计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2020,29(8):217−223 [doi: 10.15888/j.cnki.csa.007560]
©中国科学院软件研究所版权所有.
E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

GEC 架构下 RTOS 对外接口函数重映射机制①

刘长勇[1,2,3], 王宜怀[2], 孙亚军[2]

[1](武夷学院 数学与计算机学院, 武夷山 354300)
[2](苏州大学 计算机科学与技术学院, 苏州 215006)
[3](认知计算与智能信息处理福建省高校重点实验室, 武夷山 354300)
通讯作者: 刘长勇, E-mail: 121903852@qq.com

摘　要: 实时操作系统 (RTOS) 是嵌入式人工智能与物联网终端中重要工具, 不同的机构开发的 RTOS 其实时性、调度规则、任务间通信机制等稍有差异, 但基本要素相同. 基于通用嵌入式计算机 (GEC) 架构研究了 RTOS 在 BIOS 驻留方法及对外接口函数重映射机制, 以 NXP 的 KL36 芯片为例给出了 mbedOS 在 BIOS 中的驻留实现, 并给出对外接口函数重映射实例. 实践表明 RTOS 驻留在 BIOS 中, 不仅缩短了编译链接时间, 同时通过对外接口函数的重映射, 简化了对 RTOS 调度机制的理解, 降低了编程难度, 为有效地实现不同 RTOS 下应用程序的可移植性提供了技术基础.

关键词: 实时操作系统; 通用嵌入式计算机; 对外接口函数重映射; BIOS; KL36

引用格式: 刘长勇,王宜怀,孙亚军.GEC 架构下 RTOS 对外接口函数重映射机制.计算机系统应用,2020,29(8):217–223. http://www.c-s-a.org.cn/1003-3254/7560.html

RTOS External Interface Function Remapping Mechanism under GEC Architecture

LIU Chang-Yong[1,2,3], WANG Yi-Huai[2], SUN Ya-Jun[2]

[1](Department of Mathematics and Computer Science, Wuyi University, Wuyishan 354300, China)
[2](School of Computer Science and Technology, Soochow University, Suzhou 215006, China)
[3](Key Laboratory of Cognitive Computing and Intelligent Information Processing of Fujian Education Institutions, Wuyishan 354300, China)

Abstract: Real-Time Operating System (RTOS) is an important tool in embedded artificial intelligence and IoT terminals. RTOS developed by different institutions has slight differences in real-time properties, scheduling rules, and communication mechanisms between tasks, but the basic elements are the same. This work is based on the general embedded computer(GEC) architectures, studies the RTOS resident method in BIOS and the remapping mechanism of external interface function. Taking the KL36 chip of NXP as an example, the resident implementation of mbedOS in the BIOS is given, and living example of external interface function remapping is given. Practice shows that the RTOS resides in the BIOS, which can shorten the compilation and linking time. At the same time, by remapping external interface functions, the understanding of the RTOS scheduling mechanism is simplified, the programming difficulty is reduced, and the technical basis is provided for effectively implementing the portability of applications under different RTOS.

Key words: real-time operating system; general embedded computer; external function interface remapping; BIOS; KL36

① 基金项目: 国家自然科学基金 (61672369); 福建省自然科学基金 (2017J01651)
Foundation item: National Natural Science Foundation of China (61672369); Natural Science Foundation of Fujian Province (2017J01651)
收稿时间: 2020-01-09; 修改时间: 2020-02-08; 采用时间: 2020-03-11; csa 在线出版时间: 2020-07-29

© 中国科学院软件研究所 http://www.c-s-a.org.cn

为了提升编程颗粒度、提高可移植性, 借鉴通用计算机的概念与做法, 把基本输入输出系统 (Basic Input and Output System, BIOS) 与用户程序分离开来, 实现彻底的工作分工, 形成了通用嵌入式计算机 (General Embedded Computer, GEC)[1]. GEC 架构将嵌入式软件系统分为 BIOS 工程程序 (简称 BIOS) 和 USER 工程程序 (简称 USER) 两部分, BIOS 先于 USER 固化于微控制器 (Microcontroller Unit, MCU) 内的非易失存储器 (如 Flash) 中, 为实时操作系统 (Real-Time Operating System, RTOS) 的驻留提供了空间. 实时操作系统能提供精确的实时控制和任务管理功能, 保证系统的实时性需求[2].通过将 RTOS 驻留在 BIOS 中, 降低用户的编程难度、简化程序的串口写入以及方便用户调用, 同时也可以很好地帮助用户解决因 RTOS 在不同开发环境的编译困难而造成的烦恼.

