uC/OS-II 内核在 80C51 系列单片机上的移植

赵建华， 汪文勇
（电子科技大学 计算机科学与工程学院，四川 成都 610054）

摘 要：介绍了一种实时操作系统——uC/OS-II 的内核结构，分析了它的移植技术；讨论了将其移植到 80C51 系列单片机硬件平台上的关键技术，并详细描述了此移植的实现过程，分析了移植测试实验情况；针对 keil Cx51 编译器环境下的一些特殊情况进行了说明，并总结了移植的一般方法。

关键词：8051 单片机；实时操作系统；移植；交叉编译器；测试
中图法分类号：TP301.6 **文献标识码**：A **文章编号**：1000-7024 (2007) 09-2096-04

Porting uC/OS-II kernel to 80C 51-family microprocessor

ZHAO Jian-hua, WANG Wen-yong
(School of Computer Science and Engineering, UEST of China, Chengdu 610054 China)

Abstract: The kernel structure and the porting technology of uC/OS-II is introduced, which is a kind of embedded realtime operating system. The key technology and the process of porting uC/OS-II kernel to 80C51 family microprocessors are discussed, the testing experiment after porting is analyzed. Finally the difference about the porting to the compiler of keil Cx51 is pointed out, and the general transplanting methods are summarized.

Key words: 8051 microprocessor; realtime operating system; port; cross-compiler; test

0 引 言

近年来，嵌入式系统已成为后 PC 时代一个广阔的研发领域，其应用范围越来越广。实际应用中，由于移植所花费代价最小，原有操作系统无法支持新增应用功能时，常会采用系统移植来解决问题。uC/OS-II 作为一个高可靠，开源的嵌入式实时操作系统，具有广泛的应用；而利用廉价的 80C51 实现众多传感器、控制器等电子设备的网络互联，更是具有先天优势。因此在 80C51 上移植 uC/OS-II 具有重要的意义。本文以 uC/OS-II 为移植对象，以 80C51 为移植目标来详细讨论移植的过程，其中自己设计了一个堆栈结构；最后给出移植测试实验，分析了移植中要注意的一些问题，总结了移植的一般方法。

1 移植前的准备工作

移植前首先要对移植对象，移植目标及其结构，以及移植的编译环境进行了解。这样才会起到事半功倍的效果。以下便是从这几个方面进行简单分析。

1.1 移植对象——uC/OS-II

uC/OS-II 具有很强的可移植性，可以广泛应用于各类 8 位，16 位，32 位微控制器或 DSP 中。它具有完全可剥夺型的实时内核，其核心工作原理是让最高优先级的就绪任务处于运行状态；它具有多任务的特点，可以管理 64 个任务，其中 56 个任务分配给用户；另外它具有内核可裁减性，可确定性的特点，并提供很多系统服务，比如信号量，互斥信号量，事件标志，消息邮箱，消息队列，内存的分配和释放等。

1.2 目标机——80C51 硬件资源

在系统移植之前，必须先了解目标机的硬件资源，然后根据特定的硬件编写相应的代码。80C51 系列单片机具有结构简单，应用灵活等特点。其硬件资源如下：CPU：8 位的微处理器；内存：片内 RAM (128B)，片外 RAM (64KB)、片内 ROM (4KB)，片外 EPROM (64KB)；时钟：片内振荡器和时钟产生电路，振荡频率为 6~12 MHZ，2 个 16 位定时/计数器；中断：5 个中断源，两级中断；外设：4 个 8 位并行 I/O 接口 P0~P3；1 个全双工的串行 I/O 口(UART)。

1.3 移植中采用的编译器

移植中采用 Keil Cx51 编译器，整个移植在 Keil Cx51 开发平台上进行。Keil Cx51 是 Keil 公司的一款针对 C51 系列单片机的编译器，版本为 V7.0，它是目前最高效的，灵活的 80C51 开发平台。在 Keil Cx51 环境下移植 uC/OS-II 可直接进行软件仿真，仿真过程中不必将程序下载到硬件上运行。等程序在软件仿真平台测试通过后，直接将其烧录到硬件芯片上。

1.4 uC/OS-II 的模块简介

由于设计 uC/OS-II 时就考虑到了在不同处理器上移植，因而移植 uC/OS-II 实际上需要修改的代码量很小。整个嵌入

收稿日期：2006-04-23 E-mail：z_1982_jh@yahoo.com.cn
作者简介：赵建华（1982－），男，陕西商洛人，硕士研究生，研究方向为无线传感器网络（WSN）；汪文勇（1967－），男，教授，研究方向为无线传感器网络（WSN）、网络安全技术。

