中文摘要

嵌入式系统在如今有日益广泛的应用，用户在开发嵌入式应用系统时，它的实时性能是用户很关注的问题，是软件质量的一个很重要的方面，软件系统中，任务的划分、模块的划分、任务优先级的设置、算法设计以及各模块计算时间的长短、任务间的通信机制等等都将影响系统最终的实时性能。所以为了确保软件质量，实时软件的开发应从需求分析开始，实时性就应受到关注。“嵌入式应用系统设计仿真环境”就是旨在实时软件设计的早期阶段帮助应用开发人员分析其早期软件模型的执行流程和实时性能，发现设计中不合理的地方，以便及早做修改。

CORBA 为可移植的、面向对象的分布式计算应用程序提供了不依赖于平台的编程接口和模型，许多用户会使用 CORBA 开发嵌入式应用系统，DeltaCORBA 是科银京成公司开发的基于 miniCORBA 规范，支持 DeltaOS，Tornado 等嵌入式操作系统的嵌入式 CORBA，本仿真环境同时提供了基于 DeltaCORBA 开发嵌入式应用系统的解决方案。

本论文是作者在科银京成公司参与开发“基于 DeltaCorba 的嵌入式应用系统设计仿真环境”的基础上成文。本文首先介绍了科银京成公司开发的嵌入式操作系统 DeltaOS 和嵌入式 CORBA-DeltaCORBA，然后阐述了“基于 DeltaCorba 的嵌入式应用系统设计仿真环境”的设计方案及其实现方法，最后说明了今后的改进构想，并进行总结。

关键词：嵌入式，DeltaOS,DeltaCORBA,仿真环境

ABSTRACT

Nowaday, the embeded system is applied more and more extensively.When the user designs the embeded application system,the real time capability will be cared by the user.It is one of the important aspects about quality of software.In the software system, the division of tasks and modules, the setting of task priority,the arithmetic design,the calculating time of different modules and communications mechanism among the tasks will affect the real time capability of the embeded application system.So to ensure the quality of software,the real time capability must been cared from the beginning of requirment analyse . "Emulating environment of design of embeded application system" will help the application developer to analyse execution flow and real time capability, finding unreasonable places which can be modified in time.

CORBA offers the interfaces and modules independent of platform for object oritented ,distributing application program.Many users use CORBA to develop the embeded application system.DeltaCORBA developed by Coretek company is based on miniCORBA standard .It is embeded CORBA whick supports some embeded operating system ,such as DeltaOS, Tornado. At the same time,this emulating environment offers the resolving scheme about developing embeded application system based on DeltaCORBA.

This paper is based on a implementation of an emulating environment of design of embeded application system.,which was realized by the Coretek company that the author participated in.In this paper , DeltaOS and DeltaCORBA developed by Coretek company are introduced at first,then the design and implementation solution of this emulating environment are introduced.At last ,the author sets forth the ideas for the future improvement and sum up this paper.

Key words: embedded, DeltaOS,DeltaCORBA, emulating environment

独 创 性 声 明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

签名：刘韬　　日期：2004年2月27日

关于论文使用授权的说明

本学位论文作者完全了解电子科技大学有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权电子科技大学可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后应遵守此规定）

签名：刘韬　　导师签名：王忠仁

日期：　　2004 年　3 月

第一章 概述

1.1 项目背景及其意义

随着计算机的发展和应用的普及，实时计算机系统(以下简称实时系统)已经在工业、交通、能源、科学研究和科学试验、国防等各个领域发挥极其重要的作用。渗透到越来越多的领域，包括过程控制、核电站、智能车辆公路系统、航空、飞行控制、通信、多媒体、办公自动化、自动化控制、计算机外设、消费电子、实时模拟、虚拟现实、医疗应用、军事等等，实时操作系统得到了广泛地应用和发展。大多数实时系统都是嵌入式应用(Embedded Applications)。在嵌入式应用中，嵌入式计算机是内装于专用设备/系统的一种智能部件，其主要功能是在一个大型的工程系统中作为信息处理部件，它是一种看不见的计算机。嵌入式的典型应用包括：生产过程中各种动作流程的控制、通讯设备、智能仪器、军事电子设备和现代武器等等，并且伴随着网络、通讯技术的发展，国内外的 IT 界提出了信息智能家电的概念。从对机顶盒、能上网的微波炉、多功能 DVD，到微软的维纳斯计划、女娲计划等的报道，我们可以看到实时、嵌入式这一领域的发展前途十分巨大。

与一般的计算机应用相比，嵌入式实时应用系统具有及时处理、配置专一、资源有限、结构紧凑和坚固可靠等特点，相应的软件系统应是一种别有特色、要求更高的实时软件。这种实时软件的基本特征是：实时性、有处理异步并发事件的能力、快速启动、并有出错处理和自动复位功能等，从嵌入式实时系统的应用来看，它的实时性是非常重要的特性，是用户很关注的问题，是软件质量的一个很重要的方面，软件系统中任务的划分、模块的划分、任务优先级的设置、算法设计以及各模块计算时间的长短、任务间的通信机制等等都将影响系统最终的实时性能。所以为了确保软件质量，实时软件的开发应从需求分析阶段开始，实时性应受到关注，“嵌入式应用系统设计仿真环境”就是旨在实时软件设计的早期阶段帮助应用开发人员设计嵌入式应用系统的早期软件模型，并分析早期软件模型的执行流程和实时性能，发现设计中不合理的地方，以便及早做修改。

随着嵌入式系统的日益广泛应用，嵌入式系统之间及嵌入式系统与普通桌面系统之间必然会出现更广泛的协同工作的需求。而在当前的桌面和企业应用系统中，这样的协同工作问题已经得到很好的解决，那就是使用一种叫软总线（software bus）的技术（通常也称中间件）。通过软总线，应用系统的对象能达到透明合作的效果。这样的应用系统，具有易于开发，易于维护，易于升级的特点。而 CORBA 作为该技术的主流，已经在桌面和企业应用系统中，得到了广泛的应用。DeltaCORBA 是科银京成公司开发的基于 miniCORBA 规范，支持 DeltaOS，Tornado 等嵌入式操作系统的嵌入式 CORBA，许多用户会使用 CORBA 开发嵌入式应用系统，所以在设计嵌入式应用系统设计仿真环境时，就应该考虑到用户使用 DeltaCORBA 开发嵌入式应用系统。

1.2 进入项目的准备

本项目是在科银京成公司开发的嵌入式实时操作系统 DeltaOS 和嵌入式中间件 DeltaCORBA 的概念基础上开发嵌入式应用系统设计仿真环境，所以必须对 DeltaOS 和 DeltaCORBA 的概念具有相当的了解，才能对它们进行仿真。本人从研一下学期开始就在科银京成公司实习，对实时、嵌入式的基本概念和内核的工作机制已有清楚和深刻的理解，已圆满完成多个和嵌入式相关的项目，在这些工作经验的基础上，在科银京成公司的组织下，展开了该项目的研究和设计，并取得预期结果。

1.3 论文的内容安排

第二章和第三章分别介绍 DeltaOS 和 DeltaCORBA 的基本概念，第四章是描述仿真环境的整体设计，介绍仿真环境的设计思路，第五章介绍仿真环境是如何实现的，第六章介绍了一个嵌入式应用系统，如何使用仿真环境对它们进行仿真设计，并在第七章对全文进行了总结。

第二章 嵌入式实时操作系统 DeltaOS

“嵌入式应用系统设计仿真环境”就是旨在实时软件设计的早期阶段帮助应用开发人员设计嵌入式应用系统的早期软件模型,而该嵌入式应用系统所基于的嵌入式实时操作系统就是科银京成公司开发的 DeltaOS.

2.1 嵌入式实时应用系统软件的基本特征

与一般的计算机应用相比，嵌入式实时应用系统具有及时处理、配置专一、资源有限、结构紧凑和坚固可靠等特点，相应的软件系统应是一种别有特色、要求更高的实时软件。这种实时软件的基本特征是：

1. 实时性

实时软件要求对外部事件作出及时反应，在某些情况下，响应时间还需要是确定的，不管当时系统内部状态如何，都是可预测的(predictable)。

2. 有处理异步并发事件的能力

实际环境中，嵌入式实时系统处理的外部事件往往不是单一的，这些事件往往同时出现，而且发生的时刻也是随机的，即异步的。实时软件应有能力对这些外部事件有效地进行处理。

3. 快速启动、并有出错处理和自动复位功能

这一要求对机动性强、环境复杂的智能系统显得特别重要。通常，嵌入式实时软件需要事先固化到只读存贮器中，开机即用，并在运行出错或死机时能自动恢复先前运行状态。因此嵌入式实时软件应采用特殊的容错、出错处理措施。

4. 嵌入式实时软件是应用程序和操作系统两种软件的一体化程序。

对于通用计算机系统，例如 PC 机、工作站，操作系统等系统软件和应用软件之间界限分明。换句话说，在统一配置的操作系统环境下，应用程序是独立的运行软件，可以分别装入执行。但是，在嵌入式实时系统中，这一界限并不

明显。这是因为，应用系统配置差别较大，所需操作系统繁简不一，这就要求采用不同配置的操作系统和应用程序，链接装配成统一的运行软件系统。也就是说，在系统总设计目标指导下将它们综合加以考虑、设计与实现。

5. 嵌入式实时软件的开发需要独立的开发平台

嵌入式实时应用系统的软件开发常常需要在专门的交叉开发环境中进行，如图 2-1 所示：

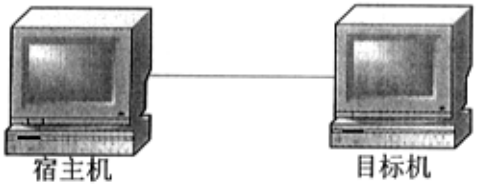


图 2-1 交叉开发环境

2.2 嵌入式实时操作系统

实时操作系统是事件驱动的(event_driven)，能对来自外界的作用和信号在限定的时间范围内作出响应。它强调的是实时性、可靠性，为应用软件提供良好的软件运行环境。

IEEE 的实时 UNIX 分委会认为实时操作系统应具备以下几点：

1. 对异步事件的响应

为了能在限定时间内响应异步事件，实时操作系统需要具有中断处理能力。

2. 任务切换时间和中断延迟时间确定

3. 优先级任务

必须允许用户定义任务的优先级。

4. 抢占式调度

为确保响应时间，必须允许高优先级任务一旦准备好就可马上抢占低优先级任务的运行。

5. 内存锁定

必须具有将程序或部分程序锁定在内存的能力，锁定在内存的程序减少了

为获取该程序而访问盘的时间，从而确保了及时、确定的响应时间。

6. 连续文件

应提供存取盘上数据的优化方法，使得存取数据时查找时间最少。通常要求把数据存储在连续文件上。

7. 同步与互斥

提供同步与协调共享数据使用的手段。

从实时系统的应用特点来看，实时操作系统可以分为两种：一般实时操作系统和嵌入式实时操作系统。一般实时操作系统与嵌入式实时操作系统都是具有实时性的操作系统，它们的主要区别在于应用场合和开发过程。一般实时操作系统应用于实时处理系统的上位机和实时查询系统等实时性较弱的实时系统，并且提供了开发、调试、运行一致的环境。而嵌入式实时操作系统应用于实时性要求高的实时控制系统，而且应用程序的开发过程是通过交叉开发来完成的，即开发环境与运行环境是不一致的。嵌入式实时操作系统具有规模小(一般在几十 KB 内)、可固化使用、实时性强的特点。

据嵌入式系统杂志(Embedded Systems Programming)报告，世界各国有四十多家公司，已成功推出 200 余种可供嵌入式应用的实时操作系统。其中几个著名的实时、嵌入式操作系统是 Wind River System 公司(WRS)的 VxWorks、pSOS，Mentor Graphics 公司的 VRTX，Microsoft 公司支持 Win32 API 编程接口的 Windows CE，Microware 公司的 OS-9，3COM 公司的 Palm OS。

实时、嵌入式操作系统及其应用开发环境的发展动向是：

1. 提供开放的操作系统应用程序接口（API）

商用实时、嵌入式操作系统为了支持开发商根据需要自行开发所需的应用程序，除了提供自身的一套 API 以外，还要提供支持 POSIX 标准、ITRON 标准的 API。

2. 面向 Internet、面向特定应用是实时、嵌入式操作系统的重要发展趋势。

伴随着通用型实时、嵌入式操作系统的发展，一个面向 Internet 网络、面向特定应用的实时、嵌入式操作系统正日益引起人们的重视。嵌入式系统与 Internet 的结合、嵌入式操作系统与应用设备的无缝结合代表着嵌入式操作系统发展的真正未来。

3. 实时、嵌入式 Linux 成为实时、嵌入式操作系统领域的新热点。

实时、嵌入式 Linux 操作系统的迅速崛起，主要由于人们对自由软件的渴望和嵌入式应用的特制性对系统源代码的需求。实时、嵌入式 Linux 正适应了这一需求。它具有开放的源代码、精巧，高效的内核、完整的网络功能、良好的可剪裁性，非常适合信息家电一类的嵌入式系统的开发。

4. 开发环境向开放的、集成化的方向发展

由于嵌入式应用软件的特殊性，往往要求应用程序的设计者具有一定实时操作系统的专门知识，能合理划分任务，合理配置系统以及目标联机的调试。因此，要设计实现一个高性能的实时应用软件，需要强有力的交叉开发工具的支持。国外十分重视发展与实时操作系统配合的嵌入式应用的集成开发环境，现已发展到第三代，它以客户－服务器的系统结构为基础，具有运行系统的无关性、连接的无关性、开放的软件接口(与实时、嵌入式操作系统的接口、与开发工具的接口、与目标环境的接口)、环境的一致性、宿主机上目标仿真等特点。

2.2 DeltaOS 结构

DeltaOS 是科银京成公司开发的强实时、嵌入式多任务操作系统。

2.2.1 DeltaOS 体系结构的设计

嵌入式实时操作系统DeltaOS具有一个可剪裁、可扩展、可配置和可移植的多层次体系结构，如图2-2所示。

DeltaOS是应用程序与硬件之间的接口。至底向上，DeltaOS分为三个层次，依次是设备驱动程序层、内核层和系统程序层。

1. 设备驱动程序层

这一层提供硬件的驱动程序和调用设备驱动程序的接口。设备驱动程序接口（硬件抽象层）提供对抽象硬件操作的接口。抽象硬件是指硬件类型确定，但不指明设备的具体类型。例如定义了网卡驱动程序的接口，但是不区分网卡的类型。不同类型的网卡有不同的驱动程序，但是操作的接口是一致的。

1）内核相关的设备

中断控制器、时钟部件

2）OS其他组件相关的设备

字符设备、网络设备、块设备、显示设备、键盘设备和定点设备。

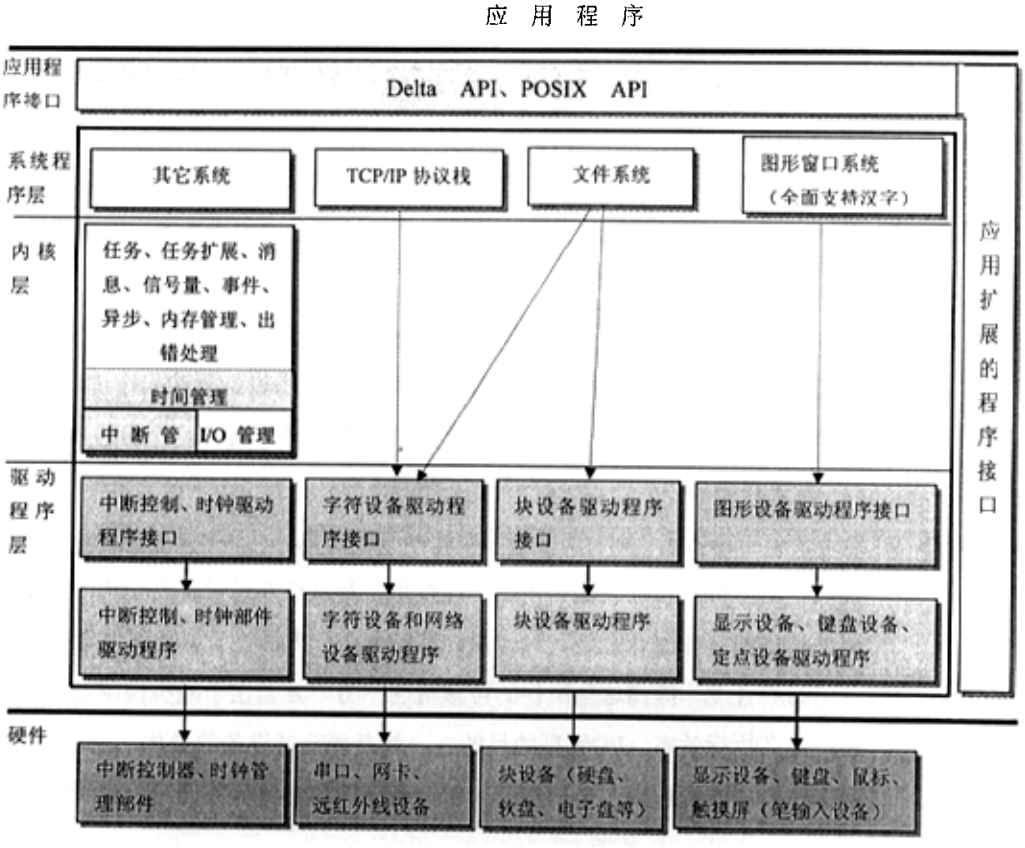


图 2-2 DeltaOS和DeltaCORE体系结构图

2. 内核层

这一层是DeltaOS的内核DeltaCORE的内容。根据与硬件的关系，DeltaCORE的体系结构可以分为与硬件完全无关部分、与抽象硬件相关部分和与具体硬件相关部分。

1）硬件完全无关部分

这部分内容完全独立于硬件，在内核移植时不需改动，具有完全可移植性。它包括任务管理（任务上下文切换除外）、扩展处理、同步，通

信管理（信号量、消息、异步信号和事件）和内存管理。各管理模块之间无调用关系，无层次之分。

2）抽象硬件相关部分

抽象硬件的概念参见本节有关设备驱动程序层中的说明。DeltaCORE对抽象硬件的操作必须通过I/O管理实现，其目的是通过I/O管理提供的用于操作设备的标准接口来规范内核对设备的操作。DeltaCORE中与抽象硬件相关的部分为时间管理。例如，对时钟的初始化是通过I/O管理提供的初始化调用delta_io_initialize调用时钟驱动程序接口，再由它调用具体时钟的驱动程序。所以，内核移植时这部分的代码基本上不用修改。

3）硬件相关部分

DeltaCORE中硬件相关部分为任务上下文切换和中断处理。出于对响应时间的考虑，中断处理不通过I/O管理，直接由设备驱动程序接口调用具体设备的驱动程序。这部分代码带有浓厚的硬件特色，不可移植。

3. 系统程序层

这一层包括了实时操作系统DeltaOS的其它组件：TCP/IP协议栈、文件系统、图形窗口系统及其它系统。这些组件都直接通过设备驱动程序接口调用具体的设备驱动程序。这样做的目的是：一方面在于减小其它组件与内核的耦合度，提高这些组件的可移植性；另一方面在于减少调用层次关系，提高执行效率。I/O管理的目的之一就是规范对设备的操作，而设备驱动程序接口也能起到一定的规范作用。所以，其它组件不使用I/O管理也是可行的。

DeltaOS 向应用提供两类编程接口：本地化的接口 Delta API 和标准化的 POSIX API。用户还可以根据需要建立自己的编程接口，这里称为应用扩展的程序接口。例如，汽车行业的用户、数字电视领域的用户就希望专门针对本行业建立一套编程接口。对于上层的所有应用都建立在这层编程接口上，编程人员不需要关心这些接口的实现细节，只需要遵循系统约定的规则建立好自己的应用。基于 DeltOS 的应用编程人员看到的系统体系结构如图 2-3 所示。

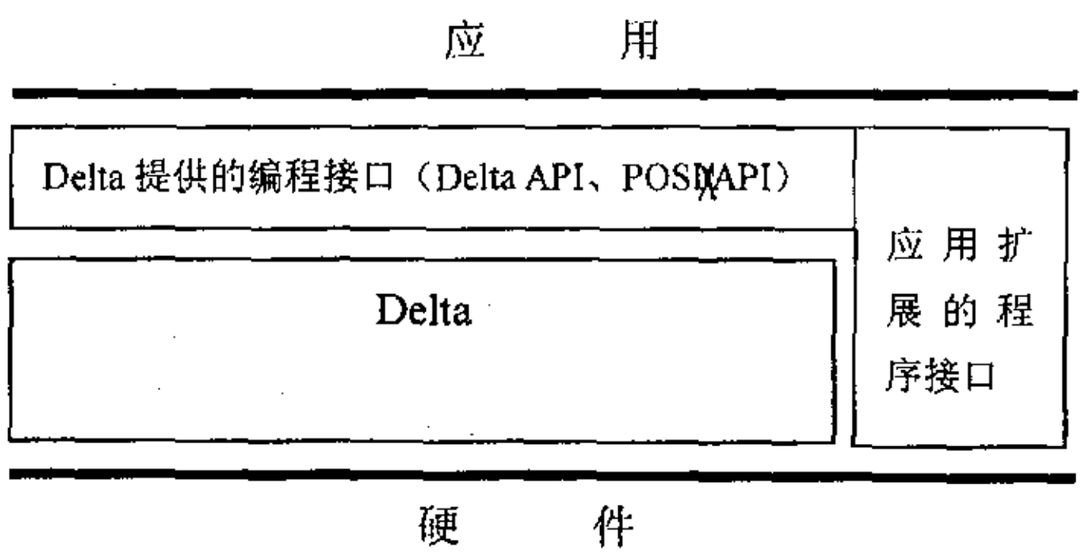


图 2-3Delta 的应用编程人员看到的系统体系结构

2.2.2 DeltaCORE2.0 结构原理

DeltaCORE2.0 是一个强实时、嵌入式多任务操作系统 DeltaOS 的内核。其内存模式为平模式（flat memory mode），适用于内存要求较小、可靠性要求较高的嵌入式系统。

2.2.2.1 DeltaCORE2.0 概述

1). 初始化管理

完成实时内核的初始化工作，并启动实时调度。

2). 任务管理

完成应用任务的管理，是操作系统内核的核心部分。它具有创建任务、删除任务、挂起任务、解挂任务、设置任务优先级等功能。

3). 时间管理

为应用系统的实时响应提供支持，保证整个系统的实时性、正确性，以提高整个嵌入式系统的实时工作能力。该模块允许应用任务设置和读取系统时间；允许应用任务睡眠一段时间或睡眠到某一时刻；完成系统计时和对任务时间片的计算。

4). 定时管理

完成系统的定时功能。在定时时间后调用相应的定时处理程序。

5). 中断管理

完成响应中断的一些必要处理，支持中断嵌套时任务堆栈和中断专用堆栈的切换。

6). 通信、同步、互斥管理

提供应用任务之间、任务与中断之间的通信、同步和互斥机制，对它们之间的协调工作起着重要的作用。内核提供了四种机制：

a. 消息队列----完成任务间的数据传输；

b. 信号量----实现资源的共享、互斥及同步；

c. 异步信号----支持任务间的异步通信；

d. 事件----高效的通信与同步机制。

7).内存管理

提供可变大小数据块和固定大小数据块的管理。

DeltaCORE2.0 可用于开发不同类型的嵌入式应用，具有如下特点：

- 目标环境独立性

DeltaCORE2.0需要的存储容量小，提供了真正的芯片级的支持。

- 可扩充性

用户可以容易地将应用程序和DeltaCORE2.0结合起来，既可以由应用程序独立运行自己的调用程序和例程，也可以由DeltaCORE2.0统一管理。操作系统的其它组件也可容易地加到系统中。

