))。

由任务优先级排序算法 TaPSA 得到任务优先级表：$J_7>J_2>J_5>J_3>J_6>J_1>J_4>J_8$。

使用划分算法 MuPPA 调度，加上优先级别高的任务可以抢占优先级别低的任务。

时刻 0：任务 J_7 释放，J_2 与 J_5 都没有释放，J_1 释放。J_7 安排给 P1，J_1 安排给 P2，优先级表：$J_2>J_5>J_3>J_6>J_4>J_8$；

时刻 1：J_2 释放，J_2 优先级高于 J_1，因而抢占 J_1 先执行，安排 J_2 到 P_2，优先级表：$J_5>J_3>J_6>\mathbf{J_1}>J_4>J_8$；($J_1$ 回到优先级表中原来位置，但执行时间为 2)；

时刻 2：J_2 执行完毕，J_5 还没有释放，安排 J_3 到处理器 P2，优先级表：$J_5>J_6>\mathbf{J_1}>J_4>J_8$；

时刻 3：J_3 和 J_7 都没有完成，处理器都不空闲，优先级表：$J_5>J_6>\mathbf{J_1}>J_4>J_8$；

时刻 4：J_3 和 J_7 都完成，处理器都空闲， J_5 还没有释放，J_6 依赖 J_5 完成，因而不能安排，安排 J_1 到处理器 P2(因为 J_1 是可以安排给任何处理器 5(2))，J_4 安排到处理器 P1，优先级表：$J_5>J_6>J_8$； 。

时刻 5：J_5 释放，优先级高于正在执行的两个任务 J_1 和 J_4。由于 J_1 已被抢占过一次，因而这次安排 J_5 抢占 J_4，将 J_5 安排到 P1 处理器，同时将 J_4 放回到优先级表的原来位置，优先级表：$J_6>J_4>J_8$； 。

时刻 6：J_1 完成，J_6 等待 J_5 完成， J_8 等待 J_5 完成，因而安排 J_4 到处理器 P2，优先级表：$J_6>J_8$；

时刻 7： J_5 完成了， J_6 和 J_8 都可以调度执行了，J_6 安排到 P1 处理器，J_8 安排给 P2，优先级表为空表，任务划分结束；

时刻 8：J_8 完成，J_6 在 P1 中继续执行；

时刻 9：J_6 在 P1 中继续执行；

时刻 10：J_6 在 P1 中继续执行；

时刻 11：J_6 执行完毕，整个系统执行完毕。

系统完工时间为 11，

划分结果：处理器 P1：J_4，J_5，J_6，J_7，使用效率 1(=11/11)；处理器 P2：J_1，J_2，J_3，J_4，J_8，使用效率 0.72(=8/11)。

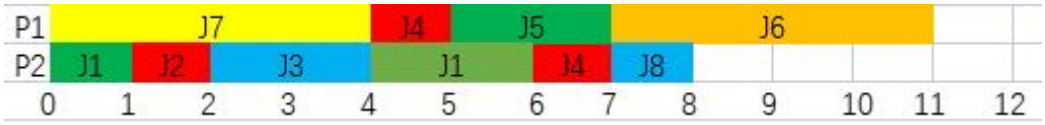

图 6-10 例 6-7 情况 1 划分结果图

情况 2. 被抢占任务回到优先级表原来位置(4(1))以及只安排给上次安排的处理器(5(1))。

情况 2 划分从时刻 6 开始与情况 1 不同。

时刻 6：J_1 完成，J_6 等待 J_5 完成， J_8 等待 J_5 完成，由于 J_4 上次安排给处理器 P1，因而不能安排到处理器 P2，这样处理器 P2 空闲，优先级表：J_6>J_4>J_8；

时刻 7： J_5 完成了， J_6 和 J_4 都可以调度执行了，J_4 安排到 P1 处理器，J_6 安排给 P2，优先级表: J_8；

时刻 8：J_4 完成，J_8 安排给处理器 P1，J_6 在 P2 中继续执行；

时刻 9：J_8 完工，J_6 在 P2 中继续执行；

时刻 10：J_6 在 P2 中继续执行；

时刻 11：J_6 执行完毕，整个系统执行完毕。

系统完工时间为 11，

划分结果：处理器 P1：J_4，J_5，J_7，J_8，使用效率 0.82(=9/11)；处理器 P2：J_1，J_2，J_3，J_6，使用效率 0.91(=10/11)。

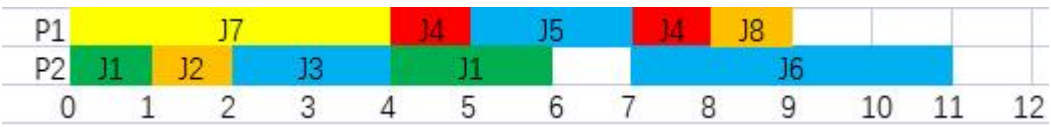

图 6-11 例 6-7 情况 2 划分结果图

第 6.6 节 本章小结

多核划分/调度问题一般情况下是 NP-完全的， 在一些约束条件下也是 NP-完全的[33]。本章介绍的计算任务优先值公式 6-1 是本书提出的，综合考虑到了

任务执行时间与任务出度，反映了这两个方面的重要性。当然只考虑其中一个方面也是可以的，只是全面性不够。文献[34]介绍了多个单核调度算法，包括最早结束时间（Earlist-Deadline-First， EDF）、最早到期（Earlist-Due-First，EDD）和最迟截止期（Lastest-Deadline-First，LDF），该文献也介绍了多处理器调度方法，包括同构多处理器和异构多处理器两种情形。有关单核调度的有效性研究成果可参看文献[35]。

