doi: 10.7690/bgzdh.2019.03.007

μC/OS-III 操作系统的优化

乔　焱，陈　媛，贾家宁，黄一敏
（南京航空航天大学自动化学院，南京 210016）

摘要：针对 μC/OS-III 系统存在启动需要延时、统计任务耗时过长等问题，以 MPC555 为平台对 μC/OS-III 系统进行优化。采用 Tick 法解决 μC/OS 系统启动延时的问题，同时采用剔除任务堆栈统计的方法解决统计任务耗时过长的问题。使用 Tick 法后，μC/OS 系统启动无延时，提高了统计任务效率，降低了 CPU 负荷，使优化后的系统可靠性更高。

关键词：μC/OS-III；操作系统优化；启动延时；统计任务
中图分类号：TP302.7　　**文献标志码：**A

Optimization of μC/OS-III Operation System

Qiao Yan, Chen Yuan, Jia Jia'ning, Huang Yimin
(*College of Automation Engineering*, *Nanjing University of Aeronautics & Astronautics*, *Nanjing* 210016, *China*)

Abstract: Aiming at the problems of startup delay and long time consuming of statistical task in μC/OS-III system, the μC/OS-III system is optimized based on MPC555 platform. The Tick method is used to solve the startup delay of μC/OS system, and the problem of long time consuming of statistical tasks is solved by eliminating task stack statistics. After using the Tick method, the startup time delay of the μ C/OS system is solved, and the time consumption of the statistical task is reduced as well. After the optimization, the system is more reliable.

Keywords: μC/OS-III; operation system optimization; startup delay; statistical task

0　引言

迄今为止，μC/OS-III 操作系统已经在 ARM、PowerPC、X86 等 CPU 上得到了广泛应用[1]，但仍存在一些问题，例如系统启动延时、统计任务耗时过长等，而 Tick 法和针对统计任务的优化能够有效地解决这些问题。笔者以 MPC555 为平台对 μC/OS-III 操作系统进行优化[2]。

在系统启动过程中，由于原系统采用 Count 法计数统计 CPU 负荷，需要延时 1 s 来获得一个计数的基准值[3]，将导致系统启动缓慢；因此，需要一种计算 CPU 负荷不需要上电延的方案。

在统计任务中，由于原统计任务中包含对堆栈信息的计算，而 μC/OS 系统采用的堆栈统计方法为数零法[3]，较为耗时，尤其现在的任务为防止堆栈溢出采用的堆栈都较大，更加重了统计任务的负担，因此笔者针对统计任务进行了优化。

1　无延时启动

1.1　Count 法计算 CPU 负荷

μC/OS 操作系统原生的 CPU 负荷计算方法为 Count 法。此方法需要上电延时产生 CPU 负荷计算的基准值，其使用 Count 法计算 CPU 负荷时系统运行时序如图 1 所示。

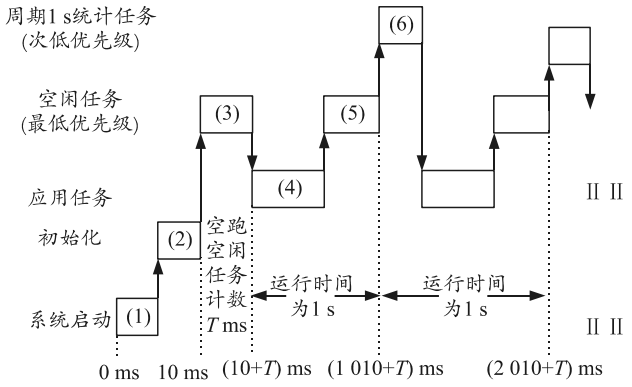

图 1　使用 Count 法计算 CPU 负荷时系统运行时序

图中(1)和(2)系统启动，初始化硬件资源和软件资源。

图中(3)此时除系统内部任务外只有空闲任务运行 T ms，能够统计出 T ms 空闲任务计数最大值，并转化为 1 s 空闲任务的最大值并清零计数值。例如 200 ms 空闲任务计数值为 1 000，则 1 s 空闲任务的最大值记为 1 000 ×10 = 10 000。