同时, 为了充分发挥 RTOS 的功能, 方便用户使用, 提高用户程序的可移植性, 通过提供对外函数接口的形式是一种比较好的做法. 对外函数接口 (也称为应用程序编程接口, API) 是软件库提供的一组可访问的接口, 软件库通过 API 向外提供服务, 开发人员通过使用 API 实现代码复用, 提高生产效率[3].因此, 在对外函数接口设计上, 要充分考虑其可用性[4,5]、稳定性[6] 和安全性[7,8].目前, 对外函数接口的研究已取了一定的研究成果, 主要集中在 API 使用规约[9–11]、API 推荐研究[12,13]、API 文档研究[14,15]、API 组合模式的应用[16] 等方面, 但有关 RTOS 的驻留及其对外函数接口研究方面文献较少. 为此, 本文首先给出通用嵌入式计算机架构下 RTOS 的 BIOS 驻留方法, 剖析了 RTOS 对外函数接口设计方法, 提出了接口函数重映射机制, 最后在 mbedOS 下进行应用实践. 实践表明, 将 RTOS 驻留在 BIOS 中, 能有效地节省用户程序的编译时间, 同时通过对外接口函数的重映射, 能有效地提升了应用程序的可靠性性和开发效率, 易于用户调用, 也使应用程序易于复用, 为嵌入式人工智能与物联网终端程序的开发提供了技术基础.

1 RTOS 的 BIOS 驻留方法简介

在 GEC 架构中, 虽然嵌入式软件系统分为 BIOS 和 USER 两部分, 但最终程序代码和各种变量数据都是放在同一个 MCU 的 Flash 和 RAM 中, 要将 RTOS 驻留在 BIOS 中, 实现 RTOS 与应用程序的物理隔离, 就必须对 MCU 的 Flash 和 RAM 空间进行合理的划分, 这样才能确保代码不重叠, 变量使用不越界, 从而保证 RTOS 能得到正常运行, 而又不影响 USER 的执行. 因此, 要考虑 RTOS 驻留的硬件载体, 即 MCU 的 Flash 和 RAM 空间大小的因素. 对 Flash 空间的划分可采用分割独享方式, 一部分给 BIOS 程序使用, 另一部分给 USER 程序使用, 两者使用的空间不重叠. 对 RAM 空间的划分则需要考虑 MCU 的 RAM 空间大小, 当 RAM 空间足够大时, 可采用分割独享方式, BIOS 和 USER 各单独使用不重叠的空间; 当 RAM 空间较小时, 可采用重叠共享方式, 即 BIOS 使用的空间和 USER 使用的空间部分重叠共享, 这样可以提高 RAM 空间的利用率. 在实现 RTOS 驻留的过程中, 还需要考虑如何合理划分 Flash 空间, 使 USER 程序占用的空间尽量大; 重叠共享方式分配 RAM 空间时, 如何避免出现数据越界与冲突; RTOS 的调度会依赖于系统服务调用, 何时将这些调用权移交给 RTOS 等问题.

2 RTOS 对外函数接口设计方法

虽然 RTOS 已驻留在 BIOS 中, 但要发挥其作用, 还需将 RTOS 的功能函数设计成对外函数接口表, 然后通过映射形成对外函数映射表, 这样才能向 USER 提供服务. 也就是说, USER 可以通过对外函数映射表实现对 RTOS 提供的接口函数的调用.

