HOPE

分布式嵌入式实时操作系统

QNX

侯业勤 张 菁 编译

宇航出版社

分布式嵌入式实时操作系统 QNX

侯业勤　张　菁　编译

宇航出版社

图书在版编目（CIP）数据

分布式嵌入式实时操作系统 QNX/侯业勤，张菁；－
北京：宇航出版社，1999.1
ISBN 7-80144-031-5

Ⅰ. 分… Ⅱ.①侯… ②张… Ⅲ.实时操作系统，
QNX-基本知识 Ⅳ.TP316.2

中国版本图书馆 CIP 数据核字（98）第 26239 号

宇航出版社出版发行
北京市和平里滨河路 1 号（100013）
发行部地址：北京阜成路 8 号（100830）
北京东升印刷厂印刷
新华书店经销

1999 年 1 月第 1 版　　1999 年 1 月第 1 次印刷
开本:787×1092 1/16　　印张:6　　字数:150 千字
印数:1-3000 册　　定价:10.00 元

前　言

QNX 是由加拿大 QSSL 公司(QNX Software System Ltd.)开发的分布式实时操作系统。该操作系统能运行于 Intel ×86、Pentium 等 CPU 环境下，也能运行于 PowerPC、MIPS 等 CPU 环境下。

QNX 是一个分布式操作系统。从用户角度来看，运行 QNX 的局域网就像一台集中式的多用户计算机，QNX 局域网上的每个用户都可使用该网络上他有权使用的任何资源，使用方式与使用本地计算机资源并无区别。

QNX 是一个实时操作系统。它提供用户可控制的、优先级驱动的、急者优先抢占的调度方式。它的自身开销小、上下文切换快，在同样的硬件条件下给实时应用留下更大的余地，因而它在实时控制、通信、多媒体信息处理等对时间敏感的应用领域大有用武之地。

QNX 是一个可嵌入的操作系统。它由微内核和一组共操作的进程构成，具有高度可伸缩性，可灵活地剪裁。最小配置只占用几十 kB 内存。因此，它可广泛地嵌入到智能机器、智能仪器仪表、机顶盒(SET TOP BOX)、通信设备、个人数字助理(PDA)等应用中去。

由于 QNX 是一个符合 POSIX 基本标准和实时标准的操作系统，因而大大方便了在不同系统之间进行应用程序的移植，许多在符合 POSIX 相应标准的其他系统上开发的应用，不需修改，在 QNX 上重新编译后即可运行。

本书主要介绍 QNX 操作系统的总体结构和原理。了解系统的总体结构和原理对于理解 QNX 的实际用途和进行应用开发都是十分必要的。

我们希望通过将这样一个操作系统引进到国内来，能推动社会生产和操作过程控制自动化，促进嵌入式新产品的应用和开发，并在应用高新技术改造传统产业中发挥一定的作用。

北京希望电脑公司是 QSSL 公司 QNX 产品在中国的总销售代理。有关该产品的销售和技术咨询问题请拨电话 010-62628148、010-62579598 进行联系，或直接传真到 010-62579598。有条件的读者也可以用下面的 E-mail 地址与希望公司进行通信联系：

qnx@hope.com.cn

在本书编写过程中，我们得到 QSSL 公司的大力支持，并得到使用他们的技术资料的许可，在此我们表示感谢。

因水平有限，书中可能有不妥之处，欢迎广大读者批评指正。

编译者　1998 年 8 月

目　录

第 1 章　QNX 概述

QNX 是一种运行在 Intel x86 兼容处理器和一些 RISC 处理器(如 PowerPC、MIPS)上的操作系统。

操作系统是一种在计算机上运行的软件。它的主要任务是管理计算机上的系统资源，并为用户提供使用计算机及其外部设备的接口。它的目的是提高计算机的各种资源（如处理器、内存、磁盘、文件等等）的利用率，提高计算机系统的可用性。

现在计算机上运行的操作系统种类繁多，小至资源很少的嵌入式处理器所用的操作系统，大到功能无比强大的巨型机所用的操作系统。这些操作系统从功能到性能各具特色，为不同的计算机硬件环境及应用提供了不同的支持和服务。操作系统为计算机的使用提供了最基本的程序运行环境和接口，是应用软件运行的基础，其运行效率和性能直接影响到计算机应用系统的运行效果。在过去的几十年中，随着计算机、网络技术的发展，操作系统一直处于不断发展和改进之中，现在的操作系统不但追求功能上更强大、更完备和更可靠，而且追求界面上更具友好性和方便性。人们将越来越多的功能加入操作系统中，致使操作系统的体积越做越大。但是，随着应用领域的扩大和增加，人们意识到对操作系统仅单纯地追求功能上的强大是不够的，为了适应不同的应用场合，还需要考虑系统的灵活性、可伸缩性以及可裁剪性。

