任务调度在 CANopen 主站设计中的应用

宋威，方穗明，张明杰，徐喆
SONG Wei, FANG Sui-ming, ZHANG Ming-jie, XU Zhe

北京工业大学，电子信息与控制工程学院，北京 100022
College of Electrical Information and Control Engineering, Beijing University of Technology, Beijing 100022 China
E-mail: benjaminweber@emails.bjut.edu.cn, Phn: +86-10-67396159-601

Task Scheduler in the Design of CANopen Master

Abstract: As the configuration manager, network management master and monitor, CANopen master plays an important role in CANopen network. According to its features, CANopen master should operate real-timely, concurrently and dynamically. Therefore, multi-task is the best work mode. In consideration of transplanting capability, an ANSI C based non-preemptive multi-task scheduling structure is proposed. Because of the pure ANSI C implementation, it can run on any C compiler. Verified by field experiments, it realizes microsecond level scheduling on both ARM7 and windows platforms, so that it satisfies the real-time requirements of CANopen.

Key words: CANopen; task scheduling; concurrent

摘要: CANopen 主站作为 CANopen 网络的配置管理者、网络管理者和监控者具有重要的意义。CANopen 网络的特性决定了 CANopen 主站必须具有高度的实时性、并发处理能力和动态灵活性，因此多任务的工作方式是最佳的选择。基于可移植性的考虑，提出了一种用标准 C 语言实现的非抢占式的多任务调度结构。由于仅用标准 C 语言实现，该结构可运用于任何 C 编译环境。经过实际测试，在 ARM7 和 Windows 操作系统上该结构都可达到毫秒级的调度速度，满足 CANopen 主站的实时性要求。

关键词: CANopen; 任务调度; 并发

文献标示码: A **中图分类号**: TP273

CANopen 是建立在 CAN（Controller Area Network）串行总线之上的应用层和传输层协议。它支持多种传输模式，具有良好的开放特性，并能极大程度地降低 CAN 网络数据的负荷，现已运用在工业控制的各个领域[1]。本文即是针对于 CANopen 网络在混合动力汽车控制系统上的应用，在汽车的嵌入式主处理器或者监控处理器上实现 CANopen 主站，以监控和管理整个汽车控制网络。

在标准[2]的基础上，标准[3]进一步规定了网络管理主站（NMT Master）、配置管理者（Configuration Manager）和服务数据对象管理者（SDO Manager）的功能和行为，以及详细的从节点启动过程，完善了应用层和传输层规范。实现这些管理和监视功能的 CANopen 节点也就是本文所指的 CANopen 主站。

1 CANopen 主站的特性

CANopen 主站是一个标准的嵌入式系统，应具有以下三个显著特性：

- 实时性 实时性是所有 CANopen 节点的基本特性。在 CANopen 网络当中，同步周期和同步窗口以微秒为时间单位，紧急事件的禁止时间以 100μs 为时间单位，心跳报文和节点保护周期以毫秒为时间单位[2]。当然，这并不意味着同步周期可以小到 1μs。一个 8 字节的 CAN 标准帧在 1Mbps 的波特率下的传输时间约为 110μs，所以合理的同步周期应认为以 100μs 为基本时间单位。尽管如此，一般操作系统以毫秒为最小时间单位的系统时钟，10ms 为最小任务时间片[4]的性能已经很难满足 CANopen 网络较为苛刻的实时性要求。
- 并发事件处理 如果说实时性是 CANopen 节点的普遍属

北京市教委重点项目和北京市自然科学基金共同资助项目 KZ20041000501
作者简介：宋威，（1983—），男，硕士研究生，主要研究领域为集成电路设计和工业现场总线；方穗明，（1952—），女，副教授，主要研究领域为数字信号处理和嵌入式系统；张明杰，（1982—），女，硕士研究生，主要研究领域为现场总线及汽车故障诊断；徐喆，（1968—），女，副教授，主要研究方向为自适应控制，传感器网络及现场总线。

性，那么并发事件处理是 CANopen 主站的独有特性。根据标准[3], CANopen 主站在 CANopen 网络的启动阶段应并行独立地启动网络中的所有节点。此外，在正常运行状态，针对于每一个节点的过程数据对象（PDO）通讯、心跳报文接收以及同步信号的生成、时间戳发送等等，也是互相独立的任务。相比从节点来说，主站需要处理大量的并发事件，因此 CANopen 主站需要具有强大的并发处理能力。