式系统的结构如图 1 所示，uC/OS-II 内核可以分为与处理器无关的代码，无处理器相关的代码以及与应用相关的代码 3 个部分。移植中只需修改与处理器相关部分的文件，即包括：CPU.H，OS_CPU_C.C 和 OS_CPU_ASM.ASM，其它代码几乎不需要改变。

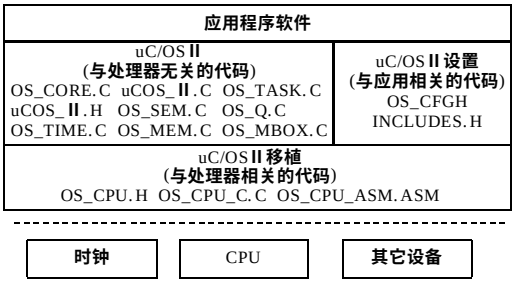

图 1　uC/OS-Ⅱ软件体系结构

2　具体移植步骤

以下是移植的步骤，分别修改各个与硬件相关的文件。

2.1　修改 OS_CPU.H 文件

OS_CPU.H 中包含两部分的代码，数据类型定义代码和与处理器相关的代码。移植主要修改与处理器相关代码。首先定义：EA=0 关中断；EA=1 开中断。这样定义即减少了程序行数，又避免了退出临界区后关中断造成的死机。由于 MCS-51 堆栈从下往上增长(1=向下，0=向上)，所以 OS_STK_GROWTH 定义为 0。最后，把 OSCtxSw()预定义为 OS_TASK_SW()。因为 MCS-51 没有软中断指令，所以用程序调用代替。实践表明，对于 MCS-51，用子程序调用入栈，用中断返回指令 RETI 出栈是没有问题的。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　修改 OS_CPU_C.C 文件

在这个文件中，只需要修改任务堆栈初始化函数 OSTaskStkInit()。uC/OS-II 中每个任务都有自己的堆栈空间，并且必须声明为 OS_STK 类型，主要完成对用户任务的堆栈进行初始化。OSTaskStkInit()函数总是返回栈顶地址。为了说明这个函数的工作流程，我们自己设计了一个堆栈空间，如图 2 所示。其中 OSTCBCur 指向当前任务控制块 TCB，TCB 结构体中 OSTCBStkPtr 指向用户堆栈的栈顶，用户堆栈长度存放在用户堆栈的最底部，长度之上空间存放系统堆栈映像，即：用户堆栈空间大小=系统堆栈空间大小+1。SP 总是先加 1 再存数据，因此，SP 初始时指向系统堆栈起始地址(OSStack)减 1 处(即 OSStkStart)。很明显系统堆栈存储空间大小=(SP － OSStkStart)。

任务切换时，先保存当前任务堆栈内容。即把系统栈数据拷贝到用户栈。方法是：用(SP-OSStkStar)得出保存字节数，将其写入用户堆栈最低地址内，以用户堆栈最低地址为起址，以 OSStkStart 为系统堆栈起址，由系统栈向用户栈拷贝数据，循环(SP-OSStkStart)次，每次拷贝前先将各自栈指针增 1。

其次，恢复最高优先级任务系统堆栈。方法是：获得最高优先级任务用户堆栈最低地址，从中取出“长度 Length”，以最高优先级任务用户堆栈最低地址为起址，以 OSStkStart 为系统堆栈起址，由用户栈向系统栈拷贝数据，循环“长度 length”数值指示的次数，每次拷贝前先将各自栈指针增 1。用户堆栈初始化时从下向上依次保存：用户堆栈长度(15)，PCL,PCH,PSW,ACC,B,DPL,DPH,R0,R1,R2,R3,R4,R5,R6,R7。不保存 SP，任务切换时根据用户堆栈长度计算得出。

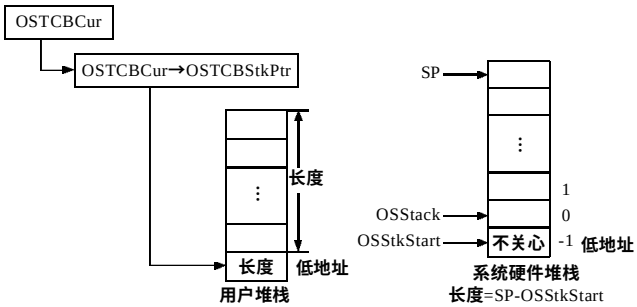

图 2　自定义的用户堆栈和系统堆栈结构

2.3　修改 OS_CPU_ASM. ASM 文件

uC/OS-II 移植实例要求用户编写 4 个简单汇编语言函数：

- OSStartHighRdy　//使就绪态任务中优先级最高的任务开始运行
- OSCtxSw　//低优先级任务切换到高优先级任务
- OSTickISR　//时钟节拍中断
- OSIntCtxSw　//在 ISR 中执行任务切换功能

