# 高级操作系统

## 3. 分布式进程与处理机

### 3.5 实时分布式系统

熊焰，yxiong@ustc.edu.cn
黄文超, huangwc@ustc.edu.cn
http://staff.ustc.edu.cn/~huangwc/advancedos

参考书目：《分布式操作系统》，《分布式系统原理与范型》

# 本章内容

1. 分布式系统模型
2. 分布式处理机分配
3. 分布式进程调度
4. 分布式系统容错
5. **实时分布式系统**

# 5 实时分布式系统

## 5.1 定义

- 当某个激励出现时，系统必须以某种方式在限定的时间内对它做出响应。如果响应正确完成，但却超出了限定的时间，那么，系统也被认为是执行失败。

  - 在实时系统中，产生结果所花费的时间与产生正确的结果同样重要。

  - 例： CD播放器、控制飞机副翼和其它部件的计算机、机动车子系统、控制反坦克导弹的军用计算机、航空交通控制系统、粒子加速器、自动化工厂、电话交换机、机器人、医院的重症病房、CT扫描仪、自动股票交易系统

除了容错分布式系统以外，还存在着另外一种分布式系统---实时分布式系统。有时候，我们可以把二者结合在一起，形成容错实时分布式系统。

对大多数程序来说，正确性只与指令执行的逻辑顺序有关，而与指令执行的时间无关。如果一个C程序能够在主频为200MHz的工作站上正确地运行，那么它也一定能够在一个主频为3.77MHz的8088个人计算机上正确运行，只不过运行结果出来慢一点而已。

但是，当实时系统与外界进行交互式操作时，时间却显得十分重要。当某个激励出现时，系统必须以某种方式在限定的时间内对它做出响应。如果响应正确完成，但却超出了限定的时间，那么，系统也被认为是失败。因此，在实时系统中，产生结果所花费的时间与产生正确的结果同样重要。

例如，一个CD音乐播放器的CPU从光盘上读取声音压缩数据并把它们还原成音乐。假设该CPU的速度刚好能够完成这项音乐还原工作。

如果我们用一个速度只有前面CPU三分之一的CPU来播放音乐。那么，音乐听起来让人难以忍受。原因是它不仅要求CPU能够正确地还原音乐，而且还对CPU还原音乐的时间有着严格的限制。

在我们现实生活中，存在着许多实时应用。例如，嵌在电视机和录像机中的计算机、控制飞机副翼和其它部件的计算机、计算机控制的机动车子系统、控制反坦克导弹的军用计算机、计算机航空交通控制系统、粒子加速器、自动化工厂、电话交换机、机器人、医院的重症病房、CT扫描仪、自动股票交易系统等等。

# 5 实时分布式系统
# 5.1 定义

- 很多实时应用系统的结构化程度要比普通分布式系统高，例如：
  - 外部设备给计算机产生了一个激励
  - 计算机必须在限制的时间内完成相应的处理
  - 处理结束后，系统处于空闲状态，直到下一个激励到来

很多实时应用系统的结构化程度要比普通分布式系统高。典型的实时应用系统是一个外部设备给计算机产生了一个激励，计算机必须在限制的时间内完成相应的处理。处理结束后，系统处于空闲状态，直到下一个激励到来。

在通常情况下，激励是周期性的，即每隔ΔT秒就产生一次激励，例如嵌入在电视机和 录像机中的计算机，每1/25秒接收一帧图像。在某些情况下，激励是非周期的，这就意味着激励是不断发生的，但不是周期的。例如，航空交通控制系统控制空中飞机的飞行。此外，还有些激励是突发的即无法预料的，例如设备过热。

# 5 实时分布式系统

## 5.1 定义

- 周期性激励
  - 每隔ΔT秒就产生一次激励
  - 如：图像帧
- 非周期性激励
  - 不断发生，但非周期
  - 如：飞机飞行
- 突发的（无法预料）
  - 如：设备过热

在通常情况下，激励是周期性的，即每隔ΔT秒就产生一次激励，例如嵌入在电视机和录像机中的计算机，每1/25秒接收一帧图像。在某些情况下，激励是非周期的，这就意味着激励是不断发生的，但不是周期的。例如，航空交通控制系统控制空中飞机的飞行。此外，还有些激励是突发的即无法预料的，例如设备过热。

# 5 实时分布式系统

## 5.1 定义

- 在一个大型周期性系统中，情况比较复杂，存在着各种各样类型的事件

- **一台机器**要满足所有的时间限制以及其它的实时限制是**非常困难**的 （如：自动流水线工厂）

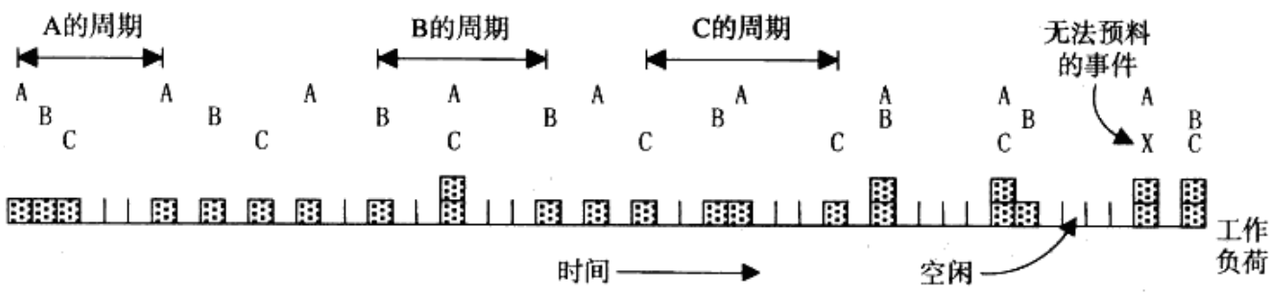



图 4.23　三个事件流加上一个突发事件的叠加情况

在一个大型周期性系统中，情况比较复杂，存在着各种各样类型的事件。例如，视频输入、音频输入以及马达驱动管理等，它们都是周期产生的，必须周期地加以处理。图4.23描述了三个周期性事件流A、B和C，还有一个突发事件X。

