Nucleus PLUS

参考手册

译者注：由于译者水平有限，阅读时请参照英文文档，欢迎批评指正

Email: nosuken@21cn.com http://icdiscovery.myrice.com/

概要

第一章　介绍

本章提供 Nucleus PLUS 内核的概况，以及和实时多任务应用程序的关系。

第二章　开始

本章描述如何安装和应用 Nucleus PLUS 软件。

第三章　功能描述

本章包含 Nucleus PLUS 完整的功能描述，包括任务运行，任务通信，任务同步，时钟，内存管理，输入/输出驱动，和中断。

第四章　Nucleus PLUS 服务

本章包含 Nucleus PLUS 每一种服务的描述。

第五章　**扩展讨论**

本章包括了不同的高级问题的扩展讨论。

第六章　样例系统

本章描述了一个很小但是很完整的 Nucleus PLUS 样例程序。

附录 A　Nucleus 常量

本附录包含所有对用户有用的 Nucleus 常量的真实值

附录 B　系统致命错误

本附录包含致命错误代码值。

附录 C　I/O 驱动器请求结构

本附录包含标准的 I/O 驱动器请求结构。

叙述文档

Nucleus PLUS 内部的，Accelerated Technology，描述，相当详细，Nucleus PLUS 内核的执行。

Nucleus PLUS 移植笔记，Accelerated Technology，描述处理器和为每个 Nucleus PLUS 端口指定的开发工具细节。

Nucleus PLUS 调试器，Accelerated Technology，描述多任务 Nucleus PLUS 调试器的安装和运行。

Nucleus FILE Manual(文件手册),Accelerated Technology,描述了 MS－DOS 兼容的 Nucleus PLUS 文件系统的安装和运行。

Nucleus NET Manual(网络手册)，Accelerated Technology,描述了 TCP/IP Nucleus PLUS 网络工具的安装和运行。

The Nucleus Reactor(反应堆)，Accelerated Technology，一般来讲就是把一个时事通信投入到实时编程当中和 Accelerated Technology 提供的实时产品家族。

The C Programming Language（第二版），由 Brian W.Kemighan 和 Dennis M.Ritchie 编写，Prentice-Hall 发布，1988，ANSI C。

风格和符号规定

本文档应用了下列规定：

程序清单，程序样例，文件名，交互显示，每种都用不同字体显示。

程序清单和程序样例 —— courier New

文件名 —— COURIER NEW ITALIC,大写

交互式命令行 —— Courier New ，粗体

商标

MS-DOS 是微软的商标

UNIX 是 X/Open 的商标

IBM PC 是国际商业集团公司的商标。

附加协助

欲得到附加协助，请按下列方式联系我们：

Accelerated Technology 公司
720 Oak Circle Dr.E
Mobile，A1..36609
(800)-468-6853
(334)-661-5770
(334)-661-5788(fax)
mailto:support@atinucleus.com(技术支持Email)
http://www.atinucleuscom(WWW)

