TOPPERS中级实装讲座

（H8/3069F版：实时系统的构建）

应用程序实习篇：第2天

TOPPERS工程

培训工作组

关于本教材的注意事项

- **关于本教材的使用请注意以下几点。**

1. 关于著作权的表述

<TOPPERS中级实装讲座 应用程序实习篇(H8-3069F版：实时系统的构建）第2天>

---

2. 关于本资料如果有任何意见、建议、想法或疑问，请通过电子邮件与TOPPERS办事处进行联系。

3. 关于本资料的内容，出于调整和改善的目的，可能会对内容进行修订，且不予通告。

日 程 表

- 第1天

1.构建应用RTOS的实时系统	1.5小时
2.介绍实习课题	0.5小时
3.确认开发环境	0.5小时
4.制作话题烧水壶1	3.5小时
4－1.制作提示1	(0.5小时)
4－2.制作提示2	(0.5小时)

- 第2天

1.制作话题烧水壶2	1.5小时
2.评审 设计、实装的评审	0.5小时
3.提高实时性的方法	1.0小时
4.任务负荷的验证与对应	0.5小时
5.话题烧水壶中断的对应	2.0小时
总结	0.5小时

制作话题烧水壶2

1. 制作话题烧水壶2
2. 评审
3. 提高实时性的方法
4. 任务负荷的验证与对应
5. 中断的对应
6. 总结

接下来开始做话题烧水壶系统。

制作话题烧水壶
请继续制作

- 如有疑问请向老师提问
- 还需要**1.5**小时

评审

1. 制作话题烧水壶2
2. 评审
3. 提高实时性的方法
4. 任务负荷的验证与对应
5. 中断的对应
6. 总结

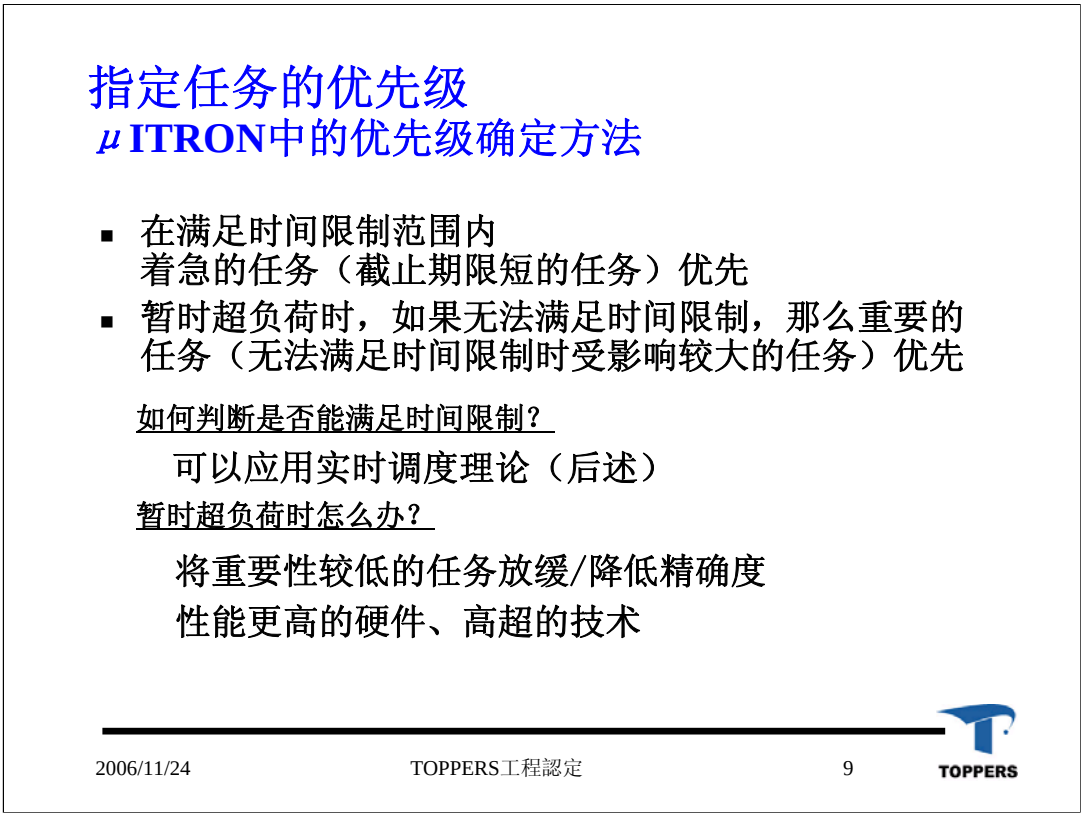
评审
对做的程序有把握的人公布

- 程序没有正确答案
- 由于各种需求不同而实装不同
- 设计质量本身经得起考验
  - 模块划分
    - 容量的最优化、系统的明确性
    - 重复的最小化、信息屏蔽
  - 模块的内聚性
    - 内聚性越高可维护性好
  - 模块的耦合性
    - 耦合性越低可维护性好

提高实时性的方法

1. 制作话题烧水壶2
2. 评审
3. 提高实时性的方法
4. 任务负荷的验证与对应
5. 中断的对应
6. 总结

话题烧水壶已经制作完成了，下面介绍一下应用RTOS的实时系统的实时性的提高方法。



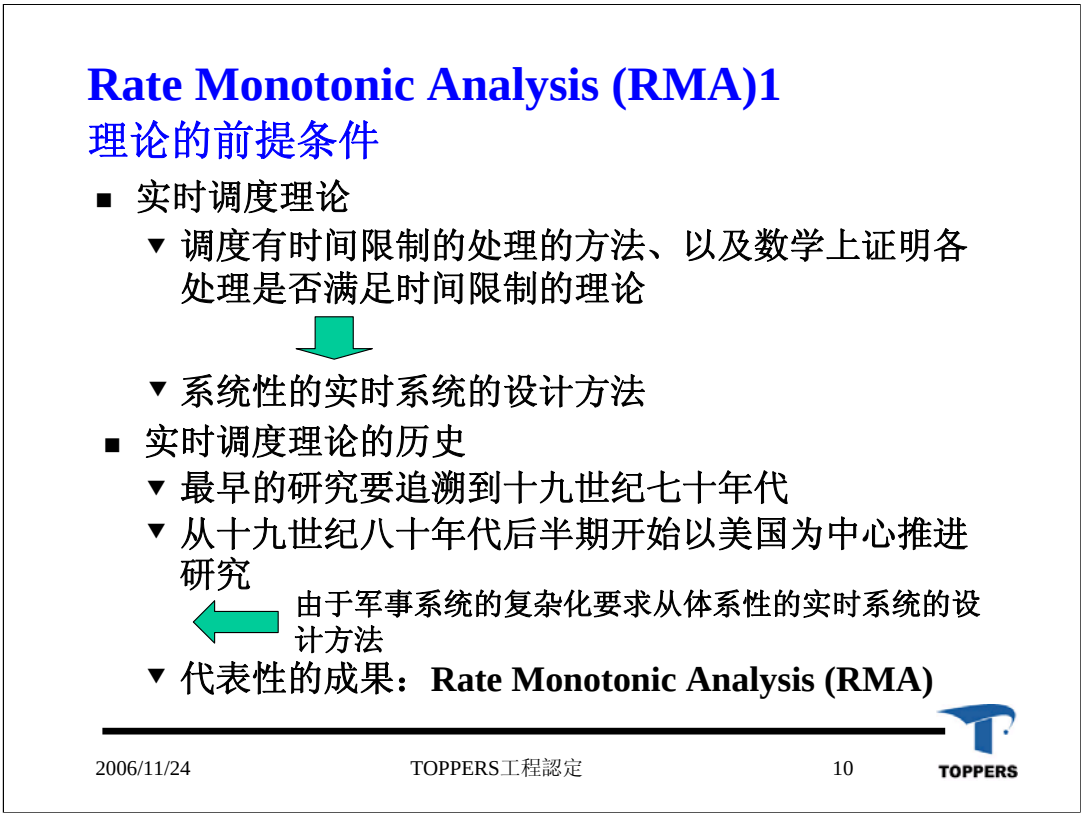
很多嵌入式RTOS（包括μITRON）采用静态优先级基本的占先调度方式。可以说，这种方式对于在单处理器上构建实时系统来说是最简单、系统开销最小的方式。而且任务优先级的指定方法的要点比较简单，有以下两点。

■在满足时间限制的范围内着急的任务（截止期限：任务周期）优先。

■暂时超负荷时，如果无法满足时间限制，那么重要的任务（无法满足时间限制时受影响较大的任务）优先。

说起来比较容易，但实际判断是否能满足时间限制是很难的。在完成产品时可以通过测试、工具等进行检测，但在初步设计时是很难判断的。后面会对作为实时调度理论的前提的Rate Monotonic analysis进行说明，但前提条件有很多，而且在要求特殊领域的特殊功能、性能的开发中不能简单地判断。

另一种情况是在暂时超负荷时发生无法满足时间限定的任务，这在最开始时是很难预测、对应的，而且通过硬件进行对应、或降低精确度等对策在实际的产品中也比较难实现。一般来讲，可以在充分进行产品设定后进行产品化，如果这样还是无法实现，可能就需要减少软件的调整、任务数，或进行中断程序化、使用周期句柄等，由高级技术者来进行对应。在本章的后半部分将对周期句柄、中断的使用等进行说明。



优先级基本调度的理论依据是Rate Monotonic Analysis （RMA）理论。它的起源可以追溯到1973年发表的Liu & LayLand的论文。给启动频率高的任务（周期任务中周期较短的任务）赋予高优先级的调度方法叫做Rate Monotonic Scheduling （RMS）。此方法非常适合于固定优先级的调度（即，如果可以用固定优先级的调度方式进行调度，那么也必定可以用RMS进行调度）。而且它的一个特点是调度的可能性可以通过基于CPU的负荷率的计算进行判断。

1980年中期以后，RMA理论被很多研究人员研究・扩展。比如说，诞生了解决优先级倒置问题的优先级继承方法、提高非周期任务的应答性的可延迟服务器或Boratec服务器等方法。可以说，RMA是对实时系统的Symantec设定作出了重大贡献的基础理论。

