

嵌入式系统以及实时软件开发

作者：郑泽胜

编者备注：根据众多零散资料整理。特别感谢龚灼先生提供了 Nucleus 的资料。

一、什么是嵌入式系统和嵌入式微处理器

嵌入式系统一般指非 PC 系统，它包括硬件和软件两部分。硬件包括处理器 / 微处理器、存储器及外设器件和 I / O 端口、图形控制器等。软件部分包括操作系统软件 (OS) (要求实时和多任务操作) 和应用程序编程。有时设计人员把这两种软件组合在一起。应用程序控制着系统的运作和行为；而操作系统控制着应用程序编程与硬件的交互作用。

嵌入式系统的核心是嵌入式微处理器。嵌入式微处理器一般就具备以下 4 个特点：1) 对实时多任务有很强的支持能力，能完成多任务并且有较短的中断响应时间，从而使内部的代码和实时内核的执行时间减少到最低限度。2) 具有功能很强的存储区保护功能。这是由于嵌入式系统的软件结构已模块化，而为了避免在软件模块之间出现错误的交叉作用，需要设计强大的存储区保护功能，同时也有利于软件诊断。3) 可扩展的处理器结构，以能最迅速地开发出满足应用的最高性能的嵌入式微处理器。4) 嵌入式微处理器必须功耗很低，尤其是用于便携式的无线及移动的计算和通信设备中靠电池供电的嵌入式系统更是如此，如需要功耗只有 mW 甚至 μ W 级。

二、面向二十一世纪的嵌入式系统综述

1、计算机工业的分类

以往我们按照计算机的体系结构、运算速度、结构规模、适用领域，将其分为大型计算机、中型机、小型机和微计算机，并以此来组织学科和产业分工，这种分类沿袭了约 40 年。近 10 年来随着计算机技术的迅速发展，实际情况产生了根本性的变化，例如 70 年代末定义的微计算机演变出来的个人计算机 (PC)，如今已经占据了全球计算机工业的 90% 市场，其处理速度也超过了当年大、中型计算机的定义。随着计算机技术和产品对其它行业的广泛渗透，以应用为中心的分类方法变得更为切合实际，也就是按计算机的嵌入式应用和非嵌入式应用将其分为嵌入式计算机和通用计算机。

通用计算机具有计算机的标准形态，通过装配不同的应用软件，以类同面目出现并应用在社会各个方面，其典型产品为 PC；而嵌入式计算机则是以嵌入式系统的形式隐藏在各种装置、产品和系统中。

2、嵌入式系统 (Embedded Systems)

嵌入式系统被定义为：以应用为中心、以计算机技术为基础、软件硬件可裁剪、适应应用系统对功能、可靠性、成本、体积、功耗严格要求的专用计算机系统。

嵌入式计算机在应用数量上远远超过了各种通用计算机，一台通用计算机的外部设备中就包含了 5-10 个嵌入式微处理器，键盘、鼠标、软驱、硬盘、显示卡、显示器、网卡、Modem、声卡、打印机、扫描仪、数码相机、USB 集线器等均是由嵌入式处理器控制的。在制造工业、过程控制、通讯、仪器、仪表、汽车、船舶、航空、航天、军事装备、消费类产品等方面均是嵌入式计算机的应用领域。

嵌入式系统是将先进的计算机技术、半导体技术和电子技术和各个行业的具体应用相结合后的产物，这一点就决定了它必然是一个技术密集、资金密集、高度分散、不断创新的知识集成系统。

今天嵌入式系统带来的工业年产值已超过了 1 万亿美元，1997 年来自美国嵌入式系统大会 (Embedded System Conference) 的报告指出，未来 5 年仅基于嵌入式计算机系统的全数字电视产品，就将在美国产生一个每年 1500 亿美元的新市场。美国汽车大王福特公司的高级经理也曾宣称，“福特出售的‘计算能力’已超过了 IBM”，由此可以想见嵌入式计算机工业的规模和广度。1998 年 11 月在美国加州何塞举行的嵌入式系统大会上，基于 RTOS 的 Embedded Internet 成为一个技术新热点。

美国著名未来学家尼葛洛庞帝 99 年 1 月访华时预言，4~5 年后嵌入式智能(电脑)工具将是 PC 和因特网之后最伟大的发明。我国著名嵌入式系统专家、华中科技大学教授沈绪榜院士 98 年 11 月在武汉全国第 11 次微机学术交流会上发表的《计算机的发展与技术》一文中，对未来 10 年以嵌入式芯片为基础的计算机工业给出了十分令人鼓舞的展望。

3、嵌入式系统工业 (Embedded System Industry, ESI) 的特点和要求

3.1、嵌入式系统工业是不可垄断的高度分散的工业

从某种意义上来说，通用计算机行业的技术是垄断的。占整个计算机行业 90% 的 PC 产业，80% 采用 Intel 的 8x86 体系结构，芯片基本上出自 Intel, AMD, Cyrix 等几家公司。在几乎每台计算机必备的操作系统和文字处理器方面，Microsoft 的 Windows 及 Word 占 80-90%，凭借操作系统还可以搭配其它应用程序。因此当代的通用计算机工业的基础被认为是由 Wintel (Microsoft 和 Intel 90 年代初建立的联盟) 垄断的工业。

嵌入式系统则不同，它是一个分散的工业，充满了竞争、机遇与创新，没有哪一个系列的处理器和操作系统能够垄断全部市场。即便在体系结构上存在着主流，但各不相同的应用领域决定了不可能有少数公司、少数产品垄断全部市场。因此嵌入式系统领域的产品和技术，必然是高度分散的，留给各个行业的中小规模高技术公司的创新余地很大。另外，社会上的各个应用领域是在不断向前发展的，要求其中的嵌入式处理器核心也同步发展，这也构成了推动嵌入式工业发展的强大动力。

嵌入式系统工业的基础是以应用为中心的“芯片”设计和面向应用的软件产品开发。

3.2、嵌入式系统具有的产品特征

嵌入式系统是面向用户、面向产品、面向应用的，如果独立于应用自行发展，则会失去市场。嵌入式处理器的功耗、体积、成本、可靠性、速度、处理能力、电磁兼容性等方面均受到应用要求的制约，这些也是各个半导体厂商之间竞争的热点。

和通用计算机不同，嵌入式系统的硬件和软件都必须高效率地设计，量体裁衣、去除冗余，力争在同样的硅片面积上实现更高的性能，这样才能在具体应用对处理器的选择面前更具有竞争力。嵌入式处理器要针对用户的具体需求，对芯片配置进行裁剪和添加才能达到理想的性能；但同时还受用户订货量的制约。因此不同的处理器面向的用户是不一样的，可能是一般用户，行业用户或单一用户。

嵌入式系统和具体应用有机地结合在一起，它的升级换代也是和具体产品同步进行，因此嵌入式系统产品一旦进入市场，具有较长的生命周期。嵌入式系统中的软件，一般都固化在只读存储器中，而不是以磁盘为载体，可以随意更换，所以嵌入式系统的应用软件生命周期也和嵌入式产品一样长。另外，各个行业的应用系统和产品，和通用计算机软件不同，很少发生突然性的跳跃，嵌入式系统中的软件也因此更强调可继承性和技术衔接性，发展比较稳定。

嵌入式处理器的发展也体现出稳定性，一个体系一般要存在 8-10 年的时间。一个体系结构及其相关的片上外设、开发工具、库函数、嵌入式应用产品是一套复杂的知识系统，用户和半导体厂商都不会轻易地放弃一种处理器。

3.3、 嵌入式系统软件的特征

嵌入式处理器的应用软件是实现嵌入式系统功能的关键，对嵌入式处理器系统软件和应用软件的要求也和通用计算机有所不同。

(1) 软件要求固态化存储

为了提高执行速度和系统可靠性，嵌入式系统中的软件一般都固化在存储器芯片或单片机本身中，而不是存贮于磁盘等载体中。

(2) 软件代码高质量、高可靠性

尽管半导体技术的发展使处理器速度不断提高、片上存储器容量不断增加，但在大多数应用中，存储空间仍然是宝贵的，还存在实时性的要求。为此要求程序编写和编译工具的质量要高，以减少程序二进制代码长度、提高执行速度。

(3) 系统软件(OS)的高实时性是基本要求

在多任务嵌入式系统中，对重要性各不相同的任务进行统筹兼顾的合理调度是保证每个任务及时执行的关键，单纯通过提高处理器速度是无法完成和没有效率的，这种任务调度只能由优化编写的系统软件来完成，因此系统软件的高实时性是基本要求。

(4) 多任务操作系统是知识集成的平台和走向工业化道路的基础。

3.4、 嵌入式系统开发需要开发工具和环境

通用计算机具有完善的人机接口界面，在上面增加一些开发应用程序和环境即可进行对自身的开发。而嵌入式系统本身不具备自举开发能力，即使设计完成以后用户通常也是不能对其中的程序功能进行修改的，必须有一套开发工具和环境才能进行开发，这些工具和环境一般是基于通用计算机上的软硬件设备以及各种逻辑分析仪、混合信号示波器等。

3.5、 嵌入式系统软件需要 RTOS 开发平台

通用计算机具有完善的操作系统和应用程序接口(API)，是计算机基本组成不可分离的一部分，应用程序的开发以及完成后的软件都在 OS 平台上面运行，但一般不是实时的。嵌入式系统则不同，应用程序可以没有操作系统直接在芯片上运行；但是为了合理地调度多任务、利用系统资源、系统一般以成熟的实时操作系统作为开发平台，这样才能保证程序执行的实时性、可靠性，并减少开发时间，保障软件质量。

3.6、 嵌入式系统开发人员以应用专家为主

通用计算机的开发人员一般是计算机科学或计算机工程方面的专业人士，而嵌入式系统则是要和各个不同行业的应用相结合的，要求更多的计算机以外的专业知识，其开发人员往往是各个应用领域的专家。因此开发工具的易学、易用、可靠、高效是基本要求。

中国的单片机应用和嵌入式系统开发走过了 15 年的历程，有超过 10 万名从事单片机开发应用的工程师，但 95% 以上是 3~5 个人的小组以孤军奋战的封闭方式开发几乎不可重用的软件。今天面对的是嵌入式系统工业化的潮流，如果我们不能认清嵌入式软件必须以工业化的方式生产开发，不理解在短时间内装配集成“数百人年”嵌入式产品软件库固化于芯片之中的方法，那么我们将失去更多“上游”产品的市场机遇；反之在我国大力推动和建设“嵌入式软件工厂”，使我国的嵌入式软件库(零件)产品化并溶入国际市场，对加速知识创新和建立面向 21 世纪的知识经济具有战略意义。

三、嵌入式处理器分类与现状

嵌入式系统的核心部件是各种类型的嵌入式处理器，目前据不完全统计，全世界嵌入式处理器的品种总量已经超过 1000 多种，流行体系结构有 30 几个系列，其中 8051 体系的

占有多半。生产 8051 单片机的半导体厂家有 20 多个，共 350 多种衍生产品，仅 Philips 就有近 100 种。现在几乎每个半导体制造商都生产嵌入式处理器，越来越多的公司有自己的处理器设计部门。嵌入式处理器的寻址空间一般从 64KB 到 16-32MB，处理速度从 0.1MIPS 到 2000MIPS，常用封装从 8 个引脚到 144 个引脚。根据其现状，嵌入式计算机可以分成下面几类。

1、嵌入式微处理器(Embedded Microprocessor Unit, EMPU)

嵌入式微处理器的基础是通用计算机中的 CPU。在应用中，将微处理器装配在专门设计的电路板上，只保留和嵌入式应用有关的母板功能，这样可以大幅度减小系统体积和功耗。为了满足嵌入式应用的特殊要求，嵌入式微处理器虽然在功能上和标准微处理器基本是一样的，但在工作温度、成本、功耗、可靠性、健壮性等方面和工业控制计算机相比，嵌入式微处理器具有体积小、重量轻、成本低、可靠性高的优点，但是在电路板上必须包括 ROM、RAM、总线接口、各种外设等器件，从而降低了系统的可靠性，技术保密性也较差。嵌入式微处理器及其存储器、总线、外设等安装在一块电路板上，称为单板计算机。如 STD-BUS、PC104 等。近年来，德国、日本的一些公司又开发出了类似“火柴盒”式名片大小的嵌入式计算机系列 OEM 产品。嵌入式处理器目前主要有 Am186/88、386EX、SC-400、Power PC、68000、MIPS、ARM 系列等。

2、嵌入式微控制器(Microcontroller Unit, MCU)

嵌入式微控制器又称单片机，顾名思义，就是将整个计算机系统集成到一块芯片中。嵌入式微控制器一般以某一种微处理器内核为核心，芯片内部集成 ROM/EPROM、RAM、总线、总线逻辑、定时/计数器、WatchDog、I/O、串行口、脉宽调制输出、A/D、D/A、Flash RAM、EEPROM 等各种必要功能和外设。为适应不同的应用需求，一般一个系列的单片机具有多种衍生产品，每种衍生产品的处理器内核都是一样的，不同的是存储器和外设的配置及封装。这样可以使单片机最大限度地和应用需求相匹配，功能不多不少，从而减少功耗和成本。

和嵌入式微处理器相比，微控制器的最大特点是单片化，体积大大减小，从而使功耗和成本下降、可靠性提高。微控制器是目前嵌入式系统工业的主流。微控制器的片上外设资源一般比较丰富，适合于控制，因此称微控制器。嵌入式微控制器目前的品种和数量最多，比较有代表性的通用系列包括 8051、P51XA、MCS-251、MCS-96/196/296、C166/167、MC68HC05/11/12/16、68300 等。另外还有许多半通用系列如：支持 USB 接口的 MCU 8XC930/931、C540、C541；支持 I2C、CAN-Bus、LCD 及众多专用 MCU 和兼容系列。目前 MCU 占嵌入式系统约 70% 的市场份额。特别值得注意的是近年来提供 X86 微处理器的著名厂商 AMD 公司，将 Am186CC/CH/CU 等嵌入式处理器称之为 Microcontroller，MOTOROLA 公司把以 Power PC 为基础的 PPC505 和 PPC555 亦列入单片机行列。TI 公司亦将其 TMS320C2XXX 系列 DSP 做为 MCU 进行推广。

3、嵌入式DSP处理器(Embedded Digital Signal Processor, EDSP)

DSP 处理器对系统结构和指令进行了特殊设计，使其适合于执行 DSP 算法，编译效率较高，指令执行速度也较高。在数字滤波、FFT、谱分析等方面 DSP 算法正在大量进入嵌入式领域，DSP 应用正从在通用单片机中以普通指令实现 DSP 功能，过渡到采用嵌入式 DSP 处理器。嵌入式 DSP 处理器有两个发展来源，一是 DSP 处理器经过单片化、EMC 改造、增加片上外设成为嵌入式 DSP 处理器，TI 的 TMS320C2000/C5000 等属于此范畴；二是在通用单片机或 SOC 中增加 DSP 协处理器，例如 Intel 的 MCS-296 和 Siemens 的 TriCore。推动嵌入

式 DSP 处理器发展的另一个因素是嵌入式系统的智能化, 例如各种带有智能逻辑的消费类产品, 生物信息识别终端, 带有加解密算法的键盘, ADSL 接入、实时语音解压系统, 虚拟现实显示等。这类智能化算法一般都是运算量较大, 特别是向量运算、指针线性寻址等较多, 而这些正是 DSP 处理器的长处所在。嵌入式 DSP 处理器比较有代表性的产品是 Texas Instruments 的 TMS320 系列和 Motorola 的 DSP56000 系列。TMS320 系列处理器包括用于控制的 C2000 系列, 移动通信的 C5000 系列, 以及性能更高的 C6000 和 C8000 系列。DSP56000 目前已经发展成为 DSP56000, DSP56100, DSP56200 和 DSP56300 等几个不同系列的处理器。另外 PHILIPS 公司今年也推出了基于可重置结构、低成本、低功耗技术上制造的 R. E. A. L DSP 处理器, 特点是具备双 Harvard 结构和双乘/累加单元, 应用目标是大批量消费类产品。

4、嵌入式片上系统(System On Chip)

随着 EDI 的推广和 VLSI 设计的普及化, 及半导体工艺的迅速发展, 在一个硅片上实现一个更为复杂的系统的时代已来临, 这就是 System On Chip(SOC)。各种通用处理器内核将作为 SOC 设计公司的标准库, 和许多其它嵌入式系统外设一样, 成为 VLSI 设计中一种标准的器件, 用标准的 VHDL 等语言描述, 存储在器件库中。用户只需定义出其整个应用系统, 仿真通过后就可以将设计图交给半导体工厂制作样品。这样除个别无法集成的器件以外, 整个嵌入式系统大部分均可集成到一块或几块芯片中去, 应用系统电路板将变得很简洁, 对于减小体积和功耗、提高可靠性非常有利。SOC 可以分为通用和专用两类。通用系列包括 Siemens 的 TriCore, Motorola 的 M-Core, 某些 ARM 系列器件, Echelon 和 Motorola 联合研制的 Neuron 芯片等。专用 SOC 一般专用于某个或某类系统中, 不为一般用户所知。一个有代表性的产品是 Philips 的 Smart XA, 它将 XA 单片机内核和支持超过 2048 位复杂 RSA 算法的 CCU 单元制作在一块硅片上, 形成一个可加载 JAVA 或 C 语言的专用的 SOC, 可用于公众互联网如 Internet 安全方面。

四、信息时代嵌入式装置的通信技术

信息时代嵌入装置 (Information Age Embedded Devices) 具有与其他装置/人进行通信的能力, 告知它们什么是需要的并提供它们所需的信息。根据市场的需求, 设计人员在设计嵌入应用时必须决定采用什么样的通信连接和如何设计通信连接而使产品推向市场的时间最短。下面简述信息时代嵌入装置的通信技术。

1、PCI方案

在高速通信中一个重要的因素是嵌入装置如何快速地传输数据而不涉及CPU。在很多低功率手持产品中, 基本的I/O设备是与主处理器集成在一起的, 不需要主CPU总线扩展。但大多数新的设计不仅需要基本的I/O设备, 而且很多都采用广泛应用PC机标准以便主CPU总线扩展, 即PCI (外设部件互连) 总线。PCI总线标准于1992年由Intel公司开发, 它为CPU和板上外设之间提供方便的高速通信连接。此总线后来在PC工业中广泛用于扩展槽, 其工作频率为33MHz (rev2.1支持66MHz)。PCI对于连接到它上面的器件是具有即插即用能力的高速总线。理论上, 高达256个PCI功能器件可挂到一个PCI总线上。然而, 总线负载是一个问题, 一般每个总线用4到8个器件。PCI总线在全系统硬件资源管理方面具有灵活性。每一个PCI器件可以配置硬件资源, 如存储器和I/O空间。Compact PCI (PCI总线的一种) 正在进入工业和通信市场。这遵从Eurocard (用于Compact PCI卡到背板的连接) 的机械标准, 很适用于机架安装的

工业应用。CompactPCI也支持带电交换能力,使其可与工业总线如VME进行竞争。价廉的PCI芯片组和广泛应用的PCI外设可以促使PCI总线成为嵌入领域的事实上的总线标准。

PC104/104+基本上分别为PC ISA和PCI总线的改进型。PC104总线与ISA总线完全兼容的。顾名思义,它有一个104引脚连接器。PC104+为PC104形状因数中的PCI总线提供一种类似的标准。由于在PC104中的引脚被ISA信号所用,所以在卡的另一边的新的高密度120引脚连接器用于处理32位PCI信号。这些总线的出现有助于PC技术进入嵌入领域。一个好的用于网络启动装置的操作系统应支持绝大部分的广泛应用的通信装置和协议。Windows CE支持其每一种通信技术。

IrDA/FastIrDA

红外数据联盟(IrDA)是一个由150多个公司组成的联合体。IrDA提供一种价廉的无线、点到点、双向红外通信技术。它旨在用于小于1米的极短距离通信。IrDA有两个速度:低速运行于9.6~115kbits/s(简称IrDA);高速运行于1~4Mbits/s(即Fast IrDA)。高达16M bits/s的更高速度的正在开发。IrDA用于笔记本电脑、PDA、打印机、照相机等产品中。其他产品如复印机、投影机 and 游戏控制等也正在考虑采用。

2、USB

通用串行总线(USB)是由IBM、Compaq、Nortel、NEC、Intel和Microsoft公司开发的一种外设总线标准。它为所有USB外设提供一种通用的连接,其数据率为12Mbits/s。USB缆线是为适用于短距离(最长5米)而设计的。连接遵从树拓扑结构,在任何时间可连接127个器件而外设可以是带电交换的。USB缆线也把功率(+5V)分配给低功率外设。它为不能处理瞬间传输、又需要保证带宽和有限执行时间的应用提供同步通信。同步工作量可以是USB总线带宽的一部分或全部。USB特别适合于需要高数据率和易于即插即用的应用,如调制解调器、游戏控制、打印机、扫描仪和数字相机。需要保证带宽和有限执行时间的应用包括PC电话和其他语音及视频通信应用。除了这些新的多媒体设备外,USB也用于传统的I/O设备,如键盘和鼠标,其处理速度为低速(1.5Mbits/s)。Windows CE为USB提供支持。USB的系统软件由两部分组成:USBD(通用串行总线驱动器)和HCD(主控制器驱动器)。USBD由Microsoft提供而用USB器件驱动器实现高级功能。HCD模块提供到实际硬件(OHCD开路主控制器驱动器或UHCD通用主控制器驱动器)的接口。

3、Ethernet/Fast Ethernet

Ethernet(以太网)和Fast Ethernet(IEEE 802.3和802.3n)是最广泛应用的局域网络技术,旨在小区域(即一个办公室)范围连接计算机。Ethernet工作在10Mbits/s而Fast Ethernet工作在100M bits/s。两个协议的差别限于物理层和通信媒体。媒体存取规则是CSMA/CD(载波检测多路存取/冲突检测)。Windows CE通过其NDIS 4.0实现支持IEEE802.3小口驱动器。Ethernet卡可以在平台上或通过一个PCMCIA槽进行热插拔。

4、IEEE1394

IEEE1394是高速串行总线,其数据率为25~400Mbits/s。它起源于Apple Computer的FireWire总线,是作为通用外设串行总线而设计的,但它的应用重点转为所有类型的消费类设备如数字相机和扫描仪。缆线型1394总线可支持63个器件。器件之一变成总线管理者,与其他器件协调之后管理总线执行。缆线越长它能够处理的数据率就越低。一般长度为几米。IEEE1394和USB都是串行协议,然而USB和IEEE1394比其竞争技术有更大的互补性,USB属于低到中带宽,而IEEE1394属于中到高带宽。各种通信技术的比较见表6-1。

