μC/OS–II 在 51 单片机上的移植

吕小纳，徐力平
（郑州大学 信息工程学院，河南 郑州 450001）

摘要：针对在 51 单片机上移植实时操作系统 μC/OS–II 的目的，以 μC/OS–II 工作原理为基础，结合 51 单片机堆栈空间少的情况，采用改变堆栈指针到不同任务寄存器组的方法，通过改变堆栈指针的实验，得出在堆栈空间较少的情况下，也能够实现 μC/OS–II 在 51 单片机上的运行的结论。

关键词：μC/OS；单片机；实时操作系统；堆栈
中图分类号：TP31 文献标识码：A 文章编号：1674–6236(2012)06–0052–03

Porting RTOS μC/OS-II to MCS-51

LV Xiao-na, XU Li-ping
(*School of Information and Engineering, Zhengzhou University, Zhengzhou* 450001, *China*)

Abstract: To transplant the real-time operating system μC/OS-II in MCS-51, based on the μC/OS-II working principle and the MCS-51 stack space, we change the stack pointer to a different task registers to change the stack pointer, and get the μC/OS-II can run in MCS-51 even if with few stack space.

Key words: μC/OS; MCU; real-time operating system; stack

μC/OS－II 是一种公开源代码、结构小巧、具有可剥夺实时内核的嵌入式开发系统，代码简短、条理清晰、实时性及安全性能很高，绝大部分代码用 C 编写，现已被移植到多种处理器的构架中。随着 51 单片机片内资源的日益丰富，在 51 单片机上移植 μC/OS－II 已成为可能，植入系统后，由系统来管理软件与硬件资源，简化应用程序的设计，并且使应用系统功能更加完善。因此在 51 单片机上移植 μC/OS－II 具有十分重要的意义。

1 μC/OS 实时操作系统概述

μC/OS–II 实时操作系统是一种可移植、可固化、可裁剪即可剥夺型的多任务实时内核，适用于各种微处理器和微控制器。μC/OS–II 主要包括任务调度、时间管理、内存管理、事件管理（信号量、邮箱、消息队列）4 大部分。它的移植与 4 个文件相关：汇编文件（OS_CPU_A.ASM）、处理器相关 C 文件（OS_CPU.H、OS_CPU_C.C）和配置文件（OS_CFG.H）。有 64 个优先级，系统占用 8 个，用户可创建 56 任务，不支持时间片轮转。

它的基本思路就是“近似地每时每刻总是让优先级最高的就绪任务处于运行状态”。为了保证这一点，它在调用系统函数、中断结束、定时中断结束时总是执行调度算法。原作者通过事先计算好数据，简化了运算量，通过精心设计就绪表结构，使得延时可预知。任务的切换是通过模拟一次中断实现的。

2 任务调度的实现原理

任务调度是 μC/OS－II 的重要部分，和具体的微处理器关系紧密。必须移植的 5 个函数有 4 个都和任务有关。任务调度就是保存当前任务的寄存器和 PC 指针（即当前任务的断点），然后把将要执行的任务的寄存器值返回给寄存器并把 PC 指向将要执行任务的断点。这些的实现要借助于堆栈和中断，为了简便起见，先看函数调用时堆栈的使用情况。在函数调用时，堆栈的一个重要功能就是保存被调函数的断点地址。若有 4 个函数，Fun1 调用 Fun2，Fun2 调用 Fun3，Fun3 调用 Fun4，Fun4 为叶子程序（无子程序调用）。

```
void Fun1(void)
{
……
Fun2(); 1–1
_nop_();1–2
……
}
void Fun3(void)
{
……
Fun4(); 3–1
_nop_();3–2
……
}
```

```
void Fun2(void)
{
……
Fun3(); 2–1
_nop_();2–2
……
}
void Fun4(void)
{
……
_nop_();4–1
……
}
```

