doi:10.3969/j.issn.1000-436x.2013.12.010

面向 CPS 节点操作系统的混合调度系统研究与设计

杜晓舟[1,2]，曹晨红[1]，乔建忠[1]，林树宽[1]
(1. 东北大学 信息科学与工程学院，辽宁 沈阳 110004；2. 总参通信工程设计研究院，辽宁 沈阳 110005)

摘 要：针对 CPS 物联性、互联性和智联性的新挑战，设计了具有实时性、灵活性及适应性特点的面向 CPS 节点操作系统的混合调度系统。在深入研究事件驱动与多线程两种调度机制本质区别的基础上，设计了协作式多线程和混合栈管理相结合的带有适配器机制的混合编程模型以及支持事件与线程 2 种任务类型的混合调度模型，并采用合理的实时算法加以实现验证。实验测试表明本调度系统在实时性能指标上明显优于 TinyOS，更符合 CPS 的属性要求。

关键词：物理信息融合系统；节点操作系统；调度系统；编程模型
中图分类号：TP316 **文献标识码：**A **文章编号：**1000-436X(2013)12-0084-10

Research and design of hybrid scheduling system of node operating system for cyber physical systems

DU Xiao-zhou[1,2], CAO Chen-hong[1], QIAO Jian-zhong[1], LIN Shu-kuan[1]
(1. Department of Information Science and Engineering, Northeastern University, Shenyang 110004, China;
2. Communication Engineering Design and Research Institute of the General Staff of PLA, Shenyang 110005, China)

Abstract: In accordance with the new characteristics of physical, internet and intelligence of CPS, a hybrid scheduling system was designed for CPS node operating system with real-time performance, flexibility, and adaptability. Based on the in-depth study of the intrinsic differences between event-driven scheduling mechanism and multithreaded one, a hybrid programming model was designed that combines collaborative multithread and hybrid stack management using adapter mechanism. A hybrid scheduling model was also designed that supports both event and thread task types. Reasonable real-time algorithms were implemented for verification. Experimental results show that the real-time performance of the scheduling system is obviously better than that of TinyOS, thus it is better suited for CPS than TinyOS does.

Key words: cyber physical systems; node operating system; scheduling system; programming model

1 引言

近年来，随着计算(computer)、控制(control)、通信(communication)等 3C 技术的迅猛发展，物理世界与信息世界全面融合的需求显得愈发迫切，物理信息系统(CPS，cyber physical systems)逐渐成为学术界和业界的研究热点。CPS 这一概念为美国国家自然科学基金会(NSF)的 Helen Gill 于 2006 年首次提出，意指信息空间(cyberspace)与物理过程(physical processes)的深度融合[1]。次年，美国总统科技顾问委员会报告《挑战下的领先：竞争世界中的信息技术研发》便将 CPS 列为 8 大优先发展领域的首位[2]。国外纷纷展开对 CPS 的深入研究的同时，国内也积极推动着 CPS 相关理论研究和应用技术的发展[3,4]。

传感反馈网是 CPS 的基础底层系统，需要实时可靠地完成对物理世界感知、互联和反馈等重要功能，节点操作系统是其核心部分，是开展 CPS 相关研究的基础。调度系统则是节点操作系统中的关键模块，如何设计出满足 CPS 属性要求的调度系统对

收稿日期：2013-09-15；修回日期：2013-11-15
基金项目：国家自然科学基金资助项目(61272177)
Foundation Item: The National Natural Science Foundation of China(61272177)

现有的理论和技术提出了新挑战。

2　CPS 与节点操作系统

物理过程是连续的、并发的、时间敏感的；而计算过程是离散的、非并发的，甚至人为忽略时间属性的[5]。由于 2 类过程属性的差异性，加上长期以来研究方法的局限性，2 种过程之间的融合难度很大，在 CPS 中表现尤为明显。

CPS 由信息理论和信息技术 2 大知识领域支撑。前者从软件方向通过开发计算模型，研究算法、程序及其特点来提升系统性能；后者从硬件方向推动信息系统计算平台向更快更可靠的方向发展。但是在系统研究时，算法和程序常常做了理想化假设，独立于依赖执行的物理资源，其具有确定性、终结性和平台无关性特点；与之相对，物理系统本身的特点却是不确定性、持续性和平台相关性。图灵奖得主 Sifakis 明确指出，这种物理平台与计算理论的不匹配使信息物理融合系统缺乏统一的理论框架，目前还没有任何理论可以预测已知软件在给定硬件平台上的准确行为。尤其是在复杂分布式系统环境中，这种不可预测性使系统行为变得更加不可捉摸和难以预知[6]。

CPS 由移动自组织网络(MANET, mobile ad hoc network)和无线传感网(WSN, wireless sensor network)/无线传感反馈网(WSAN, wireless sensor actuator network)等技术发展而来[7]。节点操作系统作为与物理过程直接交互的 CPS 底层基础信息系统，其功能特点的设计情况与 CPS 的系统整体属性息息相关。调度模型、编程模型和调度方法是节点操作系统调度系统的 3 大核心要素，其与 CPS 系统关键属性的对应关系如图 1 所示。

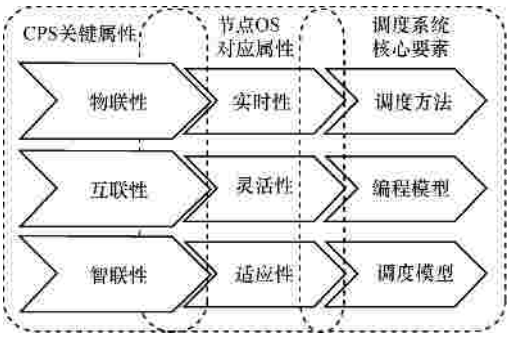


图 1　CPS 与节点系统属性关系

1) 物联性：由于 CPS 中物理信息 2 种过程相交互的特殊要求，节点系统必须满足实时性约束，这一特点需要通过调度系统中设计合理的实时调度算法保证。

2) 互联性：CPS 的系统组件部署范围广，采用分布式互联的方式，应用需求多样化且复杂。节点系统要具有灵活性，能够处理从简单事件单纯触发到复杂信息融合反馈的各种任务类型，这一特点通过具有适配器机制的编程模型实现。

