第2讲

实时软件设计基础

目录

2.1 生命周期问题

与任何软件系统一样，在开发并发与实时系统时，也应该使用软件生命周期模型，这是分阶段开发软件的方法瀑布模型是使用最为广泛的软件生命周期模型。这部分将对瀑布模型进行概述。另外还会介绍其他一些软件生命周期模型。

2.1 生命周期问题

2.1.1 瀑布生命周期模型

1. 需求分析和规范

使用瀑布模型的阶段，必须确认并分析用户的需求。用户的需求包括软件需求和系统需求。实时系统通常是较大的嵌入式系统的组成部分，所以确定系统需求分析和规范阶段的工作就很有可能要在确定软件需求分析和规划工作之前进行。

2. 构架设计

在模型阶段中，系统从结构上分解为各个组成部分。并发实时系统与其他系统区分开的重要因素一个是这种系统将自身分解为多个并发任务，另一个是对系统在行为方面的考虑。

3. 详细设计

在详细设计阶段，要使用程序设计语言表示法定义系统各个组成部分。在并发与实时系统中要注意资源共享的算法，要避免出现死锁情况，还要注意与硬件I/O设备的接口。

4. 编码

遵照编写代码和文档的标准，使用项目中所选择的编程语言来编写各个组成部分的代码。对于并发系统来说，要选择使用并发语言(Ada,Modula2)，或者选择多任务操作系统或内核所支持的序列化语言。

5. 软件测试

并发与实时系统包含了多个并发任务，或者与多个外部设备之间存接口。系统的执行具有不确定性，而且实时系统通常是嵌入式系统，所以测试更为复杂。有时需要开发环境模拟器。需要分几个阶段对软件系统测试。单元测试和集成测试都是“白盒”测试方法，需要了解软件内部结构，系统测试是一种“黑盒”测试方法，要依照软件需求规范。

6.单元测试

单元测试由在与其他组成部分结合之前，对单个组成部分进行的测试组成。*最低覆盖原则*每条语句只要执行一次，每个输出分支至少测试一次。

7.集成测试

集成测试是将经过测试的组成部分逐渐结合为一些更加复杂的组成部分，并且在将整个软件系统组合在一起并对接口进行测试之后，对这些分组进行测试。并发系统集成测试的特点是需要对并发任务接口进行测试。

8.系统测试

系统测试是对集成后的硬件和软件系统进行测试的过程，以确保系统符合需求规范。需要对并发和/或实时系统的某些方面进行测试。包括：功能测试、负载(压力)测试、性能测试

9.验收测试

验收测试通常由用户组织或其代表来实施。

2.1 生命周期问题

2.1.2 其他软件生命周期模型

1.抛弃式原型法

抛弃式原型是一种开发快速,成本低廉的工作系统,用于帮助澄清用户的需求。在初步需求规范之后就可以开发抛弃式原型。通过让用户在原型上练习操作，可以得到很多有价值的反馈信息。根据这些反馈信息，可以完成一份经过修订的需求规范。随后的开发过程遵照常规的软件生命周期模型继续进行。

2.增量式开发的演化原型法

演化原型法是一种增量式开发的形式，其中的原型要经过几个过渡的运行系统演化为可交付系统。通过在较长时间段内展开实施过程，对于测试设计的关键组成部分和降低开发风险来说，这种方法有助于确定系统是否满足了性能上的需求。

3.螺旋模型

螺旋模型是一种迭代生命周期模型, 其中的每一次螺旋循环都代表一次迭代. 径向坐标代表累积开销. 每次迭代的一个重要方面就是项目的风险评估. 要识别出具有最大不确定性的区域和潜在的重要问题. 只有在风险减少到管理层可以接受的水平时, 才可以开始进行具体的实施工作。

2.2 软件设计概念

本节介绍的是并发与实时系统软件设计的关键概念。我们会对前面介绍的并发处理概念进行深入探讨，尤其是并发任务之间的通信与同步问题。

随后会介绍系统环境以及支持并发处理的操作系统。信息隐藏的概念要从系统分解为多个模块的角度来介绍。面向对象的概念要与面向对象设计中信息隐藏的作用一同讨论，此外还引入了类和继承的概念，最后要介绍的是并发与实时系统设计中有限状态机的作用。