2.1 设计 RTOS 对外函数接口的必要性

在 GEC 架构下, 将 RTOS 提供的函数进行对外接口的设计, 不仅能够发挥 RTOS 的功能, 而且还能方便用户使用, 提高 USER 程序的可复用性. 因此, RTOS 向用户提供接口函数是非常必要的.

(1) 提高应用程序的可复用性. 由于 USER 是通过对外函数映射表实现对 RTOS 提供的对外接口函数调用的. 因此, 当在 BIOS 中驻留不同的 RTOS 时, 只要向 USER 提供相同功能的接口就可以, 即使这些接口的名称发生的改变, 也不会影响 USER 的调用, USER 程序不需要修改, 提高了 USER 程序的可复用性.

(2) 提升应用程序的稳定性和可靠性. 由于 RTOS 已经驻留在 BIOS 中, 它所提供的对外函数功能已经在 BIOS 中通过编译、测试和验证, 变成了一段可靠的、稳定的机器码. 因此, 在 USER 程序中可以放心地调用 RTOS 提供的对外函数, 不用担心会出现的代码错误, 从而提升了应用程序的稳定性和可靠性.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

(3) 缩短应用程序的开发时间. 由于 RTOS 提供了原型级的对外函数调用接口, 用户可以直接使用, 无需花大量精力深入理解 RTOS 的工作原理和调度机制, 不需要知道具体的实现细节, 只需关注用户程序的编写, 大大地提高了开发效率.

(4) 方便应用程序调用. 由于 RTOS 提供的对外函数已经变成了机器码驻留于 BIOS 内, 用户难以调用, 通过对这些函数进行重映射, 最终向 USER 提供函数调用原型接口, 用户就可以像调用普通函数一样方便使用这些函数.

2.2 RTOS 对外函数接口机制

要实现 RTOS 对外函数接口, 首先需要在 BIOS 中对函数进行重定义、声明、注册, 形成对外函数接口表; 其次要在 USER 中通过映射获取对外函数接口表的入口地址, 形成对外函数映射表, 并重定向函数名称, 最后在 USER 中实现对函数的调用, 其过程如图 1 所示.

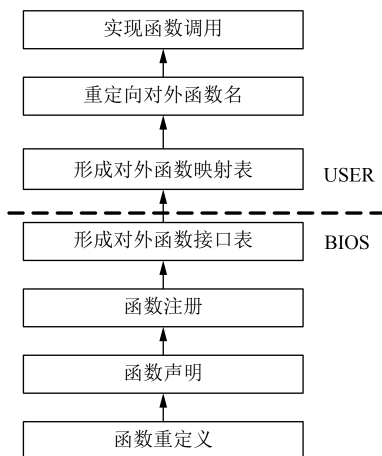

图 1 对外函数接口的设计与实现过程示意图

(1) 对外函数的二次封装

对外函数不仅可以包含 RTOS 的功能函数, 而且还可以包括各类构件函数, 本文主要介绍如何封装 RTOS 提供的对外接口函数. RTOS 一般都具备线程管理、同步与通信、中断管理等基本功能, 相应的提供了线程类、事件类及操作系统启动函数等, 在这些类中提供了大量的成员函数, 都是采用 C++实现的. 由于这些成员函数的执行依赖于类对象的创建及其成员变量, 而不是使用绝对地址的方式来实现对类中的某个成员函数的调用. 因此, 必须在 BIOS 程序中对这些函数进行重定义, 二次封装成 C 语言可以调用的函数形式, 这样才能实现在 USER 中调用它们. 对外函数的二次封装 (或称重定义) 主要包括函数名、函数的返回值类型、函数的参数、函数体以及为了便于理解程序而加入的功能说明和代码注释等. 其格式如下:

```
格式: void 重定义函数名 (参数表列)
{
    [ 相关变量声明 ]
    调用 RTOS 函数
}
```

例如, mbedOS 的延时函数名为 Thread::wait, 通过二次封装重定义为 thread_wait.