3) 智联性：CPS 部署环境多样且动态可变，节点系统需要采用适应性技术应对需求动态变化的情况，这一特点通过层次架构的调度模型实现。

在无线传感网领域，国内外已经开展了大量关于节点操作系统的研究工作并且取得了显著成果。国外知名的系统如：TinyOS[8]、Contiki[9]、SOS[10]、Mantis OS[11]、Nano-RK[12]、Pixie OS[13]、LiteOS[14]、RETOS[15]等，国内也有依托“973”项目开发的 SenSpire OS[16]等。这些节点操作系统都是针对无线传感网的需求和技术特点设计的，对于 CPS 的属性特点没有考虑以致不完全匹配。但这些研究成果可以为 CPS 节点操作系统的设计提供良好技术基础和研究参照[17]。

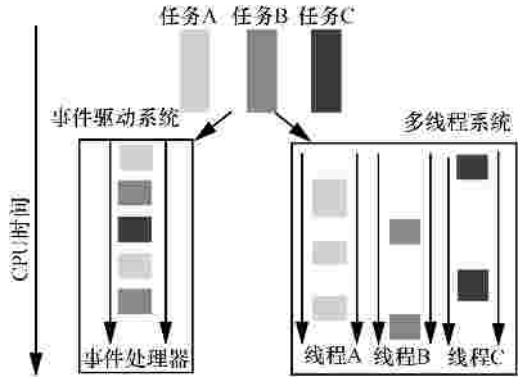
3　节点编程模型相关研究

节点操作系统调度系统的编程模型在 CPS 中属于节点层次(node level)编程[18]，是调度系统中任务编程的基础，其特点决定了节点系统的计算方式，是影响 CPS 关键属性的核心要素。目前，编程模型主要有事件驱动(event-driven)和多线程(multithreaded)2 种类型，如图 2(a)所示。

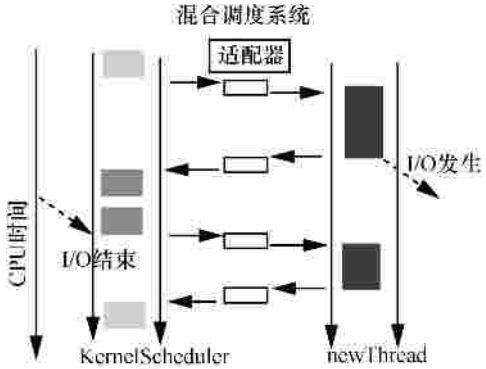
这 2 类编程模型各有特点，学术界对于两者孰优孰劣的争论也由来已久[19]。事件驱动模型具有开销低、功耗小、移植性好等优点，但存在事件循环控制复杂、缓冲区难协调、程序需分割等缺点；多线程模型解决了缓冲区限制、程序控制、调度机制、实时性等问题，但同时却带来了在内存共享机制、上下文切换、栈分析、内存占用等方面的问题[20]。实际上，2 类编程模型各有优劣，应根据应用需求和适用环境合理选用。

近年来，为了既兼顾系统原有编程模型的优势，又可以扩展系统功能并考虑到系统能耗等因素，出现了将 2 种编程模型结合起来的混合趋势。如：在事件驱动模型上建立多线程机制的改进模型，如 TinyThread[21]、TinyOS Fibers[22]、TOSThreads[23]、

Protothreads[24]等；在多线程机制上支持事件处理的模型，如TinyMOS[25]；也有尝试使用混合机制来统一调度2种模型的，如SenSpire[26]。



(a) 事件驱动与多线程原理



(b) 适配器机制混合调度原理

图2 编辑模型原理对比

然而，这些混合模型并没有真正实现事件驱动与多线程机制的彻底融合。如，TOSThreads将基于事件的TinyOS代码封装到一个高优先级的内核线程中，并在应用线程中运行所有的应用代码，虽然在机制上实现了事件驱动与多线程的共存，但是所有的应用只能编写成线程的形式。SenspireOS中虽然支持事件任务与多线程2种代码，但应用只能编写为事件代码形式，通过子调度器实现为线程，本质仍然是基于事件的。

这些调度系统模型融合不充分的根本原因在于，开发者没有抓住事件驱动与多线程2类模型的本质区别来设计混合编程模型。实际上，2种编程模型差异性的根本原因在于其内在机制中任务管理和栈管理2种管理方式的不同[27,28]。任务管理有3种方式：抢占式(preemptive)、顺序式(sequential)和协作式(cooperative)。栈管理也有2种模式：人工栈管理和自动栈管理。多线程编程模型使用了抢占式任务管理与自动栈管理方式，而事件驱动编程模型则合并了协作式任务管理与人工栈管理方式，如图3所示。可以发现，协作式任务管理和自动栈管理相结合的方式才是编程模型的最佳结合方式。本文依据此认识设计了面向CPS节点系统的混合编程模型。

