第36卷 第15期 计 算 机 工 程 2010年8月
Vol.36 No.15 Computer Engineering August 2010
·软件技术与数据库· 文章编号：1000—3428(2010)15—0051—04 文献标识码：A 中图分类号：N945

# 面向实时嵌入式操作系统的进程机制

周 昕[1]，傅 鹂[1]，黄海伦[2]
(1. 重庆大学软件学院，重庆 400044；2. 中兴通讯股份有限公司，深圳 518004)

**摘 要：**面向通信领域的嵌入式程序必须在资源受限的硬件环境中应对不断增加的通信业务，单纯依靠商用嵌入式操作系统的任务机制已不能提供足够的业务并行度和吞吐量。针对该问题，基于嵌入式操作系统任务机制提出一种更小粒度的进程解决方案，相对于任务对象，使用进程作为执行单元不仅内存资源占用少，且进程之间切换速度快，系统可以支持大量进程并行。该进程机制能够提供有效的系统监测和故障诊断手段，从而保证系统的健壮性。

**关键词：**嵌入式系统；任务；进程；并行处理

# Process Mechanism for Embedded Real-time Operating System

ZHOU Xin[1], FU Li[1], HUANG Hai-lun[2]
(1. School of Software Engineering, Chongqing University, Chongqing 400044;
2. Zhongxing Telecommunication Equipment Co. Ltd., Shenzhen 518004)

**【Abstract】**Embedded software of telecommunication area must deal with increasing communicating businesses under restricted hardware environment, and only depending on the task mechanism of commercial embedded operating system can't supply enough parallel processing ability and throughput any more. To solve the problem, this paper proposes a new solution scheme, in which the smaller executing units called process are designed and scheduled based on tasks. Compared with tasks, processes occupy less memory and cost less time to be switched, so system can support large number of running processes. It provides effective ways on monitoring system resources and diagnosing system failures, which brings robustness.

**【Key words】**embedded system; task; process; parallel processing

## 1 概述

通信领域要求嵌入式环境下的应用程序具备较高的并行处理能力，由于目前主流商用嵌入式操作系统如 pSOS、vxWorks 等只支持以任务为最小执行单元的应用，因此要提高程序并行处理能力就必须创建足够多的任务。问题在于维护一个任务对象需要耗费较多的内存和 CPU 资源，尤其是任务切换带来的时间开销已经逐渐成为性能瓶颈。此外，由于缺少对任务对象本身和它使用的系统资源进行监控的手段，导致系统异常难以定位，如堆栈溢出、异常死循环、消息队列出错等，直接影响到系统的稳定性和可靠性。

基于进程和线程机制的 Linux 操作系统[1]可以解决业务并行处理问题，但在实时性和稳定性方面不如商用嵌入式操作系统成熟。综合通信行业的特殊需求以及 Linux 操作系统并行处理的优点，本文在任务机制的基础上实现了一种更细粒度的可执行单元，暂取名为进程。由于对进程的操作不涉及内核态行为，因此使用起来比任务更加高效、安全和可控。进程解决方案可在系统资源受限的环境下有效提高系统并发处理能力和系统资源可监控性。

## 2 相关原理

在主流商用嵌入式操作系统里，任务是占有系统资源的最小执行单元和调度对象。它生存于系统提供的一个虚拟、孤立的环境之中，使用或等待系统资源，无需考虑其他任务。多数嵌入式操作系统采用基于优先级抢占的任务调度机制，以满足实时性的需求。

本文提出的进程实质上相当于子任务，它挂在操作系统的任务对象之下，是比任务更小粒度的执行单元。因为进程不是操作系统内核对象[2]，所以没有任何系统调用提供对进程的操作。因此，基于操作系统已有的任务调度机制，本文设计了进程级的调度，以提供对进程行为的控制和状态的切换。定义一个管理若干进程的任务为调度任务，进程受调度任务直接调度，在最上层进程和任务受调度管理模块统一管理。进程之间、进程与任务之间的通信由通信管理模块完成，实现了基于处理器的点对点、组播、广播等通信模式。

### 2.1 进程状态和行为

本文的进程定义为一块代码段和它的执行线索以及独立的消息队列、数据区和栈。进程控制块(PCB)来记录进程当前的所有信息。进程有且仅有就绪、运行、阻塞 3 种状态：(1)就绪(Ready)，进程的消息队列里有消息，等待获取 CPU 资源；(2)运行(Running)，进程正在使用 CPU 处理指令；(3)阻塞(Block)，进程正在等待消息或某种资源。进程由一种状态向另一种状态的跃迁通过消息机制来驱动，3 种状态之

