文章编号:1005-1228(2011)04-0054-04

嵌入式实时操作系统 μ C/OS 在 C51 平台上的移植及应用

娄小平[1,2]

(1.湖南文理学院计算机科学与技术学院,湖南常德 415000;2.中南大学信息科学与工程学院,湖南长沙 410000)

摘 要:嵌入式操作系统以其简洁、高效等优点扮演了越来越重要的角色。选择了源代码公开、体积小且可裁剪移植性好的 μ C/OS-II,选择 C51 微处理器为应用平台,根据实际项目要求把 μ C/OS-II 移植到 C51 单片机上。移植的过程主要集中在三个文件的重新编写上:一个头文件 OS_CPU.H、一个 C 代码文件 OS_CPU_C.C 和一个汇编文件 OS_CPU_A.ASM 文件。还针对 C51 的小内存所做的优化,对可重入问题的进行了分析与解决,基本达到应用的需要。

关键词:嵌入式系统;μ C/OS;C51 平台;移植

中图分类号:TP316.2 文献标识码:A

The Transplantation and Application on μ C/OS-II Based on C51 Microcontroller

LOU Xiao-ping[1,2]

(1. School of Computer Science and Technology ,Hunan University of Arts and Science, Changde 415000,China;
2. School of Information Science and Engineering ,Center South University , Changsha 410000, China)

Abstract:Embedded operating system play an increasingly important role because of its simple, efficient. We choose theμ C / OS-II whose source code is public, small and portable can be cut, and select the C51 microprocessor application platform.We migrate μ C / OS-II to the C51 microcontrollers. based on actual project requirements. Migration process is mainly concentrated in three re-write the file, a header file OS_CPU.H, a C code file and an assembly file OS_CPU_A.ASM OS_CPU_C.C file. Also made for the C51, a small memory optimization, the re-entry problems were analyzed and resolved for applications.

Key words: embedded system;μ C/OS;C51 microcontrollers;migration

作为嵌入式系统(包括硬、软件系统)极为重要的组成部分的嵌入式操作系统,通常包括与硬件相关的底层驱动软件、系统内核、设备驱动接口、通信协议、图形界面、标准化浏览器等。嵌入式操作系统具有通用操作系统的基本特点,如:能够有效管理越来越复杂的系统资源;能够把硬件虚拟化,使得开发人员从繁忙的驱动程序移植和维护中解脱出来;能够提供库函数、驱动程序、工具集以及应用程序。与通用操作系统相比较,嵌入式操作系统在系统实时高效性、硬件的相关依赖性、软件固态化以及应用的专用性等方面具有较为突出的特点[1]:

(1)体积小。嵌入式系统有别于一般的计算机处理系统,它不具备像硬盘那样大容量的存储介质,而大多使用闪存(Flash Memory)作为存储介质,这就要求嵌入式操作系统只能运行在有限的内存中,不能使用虚拟内存,中断的使用也受到限制。因此,嵌入式操作系统必须结构紧凑,体积微小;

(2)实时性。大多数嵌入式系统都是实时系统,而且多是强实时多任务系统,要求相应的嵌入式操作系统也必须是实时操作系统(RTOS)。实时操作系统作为操作系统的一个重要分支已成为研究的一个热点,主要探讨实时多任务调度算法和可调度性、死锁解除等问题;

(3)特殊的开发调试环境。提供完整的集成开发环境是每一个嵌入式系统开发人员所期待的。一个完整的嵌入式系统的集成开发环境一般需要提供的工具是编译/连接器、内核调试/跟踪器和集成图形界面开发平台。其中的集成图形界面开发平台包括编辑器、调试器、软件仿真器和监视器等。

1 μ C/OS-II 软件体系结构

嵌入式实时操作系统 μ C/OS-II 内核采用模块化

收稿日期:2011-03-22 修回日期:2011-07-05
基金项目:湖南文理学院校级教学改革研究立项项目(项目编号:JGZC0906)
作者简介:娄小平(1982-),女,在读博士,讲师,研究方向:量子保密通讯。

结构,使得 μ C/OS- II 结构清晰,源代码可读性好,给操作系统的移植也带来方便。

μ C/OS- II 的结构以及它与硬件的关系如图 1[2]所示,其文件结构可以分为三个部分:移植 μ C/OS- II (与处理器类型有关的代码)、μ C/OS- II (与处理器类型无关的代码) 和 μ C/OS- II 配置文件 (与应用程序有关)。

从图 1 可以看出,在应用程序中使用 μ C/OS- II 时需要用户提供的是应用软件和 μ C/OS- II 的配置部分。与处理器相关的代码在 OS_CPU.H (包括用 #define 设置一些常量的值,声明的数据类型和用 #define 声明的宏),OS_CPU_C.C (用 C 语言编写的简单函数) 和 OS_CPU_A.ASM (编写的汇编语言函数) 三个文件中。