Rate：周期、Monotonic：单调、Analysis：分析

参考文献

1.白川洋充、竹垣盛一：实时系统及其应用、系统控制信息库
朝仓出版（2001年）
是实时调度理论唯一的日文书籍

2.JaneW.S.Liu：Real-Time Systems、Prentice Hall
是对实时调度理论进行了广泛深入解说的书籍、共610页

3.GiorgioC.Buttazzo：Hard Real-Time Computing Systems
Kluwer Academic Publishers（1997年）
以实时调度理论为中心，还介绍了一些相关的技术。比Jane Liu更易读。

系统负荷
最大负荷的确定 / 假定

- 为了保证所有处理都严格满足时间限制，必须确定系统的最大负荷
- 无法确定系统的最大负荷，或进行系统设计时要考虑到最大负荷不太现实的情况，要假定实际的最大负荷

- 决定 / 假定最大负荷，进行设计系统
- 最大负荷的描述方法

$$\text{最大负荷} = \sum_{\text{各任务}} 1\text{次的最大执行时间} \times \text{最大执行频率}$$

为了保证实时系统严格的实时性，必须知道系统的最大负荷。但在应用了RTOS规模的系统中，是无法实现从最初设计时开始就在计算出准确的最大负荷的基础上再进行设计的。其原因有以下几点：

1.设计与实装是不一样的。设计人员不可能对所有的程序都进行仔细的研究。

2.需要的功能会随时变化。

3.由于存在中间件、硬件等条件，最大负荷是不一样的。从某种意义上来讲，与其考虑好这些条件之后再开始生产，还不如在开发后再进行验证，这样效率可能会更高。

所以，在构建实际的实时系统时，要假定最大负荷进行系统设计。

Rate Monotonic Analysis (RMA)2
理论的前提条件

- 几个大前提（以下是默认的前提）
  - ▼静态优先级分配下的占先式优先级
    基本调度 ⇨ μITRON的调度方式
  - ▼能够预测系统的最大负荷是“保证”的基本前提
  - ▼以周期任务（或知道最小启动间隔的非周期任务）为基本
  - ▼以单处理器为基本
- 适用于各种资源
  - ▼处理器 ⇨ 以下会以处理器为例进行讨论
  - ▼网络 … 并不是完全占先式这一点不一样

2006/11/24 TOPPERS工程認定 12 TOPPERS

RMA理论需要几个默认的前提。

▼在静态优先级分配下，执行占先式优先级调度
目前嵌入式用的RTOS采用该调度方式

▼能够预测系统的最大负荷是“保证”的基本前提
在生产结束时能够实现，但在实际的生产中需要的功能、制约等会发生变化
而且，生产上还存在竞争对手，受到竞争机种的式样的左右
在有些生产中很难实现

▼以周期任务（或知道最小启动间隔的非周期任务）为基础

▼以单处理器为基础
本讲座以处理器资源为重点。当然，也受到实际的硬件、外部设备、通信很大的影响

Rate Monotonic Analysis (RMA)3
Critical Instant定理的任务集的假定

1.仅考虑周期任务（或知道最小启动间隔的非周期任务）
2.事先知道各任务的最大执行时间
3－4.各任务的相对截止期限低于周期（或最小启动间隔）
5.任务间没有同步通信
6.任务不会自行中断执行
7.不考虑任务切换等系统开销
8.按处理器只有一个的情况考虑

Critical Instant定理非常重要，是RMA的出发点。假设用该定理进行处理的任务集中有以下各项。（也有一般实时系统中不可能存在的假定）

1.仅考虑周期任务（或知道最小周期启动间隔的非周期任务）

2.事先知道各任务的最大执行时间

3－4.各任务的相对截止期限小于等于周期（或最小启动间隔）

5.任务间没有同步通信

6.任务不会自行中断执行

7.不考虑任务切换等系统开销

8.处理器只有一个

注意：假设任务在任意相位启动

使3－4是共通的，这可以实现后面的“根据处理器使用率进行调度检查”的任务集的假定和比较。

Rate Monotonic Analysis (RMA)4
Critical Instant定理

- **Critical instant（定义）**
  - 对于某任务的**critical instant**是指从该任务启动到结束的时间最长的状况
- **Critical instant原理**
  - 在静态优先级分配下进行占先式优先级基本调度时，对某任务的**critical instant**是指在以下列举的前提条件下某任务启动的同时比此任务优先级更高的所有任务也同时启动

前提条件

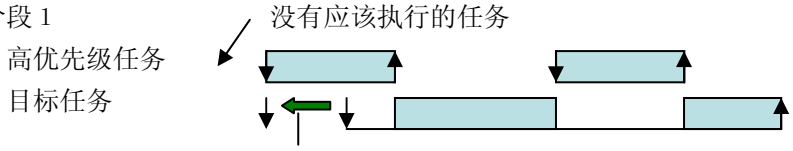
任务没有发生超时限时只考虑范围

→不会对任务进行多重启动

Critical意思是“危险的”，instant意思是“瞬间”，Critical instant定理的意思是“任务的最危险状况是启动任务的同时比此任务优先级更高的所有任务也同时启动”。也就是说，此时的任务容易发生超时限。

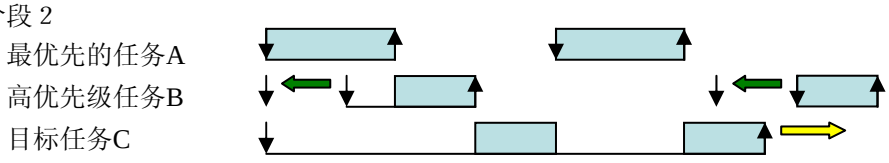
用图来进行说明

阶段 1



即使将启动时间提前，结束的时间也不会改变

阶段 2



如果将启动时间提前，目标任务的结束时间就会推后

从Critical Instant定理获知
可调度的必要充分条件

- 即使是在**Critical Instant**时启动，如果在截止期限内结束执行，该任务也是可以调度的
- 反过来也成立
- 任务***i***（任务的索引是按照优先级高的顺序）可调度的必要充分条件

$$\exists\, t, 0 < t \leqq D_i, \sum C_j \left\lceil \frac{t}{T_j} \right\rceil \leqq t$$

Tj：优先级为***j*** 的任务的周期

Dj：优先级为***j*** 的任务的相对截止期限

Cj：优先级为***j*** 的任务的最大执行时间

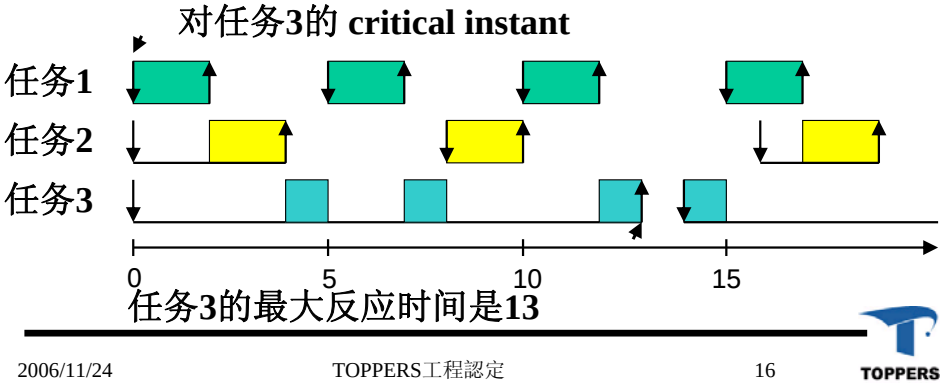
Critical Instant是指最危险的状况。也就是说，是任务的执行时间最长的状况。如果此状况能遵守截止期限，那么可以说此任务能进行调度。理论上来讲，如果实时系统的所有任务都能满足上述必要充分条件，则可以保证实时性。

ABS($\lceil t/Tj \rceil$)是最大频率、上限值。

Dj表示优先级第j的任务的相对截止期限。相对截止期限是指距启动时间的相对时间。还有与相对截止期限相对的绝对截止期限，绝对截止期限是用年、月、日、分、秒这样的绝对时间来表示截止期限。

调度可能性的验证1
准备用于验证的简单任务集

j	T_j	D_j	C_j	使用率（负荷）
1	5	5	2	40%
2	8	8	2	25%
3	14	14	3	21.4%



2006/11/24 TOPPERS工程認定 16 TOPPERS

为了验证必要充分条件，要准备简单的任务集。
任务1优先级最高，周期：5，相对截止期限：5，最大执行时间：2
任务2优先级第二高，周期：8，相对截止期限：8，最大执行时间：2
任务3优先级最低，周期：14，相对截止期限：14，最大执行时间：3
如果以任务3为例来说明使用率，则是在14个周期内执行3任务，即$\frac{3}{14}$ = 0.214285……
即使加上所有的负荷也是86.4%，从使用率来看可以进行调度。
目标的任务3即使是Critical Instant，也在t=13时结束执行，能够遵守截止期限。

调度可能性的验证2

根据必要充分条件公式进行评估

- 如果应用右边的公式 $\exists t, 0 < t \leqq D_i, \Sigma C_j \left\lceil \frac{t}{T_j} \right\rceil \leqq t$

  $t = 13 \leqq 14 \quad 2\left\lceil \frac{13}{5} \right\rceil + 2\left\lceil \frac{13}{8} \right\rceil + 3\left\lceil \frac{13}{14} \right\rceil = 13 \leqq 13$

- 最大响应时间的计算方法

  response time analysis

  $r^{(0)} = C_i$

  $r^{(i+1)} = \Sigma C_j \left\lceil \frac{r^{(i)}}{T_j} \right\rceil$

  $r^{(0)} = 3$
  $r^{(1)} = 7$
  $r^{(2)} = 9$
  $r^{(3)} = 11$
  $r^{(4)} = 13$
  $r^{(5)} = 13$