目录列表

第一章 介绍 ……. 1-1

1.1 关于 Nucleus PLUS

1.2 实时应用

1.3 为什么需要 Nucleus PLUS

第二章 开始 ……. 2-1

2.1 应用开发综述

2.2 安装 Nucleus PLUS

2.3 如何应用 Nucleus PLUS

2.4 应用程序初始化

2.5 目标系统需要考虑的事项

2.6 配置选项

第三章 功能描述 ……. 3-1

3.1 初始化

3.2 任务

3.3 任务通信

3.4 任务同步

3.5 定时器

3.6 内存管理

3.7 中断
3.8 输入/输出 驱动器
3.9 系统诊断
第四章 Nucleus PLUS 服务 …….. 4-1
4.1 任务控制服务
4.2 任务通信服务
4.3 任务同步服务
4.4 时钟服务
4.5 内存服务
4.6 中断服务
4.7 I/O 驱动服务
4.8 开发服务
4.9 服务定义
NU_Activate_HISR
NU_Allocate_Memory
NU_Allocate_Partition
NU_Broadcast_To_Mailbox
NU_Broadcast_To_Pipe
NU_Broadcast_To_Queue
NU_Change_Preemption
NU_Change_Priority
NU_Change_Time_Slice
NU_Check_Stack
NU_Control_Interrupts
NU_Control_Signals
NU_Control_Timer
NU_Create_Driver
NU_Create_Event_Group
NU_Create_HISR
NU_Create_MailBox
NU_Create_Memory_Pool
NU_Create_Partition_Pool
NU_Create_Pipe
NU_Create_Queue
NU_Create_Semaphore
NU_Create_Task
NU_Create_Timer
NU_Current_HISR_Pointer
NU_Current_Task_Pointer
NU_Deallocate_Memory
NU_Deallocate_Partion
NU_Delete_Driver
NU_Delete_Event_Group
NU_Delete_HISR

NU_Delete_MailBox
NU_Delete_Memory_Pool
NU_Delete_Partition_Pool
NU_Delete_Pipe
NU_Delete_Queue
NU_Delete_Semaphore
NU_Delete_Task
NU_Delete_Timer
NU_Disable_History_Saving
NU_Driver_Pointers
NU_Enable_History_Saving
NU_Established_Drivers
NU_Established_Event_Groups
NU_Established_HISRS
NU_Established_Mailboxes
NU_Established_Memory_Pools
NU_Established_Partition_Pools
NU_Established_Pipes
NU_Established_Queues
NU_Established_Semaphores
NU_Established_Tasks
NU_Established_Timers
NU_Event_Group_Information
NU_Event_Group_Pointers
NU_HISR_Information
NU_HISR_Pointers
NU_License_Information
NU_Local_Control_Interrupts
NU_MailBox_Information
NU_MailBox_Pointers
NU_Make_History_Entry
NU_Memory_Pool_Information
NU_Memory_Pool_Pointers
NU_Obtain_Semaphore
NU_Partition_Pool_Information
NU_Partition_Pool_Pointers
NU_Pipe_Information
NU_Pipe_Pointers
NU_Protect
NU_Queue_Information
NU_Queue_Pointers
NU_Receive_From_Mailbox
NU_Receive_From_Pipe
NU_Receive_from_Queue

NU_Receive_Signals
NU_Register_LISR
NU_Register_Signal_Handler
NU_Release_Information
NU_Release_Semaphore
NU_Relinquish
NU_Request_Driver
NU_Reset_Mailbox
NU_Reset_Pipe
NU_Reset_Queue
NU_Reset_Semaphore
NU_Reset_Task
NU_Reset_Timer
NU_Resume_Driver
NU_Resume_Task
NU_Resume_Clock
NU_Retrieve_Clock
NU_Retrieve_Events
NU_Retrieve_Hiistory_Entry
NU_Semaphore_Information
NU_Semaphore_Pointers
NU_Send_Signals
NU_Send_To_Front_Of_Pipe
NU_Send_To_Front_Of_Queue
NU_Send_To_Mailbox
NU_Send_To_Pipe
NU_Send_To_Queue
NU_Set_Clock
NU_Set_Events
NU_Setup_Vector
NU_Sleep
NU_Suspend_Driver
NU_Suspend_Task
NU_Task_Information
NU_Task_Pointers
NU_Terminate_Task
NU_Timer_Information
NU_Timer_Pointers
NU_Unprotect

第五章 扩展讨论 ……. 5-1

5.1 内存用法 …… 5-1
5.2 定制服务 ……. 5-5
5.3 执行线程 ……. 5-8
5.4 中断句柄 ……. 5-10

5.5 I/O 驱动 5-15
第六章 样例系统 6-1
6.1 例子总概 6-1
6.2 样例系统 6-2
附录 A .Nucleus PLUS 常量 A-1
Nucleus PLUS 常量(按字母顺序列表)
Nucleus PLUS 常量(按数字顺序列表)
附录 B .错误条件 B-1
附录 C .I/O 驱动器请求结构 C-1
附录 D .保存记忆的方法 D-1
数据初始化 D-1
NU_MAX_LISRS D-2
TC_PRIORITIES D-2
HISR 堆栈共享 D-2
TCD_Lowest_Set_B it D-2
应用一个更小的 INT 选项 D-3