随着应用规模变得越来越大越来越复杂（例如，拥有上千个机器人的自动流水线工厂），一台机器要满足所有的时间限制以及其它的实时限制是非常困难的。

# 5 实时分布式系统
# 5.1 定义

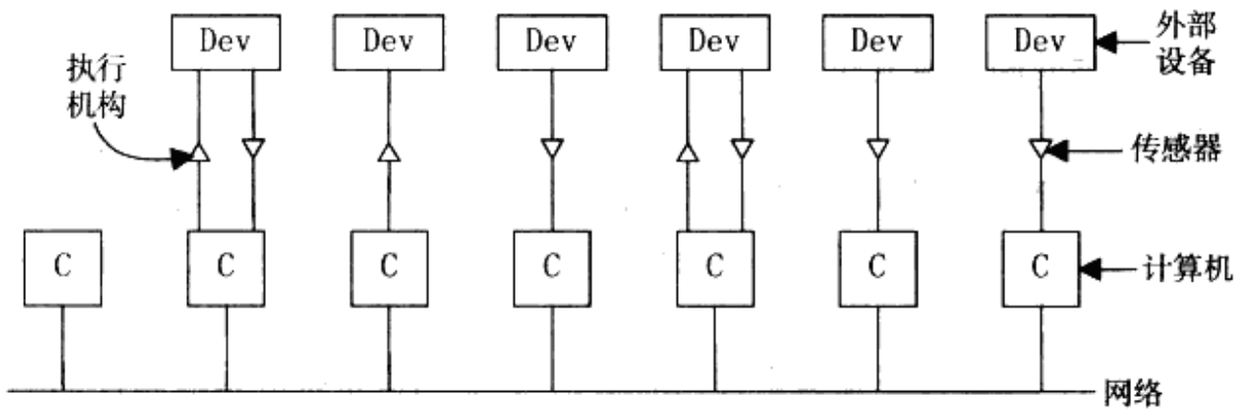
- 一种解决方案：
  - 在每一个实时设备前配备一个专用的微处理机
    - 接收实时设备输出
    - 将处理结果及时传送给实时设备
  - 实现了**对所有实时设备**的**实时响应**，因而产生了一个**分布式实时系统**

系统设计者在每一个实时设备之前配备一个专用的微处理机，用于在实时设备输出时及时进行接收并将处理结果及时传送给实时设备。这样实现了对所有实时设备的实时响应，因而产生了一个分布式实时系统，并形成了自己独有的特点和风格（例如，实时通讯）。

# 5 实时分布式系统

## 5.1 定义

- 分布式实时计算机系统




分布式实时系统通常可以按照图*的结构来构造。我们可以看到它是一个用网络连接起来的多计算机系统。其中一部分计算机与外设相连，这些计算机主要的任务就是实时控制即接收外设的激励并将响应结果传送给外设。这些计算机可能是嵌入到设备中的微型控制器或者是独立的机器。它们都有从外设接收信号的传感器和向外设发送信号的执行机构。这些传感器和执行机构可以是数字式或模拟式的。

# 5 实时分布式系统
# 5.1 定义

- 根据实时性的限制和后果，实时系统通常可以分为两类
  - 软实时系统：系统对激励的响应可以偶尔超过时间限制
  - 硬实时系统：不允许任何一次响应超过时间的限制，因为它有可能造成死亡或巨大的灾难。

根据实时性的限制和后果，实时系统通常可以分为两类：
软实时系统：系统对激励的响应可以偶尔超过时间限制。例如，电话交换机允许在超载的情况下，在105次电话中允许1次断线或串线。
硬实时系统：不允许任何一次响应超过时间的限制，因为它有可能造成死亡或巨大的灾难。实际上，也存在着处于两者之间的系统，即当超过一个时间限制时，系统立即结束当前的处理，因为它不会造成任何致命的伤害。例如，一个苏打水瓶子在传送带上走过了喷口并没有苏打水喷进去，由于继续向它喷苏打水已无意义，所以，必须立即结束喷苏打水。这种情况并不产生什么危害。
　还有一些系统中某一些子系统是硬实时系统，而另一些子系统是软实时系统。

# 5 实时分布式系统
# 5.1 定义

- 典型的认知**错误**：
  - 实时系统是用汇编语言编写的驱动程序
  - 实时计算是快速计算
  - 高速计算机取代了实时系统

虽然实时系统已经发展了很多年了，人们对它也积累了大量的认识和经验，但很多的认识是片面错误的。其中最严重的错误如下：

错误1-实时系统是用汇编语言编写的驱动程序：这在七十年代可能是正确的，因为那时的实时系统只是将少数几个外设连接到小型机上。但现在的实时系统是非常复杂的，用汇编语言来编写驱动程序已经无法满足复杂性的要求。目前，编写驱动程序已不再是实时系统设计者最关心的问题。

错误2-实时计算是快速计算：实时计算并不一定是快速计算。例如，一个计算机控制的天文望远镜必须实时地跟踪恒星和星系，但实际上，地球每一小时才旋转15弧度，并不特别快。在这里，准确性却是最重要的。

错误3-高速计算机取代了实时系统：高速计算机只是能够构造性能更高的实时系统而不是取而代之。例如，心脏病专家希望能够使用一台MRI扫描仪来实时显示病人的心脏跳动。而当他们实现这个目标时，他们又会要求能够显示三维、彩色、可缩放的图形。这就需要高速计算机来完成。如果采用多处理机系统，又会带来一些必须解决的通信、同步以及调度等方面的问题。

# 5 实时分布式系统
# 5.2 设计问题

- 时钟同步问题 （已讲）
- 事件触发系统和时间触发系统
- 预知性
- 容错性
- 语言支持

时钟同步问题：这是时间本身的维护问题。在多个处理机情况下，每一个处理机都有它自己本地的时钟，保持所有时钟的同步便成为一个非常重要的设计问题。我们已经在第二章中讨论了时钟同步问题，这里就不再重复了。

# 5 实时分布式系统
# 5.2 设计问题

- 事件触发系统：低负载时会更快响应
  - 但是，许多事件一次性发生时。。。
    - 如核反应堆的一个管道破裂
- 时间触发系统： 系统不会过载、且易于统筹控制
  - 但是，中断时间间隔ΔT必须仔细地选择