- 位置无关性

DeltaCORE 2.0具有一个可重定位的目标库，在链接时可以被定位到内存允许的任意地址空间。

2.2.2.2 DeltaCORE2.0 的体系结构

图 2-4 是 DeltaCORE2.0 基本体系结构，可以看出，DeltaCORE2.0 对硬件的基本要求即 PROM、RAM、CPU、CLOCK、字符 I/O 设备；它与多任务应用程序的接口为系统调用。

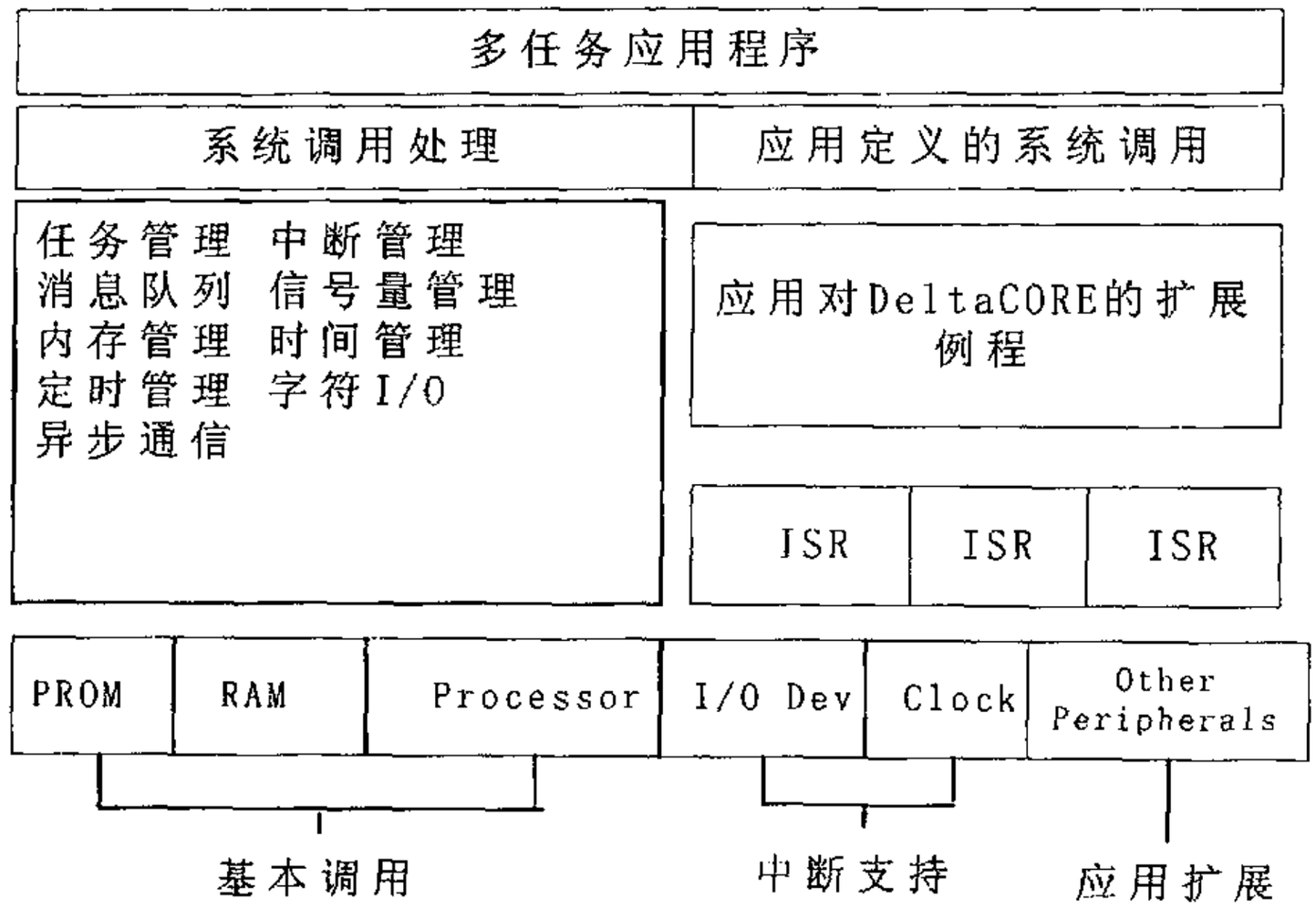


图 2-4 DeltaCORE2.0 基本体系结构

2.2.2.3 DeltaCORE2.0 的系统调用

DeltaCORE2.0 与多任务应用程序的接口为系统调用。应用程序通过系统调用使用 DeltaCORE2.0 提供的功能。DeltaCORE2.0 提供了 11 类共 70 多个系统调用，并以库（libdeltakernel.a）的形式存在，用户可在此基础上编制应用程序。只有应用中用到的功能才会与应用程序链接在一起，装入目标环境。在本防真环境中，将会模拟其中的一部分系统调用的功能，这在后面会有详细说明。

DeltaCORE2.0 提供了 11 类系统调用，如表 2-1 所示。

类别	系统调用名称	功能
任务管理	delta_task_create	创建任务
	delta_task_delete	删除任务
	delta_task_start	使任务就绪
	delta_task_restart	使任务以初始状态执行
	delta_task_suspend	挂起任务
	delta_task_resume	解挂任务
	delta_task_wake_after	任务睡眠一段时间
	delta_task_wake_when	任务睡眠到某一时刻
	delta_task_ident	获得任务 ID 号
	delta_task_set_priority	设置任务优先级
	delta_task_mode	获得任务模式

	delta_task_set_note	设置参数指定的笔记本
	delta_task_get_note	获得参数指定的笔记本
	delta_task_variable_add	添加任务变量
	delta_task_variable_get	获得任务的变量
	delta_task_variable_delete	删除任务的变量
	delta_task_disable_dispatch	关调度
	delta_task_enable_dispatch	开调度
	delta_task_is_suspended	判断任务是否被挂起
用户扩展管理	delta_extension_create	创建一个用户扩展集
	delta_extension_ident	获得用户扩展集 ID
	delta_extension_delete	删除指定的用户扩展集
region	delta_region_create	创建一个 Region
	delta_region_ident	获得一个 Region 的 ID
	delta_region_delete	删除一个 Region
	delta_region_extend	扩展 Region
	delta_region_get_segment	申请 segment
	delta_region_return_segment	返回 segment 到 Region
	delta_region_get_segment_size	获得 segment 大小
partition	delta_partition_create	创建一个 Partition
	delta_partition_ident	获得一个 Partition 的 ID
	delta_partition_delete	删除一个 Partition
	delta_partition_get_buffer	申请 buffer
	delta_partition_return_buffer	将 buffer 返回给 Partition
消息队列	delta_message_queue_create	创建消息队列
	delta_message_queue_ident	获得消息队列 id
	delta_message_queue_delete	删除消息队列
	delta_message_queue_send	发送普通消息
	delta_message_queue_urgent	发送紧急消息
	delta_message_queue_broadcast	广播消息
	delta_message_queue_receive	从队列中接收消息
	delta_message_queue_get_number_pending	获得队列中的消息数
	delta_message_queue_flush	删除队列中所有消息
信号量	delta_semaphore_create	创建互斥信号量
	delta_semaphore_ident	获得信号量 ID
	delta_semaphore_delete	删除信号量
	delta_semaphore_obtain	获得一个信号量
	delta_semaphoe_release	释放一个信号量
	delta_semaphore_flush	唤醒所有等待信号量的任务
事件管理	delta_event_send	发送事件
	delta_event_receive	接受事件
异步通信	delta_signal_catch	安装异步信号处理例程
	delta_signal_send	发送异步信号
中断管理	delta_interrupt_disable	禁止所有可屏蔽中断
	delta_interrupt_enable	开中断，恢复先前的中断

		状态
	delta_interrupt_flash	该系统调用暂时地恢复最近一个 delta_interrupt_disable 调用之前的中断级，然后立即屏蔽掉所有可屏蔽中断
	delta_interrupt_is_in_progress	该系统调用判断当前是在 ISR 中还是在任务中
	delta_interrupt_get_handler	该系统调用获得指定中断向量的 ISR 起始地址
	delta_interrupt_install_vector	该系统调用建立指定中断向量的 ISR
	delta_interrupt_remove_handler	该系统调用恢复指定中断向量的 ISR 为内核提供的缺省的 ISR 起始地址
时钟管理	delta_clock_get	获取指定类型的时间
	delta_clock_set	设置系统日历时间
	delta_clock_tick	维护系统时基
	delta_clock_tick_second	维护秒级的计时
定时管理	delta_timer_create	创建定时器
	delta_timer_delete	删除定时器
	delta_timer_cancel	中止定时器计时
	delta_timer_ident	获得定时器 ID
	delta_timer_fire_after	相对时间触发定时器
	delta_timer_fire_when	绝对时间触发定时器
	delta_timer_reset	重新设置定时器的值

表 2-1 DeltaCORE2.0 的系统调用

第三章 嵌入式中间件 DeltaCORBA

DelatCORBA 是科银京成公司基于 minimum CORBA 规范设计开发的一种嵌入式 CORBA，能够支持 DeltaCORE，Tornado 等嵌入式操作系统。同时提供支持 Windows 98，NT/2000 和 Linux 的版本，以支持嵌入式系统与桌面系统的协同工作。

3.1 CORBA 简介

计算机网络是典型的异构（Heterogeneous）体系。例如，一个小软件公司的内部网络可能是由多个计算机平台组成的。可能有一个主机来处理用作订货的事务数据库访问，UNIX 工作站用来提供硬件仿真环境和软件开发中枢，PC 机使用 Windows 操作系统并提供桌面办公自动化工具，还有其它一些专用的系统，如网络计算机，电话系统、路由器和测量设备。一个给定网络的一小部分可能是同构机型的，网络越大，它所组成的机型就越多样化。

造成这种异构的原因很多。一个明显的原因是网络技术随着时间不断地在改进。网络技术不是一成不变的，它在不断地演化，不同时期最好的技术可能在同一网络中共存。造成网络异构的另一个原因是网络的大小不是固定不变的。任何一种计算机、操作系统、网络平台的组合都是为了能在一个网络内使某一部分的性能达到最好。还有一个原因是在一个网络内的多样化使得它具有更大的回转余地，因为在某一机器类型，操作系统或应用程序所出现的问题可能在其它操作系统和应用程序上不成其问题。

造成异构的计算机网络的因素是不可避免的，因此实际从事分布式系统的开发者无论喜欢还是不喜欢都必须面对异构的问题。开发用于分布式系统的软件已是十分困难，而开发一个异构的分布式系统的软件就更困难了。这类软件必须涉及到分布式系统编程过程中通常所要遇到的所有问题，比如：网络中某些系统的故障，网络分区，及与资源争用和共享、网络安全等相关的问题。

例如，当你将一些网络上的应用程序移植到新的网络平台时，就会遇到同一个应用程序，它有好几个版本。如果你对这个应用程序任一个版本做了修改，那么你必须回过头来对其他版本也做适当的修正，然后单个地测试它们，并且在他们各种各样的组合中，确保它们都能正常工作。在同一个网络中，不同平

台的数目越多，这种情况就越复杂，难度也就越大。

从这个意义上来讲，异构不是仅仅指计算机硬件和操作系统。当从上层向下层编写一个鲁棒的分布式应用程序时，例如，从一个定制的图形用户接口一直到网络传送和网络协议，这几乎对所有的实际应用程序都是非常棘手的，因为所涉及到的细节太多、太复杂。因此，分布式应用的开发者倾向于大量使用工具和库。这就意味着，分布式应用程序本身就是异构的，通常需要将大量不同层次的应用程序和库连接在一起。遗憾的是，在许多情况下，当分布式系统增大时，能够将所有应用程序和库组合在一起的可能性就越小。

在很多情况下，你可以按照下面两条主要原则来解决异构的分布式系统的应用程序的开发问题。

1. 寻求独立于平台的模型和抽象，这样有助于解决大部分问题；
2. 在不牺牲它多性能的情况下，尽可能隐藏低层的复杂细节。

通常，这两条原则对于开发任何一个可移植的应用程序都是适用的，不论是分布式系统还是非分布式系统。但是，由于分布式体系引入的附加的复杂性也使上述原则肩负更重要的任务。虽然使用合适的抽象和模型将会提供新的异构的应用程序开发层，分布式异构的复杂性将会集中在这个层面上。在这一层上，底层的细节被隐藏起来了，并且允许应用程序的开发者只解决他们自己的开发问题，而不必面对应用程序所要涉及到的，由于不同计算机平台所带来的低层的网络细节问题。

由 OMG 编写和维护的 CORBA 规范提供了一种灵活的、切实可行的抽象集，并且确定了一些服务程序。这些服务程序对于解决由于分布式异构计算所带来的一系列问题都是必需的。

3.1.1 对象管理组

对象管理组（Object Management Group, OMG）是一个非赢利性的协会组织，组建于 1989 年，由八个著名的计算机公司发起。现在它的成员已经超过 800 个，并且分布很广泛，遍及计算机制造商、软件公司、通信公司和最终用户。OMG 的目标是，推动对象技术（OT）的理论和实践在软件行业中的使用，特别是在开发分布式计算机系统方面。为了达到这一目标，OMG 所采用的方法

是，为面向对象的应用提供一个公共框架，如果符合这一框架，就可以在多种硬件平台和操作系统上建立一个异质的分布式应用环境。

为使该组织所采纳的技术具有开放性，OMG 所采用的方法是，针对某一领域发出 RFP（Request For Proposal），然后以各方提交的建议为基础，经过一系列的讨论和协商，产生最终的规范。CORBA 就是这样诞生的。CORBA 的最初版本主要基于以下几个公司所提交的建议：DEC、HyperDesk、HP、SunSoft、NCR 和 Object Design。CORBA 的主要目标是，提供一种通用总线机制，在此基础上，对象可以透明地发出请求和获得应答。

由 OMG 制定的最关键的规范——对象管理结构（Object Management Architecture, OMA）和它的核心（也就是 CORBA 规范），提供了一个完整的体系结构。这个结构以足够的灵活性、丰富的形式适用了各类分布式系统。

OMA 包括两部分：对象模型（Objetct Model）和参考模型（Reference Model）。对象模型定义如何描述分布式异质环境中的对象；参考模型描述对象之间的交互。对 OMG RFP 提交的建议必须符合这两个标准，否则将不予采用。

在 OMA 对象模型中，对象是一个被封装的实体，它具有一个不可改变的标识，并能给客户用户提供一个或多个服务。客户只能通过已经定义好的接口才能访问对象。客户向对象发出请求,通过上述接口以获得某些服务。请求信息包括目标对象、所请求的操作、0 个或多个实际参数和可选的请求上下文。请求上下文定义有关交互的额外信息，它以（Name ， Value）对的方式描述有关客户和环境的信息。每个对象的实现和位置，对客户都是透明的。

参考模型提供了接口种类。图 3-1 描述了 OMA 参考模型的各个组成部分，其中 ORB 的主要任务是负责客户和服务器之间的交互。可以通过四种类型的接口使用 ORB 部分，它们分别是：对象服务（Object Service）、公共设施（Common Facility）、域接口（Domain Interface）和应用接口（Application Interface）。ORB(Object Request Broker)是对象总线。它能使对象透明地向其它本地或远程对象发出请求或接收响应。而客户方并不需要了解服务对象的通信、激活或存储机制。

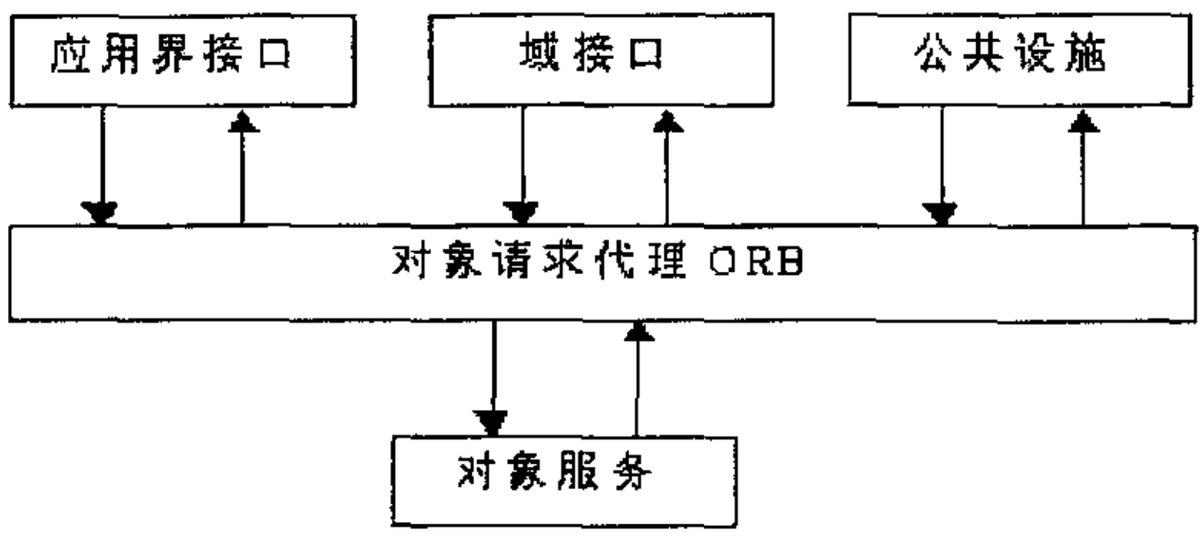


图 3-1 OMA 的参考模型

3.1.2 概念和术语

CORBA 为可移植性的、面向对象的分布式计算应用程序提供了不依赖于平台的编程接口和模型，它不依赖于编程语言、计算平台、网络协议的这一特点，使得它非常适合于现有的分布式系统新的应用程序的开发和系统集成。

与其它技术一样，CORBA 只是唯一的术语，虽然有些概念和数据从类似的技术借用而来，但是其它一些都是新的、不同的技术。透彻的理解这些术语和概念的深刻含义是掌握 CORBA 的关键。我们将在下面解释 CORBA 中最重要的术语：

CORBA 对象（CORBA Object）：它是一个“虚拟”的实体，它可以由 ORB 定位，并且可以被客户程序请求调用（invoke）。虚拟是指实际上它并不存在，除非已用某编程语言编写了它的具体的实现部分。由编程语言构造的 CORBA 对象实现部分类似于操作系统中并不存在的虚拟内存，但是可用物理内存来模拟的方式。

目标对象（target Object）：在一个 CORBA 请求调用上下文中，目标对象是指这个请求目标的 CORBA 对象。CORBA 对象模型是一个单调度模型（single-dispatching model），在这个模型中，一个请求的目标对象只能由用来调用这个请求的对象引用来确定。

客户程序（client）：它是一个实体，它调用 CORBA 对象的一个请求。它可能有一个地址空间，但完全独立于 CORBA 对象，或者客户程序和 CORBA 对象可能在一个应用程序中存在。术语“客户程序”仅仅是对一个特定请求的上下文而言的，因为这个应用程序（提出一个请求的客户程序）可能是另一个请求的

服务器程序。

服务器程序（server）：它是一个拥有一个或多个 CORBA 对象的应用程序。与客户程序一样，这个术语仅仅是对一个特定请求的上下文而言的。

请求(request)：它是一个由客户程序所提出的 CORBA 对象的调用操作。请求从一个客户机传给服务器中的目标对象，如果这个请求要求一个 CORBA 对象，作为响应，目标对象负责返回结果。

对象引用(object reference)：它是一个用来标识、定位和赋给一个 CORBA 对象地址的一个句柄。对于客户程序来说，对象引用是不透明的实体。客户程序用对象引用直接指向所请求的对象，但是客户程序不能从他们的组成部分中创造对象引用，也不能访问或修改对象引用的内容。一个对象引用仅仅指向单个 CORBA 对象。

伺服程序（servant）：它是一个编程语言的实体，它用来实现一个或多个 CORBA 对象。伺服程序也称为具体化的 CORBA 对象，因为他们为这个对象提供了函数体或者实现。伺服程序存在于服务器应用程序上下文中。在 C++中，伺服程序是一个特定类的一个对象实例。

3.1.3 公共对象请求代理体系结构（CORBA）

CORBA 是 OMG 制定的首批重要规范之一。它详细说明了 OMA 中 ORB 组件的特性和接口。最新的 CORBA 规范主要包含以下内容：

- ORB 核心（ORB CORE）；
- OMG 接口定义语言（OMG Interface Definition Language）；
- 接口仓库和实现仓库（Interface Repository and Implementation Repository）；
- 语言映射（Language Mapping）；
- 存根和框架（Stub and Skeleton）；
- 动态调用和调度（Dynamic Invocation and Dispatch）；
- 对象适配器（Object Adapter）；
- ORB 之间的互操作（Interoperability Between ORB）。

图 3-2 描述了 CORBA 的主要组成部分和它们之间的关系。下面对每个组件进行详细的论述。

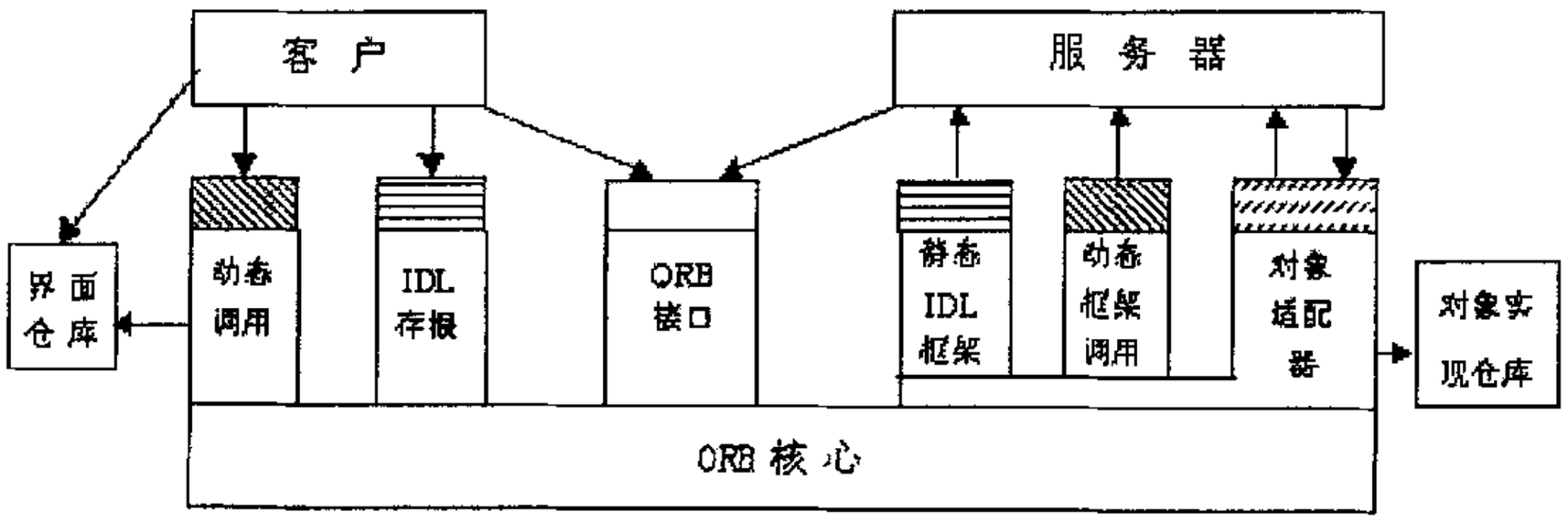


图 3-2　　CORBA 的主要组成部分

如前所述，ORB 的任务是，把客户发出的请求传递给目标对象，并把目标对象的执行结果返回给发出请求的客户。由此可以看出，ORB 的最重要的特征是，提供了客户和目标对象之间的交互透明性。具体地说，它主要屏蔽了以下内容：

