面向 NB-IoT 的微内核 RTOS 的设计与实现

张 正,贾小林*
(西南科技大学 计算机科学与技术学院 RFID&IOT 实验室,四川 绵阳 621000)

摘 要:实时操作系统(Real-Time Operating System,RTOS)被广泛应用于窄带物联网(Narrow Band Internet of Things,NB-IoT)设备之中。这类设备对体积、能耗与稳定性有着严格的限制。NB-IoT 设备多采用宏内核的 RTOS,能得到较好的运行性能,但要求更多的硬件资源,并且内核中出现的问题很可能会导致整个系统崩溃。该文对传统 RTOS 进行改进,设计开发了无内存管理单元(Memory Management Unit,MMU)的微内核实时操作系统(nM-MKRTOS)。该系统针对 NB-IoT 中资源较少的设备,利用微内核的优势,其通过动态加载与链接(Dynamic Loading and Dynamic Linking,DL^2)技术实现内存复用和快速启动,并采用模块化开发的方式提高系统稳定性。在实际测试中,nM-MKRTOS 通过内存复用技术将内存利用率提高了 56.25%;在系统的启动测试中,通过在 DL^2 技术中引入权重加载,系统的核心功能在三个任务子集上的启动时间分别减少 57.59%、52.55% 与 47.59%。该系统能够广泛应用于智慧农业、智慧校园等场合,能够降低系统成本,提高系统稳定性。

关键词:微内核;实时操作系统;窄带物联网;动态加载;动态链接
中图分类号:TP316.2 **文献标识码**:A **文章编号**:1673-629X(2022)10-0076-06
doi:10.3969/j.issn.1673-629X.2022.10.013

Design and Implementation of Microkernel RTOS for NB-IoT

ZHANG Zheng,JIA Xiao-lin*
(RFID&IOT Lab,School of Computer Science and Technology,Southwest University of Science and Technology,Mianyang 621000,China)

Abstract:Real-Time Operating System (RTOS) is widely used in Narrow Band Internet of Things (NB-IoT) devices. NB-IoT devices mostly use RTOS with macro kernel,which can get better performance but requires more hardware resources and problems in the kernel may cause the whole system to crash. In this paper,we improve the traditional RTOS and design and develop the nM-MKRTOS without Memory Management Unit (MMU). Aiming at the devices with fewer resources in NB-IoT,the system realizes memory reuse and fast startup through Dynamic Loading and Dynamic Linking (DL^2) according to the advantages of microkernel,and adopts modular development to improve system stability. In the actual test,nM-MKRTOS has improved memory utilization by 56.25% through memory reuse technology. In the system startup test,by introducing weighted loading in DL^2 technology,the startup time of the core functions of the system was reduced by 57.59%, 52.55% and 47.59% for the three task subsets respectively. The system can be widely used in smart agriculture,smart campus, etc.,which can reduce system cost and improve system stability.

Key words:microkernel;RTOS;NB-IoT;dynamic loading;dynamic linking

0 引 言

NB-IoT 被广泛应用于环境监控、安全检测等重要领域,这类设备的硬件资源通常非常有限(通常为 IETF 定义的三类设备[1]),其设备中大多运行宏内核 RTOS,例如:RT-Thread[2]、FreeRTOS[3]、uC/OS-II[4]。这种宏内核的设计思想,整个系统组件、设备驱动、内存管理、线程调度、网络栈等都被包含在特权内核中。内核中出现的错误将会导致整个系统的崩溃,并且内核之间的模块可能出现潜在的相互依赖使得模块化程度降低,此外宏内核的初始化中包含了许多导致系统启动变慢的流程,会影响系统的开机速度。为解决以上问题,出现了各类的微内核操作系统,例如:Chorus[5]、Mach[6]、L4[7]、seL4[8]等。这类内核通常只提供以下几类硬件抽象:(1)虚拟内存管理:将虚拟地

收稿日期:2021-11-17 **修回日期**:2022-03-23
基金项目:国家自然科学基金面上项目(61471306);四川省重点研发计划项目(2020YFS0360);四川省教育厅资助科研重点项目(18ZA0488)
作者简介:张 正(1995-),男,硕士研究生,CCF 会员(D3838G),研究方向为嵌入式技术、物联网技术;通信作者:贾小林,博士,教授,CCF 高级会员(11988S),研究方向为射频识别技术、物联网技术、计算机应用技术。