图1　μ C/OS- II 硬件 / 软件结构图

2　μ C/OS- II 在 C51 系统上的移植

所谓移植,就是使一个实时内核能在某个微处理器或微控制器上运行[3]。μ C/OS- II 支持几乎所有的 8、16 位、32 位微处理器,但并非所有机构的微处理器都能进行 μ C/OS- II 的移植,要使 μ C/OS- II 正常运行,处理器必须满足以下要求:

⑴ 处理器的 C 编译器能产生可重入代码;

⑵ 用 C 语言就可以打开和关闭中断;

⑶ 处理器支持中断,并且能产生定时中断;

⑷ 处理器支持能够容纳一定量数据(可能是几千字节)的硬件堆栈;

⑸ 处理器有将堆栈指针和其它 CPU 寄存器读出和存储到堆栈或内存中的指令。

为了方便移植,大部分的 μ C/OS- II 代码是用 C 语言写的;但依赖于具体硬件结构的操作必须要用汇编语言来完成,如寄存器存放,硬件堆栈操作等。操作系统的移植非常依赖于微处理器具体的机构和所用的编译器,本文以 C51 结构微处理器为例,编译器使用 Keil μ Version2,对 μ C/OS- II 进行移植。

对 μ C/OS- II 进行移植到不同处理器平台时,需要解决的主要问题有:

⑴ 数据类型的重定义。对于一个操作系统来说,基于其上进行开发的应用系统一般都使用高级语言,高级语言都有自己的数据类型。但对于不同的处理器由于字长不相同,造成同一数据类型在不同处理器中会有不同的解释,所以对不同的处理器应该重新进行数据类型的定义;

⑵ 堆栈结构的设计。当同一个操作系统应用于不同处理器或同一处理器的不同应用系统时,由于各应用系统所追求的性能各有特点,就会要求与性能有很大关系的堆栈结构尽可能与本系统所追求的性能一致;

⑶ 任务切换时的状态保存与恢复。这是多任务操作系统最主要的工作,也是最频繁的工作。所以任务切换在实现时的正确与否是操作系统运行时的基本保证,同时它的简洁与否决定操作系统的效率。

在 μ C/OS- II 移植过程中涉及以上问题的代码都包含在文件 OS_CPU.H、OS_CPU_C.C、OS_CPU_A.ASM 中。因此移植的主要工作也在源代码的基础上围绕着这三个文件的改写展开。

2.1　OS_CPU.H 文件的修改

OS_CPU.H 文件定义了与 CPU 相关的基本信息,包括了用 #define 语句定义的、与处理器相关的常数、宏以及类型。

⑴ 定义编译器相关的数据类型

因为不同的微处理器有不同的字长,所以 μ C/OS- II 的移植包括了一系列的数据类型定义,以确保其可移植性。尤其是 μ C/OS- II 代码从不使用 C 语言中的 short,int 及 long 等数据类型,因为它们是与编译器相关的,是不可移植的。相反,定义的整型数据结构等既是可移植的,又很直观。这里数据类型的 BOOLEAN 要定义为 unsigned char 类,而不能是 bit 类型,因为 bit 类型不能用在数组和结构体中[4];

⑵ 定义进入与退出临界区宏

与所有的实时内核一样,μ C/OS- II 需要先禁止中断再访问代码的临界段,并且在访问完毕后重新允许中断。这就使得 μ C/OS- II 能够保护临界段代码免受多任务或中断服务例程(ISR)的破坏。中断禁止时间

是商业实时内核公司提供的重要指标之一，因为它将影响到用户的系统对实时事件的响应能力。关中断函数 OS_ENTER_CRITICAL () 和开中断函数 OS_EXIT_CRITICAL ( )总是成对出现的，如程序清单所示。最简单的方法是分别设置为 EA=0 和 EA=1，即在 OS_ENTER_CRITICAL( )中调用处理器指令来禁止中断，以及在 OS_EXIT_CRITICAL( )中调用允许中断指令，不少移植实例都是这么处理的。这样做存在着小小的问题，如果用户在禁止中断的情况下调用了 μ C/OS- Ⅱ函数，在从 μ C/OS- Ⅱ返回的时候，中断可能会变成是允许的了。为避免这种现象的发生，我们引入关中断计数器 Os_Enter_Sum，保证了调用这类 μ C/OS- Ⅱ函数前后，中断的开关状态不会改变[5]。

程序清单：

```
//μ C/OS- II 函数访问临界资源的一般形式
    {
      OS_ENTER_CRITICAL( );
      /* μ C/OS- II 临界代码段 */
      OS_EXIT_CRITICAL( );
    }
```

