北京交通大学

硕士学位论文

单片机嵌入式操作系统研究与NoC结构的操作系统内核设计

姓名：刘彦鑫

申请学位级别：硕士

专业：微电子学与固体电子学

指导教师：李哲英

201012

中文摘要

摘要：嵌入式操作系统是嵌入式应用软件的基础和开发平台，用户的其他应用程序都建立在嵌入式操作系统之上。目前，对于 16 位/32 位处理器的嵌入式操作系统开发比较广泛，8 位处理器的操作系统开发很少。而 8 位处理器的应用软件开发又比较常见，所以本文提出一种基于 8051 单片机的简单嵌入式操作系统，大大减少了软件开发时的重复性劳动。

对于 NoC（Networks on chip 片上网络）结构，需要一个适用于这种结构的操作系统控制管理计算机资源、高效地利用软硬件、合理地组织管理计算机系统的工作流程。一种方法就是拓展以上适用于 8051 单片机的简单嵌入式操作系统，将其作为 NoC 分布式操作系统的一个基础节点。在这个节点操作系统里预留一些接口，通过这些接口实现各节点的连接通信。分布式操作系统每一个节点具有相同结构和功能。

论文对嵌入式操作系统的研究现状和发展趋势作了分析，详细研究了所使用到的基础理论。在此基础之上，设计了一个嵌入式操作系统结构，建立软件模型。这个嵌入式操作系统是一个微内核结构，针对瑞典 KTH 研究小组的 DRRA 片上网络结构，以及对应的 NoC 分布式操作的要求，提供了进程调度和进程间通信功能。为了进一步拓展到 NoC 中应用，操作系统内核中预留了三个接口，希望通过这三个接口完成每个操作系统节点间的数据通信，实现 NoC 上分布式操作系统；最后通过实现一个 FIR 低通滤波器验证这个系统的各项功能都达到了设计要求。

关键词：8051 单片机；嵌入式操作系统；进程调度；NOC 片上网络。

分类号：TP316.2

ABSTRACT

ABSTRACT: The Embedded Operating System (EOS) is the base of the embedded applications. All user applications run on the EOS. Currently, EOS for 16/32-bit microcontrollers are well developed and used widely while only limit number of EOS are for 8-bit microcontrollers. This paper proposes a simple EOS targeting Intel 8051 compatible microcontrollers to reduce the labor cost for user application development.

Nowadays, Network-on-Chip (NoC) technique is being developed rapidly. There is a need of operating system which is, (1) suitable for NoC interconnect architecture, (2) capable for optimally managing computer resources including hardware/software resource, (3) able to optimally organize the application execution flow. The proposed EOS is able to be extended to a distributed operating system to manage a building block microcontroller in the NoC architecture. Interfaces for data communication and process management are specified already in the proposed EOS. By implementing these reserved interfaces, all the building block microcontrollers play equivalently as building blocks for the entire system.

This master's thesis (1) explored the state-of-art and development trend of the research of embedded operating systems, (2) studied the details of the architecture of the Intel 8051 compatible microcontrollers, architecture of operating system and NoC architectures, (3) developed the structure of an embedded operating system and (4) built up user application programming model. The proposed operating system is a micro kernel operating system. For the NoC architecture of DRRA designed by a research group in KTH, the usages of it are process scheduling and inter-process communication. In order to extend this EOS to fit into NoC architecture, three interfaces are reserved. They provide inter-node (microcontroller) (1) data communication, (2) process management (creation and termination). A FIR filter is implemented as a user application to evaluate the proposed EOS. The experimental results show the EOS satisfies the design specifications.

KEYWORDS： 8051 Microcontroller; Embedded Operating System; Process Scheduling; Network-on-Chip.

CLASSNO： TP316.2

致谢

本论文的工作是在我的导师李哲英教授的悉心指导下完成的，李哲英教授严谨的治学态度和科学的工作方法给了我极大的帮助和影响。在此衷心感谢两年半以来李哲英老师对我的关心和指导。

李硕博士悉心指导我完成了实验室的科研工作，在学习上和生活上都给予了我很大的关心和帮助，在此向李硕表示衷心的谢意。

在实验室工作及撰写论文期间，阮孝利、李宁师兄、刘佳师姐等同学对我论文中的嵌入式操作系统研究工作给予了热情帮助，在此向他们表达我的感激之情。

另外也感谢我的家人，他们的理解和支持使我能够在学校专心完成我的学业。

1 绪论

1.1 引言

随着信息技术高度发展，计算机进入了一个充满机遇和挑战的时代。工业界对智能控制要求不断提高，形势多样的数字化智能产品应运而生。这些智能产品成为替代通用 PC 机进行信息处理的主要部件，它们都嵌入了微处理器，也就成为嵌入式系统。嵌入式系统已经融入到人们的生活之中，远远超过了通用 PC 机的应用领域。嵌入式系统带来的工业年产值在 IT 产业中占有很大的比重，具有很大研究意义和非常广阔的应用前景。

早期的嵌入式系统采用4位或8位的微处理器,存储器容量一般是在10kB以下,没有较宽余的用于存放操作系统的空间,系统中运行的是用户自行编制的专用程序,一般在一个循环结构的程序模块中对外界请求进行处理,对一些简单的系统而言,这种系统开发手段尚能应付。随着嵌入式系统规模的扩大及复杂程度的增加，不仅要求软件提供更加丰富的功能，还要求尽量短的开发周期。因此应该在嵌入式系统中嵌入一个操作系统，在此操作系统之上再构造用户控制程序，以满足嵌入式系统的开发与应用需求。从嵌入式系统软件结构的角度来看，嵌入式操作系统内部有一个实时多任务内核，将 CPU 时间片、中断、I/O、定时器等资源都封装起来，留给用户一个标准应用程序接口 API，并根据应用程序中各个任务的优先级，在不同任务之间合理地分配 CPU 时间[1]。目前，对于很多大型机 16 位、32 位机的嵌入式操作系统开发较为广泛，例如，常见的嵌入式系统 Linux、uClinux、WinCE、uCOS-II、VxWorks。简单 8 位单片机应用较多，而 8 位机的操作系统开发很少，这样使程序员工作繁琐，在软件设计中需要软件工程师从底层做起，在系统软件设计方面需要做大量的重复性劳动。51 系列单片机在我国的应用非常广泛，它是美国 INTEL 公司在 1980 年推出的高性能 8 位单片。如果开发一套基于 51 系列单片机的操作系统，那么用户只需要编写各个任务的程序，不必同时将所有任务运行的各种情况记在心中，不但大大减少了程序编写的工作量，而且减少了出错的可能性[2]。基于这些原因，本文研究提出一种适用于 8051 单片机的嵌入式操作系统 EOS（Embedded Operating System）。

半导体技术飞速发展，单个芯片上所集成的处理器数量大幅度增加，未来的处理器将会形成一个单芯片上的分布式系统。然而，随着芯片的集成度的大幅度增加，芯片的复杂度也随之大幅度增加，芯片上处理器之间的连接和通信的难度

急剧上升。片上物理连线长度的增加导致信号在线上传播的时间远大于处理器的时钟周期，这样给连线两端的处理器之间的同步通信带来了很大的问题；另一方面，连线长度增加也导致了信号传送的不确定性问题。所以 NOC(Networks on chip，片上网络)被提出来，NoC 是一种低功耗的网络化结构的片上通信模式，用以取代传统的片上总线的通信方式[3][4]。

NoC 的研究涉及到了诸多方面，从物理设计到体系结构、操作系统、应用服务、设计方法和工具。全新的 NoC 集成电路体系结构展示了优于总线结构的本质和属性，从根本上解决了总线结构的固有问题。目前对于 NoC 硬件平台的研究已经有一定基础，但是，对这种芯片的软件模型和操作系统的研究涉及较少，随着 NoC 技术的发展，需要一个适用于这种结构的操作系统控制管理计算机资源、高效地利用软硬件、合理地组织管理计算机系统的工作流程[5]。已做过的相关工作有[6]，研究了基于多核的嵌入式操作系统，并提出了多核同构与异构的嵌入式系统和编程模型，文献[7][8][9]则构造了多种多核嵌入式系统。本文提出一种基于 8051 单片机的简单嵌入式操作系统，并考虑将这个操作系统拓展成为 NoC 片上分布式操作系统。作者首先提出适用于单核的操作系统结构，并对其进行验证。验证通过后考虑拓展这个操作系统到多个 C51 核构成的 NoC 中。这个简单嵌入式操作系统只是作为这分布式操作系统的一个基础节点，分布式操作系统每一个节点具有相同结构和功能。在这个操作系统里预留一些接口，通过这些接口实现各节点的通信。从而使具有分布式的 NoC 结构中的每个 CPU 都使用相同的操作系统。

1.2 国内外现状及发展趋势

目前，世界各国有 40 多家公司成功推出了 200 余种可供嵌入式应用的实时操作系统。其中著名的操作系统是：WindRiver 公司的 VxWorks，Integrated System 公司（ISI）的 pSOS，Lynx Real-Time Systems 公司的 Lyns 实时操作系统等。这些操作系统适用于多任务、强实时性的应用环境，而且还具有相应的功能齐全的交叉环境，如 WindRiver 公司的 Tornado，MRI 公司的 XRAY。

VxWorks 是 WindRiver 公司开发的具有工业领导地位的高性能实时操作系统内核，具有先进的网络功能。VxWorks 对工业标准的支持和开放式结构，使得开发人员易于设计高效的嵌入式系统，并可以简单地移植到其他的处理器上。

Lynx Real-Time Systems 股份有限公司主要开发基于微处理器的高技术产品，Lynx 的基础产品是 Lynx 嵌入式操作系统，并提供一整套嵌入式软件产品和相关服务。Lynx OS 具有 Unix 的全部优点，即真正的实时内核，程序可重入和调度确定性[10]。

国内的嵌入式操作系统到 20 世纪 80 年代得到蓬勃发展，已有多种商业操作系统可供选择，如 Delta OS、桑夏 2000 操作系统、红旗 Linux、中软 Linux 和“女娲 Hopen”等。

在中国，嵌入式操作系统可分为两大类型 ：一类是自主版权的操作系统,另一类是基于 Linux 的操作系统。自主版权的操作系统有桑夏 2000 操作系统、“女娲 Hopen”操作系统和 Delta OS 操作系统等。桑夏 2000 操作系统是一个面向嵌入式应用的实时操作系统，具备文件系统和嵌入式数据库引擎，提供了基本的图形用户接口，支持层次化、模块化的软件模型，支持包括 TCP/IP 协议在内的网络通讯协议。“女娲 Hopen”支持所有主流的嵌入式芯片。Delta OS 主要包括：内核 Delta CORE、嵌入式图形接口 Delta GUI、嵌入式文件系统 Delta FILE 以及嵌入式 TCP/IP Delta NET。Delta OS 支持 ARM7、PPC8 XX、Strong ARM、PPC4 XX、MIPS、X86 等多种嵌入式微处理器，可应用于消费电子产品、通信产品、工业控制及军用电子产品中。

近年来，中国的嵌入式 Linux 进展较快，以 Linux 为基础的嵌入式操作系统比较活跃。例如中软 Linux、 红旗 Linux 等。中软嵌入式 Linux 是实时系统，这套基于 Linux 的嵌八式系统不仅满足了数控机床的需求，同时也可以应用于其他工业控制领域。红旗 Linux 把工控和信息家电作为主要的发展领域。红旗 Linux 为用户提供了 Windows 风格的控件集、图形中文环境和嵌入式数据库的开发工具。

嵌入式操作系统种类繁多，有资源开销很少的 IC 卡操作系统，也有功能较强的网络计算机操作系统。这些嵌入式操作系统从功能到性能各具特色，为各种硬件环境及应用提供了相应的支持和服务。目前市场上共有几十种操作系统。有实时系统，也有分时系统；有深嵌入式系统，也有浅嵌入式系统；有多任务系统，也有单任务系统。总之，不同厂商推出的嵌入式操作系统多少都有些差异，但许多嵌入式操作系统普遍具有如下一些内容和功能：系统核心、存储管理、网络协议、设备驱动程序和文件系统等。现在大部分嵌入式操作系统都有一个多任务内核，以满足多种嵌入式系统的需要。各种嵌入式系统的内核功能差别较大，但即使是微内核系统，系统核心也需要完成一些基本任务：进程间通讯、中断、线程调度和内存管理等[11]。

1.3 论文研究工作和组织结构

本文的任务，是瑞典皇家里工学院（KTH）电子系统系 NOC 研究小组工作的一个组成部分，基本任务要求作者完成适合于 8051 体系单片机的微型操作系统，并对以此类单片机 CPU 构成的 NOC 系统的操作预留相应的接口。

本文的具体任务是：

深入学习研究 8051 单片机结构以及嵌入式操作系统基本原理；

设计一个适用于 8051 单片机操作系统的内核结构，提出其软件模型，对其进行仿真分析及验证，检验其各项功能是否达到设计要求；

为了研究适用于 NoC 硬件平台的操作系统，拓展这个嵌入式操作系统作为 NoC 分布式操作系统的一个基础节点。

作者在项目直接负责人 Ahmed Hemani 教授的指导下，在瑞典皇家理工学院完成这个操作系统设计，全部实现了设计要求的功能，并顺利通过瑞典皇家工学院的项目验收论文答辩。

本文完成的操作系统主要是为应用于瑞典皇家理工学院即将开发的硬件平台而设计，该硬件平台由多个 8051 微处理器构成，是一个 NOC 结构多核处理器。它需要分布式嵌入式操作系统来控制管理多核处理器资源、高效利用软硬件、合理组织处理器工作流程。本设计中的操作系统作为这一分布式系统的基础节点，是它的重要组成部分。这个基础节点将被拓展成多个节点，其中每一个节点具有同其相同的结构和功能，这样形成分布式操作系统。

本文中的图形绘制全部使用 yed graph editor 专业图形绘制软件。绘图基本流程是：在该软件界面绘制图形；导出图片，导出文件格式有多种选择；以插入图片文件形式添加到 word 文本中。

作者提出一种应用于 8 位单片机的简单操作系统，首先通过深入学习操作系统运行平台 8051 单片机基本结构和操作系统的结构，确定本设计中嵌入式操作系统的基本结构。操作系统内核模型建立后，使用 FIR 低通滤波器对输入激励的响应过程验证操作系统的各项功能。这个操作系统将要应用到 NoC 分布式操作系统中，需要了解片上网络的硬件结构以及通信方式。整个开发过程都使用软件 Keil C51，作者学习掌握软件的使用调试方法。以上是开发这个嵌入式操作系统所必须做的研究工作。

在行文结构上，本文一共分为五章，下面具体就各章内容进行简要的介绍：

第一章（即本章），绪论。主要讲述课题背景，说明该文的研究范围以及国内外发展现状和趋势。

第二章，背景知识。该部分详细介绍开发本嵌入式操作系统所用到的理论知识，其中包括 51 单片机、操作系统以及 NoC 的结构、开发软件和开发中应注意的问题。

第三章，嵌入式操作系统的结构。该部分主要讲这个操作系统的关键技术和设计思想。作者根据学习嵌入式操作系统的理论知识以及实际需要设计操作系统的基本结构，使用 MCS-51 单片机软件 Keil C51 开发。首先建立一个工程项目，

选择芯片，建立 C 源文件，用项目管理器生成各种应用文件，检查并修改源文件中的错误，编译连接通过后进行软件模拟仿真。操作系统软件模型分为四个文件，这一章从这四个文件讲述了操作系统的结构，四个文件是系统的头文件、主函数、任务调度和进程间通信。头文件中包含操作系统需要的所有头文件、变量定义以及函数声明等。主函数包括 PCB 的初始化、创建任务和用户程序。任务调度中包括时间片设置、将任务按优先级排序、启动任务、调度任务和删除任务。进程间通信包括发送消息、查询消息和接收消息。

第四章，操作系统的验证。作者实现一个简单的 10 阶 FIR 低通滤波器，给其输入激励得到冲激响应，通过这个实现过程验证嵌入式操作系统的各项功能，检验操作系统是否达到设计要求。

第五章，总结与展望。叙述该系统的改进之处，并总结全文，指出目前该系统所存在的不足和待完善之处。

2 背景知识

2.1 8051 单片机结构

8051 微控制器是 MCS-51 家族的成员，在 20 世纪 80 年代由 Inter 最初研发。此后，8051 获得了很大的流行并且被认为在嵌入式系统产品中占有很大的使用量。最基础的 8051 核包含了一些片上的周边设备，如计时器/计数器、128 字节的片上数据存储器和 4K 字节的片上程序存储器。8051 的基本结构如图 2.1 所示。