习题

6.1 编程实现任务优先级排序算法 TaPSA，并求出下列两题的任务优先级表

(1) 设一系统有 5 个任务， 任务执行时间与依赖关系如图 6-12，其中任务 T_2 释放时间是 3，其余释放时间都是 0。

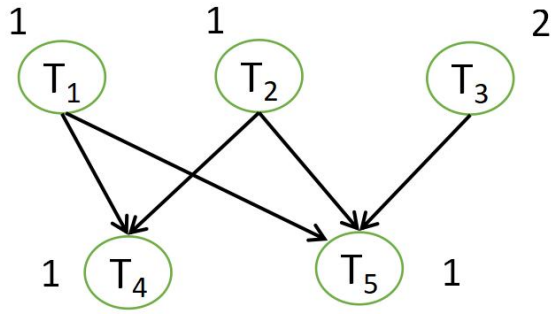

图 6-12 习题 6.1（1）任务依赖关系图

(2) 设一系统含有 11 个任务，任务执行时间和依赖关系如图 6-13，其中 J_4 的释放时间为 4，J_8 的释放时间为 6，其余任务释放时间均为 0。

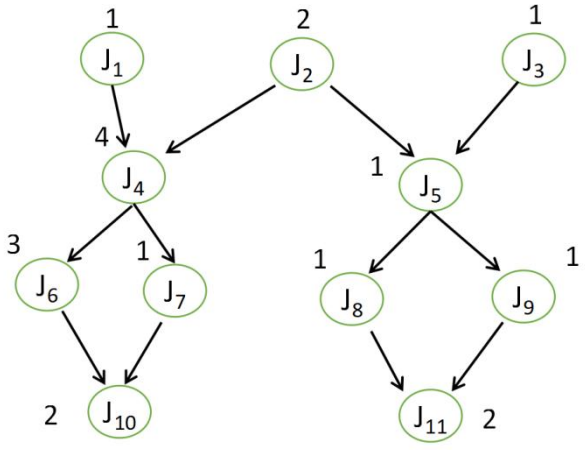

图 6-13 习题 6.1（2）任务依赖关系图

6.2 例 6.5 按照最长执行时间和最短执行时间优先级表进行调度，给出调度过程，

并与例 6.5 调度过程进行比较，指出他们的差异。

6.3 编程实现多核划分算法 MuPPA，并给出下面两题的划分结果，并计算每个处理器的使用率。

(1). 图 6-12 含有 5 个任务，释放时间都是 0，安排在两个处理器 P1 和 P2 上执行。

(2) 将图 6-13 中 11 个任务调度到 3 个处理器 P1，P2 和 P3，其中 J_4 的释放时间为 4，J_8 的释放时间为 6，其余任务释放时间均为 0。

6.4 将图 6-14 中 13 个任务调度到 2 个处理器 P1 和 P2，其中 J_7 的释放时间为 10，J_9 的释放时间为 11，J_{11} 的释放时间为 23，其余任务释放时间均为 0。调度时允许抢占， 抢占时考虑分四种情况进行调度：

情况 1. 被抢占任务回到优先级表原来位置(4(1))以及随意安排给任何处理器(5(2))；

情况 2. 被抢占任务回到优先级表原来位置(4(1))以及只安排给上次安排的处理器(5(1))；

情况 3. 被抢占任务回到优先级表的表头 (4(2))以及随意安排给任何处理器(5(2))；

情况 4. 被抢占任务回到优先级表的表头(4(2))以及只安排给上次安排的处理器(5(1))。

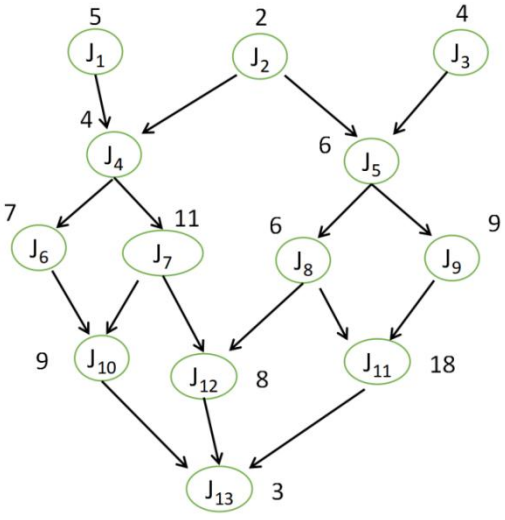

图 6-14 习题 6.4 任务依赖关系图