**基金项目：**国家“863”计划基金资助项目“面向通信行业的嵌入式软件开发平台”(2002AA1Z2306)
**作者简介：**周 昕(1984－)，男，硕士，主研方向：软件工程，嵌入式操作系统；傅 鹂，教授；黄海伦，工程师、硕士
**收稿日期：**2009-12-20 **E-mail：**zxcy2003@126.com

间的跃迁过程如图 1 所示。

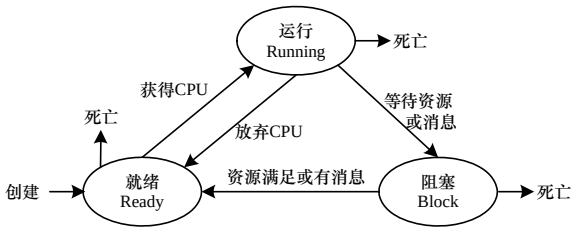


**图 1　进程状态跃迁过程**

调度任务负责控制进程状态的切换，且通过操作就绪进程队列和阻塞进程队列管理该任务下的所有进程，2 个队列均采用 FIFO 排队规则。同一调度任务下最多只能有一个进程处于运行状态。

进程创建后进入启动状态，完成一系列的初始化工作后进入就绪队列。调度任务从就绪队列头摘取进程，将控制权交给进程，进程在消息激励下进行处理，完成状态跃迁，最后把控制权交还给调度任务。如果该进程消息队列没有任何消息，则被挂到阻塞队列尾，否则挂到就绪队列尾，然后从就绪队列头摘取新的进程。如果没有就绪进程，调度任务负责从任务消息队列中提取消息，派发到目的进程的消息队列尾，同时把目的进程从阻塞队列摘除，置入就绪队列尾；若任务消息队列为空则阻塞任务。调度任务派发消息后，继续调度就绪进程。调度任务重复这样的过程，保证每个进程都有平等的运行机会。

### 2.2　进程堆栈操作

为了保证调度任务和进程之间尽量低耦合，采用了独立的进程堆栈，即每个进程都有一块独立的堆栈空间。为了实现这一功能，在开始进程调度前要保存其栈指针，调度结束时要返回该指针。

当进程运行完毕时需要将 CPU 的控制权交还给调度任务，由调度任务重新调度，在进程交还控制权的时候需要对进程堆栈进行处理，分为 2 种情况：(1)正常返回。进程处理完事件后正常返回，此时进程需要取出调度任务堆栈指针，然后恢复调度任务之前的运行环境，让调度任务继续运行。(2)非正常返回。进程在处理事件过程中被更高优先级的任务或者中断抢占后返回，此时需要获取调度任务堆栈指针，恢复调度任务之前的运行环境，然后，保存当前进程运行环境及堆栈，从而在下一次获取 CPU 控制权的时候可以继续执行。由于对进程堆栈的操作和 CPU 类型以及操作系统相关，因此需要针对不同类型 CPU 和操作系统分别提供保存和恢复进程堆栈的指令，图 2、图 3 是基于 X86 架构 CPU 的 WinNT 操作系统对进程堆栈的操作。

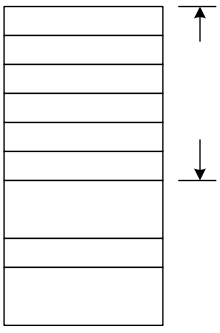

**图 2　保存进程堆栈操作**

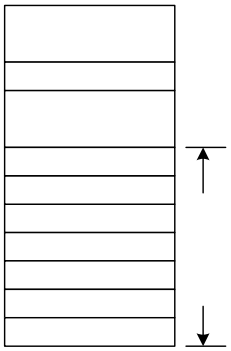

**图 3　恢复进程堆栈操作**

目前主要实现了 X86 下的 WinNT 以及 ARM、PPC 系列 CPU 下的 pSOS、vxWorks、Linux 操作系统的进程堆栈操作。

### 2.3　进程调度模型

一个进程只受某一个调度任务的调度，即进程和调度任务是一对一的关系。任务的调度[3]由操作系统负责，而进程的调度统一由调度管理模块负责。调度管理模块的结构如图 4 所示。