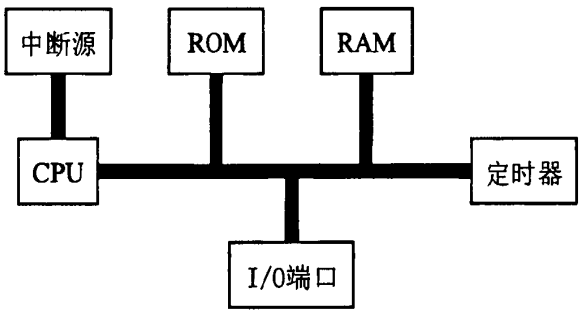

图 2.1　8051 基本结构

Fig.2.1 The basic structure of 8051 core

8051 的单片机特点如下：

a) 为控制应用优化的 8-bit CPU；
b) 广泛的布尔处理能力；
c) 64K 程序存储寻址空间；
d) 64K 数据存储寻址空间；
e) 多达 4K 的片上程序存储空间；
f) 128 bytes 的片上数据存储空间；
g) 32-bit 双向单独寻址 I/O 线；
h) 两个 16-bit 计时器/计数器；
i) 两级 6-源/5-向量中断结构。

2.1.1 8051 单片机存储区

8051 单片机存储器包括程序存储器 ROM 和数据存储器 RAM。由于单片机寻址空间是 16 位，所以它有 64K 的数据寻址空间和 64K 的程序寻址空间。8051 单

片机的存储区对于程序存储和数据存储有各自的寻址空间。这种存储区的逻辑分隔是有好处的，它允许数据存储区被 8 位地址总线寻址，这样数据存储器就可以被 8 位 CPU 快速访问和操作。需要 16 位数据存储器寻址时，可以由 DPTR（数据指针）寄存器产生。

程序存储器只能被读取，不能被写入。8051 的程序寻址空间是 16 位的，所以最大的存储空间是 64K。4K 是片上存储器，即 8051 核的内部程序存储器。为了访问外部程序存储器，8051 使用了 PSEN 寄存器。当寄存器 PSEN 的位“ea”为 1 时，选通内部程序存储器，当 ea=0 选通外部程序存储器。数据存储是独立于程序存储器的一块寻址空间。对于访问外部数存，CPU 根据需要产生读写信号 RD 和 WR。8051 核的存储结构包括了 128 字节的片上数存，这些空间由指令更容易直接被访问，并且还有一些 SFR（Special Function Register 特殊功能寄存器）。内部数据存储器包含了 4 个库(每个库中有 8 个寄存器)和一个特殊的 32 字节的长区间，这个区间中的每一位都可以由一个特殊的子 8051 指令集寻址，位操作指令。多达 64k 字节的外部数据存储器可以由“movx”指令访问。图 2.2 显示了 128 bytes 的片上数据存储器，其中有 4 个包含寄存器和可位寻址区的库。

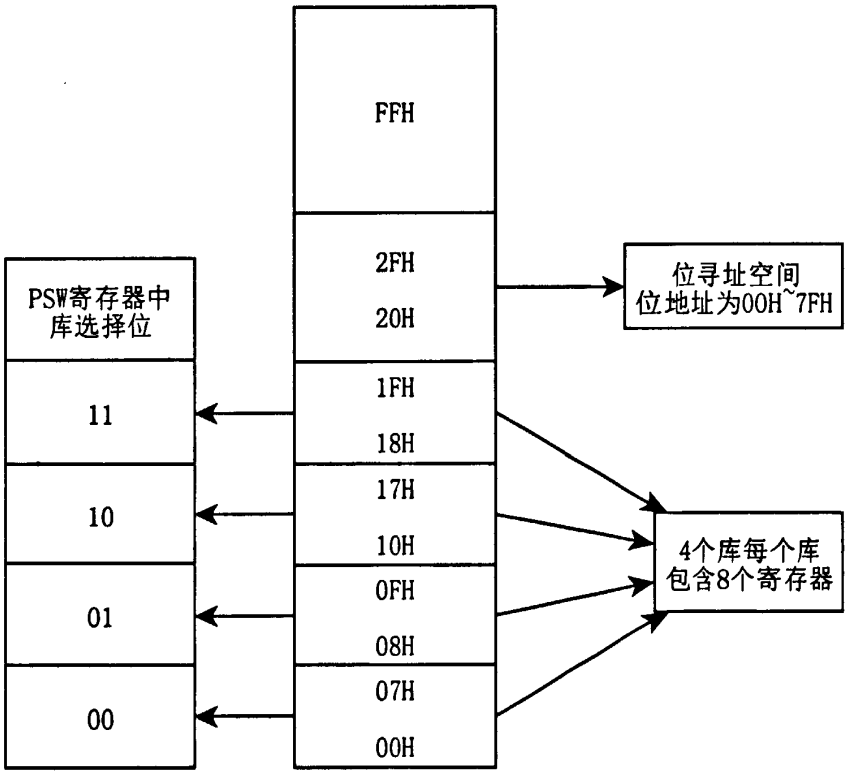

图 2.2　片上数据存储器 128 字节

Fig.2.2 Internal data memory, 128k Bytes

外部程序存储器和数据存储器可以结合起来同时使用。外部程序存储器和数据存储器的选通脉冲是由 RD 和 PSEN 信号组成的，它们必须输入到与门中，这个与门的输出作为读选通信号。

2.1.2 特殊功能寄存器

8051 具有一些特殊功能寄存器 SFRs。表 2.1 中包含了所有的特殊功能寄存器和它们的地址。从表中可以看出只有少量的地址被占用，如果对没有被使用的地址进行访问，将会得到没有被定义的数据，写这些地址时，不会起作用。

表 2.1　寄存器地址

Table 2.1 The address of registers

f8h									ffh
f0h	b								f7h
e8h									efh
e0h	acc								e7h
d8h									dfh
d0h	psw								d7h
c8h									cfh
c0h									c7h
b8h	ip								bfh
b0h	p3								b7h
a8h	ie								afh
a0h	p2								a7h
98h	scon	sbuf							9fh
90h	p1								97h
88h	tcon	tmod	tl0	tl1	th0	th1			8fh
80h	p0	sp	dpl	dph				pcon	87h

有些特殊功能寄存器可以有它们对于不同函数分配的位。这些寄存器是 PSW, IE, IP, TCON 和 TMOD。在下面的部分里有关于这些 SFR 的简单介绍。

a) 程序状态字 (PSW)

程序状态字寄存器包含了一些反映 CPU 当前状态的状态位。

CY	AC	F0	RS1	RS0	OV	-	P

CY:　进位标志位；

AC:　辅助进位标志位；

F0:　可以被用户自定义用途的0标志位；

RS1:　寄存器库选择位1 ；

RS0:　寄存器库选择位0；

OV:　溢出标志位；

一:　用户定义标志位；

P:　奇偶标志位，在每个指令周期由硬件置1或清0来指示加法器中1的个数是奇数还是偶数。

RS1和RS0位用来选择相应的寄存器库，如表2.2。

表 2.2　寄存器区选择

Table 2.2 Choose register bank

RS1	RS0	Register Bank	Address
0	0	0	00H-07H
0	1	1	08H-0FH
1	0	2	10H-17H
1	1	3	18H-1FH

b)　中断处理寄存器

IE 和 IP 寄存器被用来管理和控制中断。IE 寄存器被用来使能中断，IP 被用来设置中断优先级。

中断使能寄存器 (IE)

EA	-	-	-	ET1	EX1	ET0	EX0

EA:　屏蔽所有中断。如果EA=0, 没有中断被响应。如果EA=1,每个中断源由各个使能位分别管理。

ET1:　使能或屏蔽计时器1的溢出中断（1为响应的中断被使能，0为屏蔽）；

EX1:　使能或屏蔽外部中断1（1为响应的中断被使能，0为屏蔽）；

ET0:　使能或屏蔽计时器0的溢出中断（1为响应的中断被使能，0为屏蔽）；

EX0:　使能或屏蔽外部中断0（1为响应的中断被使能，0为屏蔽）。

中断优先级寄存器 (IP)

如果该位是 0，相应的中断有较低的优先级，如果该位是 1，相应的中断有较高的优先级。

-	-	-	-	PT1	PX1	PT0	PX0

PT1:　定义计时器1的中断优先级；

PX1:　定义外部中断1的中断优先级；

PT0:　定义计时器0的中断优先级；

PX0:　定义外部中断0的中断优先级。

c)　计时器/计数器控制寄存器

寄存器 TCON 和 TMOD 是用来与计时器/计数器一起工作的。这两个寄存器的作用是控制计时器/计数器和变换它们的工作模式。

计时器/计数器控制寄存器 (TCON)

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1:　计时器1溢出标志位。当计时器1溢出时由硬件置1。当处理器处理中断响应时由硬件置0;

TR1:　计时器1运行控制位。由软件控制置1或置0;

TF0:　计时器0溢出标志位。当计时器1溢出时由硬件置1。当处理器处理中断响应时由硬件置0;

TR0:　计时器0运行控制位。由软件控制置1或置0;

IE1:　外部中断1边沿标志位。当外部中断跳变沿被检测到时由硬件置1;

IT1:　中断1类型控制位。由软件指定是上升沿还是下降沿触发外部中断;

IE0:　外部中断0边沿标志位。当外部中断跳变沿被检测到时由硬件置1;

IT0:　中断0类型控制位。由软件指定是上升沿还是下降沿触发外部中断。

计时器/计数器模式控制寄存器 (TMOD)

TMOD 寄存器的前 4 位是控制和管理计时器 1 的，后 4 位是为了控制和管理计时器 0 的。

GATE	C/T	M1	M0	GATE	C/T	M1	M0

GATE: 当TCON中的TR0（1）被置1并且GATE为1时，计时器0（1）只有在外部中断0（1）为高电平时工作。当 GATE为0时，计时器只有在TR0（1）＝1时才工作;

C/T:　计时器/计数器选择位。为0时选择计时器，1时选择计数器;

M1:　模式选择位;（见表2.3）

M0:　模式选择位。（见表2.3）

表 2.3　计时器模式选择

Table 2.3 Timer/Counter operating mode selection

M1	M0	运行模式
0	0	13位计时器
0	1	16位计时器/计数器
1	0	可自动加载的8位计时器/计数器
1	1	计时器0的TL0和TH0是作为两个分离的计数器

2.1.3 计时器/计数器

8051 单片机有两个 16-bit 计时器/计数器：计时器 0 和计时器 1。它们都可以作为计时器或计数器工作。它们都有 4 种不同的工作模式可供选择。

特殊功能寄存器 TMOD 的控制位 C/T 选择计时功能或计数功能。计时器，计数器都有 4 个运行模式，其中模式 0、1、2 对计时计数功能是一样的，但模式 3 是不一样的。模式的选择是由特殊寄存器 TMOD 中的 M1 和 M0 两位完成的。

1) 模式0

在这种模式下，计时器被配置为一个 13 位的寄存器。以计时器 1 为例，它由 8 位 TH1 和低 5 位 TL1 组成，TL1 的高 3 位是不确定的，可以忽略掉。设置运行标志位(TR1)并且寄存器不清零。计时器中断标志位 TF1 在设置的计数次数到达时被置 1。计时器 0 和计时器 1 在模式 0 运行时是一样的，只是将计时器 1 的控制信号换成计时器 0 的控制信号。

2) 模式 1

模式 1 和模式 0 对于两个计时器是基本相同的。不同的是在模式 1 下，计时器被配置为 16 位寄存器。

3) 模式 2

在模式 2 中，两个计时器寄存器被配置为可以自动加载的 8 位计数器 TL1（TL0）。TL1（TL0）中产生溢出将 TF1（TF0）置 1。同时将由软件预先设置好的 TH1（TH0）中的内容重新加载到 TL1（TL0），TL1（TL0）的重载不会改变 TH1（TH0）的值。

4) 模式 3

在模式 3 条件下，计时器 1 和计时器 0 的运行方式是不一样的。计时器 1 仅保持它的计数，把它当做 TR1 被置 0 时的运行状态。对于计时器 0 则不一样，在模式 3 中计时器 0 的 TL0 和 TH0 是作为两个分离的计数器。TL0 在工作时使用计时器 0 控制位：C/T,GATE,TR0 和 TF0。TH0 被锁定为计时功能（对机器周期计数）并控制了计时器 1 的 TR1 和 TF1。TH0 在这种状态下实际上是计时器 1 的中断。

模式 3 满足了需要额外 8 位计时或计数的应用，工作在模式 3 的计时器 0 使 8051 核看上去像是有 3 个计时/计数器。当计时器 0 在模式 3 时，可以通过把计时器 1 设置成模式 3 或设置为其他模式以使它被打开或关上，或是把它应用在不需要中断的计数应用中。

2.1.4 中断结构

8051 单片机提供 4 个中断源：2 个外部中断源，2 个计时器中断。这些中断由两个特殊功能寄存器 IE 和 IP 控制。通过置位或清除特殊功能寄存器 IE 中的位，可以单独使能每个中断源，寄存器中有一位是全部中断源使能位，清除这个位所有中断源不能使用。通过置位或清除特殊寄存器 IP 中的位，每个中断源可以被单独的设置一个共有两级的优先级。

一个低优先级的中断可以被高优先级的中断所中断，但不能被另一个低优先级的中断，一个高优先级的中断不能被任何中断所中断。如果具有同样优先级的中断请求同时到达，一个内部查询顺序决定哪个要求被响应。所以每个优先级别内还有次级优先结构。这个内部优先结构由查询顺序决定，其优先级顺序由低到高为 TF1、IE1、TF0、IE0。

外部中断源有两种激活方式，可以是电平激活或是跳变沿激活。这决定于特殊功能寄存器 TCON 中的 IT0 和 IT1。产生中断的标志位是特殊功能寄存器中的 IE0 和 IE1 位。片上硬件在服务程序响应中断后会清除产生外部中断的标志位，但这只有发生在外部中断是跳变沿激活的情况下。当中断是电平激活的时候，中断请求标志位是由外部中断请求源控制而不是片上硬件。

除了外部中断，计时器 0 和计时器 1 也可以产生中断。计时器中断是由各自计时器寄存器的标志位 TF0 和 TF1 产生的。与跳变沿激活的外部中断相似，产生计时器中断的标志位是由片上硬件在中断响应程序响应中断后清除的。

当中断发生时，中断系统产生一个指向正确程序存储器地址的调用，由以下条件阻碍中断：

- 同样级别或更高级别的中断正在被处理。
- 当前的周期不是正在执行的指令的最后一个周期。
- 当前运行的指令是 RETI 或其他写入 IE 或 IP 寄存器的指令。

如果激活一个中断标志但是因为上述的原因这个中断没有被响应，它会保持激活状态直到阻碍它的条件被移除或是被否定的中断不会被响应了。当中断被响应后，处理器在堆栈中保存寄存器。硬件只需要把程序计数器的内容压入堆栈中，然后重新将响应程序的起始地址读入程序计数器中。在中断是由跳变沿激活的情况下，它清除外部中断标志位（IE0 或 IE1）。只有程序计数器会被自动压入堆栈中，而不是 PSW 或是其他寄存器。只将 PC 自动的压入堆栈，编程者必须对保存哪些寄存器的选择十分小心。

每个中断的服务程序起始于固定的地点。中断地点之间有着 8 字节的间隔，对于外部中断 0 开始于 0003H，对于计时器 0 开始于 000BH，对于外部中断 1 开

始于 0013H，对于计时器 1 开始于 001BH。

执行服务程序跟随在上述的地址后直到遇到 RETI 指令。RETI 指令完成两件事情。一是它通知处理器中断服务程序已经执行完。二是重载堆栈的顶上的两个字节到程序计数器中。指令 RET 也可以完成以上结果，但是中断控制系统会认为中断仍然在进行中[12]。

2.2 操作系统结构

2.2.1 操作系统的概念

计算机系统一般有两部分组成：硬件和软件。计算机硬件是由中央处理器（运算器和控制器），存储器，输入输出设备等部件组成，它构成了系统本身和用户作业赖以活动的物质基础和工作环境，计算机软件又包括系统软件和应用软件。应用软件是为了用户应用计算机而编写的程序。系统软件如操作系统、系统实用程序、连接装配程序、多种语言处理程序、多种工具软件等。

任何一个计算机系统都是硬件支持下的软件执行系统。硬件提供了软件执行的条件，并把软件所要完成的工作用物理的方式表现出来。没有硬件的支持，软件无法工作，同样，没有软件的运行，硬件系统将无所作为。

由于微处理器系统的复杂性和通用性，使得应用系统的设计变成了一项十分复杂的工作。为了能够尽快地完成应用系统设计，同时也为了尽量降低对设计人员微处理器专业技术的要求，微处理器应用系统的设计都希望能够在操作系统支持下完成。操作系统提供了系统的基本操作平台，极大地降低了专业系统设计的复杂度，为应用系统设计者提供了最基本的开发工具和开发环境，并提高了系统开发的速度。目前微处理器应用开发技术提供了专用微处理器系统的应用设计平台，包括核心硬件电路和操作系统。对于微处理器系统来说，操作系统起到了核心和灵魂的作用。

图 2.3 为微处理器系统结构图，从图中可以看出，系统的核心是微处理器，操作系统起到了硬件与软件连接的作用。通过操作系统，应用设计人员不需要了解处理器系统的硬件操作具体方法，仅需要在设计开发平台中完成自身需要的软件即可。