(2) 对外函数的声明

对外函数重定义好之后, 一般应在与之同名的.h 头文件中进行声明, 函数的声明要给出函数名、函数的返回值类型、函数的参数以及函数的功能说明, 即使用者通过函数的声明就能了解函数的功能和使用方法, 而不需要查看函数的具体实现.

格式: void 重定义函数名 (参数表列);

(3) 对外函数的注册

当对外函数定义和声明之后, 还要对函数进行注册才能形成对外函数接口表. 借鉴中断向量表的定义做法, 可以给所有的或部分的函数编号, 并将函数名 (即函数的入口地址) 集中在一起按编号有序地放在一个统一的区域中, 形成对外函数接口表. 在对外函数接口表中, 对外接口函数的入口地址用 32 位的二进制表示, 可以看作和定义成 long 类型的数据, 一般采用汇编语言编写一个 SVC 中断来注册. 对外函数接口表采用数组 (如 BIOS_API) 存储, 其入口地址就是数组名或数组的首地址, 函数的编号与数组的下标元素的序号一一对应, 其中,0 号表示对外函数的数量,1 号函数对应 BIOS_API[1], 2 号函数对应 BIOS_API[2], 依此类推, 换句话说, 可以通过数组元素来访问这些对外函数. 同时, 为了便于扩充或更新对外函数的个数, 还预留了一些缺省的函数名 (如 DefaultFUN).

```
格式: BIOS_API:        //对外函数接口表数组
.long 函数个数          //0 号表示函数个数
.long 重定义函数名      //从 1 开始对函数编号
……
例如: BIOS_API:
.long 134              //共有 134 个对外函数
```


© 中国科学院软件研究所 http://www.c-s-a.org.cn

```
.long OS_start       //1 号对外函数 (启动操作系统)
……
.long thread_wait    //4 号对外函数 (延时函数)
.long DefaultFUN     //预留缺省函数
```

3 接口函数重映射机制

当 RTOS 提供的对外函数经过重定义、声明、注册, 形成对外函数接口表后, USER 程序还需进一步通过重映射机制形成对外函数映射表, 才能最后在 USER 中实现对函数的调用.

3.1 对外函数接口表的映射

当 RTOS 的对外函数接口表形成之后, 此时 RTOS 提供的函数已经变成了一段机器码, 必须先将 BIOS 的对外函数接口表映射成 USER 的对外函数映射表, 获得存放对外函数接口表的数组首地址, 这样 USER 程序才能使用它. 为了与 BIOS_API 数组的元素一一对应, USER 程序的对外函数映射表也采用数组 (如 USER_API), 用它来存放对外函数接口表的地址. 这样, 当 USER 使用 USER_API 时就相当于使用 BIOS_API, 也就是说, USER 通过 USER_API 就可以访问 BIOS 提供的对外函数.

在 USER 中, 采用 SVC 中断的方式来实现对外函数接口表的映射. 因此, 在 BIOS 中先要将 SVC 中断重定向为用户的 SVC_IRQ, 接着在 USER 中调用 svc1_init 函数触发 BIOS 中的 SVC_IRQ 中断, 然后由 SVC_IRQ 中断触发实际的 SVC 封装函数 SVC_HandlerS, 最后在这个汇编程序 SVC_HandlerS 中实现将 BIOS 提供的对外函数接口表的入口地址映射到对外函数映射表中, 汇编函数 SVC_HandlerS 实现流程如图 2 所示.