- 对象位置：客户不必知道目标对象的物理位置。它可能与客户一起驻留在同一个进程中或同一机器的不同进程中，也有可能驻留在网络上的远程机器中。
- 对象实现：客户不必知道有关对象实现的具体细节。例如，设计对象所用的编程语言，对象所在节点的操作系统和硬件平台等。
- 对象的执行状态：当客户向目标对象发送请求时，它不必知道当时目标对象是否处于活动状态（即是否处于正在运行的进程中）。此时，如果目标对象不是活动的，在把请求传给它以前，ORB 会透明地将它激活。
- 对象通信机制：客户不必知道 ORB 所用的下层通信机制，如，TCP/IP、管道、共享内存、本地方法调用等。
- 数据表示：客户不必知道本地主机和远程主机对数据表示方法是否有所不同。

ORB 的这些特点，使应用开发者不必过多地担心底层的分布式编程问题，从而可以集中精力设计自己的具体应用。

发送请求时，客户用对象引用来说明目标对象。当创建一个 CORBA 对象时，同时也创建了它的对象引用。对象引用是不透明的，且客户不能对它进行修改。只有 ORB 才知道对象引用的“内幕”。

3.1.4 请求调用

在图 3-2 中，客户应用程序提出请求，服务器程序接收这些请求并作出响应。请求流由客户程序向下提交，通过 ORB，上传到服务器应用程序。可由下列几种形式实现这个过程：

1. 客户机有两种选择方式提出请求。第一种，使用由对象接口定义，用 C++编译的静态存根（static stub）；第二种，使用动态调用接口 DII(Dynamic Invocation Interface)。不论那种方式，客户机都直接将请求传送给与这个进程连接的 ORB 核。
2. 客户机 ORB 核通过网络传送给与服务器应用程序相连接的服务器 ORB 核。
3. 服务器 ORB 核将这些请求分配给对象适配器（Object Adapter），由它产生目标对象。
4. 对象适配器进一步将请求分配给实现目标对象的伺服程序。与客户机一样，服务器程序可以选择静态或动态调用机制用于它的伺服程序。这取决于是由对象接口定义，用 C++语言编译的静态框架(static skeleton)，还是其伺服程序可使用动态框架接口（Dynamic Skeleton Interface, DSI）。
5. 伺服程序执行请求后，它返回结构给客户应用程序。

客户机通过发送消息来控制对象。每当客户机调用一个操作时，ORB 发送一个消息给对象。为了发送一个消息给一个对象，客户机必须拥有该对象的对象引用。对象引用起着一个句柄的作用，句柄标识唯一的目标对象，并且封装了要将所用消息发送给正确的目标 ORB 所需的消息。

3.2 DeltaCORBA 简介

3.2.1 Mininum CORBA

随着嵌入式系统的日益广泛应用，嵌入式系统之间及嵌入式系统与普通桌面系统之间必然会出现更广泛的协同工作的需求。特别是，由于各种原因，如系统的功能、性能及系统的成本等，这些系统会更多的采用 COTS（Commercial off-the-shelf）技术。这就会造成应用系统之间，甚至同一个应用系统内，具有很大的差异。使用较低层次的通信协议（如 TCP/IP）来实现协同工作，其代价较大。应用开发者不但要专注具体应用的问题，更是要花费大量的精力去了解下层平台的特性，并解决的所处平台的差异。

而在当前的桌面和企业应用系统中，这样的协同工作问题已经得到很好的解决，那就是使用一种叫软总线（software bus）的技术（通常也称中间件）。通过软总线，应用系统的对象能达到透明合作的效果。这样的应用系统，具有易于开发，易于维护，易于升级的特点。而 CORBA 作为该技术的主流，已经在桌面和企业应用系统中，得到了广泛的应用。

CORBA 的规范出现，为分布式应用提供了很好的互操作性、平台无关性、语言无关性等优点。CORBA 的这些优点，使它对嵌入式系统仍然具有很强的吸引力。将 CORBA 引入嵌入式系统中，会很好地解决嵌入式系统之间及嵌入式系统与普通桌面系统之间的协同工作的问题，而且会降低开发这样的系统的代价，缩短开发周期，更易于维护和升级系统。

由于嵌入式系统与普通桌面系统存在很大的差异，这主要是嵌入式系统的资源（如 CPU 的处理能力、内存的容量、功耗等）有限，因而普通的 CORBA 系统并不适合于嵌入式系统。针对嵌入式系统的特点，OMG 对普通 CORBA 系统进行很大的裁减，在近期推出了适用于嵌入式系统的规范：minimum CORBA。minimum CORBA 规范主要从以下几方面对 CORBA 规范进行了裁减：

（1） 运行安全性

如果应用需要，需要应用自己解决该问题；

（2） 动态分配

所用的资源在设计时进行预分配。CORBA 中支持动态分配的部分被裁减。这就将动态调用界面，动态调用框架，及可移植对象适配器的动态部分被删除。

（3） 截取器

截取器被放置在客户调用服务的通信路径上，负责一个或多个 ORB 服务的执行。由于它具有很大的动态性，也被省略。

（4） 界面仓库

ORB 提供界面定义的永久存储，界面仓库则管理和提供 OMG IDL 所规范的界面定义的访问。界面仓库主要为动态编程提供服务，而在嵌入式中，动态编程被删除。因此，界面仓库只保留了仓库 ID 和 TypeCode 两部分。

（5） 构造策略

嵌入式 CORBA 中更多的是使用缺省策略。

（6） 互操作

CORBA 与 DCE 和 DCOM 的互操作被省略。

这样，嵌入式 CORBA 解决了其尺寸大小和性能问题后，对嵌入式应用提供了许多好处：

（1） 提供很好的界面

CORBA IDL 规范为开发者提供了规范的界面定义，这使得嵌入式应用的开发规范而容易。

（2） CORBA 为分布式计算系统而设计

CORBA 为分布式计算系统而设计，它具有很好的分布处理能力，易于开发数据密集型的应用。CORBA 不仅提供了透明的通信能力，而且有很好的协同工作能力，特别适合那些需要大量处理处于分布的数据而后才能作出决策的嵌入式应用。

（3） CORBA 具有很好的伸缩性

良好的伸缩性使我们能很容易地扩展现存的系统，加入新的单元，而不需要重新设计整个系统。

（4） CORBA 使得应用独立于平台

基于 CORBA 的应用具有很好的可移植性。

DeltaCORBA 正是基于 minimum CORBA 规范，能够支持 DeltaOS，Tornado 等嵌入式操作系统的嵌入式 CORBA。同时提供支持 Windows 98，NT/2000 的版本，以支持嵌入式系统与桌面系统的协同工作。

DeltaCORBA 的应用领域将主要针对电信、工控和军工三个领域。

3.2.2 DeltaCORBA 的基本结构

CORBA 规范是针对对象请求代理系统制定的规范。ORB 直接置于操作系统之上，这使得 ORB 对具体的操作系统具有较强的依赖关系。为了使 ORB 具有很好的可移植性，我们在设计 ORB 时，将这些依赖关系抽象出来，作为平台依赖层（PDL, Platform Dependent Layer）来实现（如图 2），这就去掉了 ORB 与具体操作系统的强耦合关系，使其具有很好的可移植性。针对不同的操作系统，只要对平台依赖层进行相应的修改，就可以很容易地将 ORB 移植到不同的操作系统之上。

通过深入的分析，我们发现 ORB 对具体操作系统的依赖主要集中在线程(任务）管理、内存管理和线程间通信（同步、互斥）的问题上。当前的操作系统在这些方面的支持，差距较大，尤其是嵌入式操作系统，缺乏通用的标准，这些方面的的管理更是各不相同。在多线程 ORB 的系统中，必须保护操作的安全性，这就更需要操作系统的内核调用（线程（任务）管理、内存管理和线程间通信（同步、互斥））的支持。

通过平台依赖层，ORB 的原码不再依赖具体的操作系统，从而具有很好的可移植性。同样的 ORB 原码可以适用于多种不同的操作系统。这就大大缩短了 ORB 在不同操作系统上的开发时间。PDL 简化了线程（任务）管理、内存管理和线程间通信（同步、互斥），它提供了更通用、更强的同步、互斥和线程（任务）的控制原语。平台依赖层向 ORB 提供了独立于操作系统的接口，我们称为 DeltaCORBA 任务接口，相应的库称为 DeltaCORBA 任务库。

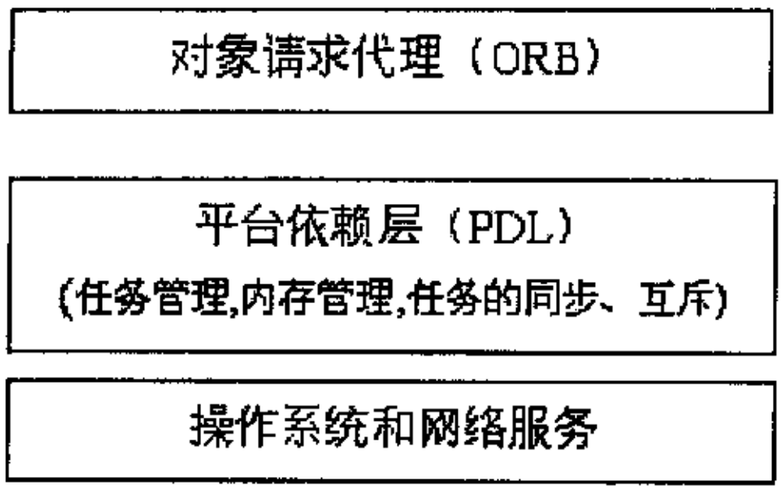


图 3-3 平台依赖层

DeltaCORBA 基于 minimum CORBA 规范设计，其结构与 CORBA 的参考模型相似。图给出了 DeltaCORBA 的结构框架。

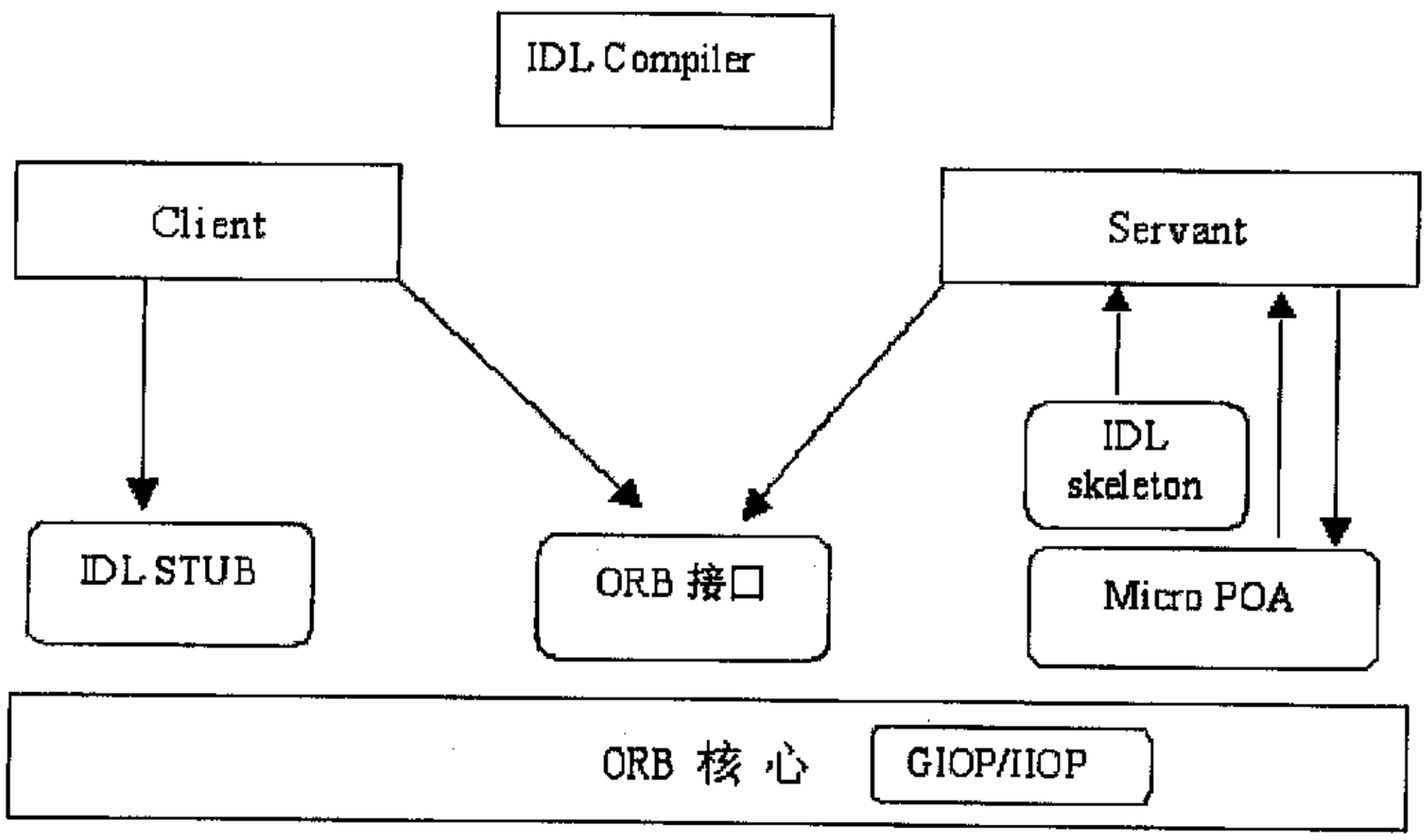


图 3-4 DeltaCorba 的结构

与普通的 CORBA 比较，DeltaCORBA 不需要动态调用，因而去掉了动态调用接口（DII）和动态框架接口（DSI），并且将接口库的大部分去掉，只保留了 RepisitoryIds 和 TypeCode 两部分。同时按照 minimumCORBA 规范，对可移植对象适配器(POA)进行了大量的裁减，向 AdapterActivator、ServantManagers 都被去掉。

DeltaCORBA 将基于 minimum CORBA 规范进行开发，能够支持 DeltaCORE，Tornado 等嵌入式操作系统。同时提供支持 Windows 98，NT/2000 和 Linux 的版本，以支持嵌入式系统与桌面系统的协同工作。

DeltaCORBA 的基本特点是：（1）较小的 ORB 核；（2）IDL 编译器支持；（3）Micro POA；（4）灵活可配置的设施；（5）多协议支持框架；（6）平台独立的线

程（任务）机制；(7)多平台支持，将移植到 DeltaOS，Tornado ，Windows 等操作系统平台上。

第四章 仿真环境的整体设计

本章将对“基于 DeltaCORBA 的嵌入式应用系统设计仿真环境”的设计目的和它的基本设计思想，以及它的结构等进行详细介绍。

4.1 设计目的

嵌入式系统在如今有日益广泛的应用，如在生产过程中各种动作流程的控制、通讯设备、智能仪器、军事电子设备和现代武器等等，在这些嵌入式应用中，往往会有很高的实时方面的要求，所以用户在基于嵌入式实时操作系统之上设计嵌入式应用系统时，往往会高度关注他所设计的系统是否能满足实时性能上的要求，由于系统中任务的划分、模块的划分、任务优先级的设置、算法设计以及各模块计算时间的长短、任务间的通信机制等等都将影响系统最终的实时性能，所以在设计嵌入式应用系统时，很难评估这种设计所形成的系统的实时性能是否最终能满足要求，所以需要一种仿真环境，帮助嵌入式应用系统设计开发人员在嵌入式应用系统软件设计的早期阶段设计早期软件模型，并分析其早期软件模型的执行流程和实时性能，提高设计的效率。

随着嵌入式系统的日益广泛应用，嵌入式系统之间及嵌入式系统与普通桌面系统之间必然会出现更广泛的协同工作的需求，在这种情况下，嵌入式应用系统

的设计人员往往会使用 DeltaCORBA 解决这些网络间或系统间的异构问题，所以该仿真环境还必须帮助用户设计使用 DeltaCORBA 的早期软件模型。

4.2 设计目标

基于以上的设计目的，该仿真环境应具有以下的基本功能：

1)应尽量采用图形化的方式表示和分析嵌入式应用系统的早期软件模型。

2)支持服务器/客户端的模式，服务器端提供 Corba 服务，而客户端是一个嵌入式应用系统。

3) 嵌入式应用系统按任务划分，并支持任务的优先级。

4）支持中断。

5）以图形和伪代码的方式表示任务及中断的程序执行流程。

6）支持信号量、消息、事件等通信机制。

7）在程序流程中支持循环和判断，以及布尔表达式。

8）当用户使用该环境完成软件模型后，对应用实时软件的性能进行分析预测，从不同角度，以不同方式表示出分析结果，可分析各任务的调度顺序，和任务及任务中各操作的执行时间。

4.3 运行环境

本仿真环境采用 VC6.0 开发，可运行于 Windows2000,WindowsNT,WinXP 等操作系统。

4.4 基本设计概念

服务端/客户端(C/S) 整体结构图:以图形化的方式表示服务端和客户端相互关系图，服务端和客户端可能运行在同一台计算机中，也可能在网络上不同的计算机中。

任务结构图：以图形化的方式表示一个嵌入式应用系统中的各任务和中断，以及它们之间的相互通信关系，包括信号量、消息、事件等。

程序结构图：以图形化的方式表示一个任务或中断中的程序执行流程，包括逻辑判断、发送事件、接收事件等等。

Corba 服务例程：服务器端具体实现某个客户服务请求的操作，客户应用程序向服务器端提出某个具体的服务请求，服务器程序接收这些请求，执行服务例程，并作出响应。

后台运行分析器：一个小型的编译器，以任务结构图和程序结构图作为输入，结合实时操作系统调用的时间数据库和 Corba 调用时间数据库产生对应于各个任务和中断的中间代码序列，供下一步进行实时性能分析。

4.5 基本设计思路和处理流程：

4.5.1 仿真环境的总体三部分

如图 4-1 所示，整个仿真环境分为三部分：

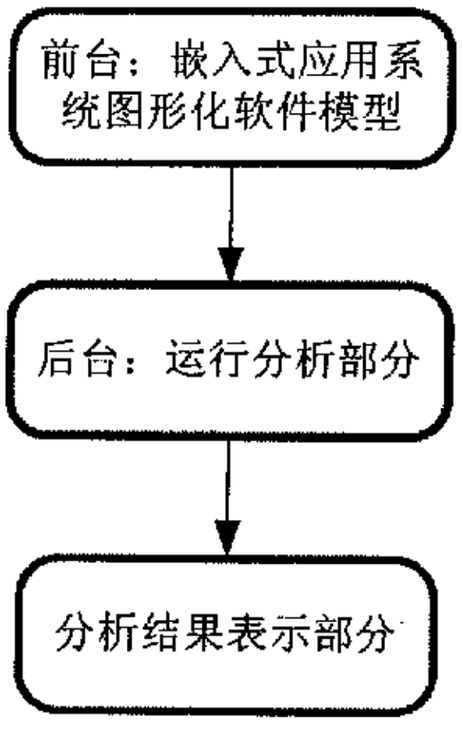


图 4-1 仿真环境的总体三部分

第一部分是前台：仿真环境帮助用户设计嵌入式应用系统的图形化的软件模型，包括客户端和服务器的设置，任务和中断的设置，优先级的安排，任务间的通信关系，任务中程序流程的伪代码序列等等。

第二部分是运行分析部分：以前台中所设计的嵌入式应用系统的图形化的软件模型为输入，结合操作系统调用的时间数据库和 Corba 调用时间数据库，模拟出运行结果，给出实时性能。

第三部分是结合第二部分的分析结果，以文字和图表两种方式把这些数据表示出来。

以下对这三部分逐一论述：

4.5.2 仿真环境前台的三层模式：

在面向用户的前台中，如何以图形化的形式表示出基于 DeltaCORBA 的嵌入式应用系统的软件模型，是一个难点。在本仿真环境中采用三层模式解决这一问题，如图 4-2 所示：

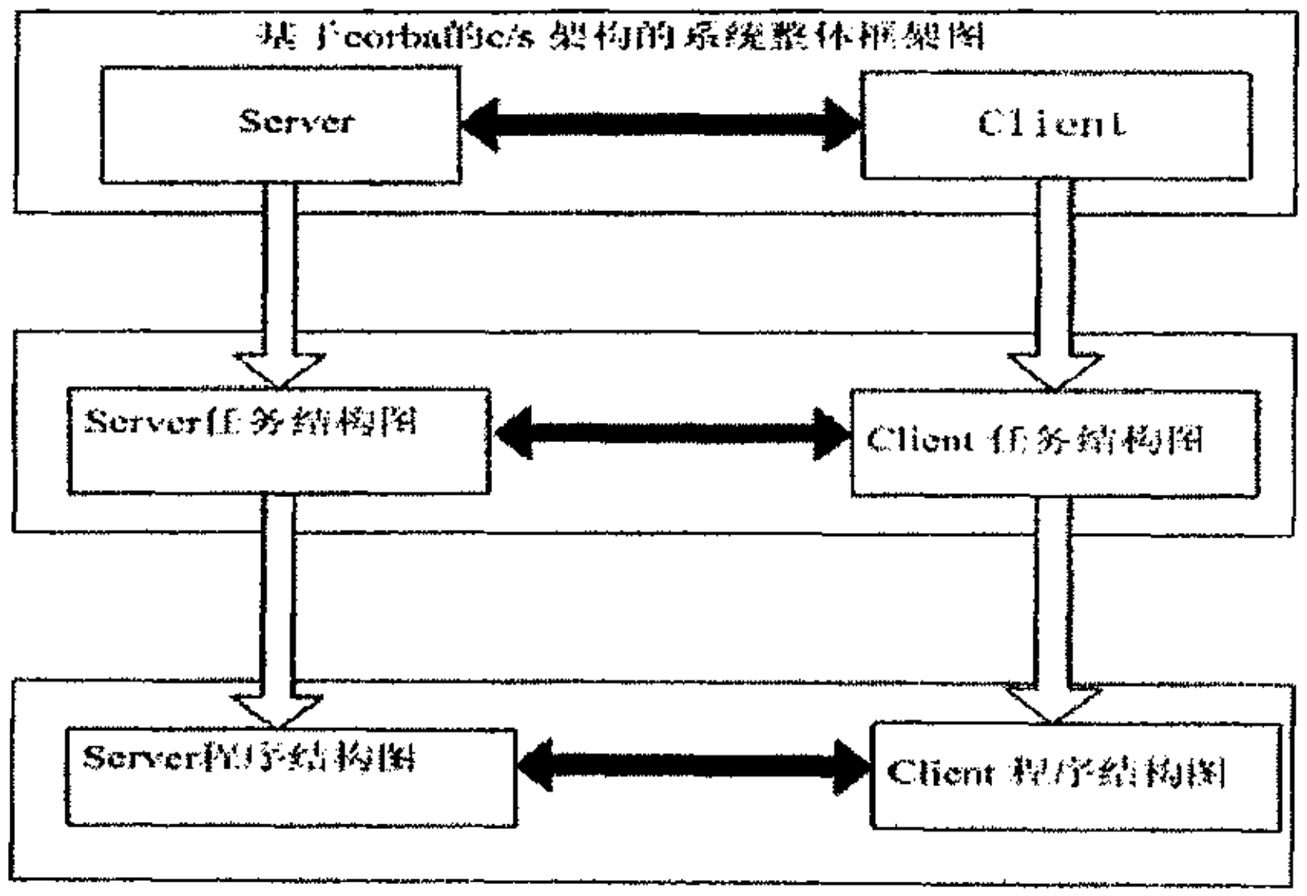


图 4-2 仿真环境的三层模式结构

仿真系统共分为三层，第一层是服务器端和客户端的整体结构图，在视图中以图形化的方式来表示，服务器端只有一个，客户端可以有若干，服务器端向客户端提供服务，客户端向服务器端提出服务请求。