事件触发和时间触发：在一个事件触发(event-triggered)的实时系统中，当一个重要的外部事件发生时，它就被传感器接收到，然后，传感器就给与之相连的处理机发一个中断请求信号。因此，事件触发系统实际上是由中断来驱动的。目前，大多数实时系统都是由中断驱动的事件触发系统。对于计算能力较强的软实时系统来说，中断驱动的方法实现起来比较简单，并且由于其工作良好，所以，一直得到广泛的应用。对于复杂的实时系统来说，只要编译器能够识别中断驱动程序，并且系统知道一个事件发生后如何进行响应，那么，中断驱动的方法也能很好地适应复杂的实时系统。

能够解决这个问题的另一个设计方案就是设计一个时间触发的实时系统。在这种系统中，每隔ΔT毫秒时间就会产生一个时钟中断。每一次时钟中断时，都对传感器进行采样，并驱动相应的执行机构。中断只是在若干时钟滴答时发生。

当管道破裂时，系统在事件发生后的第一个时钟中断时就会被发现。但中断并不会由于多个报警事件的发生而增加，因而系统中断不会超载。在发生严重问题时，系统仍然可以正常工作。

显然，中断时间间隔ΔT必须仔细地选择。如果太小，则系统就会产生很多的时钟中断，浪费了大量时间来响应中断。如果太大，一些严重问题可能在发现的时候已经太迟了。此外，系统还必须决定在每一次时钟中断时应该检查那些传感器。需要注意的是有一些事件的持续时间可能比时钟中断间隔要短，必须将这些事件保存起来，以防止他们被忽略。它们可以存储在锁存器中或外部设备的微处理机中。

我们可以通过一个例子来说明上述两种方法的不同之处，假定一个电梯控制器控制100层楼的电梯。电梯在第60层上等待顾客使用。这时一个顾客在下一层按下了按钮。在100毫秒之后，另一个顾客在100层上按了按钮。在事件触发系统中，第一次按钮产生了一个中断，于是电梯开始启动下降。紧接着第二次按钮中断发生并被记录下来以后再处理，但电梯仍然继续下降。

而在时间触发系统中，它每隔500毫秒采样一次。如果两次按钮中断都是在同一个采样周期内出现，那么，控制器就必须作出一个决定。如果按照最新顾客最先服务原则，那么它就要向上移动。

# 5 实时分布式系统
# 5.2 设计问题

- 预知性
  - 设计时就应该保证系统能够满足所有事件的时间限制
    - 有一些事件之间可能会有难以预见的联系，如
      - 管道破裂例子中温度、压力与放射性警告之间的联系
    - 多用户访问共享文件（或数据库）的事件并非独立

预知性：实时系统的一个最重要特性就是系统中的事件是可预见的。在理想情况下，设计时就应该保证系统能够满足所有事件的时间限制，包括最大负载时的时间限制。在事件独立的假定下进行系统事件统计分析将会产生一些问题。因为，有一些事件之间可能会有难以预见的联系，正如上面那个管道破裂例子中温度、压力与放射性警告之间的联系一样。

# 5 实时分布式系统
# 5.2 设计问题

- 容错性
  - 如：交通、医院、发电厂等安全性要求较高的场合
  - 常见方法：
    - 主动复制
    - 主备份（很少使用）
    - “领导者”做决定，其余机器按吩咐去工作

容错性：实时系统大都应用在交通、医院、发电厂等安全性要求较高的场合，所以，容错是非常重要的。一般采用主动复制方法来容错，但它只适用于采用非扩展协议的实时系统，这种非扩展协议能够使所有的进程在任何时刻任何事件上达成协作一致。而主备份方法却很少使用，主要原因是主服务器崩溃后，在主服务器和备份服务器的切换期间可能会超过实时系统中的时间限制。一个综合方法就是一台机器作所有的决定，所有其它机器均按照决定去做，一有命令就立即执行。

# 5 实时分布式系统
# 5.2 设计问题

- 容错性
  - 针对“最坏情况”的处理，如
    - 三个复制部件同时实效
    - 停电事件～电话系统
      - 实时系统达到最大负载通常是发生在系统中大量设备失败的时候
      - 容错实时系统必须能够同时处理最多设备出现失败以及负载达到最大的情况

在一个安全性要求非常高的实时系统中，系统应该能够处理最坏情况。不能因为三个部件同时出错的可能性是很小而将其忽略。原因是错误不是独立出现的。例如，在一次突发的断电事故中，每一个人都拨打电话，这就可能导致电话系统过载。此外，实时系统达到最大负载通常是发生在系统中大量设备失败的时候,因为系统中大多数通信故障都是由设备报告失败产生的。因此，容错实时系统必须能够同时处理最多设备出现失败以及负载达到最大的情况。

# 5 实时分布式系统
# 5.2 设计问题

- 有些实时系统可以在发生严重错误的时候，强制停止运行
  - 例如，当一个铁路信号灯突然不亮了，控制系统就必须通知所有的火车立即停止行驶。如果所有的火车总能保持足够远的距离，并且所有行驶的火车都能同时开始刹车，那么，灾难就可以避免，并且系统也可以在来电之后恢复正常。
- 对于这样一个能够停止操作并能避免危险的系统，我们将其称之为**失败-安全系统**。

有些实时系统可以在发生严重错误的时候，强制停止运行。例如，当一个铁路信号灯突然不亮了，控制系统就必须通知所有的火车立即停止行驶。如果所有的火车总能保持足够远的距离，并且所有行驶的火车都能同时开始刹车，那么，灾难就可以避免，并且系统也可以在来电之后恢复正常。对于这样一个能够停止操作并能避免危险的系统，我们将其称之为失败-安全系统。

# 5 实时分布式系统
# 5.2 设计问题

- 语言支持
  - 专用实时系统编程语言应该能够在**编译**的时候，计算出**每一个任务的最大执行时间**。
    - 不支持通常的while循环语句，
    - 支持常量参数限制的for循环语句
    - 不允许递归使用for循环
    - **但是**，仍难以计算，如高速缓冲未命中、页面错误以及DMA通道周期窃取

