

**《实时计算系统设计》**

**课程报告**

|  |  |
| --- | --- |
| 报告内容名称: | 第二次实时计算系统作业 |
| 院系: | 信息科学与工程学院 |
| 专业班级: | 计科1502班 |
| 学号: | 201507010206 |
| 姓名: | 黄郭斌 |
| 课程指导教师: | 安吉尧 教授 |

实验提交时间：2018-11-11

1. 实训实验环境

无

1. 实训实验内容与完成情况

## 还有哪些实时调度算法？

### 1) 单处理器实时调度

#### RMS算法

(1) 任务T i (P i, Ci, D i) 模型: 周期为P i,计算时间为Ci, 时限D i 为周期终点。任务在周期起点释放, 高优先级任务可抢占低优先级任务的执行。

(2) 优先级分配方法: 静态固定分配。优先级与周期成反比, 周期越短优先级越高。

(3) 可调度性分析: 如果任务集满足下式, 则该任务集可调度。

定理1：n个独立的周期任务可以被RMPA调度，如果U<=n(2^(1/n)-1)。

一个任务的响应时间(response time)是指一个任务请求到这个任务实际完成的时间跨度. 在静态调度中, 任务的临界时刻(critical instant)这个概念被首先提出来. 它被定义为一个特定的时刻, 如果在这个时刻任务请求到来, 那么会导致这个任务的响应时间最大（A critical instant of a task is the time atwhich the release of a task will produce the largestresponse time）. 由此得出

定理1: 一个任务的临界时刻就是比这个任务优先级高的所有任务同时发出请求的时刻.

（Lemma: For any task, the critical instant occurs if thattask is simultaneously released with all higher prioritytasks .）

证明: 由于一个任务的响应时间是它自己的负载时间加上被其它优先级高的任务所打断的时间. 由于自己的负载时间是固定的, 我们考虑在什么时候任一高优先级的任务会有最长的打断时间. 显然, 只有当这一高优先级的任务与该任务同时请求处理时, 才能可能产生最大的打断时间.

如果有任务1和任务2，且任务1的优先级比任务2高，那么任务2的响应时间会被任务1延迟。

[](https://baike.baidu.com/pic/%E5%AE%9E%E6%97%B6%E8%B0%83%E5%BA%A6/3344983/0/8718367adab44aed69544e48b01c8701a18bfb00?fr=lemma&ct=single)

当任务1的请求到来的更早，那么任务2的响应时间就更长了。

[](https://baike.baidu.com/pic/%E5%AE%9E%E6%97%B6%E8%B0%83%E5%BA%A6/3344983/0/dcc451da81cb39dba76694fad3160924aa1830f3?fr=lemma&ct=single)

定理1的价值在于它找到了一个用于证明一个调度算法能否调度任一任务集的充分必要条件。那就是所有任务同时请求执行的情况下，每个任务仍能满足各自的期限, 那么这个任务集就可以被这个调度算法调度.

有了这个推论, 我们就可以证明RM调度的最优性了.

定理2: 如果一个任务集能够被静态调度, 那么RMS算法就能够调度这个任务集. 从这个意义上说, RMS是最优的静态调度算法.

这个定理的证明方法就是有名的交换法. 证明思路如下:

假设一个任务集S采用其他静态优先级算法可以调度,那么总有这样两个优先级相邻的任务i和j, 有Ti>Tj,且Pi≥Pj.把Ti和Tj的优先级Pi和Pj互换,明显可以看出这时S仍然可以调度, 因为在所有任务同时请求的情况下, 交换这两个任务不会影响其它任务的完成时间, 同时这两个任务都可以在各自期限内完成. 按照这样的方法,其他任何静态优先级调度最终都可以转换成RM调度.

RMS已被证明是静态最优调度算法, 开销小, 灵活性好, 是实时调度的基础性理论。即使系统瞬时过载, 也完全可预测哪些任务丢失时限。缺点是处理机利用率较低, 最坏的情况下，当n→∞时, 不超过ln2 (≈ 70%)。另外, RMS是充分但非必要条件。而在一般情况下，对于随机的任务集大约只有88%. 70%或者88%的处理器利用率对于许多实时应用来说是一个严重的限制，动态调度算法如最早截止期最先（earliest deadline first，EDF）或者最少空闲时间最先（least laxity first，LLF）已经被证明是最优的，并且能够实现100% 的处理器利用率.