在划分好服务器与客户端两个模块后，可以进一步把服务器（or 客户端）分解为数个任务。服务器端和每个客户端对应一张任务结构图，处于第二层，在任务级的结构图中，再把对应的任务和任务之间的通信，同步互斥关系表示出来，包括 corba 初始化任务和在服务器端具体实现客户请求的 Corba 服务任务。

每个任务，包括中断都对应一张程序结构图，位于第三层，表示程序的执行流程，在普通程序结构图中，用户能调用普通的操作系统的调用，而 corba 程序结构图中，用户能调用基于操作系统的调用和 DeltaCORBA 的基本调用，在客户端的程序结构图中，还能调用 corba 服务任务。

4.5.2.1 服务端/客户端(C/S) 整体结构图

设置这一层主要是基于 Corba 的考虑，由于 Corba 是解决计算机网络的异构问题的，所以该仿真环境设计为服务器端/客户端(C/S)的结构，服务器端向客户端提供服务，客户端向服务器端提出服务请求。在使用 Corba 时，服务器

端和客户端都要进行 Corba 任务的初始化(ORB_init)，其中服务器端运行 corba 初始化任务，启动监听线程，监听来自客户端的服务请求。如图 4-3 所示：

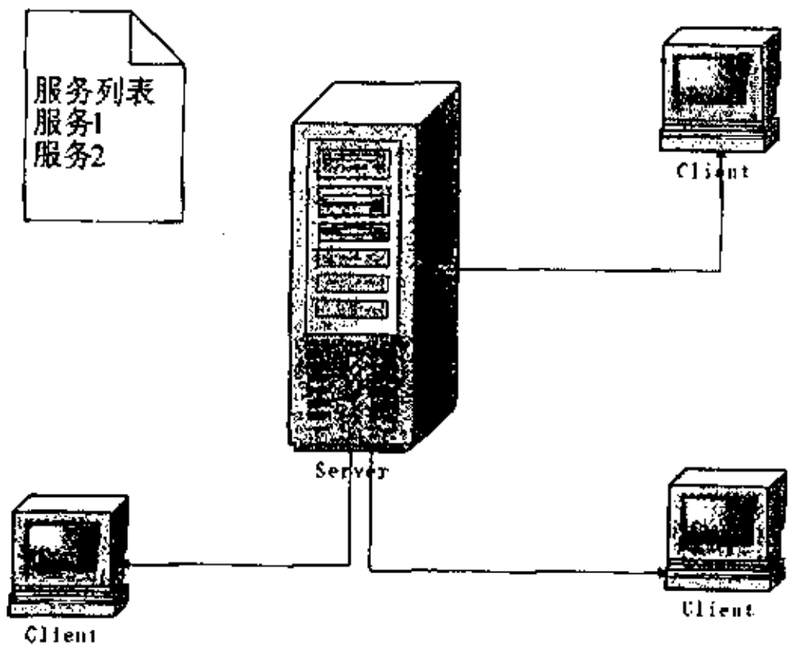


图 4-3 服务端/客户端(C/S) 整体结构图

值得注意的是，本仿真环境只是模拟 DeltaCorba 运行，而不是真正 DeltaCorba 在运行，必须考虑 DeltaCorba 调用对嵌入式应用系统实时结果的影响，以正确预测嵌入式应用系统的实时性能。

4.5.2.2 任务结构图

在划分好服务器与客户端两个模块后，可以进一步把服务器（or 客户端）分解为数个任务，在任务结构图中表示出来。每个服务器端和客户端都对应着一张任务结构图，每张任务结构图代表一个嵌入式应用系统。在任务结构图中，把对应的任务和中断，以及它们之间的通信，同步互斥关系以图形化的方式表示出来，包括 Corba 初始化任务。

在客户端的，一张简单的任务结构图如图 4-4 所示：

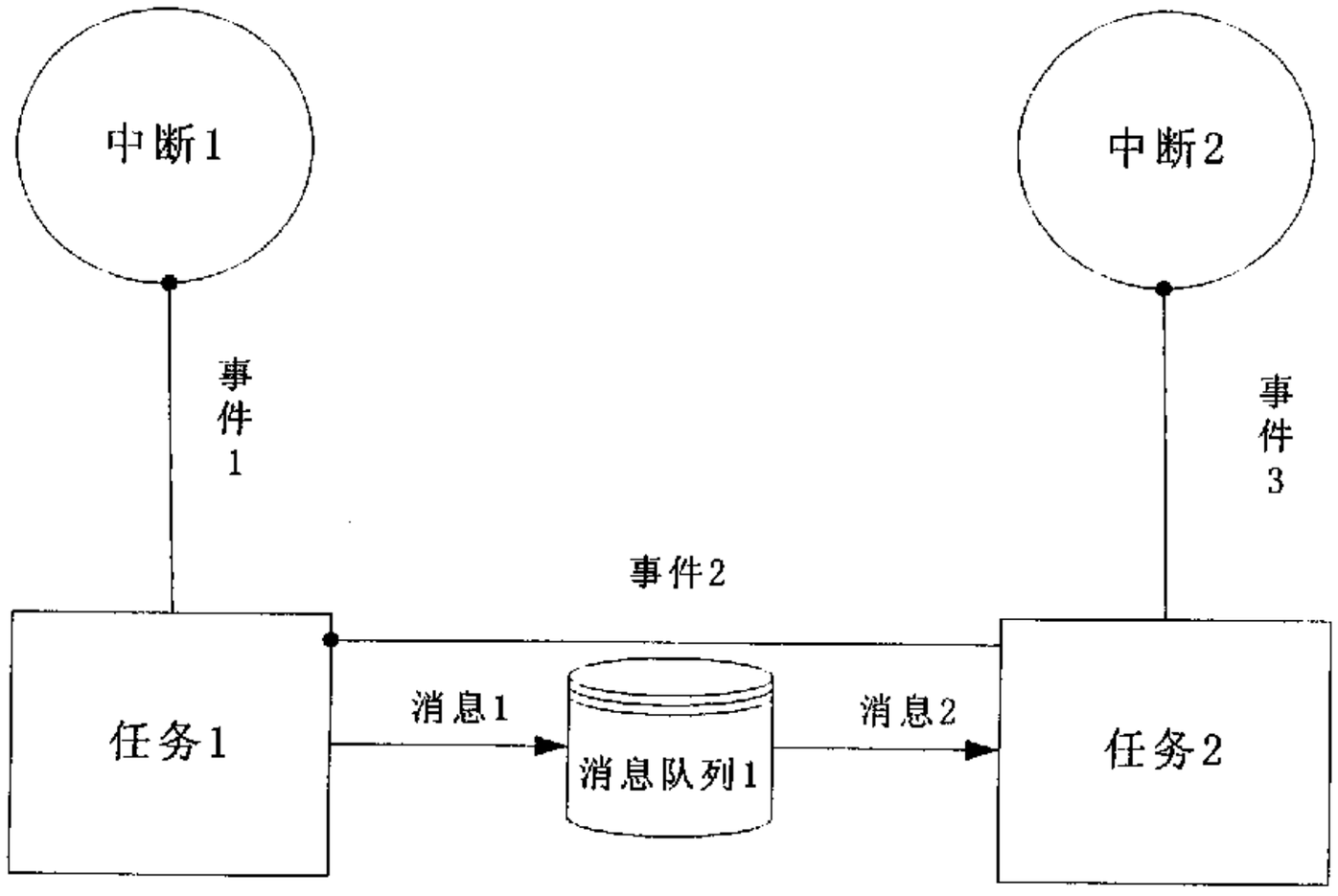


图 4-4　一张客户端的任务结构图

当用户点击键盘时，键盘中断 1 发生，在键盘中断 1 的处理程序中，发事件 1 通知“处理键盘输入任务 1” 有字符键入，然后“处理键盘输入任务 1”从键盘缓冲区中取出字符，发送消息，把它存入消息队列 1 中，并发事件 2 通知“控制屏幕显示任务 2”，然后后者从消息队列 1 中取出字符，经处理后送入屏幕处理的 I/0，在屏幕上显示出来。同时，时钟中断 2 每秒发生，发出“秒发生信号”(即事件 3)给“控制屏幕显示任务 2”，由它把表示当前时间的变量加一后，经处理后送入屏幕处理的 I/0，在屏幕上显示当前的时间。

在服务器端的任务结构图有所不同，它除了能支持上述的任务、中断或消息队列外，还能支持两种特殊的 Corba 任务：

(1) 启动 Corba 的任务，我们称之为 Corba 初始化任务，它运行 ORB_init，初始化 ORB，并运行 orb -> run()后，启动监听任务，监听来自客户端的服务请求。

(2) 在服务器端具体实现客户服务请求的操作在 Server 任务结构图中使用特殊的 Corba 服务例程来表示，用户需要在 Corba 服务例程的程序结构图中描述出响应用户请求的具体操作流程，当然如果不涉及任务间的同步或互斥的操作，也可以仅用只有时间特性的时间块来表示。一旦有来自客户端的服务请求，监听任务执行相应的服务例程。

服务器端的任务结构图如图 4-5 所示：

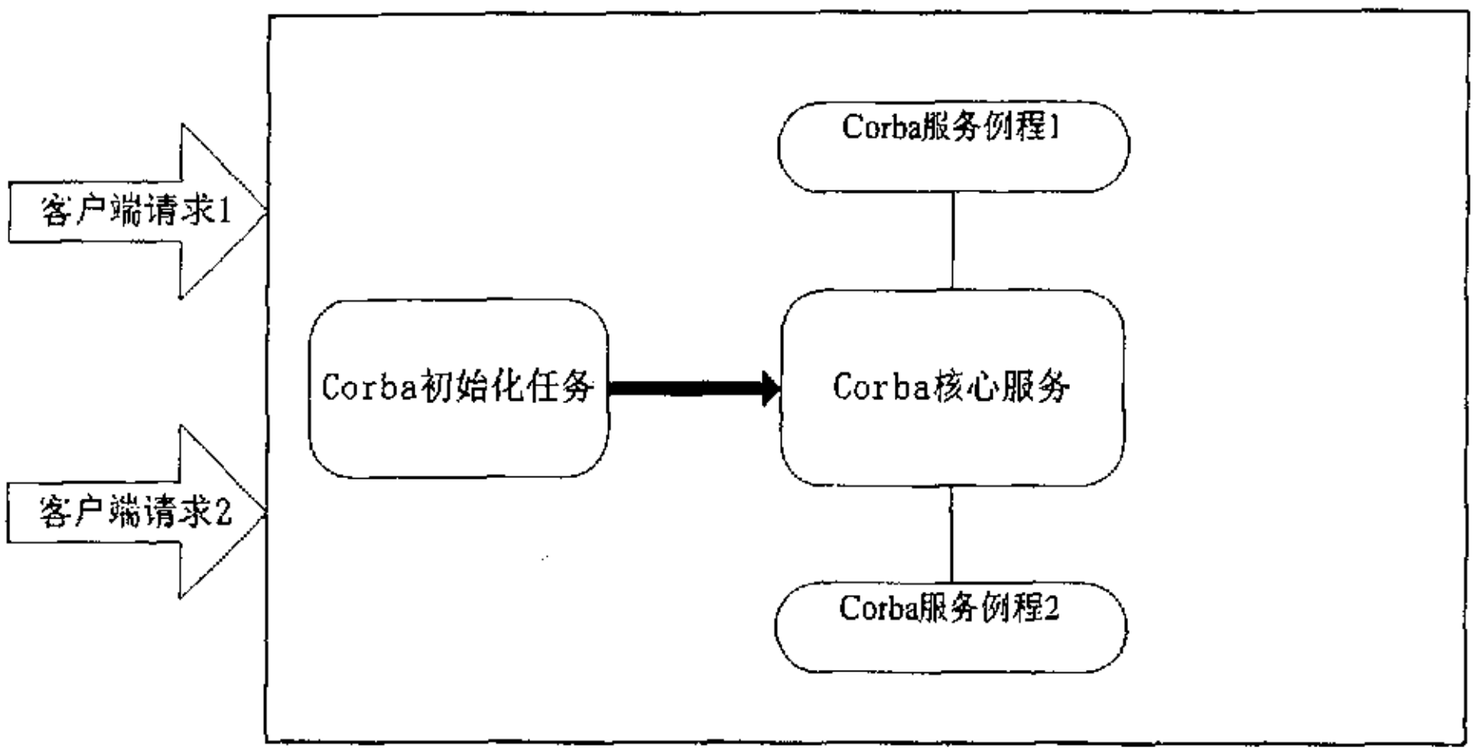


图 4-5 服务器端的任务结构图

4.5.2.3 程序结构图

代表一个嵌入式应用系统的具体工作流程，以图形化伪代码流的方式来表达。任务结构图中的每个任务都对应一张程序结构图，中断和 Corba 应用服务程序也对应一张程序结构图。在普通任务的程序结构图中，用户能调用普通的操作系统的调用，如发送消息、挂起任务等。需要注意的是，由于本仿真环境仅关注的是系统的实时性能，所以仅模拟可能引起任务状态变化，以至产生任务重调度的调用，至于任务中顺序执行的代码，只用把它们的时间特性表示出来就可以了。一张普通任务的程序结构图如图 4-6 所示：

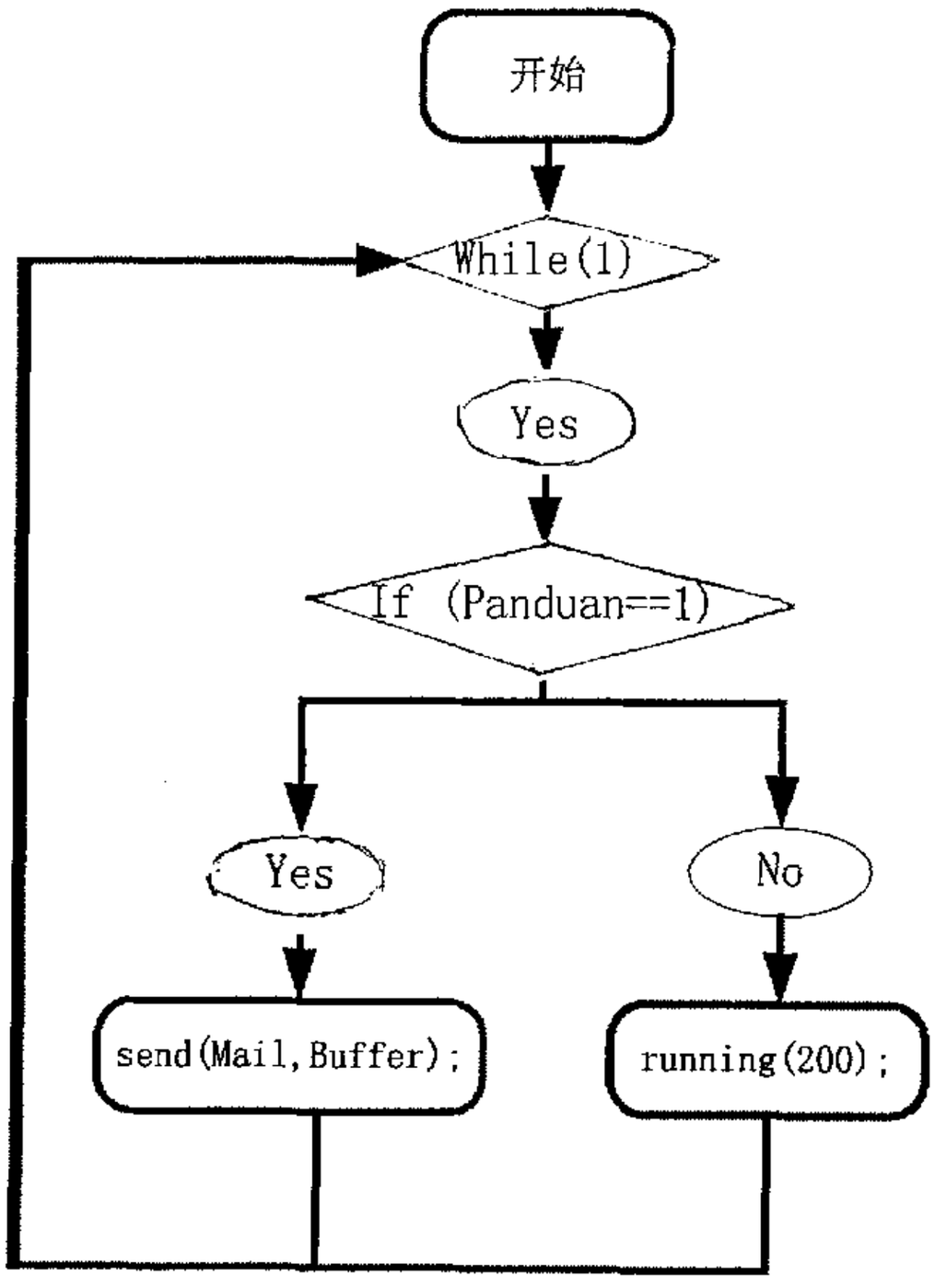


图 4-6 普通任务的程序结构例图

图中的 send 表示把消息 Buffer 发送到 Mail 的信箱，而 running 表示顺序执行了 200 个微秒的时间块。

而在 Corba 任务的程序结构图中，用户能调用基于操作系统的调用和 Corba 调用。一个客户端的 Corba 程序结构图的实例如图 4-7 所示:ORB_init 表示使用 Corba 前初始化 ORB,这里主要是模拟它的时间特性，corba_request 表示客户端向服务器端提出服务请求，服务器端执行服务例程，并把结果返回给客户端。最后 suspend 表示挂起该任务。

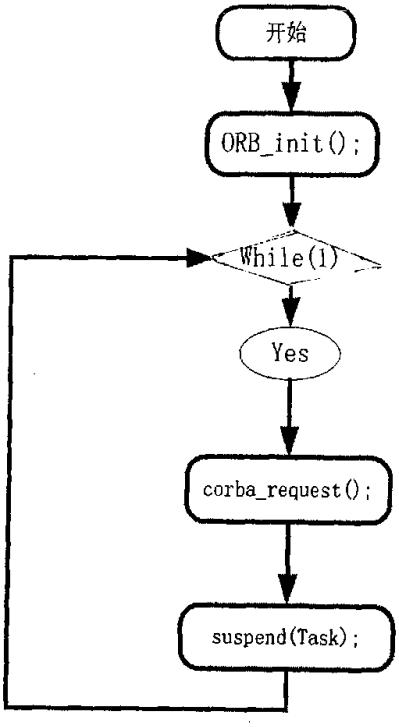


图 4-7 一个客户端的 Corba 任务的程序结构图

4.5.3 嵌入式应用系统的实时性能预测的后台设计

在用户完成对嵌入式应用系统的三层模式结构图的设计后，就可以根据三层结构图的内容对嵌入式应用系统编译分析，并模拟运行，对它的实时性能进行分析预测，并把结果存储在一个事件序列中。

后台的运行分析分为两步：

第一步，是启动一个自主开发的、小型的编译器，以整体结构图、任务结构图和程序结构图作为输入，通过结构图形分析器产生对应于各个任务的中间代码序列，每一条中间代码类似于一个四元组。程序结构图中的每个图标都会产生一个指令数组(即中间代码序列组)，记录该图标所执行的指令信息。

第二步，根据选定的实时操作系统调用的时间数据库和 Corba 调用时间数据库，模拟一个运行该实时操作系统的虚拟机，执行上一步产生的中间代码，并将其间产生的一系列事件以一定的格式记录在结果文件中。