语言支持：尽管许多实时系统和应用程序都是用通用的编程语言（例如C语言）编写的，但是，如果使用专用的实时语言来编写将更为有效。

专用实时系统编程语言应该能够在编译的时候，计算出每一个任务的最大执行时间。这就意味着语言不支持通常的while循环语句，必须使用常量参数限制的for循环语句。此外，它还不允许递归使用for循环。即使有了这些限制也还无法事先计算每一个任务的执行时间，这是由于高速缓冲未命中、页面错误以及DMA通道周期窃取等突发事件所造成的。

# 5 实时分布式系统
# 5.2 设计问题

- 语言支持
  - 需要一种方法处理自己的时间
    - 变量clock, 保存当前时间（注意单位）

**表 4.3　32 位时钟在不同分辨率下的有效范围（溢出前）**

| 时钟分辨率 | 范围 |
|---|---|
| 1 纳秒 | 4 秒 |
| 1 微妙 | 72 分钟 |
| 1 毫秒 | 50 天 |
| 1 秒 | 136 年 |

实时系统编程语言需要一种处理自身时间的方法。也就是说它需要一个特殊变量clock来保存当前的时间（即时间片的数量）。一般，clock应为64位，1纳秒的时间片。

# 5 实时分布式系统
# 5.2 设计问题

- 语言支持
  - 需要一种方法处理自己的时间
    - 表达最小和最大延迟
    - 超时的处理
    - 周期性事件的处理，如下面语句：
      - every (25 msec) {…}

实时系统的编程语言还应该有一种表示最小和最大延迟的方法。例如，在Ada®语言中，有一个延迟语句，限制进程延迟的最小时间。但是，实际延迟可能因为没有界限而大得多。因此，无法给出延迟的上限或一个进程执行的时间间隔。

必须有一个方法来说明当预期的事件未发生时系统应该如何处理。例如，如果一个进程占用信号量的时间超过了限制，那么，应该停止占用并释放信号量。同样，如果一个消息发出之后在指定的时间限制内没有得到应答，那么，消息发送者应能够在k毫秒之后采取相应措施。

实时系统中的编程语言中应包含下列形式的语句：

every(25 msec) { … }

这个语句表示大括号内的代码每隔25毫秒被执行一次。

编译器就能计算出每个语句CPU时间的百分比，并根据这些数据计算出运行整个程序所需的最小机器数及这些进程如何分配给各个机器。

# 5 实时分布式系统
# 5.3 实时通信

- 实时分布式系统中最为重要的特性
  - 预知性和确定性
    - 处理机之间的通信是可预知的
      - 以太网：无法预知数据包传输时间上界
      - 令牌环局域网：可获得传输时间上界
      - TDMA： 通讯有固定带宽

实时分布式系统中的通信与其它分布式系统中的通信并不相同。预知性和确定性是实时系统中最为重要的特性。

预知性意味着处理机之间的通信是可预知的。局域网采用随机协议（例如，以太网），当一台机器在以太网上发送一个数据包时，它有可能与其它一台或多台机器发生冲突。冲突机器都会等待一个随机的时间之后，再重新发送，并可能一直冲突下去。因此，系统无法预先得知数据包传输时间的上界。

与以太网相反的是一个令牌环局域网。每当一个处理机有一个数据包要发送，它就等待循环的令牌传到自己这儿，得到令牌后，发送数据包，然后，将令牌沿环发送给下一个邻居。假设，令牌环网上有k台机器，每一台机器在得到令牌时最多只发送n个字节的数据包。这就能保证一个数据包能在kn个字节的传输时间内到达任意一台机器。这就是实时系统所需要的传输时间上界。

除了令牌环网方法以外，还有一个方法就是TDMA(时分多路访问)。通信是以固定长度的时间帧来组织的，每一个帧具有n个时间片。每一个时间片都分配给一个处理机。只有处理机对应的时间片到来时它才能传送自己的数据包。这种方法可以避免冲突，延迟也是有限。每一个处理机在每个帧中都被分配了一定数目的时间片，因而，通讯具有固定的带宽。

# 5 实时分布式系统
# 5.3 实时通信

- 包丢失率
  - 不可行的方法：超时重传
  - 一种解决办法：每个数据包被发送2次或多次
    - 虽然，浪费带宽
    - 但大大降低丢包率，从而减少延迟

实时分布式系统中的一个问题是包丢失率。一般处理包丢失问题的方法是在发送一个数据包后启动一个定时器。如果定时器在收到确认消息之前超时，那么，这个数据包将被重新发送一次。在实时系统中，这种较大的传输延迟是不能接受的。

一个简单的解决办法是每个数据包至少被发送两次或多次。尽管这种方法浪费了至少一半的带宽，但若丢失率为十万个数据包中只有一个数据包丢失，那么，采用每一个数据包被 传输二次,就使得每10^10个数据包中只丢失两个数据包。如果传输一个数据包需要1毫秒，那么，每四个月才丢失一个数据包。若每一个数据包被传输三次，那么，每30,000年才会丢失一个数据包。这种具有多副本的数据包传输，延迟小且有限。

时间触发协议：由于实时分布式系统的限制，使得它们所使用的协议与众不同。时间触发协议TTP是Kopetz 和 Grunsteidl在1994年提出的，它是应用在MARS实时系统上的(Kopetz 等 人在1989年提出的)，所以，为了更好地描述TTP协议，有必要先介绍MARS实时系统的一些特性。

MARS中的一个节点至少包含一个处理机，但通常有二、三个处理机同时工作，对外表现为一个能够处理Fail-silent错误的节点。在MARS系统中，所有节点都是通过两个可靠且独立的TDMA广播网络连接在一起的。所有的数据包都并行地在这两个网络中传送。数据包的丢失率是每三千万年丢失一个数据包。

# 5 实时分布式系统
# 5.3 实时通信

- 时间触发协议

  - Hermann Kopetz, Günter Grünsteidl: **TTP** - A Protocol for Fault-Tolerant Real-Time Systems. IEEE Computer 27(1): 14-23 (1994), 被引用次数：**848**