址映射到物理地址;(2)线程抽象和调度;(3)进程间通信(IPC)。其余的文件系统、网络协议栈等都被隔离在内核之外。

在 NB-IoT 设备上实现微内核有一定的难度,主要原因在于其处理器上不包含 MMU,整个系统只有单地址空间,DL^{2}实现困难。DL^{2}作为微内核实现的重要技术,保证了微内核系统中的软件模块可以实时动态地加载到系统中进行运行。目前,适用于 NB-IoT 设备的操作系统比较多:(1)Contiki[9]、uC/OS-II、uC/OS-III[10]、RIOT[11]、FreeRTOS[3]、uCLinux[12]、Linux、SenSpire OS[13]、RT-Thread、mbedOS[14]、TinyOS[15]等操作系统在物联网上得到广泛使用,但都为宏内核操作系统,甚至部分操作系统还并不支持 DL^{2}技术;(2)seL4、VxWorks[16]、Chorus 等在设计之初就被设计为微内核操作系统,支持 DL^{2}技术,但是内存占用较高,在 NB-IoT 设备中并不适用[8]。

目前,对于 NB-IoT 设备还没有适用的微内核操作系统,该文针对 NB-IoT 设备硬件资源较少的情况,对现有的 DL^{2}技术进行优化,设计并实现了无 MMU 的 nM-MKRTOS,实现了加载速度和传输开销的优化。nM-MKRTOS 通过优先加载配置策略,实现了核心功能的优先加载,提高 NB-IoT 系统的启动速度。此外利用 DL^{2}技术,实现了模块间的内存复用策略,能够减少 NB-IoT 系统的成本,提高内存利用率。

1　微内核的设计

FreeRTOS、RT-Thread 等系列操作系统,在设计之初就采用了宏内核的设计方式,宏内核通过统一编译的方式,在运行阶段会带来一定的性能提升。但也导致宏内核需要设计者做大量的配置工作,并且其大量组件被一同编译运行,会拖慢开机速度。该文设计的 nM-MKRTOS 内核包含内存管理、任务间通信(Inter Process Communication,IPC)、调度管理模块与 DL^{2}模块,如图 1 所示。

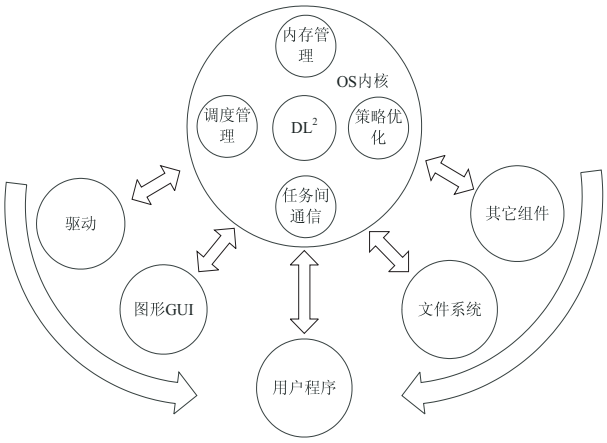

图 1　微内核操作系统结构

(1)调度管理模块:该模块进行任务调度,采用抢占式任务调度算法。

(2)任务间通信模块:任务间通信通过共享内存、消息队列等方式实现。

(3)内存管理模块:对系统的内存进行管理。

(4)DL^{2}模块:DL^{2}模块用于对模块进行快速加载并执行。对于每一个动态加载的模块,会进行初始化;对于每一个动态加载的应用,会初始化并创建进程。

(5)策略优化模块:该模块包含了加载策略优化,用于提高系统的开机速度;此外还包含了内存“卸载”功能,通过将挂起进程的内存“卸载”到外部存储模块,达到内存复用的目的。

不同的微内核系统的设计结构基本一致,故模块 2、3 不是该文描述的重点;模块 1 中包含了抢占式任务调度程序,其任务状态与普通 RTOS 有差异;模块 4 中包含了 DL^{2}模块,用于在设备上实现程序模块的动态加载,是提高系统开机速度和实现内存复用的核心技术;模块 5 实现了开机加载策略并实现了内存复用,是核心模块。