图中(4)系统已准备完毕，开始正式地运行应用任务。

收稿日期：2018-10-16；**修回日期：**2018-11-28
作者简介：乔　焱(1993—)，男，江苏人，硕士，从事无人机、操作系统研究。

图中(5)由于空闲任务处于低优先级，应用程序运行结束，继续运行空闲任务开始计数。

图中(6)周期为 1 s 的统计任务开始运行，获取 1 s 内空闲任务的计数值，将 1 s 空闲任务的最大值和 1 s 内空闲任务的计数值代入公式即可获得 CPU 负荷。计算公式如下：

$$\text{Cpuusage}=1-\frac{\text{系统运行1 s内空闲任务计数}}{\text{系统启动后}T\text{ ms空闲任务计数值}/T\text{ ms}\times 1\,000}。$$

上述公式中分母中 T ms 时间内空闲任务的计数值除以 T ms 即获得 1 ms 内空闲任务的计数值，再乘 1 000 便获得了 1 s 内空闲任务的计数值，由于系统启动后无运行应用程序任务，故此时 1 s 内空闲任务的计数值是 CPU 运行 1 s 内空闲任务计数的最大值，计算公式可表示如下：

$$\text{Cpuusage}=1-\frac{\text{系统运行1 s内空闲任务计数}}{\text{1 s内空闲任务计数最大值}}。$$

由时序图可知：Count 法计算 CPU 负荷需要先获得 1 s 内 μC/OS 操作系统仅运行空闲任务的统计值，便需要 μC/OS 系统启动时空跑空闲任务 T ms 来获得统计值。这样便延长了系统的启动时间，并且由于硬件设备和 μC/OS 操作系统刚启动，空闲时间 T 较长，空闲任务计数才相对稳定，可能会造成 CPU 负荷计算偏差的情况。

1.2　Tick 法计算 CPU 负荷

针对 Count 法计算 CPU 负荷的弊端，开发了基于 Tick 法的 CPU 负荷计算方法[4]，Tick 法计算 CPU 负荷实现流程如图 2 所示。

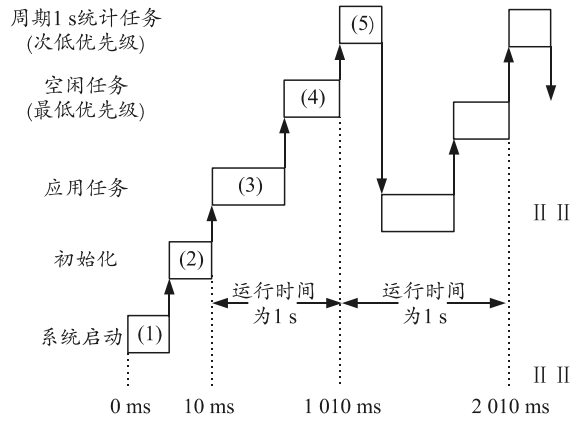

图 2　使用 Tick 法计算 CPU 负荷时系统运行时序

图中(1)和(2)系统启动，初始化硬件资源和软件资源。

图中(3)系统已准备完毕，开始正式运行应用任务。

图中(4)空闲任务处于最低优先级，原本用于计算 Count 法 CPU 负荷，但在运用 Tick 法计算 CPU 负荷时本可以不需要空闲任务。为了让 CPU 空闲时有事可做，仍保留这一任务。

图中(5)统计任务是周期为 1 s 的任务，Tick 法计算 CPU 负荷只需获取 1 s 内空闲任务的运行时间，将其值代入公式为：

$$\text{Cpuusage}=\frac{\text{系统运行1 s内任务运行时长}}{\text{1 s}}。$$

由上式可知，Tick 法计算 CPU 负荷需要获得系统 1 s 内所有任务运行时长。这样需要将所有单任务运行时间相加，较为繁琐，系统不再运行应用任务，便运行最低优先级的空闲任务，应用任务运行的时间就等于 1 s 减去空闲任务的运行时间，所以计算空闲任务运行时间即可，计算公式如下：

$$\text{Cpuusage}=1-\frac{\text{系统运行1 s内空闲任务时长}}{\text{1 s}}。$$

μC/OS-III 中每个任务有单独的结构体数组来存储其相应的统计值包含任务的运行时间。μC/OS-III 系统获取任务运行时间是通过记录任务切入和切出时的时间戳做差，以获得任务运行的时间，任务时间获得原理如图 3 所示。