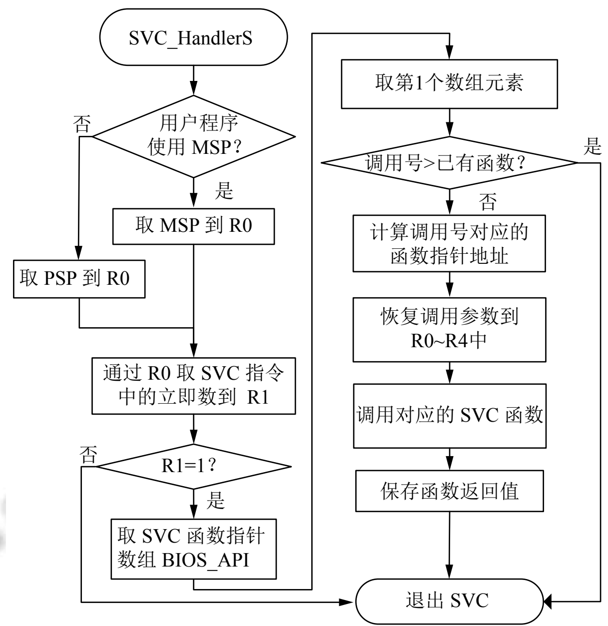

图 2 汇编函数 SVC_HandlerS 实现流程

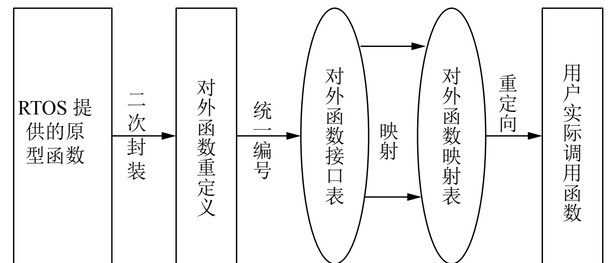

图 3 对外函数映射关系示意图

3.2 对外函数的重定向

当对外函数映射表形成之后, 此时 USER 就可以通过 USER_API 数组访问函数, 如 USER_API[1] 访问的是 1 号对外函数. 但采用 USER_API[i] 这种形式对具体要访问的对外函数的类型、函数名、参数以及功能等不够清晰明了. 因此, 类似中断向量重定向的做法, 也可以重定向对外函数, 重新给 USER_API[i] 取另外一个用户熟悉的函数名, 这个函数名可以与对外函数名同名, 也可以是不同名的, 这样就可以为用户提供函数原型级接口, 易于用户记住和使用, 图 3 描述了从 RTOS 的原型函数到最终实现用户实际调用函数的映射关系. 函数重定向宏定义的一般形式如下:

#define 函数重定向名 ((对外函数声明指针表达形式) (全局数组 [对外函数编号]))

例如:

```
#define delay ((void (*)(uint_32 millisec))(USER_API[4]))
```

3.3 对外函数的调用

当对外函数重定向之后, 就可以利用重定向后的名字来调用函数. 例如, 当 USER 调用 delay 函数时, 实际上是指向了 USER_API[4], 而 USER_API[4] 对应 BIOS_API[4], BIOS_API[4] 存放的是 thread_wait 函数的入口地址, 而 thread_wait 函数实际上就是 Thread::wait 函数. 因此, 可以认为 USER 通过调用 delay 函数达到调用 Thread::wait 函数的目的.

4 mbedOS 下的应用实践

2014 年 ARM 公司推出了 mbedOS, 它是一种专为

© 中国科学院软件研究所 http://www.c-s-a.org.cn

物联网 (IoT) 中的"物体"设计的开源嵌入式实时操作系统[17].本文选用 mbedOS 进行应用实践, 测试工程在 Kinetis Design Studio 3.0.0 IDE 集成开发环境和金葫芦 AHL-A 系列 Cortex-M0+内核的 KL36 微控制器[18] (即 AHL-AN100VL 型号开发板) 上进行. KL36 片内 Flash 大小为 64 KB, 一般用来存放中断向量、程序代码、常数等; 片内 RAM 为静态随机存储器 SRAM, 大小为 8 KB, 一般用来存储全局变量、静态变量、临时变量 (堆栈空间) 等.