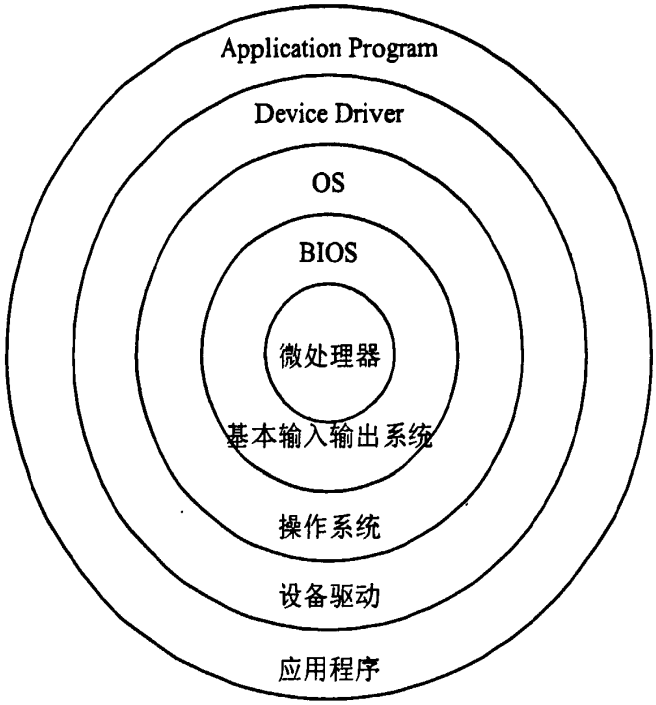

图 2.3　微处理器系统结构

Fig.2.3 Microprocessor system structure

微处理器系统在工作时，程序都是在操作系统内核管理之下执行的。微处理器系统具有十分强大的功能，完成这些功能主要是通过执行各种程序。例如，在同一屏幕中开设几个不同的窗口。这相当于设置了几个不同的工作区域，窗口之间的切换就需要执行不同的程序。对于复杂的微处理器系统来说，这种程序的交叉十分频繁，由于微处理器硬件自身并不能控制程序的切换。所以，这种切换就必须通过相应的程序进行控制，这种控制切换的程序就是微处理器系统的操作系统。所以说，操作系统内核实际上就是程序的基本管理程序。

从用户的观点来看，计算机是为用户提供服务的。引入操作系统是为了构建一个用户和计算机之间的和谐交互环境，让计算机为用户提供最好的服务。这就要求计算机有一个良好的用户界面，使用户无需了解许多有关硬件和系统软件的细节，能够方便灵活的使用计算机，同时，计算机还能为用户提供一个可靠和安全的服务管理。从发展的观点看，引入操作系统是为了给计算机系统的功能扩展提供支撑平台。

综上所述，可以把操作系统定义为：操作系统是管理计算机系统的全部硬件资源包括软件资源及数据资源；改善人机界面；控制程序运行；为其它应用软件提供支持等，使计算机系统所有资源最大限度地发挥作用，为用户提供方便的、有效的、友善的服务界面。

随着计算机技术和软件技术长期发展，已经形成了各种类型的操作系统，以

满足不同的应用要求。根据其使用环境和对作业处理方式，操作系统分为 6 种基本类型。批处理操作系统（batch processing operating system)；分时操作系统（time sharing operating system)；实时操作系统（real time operating system)；个人计算机操作系统(personal computer operating system)；网络操作系统(network operating system) 和分布操作系统(distributed operating system)。

2.2.2 操作系统的功能

操作系统的职能是管理和控制计算机系统中所有硬件和软件资源，合理的组织计算机工作流程，并为用户提供一个良好的工作环境和友好的接口。下面从资源管理和用户接口的角度分几个方面来说明操作系统的基本功能。

a) 进程管理

进程是操作系统中最重要，最基本的概念之一，它是系统分配资源的基本单位，是一个具有独立功能的程序段对某个数据集的一次执行活动。引入进程的概念是由于处理上的并行性和操作系统的资源有限性以及系统用户的执行起始时间的随机性决定的。一切仅有静态特征的概念，程序不能反映系统的上述特征。因此导入了具有动态特征的进程概念。

尽管进程是一个动态的概念，但是，从处理机执行的观点看，进程在执行中由于中断，等待，或程序出错发生进程调度，操作系统需要知道和记忆进程执行到何处和新进程从何处执行，进程仍需要静态描述。一个进程的静态描述是处理机的一个执行环境，被称为进程上下文。进程上下文包括：进程控制块（PCB)，数据段和正文段以及各种寄存器和堆栈中的值。进程控制块是系统感知进程存在的唯一实体。寄存器中主要存放将要执行指令的逻辑地址，执行模式以及执行指令时所要用到的各种调用和返回参数等。堆栈中存放 CPU 现场保护信息，各种资源控制管理信息等。

反映进程动态特性的是进程状态的变化，它有 5 个基本状态，初始态，就绪态，执行态，等待态和终止态。如图 2.4，进程的状态转换图。进程的并发特性反映在进程对资源的竞争以及由资源竞争所引起的对进程执行速度的制约，这种制约可分为直接制约和间接制约。直接制约是被制约进程和制约进程之间，存在着使用对方资源的要求。当制约进程执行后，被制约进程才能继续往前推进。即一个程序段等到另一程序段执行的信息结果后才能执行。间接式制约是指并发程序段竞争同一资源，获得者执行，未获得者挂起,等待资源。进程间的直接制约可利用 P,V 原语实现，也可以用其它互相传递信号的方式实现。间接制约可利用加锁法和 P,V 原语操作实现。

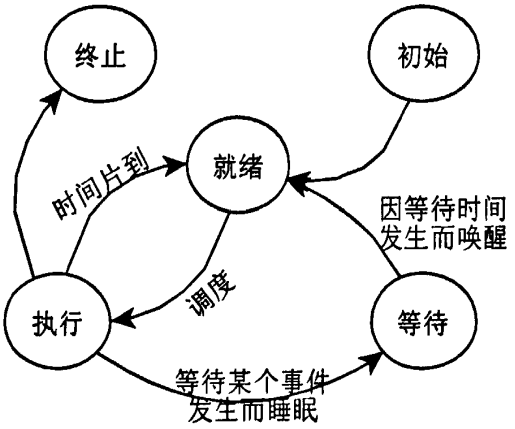

图 2.4　进程状态转换图

Fig.2.4 The process state transition graph

进程通信是进程管理中比较重要的概念，通信意味着在进程间传送数据。无论是互相通信的进程或是共享某些不同类型资源的进程，都可能因为通信顺序不当或资源分配顺序不当而造成死锁。死锁是指各并发进程之间互相等待对方所拥有的资源，且这些并发进程在得到对方资源之前不会释放自己所拥有的资源，造成各并发进程不能继续执行。比较常用的解决死锁的方法是检测与恢复法，即系统设有专门的机构，当死锁发生时，该机构能够检测到死锁的位置和原因，并能够通过外力破坏死锁发生的必要条件，从而使得并发进程从死锁状态中恢复出来。线程是为了提高操作系统的执行效率而引入的，它是进程内的一段程序的基本调度单位。线程可分为系统级线程和用户级线程。用户级线程的管理过程全部由用户程序完成，操作系统内核只对其进行管理。只要有一个能提供线程创建、调度、执行、撤销，以及通信等功能的线程库就行。

b) 处理机调度

处理机的调度可以分为 4 级，交换调度、作业调度、进程调度和线程调度。如图 2.5 是处理机调度图，可以看出 4 级调度的主要任务。

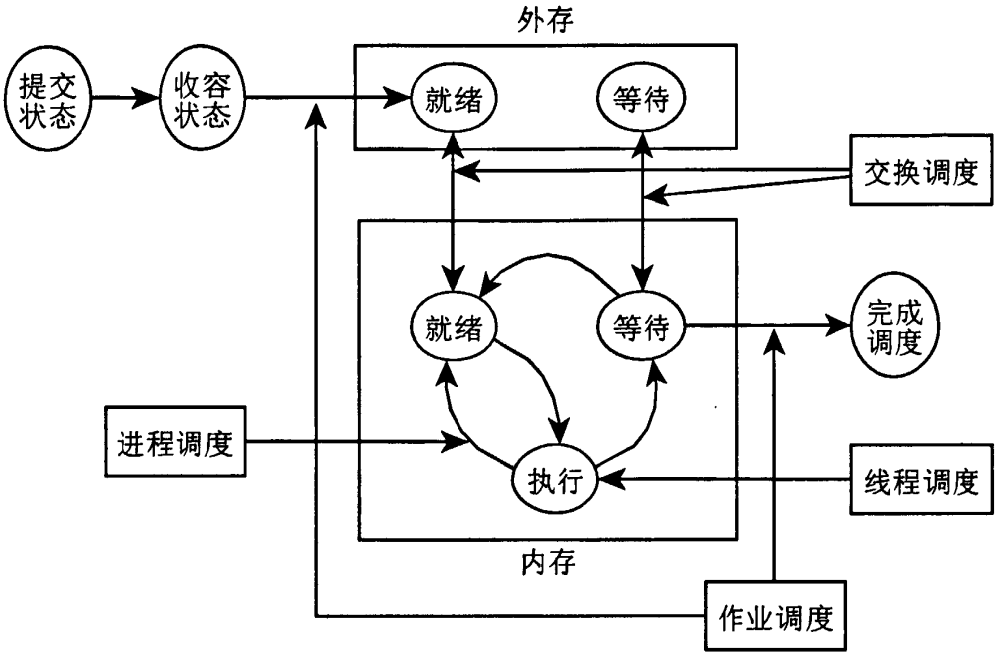

图 2.5　处理机调度

Fig.2.5 The processor scheduling

因为处理机调度程序不可能选择全部驻留在外存的进程，因此，在调度一个进程占有处理机之前，系统必须按某种策略把外存中处于后备状态的作业选择出来，并创建进程和分配内存，为进程执行准备必须的资源。这一步称为作业调度或高级调度。比较常用的作业调度算法有：最短作业优先法（SJF）、先来先服务（FCFS）方法、最高响应比法（HRN）。其中 FCFS 方法系统开销小，但不利于短作业。SJF 法可得到最大系统吞吐率，即处理的作业个数最多。但可能长作业永远没有机会执行。HRN 介于前两种方法之间，它同时考虑每个作业的等待时间和估计需要执行的时间长短。从中选出响应比最高的作业投入执行。响应比 R 可以定义如下：

$$R = (W + T)/T = 1 + W/T \qquad \ldots(2.1)$$

T 为该作业估计需要的执行时间，W 为作业在后备状态的等待时间。

进程调度的主要任务是按照某种策略方法选取一个处于就绪状态的进程占据处理机。在确定了占用处理机的进程后，系统必须进行进程上下文切换以建立与占用处理机进程相适应的执行环境。比较常用的调度算法有：轮转法（RR）、FCFS 法、线性优先级法（SRR）和优先级法。轮转法是将 CPU 的处理时间分成固定大小的时间片，当一个进程的时间片用完，但任务未完成，则它自行释放自己所占

有的CPU而排到就绪队列末尾，进程调度程序又去调度当前就绪队列中的第一个进程或作业，轮转法主要用于分时系统，它具有较好的响应时间。FCFS不利于执行时间短的进程。RR 算法是确定进程的优先级。SRR 算法是将新创建的进程按FCFS方式排成就绪队列，而其他已得到过时间片服务的进程也按FCFS方式排成另一个就绪队列[13]。

c) 存储管理

存储器是计算机系统的重要资源之一。因为任何程序和数据以及各种控制用的数据结构都必须占用一定的存储空间，因此，存储管理直接影响系统性能。

内存管理的核心问题是如何解决内存与外存的统一，以及他们之间的数据交换问题。内存与外存的统一管理使得内存的利用率得到提高，用户程序不再受内存可用区大小的限制，内存管理解决内存扩充、内存的分配与释放、内存保护与共享、内外存之间数据交换的控制、虚拟地址到内存物理地址的变换等问题。

虚拟存储器是存储管理的核心概念。内存价格昂贵，不可能用大容量的内存存储所有被访问或不被访问的程序与数据段，而外存尽管访问速度慢，但其价格便宜，适合存放大量信息。在一个进程执行时，大部分的程序和数据并不经常被访问，存储管理系统把不经常被访问的程序段和数据放入到外存中，等到需要访问时再将其放入内存。编译连接程序把用户源程序编译后连接到一个以 0 地址为始地址的线性或多维虚拟地址空间。每一个进程都拥有一个这样的空间，每个指令或数据单元都在这个虚拟空间中拥有确定的地址，把这个地址称为虚拟地址。将进程中的目标代码、数据等的虚拟地址组成的虚拟空间称为虚拟存储器。进程在该空间的实际物理地址由虚拟地址到实际物理地址的变换得到。

d) 文件系统

文件系统为用户提供了按名存取的功能，使得用户能透明地存储访问文件。为了实现按名存取，文件需要对文件存储设备进行合理的组织、分配和管理；对文件进行保护、保密和提供共享手段，文件系统还提供检索文件或文件中记录的手段。文件系统就是完成以上功能的一组软件和数据结构的集合。

e) 设备管理

设备管理的主要任务是控制 CPU 和外设之间进行 I/O 操作。设备管理模块在控制各类设备和 CPU 进行 I/O 操作的同时，还要尽可能地提高设备和设备之间、设备和 CPU 之间的并行操作度以及设备利用率。设备管理模块还应该为用户提供一个透明地、易于扩展的接口，以使得用户不必了解具体设备的物理特性和便于设备的追加和更新，常用的设备和 CPU 之间数据传送控制方式有 4 种。中断控制方式、程序直接控制方式、通道方式和 DMA 方式[14]。

2.3 NoC 基本结构

NoC（Networks-on-Chip）是本世纪初出现的新一代集成电路设计技术之一，因其面向纳米工艺的体系结构而成为科学家关注的焦点。现在一颗芯片上集成的晶体管数量越来越高，而特征尺寸越来越小。工作频率达到 10GHz，单颗芯片内的 IP 核数目也可以达到 1000 以上，基于总线的互联结构不再适合目前的片上互连需求。这种多核芯片互连结构已经成为当前国际多核芯片研究领域的新热点[15]。

集成电路发展到 SoC 时代，已经把整个数字计算机系统集成到了一块芯片上。SoC 是把芯片结构、模型算法、各层次电路甚至器件的设计紧密结合起来考虑，在一个芯片上一次性实现整个计算机系统的功能。实现 SoC 中 IP 核互连主要使用片上总线的技术方式。这种设计技术使用简单，速度更快，SoC 由于可以提供高性能的互连而被广泛应用。然而随着半导体工艺技术的持续发展，总线结构出现了一些局限性，主要表现在以下几方面：①可扩展性差。②平均通信效率低。③单一时钟同步问题。NoC 的提出解决了长期困扰总线结构的难题：全局时钟问题，并解决了多 CPU 系统的体系结构问题。NoC 的研究涉及到从物理设计到体系结构、应用服务、设计方法、操作系统和工具等诸多方面[16]。

一个 NOC 的基本结构应该包括以下几个部分，如图 2.6 所示。

1) 网络适配器（network adapter）：实现了 IP 核和 NoC 网络之间连接接口。分离网络通信和 IP 核的计算功能。
2) 路由节点（routing node）：通过路由协议进行路径选择，实现路由策略。
3) 链路（link）：实现节点连接，包含了一个或多个逻辑、物理信道。
4) IP 核（IP core）：IP 核可以是同质的，也可以是非同质的；可以是细粒度的，也可以是粗粒度的[17]。

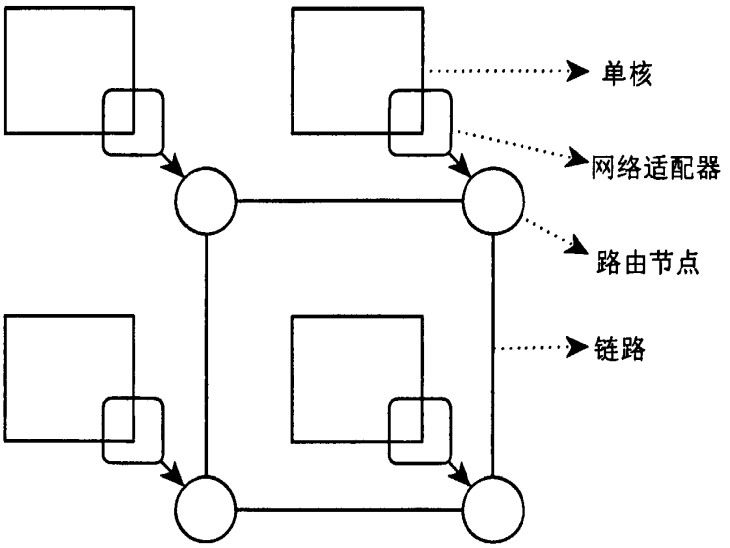

图 2.6 NoC 基本结构

Fig.2.6 The basic structure of NoC

图 2.6 以 4 核处理器为例显示了 NoC 基本结构，其中采用了一般网格拓扑结构，通过一定的路由算法可以实现任意两个 IP 核之间的通信，可采用包交换、电路交换或其它的路由方式，可使用同步、异步或其它逻辑实现。