表6-1 各种通信技术比较

通信技术	最大通信距离	最大通信速度
USB	几米	12Mbits/s
IEEE1394	几米	40Mbits/s
Ethernet/Fast Ethernet	100米以上	100Mbits/s
IrDA/Fast IrDA	小于1米	4Mbits/s
Serial	10米以上	115Kbits/s
Parallel (EPP/ECP)	几米	2Mbytes/s

5、PC卡

PC卡通常用于移动计算和嵌入市场的通信和存储装置。1990年首先为存储器卡定为标准。该标准定义外设卡和主系统插座之间68引脚接口。有三个标准：Type I、II和III。做为标准的一部分也提供软件即插即用能力。软件标准由插座服务和卡服务组成，它们也小心地分配请求资源（中断，DMA通道等）。卡服务的作用类似于一个服务器，对请求应用程序和设备驱动器提供不同的服务。卡服务接口定义客户/服务器通信接口。插座服务为实际的插座提供硬件独立的软件接口。1996年PCMCIA（个人计算机存储器卡国际联合会）为PC卡制定了新的高性能总线标准，称之为Cardbus，这是一种PCI型总线，它用32位总线控制，可工作在33MHz。Windows CE为标准PCMCIA总线提供支持。

6、传统I/O

除新的高速通信设备外，对于像串行、并行和键盘端口的硬件接口仍存在设计需要，然而过去几年这些器件发展到要处理更高数据率。例如，标准Centronics并行端口现在被高速双向并行端口ECP（扩展通信端口）替代，后者所处理的数据率超过2Mbytes/s。Windows CE支持高速ECP并行端口通信，而用于ECP并行端口的一个实例驱动器包含在Window CE ETK中。

这里所讨论的总线结构和通信接口可由一个基准平台提供。为产品开发选择适当的基准平台是非常重要的，因为在大部分情况下它也是新产品的基准设计。在产品设计的早期阶段它也可提供概念样机机理的快速验证，而不用很大的投资。Microsoft开发一种称之为HARP（硬件结构基准平台）的新Windows CE基准平台规范，HARP以Compact PCI标准为基础。除处理器板外，HARP也规定一个6U Compact PCI I/O卡（RIO），它为Windows CE确证提供所需的全部I/O。在Compact PCI上需要一个Compact PCI背板来连接带I/O板的处理器板。HARP保证与Windows CE的硬件和软件兼容性，并提供灵活的连接测试环境。Windows CE不依赖于PCI总线的存在。事实上很多Windows CE产品不需要外部总线。

五、嵌入式处理器开发工具(Development Tools)

嵌入式处理器是一个复杂的高技术系统，要在短时间内掌握并开发出所有功能是很不容易的，而市场竞争则要求产品能够快速上市，这一矛盾要求嵌入式处理器能够有容易掌握和使用的开发工具平台。提高用户和程序员的时间-投入回报率。

从事嵌入式开发的往往是非计算机专业人士，面对成百上千种处理器，选择是一个问题，学习掌握处理器结构及其应用更需要时间，因此以开发工具和技术咨询为基础的整体解决方案是迫切需要的。好的开发工具除能够开发出处理器的全部功能以外，还应当是对用户

友好的。目前嵌入式系统的开发工具平台主要包括下面几类。

1. 实时在线仿真系统ICE(In-CircuitEmulator)

在计算机辅助设计非常发达的今天，实时在线仿真系统(ICE)仍是进行嵌入式应用系统调试最有效的开发工具。ICE 首先可以通过实际执行，对应用程序进行原理性检验，排除人的思维难以发现的设计逻辑错误。ICE 的另一个主要功能是在应用系统中仿真微控制器的实时执行，发现和排除由于硬件干扰等引起的异常执行行为。

此外，高级的 ICE 带有完善的跟踪功能，可以将应用系统的实际状态变化、微控制器对状态变化的反应、以及应用系统对控制的响应等以一种录像的方式连续记录下来，以供分析，在分析中优化控制过程。很多机电系统难以建立一个精确有效的数字模型，或是建立模型需要大量人力，这时采用 ICE 的跟踪功能对系统进行记录和分析是一个快而有效的方法。嵌入式应用的特点是和现实世界中的硬件系统有关，存在各种异变和事先未知的变化，这就给微控制器的指令执行带来了各种不确定性，这种不确定性只有通过 ICE 的实时在线仿真器才能发现，特别是在分析可靠性时要在同样条件下多次仿真，以发现偶然出现的错误。ICE 不仅是软件硬件排错工具，同时也是提高和优化系统性能指标的工具。高档 ICE 工具(如美国 NOHAU 公司的产品)是可根据用户投资裁剪功能的系统，亦可根据需要选择配置各种档次的实时逻辑跟踪器(Trace)、实时映象存储器(Shadow RAM)及程序效率实时分析功能(PPA)。

2. 高级语言编译器(CompilerTools)

C 语言作为一种通用的高级语言，大幅度提高了嵌入式系统工程的工作效率，充分发挥出嵌入式处理器日益提高的性能，缩短产品进入市场时间。另外，C 语言便于移植和修改，使产品的升级和继承更迅速。更重要的是采用 C 语言编写的程序易于在不同的开发者之间进行交流，从而促进了嵌入式系统开发的产业化。

区别于一般计算机中的 C 语言编译器，嵌入式系统中的 C 语言编译器要专门进行优化，以提高编译效率。优秀的嵌入式系统 C 编译器代码长度和执行时间仅比以汇编语言编写的同样功能程度长 5-20%。编译效率的不同，是区别嵌入式系统 C 编译器之间性能差别的重要根据之一。而 C 编译器残余的 5-10%效率差别，完全可以由现代微控制器的高速度、大存储器空间以及产品提前进入市场来弥补。

新型的微控制器指令速度不断提高，存储器空间也相应加大，已经达到甚至超过了目前的通用计算机中的微处理器，为嵌入式系统工程采用过去一直不敢问津的 C++语言创造了条件。C++语言强大的类、继承等功能更便于实现复杂的程序功能。但是 C++语言为了支持复杂的语法，在代码生成效率方面不免有所下降。为此，1995 年初在日本成立的 EmbeddedC++ 技术委员会经过几年的研究，针对嵌入式应用制订了减小代码尺寸的 EC++标准。EC++保留了 C++的主要优点，提供对 C++的向上兼容性，并满足嵌入式系统设计的一些特殊要求。

在嵌入式高级语言编译器方面处于领先地位的 Tasking 公司，是 EC++技术委员会成员之一，也是最先推出 EC++产品的公司。C/C++/EC++引入嵌入式系统，使得嵌入式开发和个人计算机、小型机等之间在开发上的差别正在逐渐消除，软件工程中的很多经验、方法乃至库函数可以移植到嵌入式系统。在嵌入式开发中采用高级语言，还使得硬件开发和软件开发可以分工，从事嵌入式软件开发不再必须精通系统硬件和相应的指令集汇编。

另一种高级语言 JAVA 的发展则具有戏剧性。JAVA 本来是为设备独立的嵌入式系统设计的、为了提高程序继承性的语言，但是目前基于 JAVA 的嵌入式开发工具代码生成长度要比嵌入式 C 编译工具差 10 倍以上。因此 EC++很可能在未来的一段时间内仍是嵌入式系统的主流。

3、源程序模拟器(Simulator)

源程序模拟器是在广泛使用的、人机接口完备的工作平台上,如小型机和 PC,通过软件手段模拟执行某种嵌入式处理器内核编写的源程序测试工具。简单的模拟器可以通过指令解释方式逐条执行源程序,分配虚拟存储空间和外设,供程序员检查;高级的模拟器可以利用计算机的外部接口模拟出处理器的 I/O 电气信号。不同档次和功能模拟器工具价格差距巨大。模拟器软件独立于处理器硬件,一般与编译器集成在同一个环境中,是一种有效的源程序检验和测试工具。但值得注意的是,模拟器毕竟是以一种处理器模拟另一种处理器的运行,在指令执行时间、中断响应、定时器等方面很可能与实际处理器有相当的差别。另外它无法和 ICE 一样,仿真嵌入式系统在实际应用系统中的实际执行情况。

六、嵌入式系统的软件开发平台

1、实时多任务操作系统(Real Time multi-tasking Operation System, RTOS)

这里只对实时操作系统做一个简单的介绍。有关实时操作系统更多更加详细的资料,将在下文专门阐述。

实时多任务操作系统(RTOS)是嵌入式应用软件的基础和开发平台。目前大多数嵌入式开发还是在单片机上直接进行,没有 RTOS,但仍要有一个主程序负责调度各个任务。RTOS 是一段嵌入在目标代码中的程序,系统复位后首先执行,相当于用户的主程序,用户的其它应用程序都建立在 RTOS 之上。不仅如此,RTOS 还是一个标准的内核,将 CPU 时间、中断、I/O、定时器等资源都包装起来,留给用户一个标准的 API(系统调用),并根据各个任务的优先级,合理地在不同任务之间分配 CPU 时间。

RTOS 是针对不同处理器优化设计的高效率实时多任务内核,RTOS 可以面对几十个系列的嵌入式处理器 MPU、MCU、DSP、SOC 等提供类同的 API 接口,这是 RTOS 基于设备独立的应用程序开发基础。因此基于 RTOS 上的 C 语言程序具有极大的可移植性。据专家测算,优秀 RTOS 上跨处理器平台的程序移植只需要修改 1-4%的内容。在 RTOS 基础上可以编写出各种硬件驱动程序、专家库函数、行业库函数、产品库函数,和通用性的应用程序一起,可以作为产品销售,促进行业的知识产权交流,因此 RTOS 又是一个软件开发平台。

2、RTOS的基本结构

RTOS 最关键的部分是实时多任务内核,它的基本功能包括任务管理、定时器管理、存储器管理、资源管理、事件管理、系统管理、消息管理(队列管理)、信号量管理等,这些管理功能是通过内核服务函数形式交给用户调用的,也就是 RTOS 的系统调用,或者叫做 RTOS 的 API。实时操作系统中的任务(Task)等同于分时操作系统中的进程(Process)的概念,是操作系统做调度最基本的单位。

3、RTOS是嵌入式系统的软件开发平台

RTOS 的引入,解决了嵌入式软件开发标准化的难题。随着嵌入式系统中软件比重不断上升、应用程序越来越大,对开发人员、应用程序接口、程序档案的组织管理成为一个大的课题。引入 RTOS 相当于引入了一种新的管理模式,对于开发单位和开发人员都是一个提高。基于 RTOS 开发出的程序,具有较高的可移植性,实现 90%以上的设备独立,一些成熟的通用程序可以作为专家库函数产品推向市场。

中国在嵌入式系统中的机会:

中国在计算机基础工业上落后于西方国家,在嵌入式处理器上也是如此。但是嵌入式系统面向应用的特点决定了处理器应用开发的产值要占有整个嵌入式工业的大部分,而

且将嵌入式处理器与具体应用结合这种知识创新，只能由精通应用系统的用户来完成。因此中国在嵌入式系统方面存在着相当大的发展机会。中国已经有 10 万余名单片机开发工程师，其中很多人都是在资料和信息有限的条件下通过实践，精通了单片机，并研制出了自己的产品。但是和国外的开发相比，开发手段和水平还相对较低，标准化程度不够、重复劳动较多。这些问题主要是由于单片机开发中缺乏工程化、标准化管理，缺少行业联合，在引入 RTOS 和嵌入式系统软件工程管理后可望得到很大的改变。

嵌入式系统是信息产业走向二十一世纪知识经济时代的最重要的经济增长点之一，这是一个不可垄断的工业，对中国的信息产业来说充满了机遇和挑战。

嵌入式工业的基础是以应用为中心的芯片设计和面向应用的软件开发。实时多任务操作系统 (RTOS) 进入嵌入式系统工业的意义，不亚于历史上机械工业采用三视图后的发展，对嵌入式软件的标准化和加速知识创新是一个里程碑。这两点应特别引起中国信息产业界的关注。

七、实时操作系统的一些主要概念

1、实时系统与实时操作系统

实时的含义是指在规定的时间限内能够传递正确的结果，迟到的结果就是错误。实时系统并非是指“快速”的系统，实时系统有限定的响应时间，从而使系统具有可预测性。实时系统又可以分为“硬实时系统”和“软实时系统”。二者的区别在于：前者如果在不能满足响应时限、响应不及时或反应过早的情况下都会导致灾难性的后果（如航空航天系统）；而后者则在不满足响应时限，系统性能退化，但并不会导致灾难性的后果（如交换系统）。

实时操作系统是指具有实时性，能支持实时控制系统工作的操作系统。实时操作系统的首要任务是调度一切可利用的资源完成实时控制任务，其次才着眼于提高计算机系统的使用效率，其重要特点是要通过任务调度来满足对于重要事件在规定的时间内做出正确的响应。

实时多任务操作系统与分时多任务操作系统有着明显的区别。具体的说，对于分时操作系统，软件的执行在时间上的要求，并不严格，时间上的延误或者时序上的错误，一般不会造成灾难性的后果。而对于实时操作系统，主要任务是对事件进行实时的处理，虽然事件可能在无法预知的时刻到达，但是软件上必须在事件随机发生时能够在严格的时限内做出响应（系统响应时间）。即使是系统处在尖峰负荷下，也应如此——系统时间响应的超时就意味着致命的失败。另外，实时操作系统的重要特点是具有系统的可确定性，即系统能对运行情况的好坏和最坏等的情况能做出精确的估计。

2、实时操作系统的发展过程

实时操作系统 (RTOS) 的研究是从六十年代开始的。从系统结构上看，RTOS 到现在已经经历了如下三个阶段：

1) 早期的实时操作系统

早期的实时操作系统，还不能称为真正的 RTOS，它只是小而简单的、带有一定专用性的软件，功能较弱，可以认为是一种实时监控程序。它一般为用户提供对系统的初始化管理以及简单的实时时钟管理，有的实时监控程序也引入了任务调度及简单的任务间协调等功能，属于这类实时监控程序的有 RTMX 等。这个时期，实时应用较简单，实时性要求也不高。应用程序、实时监控程序和硬件运行平台往往是紧密联系在一起。

2) 专用实时操作系统

随着应用的发展，早期的 RTOS 已越来越显示出明显的不足了。有些实时系统的开发者为了满足实时应用的需要，自己研制与特定硬件相匹配的实时操作系统。这类专用实时操作系统在国外称为 Real-Time Operating System Developed in House。它是在早期用户为满足自身开发需要而研制的，它一般只能适用于特定的硬件环境，且缺乏严格的评测，移植性也不太好。属于这类实时操作系统的有 Intel 公司的 iMAX86 等。

3) 通用实时操作系统

在各种专用 RTOS 中，一些多任务的机制如基于优先级的调度、实时时钟管理、任务间的通信、同步互斥机构等基本上是相同的，不同的只是面向各自的硬件环境与应用目标。实际上，相同的多任务机制是能够共享的，因而可以把这部分很好地组织起来，形成一个通用的实时操作相同内核。这类实时操作系统大多采用软组件结构，以一个个软件“标准组件”构成通用的实时操作系统，一方面，在 RTOS 内核的最底层将不同的硬件特性屏蔽掉；另一方面，对不同的应用环境提供了标准的、可剪裁的系统服务软组件。这使得用户可根据不同的实时应用要求及硬件环境选择不同的软组件，也使得实时操作系统开发商在开发过程中减少了重复性工作。这类通用实时操作系统，有 Integrated System 公司的 pSOSystem、Intel 公司的 iRMX386、Ready System 公司（后与 Microtec Research 合并）的 VRTX32、Wind River Systems 公司（位于美国加州 Alameda 市）的 VxWorks、Accelerated Technology Inc 的 Nucleus PLUS 等。它们一般都提供了实时性较好的内核、多种任务通信机制、基于 TCP/IP 的网络组件、文件管理及 I/O 服务，提供了集编辑、编译、调试、仿真为一体的集成开发环境，支持用户使用 C、C++ 进行应用程序的开发。

3、实时操作系统的主要研究方向

实时操作系统经过多年的发展，先后从实模式进化到保护模式，从微内核技术进化到超微内核技术，在系统规模上也从单处理器的 RTOS 发展到支持多处理器的 RTOS 和网络 RTOS，在操作系统研究领域形成了一个重要分支。

今后，RTOS 研究方向主要集中在如下几个方面：

1) OS 的标准化研究

如今国外的 RTOS 开发商有数十家，提供了上百个 RTOS，它们各具特色。但这也给应用开发者带来难题，首先是应用代码的重用性难，当选择不同的 RTOS 开发时，不能保护用户已有的软件投资，RTOS 的标准化研究越来越被重视。美国 IEEE 协会在 UNIX 的基础上，制定了实时 UNIX 系统的标准 POSIX 1001.4 系列协议，但仍有许多工作还待完成。

2) 多处理器结构 RTOS、分布式实时操作系统和实时网络的研究

实时应用的飞速发展，对 RTOS 的性能提出了更高的要求。单处理器的计算机系统已不能很好地满足某些复杂实时应用系统的需要，开发支持多处理器结构的 RTOS 已成为发展方向，这方面比较成功的系统有 pSOS+m 等。至于分布式 RTOS，国外一些 RTOS 厂家虽已推出部分产品，如 QNX、Chorus、Plan 9 等，但分布式实时操作系统的研究还未完全成熟，特别是在网络实时性和多处理器间任务调度算法上还需进一步研究。

3) 集成的开放式实时系统开发环境的研究

RTOS 研究的另一个重要方向是集成开发环境的研究。开发实时应用系统，只有 RTOS 是不够的，需要集编辑、编译、调试、模拟仿真等功能为一体的集成开发环境的支持。开发环境的研究还包括网络上多主机间协作开发与调试应用技术的研究、RTOS 与环境的无缝连接技术等。

4、实时操作系统中的几个重要的评价指标

RTOS是操作系统研究的一个重要分支，它与一般商用多任务OS如Unix、Windows等有共同的一面，也有不同的一面。对于商用多任务OS，其目的是方便用户管理计算机资源，追求系统资源最大利用率；而RTOS追求的是调度的实时性、时间响应时间的可确定性、系统的可靠性的可靠性。评价一个实时操作系统一般可以从任务调度、内存管理、任务通讯、内存开销、任务切换时间、最大中断禁止时间等几个方面来衡量。

1) 任务调度机制：

RTOS的实时性和多任务能力在很大程度上取决于它的任务调度机制。从调度策略上来讲，分优先级调度策略和时间片轮转调度策略；从调度方式上来讲，分可抢占、不可抢占、选择可抢占调度方式；从时间片来看，分固定与可变时间片轮转。

在大多数商用的实时系统中，为了让操作系统能够在有突发事件时，迅速取得系统控制权以便对事件作出反应，所以大都提供了“抢占式任务调度”的功能，也就是操作系统有权主动终止应用程序（应用任务）的执行，并且将执行权交给拥有最高优先级的任务。

下面简单介绍两种可以作出精确描述实时应用的时间测定正确性的著名的算法：

速率单调：在工作量由一组定期任务组成的应用中，每个任务的执行时间定长，这种速率单调调度算法能够保证其可调度性。速率单调调度算法简单地说就是越是高频地任务安排越高地优先级。因此，在系统中，最高频的任务具有最高的优先级。如果一个应用本来就是就全是定期地或者能够全部变成定期地，那么开发人员实在是太幸运了，因为可以使用速率单调调度算法，并且可以完全正确地断言这个应用能够满足时限。

时限驱动：对于一个由定期和不定期任务混合或者任务的执行时长随着时间变化的应用，可以使用时限驱动算法。这个算法的准则是下一个要安排执行的任务是一个时限最早的任务，该任务完成之后，下一个时限最早的任务被选择调度和执行。

2) 内存管理：如同分时操作系统一样，实时操作系统使用内存管理单元（MMU）进行内存管理。实时操作系统内存管理模式可以分为实模式与保护模式。目前主流的实时操作系统一般都可以提供两种模式，让用户根据应用自举选择。

3) 最小内存开销：

RTOS的设计过程中，最小内存开销是一个较重要的指标，这是因为在工业控制领域中的某些工控机（如上下位机控制系统中的下位机），由于基于降低成本的考虑，其内存的配置一般都不大，例如康拓5000系列5185板，其基本内存配置仅为256K SRAM+128K EPROM，而在这有限的空间内不仅要装载实时操作系统，还要装载用户程序。因此，在RTOS的设计中，其占用内存大小是一个很重要的指标，这是RTOS设计与其它操作系统设计的明显区别之一。

4) 中断禁止时间与中断延迟事件：

当RTOS运行在核心态或执行某些系统调用的时候，是不会因为外部中断的到来而中断执行的。只有当RTOS重新回到用户态时才响应外部中断请求，这一过程所需的最大时间就是中断禁止时间。

中断延时时间是指系统确认中断开始直到执行中断服务程序的第一条指令为止整个处理过程所需要的时间。实时操作系统的中断延迟时间由下列三个因素决定：

- 处理器硬件电路的延迟时间，通常这个时间可以忽略。
- 实时操作系统处理中断并将控制权转移给相关处理程序所需要的时间。
- 实时操作系统的中断禁止时间。

为了缩短系统对于中断请求的响应时间——中断延迟时间，大多数商用实时操作系统都采用了“可中断式”的核心程序，也就是说，当中断发生时，即便正在执行的是核心服务函数，实时操作系统也能够保证会在一定的时间（也就是实时操作系统厂家公布的实时操作系统的中断延迟时间）内，调用恰当的中断服务例程。当然也有相当多的实时操作系统，例如实时 Linux，采用非抢占式的核心程序，也就是执行中的系统核心服务函数不能被其它的程序所中断（注意这与上文所讲述的非抢占式的任务调度不相同，前者的对象一般是应用程序或任务）。在这类系统中，如果系统核心函数服务的对象是某个低优先级的任务，那么即使提出中断请求的是应该拥有更高优先级的任务，系统都必须等到目前这个核心函数结束后，才能将控制权转移给等待中的高优先级任务。图 7-1 表示了抢占式核心和非抢占式核心之间对于中断响应的可能的差别。