  - Hermann Kopetz, Andreas Damm, Christian Koza, Marco Mulazzani, Wolfgang Schwabl, Christoph Senft, Ralph Zainlinger: Distributed fault-tolerant real-time systems: the **Mars** approach. IEEE Micro 9(1): 25-40 (1989). 被引用次数：763

时间触发协议：由于实时分布式系统的限制，使得它们所使用的协议与众不同。时间触发协议TTP是Kopetz 和 Grunsteidl在1994年提出的，它是应用在MARS实时系统上的(Kopetz 等人在1989年提出的)，所以，为了更好地描述TTP协议，有必要先介绍MARS实时系统的一些特性。

# 5 实时分布式系统
# 5.3 实时通信

- 一个节点至少包含一个处理机
  - 通常有二、三个处理机同时工作，对外表现为一个能够处理Fail-silent错误的节点
- 所有节点都是通过**两个可靠且独立**的TDMA广播网络连接在一起的
  - 所有的数据包都并行地在这两个网络中传送。数据包的丢失率是每三千万年丢失一个数据包

MARS中的一个节点至少包含一个处理机，但通常有二、三个处理机同时工作，对外表现为一个能够处理Fail-silent错误的节点。在MARS系统中，所有节点都是通过两个可靠且独立的TDMA广播网络连接在一起的。所有的数据包都并行地在这两个网络中传送。数据包的丢失率是每三千万年丢失一个数据包。

# 5 实时分布式系统
# 5.3 实时通信

- TTP假定所有的**时钟**脉冲总是能够在**数十微秒**的数量级上保持**精确同步**
- MARS系统中所有节点都知道其它节点上正在运行的程序
- 任意节点都知道其它节点在什么时候发送一个数据包

MARS是一个时间触发系统，TTP假定所有的时钟脉冲总是能够在数十微秒的数量级上保持精确同步。这是因为协议本身能够提供一个连续的时钟同步，并且其硬件实现精度极高，所以，上述的同步精度是能够达到的。

MARS系统中所有节点都知道其它节点上正在运行的程序并且任意节点都知道其它节点在什么时候发送一个数据包。此外，数据包存在与否很容易被检测。由于数据包都被假定是不会丢失的，所以，当一个节点应接收到的数据包未到达时，它就意味着发送数据包的节点崩溃了。

# 5 实时分布式系统
# 5.3 实时通信

- 每个节点都保留系统的全局状态
  - 当前模式
  - 全局时间
  - 当前系统成员位图
- TTP协议包括单独的一个层来处理点对点的数据传输、时钟同步以及成员管理。

每一个节点都保留了系统的全局状态。这些状态在任何地方都是完全相同的。如果某一个节点与其它节点步调不一致，那么，这将产生一个严重的可检测错误。
全局状态包括三个域：
当前模式：由应用程序定义的，并与系统所处的阶段相关。例如，在航天管理程序中，倒计数、发射、飞行以及着陆等都是独立的模式。每一个模式都有它自己的一组进程和这组进程的运行顺序、参加运行的节点列表、TDMA时间片分配、消息名称和格式以及合法的后继模式。
全局时间：它也是由应用程序定义的，但在任何情况下，都必须足够的大，以便所有节点都能达成一致。
当前系统成员的位图：记录节点的动态增删。

TTP协议包括单独的一个层来处理点对点的数据传输、时钟同步以及成员管理。

# 5 实时分布式系统

## 5.3 实时通信

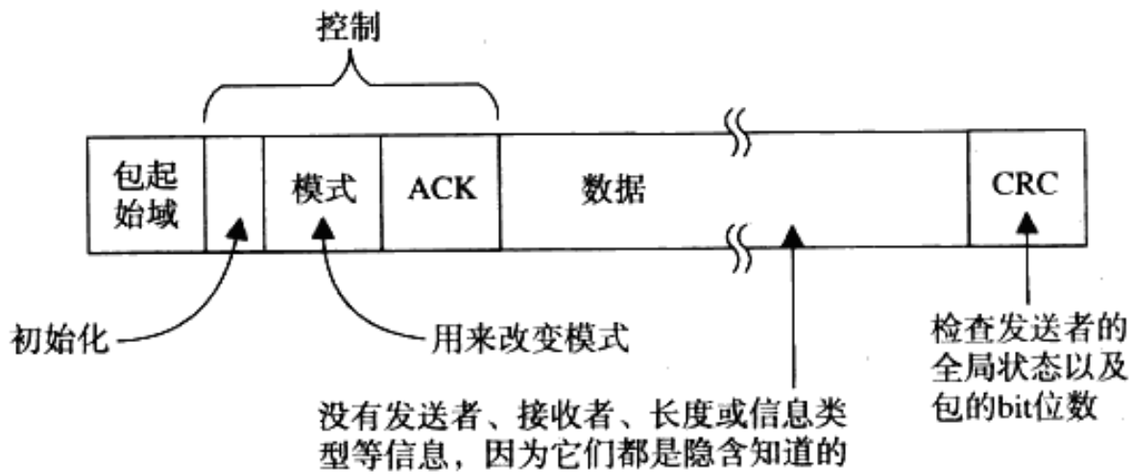



图 4.26　典型的 TTP 包

一个典型的数据包格式如图4-18所示。它包括了一个头域、一个控制域、一个数据域、以及一个CRC校验域。

控制域包含一个初始化位用于初始化，一个存放当前模式的域和一个对前一节点发送数据包的确认域。使用这个域的目的是让前一节点知道自己在正常工作，并且它的数据包已发送到网上了。如果没有进行一个预期的确认，那么，所有的节点都将应发确认数据包的节点标成崩溃，并把它从当前成员位图中删去。被删去的节点将无条件地从系统中消失了。

数据域包括所需的数据。CRC校验域的CRC校验提供了一个既包括整个数据包的校验和，又包括了全局状态的校验和。这就意味着，如果发送者的全局状态不正确，那么，它发送的所有数据包中的CRC校验值都与数据接收者根据自己全局状态计算出的值不一样。于是，接收者将不会给出确认应答，这样，包括出错节点在内的所有节点都会在系统成员位图中将这个出错节点标为失败。