NoC 网络传输信息的能力主要依靠其拓扑结构。它对吞吐量、面积、网络延迟、容错、功耗有很大影响，而且对设计策略和内核到网络节点的映射起着重要的作用。目前，理论上还不能证明可以用最优的拓扑结构来实现任何一个给定的应用。专用的网络结构能减小面积、提高性能、降低功耗。采用不规则的网格拓扑结构会引起版图设计、不均匀的线长等设计问题。因此，使用有效的拓扑结构以平衡高层性能与细节的实现方法受到了微米或纳米级工艺的制约。

大部分的 NoC 研究和设计借鉴了并行计算机体系结构中的静态网络结构，包含规则和不规则两种结构。常见规则结构如 Torus 结构、UPMC SPIN 的扁平树结构、KTH Nostrum 的 2D-mesh、3D-mesh 结构八角形结构和 Proteo 的环形结构等。目前大部分采用了 2D-mesh 或者其变形的 Torus 结构为网络的拓扑结构。不规则的结构是由规则结构组合而成的。

NoC 设计中一个很重要的问题就是路由的类型，它在很大程度上影响了网络的性能和功耗。而且，结构复杂的路由设计也会加大设计难度。

在选择路由算法时，主要考虑复杂度和性能两方面要求。在保证有序的数据传输时，自适应路由比确定性路由需要更多的资源。但是，自适应路由在网络拥塞时可以交换传输路径，所以它有更优化的吞吐量和延迟。如果信息无序传输，自适应路由性能就会变差。路由算法应该有效解决死锁和活锁的问题，它们的检

测恢复机制是很重要的，他们可能导致无法预测的延迟。完全自适应算法需要采取预防死锁和活锁的措施，基于转换模型的部分自适应和确定性算法自身可以完全避免。

当应用中产生的信息量可以预测时，可以采用确定性路由，也可以从应用出发，设计路由算法来匹配应用数据传输模式。在[18][19]中讨论了 NoC 中高容错随机路由。其中的研究主要是通过实验得到而没有理论基础，现在需要性能模型来解释由传输冗余数据所消耗的能量[20]。从功耗考虑，[21]中提出了优化功耗峰值的自适应路由。这种自适应算法不涉及到时序约束，时序约束可能并存功耗约束。因此，需要从功耗和性能考虑。

网络中的交换技术问题也是比较关键的一个问题。路由算法确定后，交换技术决定怎样建立和恢复路由内的转换器以及怎样通过转换器传输包。因此，满足了不同交换技术下应用的限制还要在复杂度和性能中取舍。虫孔交换是目前 NoC 中的主流交换机制，它的优点是缓存要求低，时延小。在数据网络中，相比较虫孔交换，电路交换的信道利用率低，但是对于特定应用的 NoC，这并不是主要的障碍，使用电路交换更能保证一些特定应用的服务操作。因此，复杂性和缺乏灵活性并不影响电路交换成为一种常用的交换技术[17]。

NoC 中每一个路由的输入缓冲都会影响整体的面积。每个输入缓冲增加 2 到 3 个字，NoC 的路由面积增加 30%。因此，为了减少 NoC 的面积需要最小化使用缓冲。根据网络的工作量增加缓冲区的大小可以减少几个数量级的网络延迟。但是，由于很多特定应用 NoC 中通信模式的不均匀，重载信道需要多分配些缓冲资源。因此相同数量缓冲区时，平均包延迟的值却是截然不同的。

[22]中研究了片上缓冲的一些特性。对门级面积进行估计，分析了网络性能和整个网络的缓冲利用率。[23]中提出了一种缓冲分配算法。这种方法假设是存储转发还是通断式分组交换技术，是确定还是未知的路由算法，并假设网络中所有包传输服从泊松分布。在这些假设条件下，研究人员推导出每个通道阻塞率，然后给高效率信道添加更多的缓冲资源[24]。

在 NoC 设计时也要考虑 IP 映射的问题，映射是在给定任务图、延迟、功耗和 IP 库的基础上，将每个任务分配到合适的 IP 核上并安排每个核上的任务执行顺序，然后再决定每个 IP 核在 NoC 拓扑结构中的位置。在映射时，搜索空间随着网络尺寸的增长呈阶乘递增。一个包含 25 个 IP 核的 NoC 结构，映射有 251 种可能结果。因此，映射问题也是 NoC 设计中的一个重要问题。当前映射可以采用启发式的算法进行。

以上是NoC设计的一部分问题。在具体实现一个NoC的时候，从软件到硬件，从物理层到应用层， NoC 涉及的方方面面远多于这些问题。全新的 NOC 集成电

路体系结构从根本上解决了总线结构的固有问题，片上网络的体系结构一开始就建立在并行通信的基础上，在通信效率方面优于总线。片上网络从消除时钟树以节省资源、多处理器体系结构、实现并行通信等几个重要方面，展示了优于总线结构的本质和属性。

2.4 开发软件 Keil C51

本操作系统的开发全部使用软件 Keil C51。Keil C51 是美国 Keil Software 公司出品的 51 系列兼容单片机 C 语言软件开发系统。与汇编相比，C 语言在功能上、可读性、结构性、可维护性上有明显的优势，因而易学易用。Keil C51 软件提供丰富的库函数和功能强大的集成开发调试工具，它是全 Windows 界面。另外重要的一点，可以看出 Keil C51 通过编译后生成的汇编代码效率非常的高。多数语句生成的汇编代码很紧凑，容易理解。在开发大型软件时更能体现高级语言的优势。

图 2.7 为 Keil C51 工具包的整体结构，uVision2 是 Keil C51 的 Windows 集成开发环境(IDE)，可以完成编辑、连接、调试、编译、仿真等整个开发流程。开发人员可用 IDE 本身或其它编辑器编辑 C 或汇编源文件，然后分别由 C51 或 A51 编译器编译生成目标文件(.OBJ)，目标文件可由 LIB51 创建生成库文件，也可以与库文件一起经 L51 连接定位生成绝对目标文件(.ABS)，ABS 文件由 OH51 转换成标准的 Hex 文件，以供调试器 dScope51 使用进行源代码级调试，也可由仿真器使用直接对目标板进行调试，也可以直接写入程序存贮器如 EPROM 中[25]。

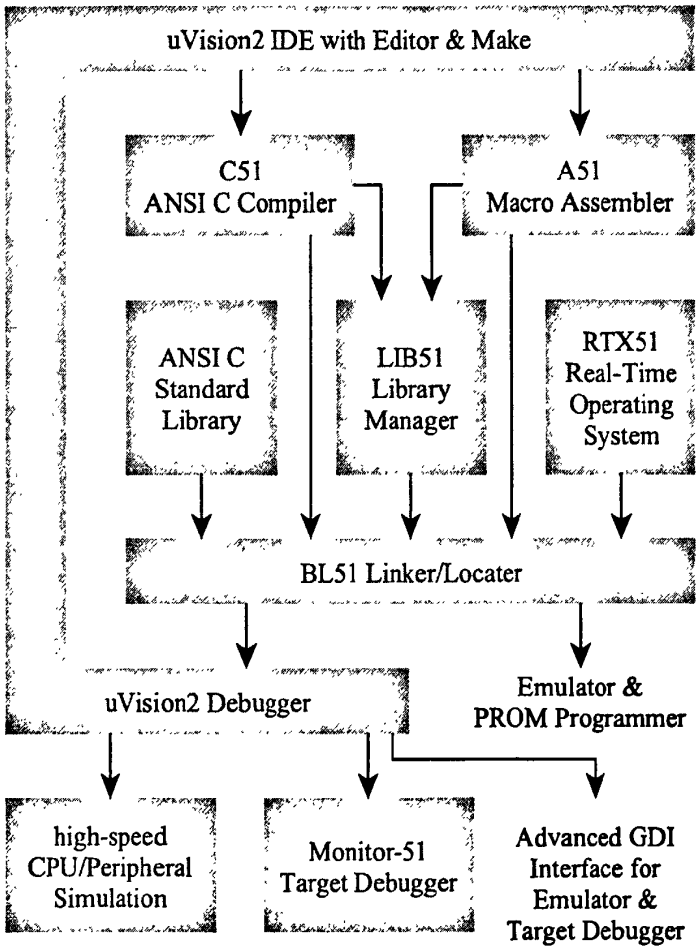

图 2.7 C51 工具包整体结构图

Fig.2.7 The whole structure of C51 kit

2.5 开发简单嵌入式操作系统需要注意的问题

1）操作系统占用空间的大小问题

51 系列单片机只有 128 字节或者 256 字节的片内 RAM 空间。如果操作系统使用大量片内的 RAM，那用户的应用程序就没有存储空间，如果用户程序把变量定义在片外 RAM 中，系统的硬件堆栈又无处可放。51 系列单片机的硬件堆栈不能放在片外存储器，所以在 51 系列单片机上开发操作系统就要尽量少用片内 RAM。因为操作系统需要传递参数和使用堆栈，使用片内 RAM 是不可避免的。C51 单片机的 C 函数是通过寄存器和存储器传递参数的,不能通过堆栈传递参数，但是可以采取一些措施使得操作系统代码占用少量片内 RAM。

2）操作系统软件代码的长短问题

51 系列单片机的系统硬件资源相对匮乏，如果操作系统的代码比应用程序的代码还大,甚至使得用户的应用程序要考虑给操作系统让出资源，这样的操作系统即使功能再完善，也不实用。现在市场上多数的嵌入式操作系统不能应用于 51 系列单片机，就是因为其代码量太大。开发一个 5000 行的基于裸机的应用程序占用大概 7-8KBROM 空间，一个操作系统用去几十 KB，不仅占用空间大，还削弱了嵌入式操作系统的实时性这一优势。

3）函数的重入问题

开发实时占先式的操作系统，使用可重入函数是非常重要的。可重入函数可以被多个任务同时调用，而不会破坏数据。可重入函数任何时候都可以被中断，经过一段时间后又可以运行，且应用数据不会丢失。函数具有可重入性必须使得函数能够满足下列三个条件之一：

① 不使用共享资源；

② 在使用共享资源时关中断，使用完毕后再开中断；

③ 在使用共享资源时申请信号量，使用完后释放信号量。

这些条件在标准 C 中编程很容易实现，但是在 Keil C51 中就比较麻烦。因为标准 C 是通过动态分配将局部变量分配到用户堆栈中，而 Keil C51 通过静态分配将局部变量分配到寄存器或内存固定地址，并通过变量覆盖分析的方法，使多个函数的局部变量使用相同的内存地址以减少占用内存。在 Keil C51 中局部变量分配在寄存器中比分配在内存中相对好些。

4）堆栈的分配问题

占先式操作系统的主要任务就是进行任务的调度，通过对任务的实时调度来完成系统的功能。任务调度过程中，不可避免的发生任务对系统资源的抢占问题。系统中每个任务都认为自己是 CPU 的绝对占用者，每一个任务都是一个死循环。任务间进行切换的依据就是各自的优先级，一个高优先级的任务可以通过任务调度函数或者中断退出函数等来中止正在运行的任务，被中断的任务只有自己的优先级在当前就绪任务表中最高时，才能从被中断处继续运行。这就需要为每个任务分配任务堆栈来保存任务的环境变量。由于每个任务在不同时刻被中断时需要保存的环境变量数目不同，所以任务堆栈空间的分配问题也是一门学问[2]。

3 嵌入式操作系统的结构设计

本设计中基于 51 单片机的嵌入式操作系统是实时多任务操作系统。51 单片机内存空间十分有限，这是在设计时需要时时注意的问题，这个嵌入式操作系统不具备大型操作系统中完备的功能，例如文件系统、存储器管理等，它是一个微内核结构基本包含操作系统所必须的部分。用于单片机 C51 核的操作系统内核结构如图 3.1 所示。在这个操作系统中，任务调度和进程通信是内核的主要部分。

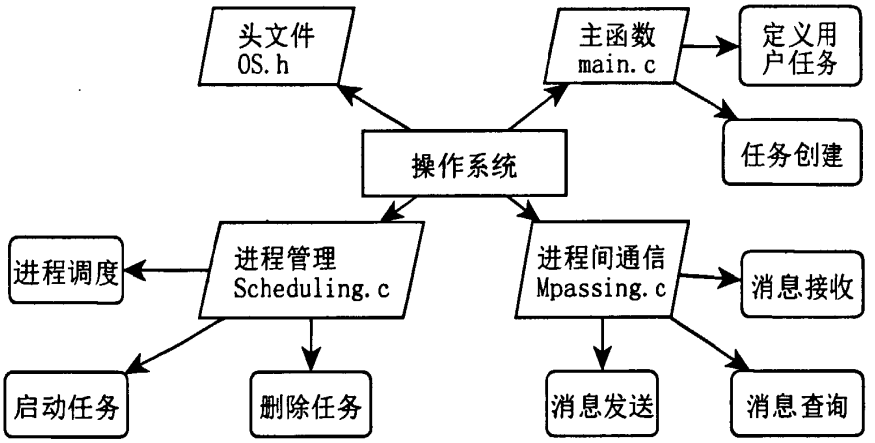

图 3.1 操作系统结构图

Fig.3.1 The operating system structure

从图中可以看出，这个操作系统包含 4 个文件；

a) OS.h　这是 OS 唯一包含的头文件，其他头文件均包含在这个文件中。

b) main.c　OS 的主函数，包含任务创建和定义用户任务。

c) Scheduling.c　进程管理部分包括启动任务、进程调度和删除任务。

d) Mpassing.c　进程间通信部分包括消息发送、消息查询和消息接收。

以下将以这四部分作为划分来分析这个操作系统的结构。

图 3.2 中“projet”窗口为 Keil C51 软件中显示的操作系统的树形结构图，从这个窗口中可以看到操作系统四个主要部分的分支结构。

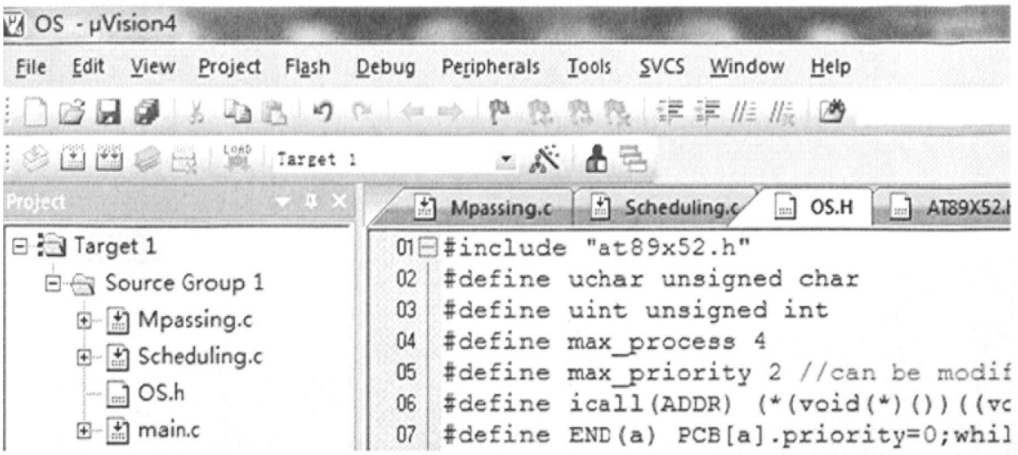

图 3.2 软件中的操作系统结构显示

Fig.3.2 Operating system source code structure

3.1 操作系统的头文件

本设计中的嵌入式操作系统中所有的头文件均包含在 OS.h 这个文件里，用户不必考虑每个文件都应该对应包含什么头文件。这样做的代价是增加了编译时间，但是现在计算机速度相当快，增加的时间并不多。这样做的一个很重要的好处是增加了编译的移植性。如果将程序由一种 CPU 移植到另一种 CPU，就省去对每一个文件都修改所包含的文件[26]。

操作系统使用 8051 单片机作为目标处理器，头文件应该包含文件“reg51.h”，该头文件中将一些硬件地址定义成对应的 C 程序中的寄存器名和位寄存器名。OS.h 还包含了整个系统中用到的宏定义、变量、结构体、数组以及对所有函数的声明，这样当一个函数被另一个文件里的函数调用时不用在该文件里声明。头文件还包括每个文件中各个函数的声明，这样只要在每个文件中包含该头文件，一个函数需要调用其他文件函数时不用在这个文件中另作声明。文件通过宏定义给出操作系统的各个参数值，需要修改参数值时可以直接在这个文件中修改。表 3.1 显示了操作系统中所有参数值,即在头文件中声明到的变量、数组等，其中一些宏定义、变量、结构体、数组的代表意义将在以后各章节详细阐述。

表 3.1 操作系统参数表

Table 3.1 Parameter list of the operating system

各项参数	数值大小
时间片	10ms
任务数量	4
第一个任务优先级	2
第二个任务优先级	2
第三个任务优先级	1
第四个任务优先级	1
优先级取值范围	1~6
数据缓冲区	8 字节
任务堆栈	15 字节