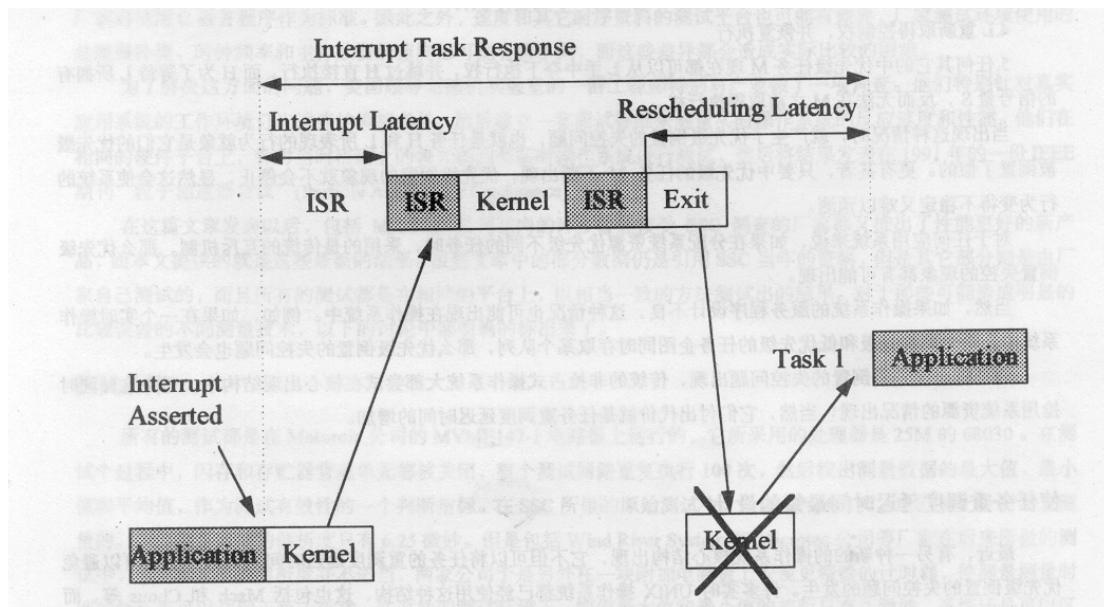


图 7-1：非抢占式核心的操作系统的中断延迟时间一般要比抢占式核心长

5) 任务切换时间：

当由于某种原因使一个任务退出运行时，RTOS保存它的运行现场信息、插入相应队列、并依据一定的调度算法重新选择一个新任务使之投入运行，这一过程所需时间称为任务切换时间。更准确地说，任务切换时间是实时操作系统将控制权从一个任务的执行中取回，然后交给另外一个任务所需要的时间。它包括保存目前正在执行任务的现场信息所需要的时间、RTOS决定下一个调度任务所需的调度时间以及RTOS把另外一个任务调入系统执行所需要的时间。

上述几项中，最大中断禁止时间和任务切换时间是评价一个RTOS实时性最重要的两个技术指标。

5、实时操作系统的功能

实时操作系统应具有如下的功能：

1) 任务管理：

分时操作系统中的基本调度单位一般是进程（或者线程），而对于实时操作系统，操作系统内核调度的基本单位就是任务。任务一般由任务控制块、程序区、数据区、堆栈区组成，对

于多数实时操作系统来说,堆栈一般又分为系统堆栈和用户堆栈,系统堆栈用于任务做系统调用访问系统核心时用到的堆栈,把它从用户堆栈中独立出来,是为了保证系统核心的安全性。任务的驱动一般是基于消息或者事件的,即任务的设计是按照依次处理可能接收到的消息和事件,周而复始轮询循环的。实时操作系统中的任务有四种状态:运行(Executing),就绪(Ready),挂起(Suspended),冬眠(Dormant)。

运行:获得 CPU 控制权。

就绪:进入任务等待队列。通过调度转为运行状态。

挂起:任务发生阻塞,移出任务等待队列,等待系统实时事件的发生而唤醒。从而转为就绪或运行。

冬眠:任务完成或错误等原因被清除的任务。也可以认为是系统中不存在了的任务。

系统中只能有一个任务在运行状态。各任务按级别通过时间片分别获得对 CPU 的访问权。

实时操作系统一个最重要的功能就是多任务管理和基于优先级的任务调度。前文已经讲述了实时操作系统的调度算法,后面的实时操作系统实例讲解部分将讨论各个实时操作系统的任务调度算法。

2) 任务间同步和通信:

目前,主要的实时操作系统的任务间同步和通信的机制有:消息、事件、信号量,而部分实时操作系统仍然在沿用邮箱机制,另外一些实时操作系统提供了共享内存的任务间通信机制。

消息机制的概念和分时操作系统没有差别,其基本思想是任务通过系统公用的数据交换区(包括私有消息缓冲区和共用消息缓冲池)来交互任务间需要通信的信息。消息机制的系统调用一般包括消息队列的创建(q_create)、删除(q_delete)、接收消息(q_receive)、发送消息(q_send)、广播消息(q_broadcast)、紧急消息(q_urgency)。目前,大多数实时操作系统支持的消息队列既可以是定长的,也可以是变长的。

事件机制适用于任务间需要同步,并且通信的数据量不大的情况,一般说来,任务之间的事件通信机制是可以覆盖的,即任务 A 先后发送三次事件给任务 B,如果 B 还没有来得及处理的话,任务 B 只需要处理一次事件就行了。事件机制的系统调用一般包括发送事件(ev_send)、接收事件(ev_receive)。目前大多数实时操作系统支持的 16—32 个事件。

如同分时操作系统中的信号量一样,实时操作系统提供的信号量机制也是为了解决对于临界资源共享的加锁机制。信号量机制提供了信号量的创建(sm_create)、信号量的删除(sm_delete)、信号量的 P 操作(sm_p)、信号量的 V 操作(sm_v)。实时操作系统与分时操作系统在信号量机制上有一个明显的区别,那就是优秀的商用实时操作系统要解决信号量机制的优先级倒置的问题。有关优先级倒置,后面将有专门论述。

3) 内存管理:和分时操作系统一样,实时操作系统会借用 CPU 的内存管理单元(MMU)来完成内存管理,实时操作系统内存管理模式可以分为实模式与保护模式。目前主流的实时操作系统一般都可以提供两种模式,让用户根据应用自举选择。一般说来,实时操作系统的内存管理,还有对于内存的优化分配,以尽量减少整个系统的内存占有量的要求。

4) 实时时钟服务:目前,商用的实时操作系统在硬件的硬时钟中断的基础上,提供了实时时钟服务。实时时钟是系统调度的基础,也是系统定时服务器的基础。实时时钟服务一般包括定时唤醒(tm_wkafter 或者 tm_wkwhen)、定时事件(tm_evafter 或者 tm_evwhen)机制。另外部分优秀的实时操作系统提供了定时消息机制,即应用任务(比如说任务 A)向系统定时服务器申请定时器,当定时时间到后,定时服务器返回任务 A 一条消息。相应的系统调用一般有定时器申请(tm_start),定时器删除(tm_delete)、定时器重置(tm_restart),定时

消息的接收一般采用消息队列的接收机制(q_receive)。是否提供灵活的、高精度的定时服务是衡量实时操作系统功能完整性的一个重要指标。

5) 中断管理服务：如同分时操作系统一样，中断管理服务是操作系统的核心和基本的功能。实时操作系统的中断管理有自己的特殊的要求，那就是中断处理程序要更加短小、精悍，以减少中断禁止时间和中断延迟时间。

6、优先级倒置发生的条件和解决途径

如前文所述，抢占式的结构在降低中断延迟时间上有很大的优势，但是为什么还有许多流行的实时操作系统采用非抢占式的核心结构？原因之一是为了避免优先级倒置失控的问题发生。要了解这个问题，请看如下的情况：

高优先级任务 H 和低优先级任务 L 要共享一片内存 Y，并且都要进行写操作，为了保证数据的完整性，它们需要通过一个信号量 S 来保证对于内存块的互斥访问。任务 M 的优先级介入 H 和 L 之间。

- (1)、低优先级的任务 L 取得了信号量 S 的所有权——即做了 P 操作，进入关键段，但还没有做 V 操作；
- (2)、此时属于另一个高优先级的任务 H 的中断事件发生，实时操作系统核心通过核心调度，将任务 L 切换下去，将任务 H 置为运行态；
- (3)、H 开始执行，到中途需要访问共享内存 Y，必须先对 S 做 P 操作，因为此信号量目前还没有恢复，于是 H 阻塞在信号量 S 上。
- (4)、L 重新取得控制权，并恢复执行；
- (5)、此时属于另外一个中优先级任务 M 的中断事件发生，因为 M 的优先级比 L 高，于是实时操作系统核心再次将 L 切换下去，任务 M 取得 CPU 的执行权，于是优先级比 H 低的 M 可以跳过 H 直接执行，而 H 为了等待 L 所拥有的信号量 S，反而无法在 M 之前取得执行权。

当出现这种情况时，就产生了优先级倒置的失控问题，也就是任务 H 和 M 所表现的行为就象是它们的优先级被倒置了似的。更有甚者，只要中优先级的任务 M 不断出现，优先级倒置的现象就不会停止，显然这会使系统的行为变得不稳定又难以预测。

对于任何应用系统来说，如果在优先级不同的任务之间分配系统资源的时候采用的是传统的互斥机制，那么优先级倒置失控的现象都有可能出现。当然，如果操作系统的服务程序设计不良，这种情况也可能出现在操作系统中。例如，如果在一个实时操作系统中，两个高优先级和低优先级的任务企图同时存取某个队列，那么优先级倒置的失控问题也会发生。为了防止优先级倒置的失控问题出现，传统的非抢占式实时操作系统大都尝试在核心结构中避免这种同时抢用系统资源的情况出现；当然，它们付出代价就是任务重调度延迟时间的增加。

抢占式实时操作系统是如何避免优先级倒置的呢？思路有两个：

(1) 利用非抢占的低级服务函数来支援高级的抢占式的服务函数，以避免优先级倒置的失控现象发生。由于这些非抢占式的服务函数的处理都很简单，执行的速度也快，因此可以降低对于任务重调度的延迟时间的影响。许多实时 UNIX 操作系统都已经使用这种结构，这也包括 Mach 和 Chous 等，Microtec 公司的高性能通用实时操作系统核心 VRTXsa 也采用这种结构。

(2) 采用“优先级继承”的技术。以前面提及的 H、L 和 M 任务交互为例，若在其中加入一种信号机制，则整个处理过程就变成如下：

- (1)、低优先级任务 L 锁定优先级继承信号量 S；

- (2)、另一个高优先级的任务 H 出现，并从 L 手中取得执行权；
- (3)、H 尝试锁定 S，不成功被阻塞；操作系统调度核心取得控制权；
- (4)、调度核心让 L 恢复执行，并通过 S 让 L 暂时继承 H 的优先级；
- (5)、其它中优先级的任务 M 出现，但是无法从 L 手中取得执行权；
- (6)、L 继续执行，直到它将 S 的所有权放弃给 H，此时调度核心将恢复 L 到原来的低优先级。
- (7)、H 恢复执行；
- (8)、H 执行完毕后，M 执行。

采用优先级继承机制的抢占式的操作系统核心，可以避免优先级倒置的失控问题发生。目前，已经有部分操作系统提供了有优先级继承机制的信号量的系统调用，即开发者开发实时应用软件时可以轻易地透过这一机制，将优先级倒置地问题控制住。例如实时操作系统 pSOSystem 3.0 提供了 Mutex semaphores 机制。

7、BSP

BSP 的全称是 Board support packages，即板支持包，说的简单一点，就是一段启动代码，和 PC 机主板的 BIOS 的地位和作用差不多，但是一般来说，BSP 提供的功能就差得太多。编写 8031 等 8 位单片机软件时前面要有一小段程序完成设置栈指针、软复位、中断屏蔽等操作，称这短程序为 BSP 其实也是可以的。

实时操作系统的 BSP 一般复杂一点，但一般也是设临时栈指针，之后初始化寄存器（控制外围器件如 RAM 条，控制 I/O 口的寄存器）……，再之后是程序跳到实时操作系统内核入口。我们用如下的表来对比实时操作系统的 BSP 与分时操作系统的 BIOS 之间的差别。

表 7-1：BSP 与 BIOS 之间的差别

driver	driver
rtos	OS(win9x, winnt..)
^	^
BSP	BIOS
^	^
实验板	主板

PC 机的 BIOS 和操作系统比较独立，BSP 和嵌入实时操作系统关系也差不多，只不过写 BSP 时用一下实时操作系统为你定义好的寄存器比较方便。实时操作系统的 driver 和 PC 的 driver 差不多，一般的 RTOS 定义了一套自己的 driver 接口，照着接口写 driver 和写 PC 的 driver 差不多；不过由于 RTOS 灵活性较大，和硬件较紧密，开发者完全可以抛开它的接口自己写，这和 PC 机区别可能就大了一点。

8、实时操作系统的标准化—— μ ITRON

操作系统（OS）技术规范差别之大及使用的微控制器（MPU）品种之繁多，使实时操作系统（RTOS）的标准化成了一个大问题。另外，低档微控制器硬件资源的欠缺，也使软件开发成为难题。设在东京的 Industrial TRON(the RTOS Nucleus，实时操作系统中心)或叫 ITRON（工业实时操作系统中心），组织制订嵌入式系统 RTOS 的标准。该组织于 1987 年发布了第

一个 ITRON 规范，即 ITRON1。其后，又着手开发针对 8 位和 16 位 μ C、有较小的功能集的 μ ITRON 规范和用于 32 位处理器的 ITRON2 规范，并于 1989 年发布了 μ ITRON 和 ITRON2 两个技术规范。

μ ITRON 规范是 ITRON 规范高度伸缩后的一种版本。设计师们不仅在 8 位 μ C 上而且也在 16 位和 32 位 μ C 上实现了 μ ITRON 规范 OS。ITRON 的创始人是想把嵌入式系统的 RTOS 规范标准化，他们不得不在优化硬件系统和提高软件效率之间权衡。

简单地说，用 μ C 构置的系统，由于硬件资源有限，只有在能从可利用的硬件中获得最大收益时，才会使用 RTOS。同时，使用 RTOS 也只是提高了软件的效率。但是，如果想在 RTOS 提供的更高的抽象服务层上工作，你会发现内核操作的运行时间是硬件功能的大障碍。

当然，探讨嵌入式系统中如何在硬件优化与软件效率之间折衷考虑，是一个至关重要的课题。例如，在一个 8 位的微控制器系统里，一些高级功能，例如存储器库管理功能，几乎都用不上，但却要增加系统软件的开销，并使系统功能降低。在一个以 32 位微控制器为基础的系统里，这些高级功能通过 RTOS 就会提高软件效率。

ITRON 在设计其 OS 规范时，就考虑到要最大限度地发挥微控制器内所含硬件的功能。尤其突出的是，ITRON 规范对应该通过微控制器硬件层次结构标准化的那些特征标志，和应该根据硬件及其功能性质优化判定的那些成份，都做了明确的区分。

此外， μ ITRON 规范也提到，嵌入式系统对目标代码的兼容性并没有多大需要。另一方面，从学习的角度看，当命名系统调用或其功能时，兼容性却是必不可少的。与其他的 RTOS 不同，ITRON 对各种特征标志都进行了标准化，如任务调度规则、系统调用名称和功能、参数名称（它们的阶级及意义），以及代码（名称及意义）。影响实时功能的问题，如中处理和参数的长度，并未标准化。

根据 ITRON 规范，把由内核程序提供的那些功能尽可能地相互独立，使每个应用程序只选择它需要的那些功能。针对这个目标，大多数 μ ITRON 规范的内核都是以程序库格式提供的。你先选择内核功能，运行时再把它们链接到应用程序上。这样，系统就只纳入那些需要的功能。此外，每次系统调用只提供一个单一的功能，以便于只纳入那些需要的功能。

μ ITRON 规范包括许多特性，其中包括处理器的层次结构不是虚拟的。虽然 ITRON 规范的 OS 运行在不同的处理器上，但是标准并没有把不需要的运行条件虚拟化，因此，可以使处理功能更高。

μ ITRON 规范也规定了高速响应问题。影响实时应用中响应时间的那些因素，一般都与作业转换功能和中断处理例行程序有关。ITRON 规范规定，在规定的高速作业转换时间上，可以将寄存器换出。也可以在发生外部中断时，旁路 RTOS 以启动运行一个中断处理程序。根据 NEC 公司 V800 系列工具高级技术应用工程师 David Easler 的观点， μ ITRON 处理内核程序调用的方式有利于加快响应时间。他还认为， μ ITRON 与其他 RTOS 之间的基本差别在于，其他 RTOS 用软件中断来自内核程序库的调用功能； μ ITRON 则是规定使用标准的 C-格式调用。

把层次结构与执行过程分开， μ ITRON 所提供的方法就优于其他的 RTOS；如果用户要改变既定层以下的执行过程，你可以继续使用既定层以上的规范而不用改动。同样， μ ITRON 也把自己限定为层次结构式的规范。作为开发者，你有充分的自由去运行你选定的应用程序。这样， μ ITRON 标准就为公平的技术竞争提供了一个层次施展的空间。

为了使 ITRON 规范易于学习和消化，开发者作了大量艰苦的努力。例如，ITRON 委员会系统地定义了 ITRON 规范的系统调用名称和功能，使它们便于记忆。所有的 ITRON 系统调用都是七个或八个字符长，形式分别为 xxx-yyy 和 zxxx-yyy。xxx 部分代表运算方法，yyy 则代表运算目标（图 1）。在八个字符的系统调用指令中，Z 代表从无 Z 的七个字符系统调用指

令中导出的系统调用指令。

Easler 认为,学习 μ ITRON 和学习 C 语言一样并不复杂,因为开发者用不着变换处理器,或学习其他 RTOS 厂家调用各种信箱序列,或其他 RTOS 如何处理缓冲器。

根据 ITRON 技术规范,“任务(task)”这个术语是指并行处理的一个单元。在单一任务内的诸多程序是按顺序执行的;多任务的诸多程序则是并行执行的。ITRON 规范采用优先权调度方式。每项任务都被指定一个优先级;优先数值越小,优先级别越高。ITRON 规范采用一个叫做任务身份证(ID)的号码指定作业。任务控制块(TCB)含有用以管理作业的信息。ITRON OS 规范通过系统调用指令按 TCB 的数值来处理设定和修改。

在 μ ITRON 的概念里,资源是执行一项任务所需要的某种要素,各种资源是执行一项任务所需要的不同要素。所谓资源包括硬件(如处理器、存储器和 I/O 器件)和软件(如各种文档和程序)。ITRON 规范提供专门用于指定资源的同步和通信功能。

根据 μ ITRON 规范包括五种作业状态:运行、准备、等待、静止和非存在(虚拟)状态。为了便于程序的开发,ITRON 规范对任务用以控制其本身的系统调用指令与作业用以控制其他任务的系统调用指令,作了明确的区分。这种方法阐明了作业状态的过渡,并使系统调用简明易懂。

根据 ITRON 规范,一个目标就是系统调用指令要加以处理,也是你用 ID 号做出识别的任何项目,例如一项任务。除各种任务外,目标还包括同步和通信机制,如存储器库、事件标记和信箱。OS 对每种目标提供目标管理表(例如,对各种任务提供 TCB)。

μ ITRON 3.0 规范把系统调用分为三级:R 级(Level R),即要求级;S 级(Level S),即标准级和 E 级(Level E),即扩展级。除这三个级别外,还有 C 级(Level C),用于处理与 CPU 相关的系统调用。R 级涉及的是实施 μ ITRON 3.0 规范所需要的各种功能。S 级覆盖实现实时、多任务 OS 的各项基本功能。E 级包括各种附加的和扩展的功能。这个级包括目标生成和删除功能、各种聚集功能、存储器库和定时器处理程序。与 CPU 相关的 C 级在 CPU 的基础上,提供要关硬件的各种功能。

μ ITRON 规范的内核在低档微控制器中,尤其能够发挥作用。这些器件以前受存储器和执行速度的限制,不能使用 RTOS。 μ ITRON 处理进程提供了相当大的优势,而且在 8 位和 16 位微控制器中,正日益变成首屈一指的标准内核规范。

对 μ ITRON 的支持也来美国的部分厂家。例如,Cygnus 公司除了本机内核应用编程接口(API)外,还提供该公司带 μ ITRON 3.0 兼容 API 的嵌入式 Cygnus OS(eCos)。有了 μ ITRON 兼容的 API,eCos 就能植入并反复使用 μ ITRON 应用程序。值得注意的是,eCos 是一个开放式的源 OS,而且 Cygnus 公司还将提供 eCos 的源代码、GNU 编译程序及排障调试程序(www.sourceware.cygnus.com)。工业界的许多专家认为,Cygnus 公司的这一举措会造成 μ ITRON 标准的迅速扩展。

NEC 公司提供与 μ ITRON 兼容的 RX850 RTOS,与该公司的 NEC V850 / E 系列微处理器一块儿工作。RX850RTOS 工具组包括一个 RTOS 任务识别调试程序,可显示状态的过渡。用 RX850 RTOS 工具组,通过验证事件状态的过渡和观察中断事件定时及功能线索持续时间,对系统的功能进行分析并画出轮廓曲线。利用 RD850 任务调试程序,还可以观察各种 OS 资源,如各种任务和信号量,以及系统缓冲器和队列。当你将其用于运行在 25MHz 的 NEC V851 微处理器上时,RTOS 内核的二进制代码占用 3k~8k 字节。其作业转换时间最快为 1~3 μ S,最慢为 5 μ S。RX850 是许可证经营的 RTOS,所以你要装多少器件,就付多少相应的费用。RX850 的起价为 800 美元。

Accelerated Technology Inc(ATI)将其实时内核程序、Nucleus Plus,纳入 μ ITRON 接口标准,使其用户能自由地将其传统的应用程序转给任务数号的 CPU,还有免费提供专利

源代码的好处。该公司计划将其内核程序植入日立公司 SH 系列的处理器。

许多迹象表明， μ ITRON 有潜力变成嵌入式 RTOS 的世界标准。Cygnus Solutions 公司工程部主任 Paul Beeskens 认为，ITRON 应摆脱其仅瞄向日本的做法，更国际化一些，则很有机会真的成为世界标准。Beeskens 还认为，“ITRON 应当提供可广泛选择的英文文件和技术信息”。即 μ ITRON 委员会必须扩展以纳入更广泛的国际支持者，例如，标准用英文开发，而且在开发时就应该能提供。

NEC 公司的 Easler 认为，美国的嵌入式市场对高功能、存储器覆盖区小的 OS，没有通用的标准。对较大覆盖区的 OS 有一个标准，如 Posix 需要大量使用存储器的许多相符数据和 API。（ μ ITRON RTOS 适合 16k 字节的存储器）。同时，这个标准也定义了调用各种功能的方式，以及内核程序运行作业的方式。Easler 还认为：“虽然这种标准化是宽松的，但 μ ITRON 的定时是决定性的，而 Posix 却并不严格”。Grape Systems 公司销售工程师 Koji Mugita 认为：“通过平台可快速移植和使用方便，是 μ ITRON 的最大优点。由于嵌入式软件行业非常分散，在全世界有 50 多个软件厂家，有一个像 μ ITRON 这样的标准，就会使嵌入式软件产品的开发者，稍加修改就可以把软件用于以后的应用中，而无须考虑他们使用的是什么样的微处理器。”