2.3.1　修改 OSStartHighRdy

OSStartHighRdy 主要完成高优先级任务寄存器的恢复。必须恢复该任务在 CPU 使用权被剥夺时保留下来的全部寄存器的值，以便让这个高优先级任务能够继续运行。即把用户堆栈中的 PCL,PCH,PSW,ACC,B,DPL,DPH,R0,R1,R2,R3,R4,R5,R6,R7 全部拷贝到系统堆栈，然后系统堆栈再进行 POP 操作，将保存的这些值弹出到 CPU 的各个寄存器。使得此高优先级任务得到运行。其中部分关键代码如下：

```
MOVX A,@DPTR     ;调整 DPTR，使其指向用户堆栈的最低地址
MOV  R5,A        ;R5=用户堆栈长度，因为堆栈长度放在用户堆栈最低处
MOV  R0,#OSStkStart ; OSSktStart 为系统堆栈起始地址
restore_stack:   ; 从用户堆栈到系统堆栈的 copy
INC  DPTR
INC  R0
MOVX  A, @DPTR
MOV  @R0, A
DJNZ  R5, restore_stack
POPALL           ; 为自定义宏，将所有寄存器实现出栈
```

2.3.2　修改 OSCtxSw

OSCtxSw 为任务切换函数，实现从低优先级任务到高优先级任务切换。要实现任务切换，一般包括两个过程：①保存当前低优先级任务的全部寄存器的值以及堆栈的长度值；②恢复高优先级任务以前在 CPU 使用权被剥夺时保存下来的全部寄存器的值，即前一个汇编程序 OSStartHighRdy 实现的功能。其中部分关键代码如下：

```
; 以下代码完成第①步操作
```

```
PUSHALL ; 用户定义的宏,将全部 CPU 寄存器保存到系统堆栈
MOV  A, SP ;  以下 3 句实现获得堆栈的长度
SUBB  A, #OSStkStart
MOV  R5,A   ;R5 里保存堆栈长度,并将其保存在用户堆栈最底下
save_stack:   ;以下实现从系统堆栈到用户堆栈的拷贝
INC   DPTR ; 指向用户堆栈
INC   R0   ; 指向系统堆栈
MOV   A,  @R0
MOVX  @DPTR, A
DJNZ   R5, save_stack ;
```

而第②步代码和 OSStartHighRdy 代码一样,除此之外,将当前优先级最高任务的任务控制块指针 OSTCBHighRdy 赋值给当前任务控制块指针 OSTCBCur。

2.3.3 修改 OSTickISR

uC/OS-II 要求用户提供一个周期性的时钟源,来实现时间的延迟和超时功能。我们选用 8051 的 T0 定时器作为 tick 时钟。OSTickISR 函数为时钟节拍中断服务程序,当产生计时和定时中断时,执行此函数。此函数和其它中断服务子程序一样,都先要保存断点和保存现场,然后执行用户代码,最后对恢复保存的寄存器值。其中关键代码:

```
CSEG AT 000BH   ;0BH 为 0 号定时器 T0 的入口地址
LJMP OSTickISR   ;跳转到中断服务子程序
OSTickISR:  PUSHALL  ;保存所有寄存器的宏
CLR TR0 ;后面 4 句为设置 T0 的工作方式以及赋值
MOV  TH0,#70H   ;定义 Tick=50 次/秒(即 0.02 秒/次)
MOV  TL0,#00H   ;OS_CPU_C.C 和 OS_TICKS_PER_SEC
SETB TR0
LCALL _? OSTimeTick ; 调用系统 Tick 函数
LCALL _? OSIntExit ; 系统函数,决定是否使得高优先级任务就绪
POPALL  ; 出栈
RETI
```

2.3.4 修改 OSIntCtxSw

OSIntExit 通过调用 OSIntCtxSw,在 ISR 中执行任务切换功能。因为 OSIntCtsSw 是在 ISR 中被调用的,所以假定寄存器都被正确地保存了被中断的任务的堆栈之中。因此,OSIntCtxSw 和上文中 OSCtxSw 的代码基本相同,差别仅仅在于开始之前少了一个 POPALL 操作。