  一致的是响应时间

- 此方法的缺点

  需要知道各任务的正确最大执行时间

TOPPERS

应用右边的公式，任务3的相对截止期限为14，那么下面的$t = 13$时，$13 \leqq 13$成立。

如果$t = 12$，$2\times2 + 2\times3 + 3\times1 = 13 \leqq 12$

如果$t = 14$，$2\times2 + 2\times3 + 3\times1 = 13 \leqq 14$

如果$t = 13$，那么可以判定为可调度。

最大响应时间的计算使用计算机，表示计算最大响应的方法。

$r^{(0)} = 3$

$r^{(1)} = 2\times \mathrm{ABS}\left(\left\lceil \frac{3}{5} \right\rceil\right) + 2\times \mathrm{ABS}\left(\left\lceil \frac{3}{8} \right\rceil\right) + 3\times \mathrm{ABS}\left(\left\lceil \frac{3}{14} \right\rceil\right) = 2 + 2 + 3 = 7$

$r^{(2)} = 2\times \mathrm{ABS}\left(\left\lceil \frac{7}{5} \right\rceil\right) + 2\times \mathrm{ABS}\left(\left\lceil \frac{7}{8} \right\rceil\right) + 3\times \mathrm{ABS}\left(\left\lceil \frac{7}{14} \right\rceil\right) = 4 + 2 + 3 = 9$

$r^{(3)} = 2\times \mathrm{ABS}\left(\left\lceil \frac{11}{5} \right\rceil\right) + 2\times \mathrm{ABS}\left(\left\lceil \frac{11}{8} \right\rceil\right) + 3\times \mathrm{ABS}\left(\left\lceil \frac{11}{14} \right\rceil\right) = 6 + 2 + 3 = 11$

$r^{(4)} = 2\times \mathrm{ABS}\left(\left\lceil \frac{13}{5} \right\rceil\right) + 2\times \mathrm{ABS}\left(\left\lceil \frac{13}{8} \right\rceil\right) + 3\times \mathrm{ABS}\left(\left\lceil \frac{13}{14} \right\rceil\right) = 6 + 4 + 3 = 13$

最大执行时间可以在写完程序时进行测定。想在系统设计阶段确定该时间是很难的。为了测定系统所有的时间，会需要很大的工作量。

调度可能性的验证3
最优的优先级分配

假设与**Critical instant**原理相同，
那么如何对任务分配优先级才能达到最优呢？

- **Deadline monotonic scheduling**

  相对截止期限越短的任务分配的优先级越高

  → *越急的任务优先级越高*

  最初是在“相对截止期限=周期”的前提下开始争论的。周期越短的任务分配的优先级越高的分配方式叫做**ratemonotonic scheduling(RMS)**

Monotonic的意思是“单调”，Deadline monotonic scheduling是相对截止期限越短的任务分配的优先级越高的分配方式。在最初的研究中是在“相对截止期限=周期”的前提下开始争论的，所以经常把deadline换成rate来给此种方式命名，这种命名方式比较普遍。

处理器的使用率和调度的可能性1

任务集的假定

1.只考虑周期任务（或知道最小启动间隔的非周期任务）
2.事先知道各任务的最大执行时间
3.各任务在周期开始时可以执行
4.各任务的截止期限是周期的结束
“相对截止期限=周期”
5.任务间没有同步通信
6.任务不会自行中断执行
7.不考虑任务切换等系统开销
8.处理器只有一个

➡ 对处理器的负荷不改变的任务集

在根据处理器的使用率验证调度的可能性之前，要假定任务集。该假定与前面提到的critical instant定理的任务集的假定基本相同，只是3和4有一些差异。

critical instant定理的3、4的条件是假定各任务的相对截止期限小于等于周期，而该假定是假设在周期开始时可以执行、相对截止期限是周期的结束。该假定的任务集假设对处理器的负荷不改变。

处理器的使用率和调度的可能性2
任务集可调度的充分条件

- 处理器使用率方面的充分条件

$$\sum_{j=1}^{n} \frac{C_j}{T_j} \leqq n(2^{1/n} - 1)$$

n	右边
2	0．83
3	0．78
∞	0．69

n：任务数

Tj：优先级第**j**的任务周期

Cj：优先级第**j**的任务最大执行时间

左边：处理器的使用率

➡ 悲观的上限值（最充分条件）
平均达到**0.88**也可以

!损失的31％是由静态分配优先级带来的本质性的极限

2006/11/24 TOPPERS工程認定 20 TOPPERS

对处理器使用率、处理器的负荷不改变的任务集可调度的充分条件如下所示：

$$\sum_{j=1}^{n} \frac{Cj_n}{Tj} \leqq n(2^{1/n} - 1)$$

n：任务数

Tj：优先级第*j*的任务周期

Cj：优先级第*j*的任务最大执行时间

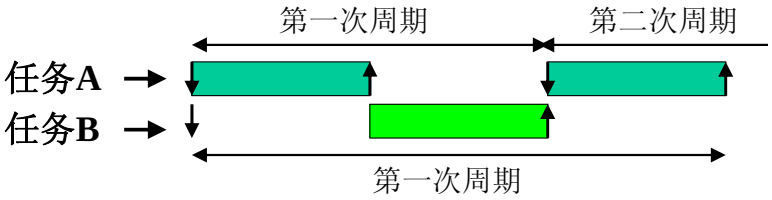
左边：处理器使用率（换句话说是CPU的实际负荷）

问题在于处理器的使用率非常低。这是最充分条件，比如说 n 为∞时，如果实际上处理器只能使用69％，在实时性和性能上就会出现大问题。请考虑成剩下的31％只是无法保证，实际上也存在可以调度的情况。这取决于静态优先级的设定，在下一页会对验证、回避方法进行介绍。

该充分条件是悲观的上限值。曾有科学者使用随机数计算平均上限值，结果是达到0.88也可以。

处理器的使用率和调度的可能性3
损失的31％是什么？（证明的入口）

- 考虑周期比率为1比1.5的两个任务



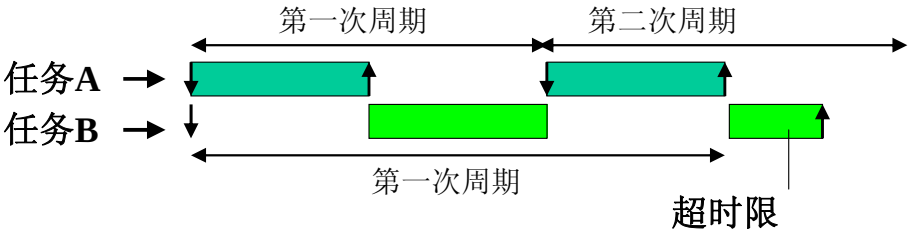
虽然任务A的处理器使用率约占一半，但任务B的第1次周期中能使用的处理器时间只有三分之一

剩下的六分之一的处理器的使用率无法利用

实际上周期比为1：√2 时是最差的

在周期比率为1：1.5的例子中，如果任务B的最大执行时间超出此长就会发生超时限。而距离截止期限还有三分之一的时间残余。也就是说，根据周期的比率，即使处理器的使用率有剩余，也可能无法进行调度。

处理器的使用率和调度的可能性4
静态优先级分配的本质界限



- 任务B的第1次周期的绝对截止期限早于任务A的第2次周期的截止期限，但相对截止期限是任务A较短，所以给任务A分配更高的优先级

- 由于优先级固定导致的本质问题

在静态任务优先级分配中，即使优先级低的任务马上就要发生超时限，也会优先执行距离截止期限还有一定时间的优先级高的任务。这是静态任务优先级方式的本质问题。在实际的系统设计中，可以为了实现harmonic task set进行设计，或为了减少任务执行时间进行程序设计等，通过正确使用中断等，在一定程度上减少处理器的负荷、确保实时性。

处理器的使用率和调度的可能性5
上限值悲观的理由

- 处理器的使用率为**69％**是在极其特殊的任务集中产生
  ***n*个任务周期比率如下的情况**

$$1 : 2^{\frac{1}{n}} : 2^{\frac{2}{n}} : \cdots : 2^{\frac{n-1}{n}}$$

  所有任务比率在**1：2**以内

  ⇒ 如果周期比率变大，上限值就会变大

- **Harmonic task set** 的情况
  各任务周期的比率为倍数关系
  可以调度到处理器使用率达到**100％**
  包括**Harmonic task subset**的情况也很有利

TOPPERS

处理器的使用率最差（69％）时是任务周期的比特殊时。

是所有任务比率在1：2以内时。在系统设计上可以避免此种周期比的设计。处理器的使用率能达到最高的周期比的任务集叫做harmonic task set，任务周期比为整数倍率关系。

处理器的使用率和调度的可能性6
动态优先级分配时（参考）

通过动态变更优先级分配来进行解决

- **Earliest deadline first scheduling (EDFS)**
  - 从截止期限较近的任务开始按顺序分配高优先级
    这是最优的动态优先级分配算法
  - 任务集可以进行调度的必要充分条件

$$\sum_{j=1}^{n} \frac{C_j}{T_j} \leqq 1$$

处理器的能力

100%发挥

***n*：任务数**

***Tj*：优先级第*j*的任务周期**

***Cj*：优先级第*j*的任务最大执行时间**

作为参考，下面介绍一下进行动态任务优先级调度的Earliest deadline first scheduling（EDFS）。该调度方式是通过将截止期限较近的任务优先级动态变高来挽救超时限的方式。

RTOS中也有支持EDFS的。那么，多数RTOS都没有采用EDFS吗？实际上EDFS也有缺点。

在优先级基本调度中，如果任务的执行时间变长，优先级低的任务就会很不利。但在EDFS中，不知道任务的执行时间变长后会对哪个任务造成不利。这样在存在最不想出现超时限的任务时，就无法制定如何才能回避的对策。

放宽对任务集的假定1

- **“1.只考虑周期任务”和“2.知道最大执行时间” 是能够预测最大负荷的大前提**
- **正在研究既要遵守已知最大执行时间的周期任务的截止期限，又要最早执行不知道最大执行时间的任务的方法**
- **关于“3.各任务在周期开始时变为可以执行”，可以考虑任务的可执行比周期的开始要晚的效果**
- **关于“3－4.相对截止期限短于周期”和“4.各任务的截止期限是周期的结束”，必须考虑多重启动任务的情况，虽然条件很复杂，但可以除去假定**

讨论一下除去对RMA中的任务集的假定的可能性。

放宽对任务集的假定2

- 关于“**5.**任务间没有同步通信”，如果知道优先级低的任务等待的最大时间（最大**Block**时间：***Bi***），那么可以将其效果放到公式中

$$\forall i,\ \exists t, 0 < t \leqq D_i, \sum C_j \left\lceil \frac{t}{T_j} \right\rceil + B_i \leqq t$$