具体过程如图 4-8 所示：

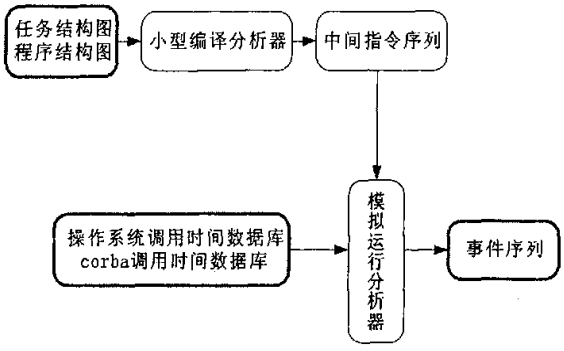


图 4-8 后台运行分析简图

4.5.4 分析结果表示

为了方便用户对分析结果的观察，本仿真环境可以用两种方式显示输出的事件序列。一种是列表方式，一种是图形方式。

a. 列表方式是指将事件序列按时间先后顺序列表以文字的形式显示出来，每个事件条目的内容如下：

事件序号	发生该事件的任务名称	事件的名称	事件的起始时刻	事件的持续时间（即时间消耗）

b. 图形方式，以“事件——时间”图描述软件的运行情况，事件在不同的任务或中断上展开，用户可清晰地看到系统的运行情况，如图 4-9 所示：

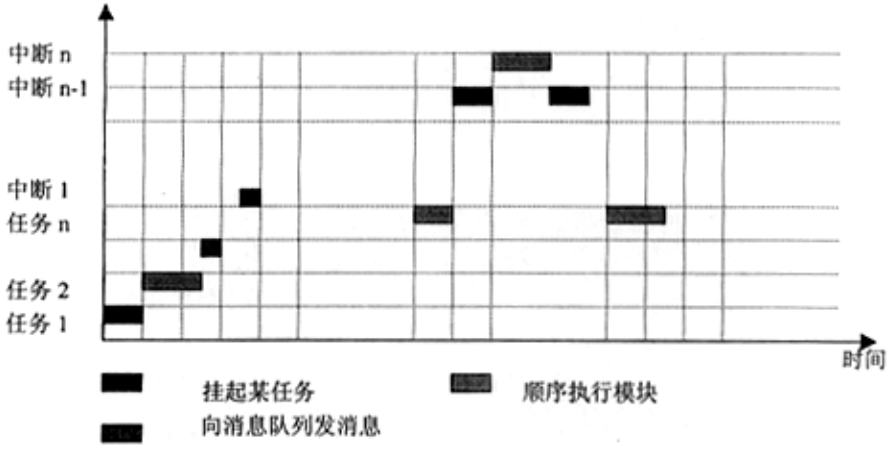


图 4-9 事件—时间图

4.5.5 结构

1)前台视图及文档模块：与用户交互的部分，用户通过若干张视图输入嵌入式应用系统的软件模型，并存储于该部分的文档中。

2）编译模块：一个小型编译器。把存储于前面部分的文档中的包含图形流程信息的数据进行编译，得到中间指令序列。

3）模拟运行模块：模拟一个运行该实时操作系统的虚拟机，执行上一步产生的中间代码，并将其间产生的一系列事件以一定的格式记录在结果文档中。

4）结果显示视图及文档模块：负责把存储在结果文档中的事件序列，在两张视图中以列表和图形两种方式显示出来。

第五章 仿真环境的实现

5.1 仿真环境的功能模块

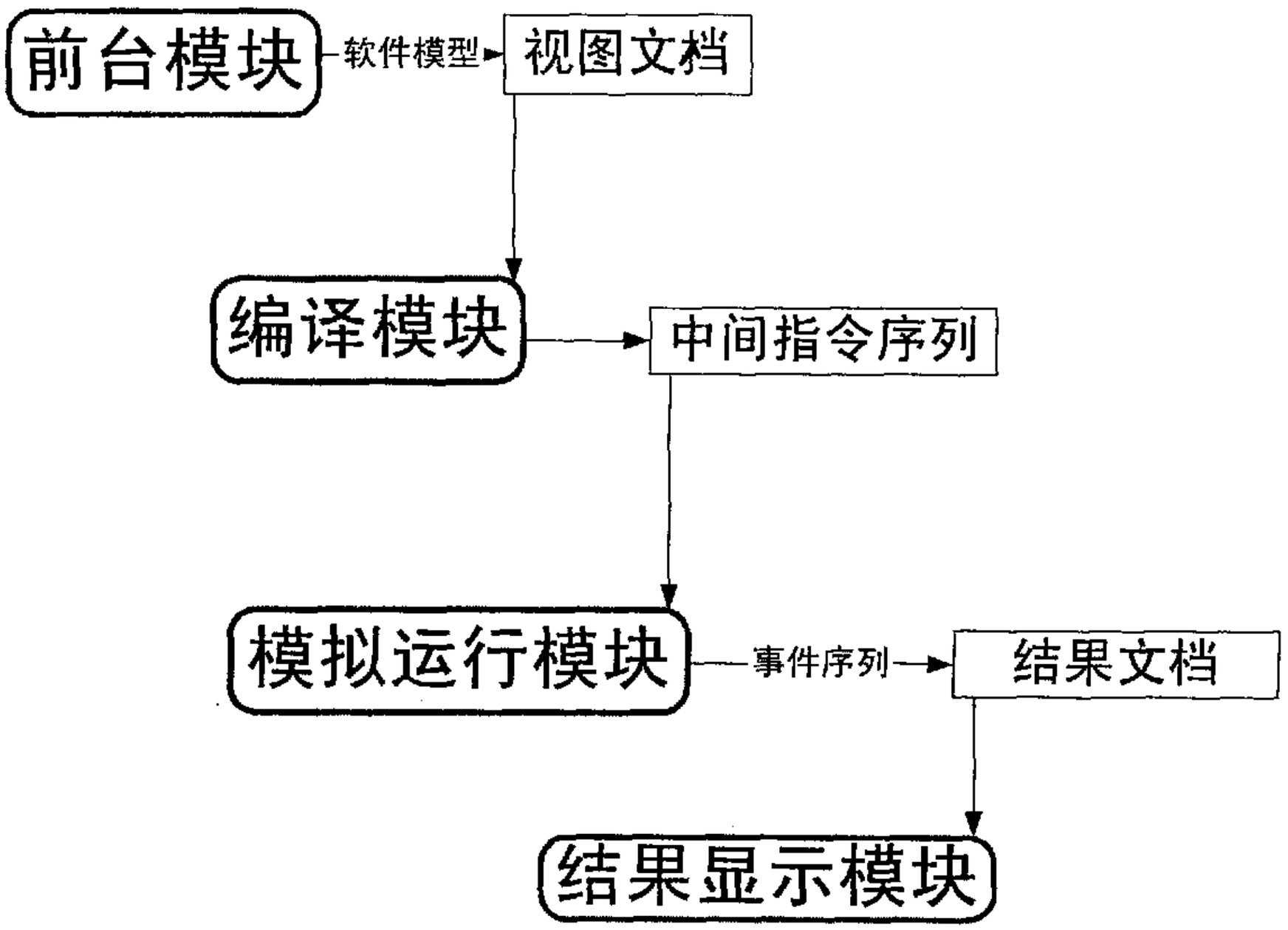


图 5-1 仿真环境的功能模块

如图 5-1，仿真环境将按如上所示的四大功能模块依次构建，前台模块向用户提供若干视图，以图形方式帮助用户构建嵌入式应用系统的软件模型，并以某种形式存储于该视图的文档中。编译模块负责分析该软件模型，得到中间代码，即中间指令序列。然后启动模拟运行模块，模拟运行该嵌入式应用系统，得到按时间顺序排列的事件序列，存贮于结果文档中，结果显示模块负责把存储在结果文档中的事件序列，在两张视图中以列表和图形两种方式显示出来。

5.2 仿真环境需要创建的几个主要类

本仿真环境是一个多窗口、多文档类型的 Windows 应用程序，它主要的几个类为：

1)和前台模块相关的类：绘制服务器端/客户端(C/S) 整体结构图和处理相关用户输入的视图类 CCorbaView 和存储服务器端/客户端(C/S) 整体结构图的文档类 CCorbaDoc；绘制任务结构图和处理相关用户输入的视图类 CPjView 和绘制程序结构图的视图类 CTaskView,以及存储两者信息的文档类 CPjDoc;另外还包括描述任务程序结构的图标类 CGraph 类，描述任务 CNodeNew 类，描述邮箱等任务间同步对象类 Cmail。

2)编译模块：CMinicompiler 类。

3)模拟运行模块：CCalculate 类。

4)结果显示模块：存储评价结果数据的文档类 CResultDoc，还有视类 CResultView 和 CDrawView, 在视类 CResultView 中，CResultDoc 类对象中的数据被列表显示出来。在视类 CDrawView 中，CResultDoc 类对象中的数据以“事件——时间图”的方式显示出来。

5.3 仿真环境所模拟的嵌入式系统调用

在 DeltaCORE2.0 的基础上，按照只模拟实现那些有可能产生任务重调度的系统调用的原则，本仿真环境所模拟的嵌入式系统调用如下：

running(n):一个顺序执行的时间块，n 代表执行了多少微秒。

send(MailBox,MailText):发送消息 MailText 到信箱 MailBox。

receivef(MailBox,MailText):从信箱 MailBox 中接收消息 MailText，若信箱中没有消息被接收，则调用任务被挂起，直到信箱中有消息，则被激活。

sunit(sem):申请信号量 sem，若申请不到，则调用任务被挂起。

runitf(sem):释放信号量 sem。

eventsend(Task,event):向任务 Task 发送事件 event。

eventreceive(event,WAIT/NOWAIT):接收事件 event,若没有事件可以接收，则依据参数 WAIT/NOWAIT 判断是否挂起调用任务。

suspend(Task):挂起任务 Task。

resume(Task):激活任务 Task。

5.4 四大功能模块的设计与实现

5.4.1 前台模块

前台模块是和用户交互的部分，以三层结构图的方式从上到下，从大到小，从整体到局部帮助用户建立嵌入式应用系统软件模型。三层结构图如图 4-2,分别为服务器端/客户端(C/S) 整体结构图，任务结构图和程序结构图，显示和处理这三个结构图的视图类分别为 CCorbaView,CPjView,CTaskView,另外还有存储这三张视图信息的两个文档类 CCorbaDoc 和 CPjDoc。

5.4.1.1 前台模块的文档模板

在 MFC 的应用程序中，文档模板将文档类与子边框窗口类、视类紧密联系在一起。和前台模块相关的有三个文档模板，将上面所述的三个视图类和两个文档类联系在了一起。

```
m_pCorbaTemplate=new CMultiDocTemplate(
    IDR_PJTYPE,
    RUNTIME_CLASS(CCorbaDoc),
    RUNTIME_CLASS(CChildFrame),
    RUNTIME_CLASS(CCorbaView));
AddDocTemplate(m_pCorbaTemplate);
EnableShellOpen();

m_pPjTemplate = new CMultiDocTemplate(
    IDR_PJTYPE,
    RUNTIME_CLASS(CPjDoc),
    RUNTIME_CLASS(CChildFrame),
    RUNTIME_CLASS(CPjView));
AddDocTemplate(m_pPjTemplate);

m_pTaskTemplate= new CMultiDocTemplate(
    IDR_TASKTYPE,
    RUNTIME_CLASS(CPjDoc),
    RUNTIME_CLASS(CChildFrame),
    RUNTIME_CLASS(CTaskView));
AddDocTemplate(m_pTaskTemplate);
```

5.4.1.2 前台模块的三张结构图

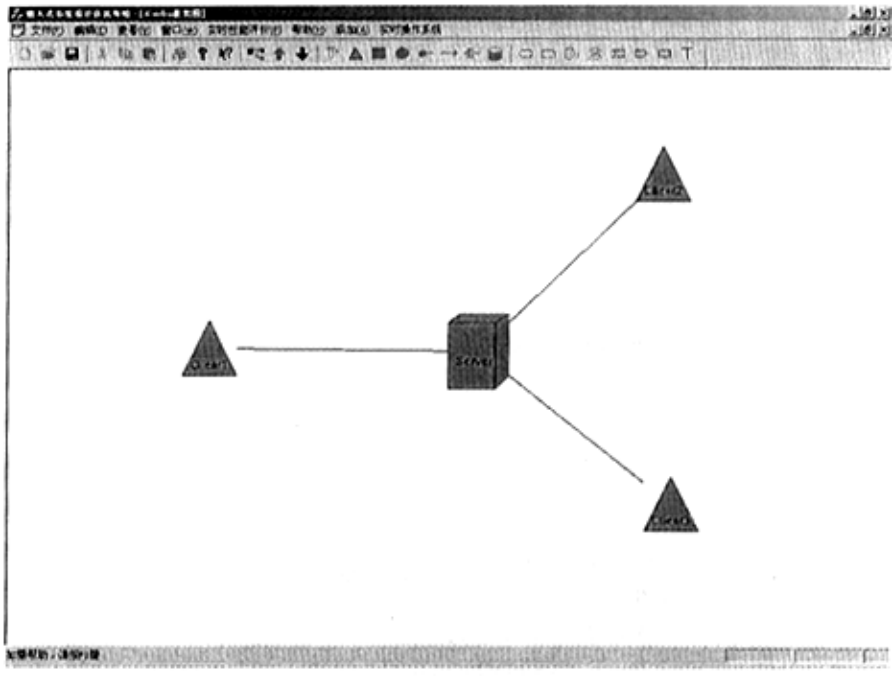


图 5-2 服务器端/客户端(C/S) 整体结构图

如图 5-2,在三层模式的第一层服务器端/客户端(C/S) 整体结构图中，可以看到四个图标，并用线条表示它们之间的相互关系，代表一个服务器端和三个客户端。双击任何一个图标，都可以进入到该客户端或服务器端相对应的任务结构图中，如图 5-3。

图 5-3 所实现的是图 4-4(一张客户端的任务结构图),其中有任务、中断、消息队列（信箱），还有事件和消息两种通信机制。双击任务或中断的图标，则可以进入程序结构图，进行程序结构流程的设计。如双击图 5-3 中任务 1 的图标，就可以进入任务 1 的程序结构图，如图 5-4，有代表循环和判断两种逻辑关系的图标和线条，还有接收事件 eventreceive,发送消息 send,发送事件 eventsend 的图标，这些图标都是依据任务结构图(5-3)中的通信关系自动生成的。这样依次就可以建立起应用系统的软件模型。

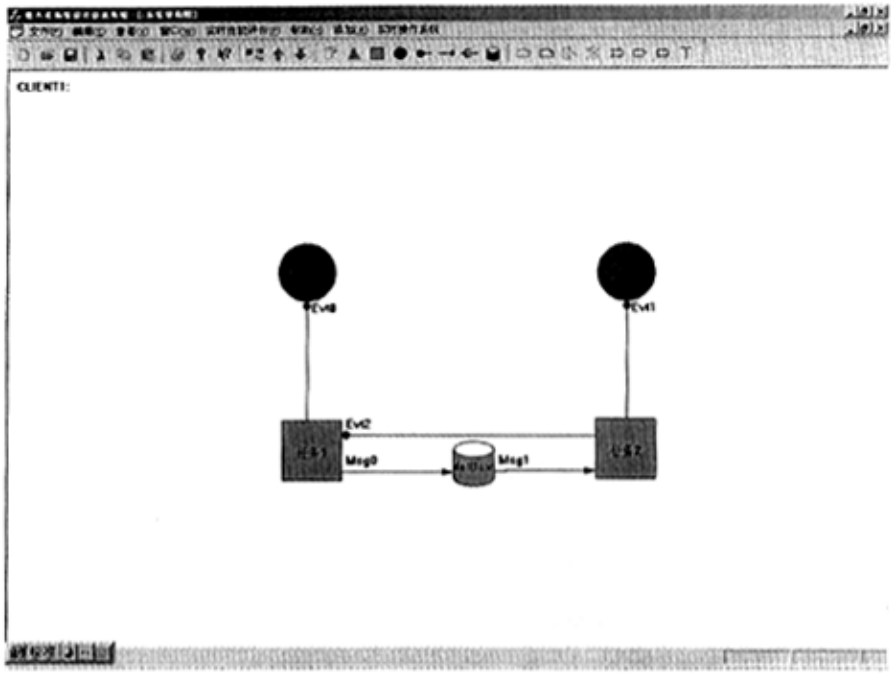


图 5-3 任务结构例图

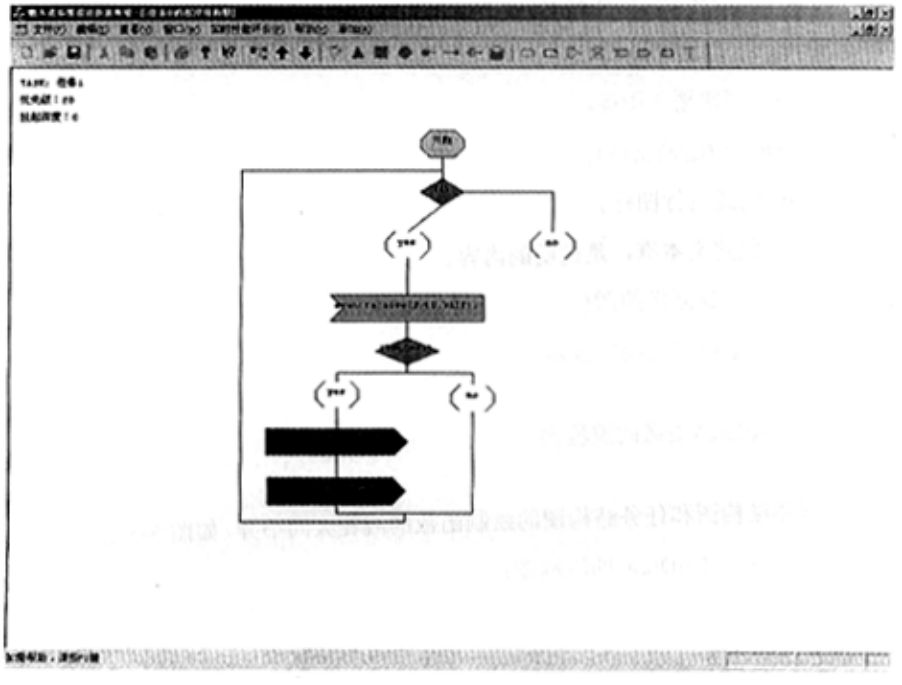


图 5-4 任务 1 的程序结构图

5.4.1.3 前台模块的几个主要函数

CCorbaView：

OnAddClient：往整体结构图中增加一个客户端图标。

OnDraw：整体结构图的绘制函数。

CPjView：

OnAddTask：新建一个任务图标，包括创建一个新的任务类 Cnodenew 实例。

OnAddInt：新建一个中断图标。

OnAddMailbox：新建一个信箱（消息队列）图标。

OnAddMux：新建一个信号量图标。

OnAddMessage：新建一个消息图标。

OnAddMail：新建一个事件图标。

OnAddText：新建一个文本框。

OnDraw：任务结构图的绘制函数。

CTaskView：

OnAddBegin：创建 begin 开始图标。

OnAddBranch：创建判断图标。

OnAddInput：创建输入图标。

OnAddOutput：创建输出图标。

OnAddRun：创建运行图标。

OnAddText：创建文本框，是说明的内容。

OnAddWhile：创建循环的图标。

OnDraw：程序结构图的绘制函数。

5.4.1.4 结构图绘制函数的流程图

C/S 整体结构图和任务结构图的绘制函数的流程大同小异，如图 5-5 是任务结构图的绘制函数 OnDraw 的流程图：

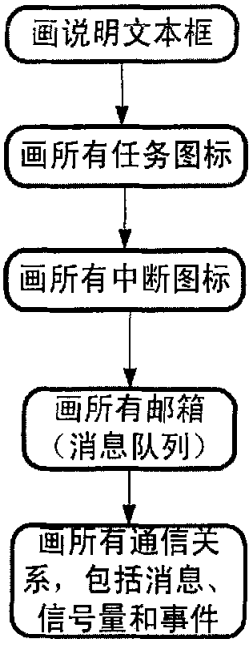


图 5-5 任务结构图的绘制函数 OnDraw 的流程图

至于程序结构图的绘制函数的流程则要复杂得多，采用先序遍历，从 begin 开始图标为根节点，绘制程序结构图的流程树。图 5-6 简要表示如下：

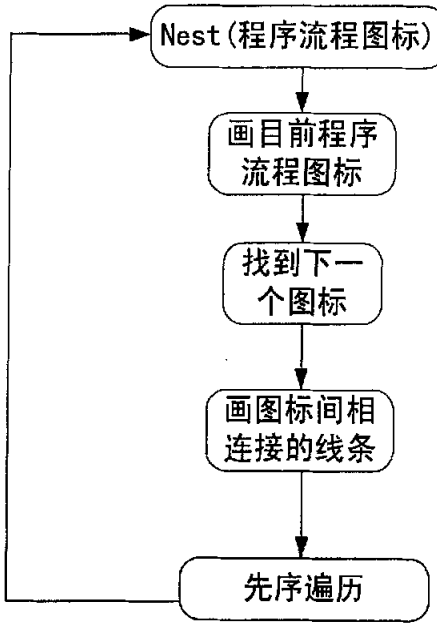


图 5-6 程序结构图的绘制函数的流程图

5.4.2 编译模块

在前台模块中所得到的软件模型的信息，将会被存储到CCorbaDoc和CPjDoc的若干对象数组中，编译模块的作用就是负责分析该软件模型，得到中间代码，即中间指令序列。

本仿真环境的编译程序只是一个小型的编译器，它的目标仅是生成中间代码，不包括优化和生成目标代码，编译模块的总框如图 5-7 所示，

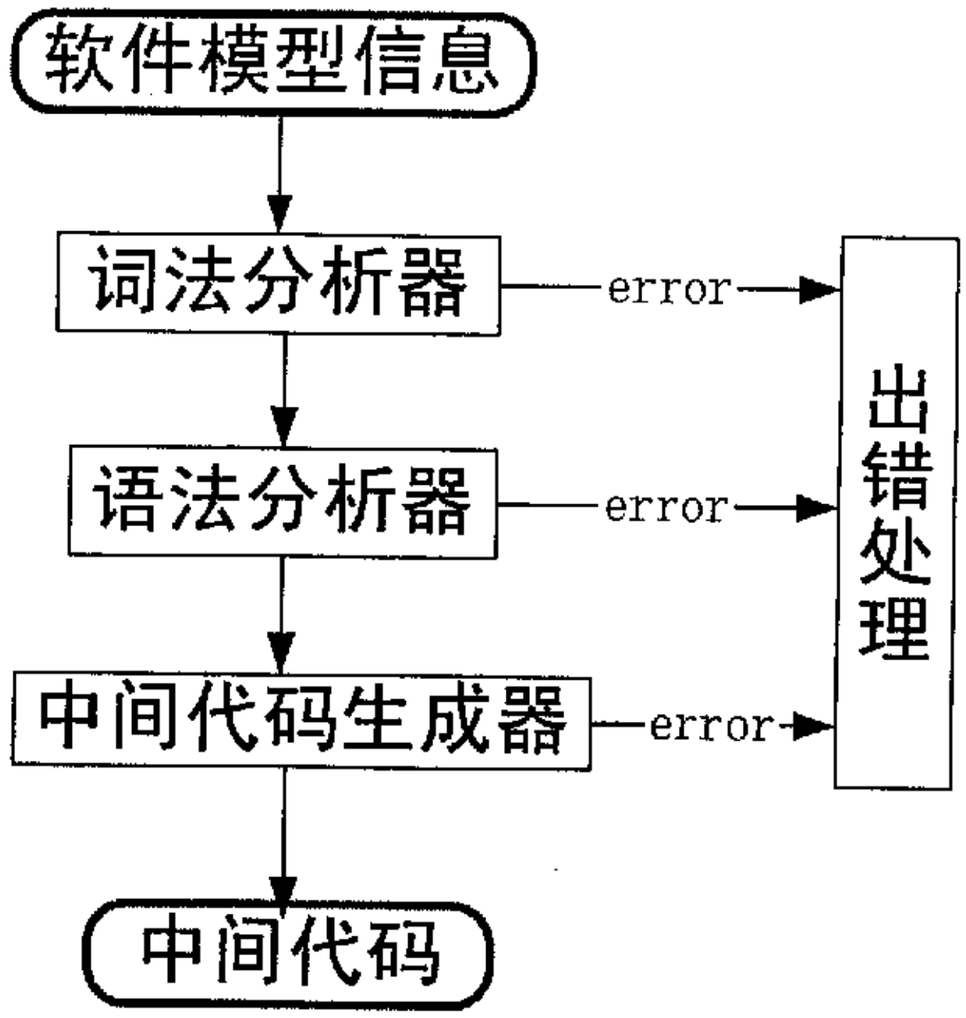


图 5-7 编译模块总框

应用系统软件模型的信息存储在 CCorbaDoc 和 CPjDoc 的若干对象数组中，编译模块的词法分析器以这些对象数组为输入参数，进行词法分析，输出单词符号。

语法分析器对单词符号串进行语法分析，输出由语法单位构成的语法树，判断输入串是否构成语法上正确的“程序”。例如判断或循环的逻辑表达式的语法分析：每当读入一个逻辑表达式的单词符号串，则结合 AND 运算量堆栈和 OR 运算符堆栈进行语法分析，首先对逻辑表达式相邻两个逻辑字进行逐步检查，看是否符合逻辑，并归类，最后通过 temp 返回逻辑表达式的最终结果。

中间代码生成器按照语义规则把语法分析器归约出的语法单位翻译成一定形式的中间代码，即中间指令式，存放于相应程序流程图标对应的指令式表中，供模拟运行时使用。中间指令式的格式如图 5-8，其中操作数由两个域所构成：

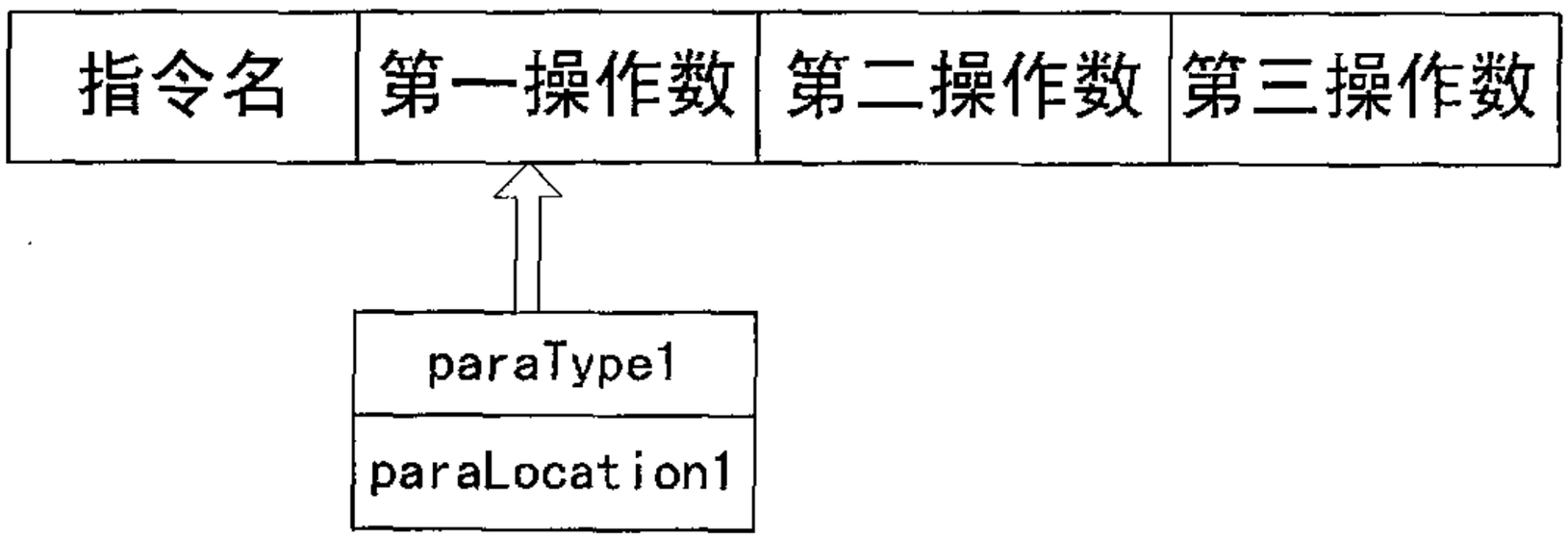


图 5-8 中间指令式

paraType 和 paraLocation,paraType 代表的是操作数的类型，paraLocation 代表的是操作数的数值或是在全局或局部变量表中的位置序号。