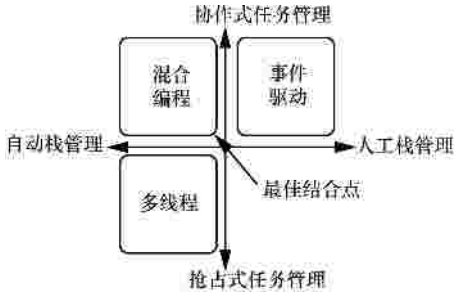


图3 编程模型机制关系

4 混合编程模型设计与实现

本文设计的节点操作系统混合编程模型使用协作式任务管理和混合栈管理相结合的方式。

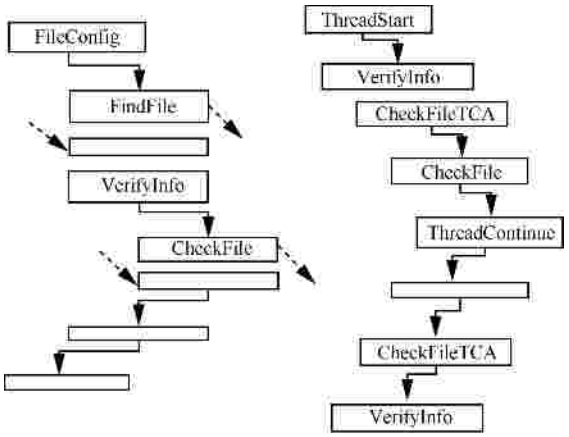
第3节已经阐明，自动栈管理是混合编程模型适宜采用的方式。但是考虑到在传统节点操作系统应用领域，事件驱动编程模型长期占据主流。为了降低开发模式转换带来的成本，本文采用混合栈管理方式实现自动栈管理和人工栈管理的代码共存，确保系统对2种任务代码方式都可以兼容。系统在支持原有事件驱动机制中人工栈管理代码的同时，将更适用的自动栈管理方式提供给开发人员。这样，既借助编程模型的最佳结合实现系统的便利性，又通过兼顾2类编程模型保证了系统的灵活性。

混合栈管理是通过适配器(adaptor)机制实现的，如图2(b)所示。适配器使用续体(continuation)方法[29]确保2种代码的顺畅切换，实现多线程与事件驱动2种机制的完全融合，从而保证整个调度系统的适应性。

调度系统在封装事件调度器的单个线程中调度多个事件任务，并通过内核调度器总体调度线程调度器和事件调度器。在任意时刻调度系统中仅有一个线程是活动的。内核调度器运行在特殊线程KernalScheduler上，同时调度人工栈管理代码和自动栈管理代码。由于自动栈管理代码经常在I/O上阻塞，将其运行在KernalScheduler之外的线程上。当其阻塞时，将控制权直接转交给KernalScheduler，

由内核调度器从任务队列中选择下一个要调度的任务。

本文通过节点系统中文件修改操作的过程来说明系统运作机制。该操作涉及到以下 4 个函数：*FileConfig*、*FindFile*、*VerifyInfo*、*CheckFile*。*FileConfig* 通过 *FindFile* 获取文件元数据指针，先从内存中的文件描述符表中查找，若没有则进行 I/O 操作。然后调用 *VerifyInfo* 确认文件是否修改成功。在 *VerifyInfo* 中调用 *CheckFile* 确认要修改的信息是否在内存中，若没有则通过 I/O 操作读取闪存中的文件元数据并进行修改。其中 *VerifyInfo* 为自动栈管理代码，其余为人工栈管理代码。这种带有 I/O 操作的函数执行顺序如图 4(a)所示。



(a) CheckFile 带 I/O 操作顺序执行　(b) CheckFile 无 I/O 操作顺序执行

图 4　CheckFile 操作顺序执行

与之相对应，存在无 I/O 操作的另一种情况，*CheckFile* 可以立刻返回结果而不需要等待 I/O。采用短路（short circuit）机制对其优化，即不将控制权直接传递给主调度器，而让调用者自行决定一个例程是否退出控制。图 4(b)给出了短路机制执行序列：短路代码检测到 *CheckFile* 在本地运行且没有 I/O 操作，则立刻调用 *ThreadContinue* 执行当前续体。*ThreadContinue* 检测到它没有被调度器直接执行，便设置 *ShortCircuting* 标识比特为 1 来防止适配器切换回主线程。

4.1　人工栈调用自动栈方法

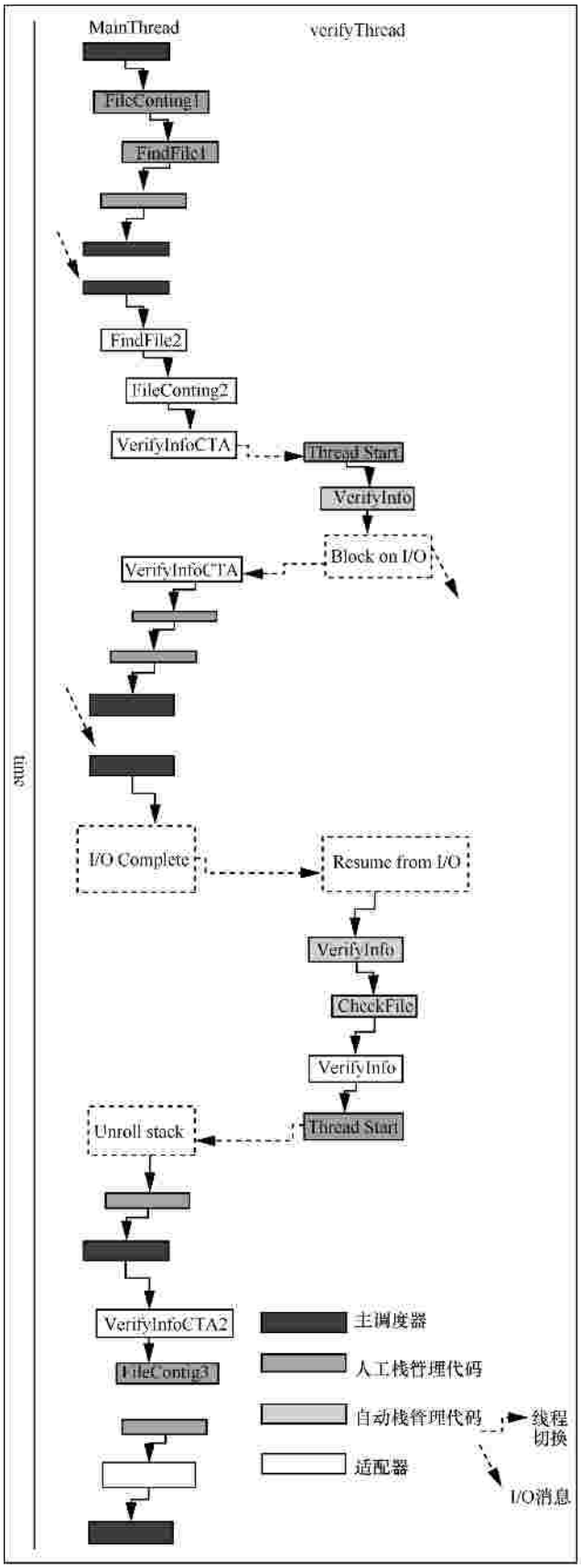
当使用人工栈管理方式编写的函数调用自动栈管理方式编写的代码时，必须采取措施协调这 2 种不同的方式。发起调用的人工栈代码(调用者)不希望在 I/O 上发生阻塞，而被调用的自动栈代码(被调用者)希望阻塞只发生在 I/O 上。为了使二者协调一致，本模型创建一个新的线程来执行被调用的代码。当线程上的代码开始触发执行时，调用者的人工栈便可以进行状态恢复而不至于被自动栈发生的 I/O 阻塞所影响。而在线程上发生的阻塞和恢复次数则没有限制。一旦线程为调用者完成了工作，它便执行调用者的续体来恢复调用者的任务。这样设计的适配器机制可以保证，在整个执行过程中，调用者代码不会阻塞，而被调用者可以按需阻塞。