# 5 实时分布式系统
# 5.3 实时通信

- 周期性的广播将只发送含有初始化位的数据包
  - 这个数据包包含当前的全局状态
  - 任何一个已被标为非成员的节点通过广播后可以作为被动成员加入到系统
    - 如果一个节点已被认为是一个成员
      - 它就会被分配一个TDMA时间片
      - 并通过确认数据包，被标记为活动节点

周期性的广播将只发送含有初始化位的数据包。这个数据包包含当前的全局状态。任何一个已被标为非成员的节点通过广播后可以作为被动成员加入到系统。如果一个节点已被认为是一个成员，那么，它就会被分配一个TDMA时间片，因而它就可以用自己的TDMA时间片来响应其它节点发给自己的数据包。一旦它的数据包被确认，所有其它节点又会把它标为一个活动节点。

TDMA协议处理时钟同步的方法比较简单。因为每一个节点都知道TDMA帧何时启动以及它的时间片在帧内的位置，所以，它就会准确地知道何时能够发送一个数据包。这种方法避免了冲突。

# 5 实时分布式系统
# 5.4 实时调度

- 动机
  - 实时系统通常是由一些短任务（即进程或线程）组成
  - 完成一个激励的响应通常需要运行多个任务
  - 这些任务在执行顺序上有一定的限制
  - **必须**决定每一个任务在哪一个处理机上运行

实时系统通常是由一些短任务（即进程或线程）组成，这些短任务都有明确的功能以及有限的执行时间。完成一个激励的响应通常需要运行多个任务，这些任务在执行顺序上有一定的限制。另外，还必须有一个实时调度算法决定每一个任务在哪一个处理机上运行。

# 5 实时分布式系统
# 5.4 实时调度

- 实时调度算法的特征
  - 硬实时还是软实时
  - 抢占式还是非抢占式
  - 动态还是静态
  - 集中式的还是非集中式的

实时调度算法涉及到以下几个问题：

硬实时还是软实时：硬实时算法必须保证所有的时间限制都必须满足。而软实时算法允许偶尔超过时间限制但不会产生一些致命的后果。一般来说，硬实时算法要比软实时算法重要。

抢占式还是非抢占式：允许当一个具有更高优先级的任务到来时暂时挂起当前低优先级的任务，直到没有更高优先级任务执行的时候，才重新执行它。非抢占式调度算法是将每一个任务一直执行到结束。也就是说，一旦一个任务已经开始，那么，它就占有处理机,直到该任务运行结束。

动态还是静态：动态调度算法是在执行期间进行调度。当检测到一个事件时，动态抢占式算法就会立即决定是执行与这个事件相关的任务还是继续执行当前的任务。而动态非抢占式算法只是知道又增加了一个任务要执行。只有等当前任务完成后，算法才在就绪任务中选择一个来运行。对于静态算法，不管是否是抢占式的，任务调度都是在执行之前事先定好的。当一个事件发生时，任务调度程序只是到一个表中查看应该执行什么任务并执行该任务。

集中式的还是非集中式的：调度可以是集中式的，由一个机器收集所有的信息，并做出所有的任务调度决定；或者是非集中式的，由每一个机器自己做出任务调度决定。在集中式任务调度中，可以同时将被调度的任务分配给处理机。而在非集中式任务调度中，将任务分配给哪一个处理机与将哪一个任务分配给指定处理机是不相同的。

# 5 实时分布式系统

## 5.4 实时调度

- **可调度任务**：假设，一个周期性实时分布式系统有m个任务运行在N个处理机上。
- 令$C_i$是任务i需要的处理机时间
- $P_i$是它的周期，即连续两个中断之间的时间间隔。
- 显然，系统的利用率μ与N有如下的关系：

$$\mu = \sum_{i=1}^{m} \frac{C_i}{P_i} \leq N$$

假设，一个周期性实时分布式系统有m个任务运行在N个处理机上。令Ci是任务i需要的处理机时间，Pi是它的周期，即连续两个中断之间的时间间隔。显然，系统的利用率μ与N有如下的关系：

# 5 实时分布式系统
# 5.4 实时调度

- 例如，如果一个任务每20毫秒执行一次，每次运行10毫秒，那么，它就占用了0.5个处理机。5个这样的任务就需要3个处理机来承担。

- 一组能够满足上述要求的任务我们称其是可调度的。需要注意的是，上面等式所给出的处理机数目比实际需要的处理机数目要小，因为它忽略了任务切换时间、消息传输时间以及其它一些额外开销，并且假定了调度算法是最佳的。

# 5 实时分布式系统
# 5.4 实时调度—动态调度

- 动态调度
  - 典型动态调度算法是**比率单调算法**
  - C. L. Liu, James W. Layland: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. J. ACM 20(1): 46-61 (1973) (**被引用次数：10960**)
  - 为抢占式调度在**单一**处理机上没有顺序关系和互斥限制的周期性任务而设计的

动态调度：

典型动态调度算法是比率单调算法是Liu 和Layland在1973年提出的。它是为抢占式调度在处理机上没有顺序关系和互斥限制的周期性任务而设计的。

# 5 实时分布式系统
# 5.4 实时调度—动态调度

- **比率单调算法**
  - 事先给每一个任务分配一个与其执行频率相等的优先级，例。。。
  - 在运行的时候，调度程序总是选择**优先级最高**的任务运行，如果需要的话，也可以暂时停止当前任务的运行

算法工作过程：事先给每一个任务分配一个与其执行频率相等的优先级。例如，对于每20毫秒执行一次的任务，我们给它分配的优先级为50；而给每100毫秒执行一次的任务分配的优先级为10。在运行的时候，调度程序总是选择优先级最高的任务运行，如果需要的话，也可以暂时停止当前任务的运行。

# 5 实时分布式系统
# 5.4 实时调度—动态调度

- **比率单调算法**
  - Liu和Layland证明该算法是**最优**的
  - 对于任何一组任务，只要满足下面利用率的条件

$$\mu = \sum_{i=1}^{m} \frac{C_i}{P_i} \leq m(2^{1/m} - 1)$$

  - 它们可用比率单调算法进行调度

Liu和Layland证明该算法是最优的。同时他们还证明，对于任何一组任务，只要满足下面利用率的条件：

它们可用比率单调算法进行调度。当m→∞的时候，不等式右边就趋向于ln2（约0.693）。实际上，这个条件过于苛刻了，对于μ为0.88的一组任务来说，也是可调度的。