收稿日期：2012–02–15 稿件编号：201202067
作者简介：吕小纳（1984—），女，河南邓州人，硕士研究生。研究方向：信息与信号处理。

假设现在从 Fun1 一直运行到 Fun4，此时堆栈结构如图 1 所示，中间的 ADD_A 到 ADD_D 为堆栈中的数据，左边的 SP 到 SP-7 为堆栈指针，右边的 Fun1 到 Fun4 为对应的调用函数。运行 Fun4 时，此时 SP 与 SP-1 所存的值为 ADD_D，而 ADD_D 为 Fun3 中子函数 Fun4 的下一行的地址，即 Fun3 中 3-2 行的地址，以此类推，ADD_C 为 2-2 行地址，ADD_B 图 1 函数运行及堆栈结构图为 1-2 行地址。

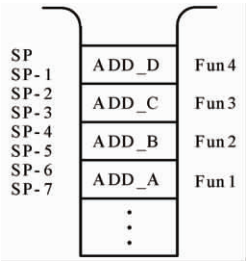

图 1　函数运行及堆栈结构图
Fig. 1　Function run and stack chart

当函数 A 调用函数 B 时，进入函数 B 时就会把函数 A 的断点地址压栈，而当函数 B 运行结束时则把堆栈中函数 A 的断点地址弹出到 PC 指针，程序接着从函数 A 的断点开始运行。如果在函数 B 中更改 SP 及 SP-1 中的数据，则函数 B 运行结束时就不会再返回函数 A 中，而返回到 SP 及 SP-1 更改后的数据所代表的地址。

以上是函数调用时的基本情况，如果是中断则堆栈不仅保存断点地址还会自动保存寄存器的值。任务调度就是靠中断来实现，中断中所保存的断点地址就是任务的断点地址，当本任务要再次执行时就把断点地址赋给 PC 就可以接着任务被中断时地址顺序执行。

3　头文件移植

与移植相关的 4 个文件中有 2 个头文件，这 2 个头文件的移植比较简单，可以参考其它的移植程序。其中 OS_CPU.H 中主要是数据类型的定义、堆栈生长方向的定义、开关中断的定义以及函数级任务切换的宏定义。OS_CFG.H 中主要是任务数、优先级数、事件数、每秒中断节拍数以及各种系统函数的使能定义。

4　汇编与 C 文件的移植

在要移植的汇编与 C 的两个文件中有 14 个函数，其中 9 个是接口函数，可根据实际需要来决定，有 5 个是必须写的。这 5 个函数分别是：OS_CPU_C.C 文件中的 OSTaskStkInit() 和 OS_CPU_A .ASM 文件中的 OSStartHighRdy()、OSCtxSw()、OSIntCtxSw()与 OSTickISR()。下面就这 5 个函数来做具体分析。

4.1　任务堆栈初始化函数 OSTaskStkInit()

此函数是在任务创建函数 OSTaskCreat () 或 OSTaskCreatExt()中调用的。因为系统为每个任务申请了一个数组作为栈，当一个任务运行时，就把堆栈指针指向本任务的栈，任务堆栈初始化函数就是在任务创建时将要创建任务的堆栈进行初始化。但 C51 的堆栈指针 SP 是 8 位的，只能在片内 RAM 的 256 个字节内寻址。因其寻址空间有限且 SP 唯一，不能像 DSP 或 ARM 那样为每一段程序或每一种模式定义堆栈，需小心管理堆栈空间。为了适应上述情况，需要换一种思路，不是让 SP 去指向各任务堆栈空间，而是把各任务堆栈空间的内容复制到系统栈中。至于堆栈数组空间要有多大以及堆栈数组空间里放些什么内容，可以借鉴 keil 中中断函数的压栈情况，当中断函数不指定寄存器组时，编译器一般将 PC、ACC、B、DPTR、PSW、R0~R7 寄存器入栈，其中 PC 和 DPTR 是双字节的，其它都是单字节的，一共 15 个字节，所以把堆栈数组设计成至少 15 个字节的，以保证任务所用的寄存器都在堆栈数组中包含着。因为每个数组里放的是寄存器的值，在此就把这每个任务的堆栈数组叫做寄存器数组，暂且把寄存器数组设计成 15 个字节，依次存放 PC、ACC、B、DPTR、PSW、R0~R7。