μ ITRON 的主要缺点是效率欠佳，另外就是主流的通用实时操作系统还没有支持它的迹象。

八、实时系统需求分析、设计方法综述以及实时程序设计

1、实时系统设计的一些基本问题

实时系统在工业、商业和军事等领域都有非常广泛的用途，并且已经有很多实际的应用。一般来说，实时系统通常是比较复杂的，因为它必须处理很多并发事件的输入数据流，这些事件的到来次序和几率通常是不可预测的，而且还要求系统必须在事先设定好的时限内做出相应的响应。

那么，是不是响应时间在多少毫秒或多少微秒以内的系统就是实时系统，而超出这个时限的就不算呢？事实上，实时系统并非是指“快速”的系统，实时系统有限定的响应时间，从而使系统具有可预测性。实时系统又可以分为“硬实时系统”和“软实时系统”。二者的区别在于：前者如果在不满足响应时限、响应不及时或反应过快的情况下都会导致灾难性的后果；而后者则在不满足响应时限，系统性能退化，但并不会导致灾难性的后果。

实时系统具有以下一些特点，从而区分于其他系统：

嵌入式系统：实时系统通常是嵌入式的系统，也就是由封装好的软件系统控制与其相关的硬件。

与外部环境交互：实时系统通常需要与外部环境进行交互，例如，可以控制机器及生产过程，或者监控化学反应并随时汇报危急情况，这种情况通常需要从外部接收数据并提供输出和控制外部环境。

时限响应：实时系统对紧急事件必须在确定的时限那给出响应，无论现在的系统是否处于峰值处理状态。

实时控制：实时系统经常包括实时控制，从接收到的输入数据中做出控制决策。

“反应”系统：很多实时系统都是“反应”的系统，也就是说，由事件驱动并且必须对外界事件进行响应。

并发处理：实时系统通常需要处理不同类型的任务：一类是周期性任务——按照固定时

间间隔执行的任务；另外一类任务是非周期性的，常常是随机性任务——要求在任务出现的任意时刻能进行相应处理。一般对非周期性任务的处理都有响应要求，这样当新的事件到来时，即使系统正在处理别的任务，也必须及时响应，从而导致需要并发地处理多个相互竞争地请求。

实时系统与周围环境有密切联系，因而可能造成处理机负荷过载。在过载情况下，一般允许系统性能合理降级。由于资源有限，导致有些任务必须等待处理，甚至造成任务丢失。因此，最简单地实际处理方案可以将任务按重要性分为关键、基本、不重要等级别。关键地任务必须达到时间限期要求，需要优先处理；基本任务也有时间限期，但达不到要求关系不大，则可稍后处理；不重要地任务理论上可以无限期等待。这样，许多情况下只要能够做到关键任务集的规模合理，往往可以在有限资源条件下，保证关键任务完成的前提下达到系统目标。但是，在一些复杂的环境中，在无法用这种简单办法来解决问题时，就需要仔细分析系统的特性，进行有效设计以保证所开发的实时系统的目标。而实时系统的设计有自己的许多如上所述的特点，一般系统的设计或开发方法尚不能解决实时系统中的并发处理、时限响应等问题。所以，对于实时系统，有必要研究其专门的一些分析、设计和开发方法。

2、实时系统设计的一些基本概念

与一般的系统设计一样，实时系统软件也包括一些和一般系统相同的分析和设计概念，这里就不再讨论。下面给出几点对于实时系统设计有特别重要意义的概念：

有限状态机 (FSM)：很多实时系统，特别是实时控制系统，其整个分析机制与系统的状态有相当大的关系。有限状态机由有限的状态和相互之间的转移构成，在任何时候只能处于给定数目的状态中的一个。当接收到一个输入事件时，状态机产生一个输出，同时也可能伴随着状态的转移。主要有两种方法来建立有限状态机，一种是“状态转移图”，另一种是“状态转移表”，分别用图形方式和表格方式建立有限状态机。实时系统经常会应用于比较大型的系统中，这时采用图形或表格方式对理解复杂的系统具有很大的帮助。

并发任务：实时系统通常都会有很多事件同时产生。一个任务描述一个事件，整个系统的并发机制是通过让很多任务并行运行而实现的。一个强调并发任务的系统设计通常都会显得更加清楚并且易于理解。有关实时系统的并发，下面将有专门讨论。

信息隐藏：信息隐藏是一个与各种系统设计都有关的设计概念，其原则是每个模块应该隐藏可能会做改变的设计结果。采用信息隐藏的优点是使模块可供修改，易于理解，因此增强了软件的可维护性。

3、实时系统分析和设计常用的方法

首先要区分两个不同的概念：系统分析和系统设计。系统分析指的是由用户提供的，说明软件系统应该做什么以及需要在什么环境下运行等情况的方案；系统设计则是指那些由系统分析员或设计者提供的描述软件系统如何实现这些需求的方案。在实时系统的设计中，系统的瞬时表现是最难描述的，但同时也是最重要的因素。以下讲述的传统的实时软件分析和设计常用的一些方法。

语言描述及数学分析：在系统分析和设计中，语言描述是不可缺少的，但其功能也只是对其他方法提供的分析和设计进行补充。在系统分析中，也可以采用数学分析来描述对系统功能的需求。数学分析是一种精确而有效的方法，对系统性能的优化可以通过优化相应的数学模型而获得，但在大系统中，其作用相对受限。

流程图：流程图也许是最早出现的软件系统模型，而且也有着很广泛的应用，因此，简单的系统可以采用流程图来进行建模。流程图适合指令不超过 5000 到 10000 条的系统。在多任务的系统中，流程图可用来对各个任务进行分别描述，但是进程之间的交互就不太容易描述，同时也无法描述其瞬时的表现。

结构图：结构图是一种广泛采用的、用以描述系统模块结构的方法。结构图从左到右代表了执行的顺序，从上到下代表了程序模块的细节化。结构图可以在相当多的计算机科学的文献以及其他描述系统的文章中见到。

伪代码和编程设计语言 (PDL)：编程设计语言与伪代码只有细小的差别。编程设计语言是一种与系统运行的底层硬件环境相分离开来的抽象语言，它以一种类似于真正的编写代码的方式来说明系统。ADA 就是一个成功应用编程设计语言的例子。使用编程设计语言或伪代码的好处是，它们比直接应用语言来描述系统需求更抽象，而同时比前述所有方法都更接近实际的编程语言。

有限状态机：有限状态机是实时系统设计中的一种数学模型，是一种重要的、易于建立的、应用比较广泛的、以描述控制特性为主的建模方法，它可以应用于从系统分析到设计的所有阶段。有限状态机的组成如下：

(1) 一个有限的状态集合 Q

(2) 一个有限的输入集合 I

(3) 一个变迁函数 $\delta: Q \times I \rightarrow Q$ ，变迁函数也是一个状态函数，在某一状态下，给定输入后，FSM 转入该函数产生的新状态。 δ 是一个部分函数，它的定义域内的某些数值可以是未定义的。有限状态机通常用图的方式来表示，其节点代表状态。若在输入 I 下状态由 $q1$ 转变为状态 $q2$ ，则有一条标有输入 I 的弧线从状态 $q1$ 指向 $q2$ 。此时，其变迁函数 $\delta(q1, i)=q2$ 。图 1 示意了一个简单的有限状态机。

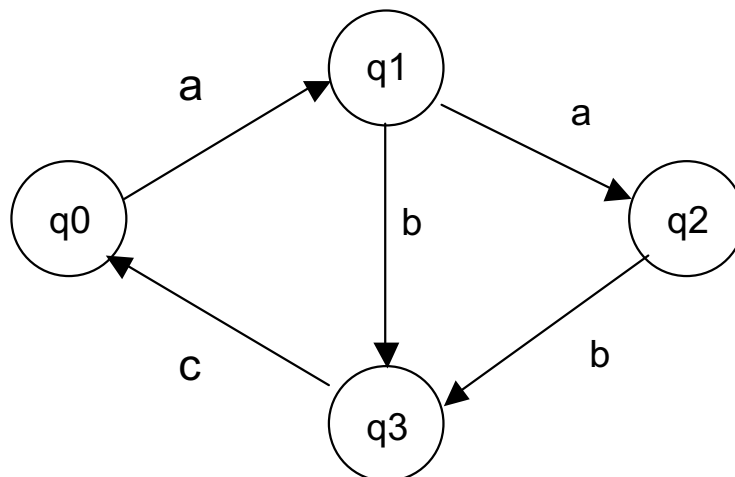
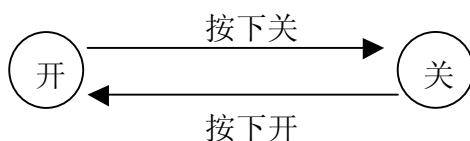


图 8-1 简单有限状态机

图 8-1 中的左图表明该 FSM 有 q_0 、 q_1 、 q_2 、 q_3 四个状态，输入集中有 a 、 b 、 c 三个元素。有限状态机非常适合于描述这样的系统：系统含有有限个状态，不同事件的发生（以不同的输入符号来表示）可以用不同状态之间的转换来模拟。举例来说，一盏灯可以是开着的，也可以是关着的，“开”与“关”都是指状态；还可以通过某种外部的控制（如按一下开关）使灯的状态从开变换到关，而我们在按一下开关就会导致反方向的状态变换。我们可以用图 8-2 中的有限状态机来描述这个简单的系统。根据控制系统的不同要求，我们可以利用有限状态机来的方法建立不同复杂性的模型。



图形 8-2：灯的有限状态机

总的来说，有限状态机的优点在于简单易用，状态间的关系能够直观看到。但应用在实时系统中时，其最大的缺点是：任何时刻系统只能有一个状态，无法表示并发性，不能描述异步并发的系统。另外，在系统部件较多时，状态数随之增加，导致复杂性显著增长。为了消除这些缺点，一些新的方法应运而生。下面将介绍的 Petri 网就是其中一种，与 FSM 相比，它具有更强的建模能力，并能描述系统的并发、异步、同步等特性。

Petri 网：Petri 网是一种使用图形方式对系统进行需求规格说明的技术，用来定义多进程、多任务系统的数学模型，易于描述系统的并发、竞争、同步等特征，并可用于评价和改进系统。如今，Petri 网已经大量应用于包括硬件、软件和社会领域等各种系统的模型化。类似于有限状态机，Petri 网也是通过一些定义好的状态来描述事物的抽象的虚拟机。但与 FSM 相比，Petri 网不仅能描述同步模型，更适合于相互独立、协同操作的处理系统。Petri 网的组成成分包括：

- (1) 一个有限的库所 (place) 集合，表示系统的状态。
- (2) 一个有限的变迁 (transition) 集合，表示系统中的事件。
- (3) 一个有限的连接库所到变迁或者反向的有向箭头的集合，又分输入 (input) 和输出 (output)。Input 表示变迁/事件的前提，output 表示变迁/事件的结果。

图 8-3 为一 Petri 网示例，图中的库所用圆圈表示，变迁由矩形表示。Petri 网有如下特性：

- (1) 状态：通过标记 Petri 网的库所来给出其状态，标记 Petri 网的库所在图形中表现为对库所插入数目不同的令牌。
- (2) 状态变化规则：一个变迁可以有多个输入或输出库所，如果一条有向箭头是从库所到变迁，则该库所是该变迁的一个输入库所，反之则为输出库所。如果一个变迁中每一个输入库所都至少有一个令牌，则称该变迁是一个使能变迁。
- (3) 点燃：一个使能变迁可以被点燃，即从该变迁的每一个输入库所中移走一个令牌，在该变迁的每一个输出库所中增加一个令牌。

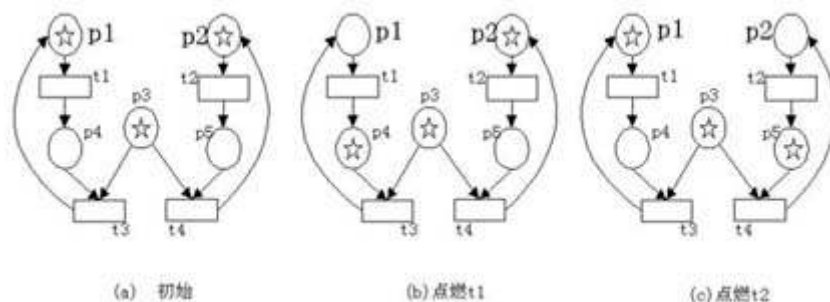


图 8-3 Petri 网示例

图 8-3 中，库所中的星号代表令牌。由(a)中可以看出，变迁 t1 和 t2 都是使能的，(b)表示了点燃 t1 的过程，(c)表示了点燃 t2 的过程。从这个模型中可以看出，在给定初始令牌后，Petri 网的发展过程可能是不同的。在 Petri 网中，经常利用变迁来模拟一个事件，而点燃则用来表示事件的发生。这样，如果一个变迁所表示的事件的发生条件是满足的，那么这个变迁就是使能的，库所中的星号标记则说明某个条件是满足的。图 8-3 中的 Petri 网可分为两部分，一部分是由变迁 t1、t3 组成，另一部分由 t2、t4 组成，这相当于是两个独立的工作流，并且共享了资源 p3。开始时，双方可以互不干涉地、异步地进行，因为变迁 t1 和 t2 双方互不妨碍。点燃 t1 后，t3 处于使能状态，点燃 t2 后，t4 处于使能状态。在两个变迁都点燃后，两个工作流都有新的变迁处于使能状态，但是这两个变迁的点燃处于竞争状态，两个工作流中只能有一个获得 p3 的资源从而得以继续变迁。如果资源满足的情况可以解决这种冲突，就可以设 p3 的令牌有两个，两个工作流就互不干扰，可以并发执行。

数据流图：数据流图主要是作为软件系统建模中的结构化分析工具。建立数据流图从分析系统的功能需求开始，分析系统中的数据流并且确定主要的函数。数据流图需要细分系统和子系统，从而在图中对数据流有清楚的表现。控制流图是一种在系统中实现控制信号流的数据流图。这些控制信号能够描述离散的从开关或感应器得到的信号数据，同时也能包含从离散时钟得到的离散的定时信息。数据流图和控制流图结合使用，可以提供更具控制性的信息。总体来说，数据流图特点是：强调数据的流动，不太强调控制信号的流动，对于确定

并发性很有用。实际上，如果缺少了并发性的特点，数据流图就是有限状态机。数据流图最主要的缺点是描述同步很困难。数据流图容易理解，应用广泛，通常是和一些其他方法结合使用来产生连贯的软件需求文档。

表 8-1 给出了上面讲述的各种系统分析方法的优缺点的对比。

表 8-1：系统分析方法的对比		
方 法	优 点	缺 点
语言描述	广泛应用，能起说明作用。	不够明确，无法相应产生代码。
数学分析	精确，是一种正式的分析方法，可在此基础上进行代码优化。	难于应用，并非很多系统设计人员都能够通过数学方式对系统分析。
流程图	广泛应用，对单任务系统非常适合。	没有并发处理，不能表现瞬时状态，多采用 GOTO 语句。
结构图	广泛应用，适合于小系统，鼓励自顶向下的设计方式。	没有条件转移，没有并发处理，不能表现瞬时状态。
伪代码和 PDL	与编程语言接近，是正式应用的分析和设计方法。	容易出错且不易发现。
有限状态机	广泛应用，易于建立，方便于代码产生和优化。	没有并发处理，随着状态的增加系统复杂性显著增加。
Petri 网	广泛应用，适合于多进程处理，对并行性有很好的表示。	应用于过小的系统时显得不必要。
数据流图	广泛应用，并发处理，易于理解。	不易于表示同步性

4、实时系统的并发

并发 (concurrency) 是实时系统的一个重要特征。在实时系统中，外部事件在任意时刻都可能出现，有时甚至会同时出现，必须保证在规定的时间内及时处理。多个事件可以列入队列进行调度，或依次顺序处理，或并发处理。并发处理具有以下明显优越性：

- 可以有效利用硬件资源；
- 帮助程序设计员将一个程序分解为几个不同的活动，而不必考虑这些活动的具体执行顺序；

但是，并发常常会给处理带来一些难以解决的问题。下面我们举例说明：见图 8-3，假定系统 S 有以下特性：

- (1) 系统 S 能够响应两种事件 E1 和 E2。
- (2) 系统能够执行四个原子操作：T1、T2、T3 和 T4。原子操作是指操作不再可分割。
- (3) 每个原子操作需要一个单位时间。
- (4) 在响应某个事件时，将执行一系列顺序操作，顺序操作执行的第一个操作最关键；系统必须在事件发生后的一个单元时间内开始执行该操作，并必须在该事件发生后两个单位时间之内完成。顺序执行的操作系列之间相互独立。
- (5) 系统 S 响应时间 E1 时，顺序执行三个原子操作：T1、T2 和 T3，且在事件发生之后 6 个单元事件内完成 T3 操作。

- (6) 系统 S 响应事件 E2 时，顺序执行两个原子操作：T4 和 T2，且 T2 操作必须在事件发生之后的 4 个单位时间内完成。

对于系统的外部环境，存在如下情况：

- (1) 在时刻 0，事件 E1 发生。
- (2) 在时刻 1.5，事件 E2 发生。
- (3) 在时刻 3.5，事件 E1 发生。

以上系统必须同时响应多个事件，执行多个操作系列的情况，称为并发。理想并发(ideally concurrent)是指事件发生后，系统立即响应，执行相应的操作序列；且多个事件发生时，可以同时执行任意多个独立操作序列。见图 8-4。

然而理想并发只是一种理想化处理，一般系统只有一个处理器，一次只能执行一个原子操作。在这样的系统中，实际上不可能处理任意数量的并发操作，操作必须顺序执行。但若按先进先出的策略执行操作，结果将如图 8-5 所示。在时刻 1.5 发生的 E2 事件，必须到时刻 3 以后才能获得处理器；在时刻 3.5 发生的 E1，至少要到时刻 5 才可能获得处理器；因而系统无法达到规定的时间要求。

为达到时间要求，系统必须在一个操作执行结束后，打断正在执行的操作序列，转而处理另一操作序列。如果能够合理地安排以适当延迟前一个操作，往往可以达到时间要求。这种利用延迟机制来满足并发要求的情况称为准并发(quasi-concurrent)，见图 8-6。

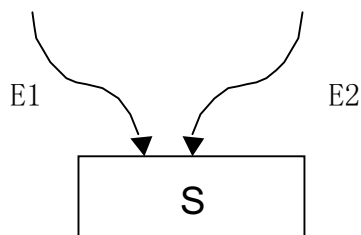


图8-3：系统响应两种事件

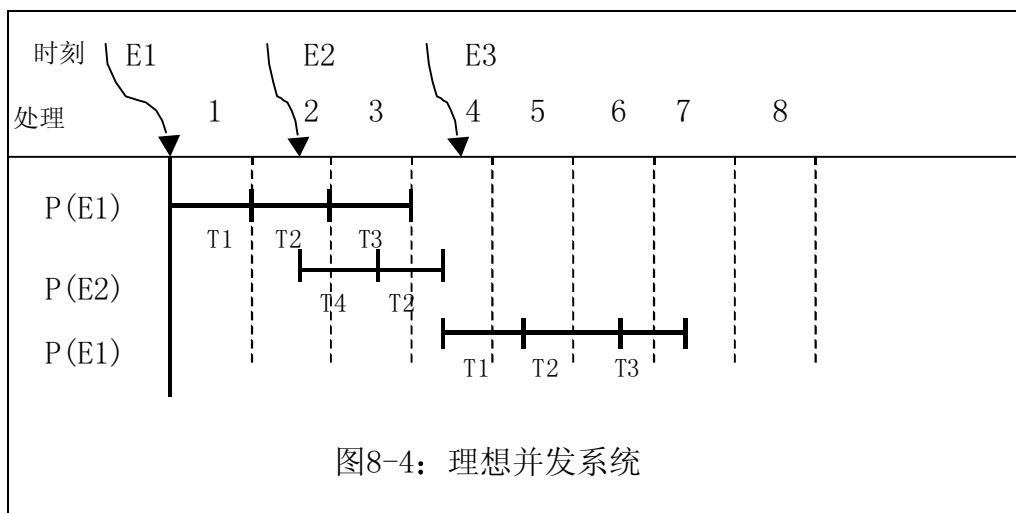
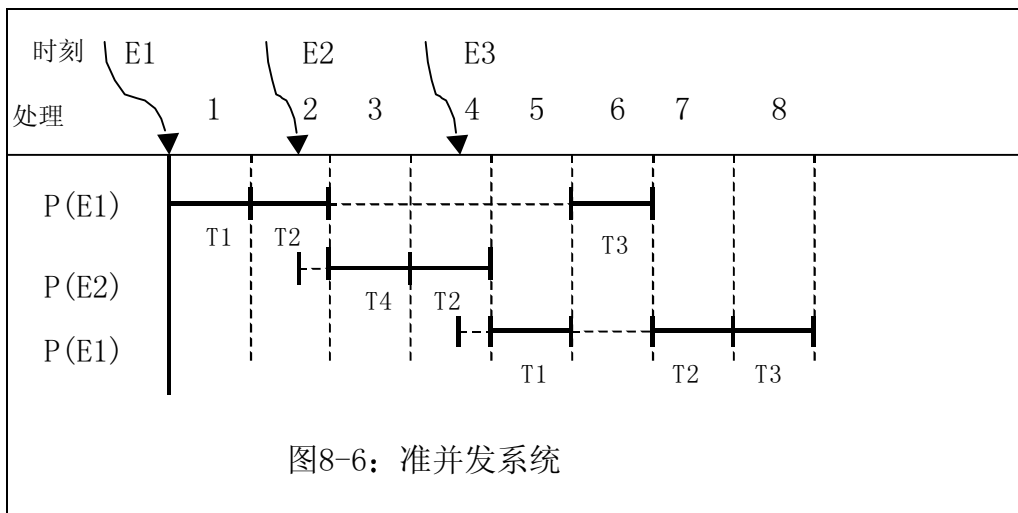
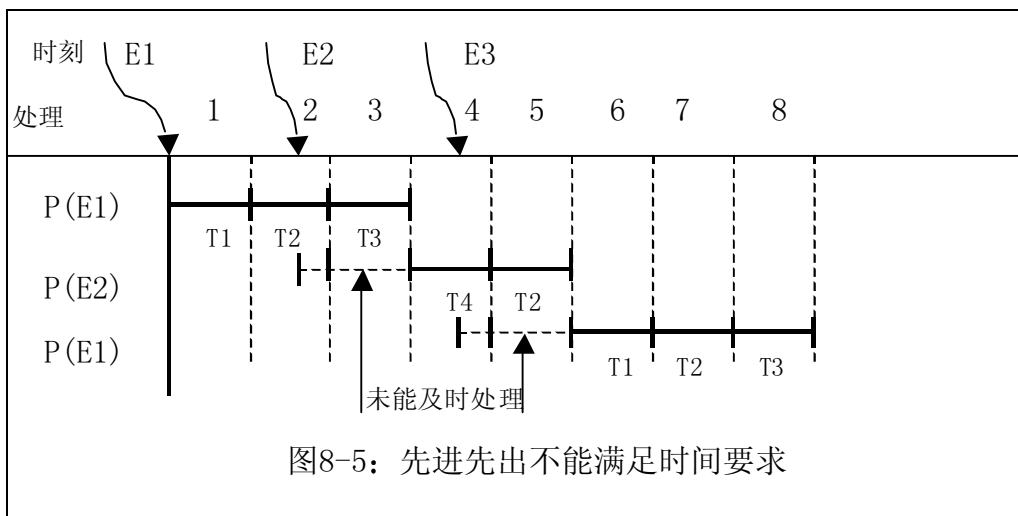