# 5 实时分布式系统
# 5.4 实时调度—动态调度

- **最早时限优先算法** (earliest deadline first)
  - 每当检测到一个事件时，调度程序就把它加在等待队列上
  - 这个队列是按照任务时限到期的早迟来排序，时限最先到期的任务排在最前面
  - 它产生结果也是最佳的
  - 即使对μ=1的一组任务，它也是可调度的

另一个常用的抢占式动态调度算法就是最早时限优先算法(earliest deadline first)。每当检测到一个事件时，调度程序就把它加在等待队列上。这个队列是按照任务时限到期的早迟来排序，时限最先到期的任务排在最前面。

（对于一个周期性任务，时限就是该任务所对应的事件下一次发生的时间）这个调度程序总是从队列中选择第一个任务（也就是最早到期的任务）来执行。同比率单调算法一样，它产生结果也是最佳的。即使对μ=1的一组任务，它也是可调度的。

# 5 实时分布式系统
# 5.4 实时调度—动态调度

- 最小余度法
  - 先计算每一个任务已完成的时间量，称之为松弛度或余度(laxity)
  - 算法只选择松弛度最小的任务来运行

第三个抢占式动态算法先计算每一个任务已完成的时间量，称之为松弛度(laxity)。例如，对于一个必须在200毫秒内完成的任务，如果它还需要运行150毫秒，那么，它的松弛度就是50毫秒。算法只选择松弛度最小的任务来运行，也就是选择最不能拖延的任务来运行。

# 5 实时分布式系统
# 5.4 实时调度—动态调度

- 上述算法没有一个在分布式系统中是最优的
  - 但它们在分布式系统中可作为启发式算法
- 没有一个算法考虑了任务的顺序关系和互斥限制
- 这些算法只是在**理论上有用**，而在实际应用中却不十分有效

上述算法没有一个在分布式系统中是最优的，但它们在分布式系统中可作为启发式算法。同样，没有一个算法考虑了任务的顺序关系和互斥限制，即使在单一处理机上也没有考虑任务顺序关系和互斥限制。这些算法只是在理论上有用，而在实际应用中却不十分有效。

# 5 实时分布式系统

# 5.4 实时调度—**静态**调度

- **定义**： 静态调度是在系统开始运行之前进行的
- **输入**：
  - 所有任务的列表
  - 每一个任务运行需要的时间
- **输出**：
  - 将任务分配到各个处理机上，并为每一个处理机给出任务的执行顺序

理论上：穷举，并获得最优方案
实际上：启发式算法

静态调度是在系统开始运行之前进行的。算法输入包括所有任务的列表和每一个任务运行需要的时间。静态调度算法的目的就是将任务分配到各个处理机上，并为每一个处理机给出任务的执行顺序。在理论上，静态调度算法可以穷举所有的调度方案，并得到最优的方案，但是穷举时间与任务的个数成指数增长，因此，一般只使用启发式静态调度算法。

# 5 实时分布式系统
# 5.4 实时调度—静态调度

- 假设 （前提条件）
  - 每当检测到一个特殊事件时，任务1就在处理机A上启动
  - 任务分配给处理机由外部来完成（已完成处理机分配）
  - 所有的任务都只需要一个处理机时间单位

首先假设，每当检测到一个特殊事件时，任务1就在处理机A上启动，如图3-19所示。这个任务又依次在本地和处理机B上启动其它任务。为了简化，我们假定任务分配给处理机由外部来完成，所有的任务都只需要一个处理机时间单位。

# 5 实时分布式系统

# 5.4 实时调度—静态调度

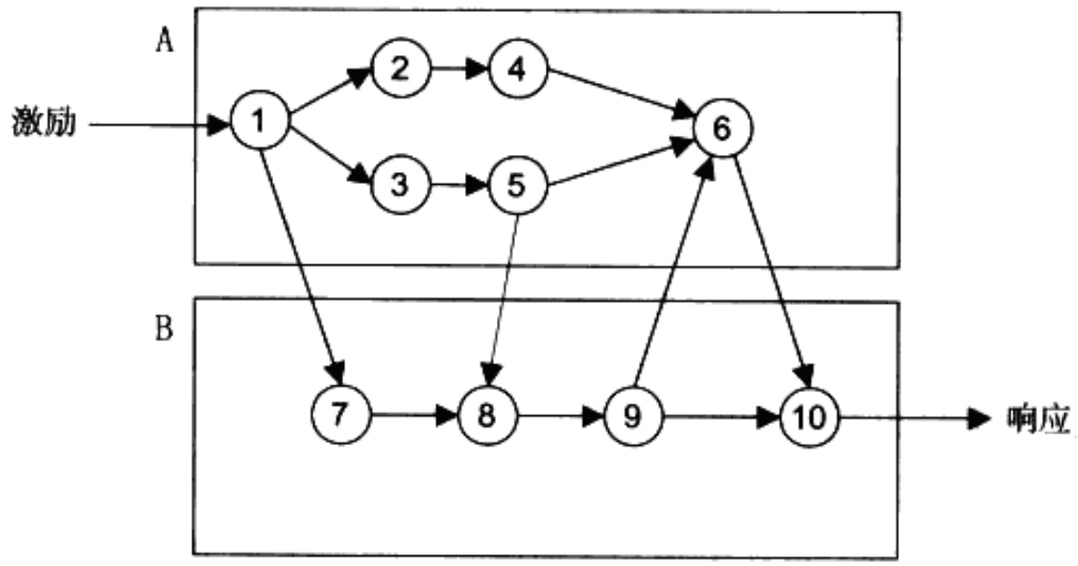



图 4.27　在两个处理机上执行 10 个实时任务

任务1可以启动本地处理机上的任务2或任务3，同时在处理机B上启动任务7。这三个任务又都启动另一个任务，依次类推，如图3-19 任务1可以启动本地处理机上的任务2或任务3，同时在处理机B上启动任务7。这三个任务又都启动另一个任务，依次类推，如图3-19所示。箭头表示任务间有消息传递，X→Y理解为：在X的消息没有到达Y之前，Y不能运行。有一些任务，如任务8，在它启动运行之前，必须得到两个指定的消息。只有当任务10运行完毕，并对初始激励产生正确的响应后，整个周期才算结束。