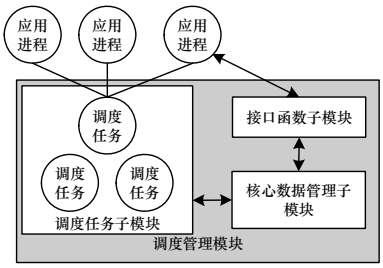


**图 4　调度管理模块结构**

调度管理模块总共由系统启动控制子模块、调度任务子模块、接口函数子模块、核心数据管理子模块、系统监测子模块、打印观察子模块 6 个子模块组成。系统启动控制子模块按顺序完成系统数据结构的初始化、进程的创建、任务启动、启动主控进程，任务与进程的创建与启动信息分别存放在任务配置表以及进程配置表中。调度任务子模块实现对进程的直接调度，负责接受任务消息并派发到目的进程进行消息处理和状态跃迁。接口函数子模块向上层应用提供操作进程对象的接口。核心数据管理子模块负责进程控制块的分配和释放，对与进程运行状态相关的各种队列(诸如就绪队列、阻塞队列等)进行管理。系统监测子模块负责对系统资源的使用情况、进程异常状态等信息的监测。打印观察子模块提供打印输出接口，为定位系统故障提供便利。

具体而言，调度管理模块主要提供以下功能：

(1)在任务调度基础之上构造针对进程的二级调度策略；

(2)依据一定的策略对由信号所引起的进程跃迁过程进行控制；

(3)对由进程所发起的创建、杀死和延时等动作进行支持；

(4)提供获取当前进程标识(PID)的接口、对 PID 及相关数据管理的接口、查询进程信息的接口；

(5)实现对进程的实时和定时监测；

(6)向其他模块提供获取当前进程 PCB 指针的接口；

(7)完成后台或前台应用进程对前台预定义定时器、时区、CPU 占用率高限和低限、打印输出控制等信息的配置；

(8)提供调试钩子函数，打印输出调试信息。

### 2.4 进程通信模型

在多进程并行环境中，进程通信模块被设计出来以支撑系统中所有调度任务下进程之间的通信。它实现了基于多处理机的进程通讯机制，支持处理机内消息广播、处理机间的消息组播，以及处理机内的同步和异步消息、处理机间的异步消息机制。进程通讯模块处理消息的流程如图 5 所示。

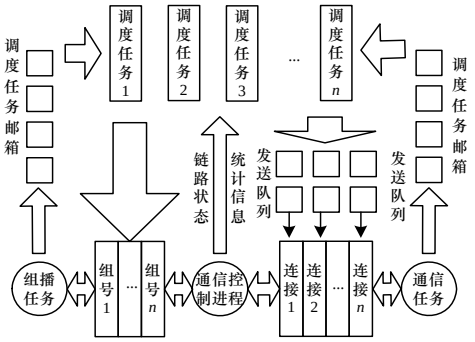


**图 5　进程间通信模块框架和处理流程**

根据通信目端数量的不同将通信处理流程分为以下 4 类：

(1)点对点可靠通信，基于 RUDP 传输协议。

每个需要通信的处理器之间维护一条连接，所有连接组成连接表。进程通过接口发送的消息首先挂在相应连接的发送队列中，发送队列中的消息在该连接发送器的发送窗口控制下采用无等待并包方式发送，发送时采用一个公用的 UDP 发送套接口。UDP 守护任务监测一个公用的接收 UDP 套接口，当套接口读就绪的时候读取消息，并在该消息对应的连接项的接收窗口控制下向上层转发。链接表结构定义如图 6 所示。

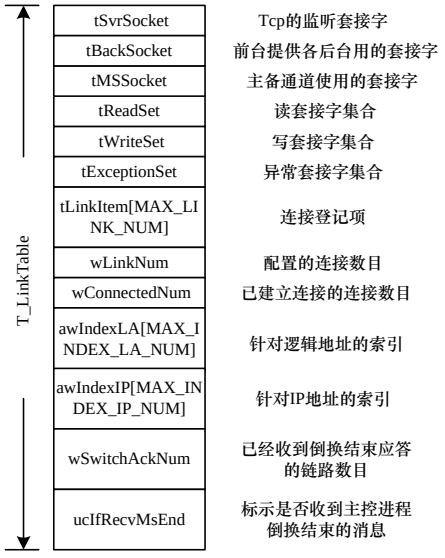


**图 6　链接表结构定义**

(2)点对点不可靠通信，基于 UDP 协议。

使用一个公用的发送 UDP 套接口，发送时指定协议类型为不可靠传输协议，然后根据逻辑地址查找到对应的 IP 地址，直接从 UDP 套接口上发送出去。UDP 守护任务守护在公用的接收 UDP 套接口上，当接收套接口读就绪时接受消息并向上层转发。