2　调度管理模块

任务调度算法为微内核的核心模块,对于 NB-IoT 来说,系统要求对于某些紧急任务做出最快的处理(如:环境检测报警),这就要求系统拥有一定的实时性。该系统中的任务调度算法,采用抢占式设计,但支持同优先级任务时间片轮转执行。系统任务调度算法示意图如图 2 所示。

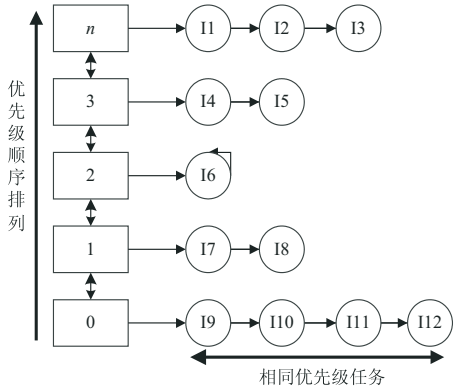

图 2　实时调度算法

系统维持一个按照优先级顺序排列的双向链表,链表的每一个节点中存放的是同优先级的任务。变量 x 将存放最高优先级任务的头节点,每次最高优先级任务被挂起或阻塞时,会查找最高优先级节点以更新 x。在 x 未发生变化时,其调度算法的时间复杂度为 $O(1)$,在发生变化时,其调度算法复杂度为 $O(n)$,n 为系统中优先级的个数。在 x 发生变化时,利用位图的方式进行查找,其算法复杂度还可以优化,但是要求

系统最大任务数与优先级必须在编译时就确定,使得系统灵活性降低,所以这里不进行论述。

图 3 中,系统中的任务总共包含 6 种状态:挂起、就绪、运行、卸载、阻塞、被阻塞。挂起状态为用户执行 Suspend 操作时挂起任务,只有执行 Resume 才能将任务恢复为就绪状态;阻塞状态为任务延时、等待事件等都会进入阻塞状态;卸载状态为用户执行 Unload 操作将任务“卸载”到外部存储设备,只有执行 Load 操作才能够恢复,并执行 Unload 与 Load 将会占用较多的时间。

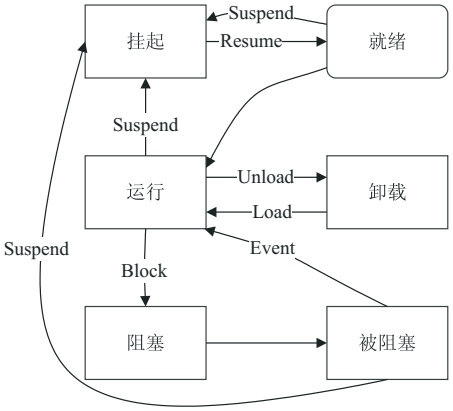

图 3 任务状态转换示意图

3 DL²模块

DL²用于微内核中模块动态加载与执行,可加载文件体积与加载速度都会影响系统性能。在许多物联网操作系统中,如 SenSpire OS[13],都实现了 DL²,并且针对其使用的场合进行优化。该文参考了文献[17-18]中 DL²的实现,并优化可加载文件大小与符号表。

3.1 可加载文件大小优化

在标准的可执行可链接格式(Executable Linkable Format,ELF)文件中包含了许多的段,如图 4 所示,然而这些段不参与程序的执行,通过对这些段的优化,能够减少可加载文件的体积,在网络传输时能够明显提高速度,减少能耗。

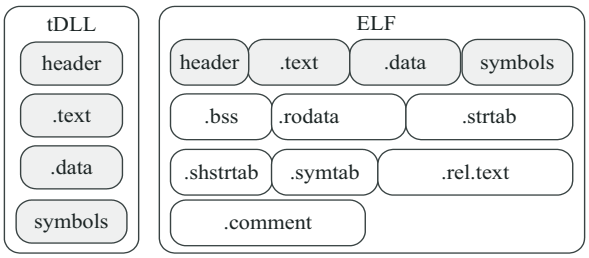

图 4 tDLL 文件格式与 ELF 文件格式

3.2 符号表优化

在标准 ELF 中,符号表被分成了很多类,并且某些符号表之间还存在一定的关系。符号表中存储了函数、变量、重定向变量等信息,系统依靠符号表完成函数的重定向,ELF 在符号表中利用 Hash 表对符号表进行快速查找,但是当符号数量增多时,Hash 表的碰撞会导致性能下降。该文对符号表进行了修改与整合,如表 1 所示,在符号表中添加预链接信息(目标符号在目标库中的偏移地址),在进行重定向时,通过使用预链接信息则无需在目标符号表中进行查找,查找效率为 $O(1)$。