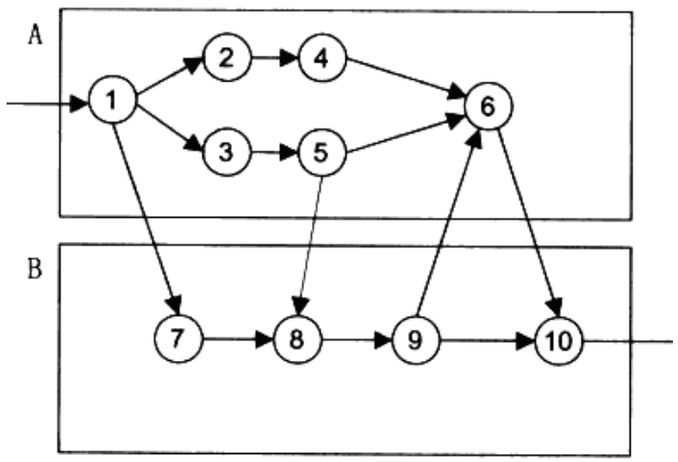
在任务1运行完毕，任务2和任务3都是可运行的。调度程序要选择其中之一来运行。假定选择了任务2。任务2完成后，它必须从任务3 和任务4中选择一个任务来运行。如果它选择任务3，那么，当任务3运行完毕，它必须在任务4和任务5之间做出选择。然而，如果选择了任务4而不是任务5，那么，它就必须在任务4完成之后运行任务5，而不能运行任务6，因为任务6只有在任务5和任务9都运行结束后才能运行。

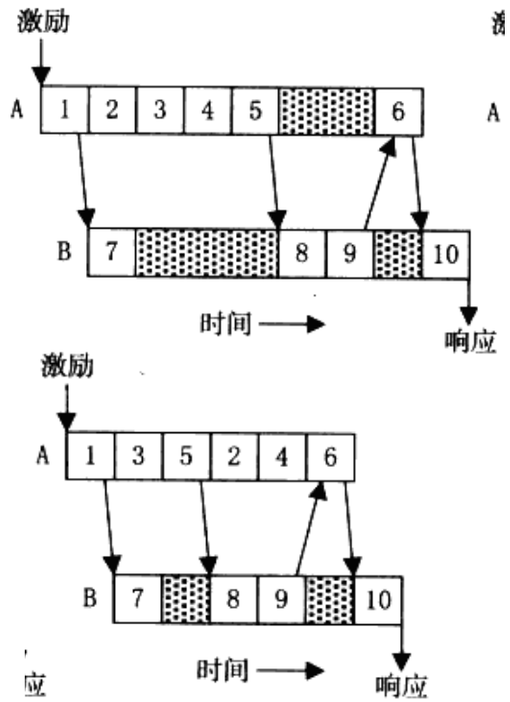
同时，在处理机B上的工作也可以并行地进行。一旦任务1启动了处理机B上的任务7，任务7就可以与任务2或者任务3同时运行。当任务5和任务7都已经运行结束之后，任务8就能够启动运行了，如此执行下去。注意，任务6需要从任务4、任务5和任务9来的输入，并给任务10输出消息。

# 5 实时分布式系统

# 5.4 实时调度—静态调度

- 两种可能的调度方法




两个可能的调度方法如图3-20(a)和(b)中所示。不同处理机上任务之间的消息传递用箭头来表示；同一台处理机上任务之间的消息传递在处理机内部处理，故而不用标出。在两个调度算法中，图3-20(b)中的算法更好一些，因为它能让任务5更早地运行，也就是让任务8更早地运行。如果任务5被延迟执行的话，如图3-20(a)中所示，那么，任务8和任务9就会被延迟执行，从而任务6直至任务10都被延迟执行。

对于静态调度，调度程序在系统开始运行前就必须决定是使用3-20(a)调度策略还是使用3-20(b)调度策略或者使用其它静态调度策略。它对图3-19进行分析，同时将所有任务的运行时间作为输入信息，然后，使用一个启发式算法来找到一个好的调度策略。一旦一个调度策略被确定下来之后，所有的调度信息存入表中，当系统运行时，使用一个简单的分配器就能够进行调度，并且开销较小。

# 5 实时分布式系统
# 5.4 实时调度—对比

- 动态调度和静态调度的比较:
  - 静态调度比较适合于时间触发的系统，而动态调度比较适合于事件触发的系统
  - 动态调度比静态调度更能充分地利用资源
    - 对于静态调度，系统在设计中必须考虑各种各样的情况，并且还必须考虑最坏情况，因而，浪费了大量的资源
- 寻找最优策略 —》大量测试

动态调度和静态调度的比较:
静态调度比较适合于时间触发的系统，而动态调度比较适合于事件触发的系统。静态调度必须事先认真仔细设计，并需要花费较大的精力来选择各种各样的参数。而动态调度则不需要事先做这么多工作，因为它是在运行期间动态地进行选择。
动态调度比静态调度更能充分地利用资源。对于静态调度，系统在设计中必须考虑各种各样 的情况，并且还必须考虑最坏情况，因而，浪费了大量的资源。然而，在一个硬实时系统中，浪费大量的资源是为了满足系统所有的实时限制。
如果具有足够的计算能力，那么，静态系统可以事先得到一个最优或者次优的调度策略。然而，对于核反应堆控制的应用来说，花上好几个月的计算时间来寻找一个最优策略是值得的，也是必须的。但是，一个动态系统要在运行期间花费大量的时间来进行调度计算几乎是不可能的。因此，为了安全起见，人们就不得不将大量的时间花费在设计期间。但即使是这样，仍然不能保证所有的实时限制都能够得到满足。因此，必须进行大量的测试。