以下是本嵌入式操作系统头文件的代码段：

```
#include "at89x52.h"
#define uchar unsigned char
#define uint unsigned int
#define max_process 4
#define max_priority 2 //can be modified by user
#define icall(ADDR) (*(void(*)())((void*)ADDR))(); //call task by address
#define END(a) PCB[a].priority=0;while(1); // task end flag
#define num 15
#define mnumber 4
typedef struct PCB
{
   uint process_p;     // task address
   uchar process_id;
   uchar priority;     //The greater value, the higher priority,range from
1 to 6
   uchar cur_pcb[num]; // task current state,include all work registers
and R0~R7

} PCBtype;
extern PCBtype PCB[max_process];
void Init_PCB();
void create_process(/*uchar,*/uint,uchar,uchar);
void task_0();
void task_1();
void task_2();
void task_3();
```

```
extern uchar im; // the queue ID of current implementing task
extern uchar im_temp;
extern uchar re_flag; // the number of removed processes
extern uchar que[max_process];
void make_queue();
void process_start();
void remove_process();
void timer0();

extern uchar message [2*max_process];
uchar message_send (/*uchar,*/uchar,uchar);
uchar message_inquire(uchar);
uchar message_receive(/*uchar,*/uchar*,uchar);
```

3.2主函数

主函数部分首先通过结构体数据结构定义并使用了进程控制块（PCB），进程控制块(PCB)是系统为了管理进程设置的一个专门的数据结构，用它来记录进程的外部特征，描述进程的运动变化过程。系统利用 PCB 来控制和管理进程，所以 PCB 是系统感知进程存在的唯一标志。进程与 PCB 是一一对应的。操作系统的 PCB 一般包括：进程标识符、进程当前状态、进程相应的程序和数据地址（以便把 PCB 与其程序和数据联系起来）、进程优先级 priority 以及与进程有关的其他信息。

进程调度的过程中需要进行上下文切换。在多任务系统中，上下文切换是指 CPU 的控制权由运行任务转移到另外一个就绪任务时所发生的事件，当前运行任务转为就绪（或者挂起、删除）状态，另一个被选定的就绪任务成为当前任务。上下文切换包括保存当前任务的运行环境，恢复将要运行任务的运行环境。上下文的内容依赖于具体的 CPU。操作系统的上下文切换在不同硬件体系结构上的实现机制不同，例如，UCOS 就是利用中断机制进行上下文切换的。在 UCOS 中，如果调度程序决定任务需要切换，就会调用上下文切换 OS_TASK_SW()进行实际的上下文切换。OS_TASK_SW()是宏调用，含有微处理器的软中断指令，利用此中断来实现任务之间的上下文切换。所以 OS_TASK_SW()是一个体系结构相关的宏，对于不同的硬件体系机构，有不同的实现方式，这也是 UCOS 在不同硬件体系结构中移植的一个要点。在一般的操作系统中都使用任务堆栈来存储当前任务的运行环境。任务堆栈的大小要根据硬件结构，也是当前任务的运行环境来确定。本设计中的操作系统将任务堆栈合并到 PCB 结构体中，而没有单独定义。根据 51 单片机结构，确定 PCB 中这一部分的大小。本操作系统的 PCB 包含以下内容：

a) 任务地址（Task address）　占用 2 字节（byte）存储空间。根据 51 单片机 16 位（bit）寻址空间，任务地址为 16 位。用来存储每个任务在程序寄存器中的地址。

b) 任务标识符（The identifier of task）　占用 1byte 存储空间。用来标示不同任务。本设计中操作系统任务数量是 4 个，任务标识符的取值范围为 0 到 3。

c) 任务优先级（Priority）　占用 1byte 存储空间。数值越大任务的优先级别越高。优先级的取值范围是 1~6。 优先级取值范围是根据时间片的大小以及硬件结构确定的，方法将会在进程调度一部分详细介绍。

d) 任务堆栈（Task stack）　这是一个数组变量，共有 15 个元素，占用 16byte 存储空间。在进行上下文切换时，这一数组用来存储当前任务的运行环境。根据 51 单片机的运行机制，这 15 个元素分别是程序寄存器（PC）、累加器（ACC）、B 寄存器（B）、数据指针高八位（DPH）、数据指针低八位（DPL）、程序状态寄存器（PSW）和工作寄存器 R0~R7。其中除了 PC 存储程序地址需要 16 位 2byte 存储空间，其他寄存器均占用 1byte。数组的元素个数在 OS.h 文件中进行了宏定义，这么做的目的是方便移植。当改变目标处理器时，根据硬件的运行环境设置这个值。

3.2.1 PCB 初始化

当启动操作系统时，需要进行的第一步就是 PCB 初始化。任务地址和优先级的初始值为 0，任务标示设置为 4 表示该 PCB 为空。堆栈数组的初始值全部设置为 0。

3.2.2 任务创建

任务创建函数共有 4 个形式参数。第一个参数是任务被创建的处理器，这个参数是为了以后扩展应用于多处理器的操作系统预留的接口。第二个参数是程序存储器中该任务的地址。还有创建任务的标识符和创建任务的优先级。考虑 51 单片机的存储空间小，操作系统不可以创建任务数量过大，设置为 4 个。为了节省存储空间，使用一种静态方法建立任务，即所有任务必须在任务启动前创建，一旦任务开始运行不能再建立新任务。这样做法有一点缺点就是降低了任务灵活性。任务创建流程图如图 3.3 所示。

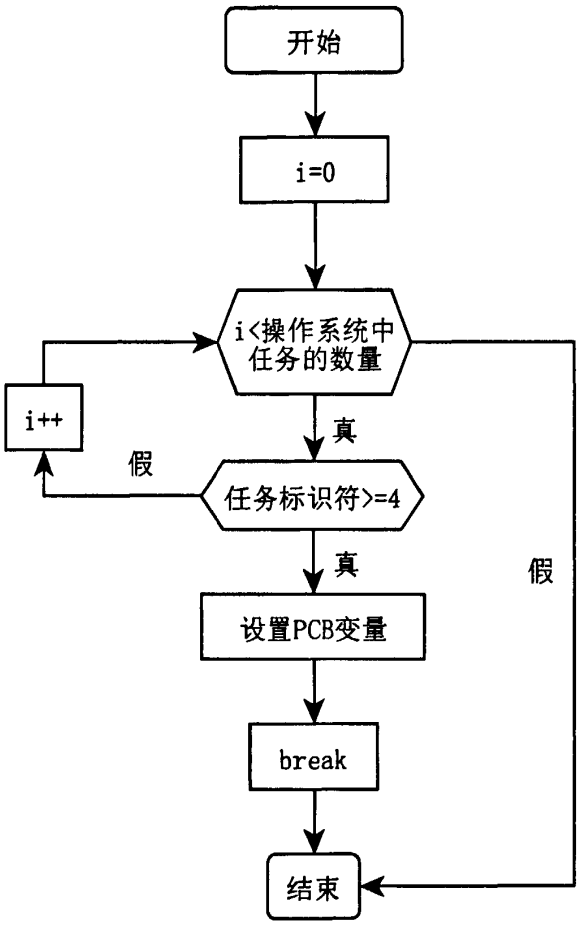

图 3.3 任务创建流程图

Fig.3.3 Task creation flowchart

任务创建函数的主要功能就是分配 PCB 给每个任务。函数先查询是否有空 PCB，然后设置 PCB 的值。通过判断每个 PCB 里任务的标识符判断 PCB 是否为空，标识符为 4 代表 PCB 为空。确认有空 PCB 后，任务的当前状态就会被存到 PCB 中。任务的标识符、优先级和地址通过形式参数传递到 PCB 中，地址存入到堆栈数组的最后两位。其余变量均设置为 0。

以下是实现创建任务功能的主要代码段：

```
void create_task(/*uchar processor,*/uint current_p,uchar
current_id,uchar current_priority)
//assign PCB for task
{
   uchar i = 0;
   for (i = 0; i < max_process; i ++)
   {
      if (PCB[i].process_id >= max_process)
```

```
        {
            PCB[i].process_p=current_p;
            PCB[i].process_id=current_id;
            PCB[i].priority= current_priority;
            PCB[i].cur_pcb[num-1]=PCB[i].process_p;
            PCB[i].cur_pcb[num-2]=PCB[i].process_p>>8;  //the last two
registers store task address
            break;
        }
    }
}
```

3.2.3 用户任务

本设计中操作系统用户任务数量值为 4，在文件 OS.h 中有定义，方便进行重新设置。一般来讲，嵌入式应用系统的很多任务是无限循环的，但也有特殊情况。因此，在每个任务最后以“END(ID of task)”作为结束标志（这样标志的原理将会在进程调度一节讲明），以便于在进程调度时 OS 不会再执行已结束的任务而节省了时间和空间。用户任务应统一写成如下格式:

```
void task_ID
{
......
END(ID)
}
```

当全部任务创建完毕，OS 将会启动第一个任务。

3.3 多任务调度

嵌入式操作系统内核中，任务调度是非常重要的一部分。在一个系统中，如果处理器的数量少于进程的数量，一些进程不能够立即被运行，需要等待空闲的处理器。任务调度所要做的就是决定下一个要执行的程序，给出需要运行程序的执行顺序。无论对于多核操作系统还是实时操作系统，任务调度都是不能够忽视的。调度涉及到进程被分配到可行目标处理器的方式，这种分配方式可以通过软件调度程序执行。为了充分利用处理器时间，在调度过程中，处理器应该总是在运行中，调度时间占用较少。

3.3.1 调度算法

由于 8051 单片机硬件条件的限制，早期在 8051 系列微处理器开发平台上开发的应用均是单任务执行，单任务编程对于明显的顺序控制要求编程简单、思路清晰开发周期短。这样并不能满足目前越来越复杂的应用系统。随着软硬件技术的发展，开发基于 8051 的多任务并行实时操作系统成为必然[27]。51 单片机的内存空间比较小，以其为平台的操作系统内核需要空间复杂度较小的任务调度算法。又由于嵌入式操作系统的实时性，调度算法的时间复杂度必须要低。

本操作系统内核使用一种基于优先级的时间片轮转调度算法。这是一种静态调度算法，即在任务开始执行前所有任务的执行顺序以及时间已经确定。操作系统启动任务按照确定顺序和时间执行，期间不再调度执行新的任务。这样可以大大节省存储空间和调度时间。轮转法是一种基于时钟的抢占策略，以一个周期性间隔产生的时钟中断依次进行各个进程的调度。当中断发生时，当前正在运行的进程被置于就需队列中，然后再基于 FCFS 策略选择下一个进程运行。这种方法也称为时间片方法。它主要应用于分时系统中的进程调度。时间片轮转算法是操作系统任务调度中最简单的算法之一，它的基本思想是：将处理器的执行时间划分为无数个相等的时间片分配给每个进程，所有进程按顺序执行，没有优先级大小。时间片轮转算法简单易执行，并且处理器的执行效率很高，不会产生空闲状态。

本设计中的操作系统使用了基于优先级的时间片轮转算法。这种算法将时间片分给所有任务，构成微观上轮流运行，宏观上并行执行的多任务效果。所有任务根据优先级由高到低的顺序排成一个队列，处理器根据这个队列顺序执行。CPU 的执行时间被分为无数个大小相等的时间片，由表 3.1，时间片的大小为 10ms。用户可以根据目标处理器的晶振频率设置时间片的大小。当操作系统中 4 个任务按顺序执行完相应时间片后，再调度第一个任务执行。之所以称为基于优先级的调度算法，是因为每个任务的执行时间与其优先级的大小成正比，如下式：

在一次轮转中任务的执行时间 = 任务的优先级大小 * 时间片

在本操作系统内核中，该调度算法的执行顺序如图 3.4 所示，从任务一开始执行。在操作系统调度任务之前，利用一个函数将所有任务按照优先级由高到底的顺序排列并保存任务序号到一个队列中。启动任务时直接调度第一个任务并按照队列顺序执行。第一个任务的优先级为 2，根据以上公式被执行两个时间片。第一个任务执行完需要的时间片时，定时器 0 发生中断执行第二个任务，第二个任务优先级也为 2，执行 2 个时间片后定时器 0 再次发生中断执行下一个任务，当所有

任务按要求的时间片执行完一次，再返回到第一个任务中断的地方继续执行，如此下去。四个任务的优先级分别为 2,2,1,1,可以得到它们的执行时间分别为总时间的 33%，33%，17%，17%。

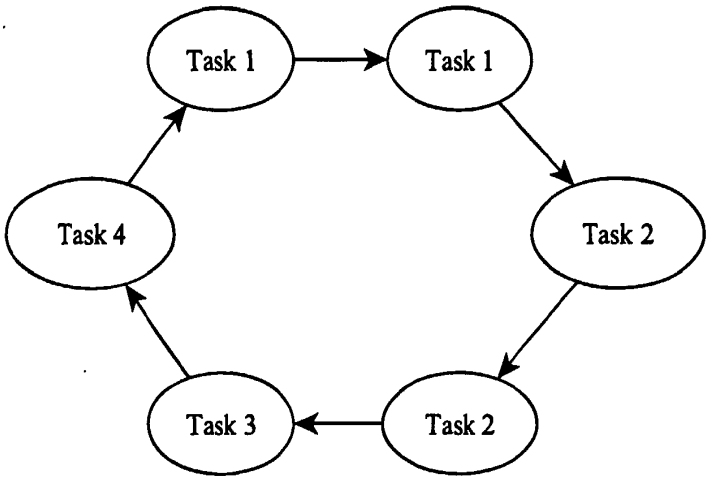

图 3.4 调度算法的执行顺序

Fig.3.4 The execution sequence of the scheduling algorithm

操作系统的目标处理器 51 单片机提供 4 个中断源，2 个外部中断，2 个定时器中断。完成任务调度需要目标处理器提供的中断源。本操作系统中使用 8051 的一个定时器中断源定时器 0 来完成任务的调度。定时器的计时功能可以实现时间片的设置。

3.3.2 时间片设置

操作系统的任务调度由 51 单片机的定时器 T0 协助完成。选用的 8051 单片机的晶振频率是 11.0592MHZ。一个机器周期包括 12 个晶振周期，计算可以得到目标处理器的机器周期大约是 1us。定时器 T0 是一个 16 位定时器，他的取值范围是 0~65535，定时器累计 10000 步是 10ms。定时器 T0 工作于 1 6 位定时器方式(方式 1) 。T0 的工作方式为: 每过一个机器周期自动加 1 当计满 OFFFFh 要溢出时，便会产生中断，并由硬件设置相应的标志位供软件查询。即中断时比启动 时经过了 N+1 个机器周期。所以，我们只要在 T0 中预先存入一个比满值 OFFFFh 小的数，然后启动定时器,便会在 N 个机器周期后产生中断。这个值便是所谓的初值[28]。操作系统中时间片的大小是 10ms（10000 个机器周期），定时器 0 的值可以由如下式子计算得到:

$$TIMER0 = 65536 - priority * 10000 \quad \cdots(3.1)$$

一个任务的执行时间等于任务的优先级乘以时间片的大小。由于一个 16 位定时器是由 2 个 8 位寄存器组成，所以需要分别设置定时器高 8 位和底 8 位的值。因为二进制八位数最大值为 256，定时器可以如下计算:

$$\text{高八位}\quad TH0 = TIMER0/256 \quad \cdots(3.2)$$

$$\text{低八位}\quad TL0 = TIMER0\%256 \quad \cdots(3.3)$$

由此得到定时器的设置方式如下：

$$TH0 = (65536 - priority * 10000)/256 \quad \cdots(3.4)$$

$$TL0 = (65536 - priority * 10000)\%256 \quad \cdots(3.5)$$

另一种定时器的设置方式：

$$TH0 = -(priority * 10000/256) \quad \cdots(3.6)$$

$$TL0 = -(priority * 10000\%256) \quad \cdots(3.7)$$

在 C 语言中，“-” 可以理解为溢出的意思“256-”。Priority*10000/256 是执行任务所需时间值的高八位，priority*10000%256 是执行任务所需时间值的低八位。比较两种定时器设置方式，在本操作系统中第一种方式比第二种设置方式多用 342 字节的程序存储器。考虑到 51 单片机存储空间小的问题，选用第二种定时器设置方式。注意时间片不可以设置太小，否则进程的调度时间过长，影响处理器效率。

时间片的大小为 10ms，即 10000 个机器周期。而定时器的取值范围是 0~65535。所以任务优先级最大可以取到 6，否则超出定时器的范围。用户设置完时间片大小后，根据定时器的取值范围得到优先级的取值范围。时间片和优先级必须满足如下关系式：

$$65536 - priority(maximum) * time\ slice > 0 \quad \cdots(3.8)$$

3.3.3 基于优先级的队列顺序

在任务调度前，首先应该给任务的执行顺序，即将操作系统的 4 个用户任务按照优先级大小排成队列。在这个操作系统内核中，使用一个排队函数，将用户任务按照优先级由大到小的顺序进行排列。定义一个全局变量数组“que”存储排列好顺序的任务 ID，以便于操作系统可以根据这个顺序执行任务。图 3.5 是这个函数的流程图。