如图 5(a)所示，人工栈管理函数 *FileConfig*2 通过适配器调用自动栈管理函数 *VerifyInfo*。*FileConfig*2 传递一个指向 *FileConfig*3 的续体，这样可以保证其重新获得控制权并执行任务的最后部分。代码 1 为续体至线程适配器(CTA，continuation-to-thread adaptor)的伪代码，分为调用和返回 2 部分。

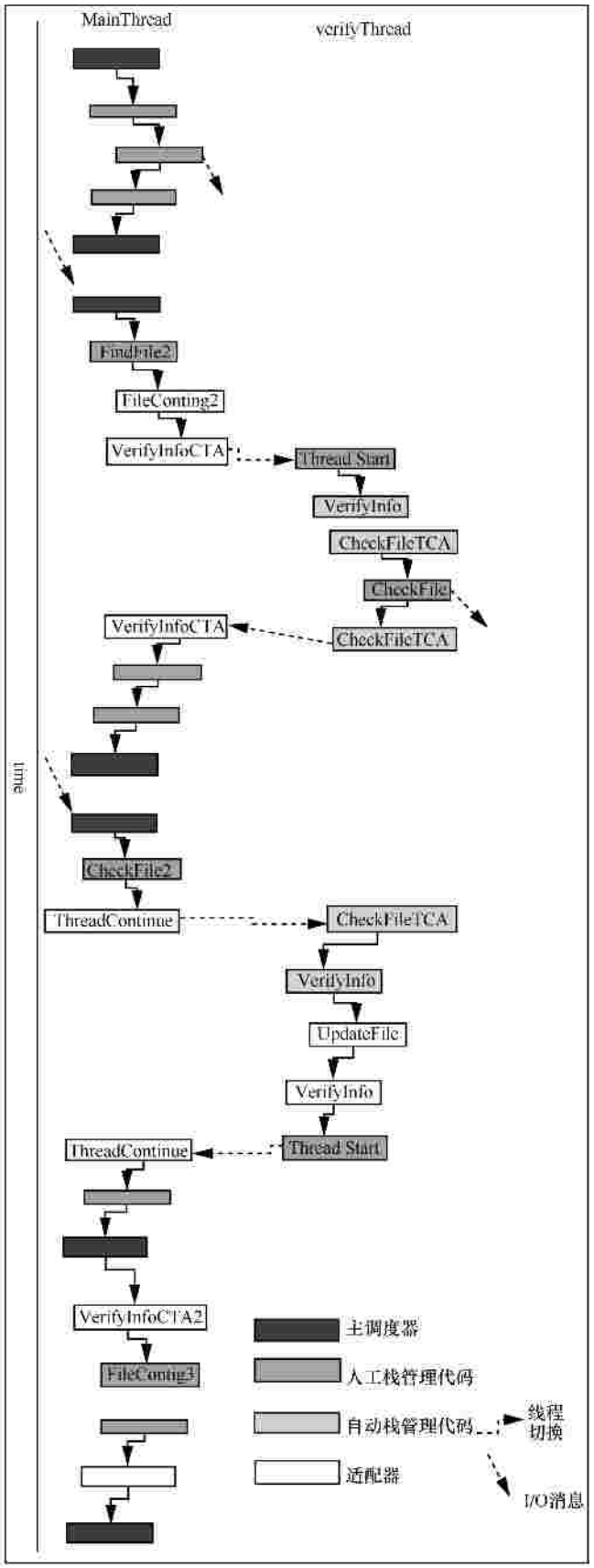
代码 1　续体至线程适配器(CTA，continuation-to-thread adaptor)**伪代码**

```
VerifyInfoCTA:
executing on MainThread
create Continuation vcaCont and initialization
create subthread verifyThread and initialization
switch to verifyThread and start
call automatic stack management VerifyInfo
when I/O request happens,
block on I/O
wait until the request is finished
switch to MainThread
VerifyInfoCTA2:
executing on MainThread
if I/O has completed then
   restore state variables from stack
   switch to original caller FileConfig
endif
```

第一个适配器函数接受其函数参数和一个调用者任务的续体参数。函数创建了自己的续体 *vcaCont* 以及作为一个新的线程的对象 *verifyThread*。该线程追踪函数参数和 *vcaCont* 以便在 *verifyThread* 工作完成时传递控制权给 *VerifyInfoCTA*2。当 *verifyThread* 开始执行时，它执行例程 *Thread_Start* 来取出参数传递给有可能阻塞在 I/O 上的 *VerifyInfo*，*Thread_Start* 伪代码如代码 2 所示。