图8-4：理想并发系统



5、面向对象的并发模型概述

在采用面向对象开发方法处理实时系统时，最困难的问题是如何把对象概念与并发概念结合起来。可以通过某些对象来组成实时软件。实时系统的基本工具包括：协作进程，中断服务例程，实时时钟和物理输入输出设备。还有一部分“纯实时软件”，它不使用进程，而利用同步技术，即在程序执行前固定和决定进程调度。更加灵活的异步技术可应用于实时和面向对象相结合的领域，但是，使用进程会限制快速响应方面的要求。可以通过改变进程的优先级或把进程迁移到其他的处理器，从而调节系统性能来影响系统的响应性能。有关系统在实时方面的问题有以下几点：

- 粒度大小。
- 协作软件间的优先级。
- 中断服务例程：控制和处理某优先级的硬件中断激励。
- 实时时钟：可用来触发进程切换和调整进程队列。
- 与进程和物理分布相关的其他复杂问题，如调度、优先级、进程间通信、进程中断、服务例程通信、错误检测和恢复、进程破坏和清除、共享资源保护等等。

这些问题将严重影响设计者在设计过程中对软件的考虑。设计者必须在某种程度上时刻考虑粗粒度行为模式，以保证整个软件正确地调度性能与健壮性问题。这将是复杂地，因为一个进程可能被比它优先级高的进程打断，后果可能是在一系列的进程上同时传播许多激

励，进而可能影响性能和健壮性。为了尽量减少产生错误的可能性，设计者和实现者倾向于使进程相对具有较粗粒度，进程数量相对少，进程结构相对静态；使得实时结构和代码结构尽量吻合，把实时问题直接反映到代码中。

在面向对象方法方面，作为类继承体系的代码结构提供了其他编程技术无可比拟的复用性和可扩展性。几组对象通过负责过程间通信的类动态建立实时结构。动态绑定技术可以避免让代码中固定的结构承担过多责任，以提高类的可扩展性，进而提高可用性。但许多动态绑定的细粒度代替进程，导致出现了较多的细粒度对象。这样，动态实时结构和类继承的静态代码结构不一致，就难以通过代码了解系统行为，即难以看到相对高层的实时动态结构——粗粒度行为动态模式。因此，我们必须建立面相对象的实时模型。

面向对象的实时模型分为两种：显示并发模型和隐式并发模型。

隐式并发模型的特点是推迟并发设计。它把对象模型作为建模基础。在进入运行阶段之前，将对象看成自主单元，各种对象的活动看成是以理想并发方式完成特定的工作。好似每个对象都拥有一个自己的处理器，这个处理器为对象提供了一个执行线程。进入系统的外部事件被看成一个处理请求，以广播方式传给一些对象，这些对象接着向其他对象进一步提出处理请求。理论上，对应一个请求，可以有任意多个对象执行相应处理。在实现时，由调度程序最终处理其对象的操作顺序。

显式并发模型则首先考虑并发。应先把并发概念与对象分开。在建立对象后，用实时操作系统支持的进程概念来表示并发，形成了对象和进程两个抽象层次，即先将系统分解为准并发进程作为开始，而在每个进程内部采用面向对象的技术。对象间交互表示成嵌套的函数调用；通过加入锁、监视器、信号量等显式同步机制，来保证对象的完整性。

因此我们说，显示模型和隐式模型的区别就是对象和进程结合的事件早晚不同。显示模型较早实施结合，在分析阶段，就要考虑并发问题，设计阶段任务轻松了。且通过将进程置于对象之上，对象中不必考虑并发，对象串行化。其缺点是需要较早决定把应用分解为进程的方案。隐式模型则可推迟考虑并发，在分析阶段，只考虑对象模型，而不考虑并发，但在设计阶段必须考虑进程结构，解决同步问题。显示并发和隐式并发的并发模型见图 8-7。

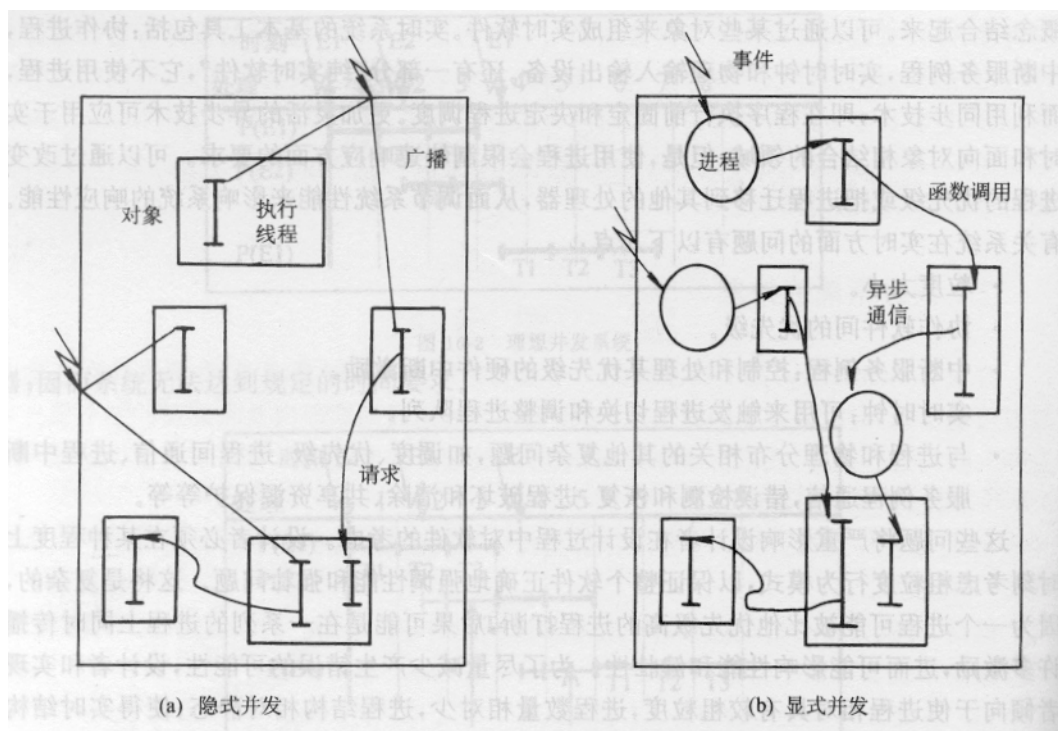


图 8-7：面向对象的并发模型

实际问题往往非常复杂，应认真考虑对象方案。从并发的程度来看，我们可提出三中并发的级别：

- 低级并发：每个对象的执行保持完整，对象依次交互，并发工作。
- 中级并发：对象的操作可以中断，不同对象之间发生并发。
- 高级并发：同一对象内部各操作之间发生并发。

越高级的并发处理越复杂。要解决并发的实时问题，可采用以下方法：

- 对已有的面向对象语言进行扩充，使之支持并发机制。
- 对已有的面向对象语言，增加实现并发机制的类库。

一般说来，并发机制可以解决实时系统的问题，但是数据的一致性（同一个对象上，两个并发请求访问同一数据时带来的不确定性）和同步问题会给问题的处理带来极大的困难。应该尽量避免高级并发，即避免对象内部的并发，使对象内部各操作互拆，以简化问题。

6、面向对象的实时系统设计方法——OCTOPUS概述

面向对象技术在实时系统设计中的应用目前仍然有限。因此对诺基亚公司在移动通信系统软件的设计开发中提出的 OCTUPUS 方法所获得的成就，Awad 等人进行了总结，较为全面地归纳了面向对象地实时系统设计方法。这可以说是一个开创行动，很有启发意义。OCTPUS 方法的命名是基于该方法设计多方面的技术，博采众长，像章鱼似的有多个触角。

OCTOPUS 方法，以对象模型技术 OMT(object modeling technique)和熔合方法(the fusion method)为基础。OMT 的对象模型符号可以简洁地表示必要细节，将结构、功能以及动态模型分开使系统易于建立、便于理解。而 Fusion 方法地可借鉴之处在于在分析阶段侧重描述外部行为，而在设计阶段侧重描述应用系统内部行为，使得不同阶段重点突出。

OCTOPUS 方法不仅尽最大可能保留了 OMT 合 Fusion 方法地长处，还提出了对实时系统响应行为、时间域以及并发地处理方法。具体有对并发、同步、通信、中断处理、ASIC、硬件界面、端对端响应时间等方面地处理。这里只对其设计方法做描述。

OCTOPUS 方法将软件开发的主要阶段很好地结合起来，从规格说明到运行，模型之间过渡紧密自然。它还支持渐增式软件开发。OCTOPUS 方法也支持异构软件开发，使面向对象地部分成为整个非面向对象系统地一部分。它将传统地实时操作系统作为运行平台，通过系统有效的步骤将对象映射到进程，从而解决了将进程与对象结合起来的难题。因而 OCTOPUS 方法当前是一个将面向对象与实时系统相结合的较好方法。

从系统角度来看，OCTOPUS 在系统、子系统、类、对象抽象层次上建立结构模型、功能模型和动态模型，这些抽象层次分布在各个不同的开发阶段。

图 8-8：OCTOPUS 方法的软件系统开发过程结构

系统规格说明阶段见图 8-8、表 8-2，通过上下文图表示系统环境，通过一系列使用实例(use case)的图形文字表示系统功能和动态行为。使用实例可以补充以系统与环境之间的场景来表示动态行为。系统各部分在考虑使用实例阶段基本上可认为是黑盒子。

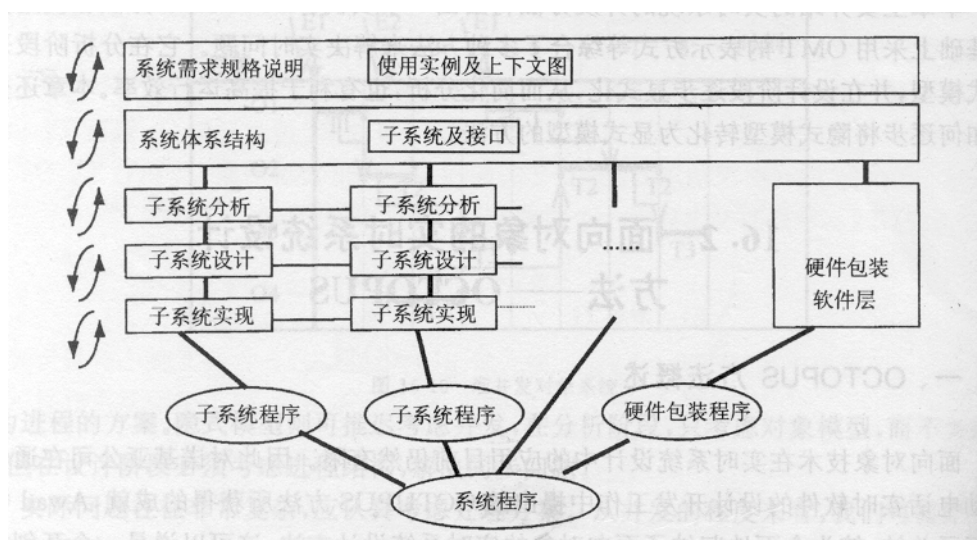


表 8-2：OCTOPUS 概貌

抽象程度	开发阶段	结构模型	功能模型	动态模型
系统 (问题域)	系统规格说明书	系统环境图	系统使用实例图/ 全部使用实例	系统使用实例图/ 全部使用实例(场景)
子系统	系统体系结构	子系统图	在子系统分析功能模型中说明	在子系统的分析动态模型中说明
类	子系统分析	类图和类说明书	子系统操作表单	子系统事件表/ 事件组图/ 事件表图/ 状态图/ 活动表/ 重要性程度表/ (复合事件)/ 场景
对象	子系统设计	类大纲/ 进程大纲/ 进程间消息大纲	成员函数大纲 (并入在结构模型中的大纲中)	对象交互线程/ 事件线程/ 对象组

系统体系结构阶段，由子系统图描述系统结构，标出子系统间的主要界面。但是功能模型和动态模型暂不进行，而到下一个开发阶段进行。在系统体系结构阶段，把子系统看作为是黑盒子。

对每个子系统都进行子系统分析，结构模型由子系统类图和类描述表共同描述，功能模型由操作清单来描述，动态模型中包括事件清单、状态图、动作表、重要性程度表，还可能包括一系列复合事件。子系统在功能模型和动态模型中都被看成是黑盒，但功能模型和动态模型可以引用结构模型中的类。

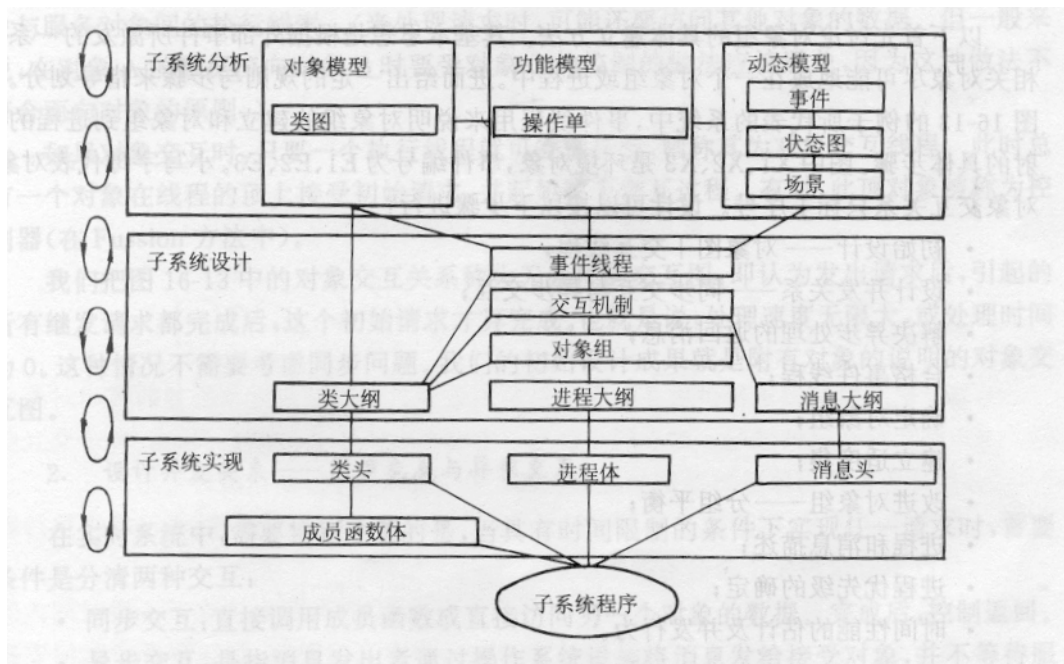
每个子系统都有子系统设计阶段，该阶段是子系统分析阶段的继续。动态模型有一系列对象交互线程表示。这些交互线程进一步设计为合格事件线程，接着由此导出对象组。结构模型可以由以上这些系统地建立起地进程描述、类描述和进程间消息描述构成。功能模型主要通过说明与结构模型中地类对应地成员函数描述。

在子系统实现阶段，见图 8-9，所有模型转换为所选编程语言地源码。图 8-8 和图 8-9 左边地圆形箭头强调了开发过程是迭代进行的。接近实现阶段时，迭代程度降低，因为此时在阶段内部的迭代更频繁，而阶段间的迭代减少了。

图 8-9：OCTOPUS 方法的子系统开发过程结构

OCTOPUS 处理实时系统并发的独特方法是：先采用隐式并发模型，然后逐渐过渡到显式模型，利用对象组实现到进程的映射。这是一种能够有效将对象和进程结合起来的方法。

7、实时程序设计的一些准则



(1) 实时单元和分时单元的合理划分

对于实时系统，实时和分时单元的合理划分，是提高整个系统实时性能的一个重要手段。例如，对于信令处理系统来说，对信令的翻译、解释和转移、传递、应答是实时单元，而对于信令的监视打印、非法信令的打印等则是分时单元。实时单元应该放到实时任务里面去处理，而分时单元的处理应该由实时任务通过消息或者共享内存模式传递数据，启动分时进程的在线或者后台处理。

(2) 实时任务的划分

任务是代码运行的一个映象,从系统的角度看,任务是竞争系统资源的最小运行单元。任务可以使用或等待 CPU、I/O 设备及内存空间等系统资源,并独立于其它任务,与它们一起并发运行(宏观上如此)。操作系统内核通过一定的指示来进行任务的切换,这些指示都是来自对内核的系统调用。

在应用程序中,任务在表面上具有与普通函数相似的格式,但任务有着自己较明显的特点:

1. 任务具有任务初始化的起点(如获取一些系统对象的 ID 等);
2. 具有存放执行内容的私用数据区(如任务创建时明确定义的用户堆栈和堆栈);
3. 任务的主体结构表现为一个无限循环体或有明确的终止(任务不同于函数,无返回)。

在设计一个较为复杂的多任务应用时,进行的合理的任务划分对系统的运行效率、实时性和吞吐量影响极大。任务分解过细会引起任务频繁切换的开销增加,而任务分解不够彻底会造成原本可以并行的操作只能按顺序串行完成,从而减少了系统的吞吐量。为了达到系统效率和吞吐量之间的平衡与折衷,在设计应用时应遵循如下的任务分解规则(假设下述任务的发生都依赖于唯一的触发条件,如两个任务能够满足下面的条件之一,它们可以合理地分开):

1. 时间:两个任务所依赖的周期条件具有不同的频率和时间段;
2. 异步性:两个任务所依赖的条件没有相互的时间关系;
3. 优先级:两个任务所依赖的条件需要有不同的优先级;
4. 清晰性/可维护性:两个任务可以在功能上或逻辑上互相分开。

从软件工程和面向对象的设计方法来看,各个模块(任务)间数据的通信量应该尽量小,并且最好少出现控制耦合(即一个任务可控制另一个任务的执行流程或功能),如果非要出现不可,则应采取相应的措施(任务间通信)使他们实现同步或互斥,以避免可能引起的临界资源冲突,因为这可能会造成死锁。

一般说来,很多时候为了提高实时系统的运行效率,我们设计时要考虑将两(多)个逻辑上应该分成两(多)个任务的任务合并为一个任务。例如:有两个任务 A 和 B,他们分别做两个优先级没有明显分别的处理,他们通过消息通信,任务 A 是消息的发送者,任务 B 是消息的接受者,如果他们之间通信的消息量很大,可能就需要考虑将这两个任务合并,然后通过任务内部的缓冲区等通信机制来实现他们之间的通信,当然前提是从逻辑上讲,这两个任务的合并是可能的。一般说来,可以把这种任务内部的两个处理模块之间的通信机制设计成为虚消息机制,即在系统分析和设计以时,我们可以认为它就是消息,在程序设计时,通过宏函数或者函数调用的封装,普通程序员仍然可以通过虚消息的收发来编写他所需要完成的应用程序,从程序员的角度来看,这一点仍然是消息驱动的机制,与实时操作系统提供的消息通信机制没有什么差别。

对于大多数商用的实时操作系统,系统内核对应用任务的数量没有作限制,实际应用时是在配置表中定义最大的任务数,并为存放有关数据结构分配足够的内存空间。在设计一个复杂应用时,上面讲述的任务分解原则仅能作一个初步的参考,真正设计时还需要更多的实际分析和设计经验,才能使系统达到预定性能指标和效率。

(3) 提高程序效率的途径

实时程序的设计相对于分时程序的设计,尤其强调程序效率的优化。很多在分时程序设计中对于提高程序效率有效的方法,仍然有效。比较常用的优化手段有:

- 1) 使循环体内工作量最小化:

应仔细考虑循环体内的语句是否可以放在循环体外,使循环体内工作量最小,从而提高

程序的时间效率。

例如，如下代码效率不高。

```
for(i=0; i < MAX_ADD_NUMBER; i++)
{
    sum +=i;
    total = sum;
}
```

语句“total = sum;”完全可以放在 for 语句之后。如下：

```
for(i=0; i < MAX_ADD_NUMBER; i++)
{
    sum +=i;
}
total = sum;
```

2) 使用寄存器变量：优化关键函数时，要按照目标计算机支持的寄存器变量数，将使用得最频繁的循环变量、指针变量依次设置为寄存器变量，以提高效率。

3) 使用经典的高效率的算法：对于一些常用的算法（例如查找、排序、hash 等等），不妨去查找经典的库或者在原有的代码的基础上改写。

4) 优化函数的组织结构：对模块中函数的划分及组织方式进行分析、优化，改进模块中函数的组织结构，提高效率。

而对于实时程序的设计，还有一些方法可以提高实时程序的效率。列举如下：

1) 尽量使用宏函数：对于实时程序，如果能够使用宏函数代替函数，则应该尽量采用宏函数，以提高实时系统的效率。

2) 优化主线程：主线程是指任务按照正常流程接受合法消息时，所走的最常调用分支。这些分支效率的高低，将直接影响到整个任务的平均响应速度，所以应该仔细地优化。对于实时任务来说，主线程的优化尤为重要，应该尽量避免在主线程中做 printf, sprintf, fprintf 等十分费时的流操作。对于确实需要的内存拷贝等操作，最好避免使用 C 库函数，可以考虑使用专用的、在汇编级实现的不通用的函数，因为通用的 C 库函数没有针对 CPU 做优化，时间效率较低。而主线程的算法设计也应该仔细地优化。并且优化完后，要对主线程的运行效率（或者说一个主线程的平均处理时间和最坏处理时间）做测试，这个测试数据将做为整个实时系统调度策略设计的基础数据。

九、实时操作系统 pS0System

1、pS0System概述

pS0System 是世界上最早的实时系统之一，也是最早进入中国市场的实时操作系统。pS0System 是集成系统有限公司(Integrated Systems, Inc.)研发的产品，该公司是嵌入式业界实力比较雄厚的公司，ISI 在 2000 年 2 月 16 日与 WindRiver Systems 公司合并，两