QNX 是一个很有特色的操作系统软件。它的与众不同的设计思路实现的是一个极为灵活方便、可按照需要随意裁剪的系统，从而开拓了实现计算机系统应用的另一种路子。从下面的描述中，我们能够初步地了解到 QNX 是一个什么样的操作系统。

1.1　多任务和多用户操作系统

对操作系统，如按照在一台计算机上同时可为多少用户服务来区分，可分为单用户和多用户操作系统；如按照其在一台计算机上可同时执行多少个任务（又称进程，是程序在计算机中的一个执行实例，可以被操作系统调度和管理）来区分，可分为单任务和多任务操作系统。

多任务操作系统不都是多用户操作系统，但多用户操作系统一定是多任务操作系统。多用户操作系统的特征是它可以同时与多个用户终端交互作用，随时响应多个用户终端的服务请求。有的操作系统（如 MS Windows 95）虽然允许用户在这种操作系统之上启动多个任务同时运行，但它同时只能与一个用户进行交互，因此，这样的操作系统仍然是一个单用户操作系统。

QNX 是一个遵循 POSIX 1003.1 标准设计实现的操作系统，在许多功能上与 UNIX 操作系统极为相似，既支持多个用户同时访问，也支持多个任务同时执行，因此，它既是一个多任务操作系统，也是一个多用户操作系统。

1.2　并 行 处 理

随着计算机技术的发展，出现了多处理器体系结构的计算机。随之也出现了可提供并行处理功能的操作系统以支持这种多处理器体系结构。在多处理器操作系统的统一控制

下，整个系统可以按照多指令流方式实现作业、任务的并行执行（因此人们称这种操作系统为并行操作系统），以提高系统的计算能力和速度。

在并行操作系统中，多个进程在多个处理器上被“并行处理”，是真正的“同时执行”，每个处理器同时执行着不同进程的指令。而在单处理器多任务操作系统中的“同时执行”，是指用户感觉上的同时执行，实际上是多个进程按某种规则轮流使用处理器。对后者，我们称之为进程的并发执行。

QNX 普通 RTOS 版本提供了一种单处理的操作系统，它的 Neutrino 版本可支持对称多处理。

1.3 分布式系统

多处理器计算机的特点是多个处理器共享内存、紧密耦合。另外还有松散耦合的并行处理结构，特点是每个处理器都有自己专有内存。计算机局域网是松散耦合典型例子。一般的网络操作系统是在松散耦合硬件上的松散耦合软件。而 QNX 与一般的网络操作系统不同，它是在松散耦合硬件上的紧密耦合软件。

QNX 运行在多台计算机组成的局域网上，它使任何一台计算机上的任何一个进程可以和其它任何计算机上的任何进程通信，像与本机进程通信一样；它使任何一台计算机上的任何一个进程可以使用其它任何计算机上的资源，像使用本计算机上的资源一样。用户在这样的环境下工作时，可以将他的任务分散到网络中，交给任何计算机来完成。用户感觉与在一台集中式的多任务系统上工作没有什么不同，只是感到可使用一台的资源更多而已。这样的操作系统叫做分布式操作系统。

利用 QNX 分布式操作系统，可以将许多台廉价的 PC 机连接起来，构成功能强大的计算机群，解决某些使用昂贵的计算机才能解决的应用问题。

利用 QNX 分布式操作系统，很容易建立高可用机制。将每个任务都安排到两台不同的计算机上去执行，只要两台机器不同时发生故障，任务总能得以完成，此时，关闭有故障的机器，不会影响系统运行。有故障的机器修复后，可立即加入到系统中运行。