4.1 mbedOS 提供的对外接口函数

使用 KL36 微控制器作为 mbedOS 驻留的硬件载体, 考虑其 Flash 和 RAM 空间大小的因素, 在 mbedOS 驻留于 BIOS 时, 对 Flash 空间采用分割独享方式划分. 对 RAM 空间可采用分割独享和重叠共享方式划分, 如图 4 所示, 在分割独享方式中 BIOS 和 USER 各占一半的 RAM 空间; 本文采用重叠共享方式, BIOS 占全部 RAM 空间, USER 占一半以上的 RAM 空间, 这样可以最大程度地提高 RAM 空间的利用率. 根据前面介绍的 RTOS 对外函数接口设计方法和重映射机制, 可以将 mbedOS 提供的启动、线程、延时、事件、消息队列、信号量和互斥量等函数进行重定义、声明、注册、映射、重定向, 最后提供给用户程序调用. mbedOS 提供的对外函数原型名称、对外函数二次封装名称以及重定向函数名称之间的映射关系如表 1所示, 仅列举了本测试工程中使用到的对外接口函数.

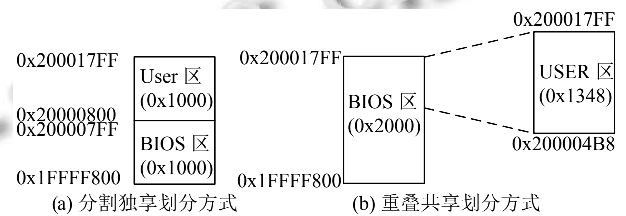

图 4 RAM 空间划分示意图

表 1 部分对外函数重定向一览表

mbedOS提供的对外函数原型名称	对外函数二次封装名称	对应对外函数接口表的位置	对应对外函数映射表的位置	重定向函数名称
void mbedOS_start(osThreadId_t &thd, void (*func)(void))	void OS_start(void (*func)(void))	BIOS_API[1]	USER_API[1]	OsStart
void Thread::constructor(osPriority priority, uint32_t stack_size, unsigned char *stack_mem, const char *name)	uint_8 thread_create(uint_32 priority, uint_32 stack_size)	BIOS_API[2]	USER_API[2]	create
osStatus Thread::start(Callback<void()> task)	void thread_start(uint_8 threadIndex, void (*func)(void))	BIOS_API[3]	USER_API[3]	start
osStatus Thread::wait(uint32_t millisec)	void thread_wait(uint_32 millisec)	BIOS_API[4]	USER_API[4]	delay
int32_t Thread::signal_set(int32_t flags)	void thread_signal_set(uint_8 flag, int_32 signals)	BIOS_API[5]	USER_API[5]	signal_set
osEvent Thread::signal_wait(int32_t signals, uint32_t millisec)	void thread_signal_wait(int_32 signals)	BIOS_API[6]	USER_API[6]	signal_wait

4.2 功能性测试

测试工程的功能是创建两个任务, 实现每 2 秒红灯闪烁一次, 蓝灯任务每 1 秒切换亮暗一次, 绿灯任务当收到蓝灯任务的信号 (17) 时, 切换绿灯亮暗. 当芯片上电之后, 首先启动 BIOS, 接着转到 USER 的启动, 在 USER 的启动过程中通过映射对外函数接口表形成对外函数映射表, 并重定向对外函数. 然后启动 mbedOS, 并创建和启动蓝灯任务和绿灯任务, 最后在一个无限循环中使红灯每 2 s 闪烁一次, 同时对蓝灯任务和绿灯任务进行调度. 蓝灯任务主要完成每秒闪烁一次, 并设置信号; 绿灯任务主要是等待信号, 当收到信号后闪烁一次. 测试工程的执行流程如图 5 所示.

在测试工程中主要调用的对外函数有: 操作系统启动函数 OsStart、任务创建函数 create、任务启动函数 start、延时函数 delay、信号设置函数 signal_set 和信号等待函数 signal_wait 等, 其功能性测试结果如图 6 所示, 从中可以看出能精准调用这些对外函数, 程序的功能得到准确的实现, 说明对外函数接口设计正确.