2.2 软件设计概念

2.2.1 并发处理

并发任务的通信与同步：

在大多数实时和并发应用程序中，并发任务必须相互协作，以执行供应用程序所要求的服务。当各个任务之间相互协作时,会出现以下三个常见问题：

1) *互斥问题*

当任务需要对某种资源(如共享数据或物理设备)具有独占的访问权时，就会发生这种错误。

为了解决这种问题，必须提供一种同步机制，以保证多个任务对关键资源的访问是互斥的。其经典解决方案最早是由Dijkstra使用二元信号量提出的。二元信号量s是一个布尔变量，仅能使用两个原子操作**P操作**(Wait(s))和**V操作**(Signal(s))。

这个问题的另一种形式是多读者/多写者问题。

2) *任务同步问题*

当两个任务需要协同运行，但不在任务之间传递数据时，使用事件同步。事件用于同步两个任务的运行。源任务执行一个Signal（事件）操作，它发出信号，说明一个事件已经发生。目标任务执行一个Wait（事件）操作，将任务挂起，直到事件接收到来自信号生成者的信号为止。如果事件已经接收到了信号，那么目标任务就不再处于挂起状态。

3) *生产者/消费者问题*

在并发系统中，每个任务都有自己的控制线程，各个任务之间采用异步方式执行。因此，当各个任务之间需要交换数据时，就要对任务的运行进行同步。数据必须要在消费者使用之前由生产者生成出来。如果消费者已经为接收数据做好准备，但生产者还没有生成数据，那么消费者就必须等待生产者。如果生产者在消费者为接收数据做好准备前就已经生成了数据，那么生产者就不得不停下来，或者缓冲传递给消费者的数据，这样才能继续生成数据。

2.2 软件设计概念

2.2.2 并发处理的环境

并发系统三种主要环境：

1) *多道程序环境*

多个任务共享一个处理器。虚拟的并发通过让操作系统为各个任务控制处理器的分配来实现的，看起来好象每个任务都有一个专用的处理器。 图2.1演示了用于小型机或者微机的典型多道程序环境。在图2.1中有一个CPU卡和内存卡用于存储任务和数据，使用设备接口卡连接了两个I/O设备，显示设备和传感器输入设备。

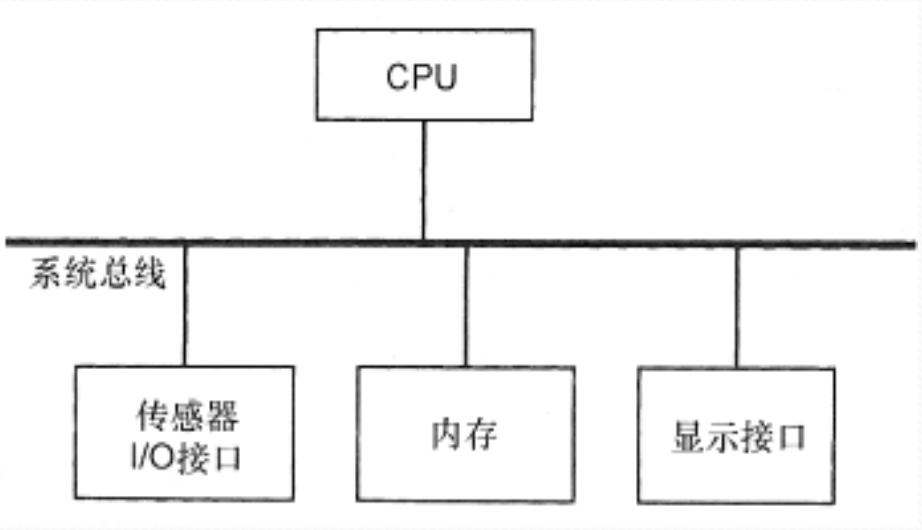


图2.1多道程序(单CPU)环境

2) *多处理器环境*

如图2.2在这种环境中有两个或两个以上的处理器使用共享内存。所有处理器都有一个公用的虚拟地址空间。所有任务都驻留在共享内存中。在多处理器环境中，当多个处理器并发执行时，所实现的就是真实的并发。