(a) **人工栈管理代码** FindFile **调用自动栈管理代码** VerifyInfo **示意**



(b) **自动栈管理代码** VerifyInfo **调用人工栈管理代码** CheckFile **示意**

图 5 **混合栈管理代码调用示意**

代码 2　线程函数 Thread_Start **伪代码**

```
executing on verifyThread
if block on I/O then
   switch to MainThread
endif
if I/O has completed then
   execute VerifyInfo
endif
switch to MainThread
```

Thread_Start **直接调用函数** *VerifyInfo*。**如果** *VerifyInfo* **因为发生** I/O **阻塞而退出，在** I/O **函数中使用** *SwitchTo* **调用切换控制权给主线程** *Main Thread*。*VerifyInfoCTA* **恢复控制权并展开续体栈(即** *FindFile*2 **和** *FileConfig*2)。**这样，模型中的混合任务只阻塞在自动栈管理代码引起的** I/O **上，确保了事件处理器** *FileConfig*2 **不会发生阻塞。**

当 I/O **阻塞结束时，** *verifyThread* **恢复执行。在所有任务都执行结束后，控制权返回给** *Thread_Start*。*Thread_Start* **将返回值存入** *VerifyInfoCTA*2 **的续体中，并调度** *VerifyInfoCTA*2 **执行，这时再次切换回** *MainThread*。**当程序执行到** *VerifyInfoCTA*2 **时，返回** *VerifyInfo* **的值给适配器调用者的续体** *FileConfig*3。

4.2　自动栈调用人工栈方法

当一个自动栈管理方式编写的代码需要调用一个人工栈管理方式编写的代码时，前者需要阻塞只发生在 I/O **上，而后者仅简单地调用** I/O **便可直接返回。本模型仍利用适配器机制和续体方法去协调** 2 **种机制。适配器在成功调用人工栈代码后，将控制权交回给主线程，以保证调用者保持阻塞状态。当** I/O **操作完成时，续体继续在主线程上运行并且恢复被阻塞的适配器线程，恢复等待** I/O **结果的原函数执行。**

如图 5(b)**所示，** *VerifyInfo* **在调用** *CheckFile* **时阻塞在** I/O **上，该函数为人工栈管理代码。***VerifyInfo* **调用线程到续体适配器(**TCA，thread-to-continuation adaptor)，**伪代码如下所示。**

代码 3　线程到续体适配器(TCA，thread-to-continuation adaptor)**伪代码**

```
CheckFileTCA：
executing on verifyThread
create Continuation cont and initialization
call CheckFile to check file(s)
if shortCircuit is zero then
    block CheckFile
    switch to MainThread
  endif
return cont.returnValue
ThreadContinue：
   execute on verifyThread
call automatic stack management VerifyInfo
switch to ThreadContinue
if no I/O in manual stack codes then
      shortCircuit ß 1
else
        block on I/O
      switch to MainThread
```

适配器 *CheckFileTCA* **设置了一个专门的续体确保可以借助** *ThreadConinue* **的代码恢复** *verify Thread*，**然后传递该续体给** *CheckFile* **以便其初始化** I/O **操作，并且返回至其事件处理调度程序位置，即将控制权返回给** *CheckFileTCA*。I/O **操作产生后，** *CheckFileTCA* **将控制权直接切换给主线程。在主线程上，续体代码开始执行线程。***VerifyInfoCTA* **返回并展开其堆栈。等到** I/O **结果到达时，调度器继续完成** *VerifyInfo* **剩下的工作** *CheckFile*2。*CheckFile*2 **填充了本地散列表并且通过调用一个续体交回控制权给调度器。此时，** *ThreadContinue* **注意到线程** *verifyThread* **被阻塞，于是调度器切换控制权转回线程** *verifyThread* **执行适配器的下半部分，提取返回值并传递给调用它的自动栈管理代码** *VerifyInfo*。**在线程所有任务都执行结束后，控制权返回给** *Thread_Start*。*Thread_Start* **将控制权交回给** *Thread Continue*，**并进一步切换至** *VerifyInfoCTA*2 **执行，并返回** *VerifyInfo* **的值给适配器调用者的续体** *File Config*3。

5　调度模型和调度方法设计

第 4 **节中已经说明，混合编程模型支持将应用编写为事件驱动或线程任务** 2 **种形式的代码。在此基础上，本文设计实现了面向事件和线程** 2 **类任务的节点操作系统调度模型。该调度模型与面向** CPS **节点操作系统的结构关系如图** 6 **所示。该节点操作系统采用层次化模块结构，对下层利用硬件抽象层实现多平台支持，对上层提供应用接口层支持应用**