1.4 分时系统和实时系统

从操作系统能否满足时间敏感的应用要求来区分，可把操作系统分成分时操作系统和实时操作系统。

分时操作系统按照相等的时间片调度进程轮流运行，追求某种意义上的公平。分时操作系统由调度程序自动计算进程的优先级，而不是由用户控制进程的优先级，自动计算进程的优先级是为追求某种意义上的公平而使用的手段。例如，耗时长的进程和耗时短的进程如不加以区别，对耗时短的进程就不公平，所以需要把耗时长的进程的优先级调得低于耗时短的进程的优先级。分时系统适用于一般的科学计算、办公事务处理等不要求在限定的极短时间内得到结果的场合。分时系统完成一个任务所需时间不仅仅取决于任务及计算机系统本身，还受计算机系统中有正在处理的任务的多少的影响，计算机系统中正执行的任务越多，完成其中一个任务所需的时间越长。

实时操作系统能够在限定的时间内执行完所规定的功能，并能在限定的时间内对外部的异步事件作出响应。执行完规定的功能和响应外部异步事件所需时间的长短是衡量实时操作系统实时性强弱的指标。实时操作系统给用户提供控制进程调度的手段，并给用户提供安排实时应用进程的依据。用户可发挥自己的智慧，安排实时应用，使系统在资源有限

的情况下，支持尽可能多的实时应用；或者证明在这样的系统资源状况下，不可能实现某些实时应用，要实现这些应用就必须改善系统资源。

实时应用与分时应用不同。一个分时系统上运行的应用增多时，每个应用只是感到运行得慢了一些而已。而在一个实时系统上，如果实时应用安排不当，可能会造成某些应用或全部应用达不到应用的实时性要求而失败。

分时系统主要应用于科学计算和一般实时性要求不高的场合。实时性系统主要应用于过程控制、数据采集、通信、多媒体信息处理等对时间敏感的场合。

一般分时应用在实时系统上执行起来不会有什么问题，但实时应用却不能简单地由分时系统来完成。与分时系统相比，实时系统更具灵活性，更能适应各种不同场合的应用。

QNX 操作系统对实时应用是理想的，它提供一个实时系统所需要的一切基本要素：多任务、由优先级驱动的急者优先式调度方式和快速上下文切换。对各种实时性要求高低不同的应用，QNX 允许人们根据需要实施特定安排，使各种不同的应用有可能在同一台运行 QNX 操作系统的计算机上得以理想地运行。

1.5 嵌入式系统

在智能化设备、仪器仪表的应用场合，出于对产品体积、成本等因素的考虑，往往要求将计算机控制部分安装于设备内部且占用空间尽可能地小。在这种情形下，处理器一般没有多少可用的内存，更没有可用的外存，而操作系统就装在这有限的内存中(一般在 ROM 中)这种系统人们称之为嵌入式系统。嵌入式系统是智能化设备、仪器仪表的灵魂。

QNX 由一个体积很小的内核及一些可以根据需要进行定制的系统模块组成。QNX 内核一般为几十 kB 大小，即便加上其他必要模块，所占用的空间也很小，且不失其实时、多任务的系统特征。由于它的高度灵活性，使用者可以很容易地对这一操作系统进行定制或作适当开发，来满足自己的实际应用需要。

因此，QNX 不仅允许按需要进行系统配置，用于向一大群用户提供服务，也允许使用几个必要的模块对系统进行配置，产生一个嵌入式系统（即指装在电子产品、机电产品或其它设备内部的占用空间小、可用资源较少的小型实时系统）。

1.6 所遵循的标准

QNX 是一个遵循 POSIX 1003.1 标准及 POSIX 实时标准的实时操作系统。

在过去很长的一段时间里，操作系统主要是由计算机硬件制造商提供的。每个厂商有自己独特的硬件，也配以独特的操作系统,这些操作系统彼此不同，没有什么标准可言。

UNIX 操作系统问世后，以它的高度灵活性、可移植性和简捷性赢得了广泛的欢迎。越来越多的计算机制造商放弃了专有的操作系统，转而采用 UNIX 操作系统。这为用户使用统一的命令操作计算机、使用统一的编程接口开发应用软件带来了很大方便。理想地说，人们可以只学习和熟悉一种操作方法或一种开发环境就可以在任何厂家的计算机系统上工作了。

然而，问题远不像想象的那么简单。在 UNIX 的发展过程中，由于种种原因，产生了许多相互不完全兼容的 UNIX 版本。因此，应用软件的开发者必须了解特定的 UNIX 版本所提供的环境来进行开发，而且在一个环境下开发的软件也不能不加修改地拿到不同版本的环境下去运行，虽然它们都是 UNIX 操作系统。