- “**6.**任务不会自行中断执行”是尚未深刻研究的课题
- 关于“**7.**不考虑任务切换等系统开销”，可以考虑单纯的任务切换的系统开销
  - ⇒ 将任务切换两次所用的执行时间追加到各任务的最大执行时间中
- 关于“**8.**处理器只有**1**个”，很难扩展到执行任务的处理器不固定的情况

TOPPERS

$\lceil x \rceil$ 是满足x≦n的最小整数，即进位值

$\lfloor x \rfloor$ 是满足n≦x的最大整数，即舍去值（高斯的符号［］是一样的）

非周期任务的处理
另一个任务的动作

- 目标
  - 保证周期任务满足时间限制的同时，尽早执行异步的任务
- 最简单的方法－ **Background**方式
  - 以最低的优先级执行非周期任务
    - 最容易实现
    - 可以用普通的方法处理周期任务的时间限制
    - 不能说尽早执行了异步任务
- 非周期任务服务器方式
  - 准备执行非周期任务的服务器任务
  - 关于服务器任务的执行控制方法，提出了几个算法

如果嵌入式系统进行扩大化，需要实时性的任务的比率与整个任务相比就会降低。实际上，构建庞大的实时系统是很难的。在这样的系统中，非周期任务的有效执行是一大课题。最简单的实现方式是正确选择以最低优先级执行的任务的执行方式。任务数较多时，可以使用rot_rdq和irot_rdq服务调用循环执行。虽然有些复杂，但也有使用异步任务服务器的方式。关于异步任务服务器，对以下算法进行了研究。

・轮询方式

指定服务器的周期和最大执行时间，与普通周期定时器一样进行调度。

没有要执行的任务时，服务器立即结束执行，不执行下一个周期。

・优先级交换服务器（priority-exchange server）

・可遲延服务器（deferrable server）

・分散服务器（sporadic server）

Slack Stealing方式

理论上能比上述方式更早调度非周期任务。但算法比较复杂，系统开销较大。

从理论到实际系统
各种课题

- 现在的**RTOS**与**RMA**最匹配
  - 大多数实时**OS**都支持占先式优先级基本调度
- 考虑实时**OS**的系统开销
  - 系统调用的处理时间要包含在调用它的任务的最大执行时间中一起考虑
  - 将切换任务**2**次的处理时间加到各任务的最大时间中（任务间没有同步・通信时）
- 将任务和中断句柄、时间事件句柄分开使用
  - ⇒ 在以后的章节中进行说明
- 任务最大执行时间的估算
  - 任务最大执行时间的估算困难是阻碍理论应用到实际系统的最大因素

2006/11/24 TOPPERS工程認定 28 TOPPERS

通过前面的内容，对μITRON中的任务相关的适当的调度方式应该已经很清楚了。具体内容如下所示：

1.Deadline monotonic scheduling

任务的相对截止期限越短分配的优先级越高。

2.Harmonic task set

通过将任务周期比设定为倍率关系，来提高处理器的使用率。

在μITRON中，任务以外也可以记述功能。任务以外叫做非任务语境，中断句柄、时间事件句柄等与其相对应。如果使用这些句柄，可以按照任务规则以外的规则记述程序。

在RMA中，任务最大执行时间的估算是个大课题。在前面已经说过了，由于实际的实时系统有各种要求，所以在生产结束之前无法确定程序本身，或由于嵌入式系统的扩大化而导致很难估算。而且，作为实时OS的一个优点的外部资源的中间件是黑盒状态。在实际的实时系统开发中，要正确设计样机，测定并预测执行时间，或在生产时也使用螺旋式开发方法、不会产生太大的任务负荷的开发方式来提高安全性等。而且，曾开发了多个系统的软件工程师的经验以及感觉等也是非常重要的要素。

非任务语境和任务语境
CPU的执行分为两个语境

- 在μITRON4.0式样中，按以下处理单位对CPU的执行进行管理

1	中断句柄、服务例程	非任务语境
2	时间事件句柄	非任务语境
3	CPU异常句柄	取决于调用语境
4	扩展服务调用例程	取决于调用语境
5	任务和任务异常	任务语境

- 被看成任务以及任务一部分的语境叫做任务语境，除此之外的叫做非任务语境
- 非任务语境比任务语境优先级高

将可以看成任务处理的一部分的语境统称为任务语境，除此之外的语境统称为非任务语境。在μITRON4.0式样中，对以下5个处理单位的执行进行控制。这些执行状态也要划分到其中一个语境中。

a)中断句柄

a.1)中断服务例程

b)时间事件句柄

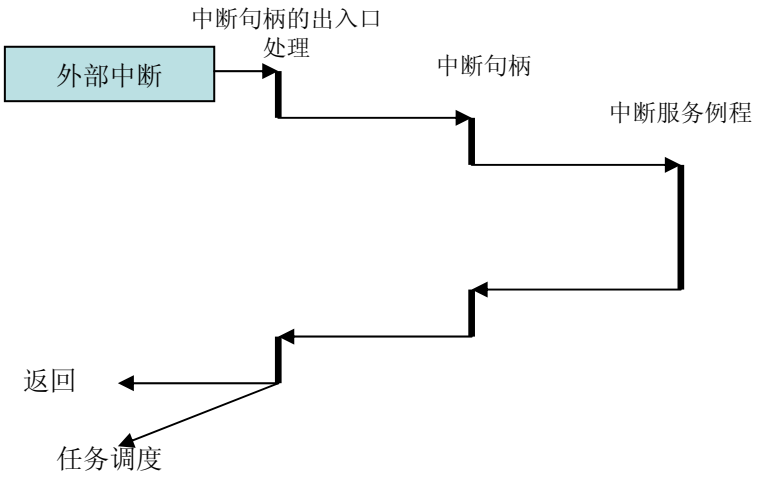
c)CPU异常句柄

d)扩展服务调用例程

e)任务

e.1)任务异常处理例程

对应如上表所示。非任务语境的任务优先级高，如果使用非任务语境，则能够实现更高的实时性以及更小的系统开销。以下中断模型在任务执行中延迟任务处理，执行中断服务例程。

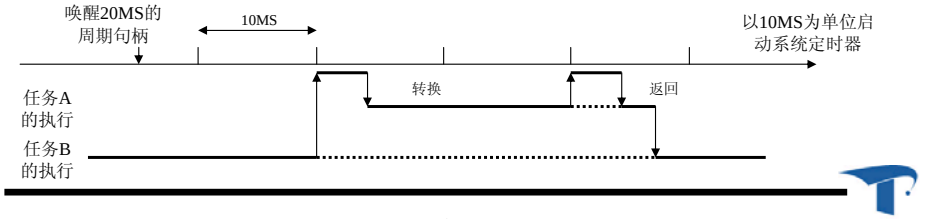


更有效的周期句柄
无任务切换的程序运行

- **μITRON4.0中定义了以下3个时间事件句柄**

周期句柄	标准版
警报句柄	全功能版
超时句柄	全功能版

- **周期句柄是在一定时间启动的时间事件句柄，在任务中优先执行**



2006/11/24 TOPPERS工程認定 30

μITRON4.0中定义了以下3个时间事件句柄。周期句柄是在标准文档中定义的句柄，也可以在TOPPERS/JSP中进行使用。

a)周期句柄（按一定周期启动）

b)警报句柄（再指定的时刻启动）

c)超时句柄（在任务超过指定的时间使用处理器时启动）

在前面的章节中已经讲过，在周期性启动任务时，优先级更高的任务的执行会导致执行的精确度异常。周期句柄由定时器中断启动，并在非任务语境下执行，所以执行的精确度较高。而且，根据句柄中的程序的结果，除执行后要进行任务转换的情况以外可以不进行任务调度而直接执行，所以执行处理效率很高。

非任务语境的执行
在非任务语境中无法制造出等待状态

- 因为在非任务语境下没有任务这样的等待状态，所以服务调用受到限制
- 不能使用处于等待状态的服务调用，只能使用向任务传递事件的服务调用

功能	在TOPPERS/JSP中能够使用的服务调用
任务管理功能	iact_tsk
任务附属同步功能	iwup_tsk, irel_wai, iras_tex
同步通信功能	isig_sem, iset_flg, ipsnd_dtq, ifsnd_dtq
系统状态管理功能	irot_rdq, iget_tid, iloc_cpu, iunl_cpu

非任务语境在CPU的异常处理中执行。所以不仅无法制造出等待状态，如果在软件延迟中制造了等待状态，还会妨碍其他非任务语境、任务语境的执行。因此，能够使用的服务调用仅限于向任务传递事件的服务调用。同样的，从非任务语境调用控制驱动等设备的函数也存在限制。

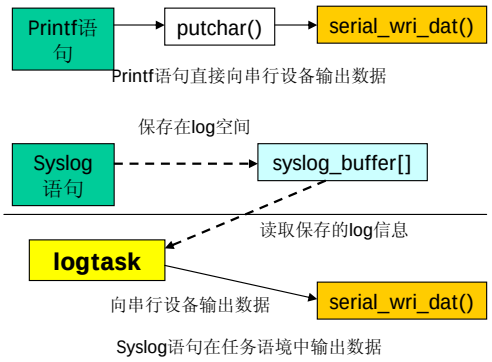
在H8/3069F中使用的串口、网络驱动内部，使用来自于硬件的中断函数来运行。周期句柄、中断例程能够实现时间临界执行，而另一方面，如果过多使用或实装很复杂，就有可能破坏整体的实时性，需要充分考虑系统设计。