#### 具有资源同步约束的RMS调度

当实时任务间共享资源时, 可能出现低优先级任务不可预测地阻塞高优先级任务执行的情况, 叫优先级倒置。这时RMS 算法不能保证任务集的调度, 必须使用有关协议控制优先级的倒置时间。常用的协议有优先级顶级协议和堆资源协议, 使用这些协议可使优先级的倒置时间最多为一个资源临界段的执行时间, 并且不会发生死锁。

#### 基于RMS 的非周期任务的调度

实时系统中的非周期任务可采用延迟服务器算法或随机服务器算法进行调度。它们的最大特点是可在周期任务的实时调度环境下处理随机请求。两者的基本思想是将非周期任务转化成周期任务, 再利用RMS算法进行调度。前者用一个或几个专用的周期任务执行所有非周期任务, 这种周期任务叫非周期任务服务器。根据周期大小,服务器有固定优先级, 服务器的执行时间被称为预算, 它在每个服务器周期Ts 的起点补充。只要服务器有充足的预算, 就可在其周期内为非周期任务服务。该算法实现简单, 但可调度性分析较难, 有时会出现抖动, 可能发生一个非周期任务在相邻两个服务器周期中连续执行2倍预算的现象, 与RMS理论不符, 需要适当修改RMS算法。随机服务器算法与延迟服务器算法相似, 但预算不是在每个周期起点补充, 而是在预算消耗Ts时间之后再补充。该算法与RMS分析算法一致, 但实现复杂。

### 2) 多处理器实时调度

#### 静态调度

问题描述：一组具有优先级关系的任务在m个处理器上运行，任务优先级关系用“<”表示，即如果两个任务（t1,t2）存在优先关系t1< t2，则t1必须在t2开始运行前完成。任务优先关系可用一个无环图来表示，称为计算图G，如图1.表示任务集S={t1,t2,t2,t4,t5}中任务存在优先关系<={(t1,t2),(t1,t3),(t1,t4),(t2,t6),(t3,t6),(t4,t5),(t5,t6)}。多处理其调度就是要找出长度最短的调度表。

静态多处理器调度又可以有如下三种调度规范：

1）基本（或非抢占）调度BS（Basic Scheduling）。任务在执行过程中不能被打断。

2）可抢占调度PS（Preemptable Scheduling），任务可被抢占，此处抢占不必基于优先级。

3）广义调度GS（General Scheduling）。 GS是一个理论上的概念，允许一个处理器可在同一时刻执行多个任务。事实上每个处理器在每一时刻最多只能执行能够一个任务。GS基于的前提是：在给定的时间段，处理器可以将其计算能力的一部分分配给一个任一个任务。并且规定同一任务不可以同时在多于一个处理器上并行执行。

定义 CA(G,k)表示调度规范A(BS、PS或GS)下，k个处理其的计算图G的最小计算时间。

定理3： 可抢占调度和广义调度的最小计算时间等于对任一k个处理器的计算图G，有

C*GS (G,k)=C*PS (G,k)

BS不能得到最短调度表，GS虽然能够得到最短调度表，但只是理论结果。定理三可知，PS和GS具有相同的最小计算时间，而PS在实际中是可实现的。

PS的一般生成过程是这样的：

1）更具任务优先级关系画出计算图G

2）按G生成GS调度表（为什么？）

3）采用某种方法将GS调度表换成PS调度表

#### 动态调度

问题描述：到达时间不确定而计算时间c和截止时间D已知的n个任务，运行在m个处理器上，n不确定，动态调度的目标是使系统能够对变化的环境做出迅速的反应。

动态调度的任务状态可以用l-c空间表示。横坐标表示任务的空闲时间l(t)，纵坐标表示任务的剩余时间c(t)。任务空闲时间l(t)=D - c(t)。图中虚线间隔表示一个时间单位，如t2的当前时刻剩余计算时间为3，空闲时间为2，因此deadline为3+2=5。

每隔一定时间，l-c空间将刷新一次。任务在l-c空间的位置变化具有不同含义：

1）任务执行：任务向下移动，c(t)变小

2）任务不执行：任务向左移动，l(t)变小

3）任务未到达：任务不动。有些任务未到达时已经确认，那么这些任务在l-c图上预留位置，当其到达时被激活

4）新任务到达：根据任务的计算时间c和空闲时间l，设置其在l-c空间的位置

5）任务执行完毕：任务到达l轴，此时c(t)=0

6）任务运行超时失败：任务落在c轴左边，此时l(t)<0

任务的截止时间的属性在l-c空间中由角度表示。相同截止时间的任务都位于125度叫的同一直线上，而截止时间沿45度叫递增。t1和t3截止时间相等，t2比t1，t3长。

在l-c空间里，任务可以按EDF和LLF调度算法调度。但是多处理器下EDF和LLF不是最优算法。

定理4：在多处理器下，如果任务计算时间、截止时间或到达时间不能预先确定，则最优调度算法不存在。

定理4表明基于l-c空间的多处理器动态调度算法没有最优解。由于该算法不能保证所有任务的截止时间，因此属于软实时调度。（软硬实时调度：）

1. 出现的问题

无

1. 解决方案（列出遇到的问题和解决方案，列出没有解决的问题）

无

1. 个人学习体会

通过这次作业，我对实时调度算法的理解变得更深入了。