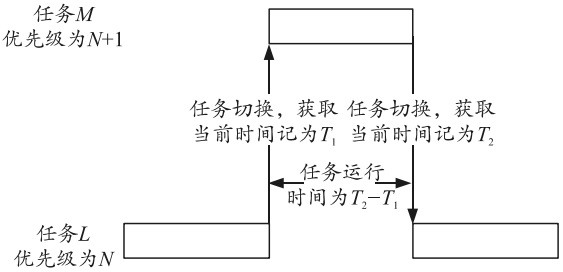

图 3　Tick 法计算任务时间原理

由于 μC/OS-III 系统只有相应的函数接口，并没有针对已有的 MPC555 处理器配置相应的时间戳函数，因而需要对其进行配置。可选择的方案有如下 2 种：通过 MPC555 的高精度定时器来计算当前的时间戳，通过 μC/OS-III 系统时钟节拍 Tick 来计算当前时间戳[5]。

采用 MPC555 高精度定时器的方案不仅会额外占用宝贵的定时器的资源，且今后若将 μC/OS-III 移植到其他 CPU 上时，由于高精度定时器的差异，还需要重新配置其时间戳；采用 μC/OS-III 系统时钟节拍 Tick 来计算当前时间戳与系统共用同一定时器，不占用额外定时器且移植方便。基于以上考虑，笔者采用 μC/OS-III 系统时钟节拍即 Tick 来计算当前时间戳。原理如图 4 所示。

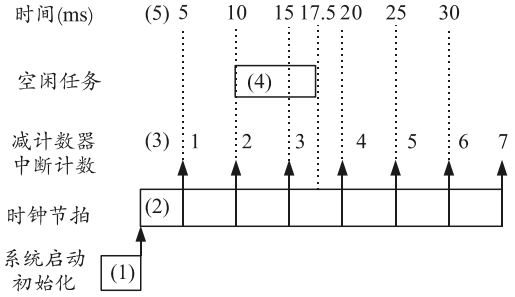

图 4 系统时钟节拍计算时间戳原理

图中(1)系统启动，初始化硬件资源和软件资源。

图中(2)时钟节拍，系统默认设置为 5 ms/Tick。

图中(3)MPC555 的 DEC 减计数器初值为 5 000，每 1 μs 减 1，减至 0 后触发中断，触发减计数器重新赋值，并且让减计数器中断计数加 1。

图中(4)空闲任务正在运行。

图中(5)通过减计数器中断计数，可以获得以 5 ms 为跨度的时间戳，只需用减计数器中断次数乘以时钟节拍 5 ms 便可。

图中(6)以 5 ms 为跨度计算任务运行时间显然太长，因而从减计数器中直接读取当前剩余计数值，用 5 000 减去当前已计数的值便为剩余计数值。例如图中空闲任务开始于 10 ms 结束于 15 ms+(5 000−2 500)/1 000=17.5 ms。运行时间计算原理如图 5 所示。