针对这一问题，许多厂家和组织都呼吁并尝试建立一个统一的操作系统标准。如 AT&T

（美国电话电报公司）、OSF（IBM、DEC、HP 等厂家组织的开放软件基金会）、IEEE（国际电子电气工程师协会）、ISO（国际标准化组织）都开展了操作系统标准化的研究，并提出了许多标准，如 SVR4、OSF/1、POSIX、ISO/IEC IS9945-1 等。

此处，我们简要介绍一下 POSIX（即可移植的 UNIX 操作系统接口）标准。POSIX 包含了一组子标准，从 POSIX 1003.0 到 POSIX 1003.13（其中一些子标准还包含下属标准，如 POSIX 1003.1a、POSIX 1003.1b、POSIX 1003.4a 等），现列表如下：

POSIX 1003.0	POSIX 开放系统环境指导，是对本系统标准的概述
POSIX 1003.1	系统应用程序编程接口（API），C 语言使用的系统服务
POSIX 1003.2	外壳和工具
POSIX 1003.3	测试和验证
POSIX 1003.4	实时和线程
POSIX 1003.5	ADA 语言编程接口
POSIX 1003.6	系统安全性
POSIX 1003.7	系统管理
POSIX 1003.8	网络，包括： 透明文件访问 独立于协议的网络接口 远程过程调用（RPC） 依赖 OSI 协议的应用程序接口
POSIX 1003.9	FORTRAN 语言的编程接口
POSIX 1003.10	超级计算所用的应用程序环境（AEP）
POSIX 1003.11	AEP 事务处理
POSIX 1003.12	图形用户接口
POSIX 1003.13	实时系统的分级

其中，POSIX 1003.1 是最基本的标准，它包括如下内容：

- 名词及概念的定义
- 数据结构、数据头文件、环境变量的规格说明
- 与语言无关的系统服务
- C 语言使用的系统服务

POSIX 只提供应用程序源代码级的可移植性，而不提供任何形式的目标代码级的可移植性。POSIX 只定义数据结构及系统服务的标准，对于如何实现这些服务不加限制。因此，可用许多不同的方式实现 POSIX 标准。这一标准不仅在 UNIX 系统上能实现，在其它系统如 VMS、CTOS、MS Windows NT 等非 UNIX 操作系统上也能实现。

QNX 遵循 POSIX 基本标准和实时子标准，在系统实现上与 UNIX 有很大的不同。因此，QNX 虽然外观很像 UNIX，但并不属于类 UNIX 操作系统。

与实时系统有关的 POSIX 子标准有：POSIX 1003.1 的涉及实时的 API(POSIX 1003.1a)、POSIX.4、POSIX.13。

综上所述，QNX 是一个多任务、多用户、实时、分布、可嵌入、符合 POSIX 标准的操作系统。它有着广泛的用途。另外，QNX 是可运行在 PC 兼容的计算机上的操作系统，PC 机的广泛使用和低廉价格更使 QNX 具有广阔的市场前景。

第 2 章　QNX 的系统特点

2.1　微内核结构

大多数操作系统至少被划分为内核层和用户层两个层次。内核提供基本功能部分，如建立和管理进程，提供文件系统，管理设备等——这些功能以系统调用方式提供给用户，系统调用是用户层使用内核层功能的唯一接口。应用程序可以在自己代码中使用系统调用，来实现对系统的访问或对内核功能的调用。系统调用在应用程序源代码中，看上去就像一般的库函数一样，只是在编译链接时，系统调用不像库函数那样把自己的代码嵌入到应用程序的可执行代码中去。

许多操作系统的内核不以进程方式工作。应用进程通过系统调用来请求内核的服务。系统调用的代码在内核态执行，并在系统调用返回后才继续执行应用程序代码；在进程进行了系统调用，等待系统调用返回时，将暂时不参加进程调度，只有当它们从内核服务退出（即系统调用返回）后，进程才被调度程序调度。

在设计操作系统时，哪些功能放入内核是由设计者决定的。运行时，内核代码和应用程序代码的分界是由硬件提供的保护机制决定的。内核在应用进程不可访问的地址空间中运行。一些特权指令，只有在内核态下才可以被执行。这就保证了内核的安全。