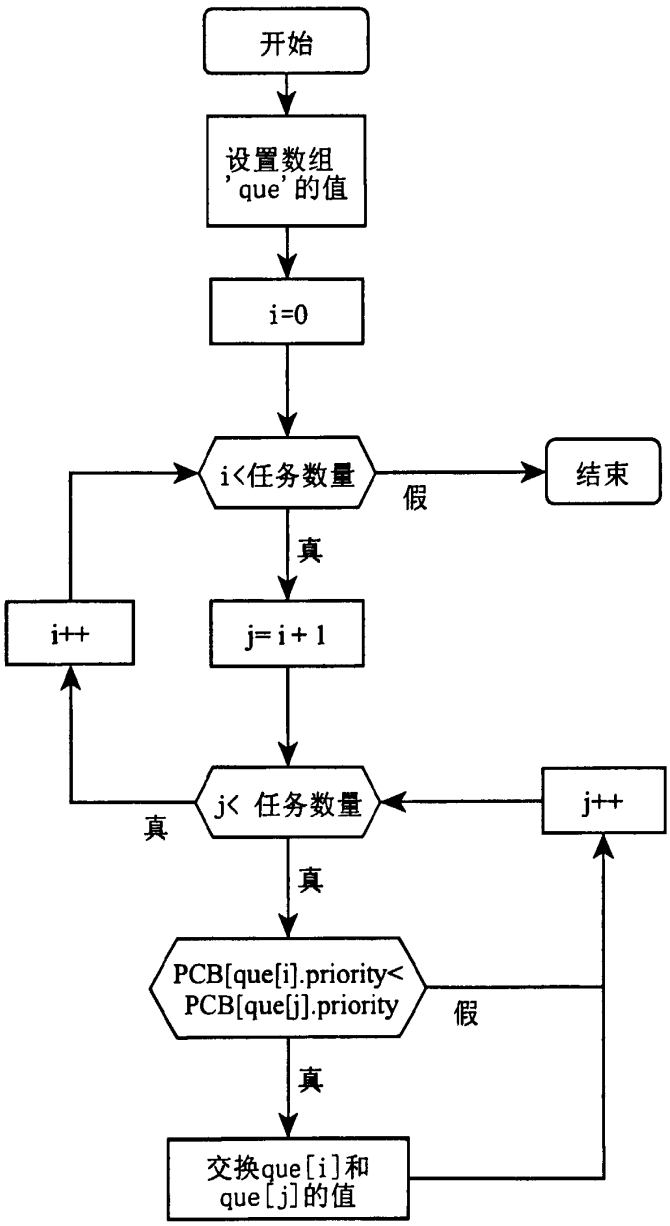

图 3.5　排序流程图

Fig.3.5 Process ordering flowchart

首先，将任务 ID 按标号顺序存入到数组“que”中。然后使用冒泡法将队列中的元素排列。这里注意队列数组中存储的是任务标号，而在冒泡法中比较的是任务的优先级。

以下是这种排序方式的主要实现过程：

```
uchar que[max_process];
void make_queue()    //tasks are arranged from high priority to low priority
by bubble sort
{
    uchar i,j,max;
    que[0]=PCB[0].process_id;   //the priority of que [0] is maximum
    que[1]=PCB[1].process_id;
    que[2]=PCB[2].process_id;
    que[3]=PCB[3].process_id;
    for(i=0;i<max_process-1;i++)
    {
        for(j=i+1;j<max_process;j++)
        {
            if(PCB[que[i]].priority<PCB[que[j]].priority)
            {
                max=que[i];
                que[i]=que[j];
                que[j]=max;
                max=0;
            }
        }
    }
}
```

因为 51 单片机的存储空间很小，在设计时，为了节省空间考虑了两种方法排列任务顺序，将这些方法进行比较从而选出最优的。第二种方法的流程与上一种方法类似，不同的是这里没有使用数组存储而直接使用 PCB 存储。在排列顺序时直接交换 PCB 中的元素，也就是将 PCB 的顺序按照优先级大小进行重新排列。这意味着操作系统的执行顺序不再是队列数组的顺序，而是 PCB 的顺序。这种方法中为了方便 PAB 的交换，需要设置一个和 PCB 大小相同的变量，而 PCB 所占用的存储空间要大于队列数组占用的存储空间，这种方法会使用更多的数据寄存器。因而不可取。第三种方法为了节省存储空间，使用 C 语言中的一种数据结构类型“位域”代替第一种方法中的数组来存储任务顺序。这样只需要使用一个字节的存储空间。这个字节的 8 位被分为 4 个 2 位，每 2 位存储一个任务 ID。这 4 个从头到位的顺序就是优先级由高到低的顺序。这种数据结构不能够使用循环排序，任务只能被逐个存储。任务的优先级会被比较 6 次。导致占用大量的程序存储空间。它的流程图如图 3.6 所示。

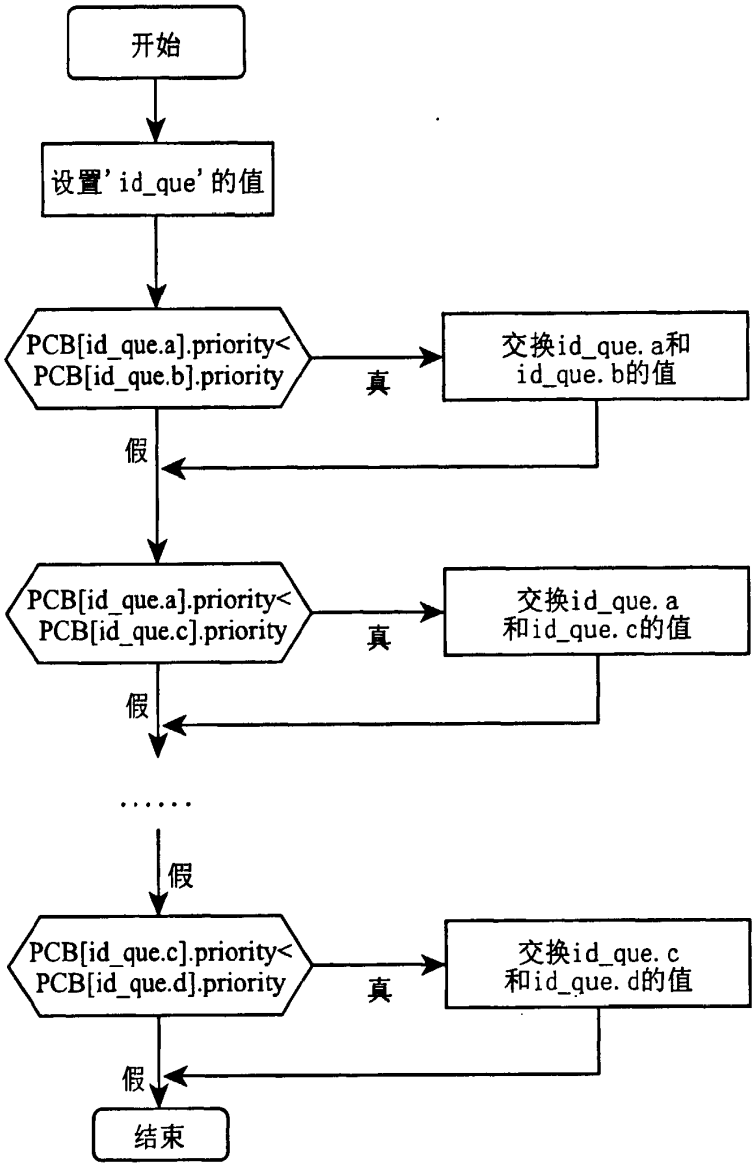

图 3.6 排序方法三流程图

Fig.3.6 The third process ordering method flowchart

表 3.2 给出三种方法中所使用的数据存储器和程序存储器的数量。从这个表中可以看出，方法二中使用的数据存储器和程序存储器都多于方法一使用的存储器。方法三中使用的数据存储器比方法一少 2 字节，但是程序存储器却远远多于方法一。几乎 700 字节的程序寄存器交换 2 字节数据寄存器是不值得的。综上，在本设计操作系统内核中使用第一种方式进行排序。

表 3.2　三种排序方法的比较

Table 3.2 Comparison of the three process ordering methods

	数据存储器	程序存储器
方法一	105bytes	1134bytes
方法二	118bytes	1485bytes
方法三	103bytes	1820bytes

以下是第二、三种排序方法实现过程的主要代码段：

```
typedef struct bitseg
{
   uint a:2;
   uint b:2;
   uint c:2;
   uint d:2;
} bitque;

bitque id_que;

void make_queue()
{

   uchar max=0;
   id_que.a=PCB[0].process_id;
   id_que.b=PCB[1].process_id;
   id_que.c=PCB[2].process_id;
   id_que.d=PCB[3].process_id;
   if(PCB[id_que.a].priority<PCB[id_que.b].priority)
   {
        max= id_que.a;
          id_que.a=id_que.b;
           id_que.b=max;
   }
   if(PCB[id_que.a].priority<PCB[id_que.c].priority)
   {
        max= id_que.a;
          id_que.a=id_que.c;
           id_que.c=max;
   }
   if(PCB[id_que.a].priority<PCB[id_que.d].priority)
   {
        max= id_que.a;
          id_que.a=id_que.d;
           id_que.d=max;
   }
   if(PCB[id_que.b].priority<PCB[id_que.c].priority)
   {
        max= id_que.b;
```

```
            id_que.b=id_que.c;
             id_que.b=max;
    }
    if(PCB[id_que.b].priority<PCB[id_que.d].priority)
    {
         max= id_que.b;
           id_que.b=id_que.d;
            id_que.d=max;
    }
     if(PCB[id_que.c].priority<PCB[id_que.d].priority)
    {
         max= id_que.c;
           id_que.c=id_que.d;
            id_que.d=max;
    }
}

void make_queue()
{
    uchar i=0,j=0;
    PCBtype expcb={0,max_process,0};
    for(j=0;j<max_process-1;j++)
    {
       for(i=j+1;i<max_process;i++)
       {
          if(PCB[j].priority<PCB[i].priority)
          {
             expcb=PCB[j];
             PCB[j]=PCB[i];
             PCB[i]=expcb;
          }
       }
    }
}
```

3.3.4 任务的启动和调度

启动任务函数的主要任务是设置定时器 0 的值和调用第一个进程。在这一部分定义一个全局变量“im”来存储当前正在执行的任务在队列数组中的标号，设置定时器 0 的运行方式为模式 1 即 16 位定时器。因为定时器 0 是由 2 个 8 位寄存器组成，所以及存取的初始值需要分别设置。定时器 0 的初始值是第一个进程需要被执行的时间，而且在队列数组中第一个任务的优先级是最高的。所以，定时器 0 的初始值可以如下设置：

$$TH0 = -(max_priority * 10000/256) \quad \cdots(3.9)$$

$$TL0 = -(max_priority * 10000\%256) \quad \cdots(3.10)$$

将51单片机中断使能寄存器中的“ET”置1，它的作用是使能溢出中断。置位时间控制寄存器中的“TR0”位开启定时器0。为了开启定时器0，必须将所有中断控制位开启，中断使能寄存器中“EA”位也被置1。在启动函数里，第一个进程是通过关键字“icall”调用的，这个关键字在OS.h被定义成如下形式：

```
#define icall(ADDR) (*(void(*)())((void*)ADDR))();
```

这时完成了任务的启动，任务将会按照既定顺序执行。

调度函数是定时器0的中断服务程序，它的主要功能是将目前的任务状态压栈、决定下一个要执行哪一个任务和保存任务状态到堆栈中。图3.7是调度函数的流程图。操作系统内核开启定时器0并启动第一个任务以后，当定时器溢出时发生第一次中断，进入定时器0的中断服务程序。这时编译器自动将第一个任务终止前的状态保存在堆栈中，终止时的地址最先进入堆栈，然后按照顺序ACC, B, DPH, DPL, PSW, R0, R1, R2, R3, R4, R5, R6, R7将状态寄存器和工作寄存器全部压入堆栈。压栈完成后，执行中断服务程序。首先判断变量“re_flag”的值，这一变量是用在删除进程中的。关于这个变量的意义和作用将在“删除进程”部分详细讲述。然后程序进入一个循环，这一循环的目的是将任务地址和堆栈内容存入到结构体变量PCB中的数组变量中。因为堆栈是先进后出的，所以堆栈指针最后是指在存储最后入栈的寄存器的数据存储器上。数组变量中存储的顺序是寄存器入栈时的逆序，顺序为R7, R6, R5, R4, R3, R2, R1, R0, PSW, DPL, DPH, B, ACC, DH, DL(PC)。保存好堆栈后需要调度下一个需要执行哪一个任务。因为队列数组是按照优先级已经排列好的顺序，只需要按照队列顺序执行。这时要判断上一个执行的是哪一个任务，如果是队列中最后一个任务，则需要调度第一个任务。知道下一个要执行的任务就可以根据这个任务的优先级来设定定时器0。中断函数结束时，任务状态和地址将自动出栈，所以在中断结束前先把下一个要执行任务的状态存入到堆栈中。出栈时是按照入栈的逆序出栈，在存储时一定注意存储顺序。中断服务程序结束后，立刻执行下一个程序直到他的时间片用完再次进入中断。

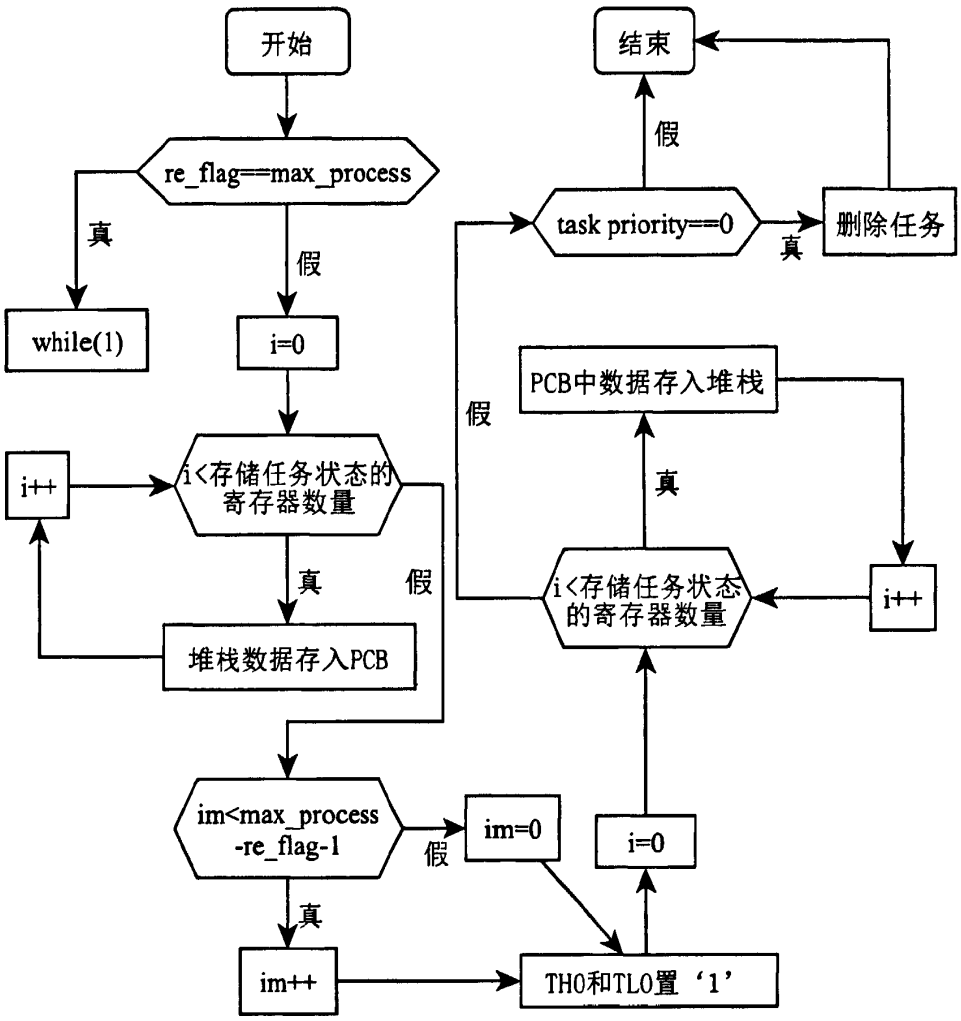

图 3.7　调度函数流程图

Fig.3.7 Scheduling function flowchart

以下为实现启动和调度任务的主要代码段：

```
uchar im; // the queue ID of current implementing task
void process_start()
{
    make_queue();
    TMOD=0x01;
    TH0=-(max_priority*10000/256); //the first task is executed
max_priority time slices
    TL0=-(max_priority*10000%256); //a slice is 10 ms
    ET0=1;
    TR0=1;
    EA=1;
    im=0;
    icall(PCB[que[im]].process_p);
}
uchar im_temp;
```

```
void timer0() interrupt 1
{
    uchar i=0;
    im_temp=im;
    //TH0=-(PCB[que[im]].priority*10000/256);
    //TL0=-(PCB[que[im]].priority*10000%256);
    if (re_flag==max_process)
    {
        TR0=0;
        while(1);
    }

    for (i=0;i<num;i++)
    {
        PCB[que[im]].cur_pcb[i]=*(((uchar *)SP)-i); // save current task
state in PCB
}

    if(im<max_process-re_flag-1)im++;
    else im=0;
    TH0=-(PCB[que[im]].priority*10000/256);
    TL0=-(PCB[que[im]].priority*10000%256);
    for (i=0;i<num;i++)
    {
        *(((uchar *)SP)-i)= PCB[que[im]].cur_pcb[i]; //save next task state
in stack
    }

if (PCB[que[im_temp]].priority==0)remove_process();
}
```