下面介绍一下非任务语境下能够使用的服务调用。

iact_tsk	iact_tsk　任务启动 —启动任务
iwup_tsk	iwup_tsk　任务唤醒 —唤醒任务
irel_wai	irel_wai　等待状态强制解除 —等待状态的强制解除
iras_tex	iras_tex　任务例外处理要求 —任务异常处理的要求
isig_sem	isig_sem　信号量资源释放 —信号量资源的归还
iset_flg	iset_flg　事件标志置位 —事件标志的设置
ipsnd_dtq	ipsnd_dtq　轮询模式下的数据队列释放 —在轮循模式下归还给数据队列
ifsnd_dtq	ifsnd_dtq　数据队列强制释放 —强制归还给数据队列
irot_rdq	irot_rdq　任务优先级轮转 —循环任务的优先顺序
iget_tid	iget_tid　运行状态任务ID的参照 —查看执行状态的任务ID
iloc_cpu	iloc_cpu　迁移至CPU锁定状态 —进入CPU锁定状态
iunl_cpu	iunl_cpu　CPU锁定状态解除 —解除CPU锁定状态

syslog和printf语句
两种LOG显示方法

- 关于非任务语境运行程序的调试，原则上不能做出等待状态，所以在没有ICE等时LOG调试成为调试的主体
- printf语句要根据设备驱动输出数据，所以不能使用
- 可以使用syslog语句或syslog_n语句的LOG输出



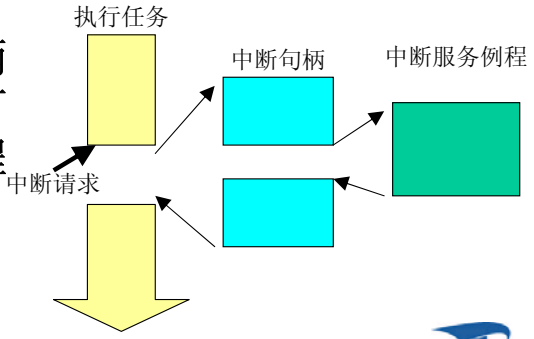
printf语句是使用UNIX的标准输入输出功能输出消息的C语言函数。包括printf语句在内的标准输入输出功能在stdio.h引用文件（在newlib中定义）中进行声明。标准输入输出功能将输入目的地、输出目的地作为文件进行定义。在UNIX的文件中字符设备与存储设备在声明上没有差异，所以可以与命令解释程序的功能相结合自由地设定输入输出的目的地。

在TOPPERS/JSP中，按照标准只支持串行设备，没有像UNIX的命令解释程序这样强大的用户接口，所以无法支持stdio.h的功能。但是，在不能及时准备开发环境时，在LINUX等UNIX环境下进行应用程序的模拟开发是非常有效的手段，在UNIX上的开发中经常使用printf()、sprintf()、putchar()。因此，如果使用任务监视器，对串行设备的printf()、putchar()就可以使用。此时，请使用monitor目录下的stdio.h引用文件。

但是，直接使用串行驱动的printf()不能在非任务语境下使用。此时，请使用TOPPERS/JSP支持的syslog()或syslog_n()。执行syslog语句时，储存SYSLOG结构体的数据空间syslog_buffer[]的LOG信息，并通过logtask输出到串行设备，所以无论语境状态如何都可以使用。

更有效的中断
只在需要硬件事件时运行程序

- 中断是在有硬件的执行请求时向处理器传递请求的机制
- 处理器无需监视硬件
- 在μITRON4.0中，中断时执行中断句柄和中断例程，用户可以记述中断服务例程

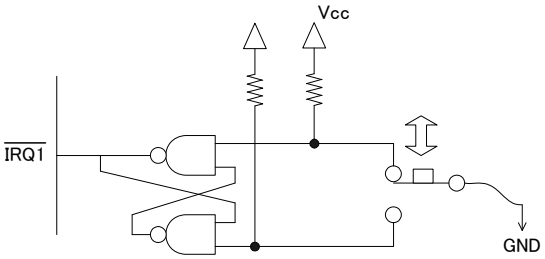


处理器依靠外部硬件实现功能时，需要能有效从硬件获取要求的机制。这就是中断功能。作为μITRON4.0的对象的通用微处理器嵌入了该机制，而且μITRON4.0中也嵌入了能顺利管理中断的功能。与周期句柄相比，周期性监视硬件的机制是不需要的，所以能够更有效的使用处理器。

μITRON4.0中的中断程序分为中断句柄和中断服务例程分别执行。中断句柄由μITRON4.0的实装者提供，而中断服务例程是用户（包括中间件的提供者）要创建的程序。使用中断能够实现更高效的截止期限的控制。

中断需要硬件的对应
需要能使中断顺利运行的电路

- 关于中断的对应，需要硬件的对应，而且在构建系统时需要从产品角度规定
- 中断电路可能会提高成本，应该从系统全局出发仔细考虑（话题烧水壶的按钮、壶盖中断是否超出规格了呢？）

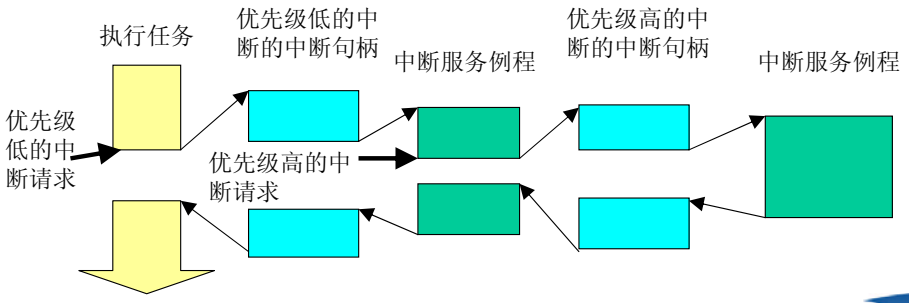


简单的按钮中断回路

上面举了一个边沿触发器的按钮中断电路的例子，但如果真像话题烧水壶这样有5个按钮，就要统一它们、通过on、off来发生中断等等，这样电路就会非常昂贵。如果要考虑产品成本，使用单片CPU的端口直接监视按钮（没有多余的电路）的方法能够大大降低成本。作为一个产品，要使用中断等来减少处理器的负荷，而且从总成本上要能看出可以降低成本。

是否做到这样？多重中断
在其他中断中执行中断

- 当处理器依靠外部硬件实现多个机能时，必须管理多个中断
- 防止截止期限短的硬件受到其他中断阻碍的机制就是多重中断



当处理器管理多个中断、且中断服务例程的执行时间比截止期限最短的中断长时，需要将优先级高（截止期限短）的中断插入到截止期限长的中断中来解决此问题。这种机制就是多重中断。

在大多数处理器中，在给中断设定了优先级、且优先级低的中断正在执行而不是处于中断禁止状态时，如果发生比它优先级高的中断的要求，可以执行此中断。

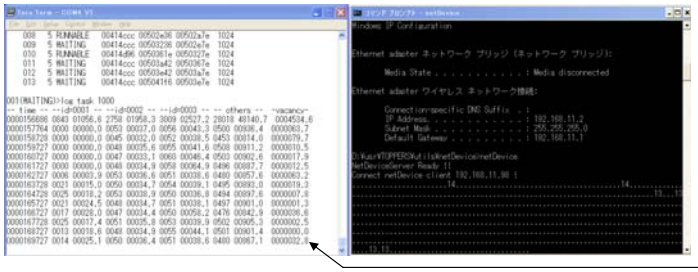
任务负荷的验证和对应

1. 制作话题烧水壶2
2. 评审
3. 提高实时性的方法
4. 任务负荷的验证和对应
5. 中断的对应
6. 总结

尝试用任务监视器调查轮询版话题烧水壶的任务执行状态。
在这里介绍一下TOPPERS/JSP H8版中中断对应的步骤。

轮询模式下的烧水壶系统任务验证
用任务监视器确认执行状态

- 在模拟运行壶时，请用任务监视器的**LOGTASK**命令确认任务的执行状态并启动定时器
- 如果启动定时器，就会知道几乎没有任务的空闲时间



如果启动定时器，则没有任务未执行的时间

请启动话题烧水壶系统（没有完成的人请下载POTTN的jsp.mot来运行），并在任务监视器上启动LOG TASK命令，时间为1000ms。

mon>LOG TASK 1000←

Vacacy的项目上显示任务1000ms执行时的空闲时间

此时间中包括中断的时间，所以并不是准确的任务未执行时间。各任务的执行时间不包括netDevice的访问时间，所以也不能说是准确的时间。

此后，如果在壶模拟器上启动厨房定时器，就会开始执行id=0001的timer_task，从0变成执行时间。

此时，观察vacacy就会发现接近0ms的显示。如果启动厨房定时器，就会知道优先级低的定时器任务的运行已接近截止期限。

轮询版的任务设定
实时性的验证

- **event_task**以外的任务周期性启动，监视话题烧水壶的内部
- **timer_task**的优先级低，所以如果所有的任务都在运行，它的运行就会受到阻碍

ID	任务函数名	优先级	功能
1	timer_task	12	如果启动各任务的启动程序、定时器，就会每200ms监视一次时间，对蜂鸣器的时间进行控制
2	button_task	8	每50ms监视一次按钮和壶盖
3	heater_task	11	每100ms控制一次加热器
4	error_task	10	每100ms监视一次加热器错误
5	event_task	4	解析来自于button_task的事件，对各任务指示动作

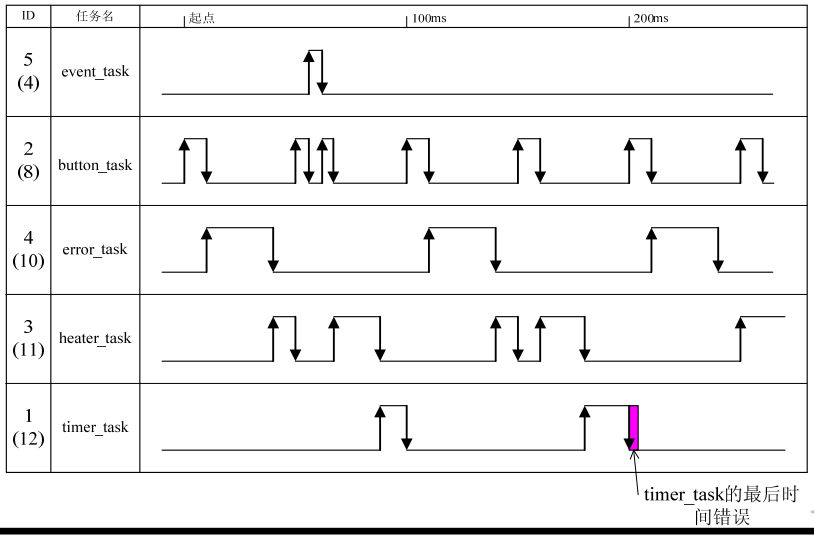
介绍一下POTTN中话题烧水壶的程序内容。

此任务的周期为harmonic task set。话题烧水壶的主体是在PC上，它访问端口要通过LAN，所以系统开销非常大。而且，虽然显示时机是每隔1秒一次，但timer_task是以200ms的间隔运行的。这是因为蜂鸣器的on、off时间也是由timer_task同时进行控制。如果是实机则没有问题，在H8/3069F中能够实现话题烧水壶的控制。

button_task每50ms监视一次按钮和壶盖的端口，如果状态发生变化就使用事件标志将事件发送给event_task。如果将button_task和event_task合为一个就能够减少系统开销，但为了在POTTN中能够容易实现中断化，所以采取了此结构。heater_task是根据模式专门控制加热器的任务，而error_task是监视温度、专门检测错误的任务。