- 动态灵活性 动态灵活性是 CANopen 主站的又一独有特性。一般情况下，从节点需要支持的 PDO 最多为 4 对，SDO 最多为 1 对，因而按照最大要求设计即可满足需求。然而，对于主站来说，网络未确定之前，主站无法确定自身需要支持的最大 PDO 数量和 SDO 数量。如果按照对象字典支持的最大 512 个 PDO 和 128 个 SDO 设计，一般情况下将会造成空间浪费。针对 CANopen 主站这种不确定的需求，最好的办法即支持动态任务和数据加载。

2 基于任务调度的实现思路

为了实现上述的特性，任务调度是最直接的选择。任务调度的方法以优先级为顺序分配运算时间。对不同的事件定义不同的优先级，从而使得需要高实时性的任务通过高优先级的方式获得优先响应。这是一般嵌入式系统的普遍做法，也是 CAN 网络的本质特性。从 CANopen 的通讯对象标识（COB-ID）分配可推得，网络管理（NMT）命令处理的优先级高于 PDO 处理，更高于 SDO 处理和节点保护。并且，任务调度本身已经提供了高度的并发处理能力和动态灵活性。

从实现的角度，传统的多任务设计有以下两种方式：

在无操作系统的环境中，用函数模拟不同进程，循环调用各函数实现并行多任务是最常用的设计思路。该方法不需要操作系统的支持，占用空间小；直接控制硬件，响应速度更快。然而，这种方法并不能满足 CANopen 主站的要求。利用循环实现并发处理，并不是真正的任务调度，因此事件响应的优先级是通过函数间接实现。一般函数在运行时已经确定，不具备动态任务加载的能力。状态机需要主循环间接实现，在复杂应用下，该方法因涉及太多细节而难以调试。

另一种方法是利用操作系统。操作系统提供多进程的机制，因此可以直接实现任务调度。但是，该方法面临可移植性差、难以嵌入现场控制器、实时性差的问题。尽管操作系统都能提供多进程，然而方式并未统一，导致一个系统上的多进程程序无法在其他操作系统上运行。其次，诸如 Windows 和 Linux 操作系统很难嵌入到所有的微控制器中，比如已经大量使用的 ARM7，即使能够嵌入，很多操作系统的实时性能仍然值得怀疑。当然，WinCE 和μCOS 在实时性能方面已经达到 CANopen 的要求，然而基于它们的应用仍然会遇到移植性问题。在本文的应用当中，CANopen 网络的设计阶段并不能确定最终的 CANopen 主站宿主处理器和系统，因而必须考虑移植性因素。

基于以上的原因，我们采用了另外一种办法。对于 CANopen 主站来说，操作系统最大的作用在于提供了多任务的实现与调度算法。因此，如果能用标准 C 语言，单独实现多任务的调度算法，脱离操作系统运行，则该算法就能运行在任何支持 C 编译环境的平台之上，兼顾了可移植性和任务调度的要求。同时，由于不使用操作系统，实时性也得到提高。唯一的问题是，标准 C 语言难以实现抢占式的内核调度。不过 CANopen 是一种通讯协议，不存在大量的算法操作，因此单任务的连续执行时间较短，非抢占式的任务调度也能满足实时性要求，同时大量减少了实现任务调度的复杂度。

3 任务结构体的构成

任务调度的前提是能够用数据结构完全代表一个任务，正如进程描述符在 Linux 系统中的作用。不过，针对于 CANopen 主站的任务调度，只需要实现和 CANopen 网络相关的内容。

3.1 优先级

任务的优先级代表任务获得执行的权力，优先级高则更容易被执行。根据 CANopen 网络的实时要求，基本优先级定义如下：

表 1 优先级定义

0	CAN 驱动和同步生成
1	PDO 报文解析
2	PDO 处理
3	SDO 报文解析
4	SDO 处理
5	NMT 报文解析
6	节点保护和心跳报文处理
7	主状态机和用户接口
8	内存清理和对象字典清理

定义最高优先级为 0，根据不同事件在 CANopen 中的实时性要求，一共分为 9 种优先级。其中，CAN 驱动直接处理紧急报文，报文解析任务通过报文的不同类别和节点号，建立或触发相应处理任务。

