uCOS51 重入问题的解决

巨龙公司系统集成开发部 杨屹 asdjf@163.com 2002/10/09

引言

自从发表《uCOS51 移植心得》以来，我收到了很多朋友们的来信，大家对公开源码表示鼓励，谢谢大家的支持！很多人对于编写自己的操作系统很感兴趣，uCOS51 是个不错的选择。它的优点是简单易懂，学习成本低，有利于向 32 位 CPU 过渡。目前，嵌入式 BBS 上的热点是：嵌入式实时多任务操作系统、单片机上网、32bitCPU（如 ARM 等）。其实通过 uCOS51 学习完全可以掌握这些热门技术的精髓，而且学习成本低廉。为此我会陆续将我在研发过程中的经验体会写出来与大家交流，共同进步。

我准备讨论以下内容：uCOS51 高效内核、OS 人机界面 SHELL 的编写、51 机开发板的硬件设计、RTL8019AS 网卡驱动程序、51TCP/IP 协议栈设计、应用协议 FTP、PPP、HTTP、SMTP、SNMP……在 51 上的实现技术、51OS 任务划分和应用程序实例、由 51 软件系统向 ARM 的移植以及其他想到的题目。欢迎大家积极参与。

注：开发板原理图、PCB 图、GAL 烧录文件、芯片手册、全部源程序可以来信索取，在整理好后会共享在网上。

讨论 1----uCOS51 高效内核

在提供了大量 uCOS51 β 测试版后，zxgllp 网友提出了 OS 程序不支持函数重入问题，具体表现在任务函数中带有形参和局部变量时若使用 reentrant 关键字，任务函数重入时会导致致命错误，经查属实。

具体原因是 OS 切换程序没有保存/恢复仿真堆栈内容。由于我刚接触 KEIL 对其细节不熟悉，参考的范例中有些不是 KEIL 编译的，没有处理仿真堆栈内容，我也理所当然地认为 KEIL 象 TC 一样自动处理了重入问题，所以导致致命错误。(最新版本中已改正！)

我仔细研究了 C51.PDF 129-131 页的内容，了解到：为了函数重入，形参和局部变量必须保存在堆栈里，由于 51 硬件堆栈太小，KEIL 将根据内存模式在相应内存空间仿真堆栈(生长方向由上向下，与硬件栈相反)。对于大模式编译，函数返回地址保存在硬件堆栈里，形参和局部变量放在仿真堆栈中，栈指针为?C_XBP，XBPSTACK=1 时，起始值在 startup.a51 中初始化为 FFFFH+1。仿真堆栈效率低下，KEIL 建议尽量不用，但为了重入操作必须使用。KEIL 可以混合使用 3 种仿真堆栈(大、中、小模式)，为了提高效率，针对 51 我推荐统一使用大模式编译。

为了支持重入，我重新设计了堆栈结构(如下图)。增加了保存仿真堆栈指针?C_XBP 和堆栈内容的数据结构。相应改变的文件有：OS_CPU_A.ASM、OS_CPU_C.C、OS_CPU.H、YY.C。由图可知，用户栈中保存的仿真栈与硬件栈相向生长，中间为空闲间隔，显然 uCOSII 的堆栈检测函数失效。硬件栈的保存恢复详见《移植心得》，仿真堆栈的保存与 8086 移植中的一样，OS 只提供堆栈空间和只操作堆栈指针，不进行内存拷贝，效率相对很高。