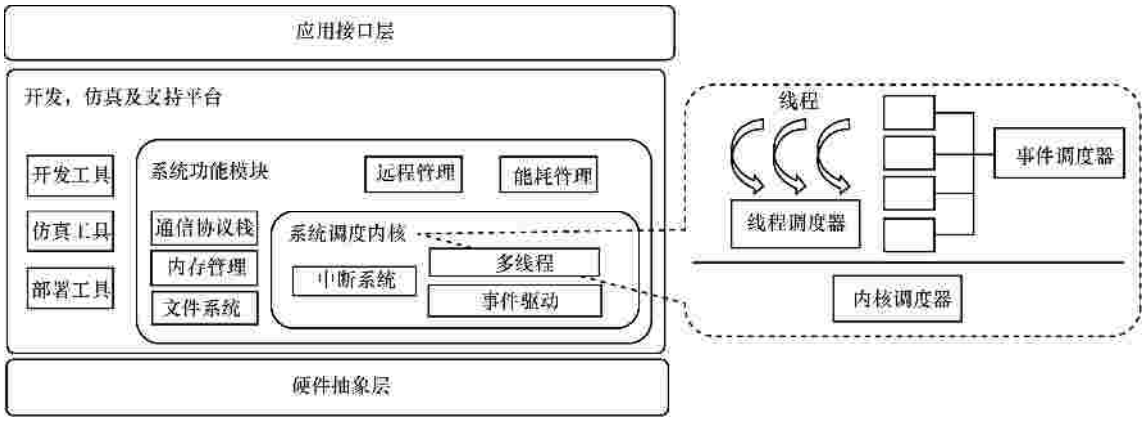


图6 节点操作系统与调度模型架构

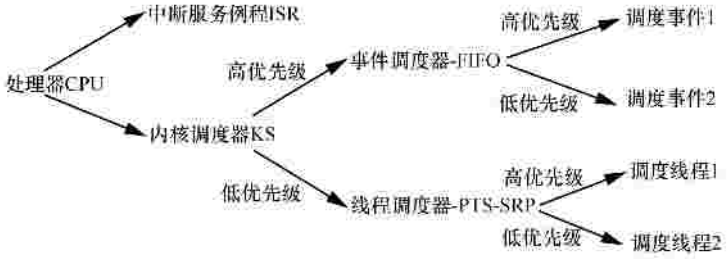


图7 调度机制分层架构

开发。系统本身由调度内核、各类功能模块和开发仿真支持平台组成，相关工作在文献[17]中有详细阐述。其中，调度模型是构成操作系统内核的关键部分，由中断系统及各类调度器构成。在本模型中，事件调度器被封装在一个线程中来调度事件任务，另设计一个线程调度器来调度线程任务，两者再由内核调度器统一调度。

调度模型的分层机制以及相应调度方法的架构关系如图7所示。

该调度模型中，以优先级抢占方式调度各类调度器。最高层由中断服务例程（interrupt service routine）和内核调度器（kernel scheduler）构成，且前者优先级高于后者。内核调度器再对事件调度器（ES，event scheduler）和线程调度器（TS，thread scheduler）进行调度，ES的优先级高于TS。

在事件调度器ES中，事件任务之间是平等的，不能相互抢占，按先入先出(FIFO，first input first output)方法进行调度。

在线程调度器TS中，本模型并没有采用通常的时间片轮转方法。因为，文献[30]已经指出，与普通抢占式及非抢占式算法相比，资源访问控制协议(SRP，stack resource protocol)与抢占阈值(PT，preemption threshold)调度算法能够显著提高实时任务集合的可调度性。因此，系统为每个任务赋予优先级及抢占阈值，只有在系统中出现更高优先级的任务，或当前的线程调用事件任务函数，再或线程主动通过yield或sleep函数主动放弃处理器控制权时，才调用内核调度函数，这样可以大大减少线程切换的次数，降低系统开销。SRP与PT相结合的实时调度算法，可以显著提高任务集合的可调度性，增强系统实时性，同时减少任务切换次数，降低任务集对内存资源的消耗，提高CPU利用率。

这种采用分层调度框架的调度模型具有明显的适应性特点，可以兼容响应CPS中事件与线程2类任务的调度需求，可以根据任务特点自适应地采用对应的调度方法进行任务调度，体现了CPS的智联性要求。

6 调度系统实时性能测试与分析

在以上两节内容中，已经根据设计特点分别阐述了编程模型的灵活性特点和调度模型的适应性特点。而物联性是CPS的系统关键属性，对应的调度系统实时性特点也是人们关心的重要特征，下面通过实验来测试整个调度系统的实时性能。

为了消除底层硬件平台及驱动的差异性对实验结果带来的影响，实验采用 Power-TOSSIM 仿真器进行测试，主要测量系统实时任务中断响应数及任务错失率 2 个指标。利用 Gnuplot 软件对测试数据进行了综合分析，并与 WSN 领域的知名节点操作系统 TinyOS 进行了性能比较。

6.1　任务中断响应数测量

系统中断响应时间 $T_{\text{I-Response}}$ 可由式(1)计算得出

$$T_{\text{I-Response}}=T_{\text{I-Delay}}+T_{\text{CPU-Register}} \tag{1}$$

其中，$T_{\text{I-Delay}}$ 为中断延迟时间，$T_{\text{CPU-Register}}$ 为 CPU 内部寄存器保存时间。

影响中断响应时间的最主要因素为 $T_{\text{I-Delay}}$，计算方法由式(2)给出

$$T_{\text{I-Delay}}=T_{\text{I-turnoff}}+T_{\text{ISR-First}} \tag{2}$$

其中，$T_{\text{I-turnoff}}$ 为中断关闭的最长时间，$T_{\text{ISR-First}}$ 为中断服务例程第一条指令开始执行的时间。

影响中断延迟最重要的因素为 $T_{\text{I-turnoff}}$。因为程序在进入临界区代码区域之前都要关闭中断，执行完临界区代码之后，再打开中断。中断关闭的时间越长，中断延迟就越大。

测试实验设计为：在调度系统中运行 2 个程序 P_1 和 P_2，P_1 每隔一定时间提交一个任务，P_2 中设置一个循环开启的计时器中断，每隔一定时间提交一个计时器中断投递任务，并开启下一个时钟。在 TinyOS 和本文设计的调度系统中，分别测试 1 000 ms 时间间隔内系统响应 P_2 的计时器中断次数。测试结果如图 8 所示。

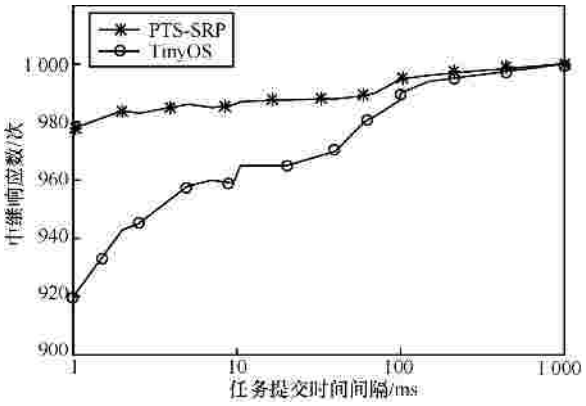


图 8　任务中断响应数测试

从图 8 中可以看到，任务提交时间间隔为 1 ms 时，本调度系统的中断响应数（978 次）明显高于 TinyOS（920 次），性能大约提升了 6.5%；时间间隔为 10 ms 时，两者的差距为 27 次，性能提升率为 2.8%。随着任务提交间隔时间增大，两类调度系统的中断响应数差距逐渐变小，但时间间隔为 100 ms 时，两者仍有 1%的差距。由此可见，与 TinyOS 相比，本调度系统对实时任务的响应率明显更好。