公司合并之后增强了在嵌入式控制领域的领先地位。

pSOSystem 是一个完全可扩展的嵌入式实时操作系统，它包含单处理器支持模块 (pSOS+)，多处理器支持模块 (pSOS+m)，文件管理器模块 (pHILE)，TCP/IP 通讯包 (pNA+)，流式通讯模块 (OpEN)，图形界面，JAVA，HTTP 等。pSOSystem 功能模块完全独立，开发者可根据应用要求扩展系统功能和存储容量。pRISM+ 是 Psosystem 的全集成开发环境，包含 C/C++ 编辑浏览器，源级调试器，编译器和图形运行分析工具。Psosystem 推出时间较早，因此它比较成熟，目前可以支持很多类型的处理器，其板级支持包 BSP 也比较全，能支持各种类型的评估板，同时支持 IS09660，MS-DOS，NFS 等多种文件系统，目前全球大约有 3500 万个设备上使用的是该操作系统。

pSOSystem 两年前在实时操作系统领域销售额占世界第一，近两年市场占有率有所下降，目前销售额为世界第二。pSOS+ 在中国市场占有率是最高的，主要应用领域包括通讯、航天、信息家电以及工业控制。

pSOSystem 的主要缺点在于其上下文切换时间长，实时性不太强，采用的集成开发环境 Sniff+ 与产品兼容性不好，部分关键功能无法使用。以下的讲解我们以中国市场占有率最高的 pSOS+ 为重点为例来介绍，也会讲到 pSOS+m 对于多处理器的支持的主要思想和部分系统调用。我们约定，pSOS+ 表示支持单处理器的内核，pSOS+m 表示支持多处理器的内核，pSOS 表示 pSOSystem 实时操作系统，需要指出的是，实时内核不等同于实时操作系统，实时内核只是实时操作系统中核心的一部分。

2、pSOSystem 的系统结构（以 pSOSystem 3.0 为例）

pSOSystem 是一个由标准软组件组成的，可剪裁的实时操作系统。其系统结构如图 9-1 所示，它分为内核层、系统服务层、用户层。

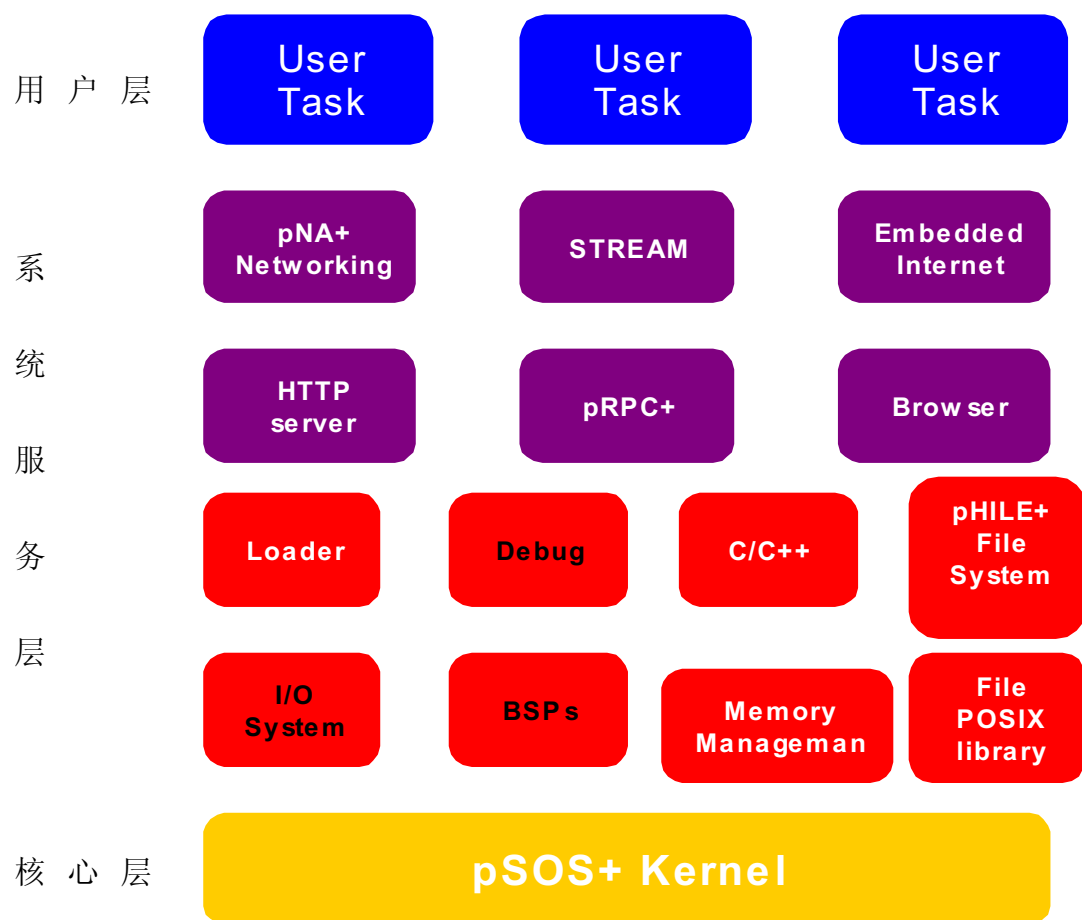


图 9-1: pSOSSystem 的系统结构

(1) 内核层

包括两中内核，即 pSOS+实时多任务内核和 pSOS+m 实时多任务内核。pSOS+m 扩展了 pSOS+ 的一些特性，使得在紧耦合或松耦合的多处理器上的操作可无缝隙实现。除非特别声明，以下以 pSOS+为例来介绍 pSOSSystem。

pSOS+内核负责任务的管理与调度、任务间通信、内存管理、实时时钟管理、中断服务；可以动态生成或删除任务、内存区、消息队列、信号灯等系统对象；实现了基于优先级的、选择可抢占的任务调度算法，并提供了可选的时间片轮转调度。pSOS+ Kernel 还提供了任务建间通信机制及同步、互斥手段，如消息、信号灯、事件、异步信号等。pSOSSystem 操作系统在 Kernel 层中将具体硬件有关的操作放在一个模块中，对系统服务层以上屏蔽了具体的硬件特性，从而使得 pSOSSystem 很方便地从支持 Intel 80x86 系列转到支持 MC68xxx 系列以及 PowerPC 系列，并且在系统服务层上对不同应用系统不同用户提供标准的软件组件如 PNA+、PHILE+等。

pSOS+内核的系统服务共分成八大类：任务管理、存储分配、消息队服务、事件和异步信号服务、信号量（Semaphore）服务、时间管理及定时器服务、错误处理服务、多处理器支援服务（pSOS+m）。

(2) 系统服务层

pSOS 系统服务层包括 pNA+、pRPC+、pHILE+等组件。pNA+实现了完整的基于流的 TCP/IP 协议集，并具有良好的实时性能，网络组件内中断屏蔽时间不大于内核模块中断屏蔽时间。pRPC+提供了远程调用库，支持用户建立一个分布式应用系统。pHILE+提供了文件系统管理和对块存储设备的管理。C/C++提供了标准的 C、C++库，支持用户使用 C、C++语言编写应用程序。由于 pSOS+内核屏蔽了具体的硬件特性，因此，pSOS+系统服务层的软组件是标准的、与硬件无关的。这意味着 pSOS+各种版本，无论是对 80x86 系列还是 MC68xxx 系列，其系统服务层各组件是标准的、同一的，这减少了软件维护工作，增强了软件可移植性。每个软组件都包含一系列的系统调用。对用户而言，这些系统调用就像一个个可重入的 C 函数，然而它们却是用户进入 pSOS 内核的唯一手段。

(3) 用户层

用户指的是用户编写的应用程序，它们是以任务的形式出现的。任务通过发系统调用而进入 pSOS+内核，并为 pSOS+内核所管理和调度。pSOS+为用户还提供了一个集成式的开发环境 (IDE)。pSOS+_IDE 可驻留于 UNIX 或 DOS 环境下，它包括 C 和 C++优化编译器、CPU 和 pSOS+模拟仿真和 DEBUG 功能。

3、pSOS+对象 (Object) 的主要概念

pSOS Kernel 是一个面向对象的操作系统内核，pSOS 系统中对象包括 Task(任务)、Memory regions、Memory partitions、Queue (消息队列) 和 semaphore (信号灯)。

pSOS+ 与其它实时内核相比的显著特点是提供了名字服务，即创建对象时为每个对象命名，但操作对象时是用的对象的 ID 号。对象名由用户定义 (4 位 ASCII 字符)，在该对象创建时利用系统调用 obj_create 的一个入口参数传给 pSOS+ Kernel。pSOS+ Kernel 将赋予该对象一个唯一的 32 位 ID 号，任务可以通过 obj_ident 取出对象的 ID 号。除 obj_create 和 obj_ident 外，所有涉及对象的系统调用都要用到对象 ID 号。这样做的好处是把应用进程区分对象时的命名——对象名字和系统内核寻址对象的索引——对象 ID 分开了，隔离了内核和应用进程，提高了 pSOS+系统内核的响应速度。

创建对象的任务通过调用 obj_create 的返回值就已经知道了该对象的 ID 号，其余任务可通过 obj_ident 或通过全局变量 (如果已经为该任务的 ID 号建立了一个全局变量的话) 获取该对象的 ID 号。对象 ID 号隐含了该对象控制块 (如 TCB、QCB) 的位置信息，这一位置信息被 pSOS Kernel 用于对该对象的管理和操作，如挂起/解挂一个任务、删除一个消息队列等。

4、pSOS+任务 (task) 的主要概念

(1) 任务的基本概念

在实时操作系统中，任务是参与资源竞争以及占有系统资源 (如 CPU、Memory、I/O devices 等) 的最小执行单位。pSOS+为每个任务构造了一个虚拟的、隔离的环境，从而在概念上，一个任务与另一个任务之间可以相互并行、独立地执行。任务与任务之间的切换、任务之间的通信都是通过发系统调用 (在有些情况下是通过 ISR) 进入 pSOS+ Kernel，由 pSOS+ Kernel 完成的。pSOS 系统中任务包括系统任务和用户任务两类。关于用户任务的划分并没有一个固定的法则，但很明显，划分太多将导致任务间的切换过于频繁，系统开销太大，划分太少又会导致实时性和并行性下降，从而影响系统的效率。一般说来，功能模块 A 与功能模块 B 是分开为两个任务还是合为一个任务可以从是否具有时间相关性、优先性、逻

辑特性和功能耦合等几个方面考虑。

每个任务都有一个优先级。pSOS+系统支持 0~255 级优先级，0 级最低，255 级最高。0 级专为 IDLE 任务所有，240~255 级为系统所用。IDLE 习惯称为空闲任务，当系统没有其它可以执行的任务时，内核将执行这个任务。在运行时，任务（包括系统任务）的优先级可以通过 `t_setpri` 系统调用改变。

一旦启动后，一个任务通常处于下面三个状态之一：

- ①Executing (Ready)：就绪
- ②Running：运行
- ③Blocked：阻塞。

只有通过任务本身或其他任务、ISR 对 pSOS+内核所作的系统调用才能改变任务状态。从宏观角度看，一个多任务应用通过一系列到 pSOS+的系统调用迫使 pSOS+内核改变受影响任务而从运行一个任务到运行另一任务向前发展的。对于 pSOS+ kernel，任务在创建前或被删除后是不存在的。被创建的任务在能够运行前必须被启动。

就绪任务是未被阻塞可运行的，只等待高优先级任务释放 CPU 的任务。由于一个任务只能由正运行的任务通过调用来被启动，而且任何时刻只能有一个正在运行的任务，所以新任务总是从就绪态开始。

运行态任务是正在使用 CPU 的就绪任务，系统只能有一个 running 任务。一般 running 任务是所有就绪任务中优先级最高的，但也有例外，如前所述的为解决优先权倒置使用了有优先权的信号量后，临时提高低优先级的任务 L 的优先级至高优先级任务的优先级。

任务是由自身特定活动而变为阻塞的，通常是系统调用引起调用任务进入等待状态的。所以任务不可能从 ready 态到 blocked 态，因为只有运行任务才能执行系统调用。

任务控制块 TCB 是 pSOS+内核建立并维护的一个系统数据结构，它包含了 pSOS+ Kernel 调度与管理任务所需的一切信息，如任务名、优先级、剩余时间片数、当前寄存器状态等。在有的 RTOS 中，任务的状态与任务 TCB 所处的队列是等同的。pSOS 操作系统将二者分为两个概念，例如任务处于阻塞状态，但它的 TCB 却处于消息等待队列、信号灯等待队列、内存等待队列、超时队列之一。

pSOS 启动时，将根据 Configuration Table 中的参数 `kc_ntask` 建立一个包含 `kc_ntask` 个 TCB 块的 TCB 池，它表示最大并行任务数。在创建一个任务时，分配一个 TCB 给该任务，在撤销一个任务时，该 TCB 将被收回。

每个任务带有一个 mode word，用来改变调度决策或执行环境。主要有以下四个参数：

`preemption Enabled/Disabled`：是否允许优先级调度；

`roundrobin Enabled/Disabled`：是否允许时间片调度；

`interrupts Enabled/Disabled`：是否允许中断；

`ASR Enabled/Disabled`：每个任务有一个通过 `as_catch` 建立起来的异步信号服务例程 ASR。异步信号类似于软件中断。当 ASR 位为 1 时 `as_catch` 所指向的任务将会被改变执行路径，先执行 ASR，再返回原执行点。

(2) 任务调度

pSOS+采用优先级和时间片的调度方式。有两个因素将影响动态调度的效果：一是优先级可变（通过 `t_setpri` 系统调用改变任务的优先级）；二是任务模式字中的 `preemption bit` 位和 `roundrobin bit` 位。`preemption bit` 位决定不同优先级的任务是否可抢占，并和 `roundrobin bit` 位一起决定任务的时间片轮转是否有效。pSOS 系统中引起调度的原因有两条：

- (1). 在轮转方式下时间片到；

(2). pSOS+系统调用引发任务调度。该系统调用可能是 ISR 发出的,也可能是某个任务发出的。

pSOS 任务调度的结果有两种:

- (1). 引起运行任务切换
- (2). 不引起运行任务切换

不论任务调度是否引发运行任务切换,都有可能引起一个或多个任务状态变迁。

下面三种情况将引发运行任务切换:

① 在时间片轮转方式下(此时任务模式字的 roundrobin bit 与 preemption bit 均为 enable),运行任务 Task A 的时间片用完,且 Ready 队列中有相同优先级的其它任务,则 Task A 退出运行。

② 在运行任务 Task A 的 mode word 的 preemption bit 位为 enable 的前提下,若 Task A 发出的某条相同调用引发一个优先级高于 Task A 的任务 Task B 从 Block 状态进入 Ready 状态,则将 Task B 投入运行。

③. ISR 使用 i_return 系统调用,则 ISR 退出运行,pSOS+ Kernel 选择 Ready 队列中优先级最高的任务投入运行(这一任务并不一定是被 ISR 打断的前运行任务)。

上述三类运行任务的切换,其具体的 pSOS+ Kernel 运作过程并非完全一样,但彼此之间差别不大。为了简单起见,我们以 2 为例对切换过程作一简单叙述。这一过程可细分为 4 个步骤:

- ① 任务 A 运行信息保存(_t_save proc far)

这一过程主要完成修改系统工作标志,保存切换点地址及运行信息、任务 A 栈调整、栈指针保存、栈切换(从任务 A 堆栈切换至内核堆栈)、参数及返址入栈等一系列工作。

- ② 任务 A 入就绪队列(void t_in_chain)

这一过程将任务 A 的 TCB 块按优先级顺序插入就绪队列。

- ③ 选择一个高优先级任务 B(void t_choice())

按一定算法从就绪队列中选出最高优先级任务 B 的 TCB 块,并使运行指针指向它。

- ④ 将任务 B 投入运行(_t_run proc far)

从系统内核堆栈切换到任务 B 堆栈,用任务 B 的 TCB 块中保存的信息恢复上次运行被打断的地方,恢复任务运行环境,于是任务 B 开始继续运行。

(3) 任务的状态迁移

① 运行态到阻塞态转移的条件:运行态的任务作下列动作之一即发生到阻塞态的转移,这些动作是以等待方式(wait)进行的:

- A. 从空消息队中请求一消息;
- B. 等待一个事件但条件尚未满足;
- C. 请求一个信号量但目前不满足;
- D. 请求一段内存但目前不满足;
- E. 等待时间间隔或特定时间点。

- ② 阻塞态到就绪态的转移:下列任何一种情况的出现均能发生转移:

- A. 消息到达消息队而被封锁任务正在等待消息且处于等待队列的首位;
- B. 一个事件被送到被阻塞任务且满足该任务等待的条件;
- C. 信号量被返回且被阻塞任务处于等待队列的首位;
- D. 存储段返回 pSOS+且满足等待内存需求任务申请的内存段要求;
- E. 一个正在等待消息、事件、信号量或内存段且带有时超要求的任务在它所提出的时超间隔到达时;

F. 一个已被延时的任务当其延时间隔到达或其唤醒时刻到达时;

G. 一个正在等待消息、信号量或内存区的任务而该消息队、信号量或内存区对象被另一任务删除时。

③ 阻塞态到运行态的转移: 满足上述 2 中条件的任务当其优先级高于当前运行任务且该任务的抢占标志使能时。

④ 就绪态到运行态的转移: 一个就绪任务在当前运行任务出现下列情况之一时就变成运行任务:

A. 当前运行任务被封锁, 并且就绪态任务优先级最高;

B. 当前运行任务的抢占标志重新使能且就绪任务的优先级比它高;

C. 当前运行任务的抢占标志已使能而它将自己的优先级或就绪任务的优先级改变, 使得该就绪任务的优先级高于当前运行任务及其它就绪任务;

D. 当前运行任务的轮转方式使能, 其运行时间超出其所占的时间片, 而就绪任务具有相同的优先级;

⑤ 运行态到就绪态的转移

A. 运行任务或中断服务程序发出系统调用, 使得某被封锁任务满足封锁态到运行态转移的条件;

B. 当 4 中后三种情况发生时, 当前运行任务将变成就绪态。

其实在 pSOS+中, 除上述三种状态外, 有第四个较次要的状态: 挂起态 (suspending state), 一个被挂起的任务在其被直接恢复之前是不能运行的。挂起与封锁相似但其功能是不同的。首先, 任务只能封锁自身但能挂起其它任务或自挂, 其结果是一个就绪任务可因挂起而脱离就绪态。其次, 一个被封锁的任务也能被挂起, 因而该任务只有在解除封锁和解挂后才能成为就绪或运行任务。

(4) pSOS+任务管理:

pSOS+通过任务管理的系统调用提供动态创建和删除任务以及控制任务属性的手段。这组系统调用有:

t_create	创建新任务
t_ident	获取任务的 ID
t_start	启动新任务
t-restart	重新启动任务
t_delete	删除任务
t_suspend	挂起任务
t_resume	恢复被挂起的任务
t_setpri	改变任务优先级
t_mode	改变任务调用任务的方式位
t_setreg	设置 (写入) 任务的便条寄存器
t_getreg	获取 (读出) 任务的便条寄存器

① 任务的创建以及激活

任务创建时将其属性传给 pSOS+, 使 pSOS+对它注册以便执行和竞争其它系统资源。

任务的代码段必须驻留在内存, 可以固化在 ROM, 也可在启动或创建时装入 RAM。任务的数据区可以静态分配也可动态地从 pSOS+申请。

建立一个任务需要两个系统调用, 即 t_create 和 t_start。父任务调用 t_create 创建任务, 传递的参数有用户指定的名称、调度用的优先级、堆栈尺寸和工作标志, 该标志仅在多处理器系统中有用。

`t_create` 建立一个任务控制块 (TCB)，分配一段足够大的内存，用于任务的监控栈和用户栈，建立任务的页面描述块以及其它必要的扩充域。扩充域是一段额外的内存区，用于可选的特性。例如，带协处理器的系统需要 FPU（浮点处理单元）保存区，其它系统部件（pHILE+，pREP/C+，pNA+等）所需的内存。这些内存段连接到 TCB。`t_create` 回送 pSOS+ 为新任务分配的 Tid（任务标识）。

`t_start` 必须在任务创建后调用，它提供新任务的启动地址，控制其初始执行行为的方式字以及一张可选的自变量表。任务启动后，处于就绪态，随后按其优先级参与调度。

有两个任务是先天存在的。一个是 ROOT（根）任务，它由 pSOS+ 在初始化期间创建并启动，启动后 pSOS+ 将控制交给 ROOT 任务。另一个是缺省的 IDLE（空闲）任务，也是系统初启时建立的。系统内同时可存在的总任务数由配置时的参数 `KC_NTASK` 指定。

② 任务控制块

任务控制块 (TCB) 是 pSOS+ 在建立每个任务后为其分配并管理的系统数据结构，TCB 包含任务执行时系统需要了解的所有信息，包括名字、优先级、剩余时间片以及环境。通常，环境是指机器寄存器状态，当任务运行时，其环境是随时动态变化的，因此环境是这些寄存器的实际内容。当任务不运行时，其环境被冻结，并保存于 TCB，以便下次运行时的现场恢复。

在 TCB 还存在某些管理用的结构，被 pSOS+ 用来管理各种队列和结构。例如，TCB 可以处于下列几种队列之一，即就绪队、消息等待队、信号量等待队、内存区等待队和超时队。

③ 任务栈

任务创建期间，pSOS+ 为之分配一个栈。任务栈是每个任务私有的内存，由任务自身的代码（包括 ASR）使用。当发生运行任务切换时，pSOS+ 自动进行栈区的切换。

pSOS+ 的系统调用和中断处理不使用任务栈，而使用它自己的系统栈，但是在处理 I/O 请求时使用任务栈。pSOS+ 的 I/O 调用开销 60 个字节的栈区，另外还需考虑目标设备驱动程序对栈区的需求。

④ 任务的消亡

任务可由自身或被别的任务终止，`t_delete` 调用将删除一个已创建的任务并将它的内存段返回 `regin 0` 域，释放其它资源。

通常，`t_delete` 应该用于任务删除自身。当必须删除另一任务时，`t_delete` 使得该任务没有机会去完成必要的清扫工作。一个较为可取的办法是调用 `t_restart` 促使任务返回其原始的入口点。由于 `t_restart` 可以传递可选的自变量表，于是可以用来区分是 `t_start`，还是有效的 `t_restart` 或者是自身删除的请求。在自身删除时，任务可以归还所分配的资源，执行必要的清理工作，然后调用 `t_delete` 删除自身。