第一章 介绍

本章提供 Nucleus PLUS 实时多任务内核的介绍。另外，本章提供实时概念的高级介绍。

章节段

1.1 关于 Nucleus PLUS

1.2 实时应用

1.3 为什么需要 Nucleus PLUS

1.1 关于 Nucleus PLUS

Nucleus PLUS 是为实时要求较高的嵌入式应用设计的实时、任务抢先式、多任务内核。大约 95％的 Nucleus PLUS 代码用 ANSI C 编写。正因为如此，Nucleus PLUS 非常轻便并且可以很容易的应用到大多数的微处理器家族。

Nucleus PLUS 通常作为一个 C 库文件实现。实时的 Nucleus PLUS 应用被链接到 Nucleus PLUS 库。目标文件可以下载到目标机，或者放到 ROM 里。在一个典型的目标环境，，假设所有服务被应用，Nucleus PLUS 指令集的二进制映象文件需要大概 20K 字节的内存。

Nucleus PLUS 通常以源代码方式交货。持有 Nucleus PLUS 源代码的访问权更好的促进了理解和允许特殊应用的改动。

1.2 实时应用

实时是什么？实时就是用于描述软件的一种方式，此软件对外部和内部的事件在正确的时间内必须作出正确的响应。实时可以分为硬实时和软实时。在软实时中，在正确的时间中作出正确的响应如果失败是令人不愉快的。象这样的失败如果在硬实时系统中出现是将是灾难性的。

今天实时应用通常对多种不同的任务和职责负责。比较有代表性的，任务有实现单个目标的，因此可以作为半独立的程序段执行。大多数应用包含硬实时和软实时任务。

1.3 为什么需要 Nucleus PLUS

由于任务重要性内在的差异，在任务间共享处理器的方法非常重要。简单的实时应用和通常软实时更多的应用，可以在应用任务中嵌入处理器分配逻辑。这种典型地执行采用了控制环的形式，连续检查任务运行。这种技术遭遇了下面的问题：

低速的**响应时间**(Response Time) — 检测危急事件的最坏的情况时间在最坏情况发生期间。

修正难点 — 从处理器分配逻辑分散贯穿应用程序代码，每个任务运行所需的时间依赖于其他任务的处理时间。因此，单个任务的代码改变可能导致整个系统的失败。

减少的吞吐量 — 随着任务数量的增加，查找运行任务的时间浪费增加。这些时间可以更好的用来干一些有意义的事情。

困难的软件开发 — 比较有代表性的，这种类型的应用程序之间有很许多需要相互依赖的地方，使得多个工程师之间的合作变得更加困难。另外，把这些应用的程序移至到其他处理器上变得更加困难。

Nucleus PLUS 排除了在应用程序软件中对处理器分配的需要。当一个更重要的任务需要运行的时候，Nucleus PLUS 挂起当前运行的任务并开始高优先级的任务。在高优先级任务完成之后，挂起的任务恢复。在 Nucleus PLUS 下最坏任务的响应时间等于挂起的运行任务和恢复更重要任务所需时间的总和。Nucleus PLUS 提供快速和恒

定的响应时间。正因为如此，修改和新任务的完全添加可以在不影响临界系统响应需要的情况下实现。

除管理任务运行之外，Nucleus PLUS 也提供包括任务通信、任务同步、时钟和内存管理的方法工具。

从软件开发的立足点来看，Nucleus PLUS 鼓励较少任务之间的依赖性和更多大的模块化性能。正因如此，大多数的工程师可以在不用担心出现在非 Nucleus PLUS 应用中出现的‘边缘效应’(side-effects)的情况下工作。Nucleus PLUS 也提供了运行时环境，它可以完全独立于目标处理器。这样在两个方面有利于开发：第一，工程师可以集中精神在运行时应用程序上而不必关注复杂的处理器底层。第二，工程师可以开发在大多数通用的微处理器上运行的应用程序。