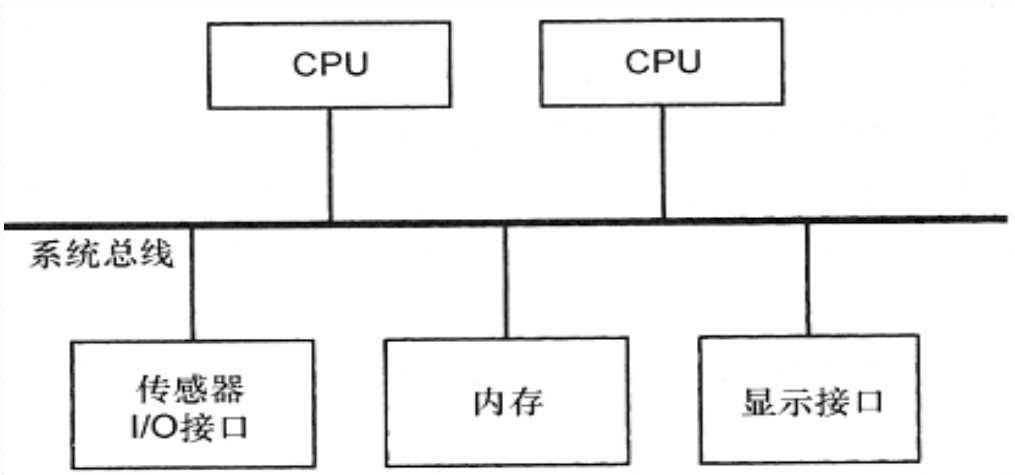


图2.2 多处理器环境

3) *分布式处理环境*

在这种环境中，有两台或多台计算机通过通信网络或高速总线相互连接。每台计算机都有自己的本地内存，在处理器之间没有共享内存。因此，由多个并发任务组成的分布式应用程序可以使分布在网络上的任务通过消息进行通信。图2.3中的结点通常都由前述图2.1所示的多道程序系统或图2.2所示的多处理器系统。

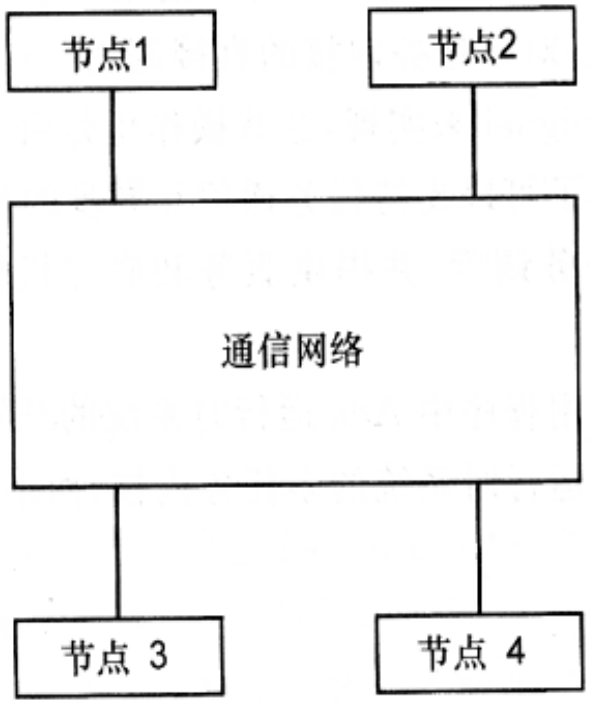


图2.3 分布式处理环境

2.2 软件设计概念

2.2.3 并发处理的操作系统支持

1) *运行时支持服务*

通过使用一条实时多任务指令(也称为操作系统内核)或执行并发语言的运行时支持系统,就可以提供并发任务的运行时支持。

如果要使用序列化语言开发并发任务应用程序，就必须使用多任务内核。多任务内核所提供的典型服务：

(1)抢占式优先级调度

(2)使用消息进行任务间通信

(3)使用信号量进行互斥访问

(4)使用事件来实现任务间同步

(5)中断处理和基本的I/O服务

(6)内存管理

而并发语言提供了可以支持任务通信和同步的构造。

2) *任务调度*

在单处理器系统中，多任务内核必须为CPU调度并发任务。内核为所有任务准备好使用CPU的Ready List。对于分时共享系统多个用户要交互式地访问系统，经常使用轮询调度算法。对于实时系统更为合适的算法是抢占式优先级调度算法。