系统调用通常通过一个硬件自陷的指令（trap）实现，它改变 CPU 的执行方式和地址空间的映射，进入内核态。

早期的 UNIX 内核是一个代码量很少的程序组，它只提供操作系统的其他服务所必需的最小功能。但这种传统在随后的许多版本中并没有被继承下来，越来越多的功能被引入内核，操作系统的内核变得越来越大。随着内核的增大，其所提供的系统调用也越来越多。对一般常用的功能，用户进程只需执行这些系统调用即可得到相应的服务支持。利用系统调用提供各种服务的好处是这种方法十分快捷和安全。但内核变大了，所占用的资源就多，剪裁起来也比较麻烦——每增加或删除一些成分时都必须重新编译链接内核，每修改一次系统配置都要重新启动计算机。

QNX 由一个体积很小的内核及一些负责系统管理的共操作进程组成。如同图 2-1 所表明的那样，QNX 的系统结构中包含了若干个管理器，各管理器之间、管理器和其“指挥者”(内核)之间都是彼此可共操作的。这样的结构，使 QNX 系统看上去具有小组型的结构，而不是具有层次型的结构。

QNX 提倡把一个任务划分成多个子任务。每个进程执行一个子任务，由协同操作的进程组共同完成整个任务。这样做不仅能简化程序设计，还能充分利用系统资源，尤其是便于把任务分布在多台机器上并行执行。

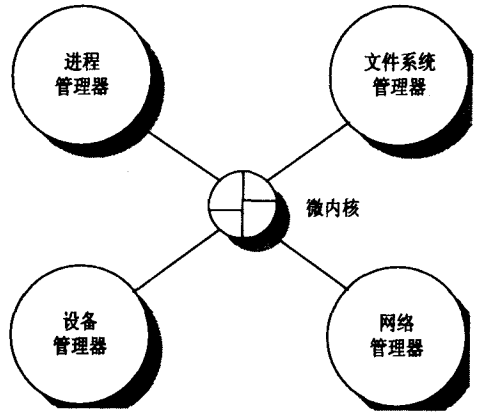

图 2-1 QNX 微内核协调系统管理器之间的工作

对任何一种操作系统来说，其内核就是它的心脏。在有些操作系统中，“内核”包含了如此之多的功能——可用于所有目的和所有用途，以致可以认为该内核本身便是一个完整的操作系统。例如，BSD Unix 系统的内核包括了进程管理、文件系统、设备管理、通信等功能，而在用户层只提供命令语言（shell 脚本语言）和实用（应用）程序。

但是，QNX 的微内核却不同，它是一个真正的内核。首先，和其它实时运行的程序核心一样，QNX 的微内核本身很小；其次它只被用来执行几种最基本的功能。QNX 微内核负责：

- 进程间通信——微内核负责监督管理消息路由，还负责管理另外两种形式的进程间通信：代理和信号。
- 低层的网络通信——微内核负责传输发向其它节点上的进程的消息及接收从其它节点上的进程发来的消息。
- 进程调度——微内核的调度程序决定下阶段将执行哪一个进程。当一条消息或一个中断等事件导致一个进程改变状态时就会使调度程序开始工作。
- 第一级中断处理——所有硬件中断和故障首先由微内核来确定路由，然后递交给相应的驱动程序或系统管理器。