这种差距的原因在于，TinyOS 中长时间的临界区操作降低了中断响应速度。TinyOS 为不可抢占型内核，每个任务要运行至完成时，才释放 CPU 使用权，只有硬件中断才可以打断运行态任务。本文设计的调度系统采用了分层调度体系以及短中断策略，将中断中实时性强的硬件操作与其他部分隔断，保证中断始终能够快速响应，显著提升了调度系统的实时性能。

6.2　任务错失率测量

为了测试 2 种节点操作系统中的实时任务错失率，设计实验仿真模型如图 9 所示。

图 9　简单节点传输模型示意

在此模型中设置了 3 个传感器节点，其中节点 P_a 通过节点 P_b 向节点 P_c 发送数据，因此在节点 P_b 上执行 2 类任务：一类为本地普通周期性任务，一类为具有截止期的实时性转发任务。设置本地任务为 T_{l1} 至 T_{l5}，设置转发任务为 T_{t1} 至 T_{t3} 带有截止期，具体属性如表 1 所示。

表 1　测试任务

本地任务			转发任务		
任务	执行时间/ms	周期/ms	任务	执行时间/ms	截止期/ms
T_{l1}	40	200	T_{t1}	60	160
T_{l2}	80	300	T_{t2}	80	200
T_{l3}	120	400	T_{t3}	100	240
T_{l4}	160	500	—	—	—
T_{l5}	200	500	—	—	—

实验初始时在节点 P_b 上仅运行一个本地任务 T_{l1} 以及所有的转发任务，之后每隔 1s 节点 P_b 上多运行一个任务 $T_{ln}(n{:}2\sim5)$。实验观察在不同的调度系统下，P_b 节点处理实时转发任务的能力，通过其对转发任务的完成时间来测量对比，实验结果如图 10 所示。

从图 10 中可以看到，在加入 4 个任务后，2 类调度系统实时任务错失率的差距明显显现。本调度系统刚刚有大约 1%的错失率，TinyOS 中的错失率已经上升到 10%。随着任务数的进一步增加，

TinyOS 的任务错失率大幅上升，加入 6 个任务后，TinyOS 为 40%，为本调度系统 12%的 3 倍。在加入 9 个任务后，TinyOS 错失率已经升高到本调度系统的 4 倍。

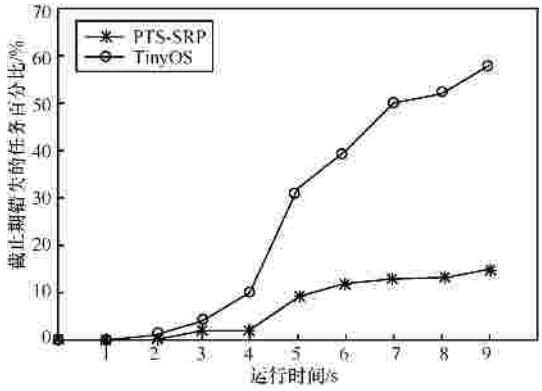


图 10 2 种调度下的任务截止期错失率比较

这是由于在 TinyOS 中默认使用了 FIFO 调度，并不区分实时任务与非实时任务，只按任务到达先后顺序处理，无法保证实时任务的响应速度。因此随着任务数目的增多，愈发难以处理多个实时任务。而在本调度系统中，任务按照分配的优先级进行分层次抢占式调度，可以充分保证任务的实时性和系统的可调度性。

7 结束语

本文主要关注了 CPS 节点操作系统中 3 个主要特性：实时性、灵活性和适应性，依据这些特性相应设计了调度系统的 3 个模块：采用适配器机制实现的协作式多线程和混合栈管理相结合的混合编程模型、支持事件和线程 2 种任务合理调度的分层调度模型、策略合理的实时调度方法。从任务基础代码到层次调度体系完整地实现了事件与线程 2 种任务类型的兼容处理。使用节点仿真器的实验测试表明，无论是单节点系统的实时任务中断响应数指标，还是多节点网络环境的实时任务错失率指标，本调度系统都具有优于 TinyOS 的表现，更符合 CPS 物联性、互联性、智联性的要求，为面向 CPS 的节点操作系统的深入研究奠定了良好基础。

参考文献：

[1] EDWARD A L, SANJIT A S. Introduction to embedded systems, a cyber-physical systems approach[EB/OL]. http://LeeSeshia.org.

[2] MARBURGER J H, KVAMME E F, SCALISE G, *et al*. Leadership under Challenge: Information Technology R&D in a Competitive World. An Assessment of the Federal Networking and Information technology R&D Program[R]. 2007.

[3] **李建中, 高宏, 于博. 信息物理融合系统(CPS)的概念、特点、挑战和研究进展**[R]. 2010.
LI J Z, GAO H, YU B. The Concept, Characteristics, Challenges and Research Progress of CPS[R]. 2010.

[4] **李仁发, 谢勇, 李蕊等. 信息—物理融合系统若干关键问题综述**[J]. **计算机研究与发展**, 2012, 49(6):1149-1161.
LI R F, XIE Y, LI R, *et al*. Survey of cyber-physical systems[J]. Journal of Computer Research and Development, 2012, 49(6):1149-1161.

[5] LEE E A. CPS foundations[A].Proceedings of the 47th Design Automation Conference[C]. Anaheim, California, USA, 2010. 737-742.

[6] MALKHI D, CORBATO F J, EMERSON E A, *et al*. Systems architecture, design, engineering, and verification-the practice in research and research in practice[A]. ACM Turing Centenary Celebration[C]. New York, NY, USA, 2012.