函数 OSTaskStkInit()传递 4 个参数，第 1 个参数 task 是所创建任务的起始地址，这个参数须保存到 PC 在寄存器数组的对应位置，第 2 个参数 ppdata 是所创建任务的参数，C51 规则中用 R1~R3 来传递参数指针，这个参数须存放到 R1~R3 在寄存器数组中的对应位置。第 3 个参数 ptos 是栈底指针，从当前地址开始初始化堆栈指针，第 4 个参数 opt 是附加参数，一般不用。

4.2　运行等待任务中优先级最高任务函数 OSStartHighRdy()

此函数在启动操作系统函数 OSStart()的最后一行调用，且此函数不返回，经过此函数后 μC/OS 接管系统。OSStartHighRdy()不是去调用用户任务函数，而是让 PC 指针指向任务函数首地址。且任务函数的传递参数只有一个，若此参数正确，则可保证任务函数运行正确。在调用 OSStartHighRdy()之前 OSStart()已经把最高优先级任务的任务表准备好了，只要把最高优先级任务表的数据恢复到堆栈中，再执行返回指令即可，以上最关键的是如何让其返回到最高优先级任务中而不是返回到被调函数中。

当函数 OSStart()调用函数 OSStartHighRdy()时，断点地址入栈；当 OSStartHighRdy () 执行完之后，返回断点。在 OSStartHighRdy()中把 SP 及 SP-1 的值改为最高优先级任务的地址，这样 OSStartHighRdy()就会返回到最高优先级任务中去运行。

4.3　任务级的任务切换函数 OSCtxSw()

此函数是保存当前任务的状态，然后运行处于就绪态中的最高优先级任务。前面介绍过不是更改 SP 去指向寄存器数组，而是把寄存器数组的数复制到堆栈中。先看下一般的情况，在用户任务 MyTask(void* ppdtat)中调用 TimeDly()，TimeDly()中调用 OSSched()，在 OSSched()中有一个宏 OS_TASK_SW()，这个宏的目的是让程序进入函数 OSCtxSw()。参看图 1，就如 Fun4 为 OSCtxSw()，Fun3 为 OSSched()，Fun2 为 TimeDly ()，Fun1 为 MyTask ()。ADD_D 存的是

OSSched()的断点,ADD_C 为 TimeDly()的断点,ADD_B 为 MyTask()的断点。如果进行任务切换,应该把高优先级任务的地址值赋给 ADD_B(即 SP-4 与 SP-5)。

以上考虑的是最简单的情况,当任务比较复杂时,可能更改了 ACC、PSW、DPTR 或 R0~R7 的值,在进入高优先任务时,寄存器并不是此任务的寄存器值,运行的结果可能不正确。

在上述情况下如何保证 CPU 寄存器的值正确,要分两个阶段。第一个阶段是把 CPU 寄存器值保存到要挂起任务的寄存器数组中,当刚进入 OSCtxSw()时,CPU 寄存器的值是要挂起任务的寄存器值,所以一开始就要锁定 CPU 寄存器的值。如果 OS_TASK_SW()定义为中断的话,在进入 OSCtxSw()时,CPU 寄存器的值被自动压栈;如果把 OS_TASK_SW()定义为函数时,在进入函数时使用内嵌汇编的方法把 CPU 寄存器入栈。这时堆栈中又压入了 13 个字节,就如在图 1 的 ADD_D 上又压入了 13 个字节的数据,然后从堆栈中把值取出来放到相应任务的寄存器数组中。第二个阶段是把将要执行任务的寄存器数组的值复制到堆栈中。此时 PC 指针在堆栈中对应的位置是 SP-17 与 SP-18,SP 到 SP-12 的 13 个字节对应 ACC、B、DPTR、PSW、R0~R7。

4.4 中断级的任务切换函数 OSIntCtxSw()