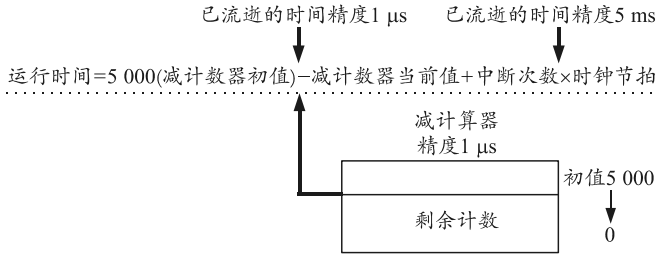

图 5 运行时间计算原理

2 优化统计任务

在 μC/OS-Ⅲ中存在一个内部统计任务，是实现软件状态监测的重要任务，帮助系统周期性地统计系统负荷、任务负荷、任务堆栈和任务运行时间等信息，较 μC/OS-Ⅱ增加了许多统计信息。统计任务将会在一个统计周期内计算出所有任务的已用堆栈，其任务堆栈检测机制采用遍历堆栈法，即系统初始化时将堆栈置零，统计任务在运行时从任务堆栈的底部到顶部遍历任务栈空间，计算为 0 的数值项个数[6]。实验采用的处理器为 MPC555，其堆栈增长方式为向下生长，从内存高地址向低地址方向增长，称为递减堆栈，其任务堆栈检测方法如图 6 所示。

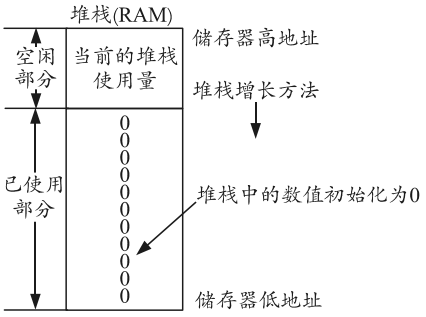

图 6 递减任务堆栈检测方法

任务堆栈的空间通常应该多分配 10%～100%，因而实验室考虑到任务的堆栈需和防止任务堆栈溢出，最终采用的任务堆栈大小为 8 K[7]，由于采用的堆栈较大，且堆栈的检测机制虽然易实现但较为耗时，会导致整个统计任务耗时过长[8]。飞控程序的任务一般由几十个任务组成，若采用 μC/OS-Ⅲ原本的统计任务，其耗时势必会随任务数量的增加而增加。统计任务整体耗时与堆栈任务耗时对比如表 1 所示。

表 1 统计任务整体耗时与堆栈任务耗时对比

应用程序任务数	统计任务耗时/ms	堆栈检测耗时/ms	占统计任务耗时/%
1	3.119	3.091	99.1
10	8.787	8.730	99.3
20	16.263	16.175	99.4
30	23.080	22.960	99.5
40	30.660	30.508	99.5
50	38.341	38.158	99.5
60	46.532	44.205	99.5

由表可知：统计任务的耗时几乎完全被堆栈检测耗时占用，且随着任务数量的增长，堆栈检测耗时呈线性增长，而其余统计信息的检测耗时却无明显增长，由此可得出堆栈检测将大大影响统计任务耗时的结论。

实验飞行控制程序的应用程序任务数在 60 左右，因而统计任务耗时在 46.5 ms 左右，由于统计任务周期为 1 000 ms 调用一次，故其统计任务的 CPU 负荷为 5%左右，占用 CPU 负荷较高因而需要对统计任务进行优化。优化方案采取剔除统计任务中的任务堆栈检测，仅在需要查询时调用，且 μC/OS-Ⅲ属于嵌入式操作系统，而为防止内存泄漏，嵌入式操作系统采用安全的静态分配内存，任务堆栈不会释放和重新分配，加上堆栈检测采取简易的遍历堆栈法，只能计算出任务非运行状态时已用的堆栈大小，因而在周期任务内计算所有任务的已用堆栈意义并不大，故将其从统计任务中剔除，减少了统计任务的 CPU 开销。

（下转第 34 页）

的订阅发布系统是指订阅发布系统支持事件代理框架所对应的拓扑结构的动态重配置，是新型高效、面向大规模的事件代理框架。重配置管理分为基于静态模式的和基于结构化模式的 2 种方法。静态模式指由多个代理组成的覆盖网络一旦形成以后，其拓扑结构几乎不会发生变化。本质上是静态的。结构化模式具有良好的自组织性、容错性和扩展性，但是网络中的每个节点需在所有生成树中向根节点传播订；因此，在重配置管理时，订阅表的维护成本很高。文中系统应用 RS3DS 的动态重配置模型，保证订阅发布的高效性，通过调节事件、订阅分区的大小，改变节点的订阅、事件负载，适应不同流量特点的应用，实现动态高效重配置和信息发送接收的灵活性互联。