在多处理器环境中，会有一个多任务内核的副本在每个处理器上执行。每个处理器都会选择执行Ready List中最上面的任务。对Ready List的互斥访问是使用硬件信号量来实现的。

3) *输入/输出问题*

实现输入/输出有两种通用机制，中断驱动I/O和轮询I/O。

- 使用中断驱动I/O时，当接受到输入数据或者完成一次输出操作之后就会产生一次中断。使用这种机制有两种常用方法，一种方法时是在读写每个字符之后产生中断。另一种方法是在I/O设备和主存之间放置一个DMA设备。
- 在使用轮询I/O机制时，系统必须定期对输入设备进行取样，以确定是否有任何输入已经到达，并定期对输出设备进行采样，以确定输出操作是否已经完成。

2.3 信息隐藏

信息隐藏是一种基本的软件设计概念。使用信息隐藏把可能发生变化的信息封装在一个模块的内部。对信息的外部访问只能通过间接调用也属于模块一部分的操作，这样隐藏信息和访问信息的操作放在一起就形成了信息隐藏模块。

2.3 信息隐藏

2.3.1 应用于内部数据结构的信息隐藏

信息隐藏可用于隐藏与数据结构，其内部联系以及操作数据结构的操作细节相关的设计决策。其他的模块只能通过调用模块的间接操作来访问封装后的的数据结构。这样对某个模块的修改只要其外部接口没有发生改变，间接访问其数据结构的其他模块就不用改变。

2.3 信息隐藏

2.3.2 应用于访问同步的信息隐藏

解决互斥问题和多读者/多写者问题的方案容易发生错误。可能会在访问共享数据的一个任务中出现编码错误，这样会在执行时产生严重的同步错误。这些潜在的问题都是由同步全局的问题引起的，而使用信息隐藏的方法，全局同步问题就可以简化为本地同步问题，只需要在同步过程中考虑一个信息隐藏模块即可。

2.3 信息隐藏

2.3.3 应用于I/O设备的信息隐藏

信息隐藏可以用于隐藏如何与具体I/O设备接口的设计决策。

解决方案是向隐藏设备具体细节的设备提供一个虚拟接口。如果该设备由另一个具有相同功能的设备替换，需要修改模块的内部操作，因为它们必须处理如何与真实设备接口的精细细节。然而由操作规范代表的虚拟接口保持不便，访问设备接口的模块也无需修改。

2.4 面向对象

Wegner把支持对象(信息隐藏模块)但不支持继承的语言(Ada,Modula-2)称为***基于对象的语言***;把既支持类又支持对象的语言(CLu)称为***基于类的语言***；而把支持对象，类和继承的语言(Smalltalk,C++,Eiffel)称为***面向对象的语言***

2.4 面向对象

2.4.1 主动对象和被动对象

对象可以是主动的，也可以是被动的。主动对象也就是自治的异步对象，可以启动自己的操作。主动对象是并发任务，有自己的控制线程，可以独立于其他任务执行。被动对象是一种信息隐藏模块，没有控制线程，其操作由主动对象来调用。本讲所提到的对象一般都指被动对象。

2.4 面向对象

2.4.2 类

类是一种对象类型，是用于对象的模板。类的概念通常要与抽象数据类型联系在一起。类一般定义为抽象数据类型的实现。对象是信息隐藏模块，则类是一种模块类型。

2.4 面向对象

2.4.3 继承

可以使用继承对类进行限定。继承是用于在类之间共享和重用代码的一种机制。通过添加新的操作和实例变量，或者对现有的操作重新进行定义，一个子类可以使用其父类的结构(即封状的数据)和行为(即操作)，用于自己使用。

类和继承 – wikipedia, Baidu

2.5 有限状态机

有限状态机可以用于为系统的行为方面进行建模。许多实时系统都非常依赖于状态，亦即它们的操作不仅依赖于输出信息，而且还依赖于以前发生过的情况。可以使用一个有限状态机来定义实时系统的状态依赖方面。