表 1 符号表

符号项大小 (2 字节)	类型 (1 字节)	符号名	符号偏移 (4 字节)	目标 库名	预链接信息 (4 字节)
11	F	dy. tDLL	无效	无效	无效
23	0x80 | ‘P’	add	8	cal. tDLL	14
25	0x80 | ‘P’	sort	16	sort. tDLL	32
12	T	main	4	无效	无效

4 策略优化模块

4.1 提高系统开机速度

系统内核的加载、系统硬件初始化、内核软件的初始化、用户软件的初始化等都会影响系统开机速度。在使用宏内核操作系统的设备中,系统通常包含了众多的功能,整个软硬件系统全部初始化完毕则标志着整个系统启动完毕。然而对于使用者来说通常只关心一部分功能亦或者一部分功能需要提前启动,而这部分功能的启动只占用整个系统启动的一部分时间。利用微内核的 DL²所具备的优势,将系统任务按照启动权重进行划分,利用优先加载策略,能够实现系统快速启动。

NB-IoT 的系统可能由若干模块组成,这些模块可能是对某个软件或者硬件的配置,也可能是需要执行某些特定的工作。整个系统模块可由图 $G=(V,E)$ 表示,图之间的权重用模块初始化执行的时间 t 表示。在图 5 中,任务可以表示为公式(1),分别代表系统所需要的模块子集 J_1、J_2 和 J_3。可得系统启动的时间为 $T_t=\sum_{i=1}^{n} t_i$,单个功能子序列 J 的启动时间为 $T_j=\sum_{j=1}^{m} t_j$,其中 n 为系统总模块数,m 为模块序列任务数,且 $T_t \geqslant T_j$。

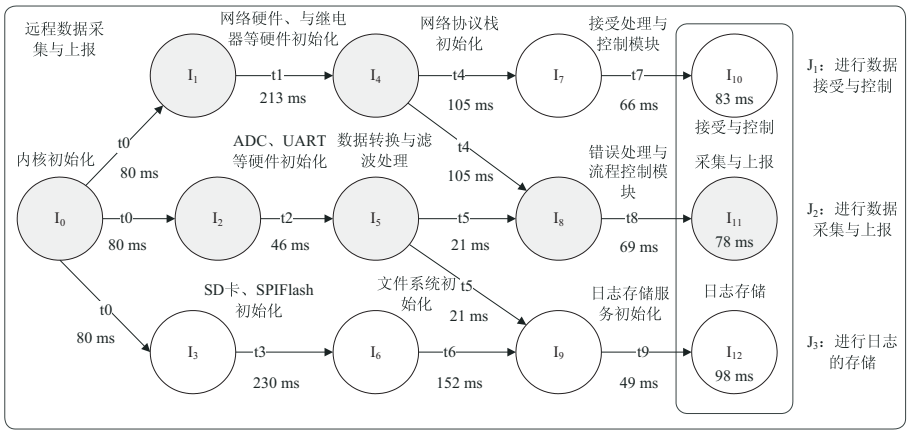

图 5　系统模块划分有向加权图

在整个系统中,不同功能对应不同的模块子集序列,这些序列通过施加权值的方式决定其启动的顺序。例如:任务 I 中希望序列 J_2 被最先加载执行,则对于子集权重的分配为公式(2)~公式(4)。

$$I=\begin{pmatrix} J_1=\{I_0,I_1,I_4,I_7,I_{10}\} \\ J_2=\{I_0,I_1,I_2,I_4,I_5,I_8,I_{11}\} \\ J_3=\{I_0,I_2,I_3,I_5,I_6,I_9,I_{12}\} \end{pmatrix} \tag{1}$$

$$J_1=\{I_0,I_1,I_4,I_7,I_{10}\}*k_1 \tag{2}$$

$$J_2=\{I_0,I_1,I_2,I_4,I_5,I_8,I_{11}\}*k_2 \tag{3}$$

$$J_3=\{I_0,I_2,I_3,I_5,I_6,I_9,I_{12}\}*k_3 \tag{4}$$

其中,$k_2>(k_1,k_3)\in N$,上述公式表示,J_2 拥有最高权重,其任务功能序列最先被加载。

4.2　内存复用优化

在有 MMU 的处理器中,通过“交换内存”实现内存的复用,即通过外部存储设备作为内存存放的临时区域,使得运行程序的规模能够超过内存所支持的大小。但 NB-IoT 设备通常没有 MMU,开发者通常只能够使用硬件所限制的内存大小。该文利用微内核的优势,提出模块内存“卸载”策略,即在系统中执行的任务具有时间上的不相关性,则这些任务也可以使用类似“交换内存”的方式实现内存的重复利用。