词法分析器、语法分析器和中间代码生成器三者并非是截然分开的，而是互相穿插的。

5.4.3 模拟运行模块

模拟运行模块的作用是根据前台模块得到的嵌入式应用系统软件模型和编译模块得到的中间指令式表，对该嵌入式应用系统进行模拟仿真运行，得到按时间顺序排列的事件序列，提供给结果显示模块，分析其实时性能。

5.4.3.1 任务链表

为了应用系统模拟仿真运行的需要，基于链表存取方便的优点，所以需要建立就绪任务链表，挂起任务链表和中断链表。

就绪任务链表和挂起任务链表是一个按优先级排列的单向链表，而中断链表把中断按中断发生时间排列。如图 5-9，是就绪任务链表或挂起任务链表的结构简图。在仿真环境中，就绪任务链表、挂起任务链表和中断链表分别为 m_PriList、m_SuspendList 和 m_IntList。这样在应用系统模拟仿真运行时，就可以很方便的进行任务状态的变换。如当需要把一个正在执行的当前任务挂起时，只需要把当前任务的任务控制单元从就绪任务链表 m_PriList 中断下，加入到挂起任务链表 m_SuspendList 中，同时从就绪任务链表 m_PriList 中选取头节点，即优先级最高的任务的控制单元，投入运行。

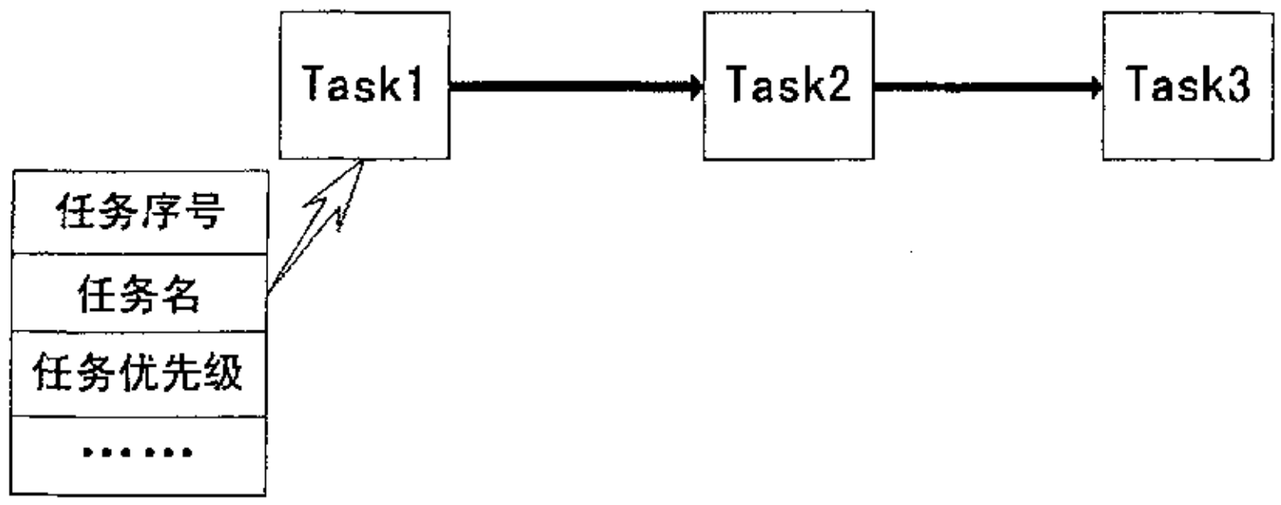


图 5-9 任务链表

5.4.3.2 模拟运行的算法

负责模拟运行的主要是 Ccalculate 类，大致过程如图 5-10。

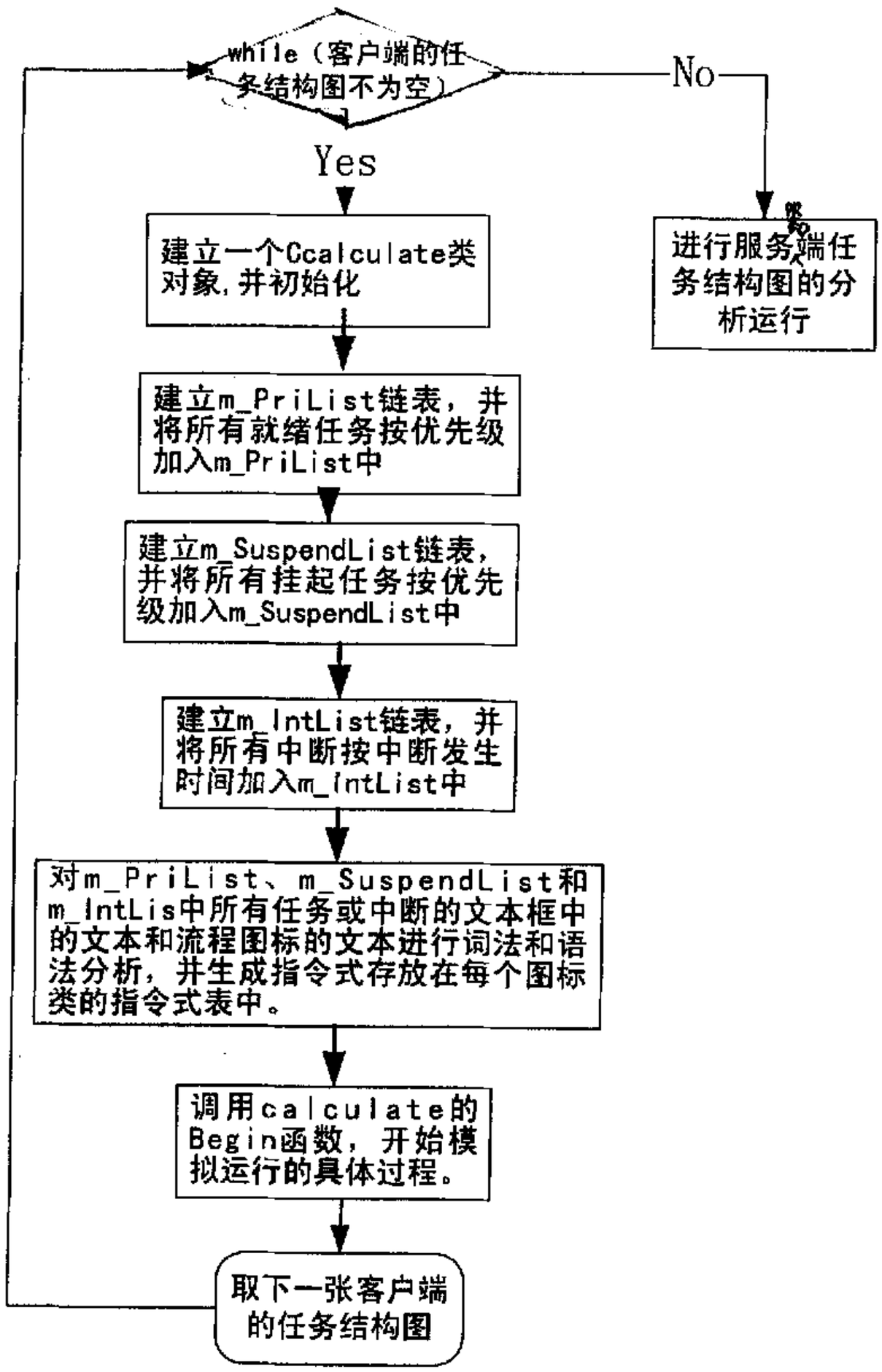


图 5-10 模拟运行过程

首先是对客户端的任务结构图进行分析和模拟运行，取一张客户端的任务结构图信息，建立一个负责模拟运行的Ccalculate类对象,并初始化，然后分别建立就绪任务链表，挂起任务链表和中断链表，并完成初始化，然后对m_PriList、m_SuspendList和m_IntList中所有任务或中断的文本框中的文本和流程图标的文本进行词法和语法分析，并生成指令式存放在每个图标类实例的指令式表中，最后调用calculate的Begin函数，基于上述指令式表，开始模拟运行的具体过程，如此重复，直到完成所有的客户端任务结构图的分析和模拟运行，最后进行服务器端任务结构图的分析运行，过程也大致相同。

其中调用 calculate 的 Begin 函数，开始实时性能评价的具体分析过程，如图 5-11：

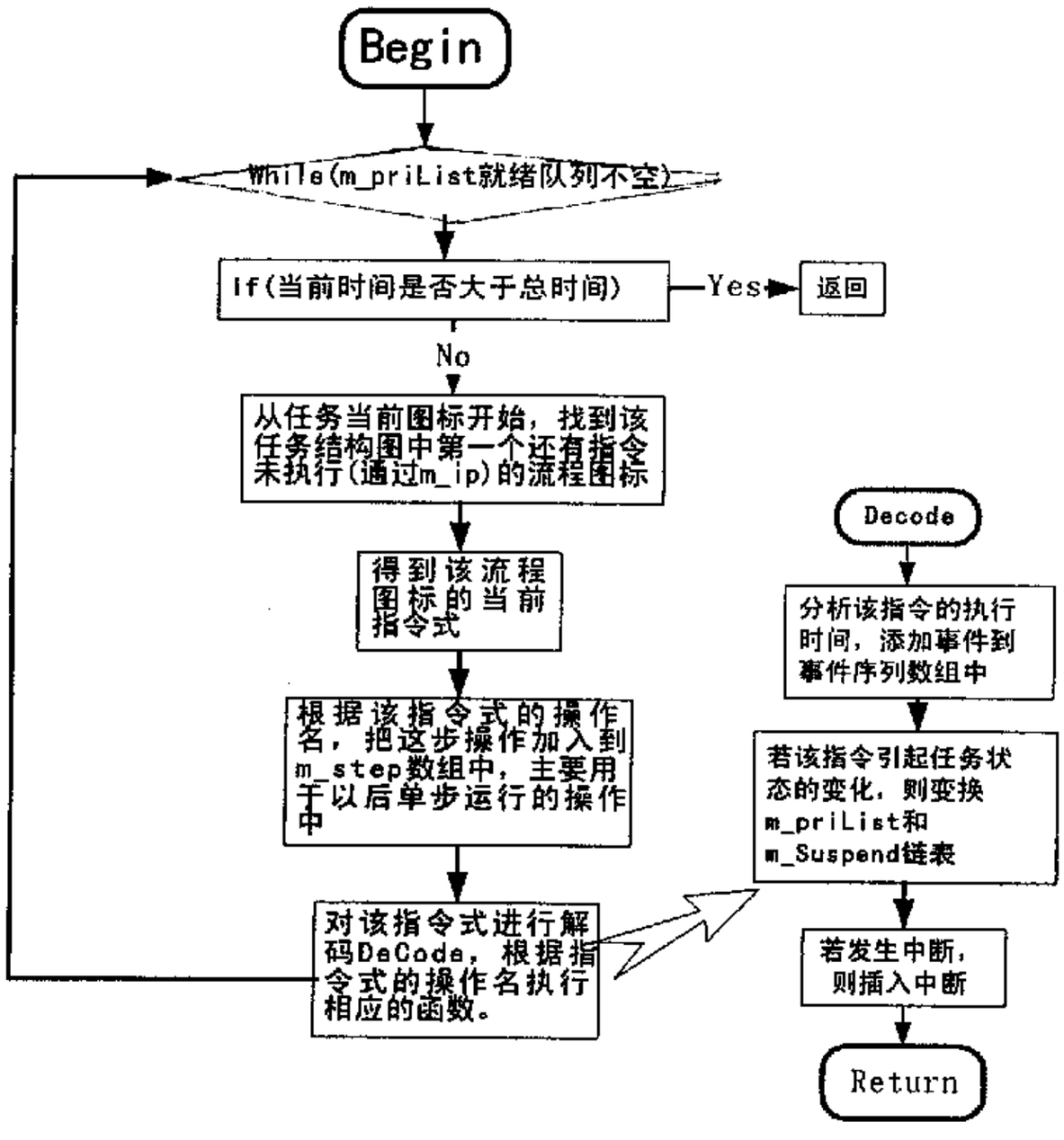


图 5-11 实时性能评价的具体分析过程

首先从就绪任务链表 m_priList 中选取优先级最高的任务，然后判断当前时间是否大于总时间，如果大于，则评价过程结束，否则开始下一步的评价过程，从任务当前图标开始，找到该任务结构图中第一个还有指令没有执行的流程图标，并从该流程图标的当前指令式表中得到该流程图标的当前指令式，根据该指令式的操作名，把这步操作加入到 m_step 数组中，主要用于以后单步运行的操作中，最后对该指令式进行解码 DeCode,根据指令式的操作名去执行相

应的函数。在 DeCode 函数中，首先分析该指令的执行时间，添加这次事件到事件序列数组中，同时当前时间加上该指令的执行时间为新的当前时间。若该指令引起任务状态的变化，则就绪任务链表 m_priList 和挂起任务链表 m_SuspendList 则会发生相应的变换，如果此时间段内有中断发生，还从中断链表 m_IntList 上取下一个中断，分析执行。完成后再从就绪任务链表 m_priList 中选取优先级最高的任务，继续执行，直到所有任务执行完毕，或者当前时间大于总时间。

5.4.4 结果显示模块

结果显示模块是依据按时间顺序排列的事件序列数组，将实时性能结果分别以列表和图形两种方式显示出来，供用户分析和判断系统的实时性能，注意每个客户端或服务器端都各有这两张表。显示这两个视图的视图类分别为 CResultView 和 CDrawView，有一个存储事件序列数组的文档类 CResultDoc。

5.4.4.1 结果显示模块的文档模板

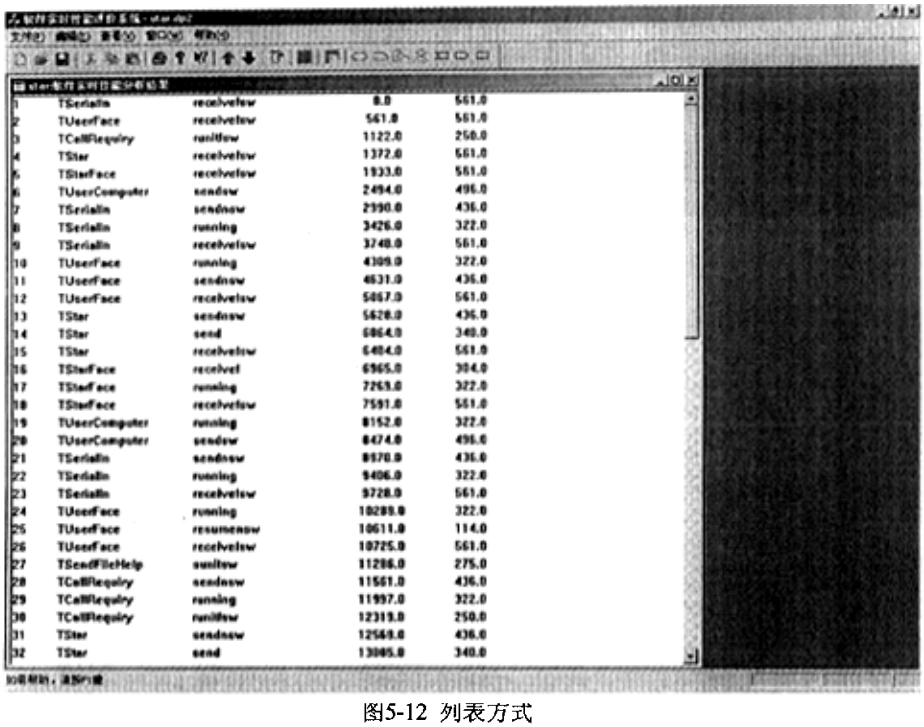
有两个文档模板，将上面所述的两个视图类和一个文档类联系在了一起。

```
m_pResultTemplate = new CMultiDocTemplate(
    IDR_RESULTTYPE,
    RUNTIME_CLASS(CResultDoc),
    RUNTIME_CLASS(CChildFrame),
    RUNTIME_CLASS(CResultView));
AddDocTemplate(m_pResultTemplate);

m_pDrawTemplate = new CMultiDocTemplate(
    IDR_DRAWTYPE,
    RUNTIME_CLASS(CResultDoc),
    RUNTIME_CLASS(CChildFrame),
    RUNTIME_CLASS(CDrawView));
AddDocTemplate(m_pDrawTemplate);
```

5.4.4.2 结果显示的两种方式

如图 5-12 是列表方式，从左到右，每列的内容依次是事件序号、发生该事件的任务名、事件名、事件的发生时刻和事件的持续时间。



1	TSerialIn	receivefrom	0.0	561.0
2	TUserFace	receivefrom	561.0	561.0
3	TCallRequiry	runitfor	1122.0	250.0
4	TSter	receivefrom	1372.0	561.0
5	TSterFace	receivefrom	1933.0	561.0
6	TUserComputer	sendnow	2494.0	496.0
7	TSerialIn	sendnow	2990.0	436.0
8	TSerialIn	running	3426.0	322.0
9	TSerialIn	receivefrom	3748.0	561.0
10	TUserFace	running	4309.0	322.0
11	TUserFace	sendnow	4631.0	436.0
12	TUserFace	receivefrom	5067.0	561.0
13	TSter	sendnow	5628.0	436.0
14	TSter	send	6064.0	340.0
15	TSter	receivefrom	6404.0	561.0
16	TSterFace	receivef	6965.0	304.0
17	TSterFace	running	7269.0	322.0
18	TSterFace	receivefrom	7591.0	561.0
19	TUserComputer	running	8152.0	322.0
20	TUserComputer	sendnow	8474.0	496.0
21	TSerialIn	sendnow	8970.0	436.0
22	TSerialIn	running	9406.0	322.0
23	TSerialIn	receivefrom	9728.0	561.0
24	TUserFace	running	10289.0	322.0
25	TUserFace	resumenow	10611.0	114.0
26	TUserFace	receivefrom	10725.0	561.0
27	TSendFileHelp	runitfor	11286.0	275.0
28	TCallRequiry	sendnow	11561.0	436.0
29	TCallRequiry	running	11997.0	322.0
30	TCallRequiry	runitfor	12319.0	250.0
31	TSter	sendnow	12569.0	436.0
32	TSter	send	13005.0	340.0

图5-12 列表方式

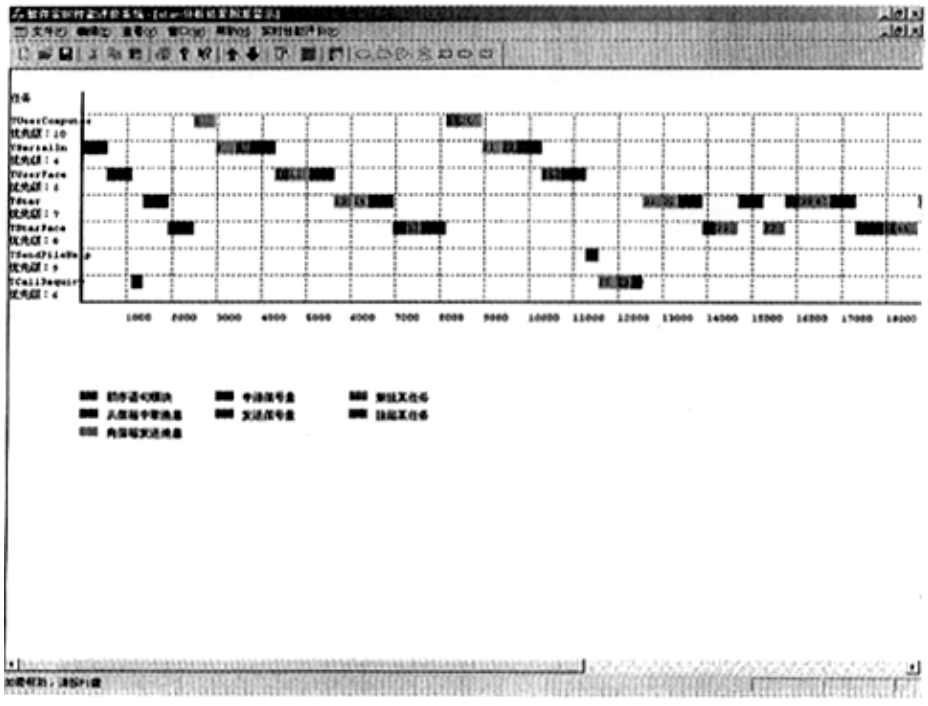


图5-13 图形方式

图 5-13 是图形方式，纵轴是任务轴，横轴是时间轴，随着时间的变化，所执行的任务也在发生，形象的表示出了任务的调度情况。

第六章 嵌入式应用系统设计仿真的实例

6.1 嵌入式实时应用软件系统的生命周期

从软件工程的角度来讲，嵌入式实时应用软件也有一定的生命周期，如下所述：

1. 需求分析与详细说明

分析用户需求，说明系统能满足这些需求的程度；

2. 系统设计

任务分解，将系统分解成任务(并发进程)，定义任务间的接口关系；

3. 任务设计

按模块方式设计每个任务，并定义出模块间的接口；

4. 模块构筑

完成每个模块的详细设计、编码和单元测试；

5. 任务与系统集成

逐个模块连接、测试以构成任务，逐个任务连接和测试形成最终系统；

6. 系统测试

测试整个系统或主要子系统，以验证功能指标的实现，为具有更强的客观性，系统测试最好由一个独立的测试小组执行。

可见同其它通用软件相比，嵌入式应用软件的开发有其独特之处：在系统设计阶段，着眼于将系统划分为多个并发的任务，而非多个模块；要定义任务间的接口关系，而非模块间接口。模块的划分以及模块间接口定义则放在了任务设计阶段。

嵌入式应用系统设计仿真环境就是在嵌入式实时应用软件生命周期前三个阶段即需求分析、系统设计、和任务设计阶段，帮助用户完成设计，生成初期的软件模型，使用户能顺利进入下一个阶段。