话题烧水壶任务的执行状态
在任务中添加了netDevice的访问时间的时序图

- 表示各任务状态的假定时序图



TOPPERS

这是启动厨房定时器时的任务的时间图。在此时间图中，各任务中包含netDevice的访问时间，看作没有LOG任务和任务监视器，并忽略中断、任务转换等的系统开销。因此，与实际的运行是不一样的。假设以下是任务的执行周期：

1)event_task 5ms

2)button_task 10ms

3)error_task 30ms

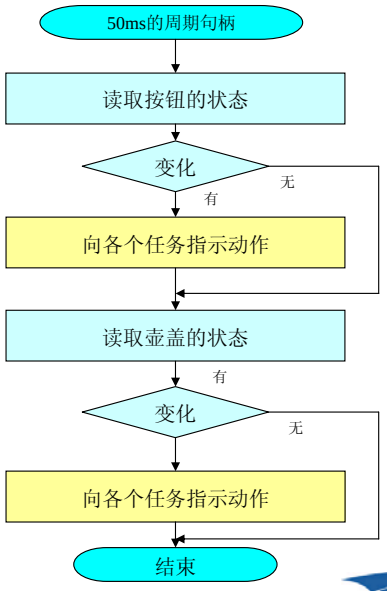
4)heater_task 30ms

5)timer_task 35ms

实际上，为了实现LCD的显示，heater_task和timer_task使用信号量进行排他控制，所以可以想象到会发生更加复杂的状态迁移。

理想的改善
考虑成本和实时性

- 组合button_task和event_task并置换成中断、周期句柄，这样可以改善执行效率
- 如果要考虑成本，周期句柄的做法比较有用



前面已经讲过，POTTN的话题烧水壶的结构很容易进行中断化。一般将button_task改写成按钮和壶盖的中断服务例程并使用事件标志将状态迁移传递给event_task，这样很简单就能对应。如果将button_task和event_task合为一个中断服务例程就可以减少两个任务，这样就能减少处理器的负荷，但netDevice的限制导致在此开发环境下无法实现。同样，如果将button_task和event_task合为一个周期句柄，就不会提高成本，还可以减少处理器的负荷。

［netDevice的限制］

****

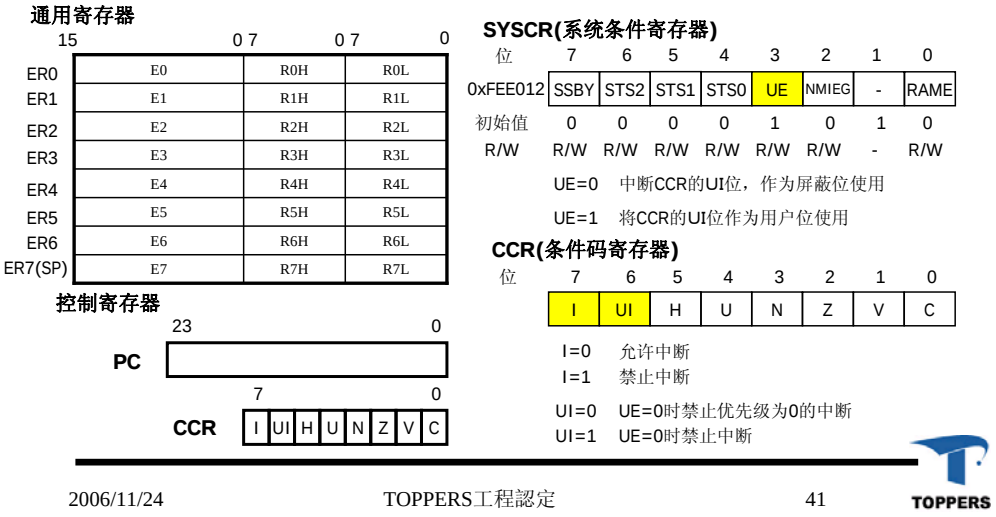
在此开发环境下无法实现用周期句柄访问netDevice上的端口。在TOPPERS/JSP的Windows版中可以用中断、周期句柄等访问端口。这是因为在Windows版中并不是在真正的中断、周期句柄内访问端口，而是把Windows的thread当作中断、周期句柄进行处理，实际上是通过COM用Windows的thread来访问话题烧水壶模拟器的。这样才实现了对端口的访问。但是，在netDevice中必须用H8的实际中断服务例程访问TINET的各任务及等待执行。这从非任务文本的属性上来看是不能的，请谅解。

****

JSP：H8/3069F版的中断控制1

H8/3069F的寄存器

- **H8/300H CPU中设置中断的允许、禁止的寄存器是SYSCR和CCR**



TOPPERS

想一想在TOPPERS/JSP中为了实现中断控制应该怎样设定H8/300H呢？在H8/300H中，普通CPU的控制相关的寄存器有“通用寄存器”和“控制寄存器”两种。控制寄存器中的I和UI位与中断相关。I位为0时允许中断、为1时禁止中断。UI位对系统状态寄存器的UE位的功能进行设定。UI位为0时，禁止UE=0优先级0的中断。UI位为1时，禁止UE=0优先级0和1的中断。任何情况下都不能禁止NMI。

JSP：H8/3069F版的中断控制2

H8-3069F的优先级设定

- 在JSP：H8-3069F版的实装中，用IPRA、B和CCR的I位、UI位设定中断和优先级

	位	7	6	5	4	3	2	1	0
IPRA	0xFEE018	IREQ0	IREQ1	IREQ2	IREQ4,5	WDT RFSH	ITU0	ITU1	ITU2
	初始值	0	0	0	0	0	0	0	0
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
	位	7	6	5	4	3	2	1	0
IPRB	0xFEE019	ITU3	ITU4	DMAC0,1	-	SCI0	SCI1	A/D	-
	初始值	0	0	0	0	0	0	0	0
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

优先级为0时相应的中断请求设定为优先级0（低），为1时设定为优先级1（高）

启动时IPRA,B通过JSP的初始化全部设定为0

SYSCR	CCR		可以执行的中断
UE位	I位	UI位	
1	0	—	允许所有的中断
	1	—	只允许NM1
0	0	—	允许所有的中断
	1	0	允许ＮＭＩ和优先级为1的中断
		1	只允许NM1

TOPPERS

IPRA和IPRB这两个寄存器进行优先级的设定。上述位为0时优先级是0、为1时优先级是1。可以根据中断原因的不同来设定位。在TOPPERS/JSP的H8版的1.4.1和1.4.2中，实装发生了变更。在1.4.2版中，打开电源后IPRA和IPRB的各位被设定为0。UE位除了导入时以外都固定为0。使用CCR寄存器的I和UI位来控制中断和中断的禁止许可，所以中断有高低两个优先级。

JSP：H8/3069F版的中断控制3
H8-3069F的优先级定义

- 可以对应CCR上能设定的值来设定3个中断级别。
  在JSP1.4.2中，此级别定义在h8.h引用文件中

  IPM_LEVEL0 许可所有的中断

  IPM_LEVEL1 许可NMI和优先级1的中断

  IPM_LEVEL2 只许可NMI

TOPPERS/JSP1.4.2中设定了3个中断级别。此设定定义在h8.h引用文件中。此级别可以通过CCR寄存器的设定来进行控制。

IPM_LEVEL0　许可所有中断
　　设定CCR寄存器的I位=0、UI位=0
IPM_LEVEL1　许可NMI和优先级1的中断
　　设定CCR寄存器的I位=1、UI位=0
IPM_LEVEL2　只许可NMI
　　设定CCR寄存器的I位=1、UI位=1

JSP：H8/3069F版的中断控制4

中断向量

- H8/300H设定了从0地址开始的256个字节的中断向量空间
- 从对应中断源的向量地址进入中断服务函数

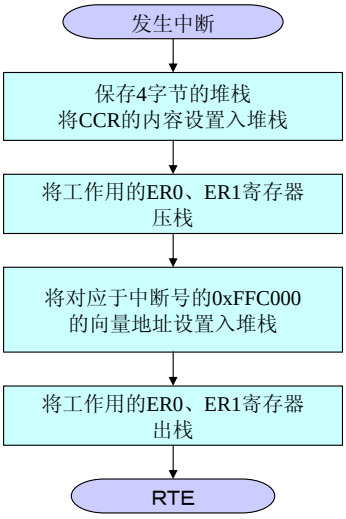
编号	因素	向量地址	IPR	备注
0	reset	0x000000		reset启动地址
7	NMI	0x00001C		
12	IRQ0	0x000030	IPRA7	
13	IRQ1	0x000034	IPRA6	作为按钮的中断使用
14	IRQ2	0x000038	IPRA5	作为壶盖的中断使用
15	IRQ3	0x00003C		
16	IRQ4	0x000040	IPRA4	
17	IRQ5	0x000044		IF_ED用中断
24	IMIA0	0x000060	IPRA2	系统定时器中断
25	IMIB0	0x000064		
26	OVI0	0x000068		
56	ERI1	0x0000E0	IPRB2	串行错误中断
57	RXI1	0x0000E4		串行接收中断
58	TXI1	0x0000E8		串行发送中断
59	TEI1	0x0000EC		