内存复用能够使得系统运行更多的应用,但也带来了性能损失,通常将其用于对应用要求不高的场合,这对于 NB-IoT 的场合非常适用,在 NB-IoT 场景中其通常要求周期性地进行工作,对系统的时延要求并不严格。在微内核的系统中运行的任务可以用图 6 表示,定义任务规则 $P=(r,k,I,b,p,a,R_t,t)$,其中:

(1) r 为 RAM 所需空间,$r<$系统可用 RAM 空间;

(2) k 为 ROM 所需空间,$k<$系统可用 ROM 空间;

(3) I 为之后需要执行的任务集(为-1 代表下一个任务为空);

(4) b 为前驱任务集;

(5) p 为执行该模块前需执行的动作(0:保持,1:挂起;2:恢复;3:卸载到外部内存;4:加载到内部内存);

(6) a 为当前任模块到下一个模块所执行的动作;

(7) R_t 为模块运行间隔;

(8) t 为运行时间。

在图 6 中,通过配置 r、k、I、p 与 a 可以指定模块 I_1 与 I_2 的启动顺序与动作,通过配置 R_t 与 t 可以实现时间不相关模块的内存复用(如:I_2 与 I_1)。

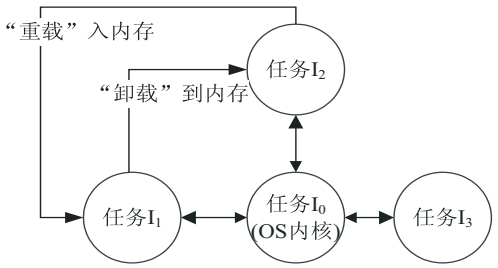

图 6　任务内存复用示意

5　实验分析

实验中,使用了真实的硬件设备,见图 7,其系统 RAM 为 64 KB,系统 ROM 为 512 KB,系统主频为 72 MHz,在本节的所有实验中,都采用该配置的处理器。

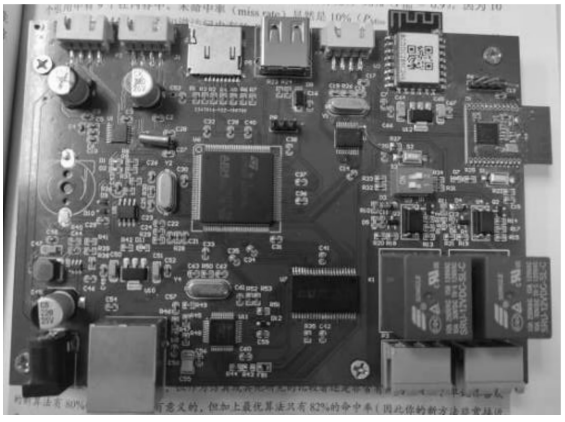

图 7　测试硬件

5.1 可加载文件性能分析

该文对可加载文件的大小以及加载速度进行了优化,为此编写了 7 个模块,对比 ELF 可加载文件在相同模块下的体积大小。在图 8 中,该文的可加载文件体积相比于 ELF,是标准 ELF 文件大小的 10.4% ~ 68.4% 。可加载文件体积的优化,意味着在 NB-IoT 网络传输过程中能够节省更多的时间与能耗。