4.3 编译时间测试

编译时间的测试是在联想笔记本电脑 K49 上进行, 其 CPU 型号为 Intel Core i7-3520M, 主频 2.9 GHz, 内存 8 GB, 采用 64 位的 Windows 7 操作系统. 测试工程针对 KL36 (采用 Kinetis Design Studio 3.0.0 IDE 编译环境)、S32K144 (采用 S32 Design Studio for ARM

© 中国科学院软件研究所 http://www.c-s-a.org.cn

v1.3 编译环境) 和 MSP432 (采用 Code Composer Studio 6.2.0 编译环境) 微控制器分别对 mbedOS 是否驻留 BIOS 的 USER 程序进行编译时间测试, 测试结果如表 2 所示. 从表 2 中可以看出, 将 mbedOS 驻留在 BIOS 中, USER 程序可以节约 2~3 倍的编译时间, 从而提高了用户程序的开发效率.

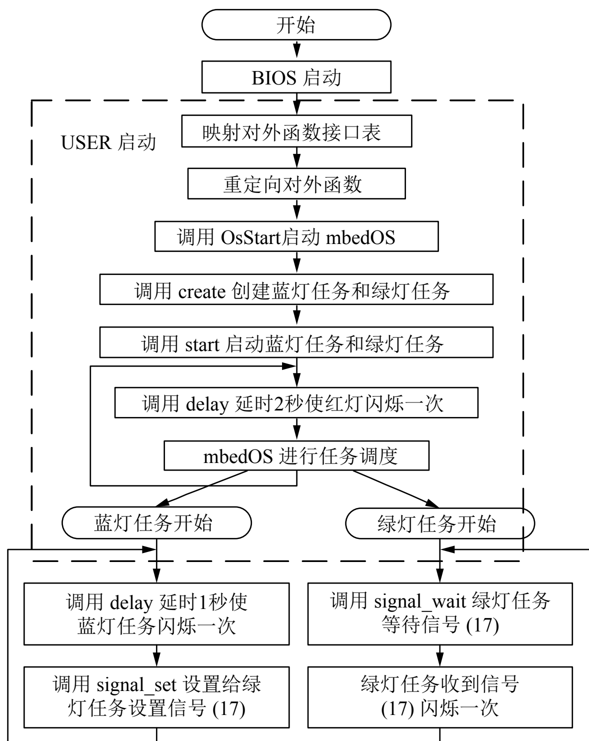

图 5 测试工程执行流程

```
***1-1.重定向SVC中断为SVC_IRQ***
***1-2.触发SVC_IRQ中断获得对外函数接口表的入口地址***
***1-3.调用OS_start函数启动操作系统***
  ###2-1.调用create函数创建蓝灯任务###
  ###2-2.调用start函数启动蓝灯任务###
  ###2-3.调用create函数创建绿灯任务###
  ###2-4.调      @@@4-1.绿灯任务调用signal_wait函数等待信号(17)@@@
用start函数启动绿灯任务###
    $$$3-1.调用delay函数使蓝灯任务延时1秒$$$
    $$$3-      @@@4-1.绿灯任务调用signal_wait函数等待信号(17)@@@
2.蓝灯任务调用signal_set函数设置信号(17)$$$
  ###2-5.调用delay函数使红灯延时2秒###
```

图 6 功能性测试结果

表 2 驻留与非驻留时 USER 程序编译时间

微控制器	测试次数	平均非驻留编译时间(s)	平均驻留编译时间(s)	平均节省时间(s)	平均节省时间占比(%)
KL36	200	89	19	70	368
S32K144	200	22	7	15	214
MSP432	200	50	12	38	317

5 结论与展望

为充分发挥实时操作系统的强大功能, 本文给出了 RTOS 在 BIOS 中的驻留方法, 在 GEC 架构下提出了 RTOS 对外接口函数的设计方法, 剖析了接口函数重映射机制, 为用户提供函数原型级的调用接口. 最后以 NXP 的 KL36 芯片为例, 在 mbedOS 进行应用实践, 测试表明对外函数接口的设计是正确的, 有效地解决了函数调用的困难, 提高用户程序的可靠性, 为 RTOS 的应用研究提供了基础. 后续还将进一步探索用户程序在不同 RTOS 上的可移植性问题, 本文涉及到的测试工程可到苏州大学嵌入式学习社区网站 (网址: http://sumcu.suda.edu.cn) 的“教学培训-教学资料-mbedOS”位置, 下载“SD_mbedOS_API”查看.