注意： 在 QNX 系统中，微内核本身是不可被调度的，它的执行直接源于进程或硬件中断对内核进行的各种调用，因此，微内核不是一个进程。

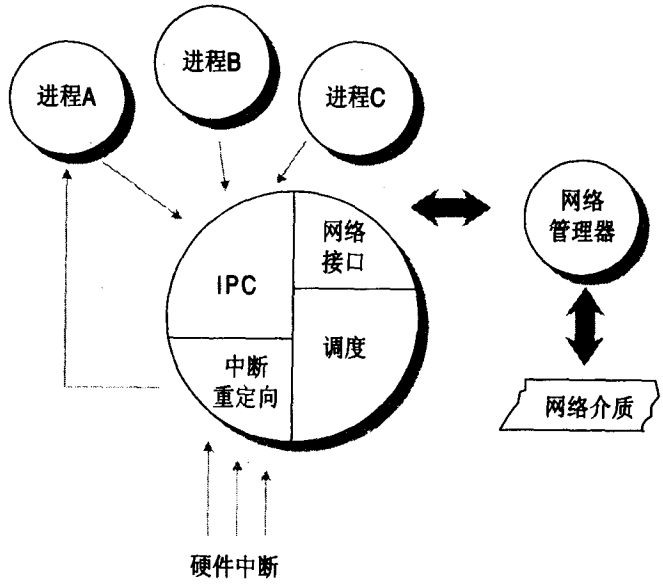

图 2-2　QNX 微内核的内部结构以及与外部的联系

2.2　系 统 进 程

除了由微内核提供的那些服务外，所有的 QNX 服务都是通过标准的 QNX 进程来提供的。一个典型的 QNX 配置拥有如下系统进程：

- 进程管理器（Proc）
- 文件系统管理器（Fsys）
- 设备管理器（Dev）
- 网络管理器（Net）

2.2.1　系统进程和用户进程

QNX 的系统进程实际上与用户所写的程序没有什么不同，甚至没有自己私有的或隐藏起来的以致用户进程不能使用的界面。

正是这种结构为 QNX 带来了无与伦比的可扩展性。在一个 QNX 系统中，由于大多数 OS 服务是通过标准的 QNX 进程提供的，因此，扩展 QNX 本身是一件极为简单的事情只需编制出能提供新服务的新程序即可。

事实上，操作系统和应用程序之间的界线现在已变得十分模糊。就 QNX 来说，系统服务程序和应用程序之间，唯一真正的区别是操作系统服务程序的工作是为应用程序管理资源，并为应用程序提供各种相关的系统服务支持。

假定你现在手头正在编制一个数据库服务器程序，你可能会问道，像这样的进程应算作哪一类（系统进程还是用户进程）？

就像一个文件系统接收打开文件和读写数据请求（QNX 消息）一样，数据库服务器也同样接收打开数据库和读写数据库记录请求。虽然发给数据库服务器的请求在内容和形式上都会更复杂一些，但这两种服务器有十分相似的地方，即：它们都提供一组（通过消息

来实现的）原语来支持对资源的存取，并且它们都是独立的进程，可由用户来编写，可依据需要来启动。

对一个数据库服务器来说，你可以在某一次安装中将它看作一个系统进程，而在另一次安装中将它看作一个应用进程。实际上，这样的进程被看作系统进程还是被看作应用进程，并没有太大关系，重要的是，QNX 允许它在系统中独立地运行，而不必对操作系统的标准成份作任何修改。

2.2.2 设备驱动程序

设备驱动程序是专门支持特定硬件的进程，它能将其他进程从驱动硬件的繁琐细节中解脱出来。既然驱动程序在系统中是作为标准进程来启动的，在 QNX 中增加一个新的驱动程序不会影响操作系统的其它任何部分，你的 QNX 环境所需作的唯一改变是实际地启动新驱动程序。

设备驱动程序启动后，首先进行初始化，然后做下列两件事之一：

- 使自己变成有关系统进程的扩充部分，自身进程结束运行。
- 维持自己原有的进程标识符，保持自己作为标准进程继续运行。

2.3 基于消息的进程间通信（IPC）

作为典型的实时多任务环境，当几个进程需要并发地运行时，操作系统必须提供相应机制来帮助进程进行相互间的通信。

对于一个依靠多个共操作进程来进行工作、并由不同的进程分别处理整个任务中不同子任务的应用来说，进程间通信将是设计中极为关键的一点。

QNX 提供一个简单而强大的 IPC 功能集合，大大地简化了建立共操作进程过程中的应用开发工作。

QNX 是一个基于消息传递的操作系统。QNX 的强大、简单和完美，主要原因是它把消息传递方法全面地融进了整个系统之中。

在 QNX 中，一条消息是将若干个字节的数据封装在一起组成的数据包，它可以从一个进程被传递到另一个进程。QNX 对消息的内容不附加任何解释信息，因此，消息中的数据仅对消息的发送者和接收者具有意义。

QNX 的消息传递机制不仅允许进程之间互相传递数据，还为多个进程的运行提供了一种同步手段。在消息的发送、接收和响应过程中，进程的状态将随之不断发生变化，并影响着其下一步运行的进入时间以及运行时间的长度。通过掌握进程的状态和优先级，微内核可以有效地对所有的进程进行统一调度，以实现对 CPU 资源的最大程度的利用。

对实时和其它重大关键应用来说，由于实现这类应用的进程之间通常具有很强的关联性，以致往往需要一种具有高可靠性的进程间通信手段。QNX 特有的消息传递机制有助于系统为应用程序提供更好的运行效率和更高的可靠性。

2.4 使用 QNX 组建网络

局域网为网络中的计算机提供了一种最简单形式的共享文件和外围设备机制。而用 QNX 组建的网络所能做的事情则远比这个多得多——它将整个网络组织成一个具有单一性及同一性的资源集合。