3.2 触发条件

时间和事件是两个重要的触发条件。

CANopen 协议当中的时间规定明显多于其他协议，因此如何设计定时器也是实现 CANopen 主站的难点之一。基于任务调度的方式，时间被作为任务触发的条件之一直接定义到任务结构体中。这个时间由两个 32 比特的整数表示，精确到 1μs，64 位的位宽足以保证该时间不会溢出。当任务处于阻塞状态时，该时间代表任务下一次运行的开始时间。这样，整个 CANopen 主站只需要实现一个微秒级的系统时钟。当被阻塞的任务时间早于系统当前时间时，该任务被触发。在这种机制下，周期执行只需要任务在每一次阻塞之前，设置下一次运行时间即可，而不用申请定时器。

事件是另外一种触发方式。传统的操作系统需要专门的数据结构表示事件，比如消息、信号量等等。这些数据结构复杂并且消耗大量资源和运算时间。对于 CANopen 这样设计意图明

确的系统，不需要如此复杂。所以任务的事件被直接定义为一个直接指向系统全局标志或者其他任务的数据空间的整型指针。指向的数据初始为 0，当非零时，任务被触发。这样的方式提供了简单的任务同步机制并极大地简化了事件的定义。当然，由于没有事件的结构体定义，事件需要小心处理。

时间和事件是任务的两个触发条件，其中任意一个满足任务就可执行。单独使用时间触发可实现周期调度，单独使用事件触发，可实现任务间的同步和全局事件等待，两个触发条件一起使用，可实现对长时未达事件的超时处理。因此，仅实现时间和事件这两种触发方式，已经基本满足了 CANopen 主站的各种触发情况。

3.3 任务函数和数据空间

和操作系统不同，CANopen 的任务有明确的目的和行为，因此任务的执行可以直接由函数完成。因此，单独定义了一种称为任务函数的特殊函数。该函数以任务结构体指针为输入，任务的执行状态为输出，函数运行的所有参数在任务结构体中定义。进而，任务结构体提供一个大小为 40 字节数据区，为该任务独有。任务函数则可以在该数据区中定义变量、保存状态、设置其他任务的触发事件等等，相当于传统进程的独立数据空间。在任务结构体中，还需保存执行该任务的任务函数指针，这样，任务函数和任务数据空间互相关联。

除了上面的 3 个组成部分，多个任务结构体须组成双向链表。由于双向链表的删除和插入操作时时间恒定，因此由双向链表形成的等待队列和运行队列能够减少操作复杂度，提高任务调度的效率，这也是操作系统的普遍做法。下面给出了该任务结构体的 C 语言定义：

```
typedef struct _TaskObj {
    char runPrio;            // 优先级
    char * pRunEvent;        // 触发事件
    char (*pFun) (struct _TaskObj *);    // 任务函数
    struct _TaskObj * pNext;
    struct _TaskObj * pPre;
    long Argu[10];           // 数据空间
    long timeHigh;           // 触发时间高位
    long timeLow;            // 触发时间低位
} TaskObj, *pTaskObj;
```

4 任务调度机

完整的任务调度，还需要构建等待任务队列和运行任务队列。其中，等待任务队列直接由任务结构体组成的双向链表构成，存储所有未满足触发条件的阻塞任务。相比而言，运行队列较为复杂。参照 UNIX SVR4[4]和 Linux 2.6 内核[5]的运行队列结构，CANopen 主站的运行队列如图 1 所示：

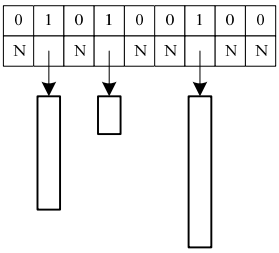

图 1 运行队列

运行队列由 9 条单独的任务队列构成，每一个任务队列代表已经满足触发条件的某一优先级的所有任务。并且，每一条队列有一个标志位标识队列的空满情况。当标志位为 1 时，说明该队列非空。因而，找寻最高优先级的任务就变为寻找一个最高优先级的非空标志位，该标志位对应任务队列的头节点即为下一个应执行任务。根据文献[4]，由于优先级数固定，寻找最高优先级任务的时间和运行队列中的任务数无关，该算法为固定时间算法，复杂度 O(1)。