总的来说，Nucleus PLUS 极大的提高了实时应用程序的开发工作。这些可以转变成更低的开发投资和更短的开发周期。自从 Nucleus PLUS 支持应用程序移值到新的处理器系列上以来，应用的开发投资已经被保障了。

第二章 开始

本章描述了软件开发过程，Nucleus PLUS 的安装和应用程序在 Nucleus PLUS 上的运行。本章也讨论了目标系统对 Nucleus PLUS 应用程序和不同 Nucleus PLUS 配置选项的需求。请看 readme.me 了解安装和应用 Nucleus PLUS 的细节。

章节段

2.1 应用开发综述
2.2 安装 Nucleus PLUS
2.3 如何应用 Nucleus PLUS
2.4 应用程序初始化
2.5 目标系统考虑事项
2.6 配置选项

2.1 应用开发综述

嵌入式实时应用程序的开发基于宿主机系统。IBM PC 和 UNIX 工作站都是很好的宿主机系统。应用程序软件通常运行在分离的计算机系统上，通常被目标系统调用。然而，IBM PC 不遵循此规则。它既可以为 Nucleus PLUS 应用程序主机也可为目标机服务。应用程序可以在 IBM PC 上以 EXE 文件的方式运行。

建立一个嵌入式实时应用程序是非常直接的。驻留在主机系统上的应用程序文件可以编译/汇编成目标文件并连接。结果映像文件既可以下载到目标系统也可以放到目标系统的 ROM 中。

针对目标系统的调试软件通常包括 ICE 仿真工具和 TRM（目标仿真）工具。拥有 ICE 工具是更好的选择，因为 ICE 工具给工程师完全控制和了解目标系统硬件状况。ICE 工具在校验新硬件时尤其有用。考虑到费用问题和 ICE 有时有局限性，许多项目采用 TRM 调试。TRM 就是一个小型的运行在目标系统（通常为 ROM）上的软件组件。TRM 提供包括下载、下断点和内存入口服务。ICE 和 TRM 都有宿主系统控制。这通常由串口来完成。

源级调试允许工程师使用真正的 C 源代码调试应用程序。这种性能需要宿主系统上附加的程序，该程序在 C 源代码和目标系统内存之间建立联系。大多数的源代码级调试使用 ICE 和 TRM 来真正控制和进入目标系统硬件。

Nucleus PLUS 整合了大量的 C 源代码调试器。另外，Nucleus PLUS 调试器对为 Nucleus PLUS 应用程序附加扩展的多任务调试能力有效。

2.2 安装 Nucleus PLUS

Nucleus PLUS 存放在双面高密 MS－DOS 兼容磁盘上。全部 Nucleus PLUS 系统要求主机有大约 2M 的磁盘空间。

Nucleus PLUS 在主机系统上的安装很简单。自从 Nucleus PLUS 可以在许多主机系统上运行以来，下列安装步骤为非主机指定方式而写：

1) 备份所有产品资料盘；
2) 在主机系统上创建名为 NUCLEUS 的目录；
3) 从每张盘拷贝所有文件至 NUCLEUS 目录；
4) 通过运行批处理文件 BUILD LI.BAT 建立 Nucleus PLUS 库；
5) Nucleus 库文件 NUCLEUS.LIB 必须与连接器更易接近。这可能要通过既要在应用程序开发目录中拷贝它又要在 NUCLEUS 目录中设置连接器以至于能找到它来完成。

6) NUCLEUS.H 文件必须贴近应用程序。这可能要通过既要在应用程序开发目录中拷贝它又要在 NUCLEUS 目录中设置编译器器以至于能找到它来完成。

Nucleus PLUS 移植笔记文档包含不同种类的处理器和详细开发问题。在安装阶段回顾这些文档。

2.3 如何使用 Nucleus PLUS

Nucleus PLUS 被设计成 C 库使用。使用内部应用程序软件的服务从 Nucleus PLUS 库文件取出并且组合成应用程序目标文件来生成最终的映像文件。此映像文件可以下载到目标系统或是存放到目标系统 ROM 中。