在这一网络中，任何计算机上的任何进程都可以直接使用任何其它计算机上的资源。从应用的角度来看，本地资源和远程资源之间并无区别，应用程序在存取远程资源时不需要在程序中增加任何特殊代码。虽然在实际运行中，程序确实需要使用一段特殊代码来判断自己所要存取的文件或设备究竟在本地计算机上还是在网络中的其它计算机上，但 QNX 使编写程序的用户不必关心如何增加这些代码和使用这些代码。

只要拥有相应权限，用户可以存取网络中任何计算机上的文件，使用网络中的任何外围设备，或在网络中的任何计算机上运行应用程序。网络中的任何进程之间也可以用同样的方式进行通信。QNX 的无处不在的消息传递式的进程间通信机制是在充分考虑了这种交互通信形式和网络透明性的基础上形成的。

2.4.1 具有单计算机特点的网络

从一开始，QNX 的设计者就是按照使 QNX 成为一种专门用于网络的操作系统这样一种想法来设计 QNX 的。从某些方面来看，一个 QNX 网络看上去更像一台大型计算机设备而不是一群被连接起来的微机，用户仅知道他的应用程序有大量的资源可用。但是，和大型计算机不同的是，QNX 提供了一个响应速度更快的环境，这是由于它可以充分地利用每个节点的资源和计算能力来满足每一个用户的需要的缘故。

比如，在一个过程控制环境中，PLC（可编程逻辑控制器）和其它实时 I/O 设备可能比其它非关键进程或程序（如字处理程序）需要更多的资源，QNX 网络能提供足够的响应速度同时支持这两种类型的应用——它可以使用户随时随地得到其所需要的计算能力，并且不防碍桌面应用的并发执行。

2.4.2 具有灵活的结构的网络

用户可以采用各式各样的硬件设备和基于工业标准的协议来组建一个 QNX 网络。由于所用的设备和协议对应用程序和用户是完全透明的，因此 QNX 允许在不对操作系统作任何改变的情况下随时引进新的网络结构。

QNX 网络的每个节点都将被分配给一个唯一的编号以作为它的标识符。这个编号是区别一个 QNX 系统是以网络形式运行着还是以单机操作系统形式运行着的唯一可见手段。

这种高度的透明性仅说明了 QNX 与众不同的消息传递体系结构的一个方面。在许多系统中，网络通信、进程间通信或消息传递这类重要的功能往往被建立在操作系统的上一层中，而不是直接集成在内核中，这种做法常常会带来问题：即产生低效率的、非统一的接口，使系统既要解决一般进程之间的通信问题，又要考虑如何使铁板一块的神秘内核与外部通过专用接口进行通信的问题。

另一方面，QNX 笃信这样的原则，只有高效率的通信才能产生高效率的操作。因此，消息传递构成了 QNX 体系结构的基础，并改善了整个系统（无论是在一台计算机内的还是用一英里长的铜缆连接起来的两台计算机之间的）所有进程之间交互通信的效率。

在后面的讨论中，将进一步说明 QNX 的结构和功能。

第 3 章　进程间通信

3.1　进程间通信的方法

当多个进程在一台计算机上同时执行时，它们之间会形成一种关系。即使是从程序上看不出有联系的几个进程，在同时运行时相互之间也会产生联系。因为它们都要使用机器上的资源，而这些资源是有限的，需要大家按照某种秩序来分别使用。当某资源被其他进程占用了，下一个要使用该资源的进程就要等待，待该资源被释放后再使用。那些在设计时就要求彼此之间共操作的多个进程，在运行时更要频繁地通信、传递数据、同步相互的动作。

进程间通信的目的有三种：

- 同步有关进程的执行
- 改变进程的执行方向
- 在进程间传递数据

所谓同步进程的执行，一方面是指一个进程要在获得另一个进程的某些执行结果后才能向下执行，另一方面是指多个进程互斥地使用共同资源。同步有关进程执行的方法有：二值信号灯、多值信号灯、文件锁、记录锁等。

改变进程的执行方向是指从进程的外部改变进程正在执行的顺序，让进程去执行另外的程序。改变进程执行方向的手段有信号、软中断和事件通告。

进程间传递数据的方法有消息、消息队列、共享内存、管道、命名管道（FIFO）等等。多数传递数据的方法需要使用同步手段协调不同进程对数据的读写。

可以证明，诸多的同步方法是能够相互表达的，诸多的数据传递方法也是能够相互实现的，只不过某种方法在某种场合用起来更加方便而已。在实现一个系统时，总是选择某些进程间通信的方法作为一个系统内部通信的基本方法，在内核中实现，而在内核外实现进程间的其他通信方法。