参考文献

1 王宜怀, 张建, 刘辉, 等. 窄带物联网 NB-IoT 应用开发共性技术. 北京: 电子工业出版社, 2019.

2 张美玉, 张倩颖, 孟子琪, 等. 实时嵌入式双操作系统架构研究综述. 电子学报, 2018, 46(11): 2787–2796. [doi: 10.3969/j.issn.0372-2112.2018.11.029]

3 李正, 吴敬征, 李明树. API 使用的关键问题研究. 软件学报, 2018, 29(6): 1716–1738. [doi: 10.13328/j.cnki.jos.005541]

4 Tsai CC, Jain B, Abdul NA, *et al*. A study of modern Linux API usage and compatibility: What to support when you’re supporting. Proceedings of Eleventh European Conference on Computer Systems. London, UK. 2016. 1–16.

5 Atlidakis V, Andrus J, Geambasu R, *et al*. POSIX abstractions in modern operating systems: The old, the new, and the missing. Proceedings of the Eleventh European Conference on Computer Systems. London, UK. 2016. 19.

6 Bavota G, Linares-Vásquez M, Bernal-Cárdenas CE, *et al*. The impact of API change- and fault-proneness on the user ratings of Android apps. IEEE Transactions on Software Engineering, 2015, 41(4): 384–407. [doi: 10.1109/TSE.2014.2367027]

7 Li L, Bissyande TF, Le Traon Y, *et al*. Accessing inaccessible android APIs: An empirical study. Proceedings of 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME). Raleigh, NC, USA. 2016. 411–422.

8 Nadi S, Krüger S, Mezini M, *et al*. Jumping through hoops: Why do Java developers struggle with cryptography APIs? Proceedings of the 38th International Conference on Software Engineering. Austin, TX, USA. 2016. 935–946.

9 汪昕, 陈驰, 赵逸凡, 等. 基于深度学习的 API 误用缺陷检测. 软件学报, 2019, 30(5): 1342–1358. [doi: 10.13328/j.cnki.jos.005722]

10 Liang B, Bian P, Zhang Y, *et al*. AntMiner: Mining more

© 中国科学院软件研究所 http://www.c-s-a.org.cn

bugs by reducing noise interference. Proceedings of the 38th International Conference on Software Engineering. Austin, TX, USA. 2016. 333–344.

11 Murali V, Chaudhuri S, Jermaine C. Bayesian specification learning for finding API usage errors. Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. Paderborn, Germany. 2017. 151−162.

12 Thung F. API recommendation system for software development. Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE). Singapore, Singapore. 2016. 896–899.

13 吕晨, 姜伟, 虎嵩林. 一种基于新型图模型的 API 推荐系统. 计算机学报, 2015, 38(11): 2172–2187.

14 Treude C, Robillard MP. Augmenting API documentation with insights from stack overflow. Proceedings of the 2016 IEEE/ACM 38th International Conference on Software Engineering. Austin, TX, USA. 2016. 392–403.

15 Zhou J, Walker RJ. API deprecation: A retrospective analysis and detection method for code examples on the Web. Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. Seattle, WA, USA. 2016. 266–277.

16 夏艳敏, 唐兵, 唐明董, 等. 利用关联规则挖掘的 Web API 组合模式发现. 小型微型计算机系统, 2019, 40(10): 2195–2201. [doi: 10.3969/j.issn.1000-1220.2019.10.031]

17 Bock C, Marquardt M, Martens A, *et al*. Smart sensors and actors with BACnetTM and mbed OS on cortex-M microcontrollers. Proceedings of the 2019 IEEE 5th World Forum on Internet of Things (WF-IoT). Limerick, Ireland. 2019. 937–942.

18 NXP. KL36 sub-family reference manual. https://www.nxp.com/docs/en/reference-manual/KL36P121M48SF4RM.pdf. (2013-07-03)[2019-09-25].

www.c-s-a.org.cn

© 中国科学院软件研究所 http://www.c-s-a.org.cn