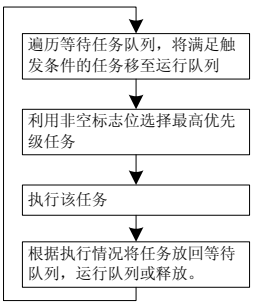

图 2 任务调度机的执行流程

不过，本文的任务调度为 O(n)算法，其中 n 代表等待队列中的任务数。根据上文的描述，任务的触发采用时间和事件两种触发方式，但是并没有操作系统中对应的事件结构，同时任务以非抢占式的方式运行，导致关于触发的判断必须由调度算法完成。因而，在选择下一个执行任务之前，任务调度算法必须先遍历整个等待队列，将所有新满足触发条件的任务移到运行队列。如图 2 所示，从一个任务结束到另一个任务执行的调度时间和任务等待队列的长度成正比，为 O(n)算法。

5 基于任务调度的 CANopen 主站结构

基于上述的任务调度机制，整个 CANopen 主站的结构如图 3 所示：

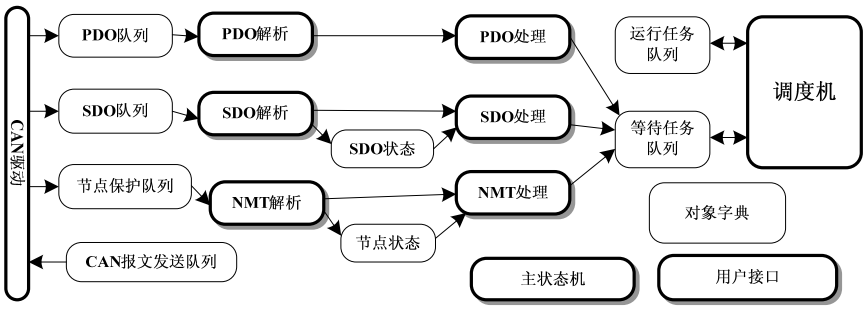

图 3 CANopen 主站的基本结构

CAN 驱动为 CANopen 主站和 CAN 控制器的接口，为最高优先级的实时任务，利用发送中断和接收中断发送和接收报文。接收到的报文，根据 COB-ID 分别放入 PDO、SDO 和节点保护 3 个全局报文队列。对于紧急事件，由 CAN 驱动任务在接收报文时直接处理。各报文队列的非空将会触发等待队列中的 PDO、SDO 和 NMT 报文解析等常置任务。这些任务根据报文的 COB-ID 以及处于 SDO 状态队列或者节点状态队列中的信息，触发等待该报文的任务或者生成新的报文的处理任务并置入等待队列。NMT 处理任务负责维护心跳报文或者节点保护机制。当长时间未收到心跳报文，或者未收到节点状态，该任务会由于超时而触发，并将节点异常上报给用户程序。

现以 SDO 上传为例描述该系统的工作过程。如图 4，首先 SDO 上传的发起任务 A 需要新建一个 SDO 处理任务 B，用需读取的服务器端对象字典索引、子索引和节点号等信息配置 B 任务，置入等待队列，并将下一运行的触发标志设为 B 的返回状态。之后，SDO 处理任务 B 获得执行，生成符合 SDO 规范的报文，送入 CAN 报文发送队列，并生成 SDO 状态节点，置入 SDO 状态队列，设置触发条件为 SDO 节点状态的选中。由于 CAN 报文发送队列非空，相应 CAN 报文发送任务被触发，并将报文发出。如果报文被正确响应，相应包含从节点数据的 SDO 报文将被驱动接收，经过分类置入 SDO 队列。同理，SDO 队列的非空将触发 SDO 报文解析任务执行。报文解析任务会自动比较接收报文和 SDO 状态队列中的信息，从而由任务 B 置入的 SDO 状态节点将被选中。进而，被选中的 SDO 状态节点触发任务 B 再次执行。任务 B 判断 SDO 操作的结果，读取数据，保存入任务发起者 A 的数据空间，并设置返回。最后，B 的返回重新触发 A 的执行，此时 A 已经获得需要读取的 SDO 值。