3.2　QNX 进程间通信

QNX 微内核支持三种基本类型的进程间通信：消息、代理和信号。下面是对这三种通信方法的说明：

- 消息——QNX 中进程间通信的基本形式。使用消息可提供共操作进程间的同步通信，进程在发送消息时需要有消息的接收者，并且需要有对消息的回答。
- 代理——一种特殊的消息通信形式，专门用于事件通知，并且此时发送进程不需要与接收进程发生交互作用。
- 信号——进程间通信的一种传统形式，用来支持进程间的异步通信。

3.2.1　通过消息进行进程间通信

在 QNX 中，一个消息是一个将若干字节数据封装在一起组成的数据包，可以从一个进

程同步（即发送进程将一直等待到对方进程接收）地传送到另一个进程。QNX 不对消息的内容附加任何解释信息，因此消息的内容仅对消息的发送者及接收者有意义。

（1）　消息传递原语

进程可使用如下 C 语言函数来直接与另一个进程进行通信。

表 3-1　消息通信用的 C 语言函数

C 函数	功能
Send( )	发送消息
Receive( )	接收消息
Reply( )	回答发送进程

这些函数既可在网络环境中使用也可在本地环境中使用。

注意：进程之间除非希望直接进行通信，一般情况下不需要使用 Send() 、Receive() 和 Reply() 函数。由于 QNX 的 C 函数库是建立在消息传递的基础之上的，所以当进程使用像管道（pipe）这样的标准服务时将以间接方式使用消息传递机制。

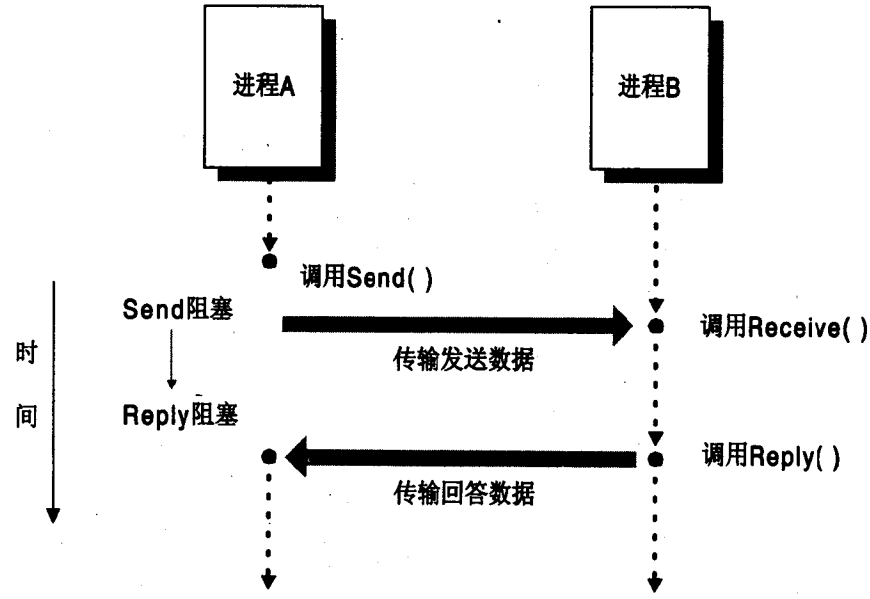

图 3-1　进程 A 向进程 B 发送一条消息，B 进程接收该消息，进行处理并给予回答

图 3-1 表明所发生的一个简单的事件序列：两个进程（进程 A 和进程 B）使用 Send()、Receive()和 Reply()相互进行通信。

1）　进程 A 通过发送 Send()请求向进程 B 发送了一条消息。这时，进程 A 进入调用 Send 后的 SEND 阻塞状态，直到进程 B 调用了 Receive()来接收消息。

2）　进程 B 调用了 Receive()，收到进程 A 的消息。此时进程 A 的状态改变为 REPLY 阻塞状态，直到收到一条对该消息的回答。这期间进程 B 不停止运行（注意，如果进程 B 在进程 A 发送消息前就已经调用了 Receive()，进程 B 在消息到达前将处于 RECEIVE 阻塞状态。在这种情况下，进程 A 则在发送消息后立即进入 REPLY 阻塞状态，而进程 B 则解除阻塞重新运行）。