3 实验及其结果分析

移植完成后,紧接着就是移植的正确性验证。下面是移植测试实验以及对实验结果的分析。

3.1 实验的条件和指导思想

实验必须确保在 Cx51 编译器和链接器的正常工作的情况下进行。主要指导思想是测试修改过的 OSStartHighRdy(),OSTaskStkInitOSCtxSw,OSIntCtxSw,OSTickISR 等 5 个函数,测试其在 Cx51 环境下运行是否正常。

3.2 实验的过程和结果

实验过程主要包括以下 3 个步骤:①创建两个任务 Task1 和 Task2,验证 OSTaskStkInit 和 OSStartHighRdy()函数;②创建 Task1 和 Task2 后,通过二者之间的切换,验证 OSCtxSw 函数;③初始化时钟,开中断。Task1 和 Task2 各自睡眠一段时间(Task2 睡眠时间是 Task1 的 2 倍)。等睡眠时间到达后,根据各自输出的不同结果,验证 OSIntCtxSw 和 OSTickISR 函数。

主要测试代码如下:

```
OSTaskCreate(Task1, (void *)0, &TaskSkt1[0],2);
OSTaskCreate(Task2, (void *)0, &TaskSkt2[0],10);
void Task1(void *data1) reentrant
  { for(;;)
    {PrintStr("\tTask1 is active.11111111\n");
     OSTimeDly(OS_TICKS_PER_SEC);}
  }
void Task2(void *data2) reentrant
  {for(;;){
   PrintStr("\tTask2 is active.22222222\n");
   OSTimeDly(2*OS_TICKS_PER_SEC);}
}
```

3.3 对实验结果的分析和讨论

输出结果为:

```
Task1 is active.11111111
Task1 is active.11111111
Task2 is active.22222222
Task1 is active.11111111
Task1 is active.11111111
Task2 is active.22222222
```

根据实验输出结果可见,Task1 和 Task2 创建成功,因此 OSTaskStkInit 工作正常;Task1 和 Task2 实现任务切换,以及发生时间中断,工作正常(Task1 运行两次后 Task2 运行一次),因此其它 4 个函数工作也正常。由此可以得出结论,uC/OS-II 已经在 80C51 上移植成功。

3.4 移植中 keil Cx 编译器要注意的问题

由于 uC/OS-II 是一个可抢占式内核,因此,系统中的绝大多数函数都应该是可重入的。而在 Keil Cx51 编译器中,在函数定义时的默认值都是不可重入的,因此,需要在系统中的每一个函数的声明以及定义处都加上“large reentrant”的修饰符,以保证函数的可重入性。

startup.a51 文件是 Cx51 编译器自带的文件,是 C51 的初始化代码,单片机复位后先执行这段代码,完成初始化后由它调用 main ()。其主要完成定义内部 RAM 大小、外部 RAM 大小、可重入堆栈位置,以及初始化 8051 硬件堆栈指针。因此我们要修改这个文件中的一些变量,比如外部 RAM 起始地址变量 XDATA START,是否大模式重入堆栈指针需初始化标志变量 XBPSTACKTOP 等等。这样的话,程序在软件仿真通过测试后,将其烧录在硬件上,硬件调试也一次成功。

4 结束语

本文介绍了 uC/OS- II 移植到 80C51 的全过程,对从事嵌

入式系统移植工作的开发人员来说有一定的参考价值。论文对其它类似系统的移植也具有很重要的参考意义，比如可以尽量把移植对象 ERTOS 划分为与硬件相关代码和与硬件无关代码，抽象出其硬件抽象层 ;接着，结合目标机的硬件资源，对移植对象中与硬件相关的代码进行修改；最后对内核自身进行测试。

参考文献:

[1] Jean J Labrosse. 嵌入式实时操作系统 uC/OS-II[M].第 2 版.北京:北京航天航空大学出版社, 2005.283-316.

[2] 马忠梅.单片机的 C 语言应用程序设计[M].北京:北京航空航天出版社, 2003.120-135.

[3] 徐爱钧, 彭秀华. Keil Cx51 V7.0 单片机高级语言编程与 uVision2 应用实践[M].北京:电子工业出版社, 2005. 490-450.

[4] 宁杰城,王春.ARM7 内核上的 uC/OS—II 嵌入式系统移植[J].中国测试术, 2005,(3):65-66.

[5] 黄涛,徐宏吉.嵌入式实时操作系统移植技术的分析与应用[J].计算机应用, 2003,(9):88-90.

[6] 张谦,竹利平. μC/OS-II 实时嵌入式操作系统的实时性分析与测试[J]. 计算机工程与设计, 2005,26(9):2422-2424.