3.3.5 任务删除

当有任务完成后，为了防止再占用时间片和调度程序的执行时间，需要把这个任务删除。消除过时任务对于微操作系统内核是十分重要的。这个删除并不是真正意义的将任务在内存中除去。而是将这个任务在排序队列数组“que”的标号删除，这样在以后的调度中不会再调用到这个任务。任务结束需要使操作系统知道，这就是为什么在任务程序里要求程序结束处标明语句“END(ID of task)”。这一语句使用宏定义定义在文件“OS.h”中，其作用就是先将这一任务的优先级设置为0，然后使任务进入空的无限循环。在调度程序里需要判断上一个执行的任务的优先级，当优先级为 0 时，表明此任务结束，将任务在队列数组中删除。设置无限循环的目的是：在任务需要的时间片没有用完而任务结束时，任务进入空的死循环。如果不这样做，任务结束，中断就会返回到最初的调用函数中重新从第一个任务执行。这样顺序就会发生混乱，导致调度不能正常进行。

在这一模块里，设置了一个变量“re_flag”。每当删除一个任务，这一变量就加一。随着任务被删除，这个变量累加。操作系统内核执行到调度程序时，判断

这个变量是否与任务数量相等。如果相等，说明所有任务已经执行完毕。这时整个操作系统进入空的无限循环中。

3.4 进程间通信

为了提高计算机系统的效率. 增强计算机系统内各种硬件的并行操作能力. 操作系统要求程序结构必须适应并发处理的需要. 为此引入了进程的概念。进程是操作系统的核心，所有基于多道程序设计的操作系统都建立在进程的概念之上。目前的计算机系统均提供了多任务并行环境. 无论是应用程序还是系统程序. 都需要针对每一个任务创建相应的进程。进程是设计和分析操作系统的有力工具。然而由于不同的进程运行在各自不同的内存空间中. 一方对于变量的修改另一方是无法感知的. 因此. 进程之间的信息传递不可能通过变量或其它数据结构直接进行，只能通过进程间通信来完成.并发进程之间的相互通信是实现多进程间协作和同步的常用工具. 具有很强的实用性，进程通信是操作系统内核层极为重要的部分。在计算机系统中，进程间通信（IPC）是为了一个或多个处理器中的不同进程交换数据。进程可以被运行在通过网络连接的多个处理器上。IPC 机制区分为多种方法，有消息传递、同步、共享内存和远程过程调用。

消息传递是一种常用的进程通信方式。使用这种方式，进程可以发送和接收消息（消息可以是多个字节、复杂的数据结构体，甚至是程序段）给其他进程。通过等待消息，进程也可以实现同步。

本操作系统使用消息传递进行进程通信，如图 3.8 所示。在操作系统中定义一个全局变量数组“message”作为消息传递的缓冲区，发送消息一方先把需要传递的信息发到这个数组中等待接收方接收消息。信息是以字节为单位发送的。数组“message”有 8 个元素，第 1,3,5,7 个元素“message”是传递的信息。第 2,4,6,8 个元素“tag”是前一个元素的标记。标记是用来区分不同消息的，每传递一个消息必须同时发送一个标记值，标记值的取值范围是 1~255。每一个被传递的消息都有一个特定的标记，接收进程就是根据这个标记判断是否是自己需要的信息。标记的初始值设为 0，同时也表示没有信息传递。用户也可以通过更改“OS.h”文件中“mnumber”的值更改这个数组的元素个数，也就是可以传递的最多的消息个数。“mnumber”代表最多可以传递的消息数量。

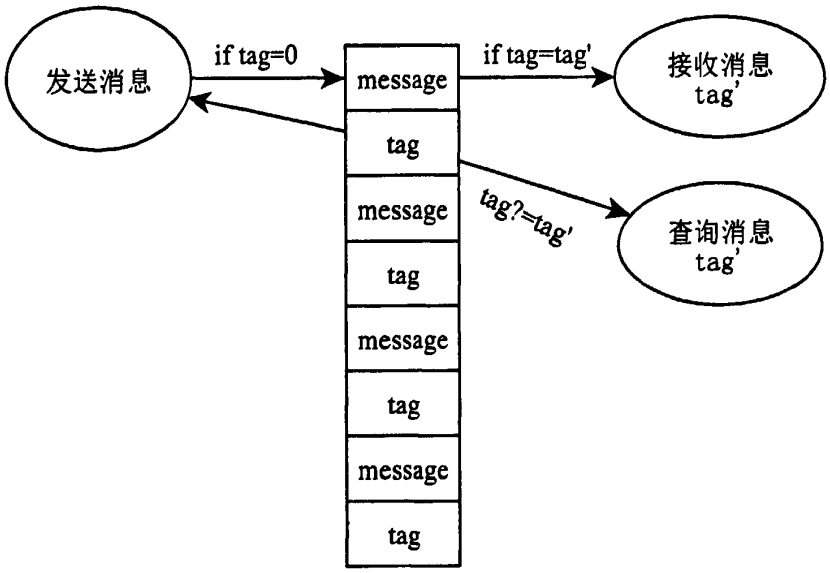

图 3.8　消息传递示意图

Fig.3.8 Message passing schematic

整个消息传递包括三个部分：消息的发送，接收和查询。

发送消息函数是一个返回值为整数的函数，有三个参数，分别是发送消息的进程所在的处理器，发送的信息和它的标记号。第一个参数“发送消息进程所在的处理器”是为以后扩展应用于多处理器的操作系统预留的接口。这个函数的功能是：首先查询消息数组“message”是否有空位置存储消息，就是通过判断消息标记位是否为 0 来实现。如果是 0，说明为空，消息就存放到这个位置并且设置标志位,函数返回 1。如果没有 0 标记位，则返回 0。由用户决定是否继续发送。发送消息的流程图如图 3.9 所示。

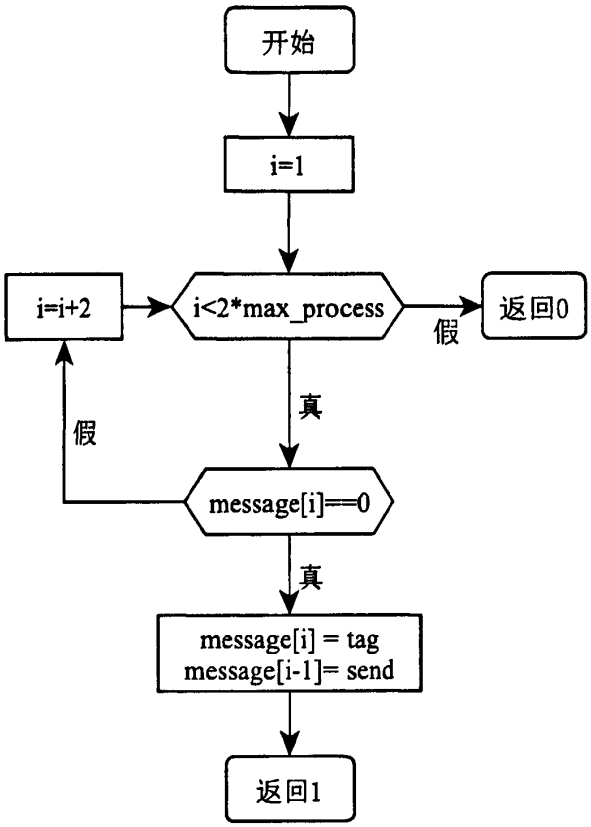

图 3.9　发送消息流程图

Fig.3.9 Sending message flowchart

接收函数有三个参数：接收消息的进程所在的处理器，接收消息存放的变量地址和需要的消息的标号。第一个参数同样是为了扩展多核操作系统预留的接口。接收函数的流程与发送函数的基本相似。它根据传递进来的参数“消息标号”查询缓冲区是否有标记值相同的消息，如果有，将标记位改为 0,保存要传递的信息并且返回 1。如果没有，则返回 0。

在进程通信中为了防止死锁或丢失数据设置了查询消息这一模块。当进程需要给其他进程发送消失时，可以先查询消息数组中是否已经有相同标记的消息等待接收。如果有，返回“1”，没有则返回“0”。用户根据返回值判断是否继续传送数据。

假设接收方需要标号为“5”和“9”的消息，发送方计算标号为“5”的消息的数值再发送，这个数值是在一个循环中被计算得到，所以会不断有标号为“5”的消息被发送而占满缓冲区。标号为 9 的数据无法发送到缓冲区，而接收函数就无法接收到这个消息而不断查询是否有 9。这样造成了死锁。缓冲区无法发送 5 消息，而接收函数也无法接收 5。

以下为实现进程间通信的主要代码段：

```
#include <OS.h>
//#include "at89x52.h"
uchar message [2*mnumber]= {0,0,0,0,0,0,0,0};
uchar message_send(/*uchar dest,*/ uchar send, uchar tag)
{
   uchar i;
   for(i=1;i<2*mnumber;i+=2)
   {
      if (message[i]==0)
      {
         message[i] = tag;
         message[i-1]= send;
         return 1;
      }
   }
   return 0;
}


uchar message_receive(/*uchar source,*/ uchar* rec, uchar tag)
{
   uchar i;
   for(i=1;i<2*mnumber;i=i+2)
   {
      if(message[i]==tag)
      {
         message[i]=0;
         *rec= message[i-1];
         return 1;
      }
   }
   return 0;
}

uchar message_inquire(uchar tag)
{
   uchar i;
   for(i=1;i<2*mnumber;i=i+2)
   {
      if(message[i]==tag)return 1;  //1 stand for having the message in
the array 'message'
   }
   return 0;
}
```

4 操作系统的验证

建立完成基于 8051 单片机单核操作系统的软件模型，通过实现一个 FIR 滤波器对输入信号的响应过程验证这个操作系统内核。创建一个任务作为 FIR 滤波器，在这个任务中以 C 程序编写设计实现的滤波器模型，另一个任务向第一个任务输入数据，由这个滤波器得到冲激响应，将这个值与理论结果相比较。在这一过程中创建任务、任务调度、发送消息、接收消息、消息查询等所有功能均被验证。

4.1 建立 FIR 低通滤波器模型

有限长单位冲激响应（FIR）滤波器是一种离散时间滤波器。滤波器对于冲激信号作为输入信号的响应称作冲激响应。对于有限长单位冲激响应滤波器来说，它的冲激响应会在有限个采样时周期归零。对于具有内在反馈的无限长单位冲激响应（IIR）滤波器来说 ，它的冲激响应也许会无限地持续。N 阶 FIR 滤波器的脉冲响应会持续 N+1 个采样周期，然后归零。

定义 FIR 滤波器输出相对于其输入的差分方程如下：

$$y[n] = b_0 x[n] + b_1 x[n-1] + \cdots + b_N x[n-N] \qquad \cdots(4.1)$$

其中：x[n] 是输入信号，y[n] 是输出信号，bi 是滤波器参数，也称作权重，N 是滤波器阶数。

一个 N 阶 FIR 滤波器右侧会有 N+1 个震荡， 他们也被称作阶数（输入的个数）。例如，5 阶/6 级滤波器。

这个方程也可以写成参数序列 bi 与输入信号的卷积：

$$y[n] = \sum_{i=0}^{N} b_i\, x[n-i] \qquad \cdots(4.2)$$

输出信号就变成当前输入信号与有限个过往输入信号的加权和。图 4.1 给出了一个 FIR 滤波器的框图。

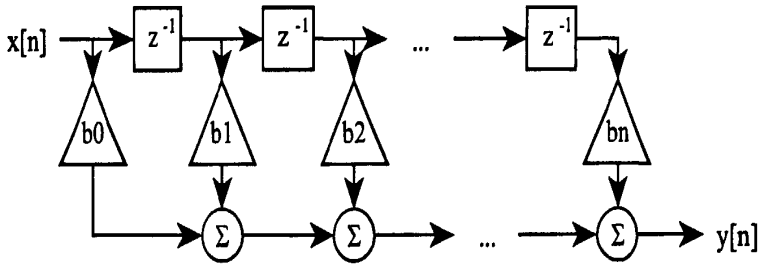

图 4.1　FIR 滤波器框图

Fig.4.1 FIR filter block diagram

以 10 阶低通滤波器的设计为例，低通滤波器的系数计算公式为：

$$c_0 = v_1 \quad \cdots(4.3)$$

$$c_n = \int_0^{v_1} H_d(v)\, cosn\pi v dv = (sinn\pi v_1)/n\pi \quad \cdots(4.4)$$

设采样频率 FN＝10kHz，截止頻率 f1=1kHz ，得到：

$$c_0 = v_1 = 2 * f_1/F_N \quad \cdots(4.5)$$

$$c_n = (sinn\pi v_1)/n\pi \qquad n = 1,2,3,4,5 \quad \cdots(4.6)$$

计算得到：

$c_0 = 0.2$

$c_1 = 0.1872$

$c_2 = 0.1514$

$c_3 = 0.1009$

$c_4 = 0.0468$

$$c_5 = 0 \qquad \cdots(4.7)$$

通过这个10阶低通滤波器的输出信号的展开式为：

$$y_{10} = c_5 * x_1 + c_4 * x_2 + c_3 * x_3 + c_2 * x_4 + c_1 * x_5 + c_0 * x_6 + c_1 * x_7 + c_2 * x_8 + c_3 * x_9 + c_4 * x_{10} + c_5 * x_{11} \qquad \cdots(4.8)$$

得到：

$$y_{10} = 0 * x_1 + 0.0468 * x_2 + 0.1009 * x_3 + 0.1514 * x_4 + 0.1872 * x_5 + 0.2 * x_6 + 0.1872 * x_7 + 0.1514 * x_8 + 0.1009 * x_9 + 0.0468 * x_{10} + 0 * x_{11} \qquad \cdots(4.9)$$

建立好滤波器的数学模型，以下为这个滤波器的软件实现过程，其中任务 1 是滤波器模型，任务 0 向其传递数据。在任务 1 中得到冲激响应，可以由编译软件得到响应结果。

滤波器实现过程主要代码段：

```
uchar x1,i1,a;
void task_0()
{
    x1=0;
    while(1)
    {
        i1=0;
        for (i1=1; i1<12; i1++)
        {
            a=0;
            x1=x1+2;
            while(1)
            {
                EA=0;
                a=message_send(/*0,*/x1,i1);
                EA=1;
                if (a==1)break;
            }
        }
    }
    END(0)
}
```

```
uchar i,*x;
float sum;
void task_1()
{
    code float b[6]={0,0.0468,0.1009,0.1514,0.1872,0.2};
    while (1)
    {
        sum=0;
        for (i=1;i<12;i++)
        {
            while(1)
            {
                EA=0;
                if(message_receive(/*0,*/x,i))
                {
                    EA=1;
                    break;
                }
                else EA=1;
            }
            if (i<7)
            {
                sum=b[i-1]*(*x)+sum;
            }
            if (i>=7)
            {
                sum=b[11-i]*(*x)+sum;
            }
        }
         P0=sum;
    }
    END(1)
}
```

4.2 验证结果

```
Build Output
Build target 'Target 1'
compiling Mpassing.c...
compiling Scheduling.c...
compiling main.c...
linking...
Program Size: data=114.0 xdata=0 code=1892
"OS" - 0 Error(s), 0 Warning(s).
```

图 4.2　编译结果

Fig.4.2 Compilation result

在操作系统中，任务 1 实现一个 10 阶 FIR 滤波器，滤波器的参数由公式(16)得到。任务 0 产生 11 个数据 2、4、6、8、10、12、14、16、18、20、22 作为输入信号，并通过消息传递将这 11 个数据输入到滤波器即任务 1 中。通过滤波器后

得到冲激响应，根据每一步的实现过程检查操作系统的各项功能。图 4.2 为 KEIL 软件编译结果，从这个界面可以看到，编译连接正确且使用 114 字节的片上数据存储器和 1892 字节的程序存储器。

Watch 1

Name	Value
PCB[0].process_id	0x00
PCB[0].priority	0x02
PCB[1].process_id	0x01
PCB[1].priority	0x02
PCB[2].process_id	0x02
PCB[2].priority	0x01
PCB[3].process_id	0x03
PCB[3].priority	0x01
que[0]	0x00
que[1]	0x01
que[2]	0x02
que[3]	0x03

图 4.3　仿真结果(1)

Fig.4.3 Simulation result(1)