⑤ 任务寄存器

每个任务有 32 个软件便笺式的 32 位寄存器，它们位于任务 TCB 中，可以利用 `t_setreg` 和 `t_getreg` 调用来设置和读取。这些寄存器的用途是为每个任务提供一组标准的全系统可用的有名变量，可被其它任务，包括其它处理器节点上的远程任务来读写。这些任务寄存器中有 16 个保留给系统使用，剩余的 16 个可被用户程序用于特定用途。

5、pSOS+存储分配

pSOS+ 的存储管理提供动态分配可变尺寸的段和固定尺寸缓冲区的手段。相应的系统调用有：

<code>rn_create</code>	创建内存域 (region)
<code>rn_ident</code>	获取内存域的 ID

rn_delete	删除内存域
rn_getseg	从内存域中分配一内存段
rn_retseg	把一内存段返回内存域
pt_create	创建缓冲区分区 (partition)
pt_ident	获取缓冲区分区的 ID
pt_delete	删除缓冲区分区
pt_getbuf	从分区中分配一缓冲区
pt_retbuf	把一缓冲区返回分区

(1)、域和段

内存域(region)是一个用户定义的、物理上连续的内存块,可具有特殊的隐含的属性。例如,一种属性是严格局限在本地 RAM 中,而另一种则可能是全系统可访问的 RAM。region 间必须是互相分开的,而且应定位在长字边界上。

与任务类似,region 也是由 pSOS+管理的动态抽象的对象。一个域由 rn_create 调用创建,输入参数有用户指定的名字、起始地址以及长度和单位尺寸,该调用返回所建域的 ID (Rnid)。其它需要了解该域的任务只能利用名字通过 rn_ident 调用获得该域的 Rnid。

段(segment)是域中一片尺寸可变的内存,可通过 rn_getseg 由 pSOS+分配,该调用的输入参数有域的 Rnid,段的尺寸和一个可选择的等待域中存在足够自由内存的时超,返回该段的地址。

(2)、专用域

pSOS+至少需要一个内存域,该专用内存域命名为“RN#0”,其域标识 RnId 为 0,其起始地址和长度在 pSOS+的配置表中指定。pSOS+启动时,首先占用 region 0 底部作为 pSOS+数据段,用于 TCB 之类的数据区和控制结构。域 region 0 的剩余块作为任务栈以及满足用户 rn_getseg 调用提出的申请。与其它 region 域相似,应用程序可以调用 rn_getseg 和 rn_retreg 从域 0 分配和释放尺寸可变的内存段。

(3)、分配算法

每个内存域均有一个单位尺寸参数,它在创建时指定。这个与域有关的参数是该域的最小分配单位,它必须是 2 的幂,而且必须大于等于 16 字节。任一由 rn_getseg 分配的段为最接近单位尺寸的整数倍。

若任务申请一个段但域内没有足够的自由空间,则任务可选择等待,并指定时超值。若时超到而仍不满足,则返回错误。等待队列的排队规则可以是按优先级的,也可以是 FIFO 原则,在区域创建时指定。

(4)、分区与缓冲区

内存分区 (Partition) 是用户定义的物理上连续的内存块,被分割为一组尺寸固定的缓冲区。与内存域相似,分区也是由 pSOS+管理的动态抽象对象。分区由 pt_create 调用创建,需要输入的参数有用户指定的名字、起始地址和长度以及缓冲区尺寸,输出 pSOS+分配的分​​区标识 (PtId),其它任务需要了解该分区,可通过名字调用 pt_ident 获取该分区的 PtId。

分区有如下限制:起始地址必须在长字边界,缓冲区尺寸必须为 2 的幂且要大于等于 4 个字节。分区可以用在紧耦合多处理器系统中作为处理节点间频繁的数据交换。

6、pSOS+通信、同步和互斥机制

pSOS+的应用程序通常分割为一组任务和中断服务程序 (ISR),在概念上,每个任务是可以并发执行的独立动作实体,但是操作系统需要交换数据、同步各动作或共享互斥的资源。

为实现任务到任务以及 ISR 到任务的通信、同步和互斥，pSOS+提供三种机制：消息队、事件和信号量。

(1)、消息队列

消息队列提供十分灵活和通用的通信和同步机制，有关的系统调用有：

q_create	创建消息队列
q_ident	获取消息队列的 ID
q_delete	删除消息队列
q_receive	获取或等待队列中的消息
q_send	把消息排入队列尾
q_urgent	把消息排入队列头
q_broadcast	广播一条消息给消息队列

消息队列由 q_create 调用创建，输入参数有用户指定的名字和几个特征，包括任务等待队的排队规则是 FIFO 还是优先级、消息队列长度是否受限以及消息缓冲区是否私有等。

消息队列具有两类用户，即发送者和接收者，它们可以是任务或 ISR。发送消息的调用有三种：q_send， q_urgent 和 q_broadcast。发送消息时，当消息到达队列时没有等待任务，则拷贝到消息缓冲区，然后将其插入消息队列。q_send 将消息插入消息队列之尾，q_urgent 则将其插入队列之首。当消息到达队列时有一个或几个任务在等候，则直接给予排在等待队列中的头一个任务，因此不需要缓冲。q_broadcast 将消息广播给所有正在等候的任务，这是用单个系统调用唤醒多个任务的有效手段。从消息队列中接收消息的唯一调用是 q_receive，若当时没有消息，则任务可直接等待、超时等待或无条件返回。如任务直接等待，则按 FIFO 或优先级顺序排入等待队列。

在早期的 pSOS+版本中，消息是固定长度的含有 4 个长字的结构，后续的版本已经支持变长队列。消息的内容与就应密切相关，它可以装载数据、数据指针、数据尺寸、发送任务的 Tid，响应队列的 Qid（消息队标识）或上述诸项的某些组合，它也可纯粹用来同步而不带任何消息。当消息到达队列而无等待任务，消息需拷贝到消息缓冲区并排入消息队。消息缓冲区占用的内存长度是消息长度加上一个长字，增加的那个长字由 pSOS+用作链指针域。

(2)、事件

pSOS+提供一组由事件同步的机制，每个任务均有 32 个事件标志，构成一个按位操作的 32 位字。其中的高 16 位保留给系统使用，低 16 位事件标志完全由用户定义。任务之间和任务与 ISR 之间由事件来同步的系统调用有：

ev_receive	获取或等候事件
ev_send	将事件发送到指定任务

ev_send 用来把一个或几个事件发送给另一任务，而任务可通过 ev_receive 去获取一个或几个属于自己的事件，而且可选择等待、带时超的等待或不等待。事件的一个重要特点是任务可以等待一个事件、几个事件中的一个（或）或者几个事件中的全部（与）。

任务和 ISR 均可将一个或几个事件送往另一任务，若目标任务并未等待事件或它所等事件与送来的事件不符，则 ev_send 将这些事件位打开，即使它们挂起；若目标任务正在等待送来事件中的一些或全部，则等待任务解锁并进入就绪，而其余不匹配的事件仍被挂起。

事件与消息的不同之处有：

- (1) 事件可用于任务同步，但不能直接携带信息；
- (2) 事件是点对点的，即必须指明具体接收任务，而消息是送往消息队列的，不需要了解具体接收任务，因此可以多点对多点。
- (3) ev_receive 一次可等待多个事件，而 q_receive 一次只能从消息队中接收一条

消息。

(4) 消息可自动缓冲和排队，而事件既不计数也不排队。若一个事件已经挂起而第二个同类事件送往同一任务，则第二事件将丢失。

(3)、信号量

pSOS+提供一组熟悉的信号量操作，通常这是实现互斥的最有用的资源标签，有关的系统调用有：

sm_create	创建信号量
sm_ident	获取信号量 ID
sm_delete	删除信号量
sm_p	获取或等待信号量
sm_v	返回信号量

信号量由 sm_create 动态创建，其输入参数有用户指定的名字、初始计数和几个特征，包括等待任务的排队规则是 FIFO 还是优先级；返回信号量标识 (SmId)。初始计数应该反映该资源可占用的有效标签数。

pSOS+提供两个传统的 P 和 V 操作：sm_p 和 sm_v。sm_p 请求使用信号量，若信号量计数非零，则其值减 1，同时操作成功；否则，调用任务可选择等待、带时超等待或直接返回，如果选择等待，则按 FIFO 或优先级顺序排队。sm_v 返回信号量，若无任务等候，则信号量计数增一；若有任务等待，则等待队列的第一个任务脱离等待队列并变为就绪。

(4)、异步信号

每个任务可选择具备一个异步信号服务子程序 (ASR)，其目的是允许任务有两个异步部分，即主体和 ASR。本质上讲，就象一个任务与另一个任务可以异步执行一样，ASR 在任务内提供类似功能。通过信号，一个任务或 ISR 可以迫使另一任务脱离正常的执行流程，由主体进入其 ASR，因此信号提供了一种软件中断机制。这种异步通信能力是许多系统设计需要的，否则类似功能需要依赖消息或事件之类同步机制，而这需要付出较大的性能损失。

有关的系统调用有：

as_catch	公布任务的 ASR
as_send	发送信号到任务
as_return	自 ASR 返回

异步信号由用户定义，每个任务有 32 个信号，按位编码为一个长字。要接收信号，任务必须用 as_catch 公布一个 ASR。as_send 用来发送一个或几个异步信号到指定任务，因而迫使该任务进入它最后一次公布的 ASR。在 ASR 结束时，调用 as_return 允许 pSOS+把任务返回到原来执行位置。

任务只能具备一个有效的 ASR，ASR 执行于任务本身的环境内，从外部看，不可能区分任务运行在主体上还是它的 ASR 上。

7、时间管理

时间管理提供以下功能：

- 1、管理日历时间和日期；
- 2、任务等候消息、信号量、事件和段的超时；
- 3、在预定间隔或指定时刻到达后唤醒或送一告警给指定任务；
- 4、追踪运行任务的时间片并实现轮转调度机制。

这些功能依赖于周期性定时器中断，而当缺少实时钟或定时器硬件时将无法工作。显式的时间管理功能有：

tm_tick	通知 pSOS+一个时钟嘀嗒的到达
tm_set	设置时间和日期
tm_get	获取时间和日期
tm_wkafter	间隔到达后唤醒任务
tm_wkwhen	指定时刻到达时唤醒任务
tm_evafter	间隔到达后发送事件给任务
tm_evwhen	指定时刻到达后发送事件给任务
tm_every	每个间隔周期发送事件到调用任务
tm_cancel	取消告警定时器

(1)、时间单位

系统的时间单位是一个时钟嘀嗒，定义为两个 tm_tick 系统调用间的间隔。通常，tm_tick 由每个定时器中断时引导的实时钟 ISR 调用。tm_tick 的频率确定了系统时钟的粒度，显然频率越高，则时间分辨率越高。但从另一方面看，处理每一时钟嘀嗒均需化费少量系统开销。

(2)、时间和日期

pSOS+维护真正的日历时间和日期，包括对闰年的补偿。一对系统调用 tm_set 和 tm_get 可用来设置和获得日期和时间。

(3)、超时

时间管理提供别的系统调用需要的超时机制，这些调用有 q_receive、ev_receive、sm_p、rn_getseg。

(4)、绝对与相对定时

存在绝对与相对两种定时，相对定时是指定以嘀嗒数计量的间隔，绝对定时则指定日历日期和时间。tm_wkafter 和 tm_evafter 两条系统调用接受相对定时参数，tm_wkwhen 和 tm_evwhen 两条系统调用则为绝对时间。

(5)、唤醒与告警

存在两种任务响应定时的不同方式。第一种方式是进入睡眠（即封锁）而在预期时间唤醒，这种调用为 tm_wkafter 和 tm_wkwhen。第二种方式是设置告警定时器并继续运行，这种异步方法由 tm_evafter 和 tm_evwhen 提供。当告警定时器到时后，由 pSOS+调用 ev_send 将指定事件送到任务，当然任务必须调用 ev_receive 测试或等待该调度事件。

告警定时器提供几个有趣的特性。首先，调用任务在定时器减量期间仍能运行。第二，任务可配备一个以上的不同时间的告警定时器，对应多个期望事件。最后，告警定时器可由 tm_cancel 取消。告警也可用于重复的间隔请求，即采用 tm_every 可实现周期性形为。

8、中断服务例程（ISR）

ISR 是任何实时系统的评估标准。一方面 ISR 处理中断并完成一些诸如设备复位、数据读/写等紧急活动，另一方面 ISR 可驱动一个或几个任务并使之响应和处理相应的中断服务。

ISR 和任务之间同步或通信的系统调用有：

as_send	发送异步信号到任务
ev_send	发送事件到任务
i_return	中断返回
k_fatal	致命错误处理
m_ext2int	外部地址转换为内部地址
pt_getbuf	从分区返回缓冲区

pt_retbuf	缓冲区返回分区
pt_sgetbuf	从分区获得缓冲区
q_broadcast	广播消息到队列
q_receive	自消息队列获取消息（不等待）
q_send	将消息发到消息队尾
q_urgent	将消息发到消息队头
sm_p	获取信号量（不等待）
sm_v	返回信号量
t_getreg	获取任务的软件寄存器
t_resume	唤醒被挂起的任务
t_setreg	设置任务的软件寄存器
tm_get	获得时间和日期
tm_set	设置时间和日期
tm_tick	通知一时钟嘀嗒到 pSOS+

由于 ISR 不能封锁，q_receive 或 sm_p 由 ISR 发出则不得等待，无条件返回。

9、致命错误的处理

由系统调用引发的多数错误，如参数错和临时性资源枯竭等，则直接将它们返回调用者。少数几类错误，如配置错误和内部资源枯竭，将无法继续运行，称之为致命错误。此外，用户应用软件也能调用 k_fatal 产生一个致命错误。当致命错误出现时，pSOS+首先检查 pSOS+配置表中的 KC_FATAL 项目，该项目用来指定用户提供的致命错误处理程序。当该项非零时，pSOS+就调用该用户过程并传递错误信息。若该项为零，则 pSOS+检查 pROBE+系统调试分析程序的存在性。若存在，则将控制传递给它。如果用户提供的错误处理程序和 pROBE+均不存在，则 pSOS+通过陷井进入系统缺省的异常处理（异常号 10，向量 VBR+\$50）。

10、关于pSOSSystem的一个总结：

表 10-1 和表 10-2 分别列出了 pSOSSystem 的开发信息和技术信息的一个总结，这对比较实时操作系统很有帮助。

表 10-1：pSOSSystem 开发信息的总结

开发平台：	Windows NT, Sun Solaris, SunOS, HP-UX, RS6000,
支持处理器：	x86, 68k, PPC, 683xx, CPU32(+), MIPS R3000/4000/5000, Coldfire 510x/520x, i960, ARM 7 (TDMI), SH1/2/3, M32R
支持实验板：	68k: Motorola, Force, EST PPC: Motorola, Force, EST, IBM MIPS: IDT, LSI Integra x86: PC, Intel, National ColdFire: Motorola
支持编译器：	Assembler, C, C++
支持工具：	pRISM+, 全集成开发环境, 含源代码调试器 (Sp0TLIGHT), C/C++ 编译器 (e g Diab), 汇编器, 连接器, C/C++ 开发环境 (SNiFF+),

	嵌入式系统监视工具 (Esp), CORBA
支持网络协议:	ZeroCopyTCP/IP, TFTP, FTP, Telnet, BOOTP, NFS, RPC, Streams, SNMP, RMON, RIP, x. 25, HDLC pSHELL, RIP, OSPF2, PPP, SLIP, SNMP v1 & v2 Streams: OpEN, X. 25, ISDN, ATM, OSI, IPX/SPX, FrameRelay HTTP Server, JAVA
标准:	POSIX. 1c
开发环境:	交互式开发
软件供应形式:	目标代码
编程语言:	C, C++
图形界面:	X-Windows, Motif, 产品自带图形界面
产品有效模块:	浮点运算模块, 通讯协议模块, 缓存模块, 网络支持模块, 数学库, 文件系统, HTTP 服务器, 文件管理器, ISO9660 文件系统, 浏览器
多线程调度策略:	固定优先级, 轮转调度, 时间片, 动态改变优先级,
优先级倒转防止策略:	互斥法

表 10-2: pSOSystem 技术信息:

系统内核:	系统内核内存:	进程最小存储区:	线程最小存储区:	消息最小存储区:	优先级数目:	线程最大数目:	典型线程切换时间:	典型进程切换时间:
2. 5K- 40K	0. 5K- 5K	不详	300 字节	200 字节	不详	无限制	1000 时钟周期	不详
最大中断潜伏期:	系统时钟最小分辨率:	多进程:	多线程:	多处理器:	MMU 支持:	集成 JAVA 支持:	自动代码生成工具:	RMA:
1000 时钟周期	支持	否	支持	支持	支持	支持	否	否

十、实时操作系统 Nucleus

1、Nucleus 概述

Nucleus 是 Accelerated Technology 公司为实时嵌入系统设计的可扩展的多任务操作系统, 约 95%的 Nucleus PLUS 代码用 C 语言编写, 因此它能很方便移植。同时可提供 Web 支持, 网络, 图形包, 文件系统等模块。Nucleus 最大的特点是价格低廉并且全部提供源代码, 免去用户购买 license 和付 Royalties, 非常符合中国的国情。Nucleus 的另外一个特

点是, 应用程序开发者只需通过 DLL 动态连接库便可进行任务级调试, 无需编写繁琐的 BSP。Nucleus 在中国市场增长较快, 98 年销售一百套左右, 现在在中国的各个行业得到广泛应用。

下面我们以 Nucleus PLUS 为例来介绍 Nucllus。

2、Nucleus PLUS 的主要技术特点

(1) 良好的可移植性

Nucleus PLUS 的 95%是由标准 C 编写的, 其余 5%为汇编语言。汇编语言部分被分为三个依赖目标系统的部分: 初始化、线程管理、定时器管理。内核的其余部分只需要重新编译后即可用于新的 CPU 结构。

(2) 高度的模块化

Nucleus PLUS 具有高度模块化的设计。例如, 队列服务与管道服务间毫无联系, 信号量与事件服务间也是一样的, 所有 Nucleus PLUS 的功能或组件按逻辑关系被划分为单独的文件, 这样, 在最终的执行文件中只包含那些需要的内容。此种划分要比标准的对管道、队列等的划分还要细。在每一组件中, 文件集分隔的方式可使用户应用程序中的内核最小。例如, 常用的队列服务包括创建、删除、发送、接收, 特殊用途的的队列服务包括强制前置和广播, 此类中的每一个服务都在一单独文件中维护。如果用户使用队列, 但不需要广播, 则此模块就不需要包含在执行文件中。由于可利用用户自定义的工具库, 所有的操作都可自动完成。

(3) 开发的方便性

在创建一嵌入系统时有两个阶段, 首先是设计、编码、测试, 当应用程序完成后, 则进入产品循环。在不同的阶段的需求是不同的。在设计编码阶段, 用户需要在内核中建立广泛的数据收集及调试功能, 这样才可方便的查出错误。但当系统作为产品开始运转时, 这些功能的系统开销则是不必要的。

Nucleus PLUS 包括了许多的可在初始化状态下提供帮助的功能, 而它们可在开发阶段很轻易的除去。通过添加一编译阶段的条件变量 (-DNU_NO_ERROR_CHECKING), 可取消对服务调用的错误核查。因此, 在普通的创建进程中, 将检查所有的 Nucleus PLUS 服务调用的变量, 除非专门声明取消此检查。如果用户希望内核检查堆栈的使用情况, 可通过定义编译阶段的条件变量 -DNU_STACK_CHECKING 来实现。如取消此项定义, 用户可减少系统的开销。Nucleus PLUS 同时提供一历史记录功能, 它可记录应用程序与内核间的相互作用过程。通过定义编译阶段的条件变量 -DNU_HISTORY_SAVING, 经过编译的代码在系统中运行时将收集历史信息并将其记录到用户定义的记录结构中去。一旦在编译阶段包含了此定义, 用户可通过一服务调用来决定是否打开历史记录, 并可在运行期间通过服务命令来恢复历史记录功能。

另一有用的特性为 Nucleus PLUS 可收集任一内核组件的状态信息, 每一组件可被查询

其状态。为查询组件状态，用户首先请求获得在一组件内的元素数（如，定义的队列的数目），在此信息的帮助下，用户程序可提供一指针指向一足够大的数组以保存每一已创建的元素的指针。内核将把这些指针的内容添入数组。最后，对每一指针，用户可恢复组件的每一元素的专有信息。

(4) 防止优先级倒置的机制

Nucleus PLUS 内核的组件是全局的。例如，多个任务可申请队列服务。每个队列的控制信息保留在一全局数据结构中，因此，必须加入保护以防止一任务使用另一任务正在使用中的服务。有几种途径可实现这一要求。内核可在它正在使用服务时屏蔽掉中断，也可在服务临界区域屏蔽掉那些可能使用全局变量的中断，或者可执行一类似监视器的服务。在 Nucleus PLUS 中采用了第三种方法，它在防止优先级逆转的情况下达到了最短的中断等待时间。即，当一任务申请内核服务时，该服务调用 NU_Protect，它在数据结构中记录了每一任务使用何种服务的情况。当另一任务使用该服务时，当前任务被挂起，而拥有此保护的任務被恢复（通过一快速调度机制，而非正常的调度），当它到达服务的非保护点后，内核对优先级进行比较，如申请保护的任务的优先级比刚刚释放保护的任务的优先级要高，则当前任务被挂起，高优先级的任务被恢复。这种机制用于防止低优先级的任务在保护内核数据结构时使高优先级任务产生延迟。

由于这是一种软件功能，并且不需要屏蔽中断，一专用的允许中断服务使用内核服务的机制得到执行。当一中断发生时，一在应用程序中断初始化时期定义的 LISR 得到执行。当该中断需要调用一核心服务时，LISR 将调度启动一 HISR，HISR 将在 LISR 退出及调度程序恢复后作为超级线程执行。在此种方法下，HISR 的优先级比系统中任何任务的都高。这样也防止了使用服务保护机制所产生的延迟，在除去一种情况下，Nucleus PLUS 防止了在发生时的优先级反转。当一低优先级任务获得一信号量且未在一高优先级任务需要此信号量前将其释放，高优先级的任务将反转。这种情况将在内核中得到处理，但是，管理它所需的开销要比从中获得的利益要大的多。因此，在有信号量的情况下防止优先级反转是一个关于系统的问题。因为 Nucleus PLUS 允许在上述的情况下动态修改任何任务的优先级，高优先级的任务可将低优先级的任务升级为同级的或更高级的任务来避免长期的反转。

(5) Nucleus PLUS 系统调用的主要特点

Nucleus PLUS 时超处理与 pSOS+ 没有太大的差别。为执行系统调用时可能产生的挂起，需定义一时超参数，其值可为永远挂起（直到此请求服务为可用）、立即返回（如服务无效则立即返回一错误码）或等待一段时间。