[7] 霍妍,孟凡荣.基于μC/OS-II 嵌入式网络实验平台的研究与实现[J].计算机工程与设计, 2006,27(4):656-657.

[8] 白智国,王芳,冯丹.嵌入式系统移植问题的研究[J].计算机工程与科学, 2005,27(6):97-99.

[9] 董余红,阎俊武 .uC/OS-II 在 51 系列单片机上的移植[J].导弹与航天运载技术, 2004,(6):36-40.

(上接第 2002 页)

步骤 3　进行单亲遗传操作。当m只蚂蚁都走完n个城市后，将每个蚂蚁的禁忌表看成一个染色体，染色体的集合看成是种群。更新具有全局最短路径的蚂蚁的信息，并将具有全局最短路径的蚂蚁的禁忌表也加入该种群。对种群中的每个染色体都按给定的概率作单亲移位、倒位、换位、变异遗传操作，生成新染色体，将每个染色体加入到种群中。

步骤 4　进行选择操作。进行完步骤 3 后，此时种群中有$m+u$个染色体，按第 2 节的方法 5 从种群中选择出m个染色体来。将蚁群中m只蚂蚁的禁忌表分别换成本次选择出的m个染色体。

步骤 5　信息素更新。按式 (2) 更新信息素矩阵后，清空所有蚂蚁的禁忌表，将m只蚂蚁随机分布于n个城市上，并将出发点城市置于蚂蚁的禁忌表中。

步骤 6　如当前进化代数<=NC，转步骤 2。

步骤 7　得到全局最短路径。

4　实验结果

为验证、比较算法的效果，我们采用了一个常用的 30 城市的 TSP 数据集 oliver30。oliver30 已知的整数型理论最优解为 420。为进行公平的比较，除单亲遗传算子所特有的一些参数之外，ACA 和 ACPGA 算法的其它参数均一样。两种算法各在该数据集上各进行 20 次实验，取平均结果进行比较。

由表 1 可看出，20 次实验中，仅平均进化了 155 次，ACPGA 算法就都得到了理论最优解。而 ACA 算法在 20 次实验中，只得到了一次最优解，且求解效率远低于 ACPGA。得到的最优路径如图 1 所示。

表 1　ACPGA 与 ACA 在 oliver30 数据集上 20 次实验结果比较

	最差解	最好解	得到最好解次数	平均解	得到平均解平均进化次数
ACPGA	420	420	20	420	155
ACA	429	420	1	424.26	554

参数Nc=1000, Q=100, α=1, β=5, ρ=0.5, τ_{min}=0.001, τ_{max}=5, p_t=p_h=p_d=0.8, p_b=0.5

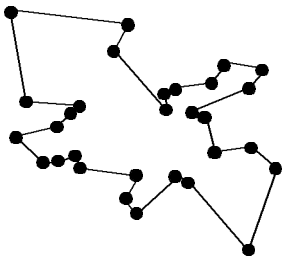

图 1　oliver30 最优路径

5　结束语

本文提出的具有单亲遗传特征的蚁群算法，可以有效的克服基本蚁群算法中收敛速度慢，解性能差的缺陷，有利于实际应用。实验结果证明，具有收敛速度快，解质量较好的特点。

参考文献:

[1] 丁建立,陈增强,袁著祉.遗传算法与蚂蚁算法的融合[J].计算机研究与发展,2003,40(9):1351-1356.

[2] 朱庆保,杨志军.基于变异和动态信息素更新的蚁群优化算法[J].软件学报,2004,15(2):185-192.

[3] 陈烨. 带杂交算子的蚁群算法 [J]. 计算机工程, 2001,27 (12): 74-76.

[4] 张静乐,王世卿,王乐.具有新型遗传特征的蚁群算法[J].微计算机信息,2006,22(2):261-263.

[5] 李茂军,童调生.单亲遗传算法及其收敛性分析[J].自动化学报, 1999,25(1):68-72.

[6] 李茂军,罗日成,童调生.单亲遗传算法的遗传算子分析[J].系统工程与电子技术,2001,23(8):84-87.

[7] 祝延军,胡纯德,高随祥. 最短路由问题的改进单亲进化遗传算法[J].计算机工程与应用,2005,41(8):64-67.

[8] 祝延军,胡纯德,高随祥.单亲进化遗传算法在配送中心选址中的应用[J].计算机工程与设计,2005,26(3):580-582.

[9] 李茂军,童调生.单亲遗传算法的选择方式[J].系统工程与电子技术,2002,24(10):87-89.