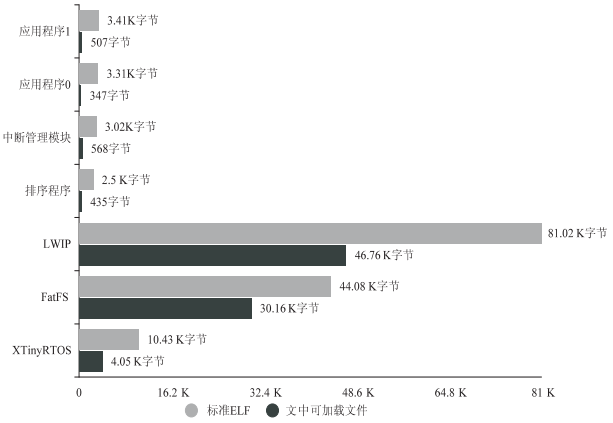

图 8 可加载文件大小对比

5.2 可加载文件执行速度测试

为对比该可加载文件与 ELF 文件在加载后执行性能的差异,利用几种经典的排序算法进行性能比较,测试结果如图 9 所示。在几种排序算法中,只有在选择排序测试中,性能损失了 6.1% ,其他排序算法在对比中几乎没有差异。

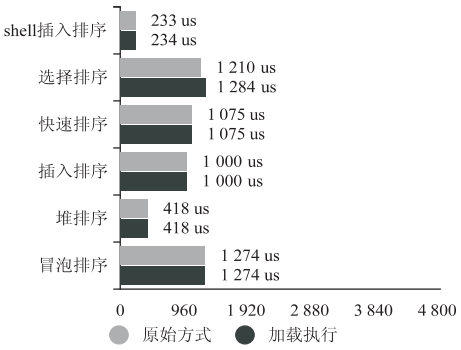

图 9 执行性能对比

5.3 系统启动速度测试

为测试系统启动速度,编写了一个测试程序,该程序与真实场景相对应,其模块划分见图 5,系统执行远程数据采集、上报与控制任务,其包含三个任务子集 J_1、J_2 与 J_3,每个子集包含若干模块,定义其启动规则为公式(5) ~公式(7)。

$$J_1=\begin{cases}I_0\\ I_1\\ I_4\\ I_7(10\text{KB},15\text{KB},I_{10},I_4,0,0,-1,66\text{ms})\\ I_{10}(0.5\text{KB},8\text{KB},-1,I_7,0,0,-1,83\text{ms})\end{cases}\tag{5}$$

$$J_2=\begin{cases}I_0(2\text{KB},10\text{KB},(I_1,I_2,I_3),-1,0,0,-1,80\text{ms})\\ I_1(0.1\text{KB},21\text{KB},I_4,I_0,0,0,-1,213\text{ms})\\ I_2(0.1\text{KB},22\text{KB},I_5,I_0,0,0,-1,46\text{ms})\\ I_4(30\text{KB},54\text{KB},(I_7,I_8),I_1,0,0,-1,105\text{ms})\\ I_5(1\text{KB},5\text{KB},(I_8,I_9),I_2,0,0,-1,21\text{ms})\\ I_8(0.5\text{KB},5\text{KB},I_{11},(I_4,I_5),0,0,-1,69\text{ms})\\ I_{11}(1.5\text{KB},10\text{KB},-1,I_8,0,0,-1,78\text{ms})\end{cases}\tag{6}$$

$$J_3=\begin{cases}I_0\\ I_2\\ I_5\\ I_3(2\text{KB},10\text{KB},I_6,I_0,0,0,-1,230\text{ms})\\ I_6(9.8\text{KB},35\text{KB},I_9,I_3,0,0,-1,152\text{ms})\\ I_9(2\text{KB},15\text{KB},I_{12},(I_5,I_6),0,0,-1,49\text{ms})\\ I_{12}(0.5\text{KB},9\text{KB},-1,I_9,0,0,-1,98\text{ms})\end{cases}\tag{7}$$

在系统未按权重进行加载时,系统总启动时间为 1 290 ms,在分别设置模块子集加载权重为:(2,1,1)、(1,2,1)、(1,1,2),三个模块子集其启动时间分别为:547 ms,612 ms,676 ms,即权重加载启动方式相比于未按权重加载其三个模块的启动时间分别减少了 57.59% 、52.55% 与 47.59% 。

5.4 内存复用测试

在内存复用测试的测试中,编写与 4.2 节中相同的应用测试模块,4 个应用模块的加载规则为公式(8)。

$$J_4=\begin{cases}I_0(10\text{KB},15\text{KB},(I_1,I_2,I_3),-1,0,0,-1,60\text{ms})\\ I_1(40\text{KB},18\text{KB},I_2,I_0,4,3,2000\text{ms},189\text{ms})\\ I_2(40\text{KB},18\text{KB},I_1,I_0,4,3,1000\text{ms},78\text{ms})\\ I_3(10\text{KB},30\text{KB},-1,I_0,0,0,5000\text{ms},123\text{ms})\end{cases}\tag{8}$$