[7] WU F J, KAO Y F, TSENG Y C. From wireless sensor networks towards cyber physical systems[J]. Pervasive and Mobile Computing, 2011, 7(4): 397-413.

[8] LEVIS P, MADDEN S, POLASTRE J, *et al*. Ambient Intelligence[M]. Berlin Heidelberg: Springer, 2005.

[9] DUNKELS A, GRONVALL B, VOIGT T. Contiki-a lightweight and flexible operating system for tiny networked sensors[A]. Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks[C]. Tampa, FL, USA, 2004. 455-462.

[10] HAN C C, KUMAR R, SHEA R, *et al*. A dynamic operating system for sensor nodes[A]. Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services[C]. Seattle, WA, USA, 2005.163-176.

[11] BHATTI S, CARLSON J, DAI H, *et al*. MANTIS OS: an embedded multithreaded operating system for wireless micro sensor platforms[J]. Mobile Networks and Applications, 2005, 10(4): 563-579.

[12] ESWARAN A, ROWE A, RAJKUMAR R. Nano-rk: an energy-aware resource-centric rtos for sensor networks[A]. Proceedings of the 26th IEEE Real-Time Systems Symposium(RTSS 2005)[C]. Miami, FL, USA, 2005.256-265.

[13] LORINCZ K, RONG C B, WATERMAN J, *et al*. Pixie: an operating system for resource-aware programming of embedded sensors[A]. Fifth Annual IEEE International Conference on Pervasive Comput ng and Communications Workshops[C]. Charlottesville, Virg nia, USA, 2008.267-271.

[14] CAO Q, ABDELZAHER T, STANKOVIC J, *et al*. The liteos operating system: towards unix-like abstractions for wireless sensor networks[A]. Proceedings of the 7th International Conference on Information Processing in Sensor Networks[C]. St Louis, Missouri, USA, 2008. 233-244.

[15] CHA H, CHOI S, JUNG I, *et al*. RETOS: resilient, expandable, and threaded operating system for wireless sensor networks[A]. Proceedings of the 6th International Conference on Information Processing in Sensor Networks[C]. Cambridge, MA, USA, 2007.148-157.

[16] DONG W, CHEN C, LIU X, *et al*. SenSpire OS: a predictable, flexible, and efficient operating system for wireless sensor networks[J]. IEEE Transactions on Computers, 2011, 60(12): 1788-1801.

[17] DU X Z, QIAO J Z, LIN S K, *et al*. The design of node operating system for cyber physical systems[J]. Procedia Engineering, 2012, 29:

3717-3721.

[18] SUGIHARA R, GUPTA R K. Programming models for sensor networks: A survey[J]. ACM Transactions on Sensor Networks(TOSN), 2008, 4(2): 8.

[19] BERNAUER A, RÖMER K, SANTINI S, *et al*. Threads2events: an automatic code generation approach[A]. Proceedings of 6th Workshop on Hot Topics in Embedded Networked Sensors[C]. Killarney, Ireland, 2010.8-12.

[20] MOUBARAK M, WATFA M K. Guide to Wireless Sensor Networks[M]. London: Springer, 2009.

[21] MCCARTNEY W P, SRIDHAR N. Abstractions for safe concurrent programming in networked embedded systems[A]. Proceedings of the 4th International Conference on Embedded Networked Sensor Systemsp[C]. Boulder, CO, USA, 2006.167-180.

[22] WELSH M, MAINLAND G. Programming sensor networks using abstract regions[A]. NSDI[C]. San Francisco, California, USA, 2004. 3.

[23] KLUES K, LIANG C J M, PAEK J, *et al*. TOSThreads: thread-safe and non-invasive preemption in TinyOS[A]. Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems[C]. Berkeley, California, USA, 2009. 127-140.

[24] DUNKELS A, SCHMIDT O, VOIGT T, *et al*. Protothreads: simplifying event-driven programming of memory-constrained embedded systems[A]. Proceedings of the 4th International Conference on Embedded Networked Sensor Systems[C]. Boulder, Colorado, USA, 2006. 29-42.

[25] TRUMPLER E, HAN R. A systematic framework for evolving TinyOS[A]. Proceedings of the Third IEEE Workshop on Embedded Networked Sensors[C]. Cambridge, MA , USA, 2006.61-65.

[26] DONG W, CHEN C, LIU X, *et al*. FIT: a flexible, lightweight, and real-time scheduling system for wireless sensor platforms[J]. IEEE Transactions on Parallel and Distributed Systems, 2010, 21(1): 126-138.

[27] KOPJAK J, KOVÁCS J. Event-driven control program models running on embedded systems[A]. Proceedings of the 6th IEEE International Symposium on Applied Computational Intelligence and Informatics(SACI)[C]. Timisoara, Romania, 2011.323-326.

[28] CHU R, GU L, LIU Y, *et al*. Versatile stack management for multitasking sensor networks[A]. Proceedings of the 30th International Conference on Distributed Computing Systems(ICDCS)[C]. Genoa, Italy, 2010. 388-397.

[29] ADYA A, HOWELL J, THEIMER M, *et al*. Cooperative task management without manual stack management[A]. Proceedings of USENIX 2002 Annual Technical Conference[C]. Monterey, CA, USA, 2002. 289-302.

[30] KESKIN U, BRIL R J, LUKKIEN J J. Exact response-time analysis for fixed-priority preemption-threshold scheduling[A]. Proceedings of the 15th Conference on Emerging Technologies and Factory Automation (ETFA)[C]. Bilbao, Spain, 2010.1-4.

作者简介：

杜晓舟（1982-），男，辽宁沈阳人，东北大学博士生，主要研究方向为 CPS、网络嵌入式系统。

曹晨红（1991-），女，安徽合肥人，东北大学硕士生，主要研究方向为 CPS、网络嵌入式系统。

乔建忠（1964-），男，辽宁沈阳人，博士，东北大学教授，主要研究方向为分布式计算、人工智能。

林树宽（1966-），女，辽宁沈阳人，博士，东北大学教授，主要研究方向为机器学习、人工智能。