JSP中使用的中断向量

H8/300H分配了从地址0开始的256个字节的中断向量。一个向量是4个字节，所以可以定义64个向量。这64个向量中有未使用的向量。上表中的内容是TOPPERS/JSP的H8/3069F版的实装中使用的中断和其他重要的中断。IRQ1和IRQ2的中断是使用netDevice的虚拟设定。启动中断时必须给向量设定中断句柄的启动地址。而且，中断句柄必须用汇编程序的RTE命令结束。

JSP：H8/3069F版的中断控制5
ROM监视器模拟运行向量

- 下载程序的中断向量空间从**0xFFC000**地址开始设定
- **ROM**监视器在发生中断时置换成从**0xFFC000**地址开始的向量，并重新执行中断句柄

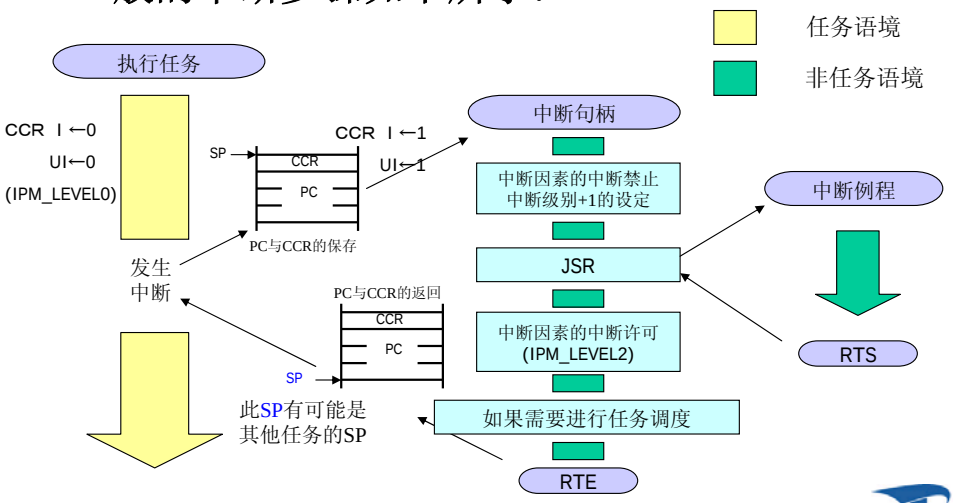


从0地址开始的384KB是FlashROM的空间，改写中断向量时需要改写FlashROM。用ROM监视器下载程序并执行时，中断向量被设定到从内置RAM的0xFFC000地址开始的256字节的空间上。在ROM监视器已写入到ROM中的状态下，如果发生中断就会读取对应该中断向量的从0xFFC000地址开始的向量地址，由于安装了切换到此地址并执行中断的功能，所以可以在下载的中断向量中进行中断的验证。

JSP：H8/3069F版的中断控制6

中断的步骤

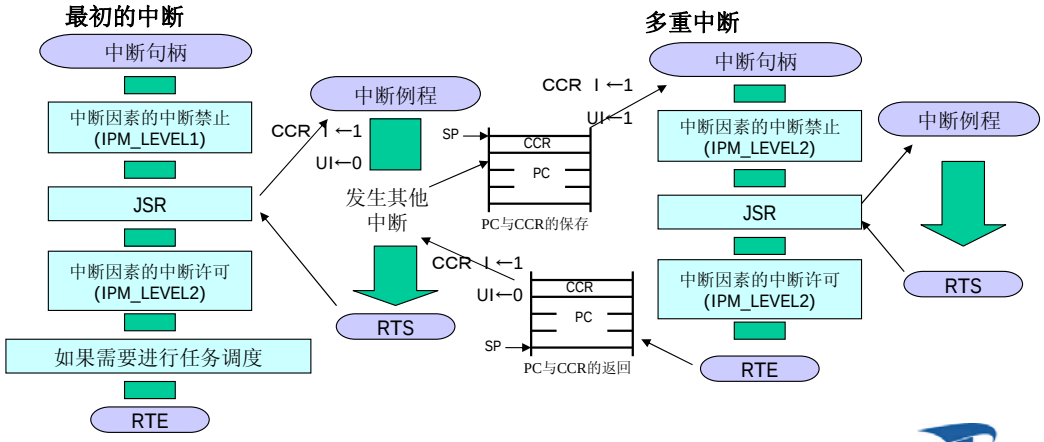
- 一般的中断步骤如下所示：



在任务语境中，是在CCR的I=0、UI=0（IPM__LEVEL0）的状态下执行。IPRA和IPRB需要根据中断的优先级进行设定。中断级别设定为IPM__LEVEL0时、IPRA、IPRB的对应位设定为0、IPM__LEVEL1时，BIT1进行设定。如果发生中断原因，在SP上保存CCR寄存器的值和中断结束后返回的PC地址。然后I位和UI位都设定1，并从中断向量的地址开始执行。中断向量设定了对应的中断句柄例程的开始地址，所以中断句柄会启动。之后对发生的中断对应的优先级进行＋1的中断设定。这样就不会发生自己的多重中断了。在此种状态下执行中断服务例程，执行后将I位和UI位设定为1禁止中断。在此之后如果有需要就进行任务调度。在不需要任务调度时，如果执行RTE命令就会从SP返回CCR寄存器和PC，并返回到任务执行。

JSP:H8/3069F版的多重中断
最初的中断为IPM_LEVEL0时

- 如果IPM_LEVEL0以普通级别发生，其中断例程就会以IPM_LEVEL1执行



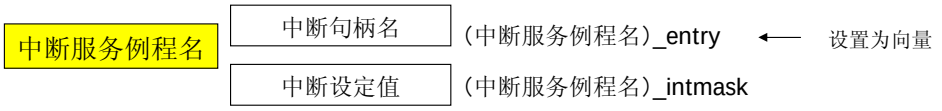
TOPPERS

最初发生的中断是IPM__LEVEL0时，中断例程以中断级别＋1即IPM__LEVEL1执行。由于此级别允许优先级1的中断，所以在中断例程执行过程中发生优先级1的中断原因时在此处理中会发生多重中断。关于此处理，除了不执行任务调度以外都与最初的中断相同。

JSP：H8/3069F版的中断句柄

中断句柄与中断许可禁止函数的命名规则

- 中断句柄按照**DEF_INH**服务调用的命名规则自动生成
- 中断时的级别设定值必须自己定义
- 级别设定值等于中断级别＋**1**

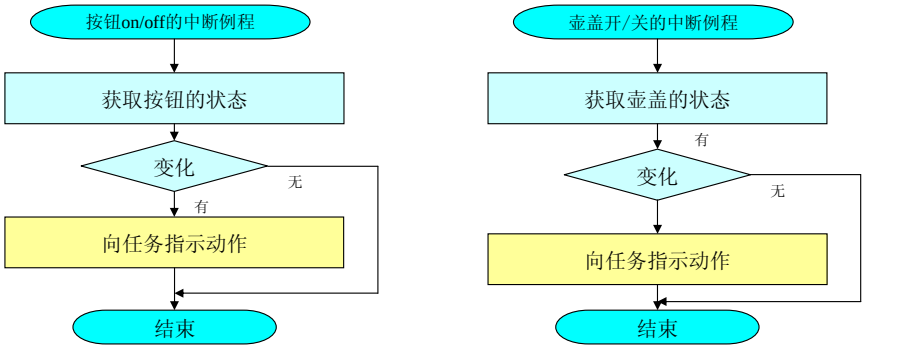


在TOPPERS／JSP1.4.2 H8版中，通过在配置文件中注册DEF_INH服务调用来自动生成中断句柄。句柄的名称按照“中断句柄的启动地址名=中断服务例程名”的命名规则来命名。例如，如果中断服务例程名是button_handler，句柄名称就是button_handler_entry。此函数由向量自动设定。（JSP1.4.1 H8版无此功能，必须自己改写向量的设定。）

有一点是H8的实装特有的，就是在发生对应的中断时只允许高于本身的上层中断，所以必须定义级别设定值。此设定值在中断句柄内自动设定。

使用中断的负荷改善
用中断对应两个任务

- 将硬件改良成通过按钮和壶盖的on、off来产生中断时，会将button_task置换成两个中断例程

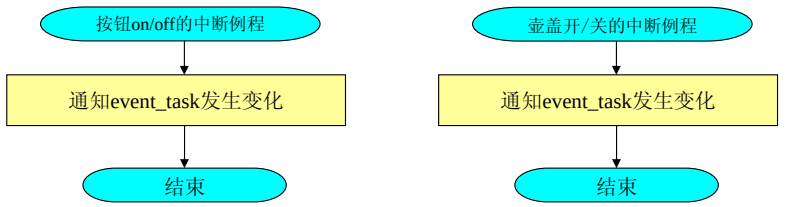


这是支持按钮和壶盖的on、off中断时的中断服务例程的实装例子。在中断开始时获取按钮或壶盖的状态并根据此状态向timer_task、heater_task发送事件。在本教材的netDevice中，在状态读取部分会发生端口的READ，所以无法对应。

使用netDevice改善

非任务语境下的任务等待？

- 实习中的话题烧水壶的按钮和壶盖分配在**netDevice**上
- 在**netDevice**中，不能从中断例程读取**netDevice**上的端口
- 所以改良成只对**button_task**进行中断化、只将按钮和壶盖的状态传递给**event_task**



TOPPERS

在netDevice上进行实装时，由于不能在中断时进行端口的READ，所以只将事件传递给event_task就可以吧！由于没有50ms周期的button_task，所以处理器的负荷得到改善。

话题烧水壶的中断对应

1. 制作话题烧水壶2
2. 评审
3. 提高实时性的方法
4. 任务负荷的验证和对应
5. 中断的对应
6. 总结

使用教材进行中断的实装吧！

按钮和壶盖的中断
设定IRQ1(按钮)、IRQ2(壶盖)

- **话题烧水壶模拟器H8-3069F对应版的设定：按钮的on、off中断发生在IRQ1上，壶盖的开闭中断发生在IRQ2上**
- **CPU获取这些中断时需要进行以下改造**