(3)对于组播，采用静态分组的方法。

每一个组播组对应一个到组播地址，所有这种对应关系组成分组表。一个处理器可以动态加入或脱离某个组播组。组播任务守护在组播组表的组播套接口上，在套接口读就绪时接收消息并向上层转发。进程直接在组播套接字上发送消息，如图 7 所示。


**图 7　组播分组表结构定义**

(4)对于广播，分组表中有一个特殊项用作广播。广播组成员包括系统内所有处理器，组播任务在该项上守护，进行和上面相同的处理。对于处理器内广播，组播任务只向同一个进程类型的实例分别发送一条消息的拷贝。

#### 2.4.1 同一任务内进程间通信

图 8 是同一个任务中 2 个进程间通信的示意图，同一任务内多进程间通信的情形与其类似。

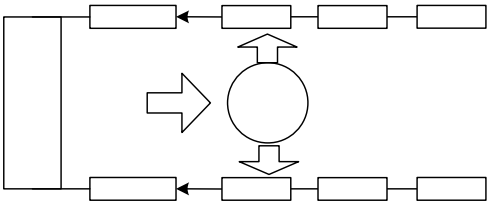

**图 8　同一任务内进程间通信**

每个进程都拥有一个属于自己的消息队列，用于承载消息体。(1)进程向同一任务内进程发送消息，通过内部接口直接将消息挂到另一个进程的消息队列中。(2)当进程向同一任务内进程发送同步应答消息，直接改变目的进程的状态即可。任务内的进程间通信不能通过任务的邮箱进行转发，以避免在邮箱满的时候任务向自己的邮箱发送消息导致任务死锁。

#### 2.4.2 同一处理器内任务间通信

图 9 为同一处理器内任务间通信示意图。

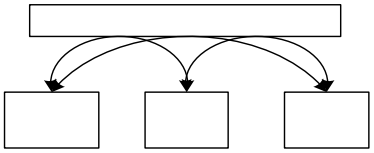

**图 9　处理器内部的任务间通信**

每个任务都拥有一个属于自己的邮箱。(1)进程发送到另一个任务的进程的消息首先发送到目的进程所在任务的邮箱，然后由调度任务派发到目的进程的消息队列中。(2)进程发送给非调度任务的消息同样发送到目标任务的邮箱。(3)非调度任务之间的通信也是通过访问任务邮箱实现。(4)发送异步紧迫消息，将消息发送到目标任务消息队列的头部。(5)发送异步普通消息，将消息发送到目标任务消息队列的尾部。

#### 2.4.3 消息的派发

调度任务从自己的邮箱获取消息，并负责向任务内各进程的消息队列中派发，在派发的过程中根据不同消息的类型对进程状态进行不同处理。(1)派发同步应答消息的时候，将

目标进程转入就绪状态，设置解除阻塞原因为应答到。(2)派发同步超时消息的时候，将目标进程转入就绪状态，设置解除阻塞原因为同步超时。(3)派发延时结束消息的时候，将目标进程转入就绪状态，设置解除阻塞原因为延时结束。(4)派发其他类型消息时，将消息挂到目标进程消息队列尾部，如果目标进程因为没有消息而阻塞，将目标进程转入就绪状态。

## 3 性能及稳定性分析

在 SDRAM 256 MB, vxWorks6.4, Freescale MPC 8323 (333 MHz)环境下，经过长时间分析和记录，统计得到进程与任务在内存资源占用、上下文切换耗时及故障发现率 3 个方面的数据。表 1 为进程机制节余内存情况。

**表 1 进程机制节余内存情况**

| 业务类型 | 任务机制下任务个数 | 进程机制下任务及进程数 | 内存节余比例/(%) |
|---|---|---|---|
| 业务 1 | 9 | 5, 8 | 10 |
| 业务 2 | 20 | 8, 18 | 12 |
| 业务 3 | 29 | 10, 30 | 13 |
| 业务 4 | 38 | 13, 38 | 13 |
| 业务 5 | 51 | 17, 46 | 14 |

从表 1 可以看出，进程对象比任务对象占用更少的内存资源。其中，第 2 列、第 3 列分别记录使用任务、进程机制实现相应业务需要创建的任务数、进程数，第 4 列数据表示进程机制相对任务机制实现相应业务节余内存的比例，该数据通过统计运行状态下系统内存的使用情况得到。

任务或进程切换所消耗的时间主要包括执行保存、恢复任务或进程堆栈 2 个连续动作所用时间。由于对任务的切换涉及操作系统内核态下的行为，且在任务机制下的任务堆栈往往比较大，因此任务切换效率比进程要低。从图 10 可以清楚地看到进程、任务切换操作的时间开销与堆栈大小成正比，且任务切换耗时的增长幅度略高于进程，图中的线条均根据统计测试数据描绘。