6.2 实例说明

本实例是一个机器人控制器，该机器人控制器控制六个转轴，并由面板指示

灯显示其当前状态，它们都由程序控制。该程序由控制面板操作启动执行，控制面板包括若干按钮和一个程序选择开关，如图 6-1 所示。

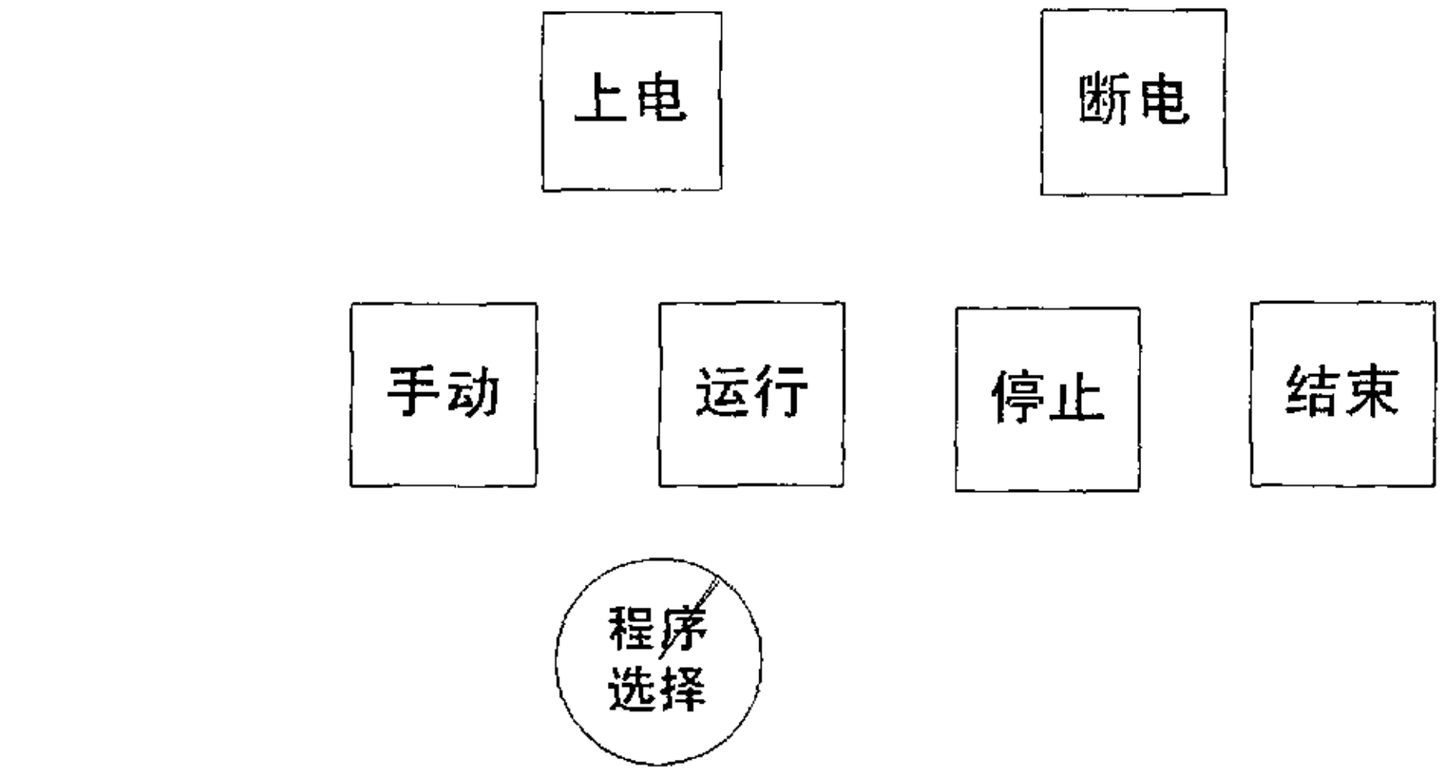


图 6-1 控制面板

按下“上电”按钮，系统进入了上电状态。成功地完成了上电后，系统进入了手动状态。这时，操作者可以通过程序选择开关选择程序，即旋转开关，使它指向期望的程序号。按下“运行”按钮，则选定的程序开始运行，系统转为运行态。程序运行中如果按下“停止”键，程序被挂起，这时，系统进入挂起态。之后，操作者可以按下“运行”键，使程序恢复执行，也可按下“结束”键，结束程序。按下“结束”键后，系统进入终止态。相应的面板指示灯显示其当前状态。状态变迁如图 6-2 所示：

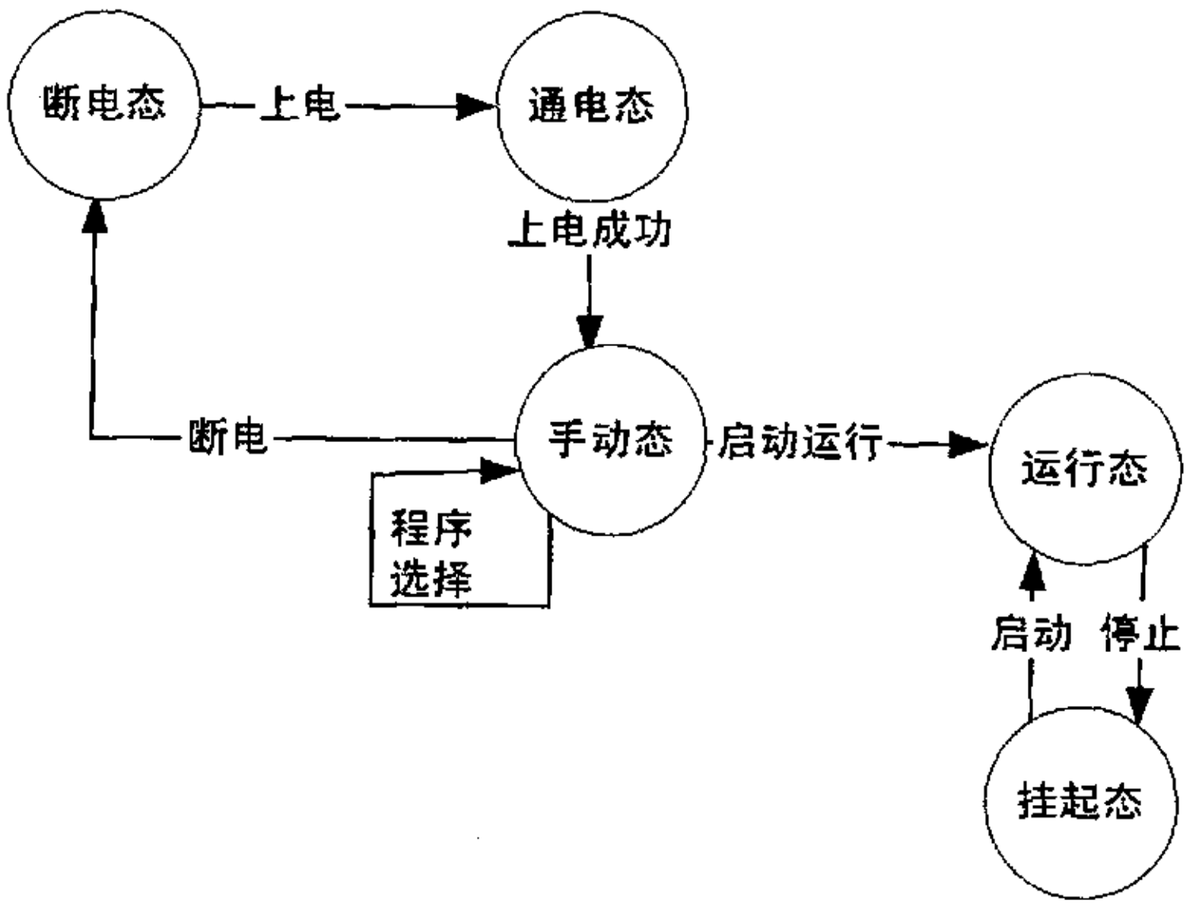


图 6-2 状态变迁图

6.3 系统设计

6.3.1 数据流分析

如图 6-3 为机器人控制器的数据流图。读入控制面板的输入并进行确认。每次按下按钮，输入就被 RPI(读面板输入)读入，并转换成系统内部格式。然后，控制面板输入被传送给 VPI(面板输入有效性检查)。经过确认的控制面板输入被传送到 PPI(处理面板输入)，在那儿对其进行处理，然后传送给相应的变换，或是 IPS(解释程序各语句)，或是 OAD（输出动作轴数据)。此外，PPI(处理面板输入)还将控制面板输出数据（对应于控制面板状态灯）送到 OTP（输出到面板)。

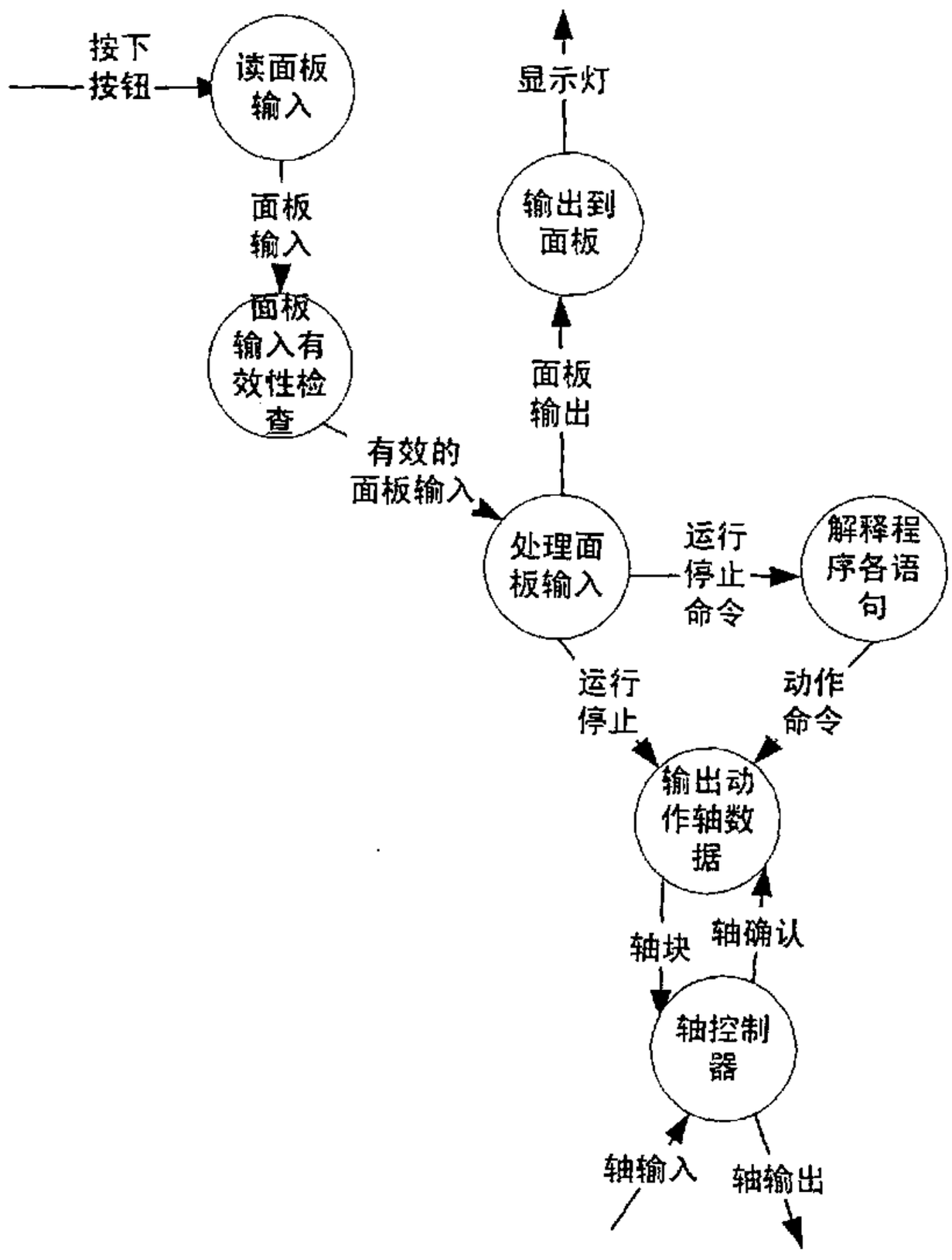


图 6-3 机器人控制器的数据流图

当程序选择开关上的设置改变时，新的开关设置值被传到 PPI（处理面板输入），它将所选程序代号更新。按下“运行”键时，有效控制面板输入产生一个“运行”信号，PPI（处理面板输入）将“运行”的请求传给 IPS（解释程序各语句），然后 IPS 开始解释程序。IPS 直接执行算术和逻辑语句，而动作命令则被传送到 OAD（输出动作轴数据）。OAD（输出动作轴数据）把数据转换成 AC（轴控制器）所需要的格式，并将一个轴块传送给 AC。

按下“停止”键时，OAD（输出动作轴数据）不再向 AC（轴控制器）输送动作轴块。再按下“运行”键，则又恢复传送。当与一个动作轴块相对应的轴动作完成后，AC 发轴确认给 OAD（输出动作轴数据），处理了这个确认后，才能发下一个轴块。

6.3.2 划分任务

识别出系统的所有功能和它们之间的数据流后，下一步要划分任务。在将一个软件系统分解成并行任务时，主要需考虑的是系统内功能的异步性。分析数据流图中的变换，确定哪些变换可以并行，而哪些变换在本质上是顺序的，通过这种方法，划分出任务：一个变换对应一个任务，或者一个任务包括几个变换。

一个变换是应该成为一个独立的任务，还是应该和其它变换一起组成一个任务，决定的原则如下：

(1). I/O 依赖性

如果变换依赖于 I/O，那么它运行的速度常常受限于与它互操作的 I/O 设备的速度。在这种情况下，变换应成为一个独立的任务。

(2). 功能的时间关键性

具有时间关键性的功能需要以高优先级运行，因而，应成为一个独立的任务。

(3). 计算需求

需要进行大量计算的功能可以作为较低优先级任务运行，消耗 CPU 的剩余时间。

(4).功能内聚

完成的功能紧密相关的变换可以组成一个任务，因为这些功能间的数据通信较多，把它们作为一个个独立的任务会增加系统的开销，反之，把每个变换都作为同一任务中一个独立的模块，不仅保证了模块级的功能内聚，而且保证了任务级的功能内聚。

（5）时间内聚

某些变换完成的功能是在同一时间执行的，这些变换可以组成一个任务，这样，每次任务接收到一个事件，它们都可以执行。

（6）周期执行

一个需要周期执行的变换可以作为一个独立的任务，按一定的时间间隔被激活。

6.3.3 定义任务接口

在数据流图中，接口以数据流和数据存储区的形式存在。下一步进行的是格式化任务间的接口。

(1)任务间的通信模块

a.用于任务间的直接通信，即事件，从一个任务/中断程序直接发到另一个任务,主要用于任务间发送控制信号。发送和接收的函数分别为 eventsend 和 eventreceive。如图 6-4。

图 6-4 事件

b.多个任务和中断程序间共享公用存储区：该区被称为信箱，在 DeltaCore 中也被称为消息队列。发送和接收的函数分别为 send 和 receivef。如图 6-5。

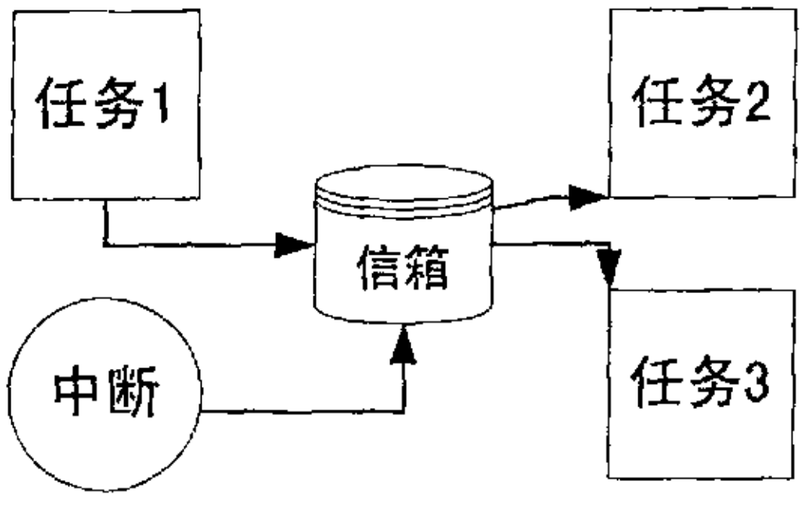


图 6-5 信箱

(2)任务同步和互斥模块

提供信号量机制，用于任务间的同步与互斥， 申请和释放信号量的函数分别为 sunit, runitf。如图 6-6。

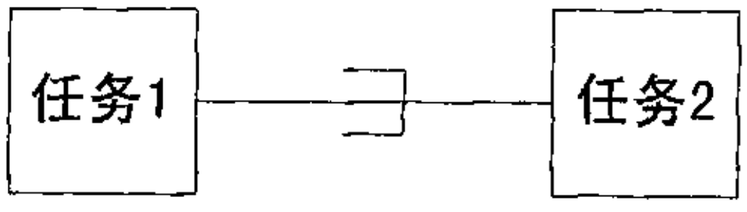


图 6-6 任务间的互斥

6.3.4 实例说明

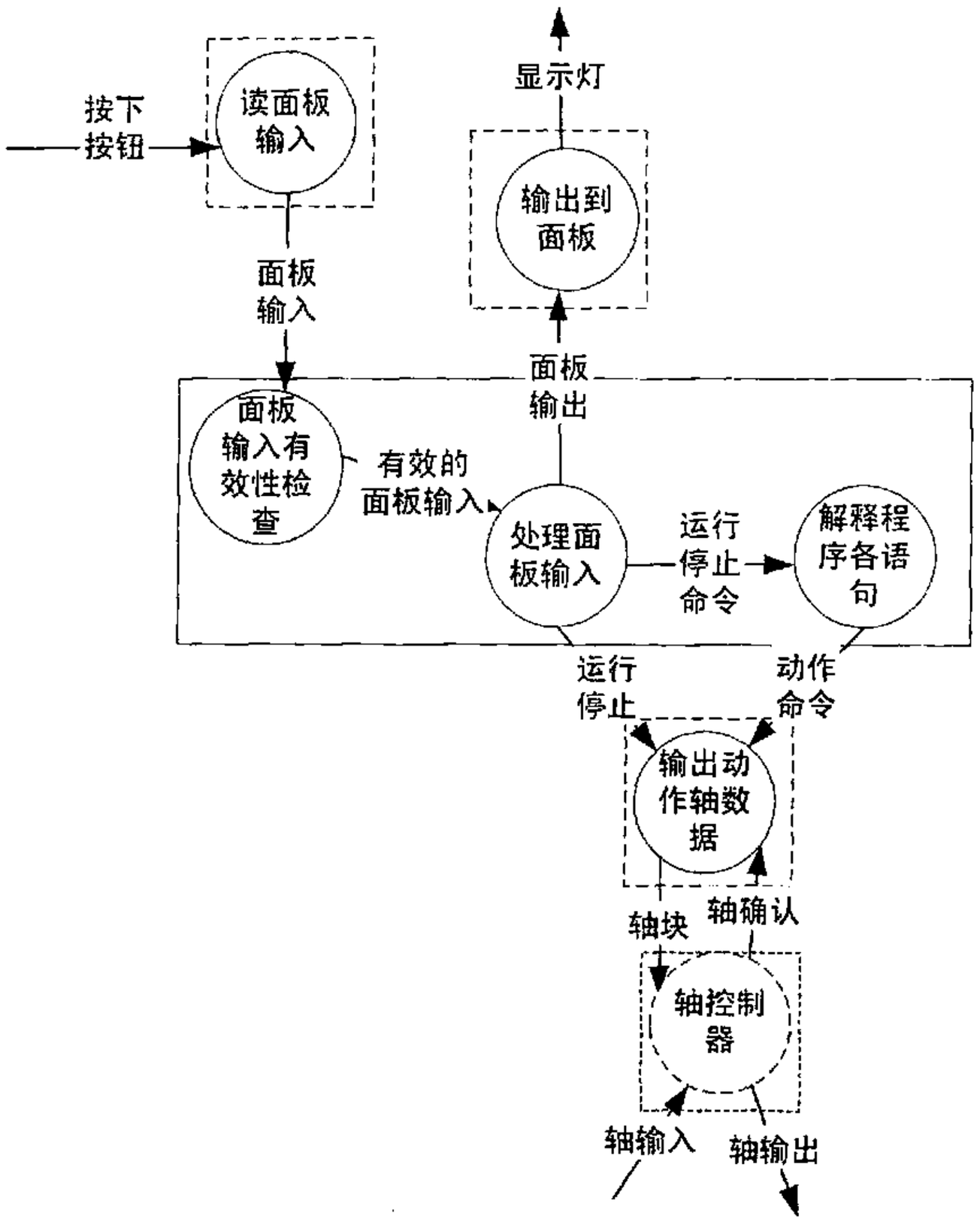


图 6-7 机器人控制器的任务划分

根据上述原则，图 6-7 中用方框框出了各变换或变换组，各方框内的变换在逻辑上构成了一个任务。

任务划分的第一条原则是：直接和 I/O 设备打交道的各功能都应成为独立的任务。因为它的运行速度受制于它互操作的 I/O 设备的速度。而 RPI（读面板输入）从控制面板接收输入，且发生是随机的，故 RPI 变换应作为一个中断-ButtonInt。而 VPI(面板输入有效性检查)、PPI(处理面板输入)、和 IPS（解释程序各语句)，它们完成的功能紧密相关，因此它们组成一个任务，为 PadPro，而 OTP(输出到面板)，控制面板状态灯，也应该成为一个独立的任务-PadOutput。

OAD（输出动作轴数据）也应成为一个独立的任务，ActionM。而 AC（轴控制器）控制动作轴，它运行的速度受限于动作轴设备，所以 AC 也应该成为一个

独立的任务-ControlM。

划分好任务后，下一步要做的是定义任务间接口。

ButtonInt（用户控制中断）将用户从控制面板输入的命令排成一个消息队列，以供 PadPro（面板输入处理）任务接收，因此，这两个任务间的接口是一个消息队列。类似的，PadPro 将处理后的命令排成一个消息队列传给输出动作轴任务 ActionM，同时还将该命令以事件的形式传给 PadOutput（面板灯控制任务）。

而输出动作轴任务 ActionM 将从消息队列中接收命令，再作处理，以事件的形式传给动作轴控制器任务 ControlM。由于动作轴设备是机械设备，速度慢，所以 ActionM 和 ControlM 还存在同步关系，即必须等待 AC（轴控制器）控制动作轴完成当前个动作后，OAD（输出动作轴数据）才能够输出下一个轴块。这两个任务间的接口包括事件和信号量机制。