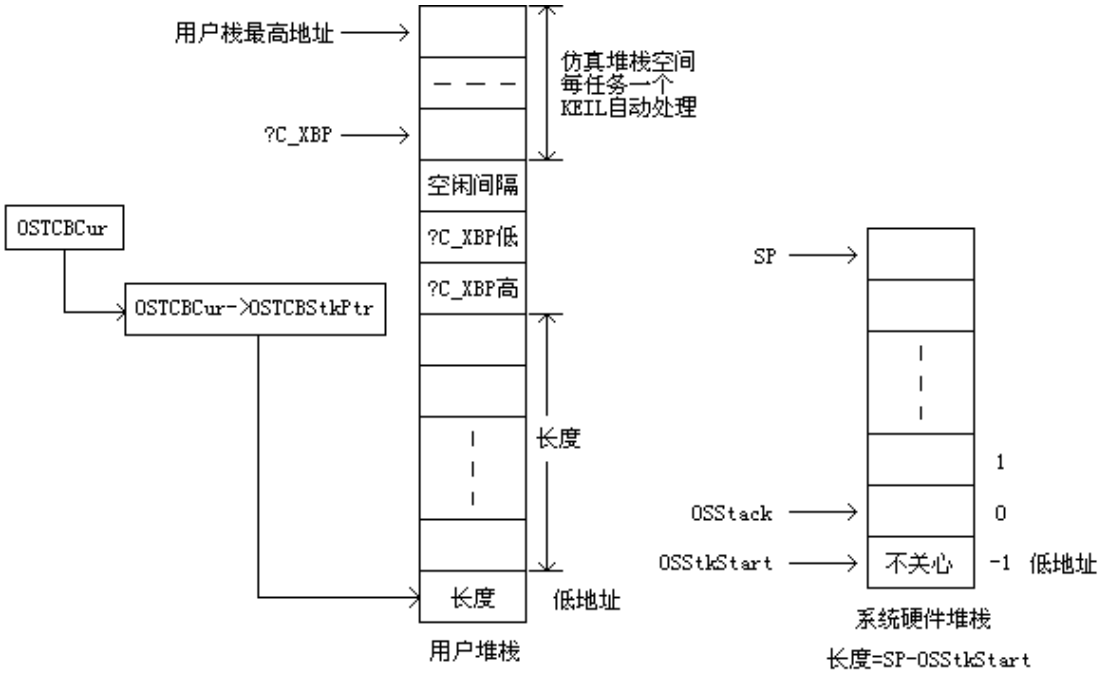
我建议使用统一的固定大小的堆栈空间，尽管 uCOSII 原作者把不同任务使用不同空间看成是优点，但为了在 51 上有效实现任务重入，针对 51 我还是坚持不使用这个优点。

用户堆栈空间的大小是可以精确计算出来的。用户堆栈空间=硬件堆栈空间+仿真堆栈空间。硬件栈占用内部 RAM，内部 RAM 执行效率高，如果堆栈空间过大，会影响 KEIL

编译的程序性能。如果堆栈空间小，在中断嵌套和程序调用时会造成系统崩溃。综合考虑，我把硬件堆栈空间大小定成了 64 字节，用户根据实际情况可以自行设定。仿真堆栈大小取决于形参和局部变量的类型及数量，可以精确算出。因为所有用户栈使用相同空间大小，所以取占用空间最大的任务函数的空间大小为仿真堆栈空间大小。这样用户堆栈空间大小就唯一确定了。我将用户堆栈空间大小用宏定义在 OS_CFG.H 文件中，宏名为 MaxStkSize。

51 的 SP 只有 8 位，无法在 64K 空间中自由移动，只好采用拷贝全部硬件堆栈内容的笨办法。51 本来就弱，这么一来缺点更明显了。其实，引入 OS 必然要付出代价，一般 OS 要占用 CPU10%-20%的负荷能力，请权衡利弊决定。切换频率决定了 CPU 的耗费，频率越高耗费越大，大到一定程度就该换更强的 CPU 了。我选了 50Hz 的切换频率，不高也不低，用户可以根据需要自行定夺。在耗费无法避免的情况下，我采取了几个措施来提高效率：1。ret 和 reti 混用减少代码；2。IE、SP 不入出栈，通过另外方式解决；3。用 IDATA 关键字声明在汇编中用到的全局变量，变 DPTR 操作为 Ri 操作；4。设计堆栈结构，简化算法；5。让串口输入输出工作在系统态，不占用任务 TCB 和优先级，增加弹性缓冲区，减少等待。

任务堆栈结构:



堆栈结构图