与 pSOS+不同的事，Nucleus PLUS 系统调用入口有一个十分重要的特征，那就是需要用户为控制块及需要的内存区域定义内存。所有 Nucleus PLUS 组件由一单独的控制块管理。例如，当用户定义一任务，则必须给内核提供一块内存区域以管理该任务。此外，一些组件要求的更多，如堆栈空间，用户必须为它们为分配空间。乍一看来这对用户造成了麻烦，但实际上此项功能是很有益处的。通过定义内存区域，用户可更紧密的控制系统的特性。例如，当用户想拥有一高速任务时，可将任务的控制块和堆栈放置在 SRAM 中。另一方面，用户可能想将固定块数的内存放置到不掉电 RAM 外。此种机制使用户有全面的控制权，可更好的适应各种实时系统的运用。

(6) Nucleus PLUS 的任务管理

在 Nucleus PLUS 中，TM (Task Management) 组件提供所有管理多任务环境所需的所有功能。任务的状态可以是不存在、已创建、准备就绪、运行、挂起的多种状态中的一种，并且在这些状态之间切换。一般而言，除非在任务显式调用了睡眠、挂起或恢复的系统调用的情况下，否则任务的状态转换不是公开的。通常，一任务的状态转换与系统的调用有关。例如，当一任务等待一事件的发生，它处于挂起状态。当一事件管理器被告之对应事件已得到满足时，挂起的任务按照其优先级被放置到准备就绪列表。系统服务总是在判断是否执行一个会导致状态转变的服务。如果在此种状态，调度程序将被调用，新的任务将得到运行。因此，调度程序从不在长列表中搜索任务去执行。它只是简单的检查准备就绪队列的头，TM 也具有修改任务特性的功能，包括其优先级、抢占状态、时间片等。所有这些可在运行时被修改。为帮助低优先级任务的循环调度，Nucleus PLUS 允许为一任务指定一时间片。当高优先级任务挂起后，有一时间片的低优先级任务运行完它的量程（时间量），然后将控制权交给下一同优先级的任务（假设它们未被高优先级的任务抢占）。和许多实时任务需要一空闲任务（idle task）一样，Nucleus PLUS 也有一个空闲任务，即只要没有其它任务准备就绪，则运行空闲任务，只是 Nucleus PLUS 的空闲任务处于调度程序中。调度循环自身为少量由汇编语言编写的指令。由于只用到了最少量的寄存器，因此，当响应中断时只需要保存少量的寄存器（当没有任务准备运行，只有一中断引起的需要调度的内容时）。这意味着空闲状态是极端有效的并允许系统以可能的最快方式响应外部事件。

(7) Nucleus PLUS 的任务间通信机制

Nucleus PLUS 提供三种在任务间传送数据的机制：管道、队列及信箱。任务间通信机制在使用上与 pSOS+没有明显区别，只做简单介绍。

管道为面向字节的通信（byte-oriented）机制，其允许用户定义一批条项及每个条项的字节数。队列为面向 32 位长字通信（32-bits oriented）机制，当然也允许用户定义一

面向字节的消息。在这两种情况下，在消息的总长未超过管道或队列总长的情况下，可以按大于或小于每条项定义的长度发送及接收消息。第三种数据传输的机制是信箱，只有单个条项，发送或者接受 4 个 32 位字长的消息。在所有情形中，使用内部通信机制的用户可按需求在满或空的条件下挂起。

Nucleus PLUS 通过尽可能的避免拷贝一全局区域来优化任务内部通信机制。例如，任务 A 在等待队列 a 的消息，任务 B 将发送一消息去队列 a，数据将从任务 B 的发送区拷贝到任务 A 的接收区。当任务 A 未在等待时，数据将从 B 的发送区拷贝到队列 a 的全局区。

强制前置和广播功能在适当的地方对任务内部通信组件而言是可用的。强制前置允许一高优先级条项放置在一队列或管道的所有消息的前面。广播则将给所有在队列或管道上等待消息的任务发送广播消息。

(8) Nucleus PLUS 的任务间同步机制

Nucleus PLUS 提供三种任务内部同步的机制：可计数的信号量、事件组、信号。这些机制同 pSOS+ 没有明显区别，只做简单介绍。

可计数的信号量用于保护如串口及其他共享设备等的系统资源。它提供可供使用的设备的范围。例如，用户可能有一个可供 10 个任务同时访问的设备，此时，可创建一有 10 个可能使用的信号量。当第 11 个任务企图访问该设备时，它可选择挂起，直到该设备可用时为止。

事件组为一 32 位的标志，它可用以设置和获取。一任务可通过对此 32 位的逻辑与及逻辑或的组合来等待及设置一事件。此外，当一任务等待一事件时（或当它已存在时获取它），它拥有一可清除它获取的事件的选项。

信号是线程用于执行其外部进程的一种机制，它用于有例外发生的情况。每一个任务可以申明一信号句柄（Signal Handle），如系统中的另一任务检测到这个信号发生的条件，它产生一例程，去调用此信号的处理组件，此处理组件在接收信号的任务的普通堆栈的顶端创建一专用堆栈，当接收信号的任务向下执行时，它启动其信号句柄。系统将依赖于任务接收到的信号的值，去执行某些进程。当此进程完成及任务从信号句柄中返回后，系统返回到在信号发生时其所处的状态。

(9) Nucleus PLUS 的内存管理机制

Nucleus PLUS 包括两种内存管理的类型：内存池及固定内存分区。内存池类似于标准 ANSI C 库提供的 malloc 和 free 服务调用，它们在许多方面不同。Nucleus PLUS 内存池管理允许用户从可分配内存的位置分配多个内存池。当一内存池创建后，用户可分配及释放任意大小的内存块（只要不超过内存池的最大数），当内存池中的内存不能满足用户需求时，用户可选择挂起任务一段时间或挂起任务直到有足够的内存可用。内存池管理在管理内存时是一灵活的机制。但是，由于它的可变的本性，它不具有可确定性。

固定分区内存管理比不上内存池管理灵活，但它具有可确定性。使用固定内存分区时，首先需创建一定定义了固定分区数的内存区，然后可自由的从该内存区分配及释放分区内存块。同内存池管理一样，当内存块不够时，用户可选择挂起。

(10) 定时器线程

Nucleus PLUS 提供可编程的定时器线程。它们可在一时间点或是一连续的时间内执行。一定时器线程是由 Nucleus PLUS 定时器管理功能启动的一执行线程。它们由标准 C 编写，通常会执行完毕。它们可象其他任务一样与内核交互。但它们不是任务。一旦它们完成了执行过程，它们将处于静止状态直到再次被定时器管理组件启动。

3、Nucleus MNT and Nucleus VNET

(1) 为什么需要 Nucleus MNT 和 Nucleus VNET

嵌入式系统开发人员在把产品加速推向市场时越来越感受到压力。由于这些产品通常需要软件和硬件，软件经常因为等待相应的硬件而延迟，而准备软件通常比硬件需要更多的时间。因此，当开发人员在开始一项目时被延迟了的情况下，时间便被浪费了。

解决此问题需要更强大的微处理器和复杂性持续增长的应用程序。大部分嵌入式系统开发人员转向于 off-the-self 实时核或操作系统以帮助他们管理此复杂性。因此，内核及操作系统的供应商有义务保证他们的用户在硬件开发出来以后有成效的使用时间。

Accelerated Technology 公司提供了一种允许用户在 PC 机上开发应用程序的软件形式——即提供三个内核专用的目标模块在一 PC 主机上运行的机制。在此种方式下，开发人员可使用 Borland 或 Microsoft 开发工具并且使用 Nucleus 创建他们的应用程序使其象 MS-DOS 可执行文件 (*.exe) 一样运行。Accelerated Technology 公司推出 Nucleus PLUS、Nucleus FILE（一 MSDOS 兼容的文件系统）、Nucleus NET（私有的 TCP/IP 堆栈）、Nucleus EPILOGUE（一 Nucleus PLUS 内核与 Epilogue 公司的 TCP/IP 堆栈的联合体）时，遵循了同一策略，即所有这些产品可在目标硬件上运行前可以在一本本地 PC 机环境中开发和调试。

Accelerated Technology 公司提供的对各种各样的嵌入式软件产品的应用程序编程接

口, 对软件而言, 在 PC 机和在目标系统上是相同的。在 PC 上经过预备开发后, 用户只需简单的为目标系统环境进行重编译。可使 Nucleus 系列软件产品在 MS-DOS 上使用的同一技术已被复制, 从而 Nucleus PLUS 可运行本地的 Windows NT 或 Windows 95 进程。Accelerated Technology 公司将此产品称为 Nucleus MNT。

为在 Windows NT 上提供 TCP/IP 服务, Nucleus NET 被植入到 Nucleus MNT 环境。为了使 Nucleus MNT 进程与其他使能了 TCP/IP 的设备间的通信更加容易 (那些在 Windows NT 上使用 WinSock 的和在 Windows NT 外部环境中的设备都在同一个实际的网络中), 一虚拟的网络设备被开发出来, 被称之为 Nucleus VNET。

(2) Nucleus MNT

Nucleus PLUS 被有意设计为可在不同 CPU 结构间移植。将系统的三个模块隔离开并将他们划分为依赖及不依赖目标系统的部分, 使得更容易移植。这三个部分分别用于执行初始化、调度及定时器管理。

Nucleus MNT 的各个模块被移植到 Windows NT 的线程环境。初始化模块为定时器及终端接口建立了一些终端向量。调度模块使用 Windows NT 的线程模块来管理任务间切换。定时器模块处理定时器滴答, 以促进 Nucleus PLUS 的任务睡眠、时间片、时超及定时器线程的能力。

Nucleus MNT 最有趣的方面是调度机制, Nucleus MNT 作为 Windows NT 的一个进程运行, 所有 Nucleus 的任务作为 Windows NT 的线程创建并分配到相等的 Windows NT 的优先级。Nucleus MNT 用于任务间的抢占及基于时间的调度。Windows NT 的线程功能简单的用于执行任务间必要的上下文保护和恢复。

(3) Nucleus VNET

仿真运行应用程序时需要创建标准的 Windows NT 设备驱动器来仿真各种外部设备, 在 Windows NT 环境下的仿真设备驱动器的例程可由 Nucleus MNT 来提供。Nucleus NET, Nucleus PLUS 的 TCP/IP 堆栈被移植到 Nucleus MNT 环境中并使用一 Windows NT 设备驱动器来处理网络通信, 这种产品称之为 Nucleus VNET。可使用同样的方法来模拟系统中的其他设备。通过写一 Windows NT 设备驱动器来模拟一件设备, Windows NT 应用程序可提供一该驱动器的用户接口, 这样, 便可模拟此设备的输入并保留其输出。

Nucleus VNET 允许 Nucleus MNT 进程通过共享内存进行彼此间的通信。图 10-1 显示了用于帮助 Nucleus VNET 通信的机制。一虚拟的 NDIS 驱动器增强了 Nucleus VNET。此启动器允许 Nucleus MNT 进程与 Windows NT 的 WinSock 应用程序及连接到一物理网络的设备间进行通信。

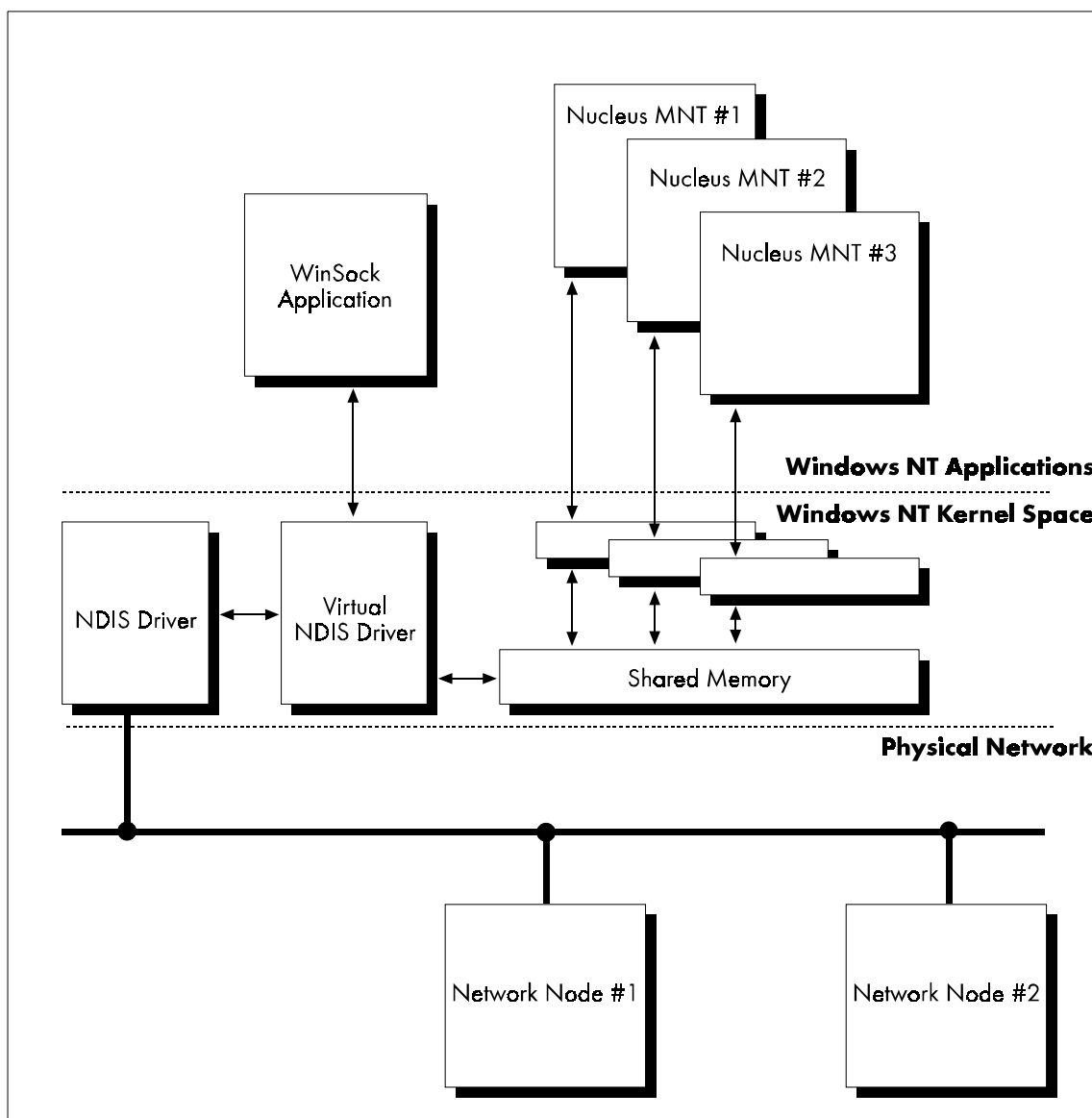


Figure 1 – Nucleus MNT/VNET

Nucleus MNT 可使用 Microsoft Visual C++ 工具集。因此，Visual C++ 的完整的开发环境包括编辑器、make/project 能力、编译器、库程序管理、汇编器、链接器及调试器都是可用的。

通过使用由 Nucleus MNT 提供的项目文件,开发人员几乎可立即加载并运行。同 Nucleus MNT 一起发布的版本文件被加载到一目录,用户可把项目文件加入 VC++ 的环境中,并调用“Build”菜单项以产生一 windows NT 控制台应用程序。此执行包括了一练习几乎所有 Nucleus PLUS 功能的演示程序。用户在开发自己的项目时可修改此程序或用自己创建的任务替代。

Nucleus VNET 允许 Nucleus MNT 进程经由共享内存来实现彼此间的通信。即多种版本的 Nucleus MNT 可在一 NT 机器上执行,每一个都有自己的 IP 地址。这些 MNT 进程可通过 TCP/IP 进行通信就象他们是独立的嵌入式设备。使用此种技术的开发人员具有在一 NT 机器上模拟一网络的能力。同时,此共享内存被用于通过虚拟的 NDIS 驱动器在 Windows NT 应用程序间通信。如果前向 IP 在 NT 机器上使能,则虚拟的 Nucleus 设备不仅可在彼此间,及本地 NT 工作站间通信,还可与执行了 TCP/IP 协议堆栈的远端设备通信。

描述 Nucleus VNET 工作原理的最好方法可能是对组成系统的每一组件进行描述。下述的讨论经常使用了图 10-1 作为参考。

最少有一个 Nucleus MNT 进程在执行,此 MNT 进程包括 Nucleus MNT 操作系统、Nucleus VNET 代码及用户的应用程序。VNET 驱动器作为 Nucleus MNT 进程和公共内存区域间的接口,也可作为 MNT 进程和虚拟 NDIS 驱动器间的接口。VNET 驱动器由 NT I/O 设备控制服务在应用层上调用。

公共内存区域(Common Memory Area)包含了一缓冲区列表及许多缓冲区环。在初始化时,所有的缓冲区都在自由环上,并且都是可用的。当缓冲区被使用时,它们被放到接收 MNT 进程的缓冲区环上。在缓冲区被读后它被放回自由环。虚拟 NDIS 驱动器也在公共内存区域拥有一缓冲区环,这样它可从虚拟网中接收消息包(packets)。虚拟 NDIS 驱动器也可访问公共内存区域,这样,它可从虚拟网设备上发送及接收消息包。同其他 NDIS 设备一样,虚拟 NDIS 驱动器安装在 NT 工作站上。它拥有自己的 IP 地址并且通过 NT 网络设置可进行配置,就象一 NDIS 驱动器对应的一物理以太网卡。

当 NT 工作站启动后,虚拟 NDIS 开始启动。此驱动器用于分配全局内存区域。开始地址写入系统寄存器,它可被其他驱动器和进程查到。VNET 驱动器由用户手工启动。VNET 必须在 NDIS 驱动器后启动,因为它依赖于 NDIS 来分配及初始化公共内存区域。在两个驱动器都启动后,一个或更多的 Nucleus MNT 应用程序将被执行。另一解释 Nucleus VNET 的途径是

描述一消息包从源地址取出并发送到目的地址。假设一消息包经过了协议堆栈，它将最终传到 Nucleus NET 驱动器进行发送。Nucleus NET 驱动器通过 NT 的 DeviceIoControl 调用与 VNET 驱动器通信。

将一消息包发送到网络介质上需要 4 个步骤，首先，Nucleus NET 驱动器从 VNET 驱动器中取得一缓冲区，返回一公共内存的偏移量。公共内存区域的开始地址在初始化过程中被映射到进程地址空间。将要发送的包被拷贝到缓冲区中并被发送到 VNET 驱动器中。然后，VNET 驱动器将缓冲区插入到接收进程的环中，最后，Nucleus NET 驱动器通知 VNET 驱动器将消息包传送到要接收它的进程上。

MNT 接收进程执行下面的步骤：在初始化阶段，一 NT 线程被创建用于接收消息包，此线程的作用类似一 ISR，当此线程运行时，它向 VNET 驱动器发送一申请可用缓冲区的请求。当没有接收缓冲区时，此线程挂起 VNET 驱动器的 I/O 请求的执行，当一消息包被此进程的 VNET 驱动器接收到后，此进程将恢复。当进程恢复后（或当有一可用包时），它从 VNET 驱动器中拷贝此缓冲区。在拷贝完成后，进程将缓冲区返回 VNET 驱动器以便重新使用。

(4)关于 Nucleus MNT 和 Nucleus VNET 的一个总结

由于 Nucleus VNET 需要两个 Windows NT 设备驱动器，它只能在 Windows NT 站上执行。Nucleus MNT 可在 Windows NT 或一 Windows 95 工作站。

Nucleus PLUS 被设计为高移植性的，它和由 Windows NT 线程环境提供的能力使 Nucleus MNT 变为现实。由于它是一本地 Windows NT 应用程序，Nucleus MNT 程序可使用标准 Visual C++工具集进行开发。这样的好处是，Visual C++提供的强大的调试能力可用于快速保证用户的应用程序进入工作。Visual C++提供的其他的调试器也可用于调试。

任何一个开发过嵌入式系统的开发人员对交叉开发的困难都很清醒。下载的时间会很长，问题经常存在与目标硬件或监视软件，其他附加的设备的也会造成混乱。通过避免对早期硬件的依赖及有时因开发环境的交叉所造成的复杂，Nucleus MNT 可用于大部分的将用于目标系统的 C 代码原型。此外，目标系统的部分外设的能力可被模拟或被 PC 机对应设备替代。这意味着开发人员可在项目中尽早的进行软件开发，以更快的将产品推向市场。

4、关于Nucleus的一个总结

表 10-1: Nucleus 开发信息的总结

开发平台:	WINDOWS NT,
-------	-------------

支持处理器:	x86, 68k, PPC, MCORE, National Semiconductor NS CR16A/16B/32A, 29K, TMS320Cxx, MIPS, H8, i960 (rx, hx, ix, kx, sx), Coldfire, MN10200, 8xx, PowerPC 403/505/601/821/823/860, SH
支持实验板:	Nucleus PLUS: Intel 8086/80186, Intel 80386/80486, Motorola 680x0/683xx, 68HC11, ARM, National Semiconductor 80486, IDT 30xx/4600/4640, LSI LR330x0/4001/4101/64008, Philips PR30100, Siemens C166/167.
支持编译器:	Metaware, Microtec Research, Diab Data, Microsoft, Borland, Watcom, Green Hills, Tasking, Intel GNU, IDT GNU, Algorithmics GNU, Intermetrics, Introl, SDS, ARM Ltd., TI, Hitachi C, GNU, Tartan C/C++.
支持工具:	Nucleus EDE, Nucleus DBUG+, Nucleus UDB, SDS, Paradigm, MULTI.
支持网络协议:	
标准:	ANSI
开发环境:	Nucleus EDE
软件供应形式:	源代码
编程语言:	C, C++, 汇编, Java
图形界面:	Nucleus GRAFIX
产品有效模块:	浮点运算模块, 通讯协议模块, 网络支持模块, 文件系统,
多线程调度策略:	固定优先级, 轮转调度, 时间片, 私有限制, after priority, 协同多任务.
优先级倒转防止策略:	Proprietary

表 10-2: Nucleus 技术信息的总结

系统内核	系统内核内存	进程最小存储区	线程最小存储区	消息最小存储区	优先级数目	线程最大数目	典型线程切换时间	典型进程切换时间
20K, 40K	2K, 4K	不详	168 + task	72 + task	256	无限制	30us	不详
最大中断潜伏期	系统时钟最小分辨率	多进程	多线程	多处理器	MMU 支持	集成 JAVA 支持	自动代码生成工具	RMA
支持	支持	支持	支持	支持	否	不详	否	否