如图 6-8 是机器人控制器的任务结构图。

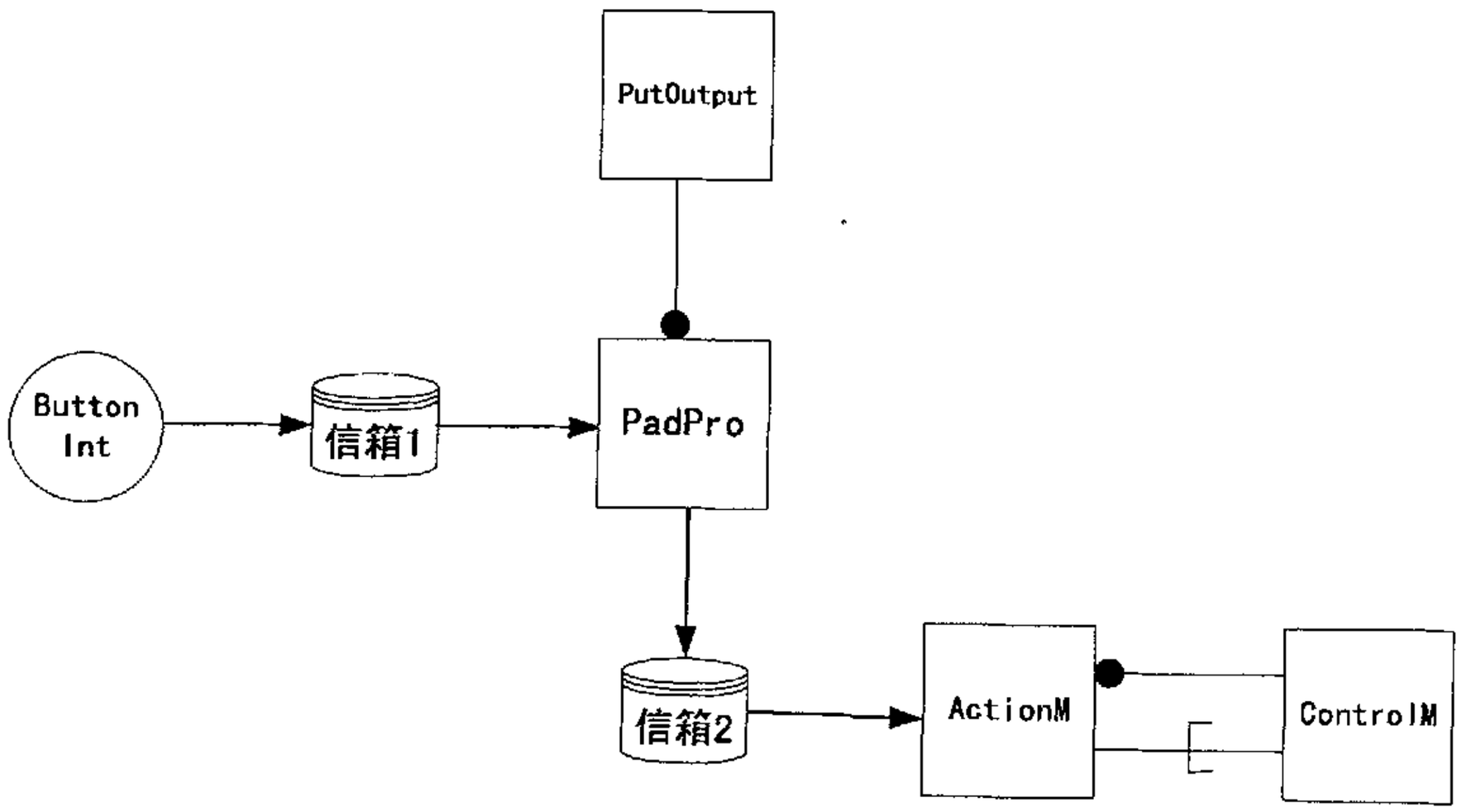


图 6-8 机器人控制器的任务结构图

6.4 系统仿真

6.4.1 软件模型设计图

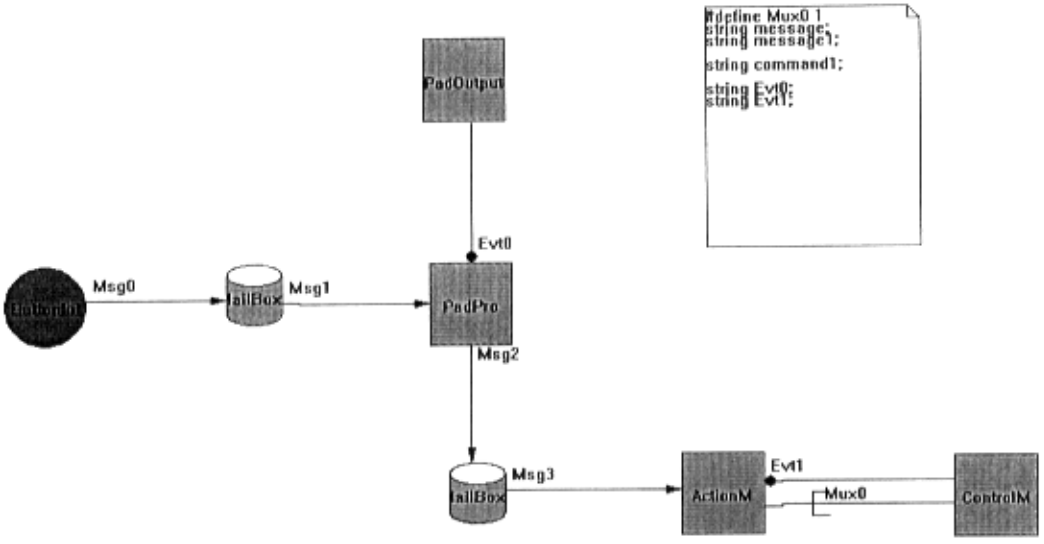


图 6-9 任务结构图

如图 6-9，是使用该仿真环境设计的机器人控制器的任务结构图，基本相同于图 6-8。分别双击中断或是任务的图标，就可以输入该中断或是任务对应的程序结构图。

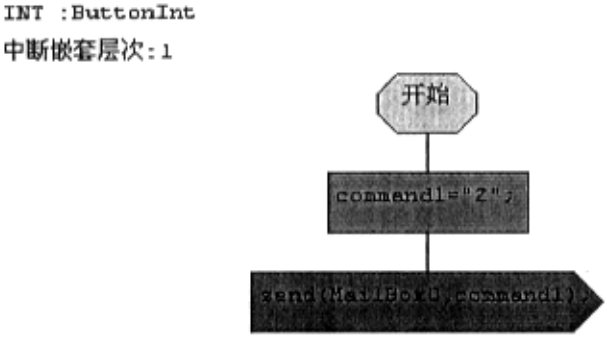


图 6-10 ButtonInt 的中断结构图

如图 6-10,是 ButtonInt（用户控制中断）的中断程序结构图，中断时间设置为 100 毫秒，代表 ButtonInt 的中断流程，表示向消息队列（信箱）1 发送第 2 号命令。

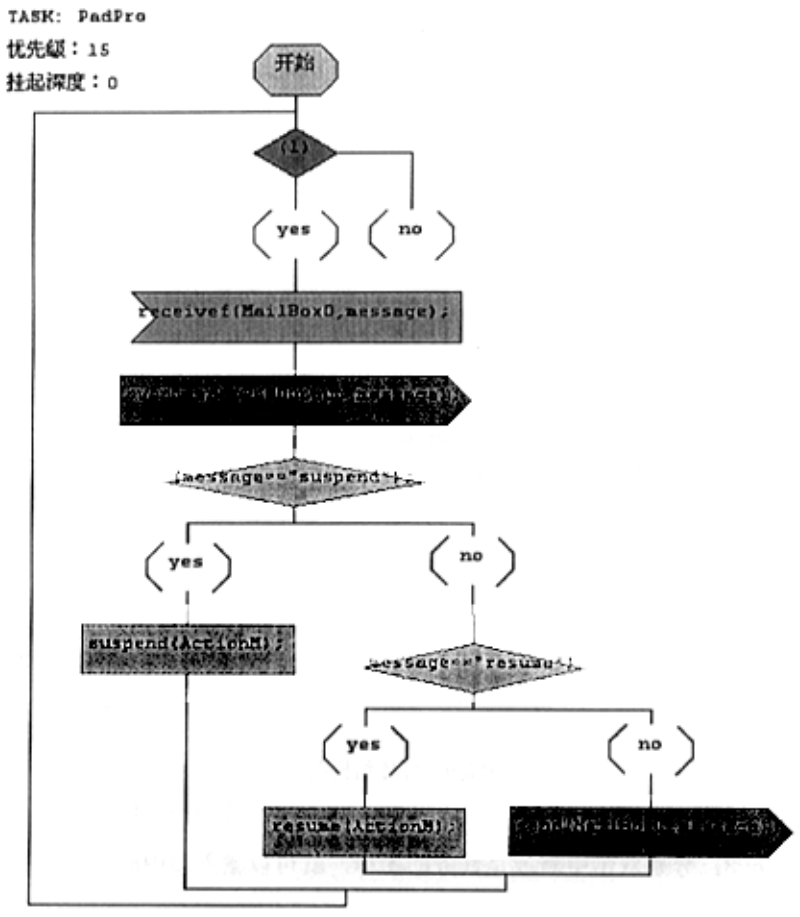


图 6-11　PadPro 的程序结构图

如图 6-11PadPro（面板输入处理）的程序结构图，从消息队列（信箱）1 中接收命令，然后将处理后的命令排成一个消息队列传给输出动作轴任务 ActionM，同时还将该命令以事件的形式传给 PadOutput。它的优先级最高。

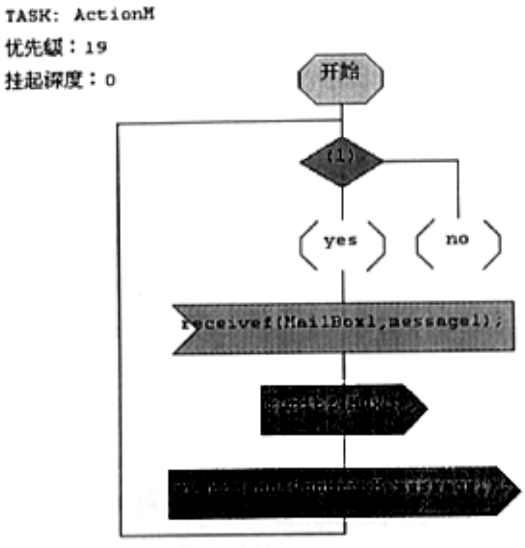


图 6-12　　ActionM 的程序结构图

如图 6-12 是输出动作轴任务 ActionM 的程序结构图，它从消息队列（信箱）2 中接收命令，申请信号量，得到该信号量后，再把该命令以事件的形式传给动作轴控制器 ControlM。

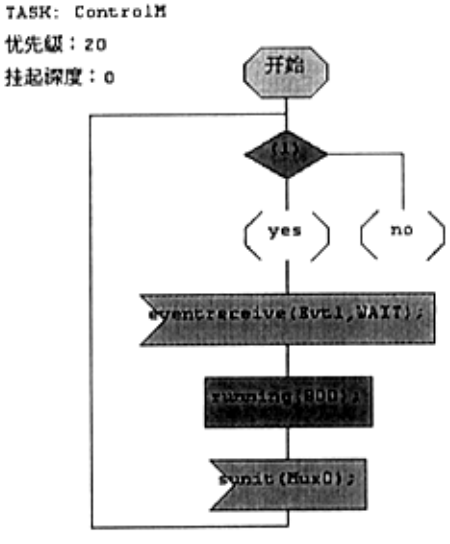


图 6-13　　ControlM 的程序结构图

如图 6-13，动作轴控制器任务 ControlM 接收命令，然后进行 I/O 处理，控制动作轴完成该命令，在这里表示为一个时间块，然后释放该信号量，即允

许进行下一步的动作。

如图 6-14，是 PadOutput（面板灯控制任务）的程序结构图。它接收命令，然后控制状态灯，显示正确的状态。

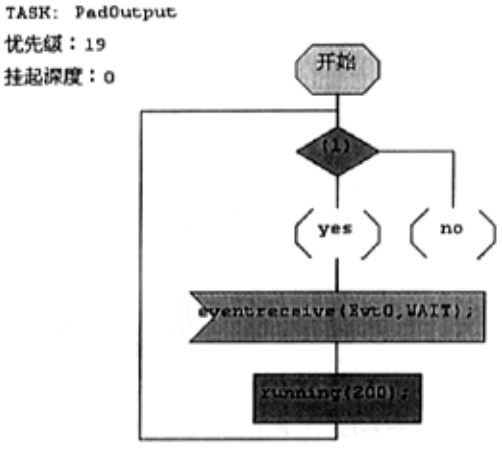


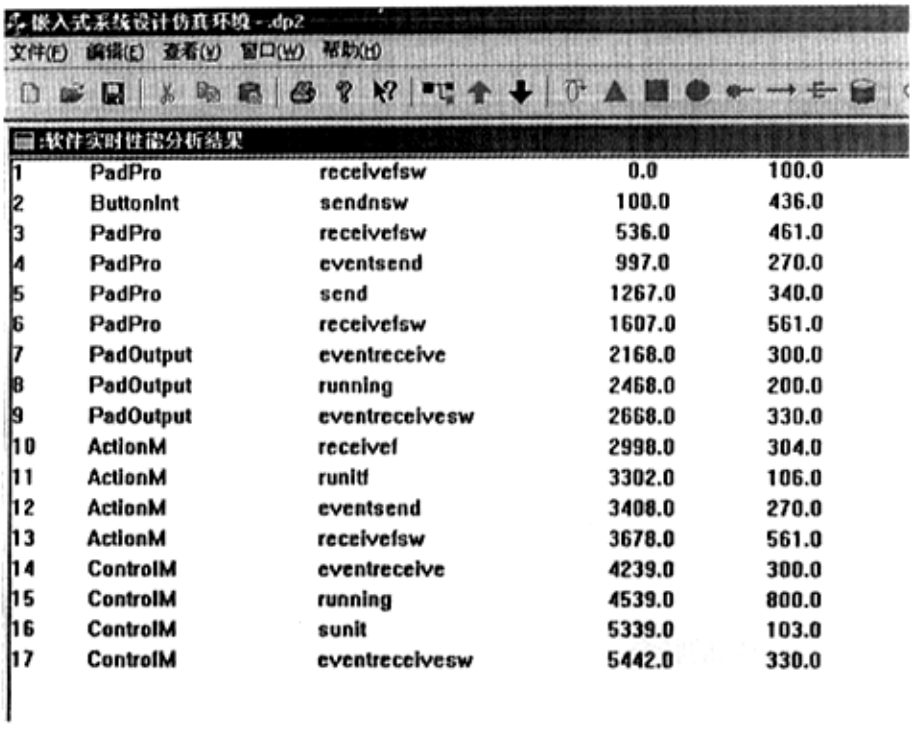
图 6-14 PadOutput 的程序结构图

6.4.2 模拟运行分析结果

完成软件模型的设计后，选择实时性能评价，仿真环境就在后台对该软件模型模拟运行，得到分析结果，并用列表和图形两种方式显示出来。

如图 6-15,是列表方式表示的实时性能分析结果，从左到右，每列的内容依次是事件序号、发生该事件的任务名、事件名、事件的发生时刻和事件的持续时间，可以一目了然的了解到任务状态的变化，任务的调度，所执行的事件以及事件的执行时间，可以对实时性能有清晰的认识，以便根据实时性的需要作出调整，使之能够满足要求。

同时，仿真环境还将上述的实时性能分析结果表示为图形方式，如图 6-16，横轴是时间轴，纵轴是任务、中断轴，在不同的时间段上以方块表示相应的任务或中断在执行，方块的不同颜色代表了不同的事件类型，这种方式非常的清晰和直观。



1	PadPro	receivefsw	0.0	100.0
2	ButtonInt	sendnsw	100.0	436.0
3	PadPro	receivefsw	536.0	461.0
4	PadPro	eventsend	997.0	270.0
5	PadPro	send	1267.0	340.0
6	PadPro	receivefsw	1607.0	561.0
7	PadOutput	eventreceive	2168.0	300.0
8	PadOutput	running	2468.0	200.0
9	PadOutput	eventreceivesw	2668.0	330.0
10	ActionM	receivef	2998.0	304.0
11	ActionM	runitf	3302.0	106.0
12	ActionM	eventsend	3408.0	270.0
13	ActionM	receivefsw	3678.0	561.0
14	ControlM	eventreceive	4239.0	300.0
15	ControlM	running	4539.0	800.0
16	ControlM	sunit	5339.0	103.0
17	ControlM	eventreceivesw	5442.0	330.0

图 6-15 列表方式

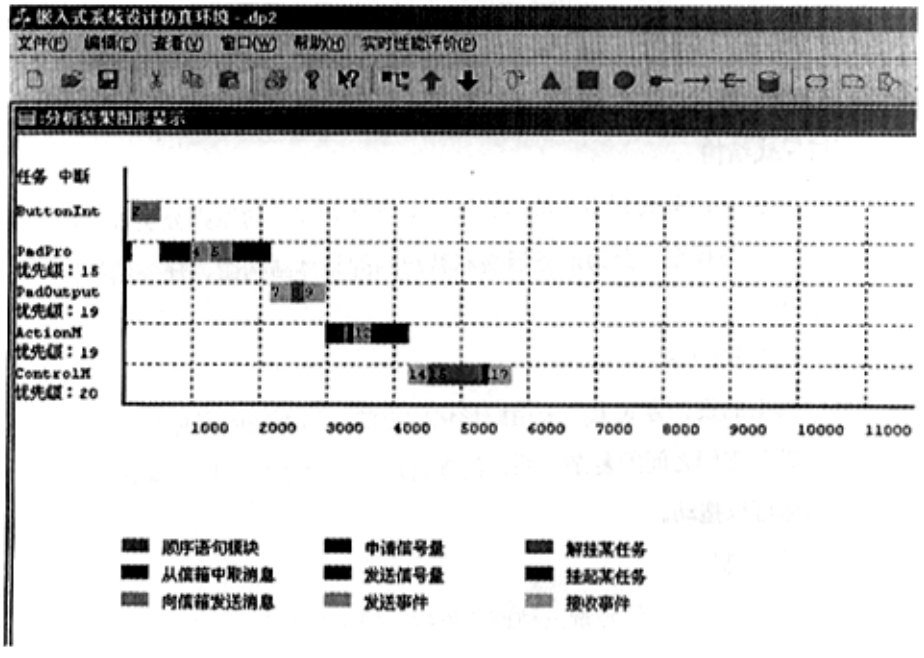


图 6-16 图形方式

第七章 全文总结

7.1 嵌入式应用系统设计仿真环境的应用

在实时嵌入式应用系统的研发中，如何设计嵌入式应用系统的前期软件模型是一个很重要的问题，包括任务的划分，任务优先级的设置，任务间的通信接口等等，都会影响到嵌入式应用系统最终的实时性能，而嵌入式应用系统设计仿真环境提供了这样一个帮助设计嵌入式应用系统的前期软件模型和分析其实时性能的工具，并且它还提供了对 DeltaCORBA 机制的支持，所以它在嵌入式领域有广泛的应用。

本项目是科银京成公司为某国防电子研究所研制开发的，即将应用于该研究所嵌入式应用系统的设计中。

7.2 仿真环境的特色

嵌入式应用系统的辅助设计软件很少见，本仿真环境不仅仅帮助嵌入式应用系统的研发人员设计嵌入式应用系统的前期软件模型，还可以分析该软件模型最终的实时性能，从而缩短嵌入式应用系统的设计周期，提高研发的效率。它具有以下的特色：

1. 三层模式结构

按照从上到下，从大到小，从粗到细，从整体到部分的原则分层设计嵌入式应用系统的软件模型，分为服务器端和客户端的整体结构图、任务结构图、和程序结构图。

2. 图形化的设计方式

在软件模型的设计方式上，采用图形化的方法，用不同的图标和线条来表示不同结构以及它们之间的复杂关系，清晰直观、一目了然，用户操作也很简便。并且图标可以拖动。

3. 多种通信机制

提供了事件、消息、信号量三种通信机制，用于任务间（中断）信息的传递、以及任务间的同步和互斥。

4．不同模式结构图之间在设计上的自动关联

较低层次的模式结构图会根据较高层次模式结构图的内容自动生成一些相应的设计元素，如在任务结构图中的某任务增加了一条到消息队列的消息，则该任务的程序结构图中，就会生成一个表示消息发送的流程图标，用户只需拖动它，把它加入到流程图中，这样减少了设计的工作量。

5．提供了对 DeltaCORBA 机制的支持

提供服务器端和客户端的模式，服务器端提供服务和客户端请求服务的机制来实现对模拟 DeltaCORBA 的支持。

6．提供中断机制，设置中断发生时间

在中断结构图中，同样可以设置中断程序流程图。

7．提供全局变量和局部变量两种方式

在任务结构图中可以设置全局变量表，其中的变量可以作用于任务结构图中定义的所有任务，在程序结构图中可以设置局部变量表，其中的变量仅作用于对应的任务。

8．可以对软件模型模拟运行

可以用户所设计的图形化的软件模型进行编译，并模拟运行，得到按时间顺序排列的事件序列。

9．对模拟运行的结果以图形化的方式显示

对模拟运行的结果除了能以列表的方式表示出来外，还能以坐标轴的图形方式表示出来，清晰直观。

7.3 后续研究内容

本仿真环境是 1.0 版，存在不足和需要进一步完善的地方，同时还需要继续增加功能，最终形成一个功能强大的嵌入式应用系统辅助设计工具。

1．增加代码生成功能，在通过本仿真环境设计好嵌入式应用系统的前期软件模型后，就能根据该模型自动生成一部分的嵌入式应用系统的源代码。

2．取消全局变量表和局部变量表，可直接在流程图标的函数中使用变量。

3．进一步丰富可以模拟的嵌入式系统的系统调用，如增加 delta_task_wake_afer。

4. . 对模拟运行的结果增加数据流向图的表示方式，可以用来表示数据流向。

致　谢

本文的最后我要向我的家人、师长和同学衷心地道一声感谢。我所取得的成绩离不开他们的大力支持和帮助。

导师王忠仁教授广博的知识、开阔的思路、严谨的治学态度和对计算机事业的热情与执着深深地感染着我们每一个人。这些年来，我在学业和事业上所能取得的进步和成绩离不开王忠仁教授对我的信赖、鼓励和支持。本文的初稿也得到了王导认真地审阅，并且提出了很好的建议。在此，我谨向导师王忠仁教授表示我衷心的感谢。

我的父母也在我的这段求学期间默默的支持、帮助、关心和鼓励我，无论是在经济上，还是在精神上，他们无私的爱是我前进的巨大动力，在这里，我衷心感谢，一定要用日后的努力工作来回报他们对我的爱。

并且我还要感谢科银京成公司的副总经理罗蕾教授，和系统部的尹立孟、周小虎、杨伟，感谢他们的帮助，感谢科银京成公司给了我实习的机会。

最后，衷心感谢为评阅本论文而付出辛勤劳动的各位专家和学者！

祝北京科银京成有限公司再创佳绩！

参考文献

[1] 陈丽蓉 。多任务软件实时性能分析和评价技术的研究。成都电子科技大学硕士论文，1998

[2] [美]Michi Henning,Steve Vinoski 著，徐金梧，徐科等译。基于 C++ CORBA 高级编程。清华大学出版社，2000

[3] [美]William Stallings 著，魏迎梅，王涌等译。操作系统—内核与设计原理。电子工业出版社，2001

[4] 陈火旺，钱家骅，孙永强。编译原理。国防工业出版社。1983

[5] [美]Jean J. Labrosse 著，袁勤勇等译。嵌入式系统构件。机械工业出版社，2002

[6] Barr Michael, David Kalinsky. Priority Inversion. Embedded Systems Programming, 2002(4)

[7] [美]William Stallings 著，魏迎梅，王涌等译。操作系统—内核与设计原理。电子工业出版社，2001

[8] 孙祥营 等著。嵌入式实时操作系统 VxWorks 及其开发环境 Tornado. 中国电力出版社，2001

[9] [美]Davis Chapman 著 ,骆长乐 译。学用 Visusal C++ 6.0。清华大学出版社，1998

[10] 龚天富 侯文永。程序设计语言与编译。电子工业出版社。1996

[11] 李江、常葆林。嵌入式操作系统设计中的若干问题。微型机与应用，2000.8

[12] 电子科技大学嵌入式软件中心—北京麦克泰,《VRTX 嵌入式实时操作系统培训教材》,1998.11

[13] 汤子瀛、哲凤屏、汤小丹，《计算机操作系统》，西安：西安电子科技大学出版社，1998

[14] 共创软件联盟，"嵌入式操作系统市场分析及技术发展",计算机世界 2000.8.28

[15] 王飞跃、吴朝晖，"ASOS：嵌入式操作系统的发展趋势",计算机世界 2000.11.20

[16] Intel Corporation,"ARM Architecture Reference Manual", from on-line, http://developer.intel.com, 2000

[17] Barr Michael, David Stewart. Rate Monotonic Scheduling. Embedded Systems Programming, 2002(3)

[18] Intergrated systems Inc,《pSOSystem System Concepts》, 1996.

[19] Wind River, 《Vxworks Programmer's Guide 5.4 Edition 1》, 1999.5

[20] Wind River, 《Vxworks Reference Manual 5.4 Edition 1》, 1999.5