5 结束语

随着试验任务规模扩大，汇集于指控中心的不同格式的试验信息种类也逐渐增多，迫切要求发展统一的、通用性强的靶场测控信息通信互联系统来保障系统间的实时传输。笔者以靶场测控试验网为平台，设计多层协议规范字典技术，实现根据不同需求自行编辑协议帧格式的灵活配置，设计适用于各种不同规范的、有高度灵活性和可扩展性的、通用化多层架构的测控信息交互系统，具有一定的推广价值。

参考文献：

[1] 张李平. 通用性地面监测系统软件架构的设计与实现[D]. 成都: 电子科技大学, 2015 (3): 28-29.

[2] 陈章毅. 基于 ATCA 的电信多标准通用平台软件设计与实现[D]. 上海: 上海交通大学软件学院, 2009: 3-5.

[3] 曲鹏, 周应旺, 崔雷. 小型无人机发动机控制系统 CAN 总线通信技术[J]. 兵工自动化, 2017, 36(8): 57-61.

[4] 张吉敏. 测控信息网传输协议研究[D]. 西安: 西安电子科技大学, 2011: 9.

[5] 张鹏伟, 于文新. 基于 Hibernate 的数据操作效率提升研究[J]. 科教导刊(电子版), 2013(19): 129-130.

[6] 罗青林, 徐克付, 臧文羽, 等. Wireshark 环境下的网络协议解析与验证方法[J]. 计算机工程与设计, 2011, 32(3): 772-772.

[7] 唐文娟, 陈丽娜. 基于 Hibernate 持久层性能优化方案的研究[J]. 智能计算机与应用, 2012, 2(1): 57.

[8] 陈正举. 基于HIBERNATE的数据库访问优化[J]. 计算机应用与软件, 2012, 29(7): 144-149.

[9] 黄辰虎, 陆秀平, 王克平, 等. 海道测量水位改正通用软件研制[J]. 海洋测绘, 2014, 34(5): 51.

[10] 张永峰. 基于 UDP 协议的航空发动机振动实时监视系统设计[J]. 测控技术, 2015, 34(3): 56.

****

（上接第 30 页）

任意任务的堆栈信息可由应用程序内调用查询，达到了既能实现任务堆栈检测的功能，又能优化统计任务的目的。

3 结束语

针对 μC/OS-Ⅲ启动延时和统计任务方面瑕疵，笔者给出了解决方案。该方案能够有效地解决启动延时问题，并能提高统计任务的效率，降低其占用 CPU 的负荷。

参考文献：

[1] 张扬, 李恒, 谭洁. 基于 Cortex-M4 处理器的 μC/OS-Ⅲ移植分析与实现[J]. 工业仪表与自动化装置, 2017(6): 15-19, 64.

[2] 童鑫. μC/OS-Ⅱ的移植与堆栈改进[D]. 武汉: 武汉理工大学, 2006.

[3] LABROSSE J J, 邵贝贝. 嵌入式实时操作系统 μC/OS-III[M]. 北京: 北京航空航天大学出版社, 2012: 20-500.

[4] 张国强. CPU利用率的估算[J]. 冶金丛, 2002(5): 16-17, 26.

[5] 尹传高, 杨跃武. 操作系统[M]. 北京: 电子工业出版社, 2000: 31-69.

[6] 吴光文, 周航慈. Cx51 程序设计的堆栈空间计算方法[J]. 单片机与嵌入式系统应用, 2010(12): 68-69.

[7] 原义盈. 嵌入式软件堆栈溢出的静态测试方法研究[D]. 北京: 北京交通大学, 2011: 21-36

[8] 黄土琛, 宫辉, 薛涛, 等. 在 CodeWarrior 编译环境下运行 μC/OS-Ⅲ[J]. 单片机与嵌入式系统应用, 2012, 12(12): 23-26.