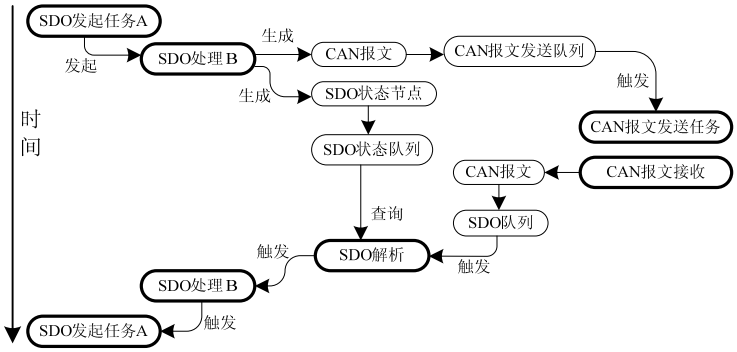

图 4 SDO 读取任务流程

6 测试及分析

根据在 PC 机和 ARM7 平台上的测试，得到如下的测试结果：

表 2 任务调度机的测试结果

测试平台	频率	平均调度时间	平均等待任务
Pentium 4	3.0GHz	2.28μs	35.1
ARM7 LPC2294	44.2368MHz	12.63μs	3.4

其中，任务的调度时间为一个任务的结束到另一个任务开始运行花费在调度算法上的时间。根据文献[6,7]关于μCOS 和 Windows CE 操作系统的实时性分析，可以得到比较结果表 3。

表 3 任务调度算法比较

操作系统	平台	频率	平均调度时间
CANopen	ARM7 LPC2294	44.2368MHz	12.63μs
μCOS-II	ARM7 S344B0	67.5MHz	4.0μs
Windows CE	ARM9 S3C2410X	556MHz	7.5μs

从比较结果可见，用标准 C 编写的任务调度算法的调度时间已接近μCOS-II 系统的水平，并远远超过 Windows CE 平台。当然，由于采用非抢占调度方法，调度时间并不能完全代表事件的响应时间。不过，只要合理分配任务的优先级，并控制任

务单次连续运行的代码量，在 ARM7 平台上的响应时间也可控制在 1ms 以内，针对于一般的 CANopen 应用来说已经足够。测试结果也表明，在 PC 上的执行效率明显高于操作系统提供的调度算法，因而在基于 PC 的 CANopen 主站设计当中，该方法也是行之有效的。

7 结论

针对 CANopen 主站的特点和可移植性的要求，提出了一种标准 C 语言实现的非抢占式的任务调度结构。由于没有使用抢占的方式，极大地减少了非原子操作和死锁的发生概率。由于只运用了标准 C 而不利用系统调用，该设计可运行在任何支持 C 语言编译器的平台。实验表明，该方法已满足 CANopen 网络的实时要求。

该算法也是一种通用的任务调度算法，并不局限于在 CANopen 主站中的使用。稍作修改和优化，就可运用到其他需要高度实时性和并发处理能力应用当中。

致谢：本文作者感谢北京工业大学电子信息与工程学院段建民教授的指导，并感谢肖进军、闫士珍和张卓同学的参与。

参考文献

[1] Mohammand Farsi, Karl Ratcliff. CANopen: The Open Communications Solution[C] // Proceedings of the IEEE International Symposium on Industrial Electronics, 1997, 1(1): 112-116.

[2] CANopen - Application Layer and Communication Profile[S] // CiA Draft Standard 301, Version 4.02, February 12, 2002.

[3] CANopen - Framework for CANopen Managers and Programmable CANopen Devices[S] // CiA Draft Standard Proposal 302, Version 3.1.2, June 5, 2002.

[4] Robert Love. Linux 内核设计与实现[M]. 北京: 机械工业出版社, 2004: 25-41.

[5] William Stallings. 操作系统——内核与设计原理[M]. 北京: 电子工业出版社, 2001: 346-347.

[6] 陈广涛, 戴胜华. Windows CE.NET 实时性能的测试与研究[J]. 微计算机应用, 2006, 27(6): 735-738.

[7] 刘淼, 王田苗, 魏洪兴, 陈友东. 基于 uCOS-II 的嵌入式数控系统实时性分析[J]. 计算机工程, 2006, 22(11): 222-224, 226.

[8] Olaf Pfeiffer, Andrew Ayre, Christian Keydel. Embedded Networking with CAN and CANopen[M]. RTC Books, San Clemente, CA, 2003.