设置堆栈增长方向 OS_STK_GROWTH

绝大多数微处理器和微控制器的堆栈是从上往下递减的，但是也有某些处理器使用的是相反的方式，μ C/OS- Ⅱ被设计成对 2 种情况都可以处理，只要再用配置常数 OS_STK_GROWTH 指定堆栈的方向就可以了：

置 OS_STK_GROWTH 为 0 表示堆栈从下（低地址）往上（高地址）递增；

置 OS_STK_GROWTH 为 1 表示堆栈从上（高地址）往下（低地址）递减。

OS_STK_GROWTH 设置为 0，因为 C51 堆栈是从下往上增长（1 是向下增长，0 是向上增长）。

定义任务切换函数 OS_TASK_SW( )为 OSCtxSw( )

OS_TASK_SW( )是一个宏，它是在 μ C/OS- Ⅱ从低优先级任务切换到最高优先级任务时被调用的，OS_TASK_SW( )总是在任务级代码中被调用的。因为 C51 没有软中断指令，所以用程序调用代替。子程序和中断两者的堆栈格式相同，RETI 指令复位中断系统，RET 则没有。实践表明对于 C51 用子程序调用入栈用中断返回指令 RETI 出栈是没有问题的，反之中断入栈 RET 出栈则不行。总之对于入栈，程序调用与中断调用效果是一样的，可以混用。在没有中断发生的情况下复位中断系统也不会影响系统正常运行。

2.2 OS_CPU_C.C 文件修改

此文件完成任务的初始化，是用 C 语言编写的。μ C/OS- Ⅱ是采用软件中断来进行任务切换的，所以要初始化任务的栈结构，使任务的堆栈结构和中断的堆栈结构一样，所有的寄存器都保留在堆栈中。

μ C/OS- Ⅱ的移植实例要求用户编写六个简单的 C 函数：

初始化任务堆栈函数 OSTaskStkInit()
任务创建钩挂函数 OSTaskCreateHook()
任务删除钩挂函数 OSTaskDelHook()
任务切换钩挂函数 OSTaskSwHook()
统计任务钩挂函数 OSTaskStatHook()
定时钩挂函数 OSTimeTickHook()

唯一必要的函数是 OSTaskStkInit( )，其它五个函数必须得声明但没必要包含代码，它们是由系统函数调用的钩子函数。OSTaskStkInit( )函数由任务创建函数 OSTaskCreate( )调用，功能是初始化任务堆栈。由于各种处理器的寄存器对堆栈的操作方式不尽相同，因此该函数需要用户在进行 μ C/OS- II 移植时，按所使用的处理器由用户来编写。为了编写该函数，必须理解 T89C51AC2 的堆栈，及怎样设置软件堆栈。堆的大小可根据任务的实际情况自行确定。5 个系统钩子函数可直接定义它们为空函数。

所设计的任务堆栈的初始结构如图 2 所示：

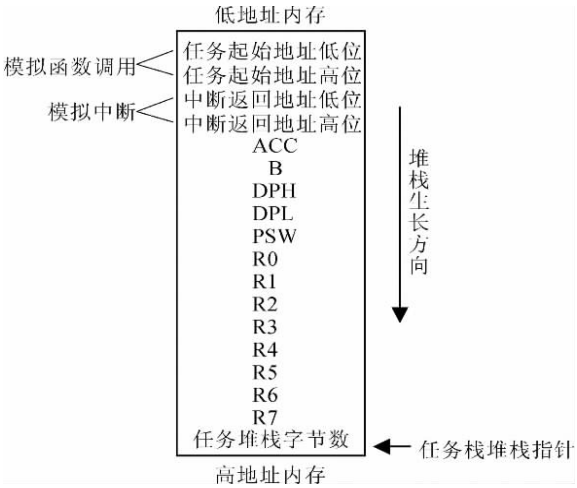

图 2 堆栈结构图

堆栈结构说明：

(1) Keil C51 编译器支持在 C 语言中写中断处理程序，当进入由 C 语言写的中断处理程序时，在 CPU 将中断返回地址压入堆栈后，紧接着，Keil C51 编译器自动将 ACC、B、DPH、DPL、PSW、R0、R1、R2 、R3 、R4 、R5 、R6、R7 按照先后顺序压入堆栈。在退出中断处理程序之前，应调用 OSIntExit( )以完成可能会发生的任务切换 .而此时若发生任务切换，则在 OSIntExit( )中将

调用汇编函数 OSIntCtxSw( ),在这里,将直接返回到程序断点处,从而屏蔽掉由 Keil C51 编译器生成的退出中断前的一系列出栈指令。由于所设计的堆栈结构与 Keil C51 编译器压栈的顺序是一致的,因此,仍可以在 C 语言中写中断处理程序;