在规则中,I_0 为系统 OS 内核;I_1 与 I_2 为应用程序,执行周期分别为 2 s 与 1 s,其为周期任务,可以做到时间上不相交,并且 I_1 与 I_2 同时运行其需要的 RAM 空间必然超过系统总空间。通过内存复用可提高 RAM 内存的利用效率,在本测试中,内存利用率提高 56.25% 。

6 结束语

针对 NB-IoT 设备资源少、稳定性要求高、系统对外界响应性强等特点,设计并实现了 nM-MKRTOS。该系统采用微内核的设计思想,其任务调度算法采用抢占式调度策略,利用模块优先加载策略提高系统的开机速度,并且针对时间上不相关的模块,实现了内存

复用策略,提高了内存的利用率。在编写的测试案例中,系统的开机时间分别减少 57.59%、52.55% 与 47.59%,内存利用率提高 45.83%。此外,该系统采用微内核的思想,其能够最大程度上保证内核的安全,防止驱动、文件系统等出现的问题影响系统的工作。nM-MKRTOS 系统能够应用于环境监测、智慧农业等各个场合,在降低系统成本的同时,还能够提高系统的安全性与稳定性。

参考文献:

[1] Internet Engineering Task Force. Terminology for constrained node networks[EB/OL]. 2014. http://www.ietf.org/rfc/rfc7228.txt.

[2] XIONG Puxiang. RT-Thread[EB/OL]. 2021. https://www.rt-thread.org/.

[3] BARRY R. A FREE open source RTOS for small embedded real time systems[EB/OL]. 2008. http://www.freertos.org.

[4] Micrium. uC/OS-II[EB/OL]. 1992. http://micrium.com/rtos/ucosii/overview/.

[5] ROZIER M, ABROSSIMOV V, ARMAND F, et al. Overview of the CHORUS distributed operating systems[J]. Computing Systems,1991,1(10):39-69.

[6] GOLUB D B,DEAN R W,FORIN A,et al. UNIX as an application program[C]//USENIX summer. Baltimore:JETS,1990:87-95.

[7] HEISER G,ELPHINSTONE K. L4 microkernels:the lessons from 20 years of research and deployment[J]. ACM Transactions on Computer Systems,2016,34(1):1-29.

[8] XU L,BAI Y,CHENG K,et al. Towards fault-tolerant real-time scheduling in the seL4 Microkernel[C]//IEEE international conference on high-performance computing & communications,IEEE international conference on smart city. Sydney:IEEE,2016:711-718.

[9] DUNKELS A. Contiki operating system[EB/OL]. 2013. http://www.contiki-os.org.

[10] TECHCON A. Micrium makes uC/OS-III RTOS source code available[EB/OL]. 2013. http://eetimes.com.

[11] BACCELLI E,HAHM O,GÜNES M,et al. RIOT OS:towards an OS for the Internet of Things[C]//2013 IEEE conference on computer communications workshops (INFOCOM WKSHPS). Turin:IEEE,2013:79-80.

[12] 朱玮玮,杨建明. uClinux——一种嵌入式 Linux 系统[J]. 舰船电子工程, 2003(4):47-50.

[13] DONG W,CHEN C,LIU Y,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.

[14] 刘长勇,王宜怀,蔡闯华,等. 实时操作系统 mbedOS 启动流程剖析[J]. 计算机工程与应用,2020,56(20):46-51.

[15] LEVIS P,MADDEN S,POLASTRE J,et al. TinyOS:an operating system for sensor networks[M]//Ambient intelligence. Berlin:Springer,2005:115-148.

[16] 刘元元. 基于 VxWorks 操作系统的某随动系统控制算法软件设计与实现[J]. 火炮发射与控制学报,2021,42(3):76-81.

[17] 张　正,贾小林. 面向 NB-IOT 智能设备动态链接库的远程技术研究及应用[J]. 计算机应用与软件,2021,38(6):170-175.

[18] HAN C C,KUMAR R,SHEA R,et al. A dynamic operating system for sensor nodes[C]//Proceedings of the 3rd international conference on Mobile systems,applications,and services. Seattle:Association for Computing Machinery,2005:163-176.