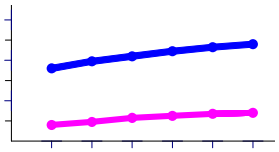

**图 10 任务、进程切换时间开销**

嵌入式程序中隐藏的缺陷通常不容易被发现，一旦触发往往使系统遭受不可恢复的破坏，所以尽早发现它们是一项非常有价值的工作。为了验证进程机制在系统缺陷发现率上的优势，分别基于任务和进程机制开发新的功能模块，在人力、物力资源相等、进度相同前提下，统计出四周之内依靠任务、进程机制提供的诊断手段[4]发现的系统故障数目如表 2 所示。

**表 2 任务、进程机制下故障发现数**

| 故障类型 | 第 1 周 | 第 2 周 | 第 3 周 | 第 4 周 |
|---|---|---|---|---|
| 任务发现 | 3 | 3 | 2 | 2 |
| 进程发现 | 6 | 6 | 6 | 5 |

在稳定性方面，嵌入式操作系统任务堆栈在任务创建时指定，且不能更改，一旦任务堆栈溢出，系统将面临不可预料的严重后果，往往只有复位设备才能恢复系统的正常状态。堆栈溢出不仅对系统稳定运行构成严重威胁，而且通常不容易定位和发现越界的代码。基于对上述问题的考虑，提出了动态增长进程堆栈的策略，为系统长时间在线稳定运行提供有力保障。此外，为提高系统资源可监控性，还提供了实时、定时监控进程对象信息及其使用的系统资源信息的方法。

## 4 结束语

实时多任务嵌入式操作系统[5]使用任务作为可调度的最小执行单元，本文在任务的基础上设计并实现了一种更小粒度、更灵活的执行单元，称作进程。使用进程对象，一方面可以让嵌入式应用程序在资源受限的嵌入式设备中实现高并行度和高吞吐量；另一方面，可以减少对系统任务对象的依赖，通过实时、定时获取进程及其相关系统资源使用情况，及时、准确地发现和解决系统中的隐患和异常，提高系统的稳定性和可靠性。本文提出的进程解决方案已应用在通信领域的 SDH、WDM 网络上，工程检验表明可以很好地满足大量通信业务并发环境下的通信需求，具有较高的应用和推广价值。

## 参考文献


[1] 陈莉君. 深入分析Linux内核源代码[M]. 北京: 人民邮电出版社, 2002.

[2] 屠 祁. 操作系统基础[M]. 北京: 清华大学出版社, 2000.

[3] 潘 清, 张晓清. 操作系统微内核技术研究[J]. 软件学报, 1998, 9(8): 609-612.

[4] 王泽民, 芦东昕, 谢 鑫, 等. 基于 VxWorks 的异常处理的研究和实现[J]. 计算机工程, 2005, 31(13): 90-92.

[5] 符意德. 嵌入式系统设计原理及应用[M]. 北京: 清华大学出版社, 2005.


编辑 陈 文

（上接第 50 页）


[4] Zhang Chun, Naughton J, DeWitt D, et al. On Supporting Containment Queries in Relational Database Management Systems[C]//Proc. of 2001 ACM SIGMOD Int'l Conf. on Management of Data. Santa Barbara, California, USA: [s. n.], 2001.

[5] Goldman R, Widom J. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases[C]//Proc. of the 23rd Int'l Conf. on Very Large Data Bases. Athens, Greece: [s. n.], 1997.

[6] Kaushik R, Shenoy P, Bohannon P. Exploiting Local Similarity for Efficient Indexing of Paths in Graph structured Data[C]//Proc. of the 18th Int'l Conf. on Database Theory. Washington D. C., USA: [s. n.], 2002.

[7] Zou Qinghua, Liu Shaorong, Chu Wesley W. Ctree: A Compact Tree for Indexing XML Data[C]//Proc. of the 6th Int'l Conf. on Web Information and Data Management. Washington D. C., USA: [s. n.], 2004.

[8] Chung C. APEX: An Adaptive Path Index for XML Data[C]//Proc. of 2002 ACM SIGMOD Int'l Conf. on Management of Data. Madison, Wisconsin, USA: [s. n.], 2002.

[9] Cooper B, Sample N, Franklin M, et al. A Fast Index for Semistructured Data[C]//Proc. of the 27th Int'l Conf. on Very Large Data Bases. Rome, Italy: [s. n.], 2001.


编辑 陈 文