有限状态机是一种概念上的机器，具有一定数量的状态。在某个特定时刻只能处于其中一种状态，状态转换是由输入事件引起的状态变化，理论上是不占用时间的，实际上发生状态转换的时间可以忽略不计。

用于定义有限状态机的表示法是状态转换图和状态转换表或矩阵。

2.5 有限状态机

2.5.1 有限状态机在实时系统设计中的使用

大规模的系统都非常依赖于状态，状态转换图或表格/矩阵对理解系统复杂性很有帮助。使用有限状态机对系统进行建模时，状态和转换是有限的，状态表示系统行为的模式。系统在某一时刻只能处于一种状态，所以一种状态代表的系统中当前发生的情况。

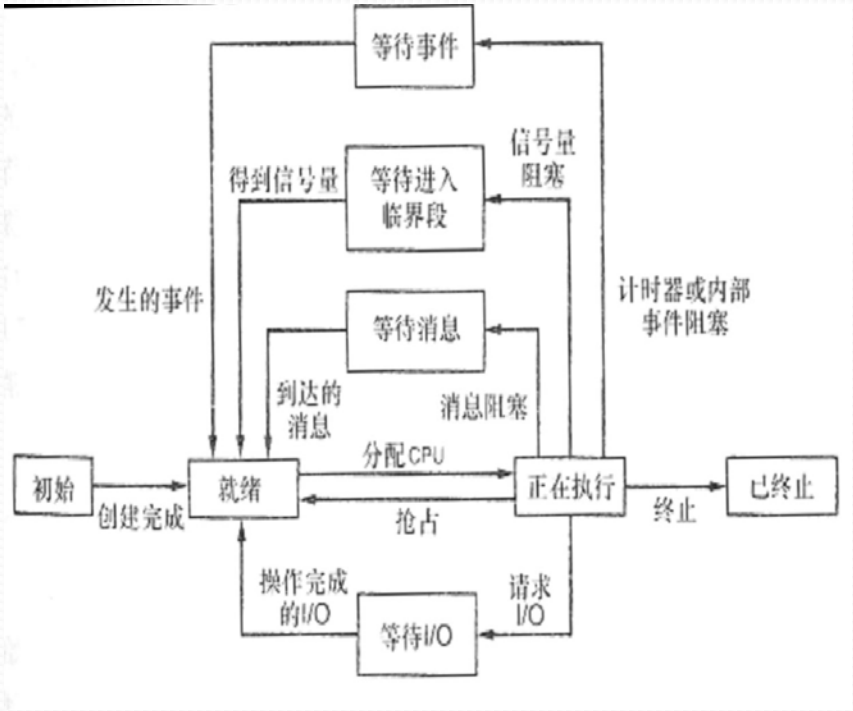
复杂系统的状态转换图也可能是复杂的，简化系统有限状态机表示的方法就是使用单独的状态转换图对系统的不同方面进行建模。

2.5 有限状态机

2.5.2 状态转换图的例子

将并发任务的不同状态作为状态转换图的例子来研究。这些状态有一个多任务内核来维持，这个内核用的是抢占式优先级调度算法。任务的不同状态可以在一个状态转换图上描述，状态用方框表示，转换用折线表示。

当首次创建任务时，它处于就绪状态，这时它位于就绪列表中。当该任务到达就绪列表的顶端时，就可以得到CPU资源，这时它就转换到了执行状态。此后该任务的资源就可以被另一个任务抢占。



进入就绪状态，这时内核就会根据它的优先级将其放置在就绪列表中的某个位置上。

当处于执行状态时，任务可能会发生阻塞，在这种情况它就会进入相应的阻塞状态。

一个任务可能因为等待I/O，等待另一个任务发送来的消息，等待计时器事件或由另一个任务激发的事件，或者等待进入临界端而发生阻塞。当I/O操作完成，消息到达，发生了事件，或者任务获得允许进入临界段，此时阻塞的任务就重新进入就绪状态。

FSM – wikipedia, Baidu

本讲主要应理解实时软件设计理念及基本概念

本讲结束，欢迎讨论！