此函数和上一个函数基本思想一致,都要保存当前任务的状态,运行处于就绪态中的优先级最高的任务。二者的不同在于,上个函数的堆栈中 SP-17 与 SP-18 是 PC 值的位置,SP 到 SP-12 是 13 个寄存器的位置。当中断来时,在中断中调用函数 OSIntExit(),函数 OSIntExit()调用函数 OSIntCtxSw(),在 OSIntCtxSw()中实现任务切换。在进入函数 OSIntExit()之前寄存器的值已经入栈,所以运行到本函数时堆栈中 SP-17 与 SP-18 是 PC 值的位置,SP-4 到 SP-16 是 13 个寄存器的位置。在图 1 上,上个函数的 13 个寄存器的值被压入 ADD_D 上面的 13 个字节中,而本函数是在 ADD_B 于 ADD_C 之间压入的这 13 个寄存器。

4.5 周期节拍中断函数 OSTickISR()

这个函数是给系统提供一个节拍,一般每秒 10~100 次。如果节拍频率太高,μC/OS 系统会占用大量硬件资源;如果太低,任务间的切换又会很慢。

此函数首先要保证产生一个周期性的中断,可以使用硬件定时器,也可以从交流电中获得 50/60Hz 的时钟频率。这个函数至少要做 3 件事:1)进入中断时,把中断嵌套层数计数器加 1,说明又进入一次中断,也可以直接调用 OSIntEnter()函数;2)调用时钟节拍函数 OSTimeTick(),告知系统又经过了一个节拍;3)调用 OSIntExit()函数,说明要退出中断了,此函数会自动处理。

5 结束语

文中阐述了在堆栈空间有限的 51 单片机上运行 μC/OS-II 系统的移植过程,利用系统栈 SP 作为数据交换的枢纽。在实际应用中,如果用系统栈来移植,只需根据文中的基本思想进行适当的改写,即可运行于其他处理器上。如果处理器的堆栈指针寻址空间足够大,也可以为每个任务开辟一个栈,通过改变堆栈指针指向不同任务的栈空间,来实现任务调度。

通过在 51 单片机上的运行,可以看出 μC/OS-II 也能在堆栈空间比较少的 CPU 上运行。

参考文献:

[1] Labrosse J J. MicroC/OS-II The Real-Time Kernel Second Edition[M]. US:CMP Media LIC, 2002.

[2] Labrosse J J. 嵌入式实时操作系统μC/OS-II [M]. 邵贝贝,译. 北京:北京航天航空大学出版社,2003.

[3] 马忠梅. 单片机的C语言应用程序设计[M]. 北京:北京航天航空大学出版社,2003.

[4] 任哲. 嵌入式实时操作系统μC/OS-II原理及应用[M]. 北京:北京航天航空大学出版社,2003.

[5] 陈是知. μC/OS-II内核分析、移植与驱动程序开发[M]. 北京:人民邮电出版社,2007.

[6] 杨宗德,张兵. μC/OS-II标准教程[M]. 北京:人民邮电出版社,2009.

[7] 胡大可,李培弘,方路平. 基于单片机8051的嵌入式开发指南[M]. 北京:电子工业出版社,2003.

(上接第 51 页)

teaching management system of design[J]. Technology Wind, 2008(4): 125.

[3] 张宏森,朱征宇. 四层B/S结构及解决方案[J]. 计算机应用研究,2002(9): 21-22.
ZHANG Hong-sen,ZHU Zheng-yu. The four layers of B/S structure and solutions[J]. Computer Application and Research, 2002(9): 21-22.

[4] 闫振福, 张晓诺. 基于J2EE平台的教学管理系统设计实现[J]. 中国教育信息化,2008(9): 52-53.
YAN Zhen-fu,ZHANG Xiao-nuo. Based on the J2EE platform teaching management system design and implementation[J]. China Education Informatization,2008(9): 52-53.

[5] 杨鹏. 基于 J2EE 和工作流技术架构的教务管理系统的设计与实现[D]. 湖南:湖南师范大学,2003.

[6] 倪晓秋. J2EE 开发案例[M]. 北京:中国水利水电出版社,2005.