(2)"任务堆栈字节数"保存了从"任务起始地址低位"一直到"R7"的任务堆栈中的字节数目。由于 51 系列单片机内部 RAM 小(仅 128- 8 个字节),通过中断切换任务要保存的内容多:寄存器 (R0～R7,A,B,PSW,DPTR 和 PC),全局变量和局部变量。如果不把所有空闲 RAM 分配给当前任务做堆栈,则堆栈肯定会溢出,故只好以空间换时间。定义了一个数组保存所有任务的堆栈的项端和底端,在任务切换时,把不是当前任务的堆栈空间上移或下移,直至所有空间都提供给当前任务使用。

2.3　OS_CPU_A.ASM 文件修改

移植工作的难点是在 OS_CPU_A.ASM 文件上,这里用户需要编写 4 个汇编语言函数 :OSStartHighRdy( )、OSCtxSw( )、OSIntCtxSw( ) 和 OSTickISR ( )。其中,OSTickISR( )用 C 语言写,即用定时器中断处理了程序来代替它。因此,只需要写 3 个汇编语言函数。

对这几个函数的关键代码段作详尽的分析。

```
NAME OS_CPU_A. ASM
?STACK SEGMENT IDATA
        RSEG ?STACK
OSStack:
        DS 60H
OSStkStart  IDATA  OSStack
```

在程序的首部,定义了所用到的一些外部变量以及系统堆栈区。当程序进入 main( )函数时,堆栈指针 SP 指向(#OSStkStart- 1)。

另外,由于硬件堆栈区被定义为 96byte(60H),因此,每个任务的任务栈也不能超过此限制。

4　可重入问题的分析与解决

C51 用户手册有以下说明[6] :许多标准库函数具有可重入性,但用户手册中特别说明的程序外,标准库函数都没有重入性。其中包括部分数学运算函数、大部分的字符串处理函数、流处理函数以及全部的内存分配函数。这要求使用者在多任务开发中必须明确其使用的函数是否为可重入的,增加了使用者的工作量。若多任务中使用了非重入函数,则必须将该函数改写为重入函数[7]

可重入函数可以被一个以上的任务调用,而不必担心数据被破坏。可重入函数任何时候都可以被中断,一段时间后又可以继续运行,而相应的数据不会丢失。由于 μ C/OS- II 是抢占式的实时多任务内核,同一个函数可能会被不同的任务调用,也可能会被中断,因此,移植 μ C/OS- II 要求 C 语言编译器可以产生可重入函数。但是正常情况下 Keil C51 编译器中的函数不能重入。原因是由于 8051 系列微控制器的硬件堆栈很小,硬件堆栈指针 SP 最多只能在内部 256 字节的 RAM 内移动,不能够指向 64K 的外部 RAM 空间。Keil C51 编译器不是把局部变量分配到堆栈中,而是把局部分量分配到内存固定地址 (当然首先尽可能使用寄存器) 。这样,局部变量实际上就是全局变量了。即使在函数一开始就开始关闭中断,也不能保证其重入性,因为在关中断函数中已经执行一些指令,把参数保存到所分配的内存固定地址了,这些指令使得函数不可重入。当编译器把所有局部变量分配到寄存器时,函数是可重入的。但当一个函数调用一个外部函数时,编译器认为所有寄存器变量已经发生变化;当一个变量要存储信息,直到调用这个函数后再使用时,这个变量就不可能分配到寄存器中,这个函数也就不可重入了。为了在 Keil C51 中实现可重入函数,可以使用 "reentrant" 关键字声明该函数是可重入的。编译器可根据编译模式为可重入函数在内部 RAM 或外部 RAM 空间开辟一个模拟堆栈来存储可重入函数的参数和局部变量。可重入函数的返回地址仍然保存在硬件堆栈中。Cx51 编译手册不推荐使用模拟堆栈,原因是受 8051 寻址方式的限制,模拟堆栈访问的效率很低。

参考文献:

[1] 李伯成. 单片机及嵌入式系统[M]. 北京 :清华大学出版社,2005 [4].

[2] JEAN J.LABROSSE 著,邵贝贝译. μ C/OS- II- 源码公开的实时嵌入式操作系统[M].北京:中国电力出版社,2001.

[3] 江平新,容太平. μ C/OS- II 在 cso51F 上的移植.单片机与嵌入式系统应用,2003(9):74- 75.

[4] 杨屹. 在 51 单片机上固化 uCOS51 的说明[EB/OL]. http://www.hjhj.com.

[5] 陈明计,周立功.嵌入式实时操作系统 Small RTOS51 原理及应用[M]. 北京 北京航空航天大学出版社,2004.

[6] Keil Elektronik GmbH.and Keil Software,Inc. Cx51 Compiler Optimizing C Compiler and Library Reference for Classic and Extended 8051 Microcontrollers User's Guide[M].keil 2001.

[7] 黄亮亮,朱欣华.基于实时多任务操作系统 μ C/OS- II 的 C8051F 系列单片机应用系统开发[J]. 测控技术,2005,24(9):39- 42.