图 4.3 是操作系统执行了初始化任务、创建任务过程、并根据优先级排列了任务执行顺序。PCB[]是进程控制块，用来存放进程当前状态包括任务地址、任务标识符、任务优先级、任务堆栈等。系统共有 4 个任务，PCB[4]分别存放了任务 0 到任务 3 的状态。process_id 存储任务标识符，当 PCB 为空时默认值是 4，图中可以看出 4 个 PCB.process_id 的值为 0、1、2、3，所以任务已按顺序存入 PCB。PCB.priority 存储任务优先级，四个任务的优先级分别是 2、2、1、1。任务按照优先级排序后将任务标识符存入队列数组“que”中，根据 4 个任务优先级的大小，队列数组 que 的顺序就是 1、2、3、4（任务标识符）。

Watch 2

Name	Value
message[0]	0x02
message[1]	0x01
message[2]	0x04
message[3]	0x02
message[4]	0x06
message[5]	0x03
message[6]	0x08
message[7]	0x04
im	0x00
re_flag	0x00
sum	0

图 4.4　仿真结果(2)

Fig.4.4 Simulation result(2)

图4.4显示了操作系统执行任务0并且发送4个数据向数据缓冲区的结果。“im”表示当前执行的任务，由图可以看出目前操作系统正在执行任务0。“message”是存储被发送消息的数据缓冲区，其中message[0]、message[2]、message[4]、message[6]存储需要发送的数据，message[1]、message[3]、message[5]、message[7]存储数据的标记值。从图中可以看出，数据缓冲区的标记值都不为0（为0即缓冲区为空），任务0已经向缓冲区发送数据。发送的数据是2、4、6、8并且给每一个数据一个标记值，标记值为1、2、3、4，这个标记值是顺序产生的。

Watch 2

Name	Value
message[0]	0x02
message[1]	0x00
message[2]	0x04
message[3]	0x00
message[4]	0x06
message[5]	0x00
message[6]	0x08
message[7]	0x00
im	0x01
re_flag	0x00

图4.5　仿真结果(3)

Fig.4.5 Simulation result(3)

图4.5中“im”为1，表明操作系统目前正在执行任务1。消息缓冲区中的数据标记值都为0，说明数据已经被接收，目前缓冲区为空，可以再接收新的数据。

Watch 2

Name	Value
message[0]	0x12
message[1]	0x00
message[2]	0x14
message[3]	0x00
message[4]	0x16
message[5]	0x00
message[6]	0x18
message[7]	0x01
im	0x01
re_flag	0x01
sum	14.0712

图4.6　仿真结果(4)

Fig.4.6 Simulation result(4)

由图4.6，这个10阶滤波器需要的11个数据已经全部输入到FIR滤波器中。Message[6]的标记值为1，这个数据是第12个输入缓冲区的数据，10阶滤波器暂时无法接收这个数据，缓冲区保存这个数据直到被接收。“im”值为1操作系统当

前正在执行第一个任务。变量“re_flag”表示已经结束的任务数量，操作系统中任务 2 被设置为空任务并以语句“END(ID of task)”结束，所以“re_flag”的值为 1。图中“sum”是滤波器的冲激响应，操作系统得到的计算结果为 14.0712。根据公式计算如下：

$$y_{10} = 0 * 2 + 0.0468 * 4 + 0.1009 * 6 + 0.1514 * 8 + 0.1872 * 10 + 0.2 * 12 + 0.1872 * 14 + 0.1514 * 16 + 0.1009 * 18 + 0.0468 * 20 + 0 * 22 = 14.0712$$

由上可知得到的结果与理论结果一致。通过以上每一步验证，操作系统中创建任务、任务启动、任务调度、发送消息、接收消息、消息查询等所有功能均达到设计目标。

5 总结与展望

5.1 总结

嵌入式操作系统是一种用途广泛的系统软件，过去它主要应用于工业控制和国防系统领域,负责嵌入系统的全部软、硬件资源的分配、调度工作，控制协调并发活动。嵌入式操作系统在系统实时高效性、软件固态化、硬件的相关依赖性以及应用的专用性等方面具有较为突出的特点。它除具备了一般操作系统最基本的功能，如同步机制、任务调度、中断处理功能等外，还有以下特点：

1) 强实时性。
2) 操作方便、简单、易学易用。
3) 统一的接口，提供各种设备驱动接入。
4) 可装卸性。开放性、可伸缩性的体系结构。
5) 强稳定性，弱交互性。嵌入式系统开始运行后不需要用户过多的干预，这要求负责系统管理的 EOS 具有较强的稳定性。嵌入式操作系统用户接口一般不提供操作命令，它通过系统调用命令向用户程序提供服务。
6) 良好的移植性。

本文主要提出一种基于 8051 单片机的简单嵌入式操作系统结构，实现其内核软件模型主要使用美国 Keil Software 公司出品的开发系统 Keil C51。以上章节细致深入地讲解了嵌入式操作系统的现状、发展趋势、基于 8051 单片机操作系统所需背景知识以及本设计中嵌入式操作系统的结构。操作系统的内核结构设计必须要考虑到微处理器的结构特点。本系统中使用 8051 单片机，它的主要特点是存储空间比较小，编写代码以及使用的算法需要有高的效率，所以这个操作系统内核不具备大型操作系统的完备结构，它是一个微内核结构主要部分包括任务调度和进程间通信。整个操作系统内核按照功能主要分为四个部分，即操作系统文件头文件、主函数、多任务调度和进程间通信。

操作系统文件头文件，包括所有需要的头文件、整个系统中所用到的宏定义、变量、数组、结构体以及对所有函数的声明。这些写在一个头文件可以方便用户修改一些可变参数 ，同时也方便操作系统的移植。

操作系统内核主函数，这一部分的主要任务是对操作系统进行初始化。初始化的过程是给 PCB 中的变量设置初始值。还要将每个用户任务的信息输入到 PCB 中，以达到创建任务的目的。这里面所使用到的创建任务的方法是静态方法建立

任务，即所有任务必须在任务启动前创建，一旦任务开始运行不能再建立新任务。这样做法的缺点就是降低了任务灵活性。但是节省了大量的存储空间，这对于存储容量小的51单片机来讲是很有好处的。这一部分还包括用户任务，用户需要根据系统要求，按格式书写任务。任务优先级可以在范围内修改。

多任务调度，在当前的嵌入式应用中，特别是对一个复杂的嵌入式应用而言，用单任务来实现是不可能的，而只能采用多任务实现，因此任务调度策略的优劣是至关重要的。在操作系统内核中任务调度使用基于优先级的时间片轮转算法。在任务启动前，根据任务优先级确定任务执行顺序，然后使用时间片轮转算法就行调度。时间片轮转算法简单，调度效率高并且占用资源较少。考虑到8051单片机存储空间的问题，这一算法是很有利的。但是，使用这种方法也有缺点。优先级调度可以分为2种：优先级抢占和优先级不抢占。优先级抢占是指当有高的优先级进程被激活后，则立即终止当前运行的进程，使其抢占CPU运行。优先级不抢占是指当有高的优先级进程被激活后，并不立即终止当前运行的进程使其抢占CPU，而是将其放到进程就绪队列中，在当前运行的进程结束后，从进程就绪队列中选择优先级最高的进程运行。操作系统内核中使用的调度方式是按照一种既定的顺序执行任务。所以，所有任务必须在开始一次性写入操作系统并且顺序是固定的，这样就不能实现任务的抢占，降低了系统的实时性。任务删除的目的是加快任务的调度速度。当有任务执行完毕时，删除其在任务执行队列中的位置，系统不再调度此任务，响应调度时间减少，调度速度加快。

进程间通信也是操作系统内核比较重要的部分。本文提出的操作系统内核在进程通信部分使用消息传递机制。分为发送消息、查询消息和接收消息三部分。进程之间的信息传递通过进程间通信来完成，设置数据缓冲区，消息传递利用这个缓冲区完成。

嵌入式操作系统是一个涉及多方面知识技术背景和设计策略要求的庞大系统，并且发展迅速，构建一个完善的嵌入式操作系统不是个人水平所能达到的。论文中仅仅实现了一个简单的操作系统，还有很多需要改进的地方。鉴于作者水平有限，论文定有许多不妥之处，恳请各位专家、老师和同学指正。

5.2进一步研究

对于嵌入式操作系统，现在还处于一个继续研究阶段。本设计中的操作系统还需要进一步改进和完善。

由于应用的需要和硬件条件的限制， 嵌入式操作系统一般都注重占用空间小和效率高等特点。大多数嵌入式操作系统采用了微内核结构，内核只提供基本的

功能。其它的应用组件以系统进程或函数调用的方式工作。因而系统组件都是可增减的，用户可以根据自己的需要选用相应的组件，这样利于操作系统的移植。在移植过程中，首先要根据自己的硬件操作平台在操作系统的移植过程中规定好自己的数据变量， 这是个很重要的前提工作。还要考虑操作系统内核的加载、任务的调度、内存的管理等[29]。本文中的操作系统也考虑到一些移植性因素，在系统头文件中规定了自己的数据变量。同时在头文件中也定义了任务切换时需要用到的寄存器数量，移植操作系统时可以根据硬件平台的工作方式修改这个数值。这个操作系统的移植性能还不够深化，有待进一步研究，希望提出一个移植方案，进行实际移植并测试。

这个操作系统的提出是作为NoC分布式操作系统中的一个节点。为了进一步拓展到NoC中应用，操作系统中预留了三个接口。这三个接口分别在任务创建、发送消息和接受消息部分。任务创建中的接口是任务被创建在哪一个处理器。如果任务所在处理器没有空间创建这个任务，就要通过这个接口寻找满足条件的处理器来创建任务。发送消息接口是发送消息进程所在的处理器。接收消息接口是接收消息的进程所在的处理器。这两个接口主要是为了不同处理器之间消息的传递，当发送进程和接受进程不在同一个处理器时，需要这两个接口寻找处理器位置实现数据的传递。下一步工作希望通过路由器与CPU端口相连在处理器之间进行数据通信，实现NoC上分布式操作系统。

参考文献

[1] 孙娟. 嵌入式操作系统的开发. 信息时代. 2009 年第三期. 59-62.

[2] 吴光文，周航慈. 51 单片机操作系统开发中的问题与技巧. 单片机与嵌入式系统应用. 2004 年 11 期. 75-76.

[3] Kumar, S., Jantsch, A., Soininen, J.-P., Forsell, M., Millberg, M., Oberg, J., Tiensyrja, K., Hemani, A. A Network on Chip Architecture and Design Methodology. Proceedings of the IEEE Computer Society Annual Symposium on VLSI，2002. 105-112.

[4] Benini, L., De Micheli, G. Networks on Chips: A New SoC Paradigm. IEEE Computer；2002，35(l). 70-78.

[5] 郑臻炜. NoC 片上分布式通用操作系统设计.浙江大学. 2008.

[6] C. Jiang, G. Zhang, C. Huang. Research of embedded operating system based on multi-core processor. Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on. ICCSIT 2010. Vol.8. 641-643.

[7] J. Chen, C. Young, D. Chang, G. Huang, C. Ke, S. Yen, T. Kuo. Building Multi-kernel Embedded System on PAC Multi-core Platform. QSIC 2010. 465-472.

[8] Kwok, T.T.-O., Yu-Kwong Kwok. On the Design of an SoPC Based Multi-Core Embedded System. CISIS 2008. 621-626.

[9] Guerin, X., Petrot, F. A System Framework for the Design of Embedded Software Targeting Heterogeneous Multi-core SoCs. ASAP 2009.153-160.

[10] 彭玉灵. 嵌入式实时系统及中国 RTOS 的发展. 四川大学学报. 2004 年 10 月. 第 41 卷.

[11] 嵌入式操作系统在中国的发展现状及前景. 信息技术与应用. 2002 年第 6 期. 6-9.

[12] Jaka Simsic, Simon Teran. 8051 Core Specification. 2001.

[13] 司新生. 嵌入式操作系统任务调度. 科技信息.2010 年第 19 期. I0045-I0045,I0159.

[14] 张尧学，史美林，张高. 计算机操作系统教程. 第 3 版. 清华大学出版社. 2006 年 10 月.

[15] 高明伦，杜高明. NOC 下一代集成电路主流设计技术. 微电子学. 第 36 卷第 4 期. 461-466.

[16] 都政. NOC 关键技术研究与实现. 国防科学技术大学. 2007.11.

[17] 刘有耀，杜慧敏，韩俊刚. NOC 关键问题. 计算机科学. 2008Vol.35No.2. 13-15.

[18] Dumitras, T., Marculescu, R., On-Chip stochastic communication. Design, Automation and Test in Europe Conference and Exhibition. 2003. 790-795.

[19] Pirretti, M., Link, G.M., Brooks, R.R., Vijaykrishnan, N., Kandemir, M., Irwin, M.J.. Fault tolerant algorithms for Network-On-Chip interconnect. Proceedings of the IEEE Computer society Annual Symposium on VLSI, 2004. 46-51.

[20] 杨晓强，高晔. NOC 系统设计的研究. 微电子学与计算机. Vol.25 No.7 July 2008. 176-183.

[21] Li Shang, Li-Shiuan Peh, Jha, N.K.. PowerHerd: a distributed scheme for dynamically satisfying peak-power constraints in interconnection networks. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on . 2006. 92-100.

[22] Saastamoinen, I., Alho, M., Nurmi, J.. Buffer implementation for Proteo Network-on-Chip. Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. II-113 -

II-116 vol.2.

[23] Jingcao Hu, Marculescu, R.. Application-specific buffer space allocation for Networks-on-Chip router design. In Proc. ICCAD, 2004. 354-361.

[24] 魏建军，康继昌，雷艳静. NOC 的平衡设计. 2007 年第 24 卷第 5 期.54-57.

[25] James. Keil C51 开发系统基本知识. Keil C51 使用详解.

[26] 麻旭东, 赵河明, 杨国平. 嵌入式操作系统 Small RTOS51 在 80C51 上的应用. 科学之友: 中旬. 2010 年第 6 期. 43-45.

[27] 包玮. 浅谈基于 8051 的嵌入式实时操作系统. 中国科技博览. 2009 年 11 期. 5-5.

[28] 牛余朋，牛敖其. 51 单片机 C 语言的编程技巧. 电子制作. 2007 年 3 期. 23-15.

[29] 滕凌巧，刘常春，戴琨. 嵌入式操作系统的移植与测试. 平顶山工学院学报. 2003 年 12 卷 4 期. 30-33.

作者简历

教育经历：

2008.9—2011.1	微电子学与固体电子学	硕士	北京交通大学电子信息工程学院
2003.9— 2007.7	电子科学与技术	学士	内蒙古大学理工学院

攻读学位期间发表的论文：

Prototyping Distributed Embedded Operating System for NoC architecture（已录用）

信息科学与技术论坛论文集(EI 检索)

项目经验：

2009.9—2010.2 仪器前端电路设计

2010.3—2010.10 嵌入式操作系统开发

学位论文数据集

表 1.1：数据集页

<table>
<tr><td>关键词*</td><td>密级*</td><td>中图分类号*</td><td>UDC</td><td>论文资助</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="2">学位授予单位名称*</td><td>学位授予单位代码*</td><td>学位类别*</td><td>学位级别*</td></tr>
<tr><td colspan="2">北京交通大学</td><td>10004</td><td></td><td></td></tr>
<tr><td colspan="2">论文题名*</td><td colspan="2">并列题名</td><td>论文语种*</td></tr>
<tr><td colspan="2"></td><td colspan="2"></td><td></td></tr>
<tr><td>作者姓名*</td><td colspan="2"></td><td>学号*</td><td></td></tr>
<tr><td colspan="2">培养单位名称*</td><td>培养单位代码*</td><td>培养单位地址</td><td>邮编</td></tr>
<tr><td colspan="2">北京交通大学</td><td>10004</td><td>北京市海淀区西直门外上园村 3 号</td><td>100044</td></tr>
<tr><td colspan="2">学科专业*</td><td>研究方向*</td><td>学制*</td><td>学位授予年*</td></tr>
<tr><td colspan="2"></td><td></td><td></td><td></td></tr>
<tr><td>论文提交日期*</td><td colspan="4"></td></tr>
<tr><td>导师姓名*</td><td colspan="2"></td><td>职称*</td><td></td></tr>
<tr><td>评阅人</td><td colspan="2">答辩委员会主席*</td><td colspan="2">答辩委员会成员</td></tr>
<tr><td></td><td colspan="2"></td><td colspan="2"></td></tr>
<tr><td colspan="5">电子版论文提交格式　文本（ ）　图像（ ）　视频（ ）　音频（ ）　多媒体（ ）　其他（ ）
推荐格式：application/msword；application/pdf</td></tr>
<tr><td colspan="2">电子版论文出版（发布）者</td><td colspan="2">电子版论文出版（发布）地</td><td>权限声明</td></tr>
<tr><td colspan="2"></td><td colspan="2"></td><td></td></tr>
<tr><td>论文总页数*</td><td colspan="4"></td></tr>
<tr><td colspan="5">共 33 项，其中带*为必填数据，为 22 项。</td></tr>
</table>