1）在配置文件中注册DEF_INH
2）创建中断服务例程
3）设定中断级别和级别设定值
4）在初始化例程中注册中断设定寄存器的初始化

在系统上实现话题烧水壶模拟器中对应的按钮的on、off中断（用IRQ1对应）和壶盖的开闭中断（IRQ2）的对应。步骤如下所示：

1）创建按钮中断和壶盖中断的中断服务例程。

2）在配置文件中追加两个中断的DEF_INH服务调用。

3）定义中断向量和级别设定值。

4）在初始化例程akih8pot_device.c的init_pot()上注册中断设定寄存器的初始化。

对JSP的中断注册

对配置文件的中断注册

- 通过**DEF_INH**服务调用在配置文件**(*.cfg)**中追加中断句柄的定义
- 按钮和壶盖的参数如下所示：

中断	中断句柄号	句柄属性	句柄启动地址
按钮	IRQ_EXT1 (13)	TA_HLNG	中断例程名
壶盖	IRQ_EXT2 (14)	TA_HLNG	中断例程名

TA_HLNG是高级语言用的接口中表示启动处理单位的对象属性

创建中断例程
从JSP：DEF_INH调用的程序

- 在DEF_INH定义的中断例程可以直接用C语言函数写
- 在pot1.h或pot2.h中追加中断例程的原型声明
- 追加pot1.c或pot2.c的直接中断例程

```
void button_int_handler(void)
{
  iset_flg(EVT_FLGID, EVT_BUTTON);
}
```


直接将中断服务例程记述到pot1.c或pot2.c中。

```
/*
 *按钮中断句柄
 */
void
button_int_handler(void)
{
        iset_flg(EVT_FLGID, EVT_BUTTON);
}
/*
 *壶盖中断句柄
 */
void
cover_int_handler(void)
{
        iset_flg(EVT_FLGID, EVT_COVER);
}
```

修改为Event_task获取EVT_BUTTON、EVT_COVER，通过get_swtch()函数在按钮和壶盖的状态下进行处理。不需要button_task。

［netDevice的限制］

****

用C语言记述中断禁止许可函数时，请记述在akih8port_device.c文件中。在实际的系统下是记述到./config/h8的文件中。

****

设定中断级别和级别设定值
设定为IPM_LEVEL0

- 将BUTTON和COVER的中断设定为IPM_LEVEL0
- 因此级别设定值是IPM_LEVEL1
- 在akih8pot_device.h中追加设定
- 中断设定写在akih8pot_device.c中

```
#define（BUTTON中断服务例程名）_intmask IPM_LEVEL1
#define（COVER中断服务例程名）_intmask IPM_LEVEL1
```


在akih8pot_device.h中定义中断级别和级别设定值。IPRA寄存器在初始化时设定为零，所以不用变更也不用定义。

追加到akih8pot_device.c的initial_pot中。如上所述，由于默认值是级别0，所以并不一定要追加。

```
#define POT_DEF_BITS    ((1<<H8IPR_IRQ1_BIT) | (1<<H8IPR_IRQ2_BIT))

void
Initial_pot()
{

    /*设定中断请求时用的优先级别 。*/
   sil_wrb_mem((VP)H8IPRA, (sil_reb_mem((VP)H8IPRA) & ~POT_DEF_BITS));

}
```

中断控制寄存器的设定1
IER和ISCR的设定

- **H8/300H的IRQ中断控制寄存器有以下3个**

ISCR(IRQ检测控制寄存器)

位	7	6	5	4	3	2	1	0
0xFEE014	-	-	IRQ5SC	IRQ4SC	IRQ3SC	IRQ2SC	IRQ1SC	IRQ0SC
初始值	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

IRQnSC
0：发生输入的Low级别中断
1：以输入的下降沿发生中断

IER(IRQ启动寄存器)

位	7	6	5	4	3	2	1	0
0xFEE015	-	-	IRQ5E	IRQ4E	IRQ3E	IRQ2E	IRQ1E	IRQ0E
初始值	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

IRQnE
0：禁止中断
1：允许中断

ISR(IRQ状态寄存器)

位	7	6	5	4	3	2	1	0
0xFEE016	-	-	IRQ5F	IRQ4F	IRQ3F	IRQ2F	IRQ1F	IRQ0F
初始值	0	0	0	0	0	0	0	0
R/W	-	-	R/W	R/W	R/W	R/W	R/W	R/W

IRQnF
中断发生时设置为1
读取后通过写入0来清除
不进行参照时不必清除

2006/11/24 TOPPERS工程認定 56 TOPPERS

H8/300H中进行中断设定的寄存器有以下3个：

1）ISCR(IRQ检测控制寄存器)
中断原因的输入是选择Low级别（=0）或下降沿（=1）
这里请选择下降沿。

2）IER(IRQ启动寄存器)
根据中断原因设定中断禁止（=0）、中断允许（=1）。
通电后默认为禁止（=0），所以请设定为许可（=1）。

3）ISR（IRQ状态寄存器）
参照中断状态。如果发生中断，就会一直保持此状态，直到复位。
因为不需要参照，所以此寄存器不用设定。
（由于在话题烧水壶的模拟机上没有对应，所以总是返回零）

中断控制寄存器的设定2
IER和ISCR的设定上的注意事项

- **IREQ1和IREQ2是netDevice的虚拟中断，如果在实际的硬件的IER上设定中断，IREQ1、2信号的状态就会发生错误的运行**
- **IER和ISCR请用akih8pot_device.c中SIL的记述进行设定**

SIL的记述例子

```
sil_wrb_mem((VP)H8IER, sil_reb_mem((VP) H8IER)|(1<<H8IER_IRQ1E_BIT));
```

记述在config/h8/akih8_3069f/h8_3069f.h中

注意：不用自己创建这些声明，使用系统中设定的声明。（3048和3069地址是不同的，但名称相同）

在本实习中，请改写akih8pot_device.c的源代码来设定IER和ISCR端口。在实际的系统中，是在./config/h8下的源代码文件中进行设定的。

```
#define         POT_INT_BITS  ((1<<H8IER_IRQ1E_BIT) | (1<<H8IER_IRQ2E_BIT))
void
Initial_pot()
{

   sil_wrb_mem((VP)H8ISCR, sil_reb_mem((VP)H8ISCR) | POT_INT_BITS);
   sil_wrb_mem((VP)H8IER, sil_reb_mem((VP)H8IER) | POT_INT_BITS);

}
```

［netDevice的限制 ］

****

本讲座的实装是组合实机和netDevice实现的。设备驱动器的源代码中有以下对应：

1）akih8pot_device.c：集合了netDevice用的驱动

2）akih8lcd_device.c：集合了实机扩展板用的驱动

这两个源文件的不同在于akih8pot_device.c引用<monsil.h>、而akih8lcd_device.c引用<sil.h>。sil.h在TOPPERS/JSP中的串行设备上也进行了使用，是记述了用SIL接口访问硬件的设定的引用文件。与此相对，<monsil.h>记述的是用SIL接口通过任务监视器访问硬件。如果netDevice与服务器已建立连接，任务监视器则不访问硬件，而是对服务器进行访问。所以，需要在<monsil.h>引用文件下的akih8pot_device.c中记述IER和ISCR的访问内容。

SIL有什么优点呢？如果已经有人制作了话题烧水壶样机的硬件部分，那么只要将sil.h的内容改为monsil.h，而不需要修改源代码，就能在实机上运行大家创建的话题烧水壶样机的程序。

在初级实装讲座的最后部分对SIL进行了说明，请参考。

****

最后提醒
中断时不能访问**netDevice**的端口

- 在进入工作前，不能从中断例程读取**netDevice**上的按钮和壶盖的端口
- 读出后，不会查找**netDevice**，而是直接读取**H8/3069F**扩展板上的端口数据（基本都是**1**），所以无法正常查找
- 请注意！

总结

1. 制作话题烧水壶2
2. 评审
3. 提高实时性的方法
4. 任务负荷的验证和对应
5. 中断的对应
6. 总结

关于嵌入式开发环境
调试方法和正确的单体测试

- 调试方法
  - ▼通过**syslog**语句、**printf**语句显示**LOG**
    在**LINUX**、**Windows**上的交叉环境下，正确使用模拟开发和单体测试的**printf**语句
  - ▼使用**remote**调试器调试
  - ▼使用**JTAG-ICE**调试
- 设备驱动的调试
  - ▼使用**voltaile**声明
  - ▼使用**SIL**等标准接口
    对模拟和测试也很有效

实时・调度理论

RMS和harmonic task set

- **DMS/RMS（Deadline Monotonic Scheduling）**
  周期越短的任务分配的优先级越高
- **Harmonic task set**
  各任务的周期比率为倍数关系
  有效发挥处理器的使用率
- 有效利用周期句柄、中断
  任务以上的优先级、实时性的提高
- 要将理论有效活用在实时系统的设计上，存在着很大的障碍
  ⇒ 通过开发得到的经验是宝贵的技术

TOPPERS/JSP的标准输入输出
newlib的使用

- 在本教材中，任务监视器支持**printf**语句等标准输入输出功能（任务监视器的输入输出依照**stdio.h**）。
- 此程序执行的是**monitor**目录下的源代码。

  stdio.h　printf.c　sprintf.c　scan.c　sscan.c

- **TOPPERS/JSP**使用的**newlib**标准输入输出函数只要改写**newlib**的部分源代码就能直接使用。在**monitor/README.txt**中追加了对应的方法，所以如果有兴趣也请挑战**newlib**版教材。

教材的编写
参考资料

- 向在教材的编写上给与协助的各位表示感谢!

  Advanced・Data・Controls股份有限公司　森田浩

- 参考资料

「豊橋市中小企業技術者研修
リアルタイムOSを活用した組込みソフトウェア設計」
名古屋大学　高田広章
「4th Open SESSAME テキスト」
組込みソフトウェア管理者・技術者育成研究会
「 H8/300Hシリーズ　プログラミングマニュアル」
ルネサステクノロジ
「H8/3069R F-ZTATハードウェアマニュアル」
ルネサステクノロジ

*参考资料中文名（意译）

《丰桥市中小企业技术人员研修
活用实时OS的嵌入式软件设计》
名古屋大学　高田广章

《4th Open SESSAME 教材》
嵌入式软件管理者・技术人员培训研究会

《H8/300H系列程序设计手册》
瑞萨科技

《H8/3069R F-ZTAT硬件手册》
瑞萨科技