使用 Nucleus PLUS 的步骤被描述成下列通用的方式：

1) 如果有必要，修改低级系统初始化文件，INT.S。注：这些文件通常以汇编语言形式交货并且它的的扩展是指定的开发工具。
2) 定义 Application_Initialize 函数，Nucleus PLUS 启动系统时它优先运行。注意 NUCLEUS.H 文件必须包含以至于能被 Nucleus PLUS 服务调用。
3) 定义应用程序任务。如果用到 Nucleus PLUS 服务，文件 NULCUES.H 必须被包含。
4) 编译 和/或 汇编所有应用程序软件，包括低级系统初始化文件 INT.s。
5) 用 Nucleus PLUS 库和必要的开发工具库连接 INT.a 和所有应用程序目标文件。
6) 下载应用程序映像文件到目标系统并让它跑起来！

请回顾处理器和开发系统文档附加信息，包括如何使用编译器、汇编器和连接器。

2.4 应用程序初始化

Application_Initialize 子程序负责定义初始化应用程序环境。它包含有任务、邮箱、队列、信号量、事件集、内存池和其他 Nucleus PLUS 对象。**Application_Initialize** 配备有指向第一个有效地内存地址的指针。之后的内存没有被编译器或 Nucleus PLUS 使用，因此对应用程序有效。虽然 **Application_Initialize** 详细内容依靠应用程序，下面的模板依然有效：

```
# include <nucleus.h>
void   Application_Initialize
                    (void *first_available_memory)
{
    /* Nucleus PLUS对象的应用程序详细初始化，包括任务、邮箱、队列、
    管道、事件集、内存池的创建。*/
}
```

从初始化子程序调用的服务不能被挂起，直到初始化子程序不再作为一个任务运行。也要注意至少一个任务或是中断处理器被 Application_Initialize 创建，并且 Application_Initialize 是优先级高于第一个任务运行的最后一个子程序。

下面是 Application_Initialize 创建一个内存池和一个任务的例子。注意本例不检测任何错误条件。

```
#include <nucleus.h>
/*定义任务控制结构*/
NU_TASK Task;
```

```
/*定义动态内存池控制结构*/
NU_MEMORY_POOL Memory_Pool;
Void Application_Initialize
                (void *first_available_memeory)
{
  void *stack_ptr;
  /*创建一个 4000 字节开始于起始地址的动态内存池*/
  NU_Create_Memory_Pool(&Memory_Pool,"SYSTEM",
       first_available_memory,4000,50,NU_FIFO);
  /*从系统内存池创建一个任务堆栈*/
 NU_Allocate_Memory(&Memory Pool,
                    &stack_ptr,500,NU_NO_SUSPEND);
/*创建一个以功能函数 abc(0,NU_NULL)为入口点的应用任务*/
    NU_Create_Task(&Task,ABC_TASK,abc,0,NU_NULL,stack_pt
    r,500,10,NU_PREEMPT,NU_START);
```

2.5 目标系统考虑事项

Nucleus PLUS 指令集尺寸在 CISC 结构中小于 20KB，RISC 架构中小于 40K。作为数据架构，Nucleus PLUS 最小需要 1.5KbRAM。这些还不包括应用任务、队列、管道和其他 Nucleus PLUS 对象所需的内存空间。

如果不修改 Nucleus PLUS 任何预设数据元素，它很容易被装入 ROM 里面。加之 Nucleus PLUS 兼容每种开发环境有效的 ROM 配置。

如果目标系统包含一个 TRM，Nucleus PLUS 应用程序必须载入到 TRM 使用的内存区中。另外，应用程序必须只带需要的中断向量，直到 TRM 使用中断向量运行断点和其他函数。

2.6 配置选项

Nucleus PLUS 应用程序有一个有条件的编辑选项。在命令行编译应用程序源文件通过定义 NU_NO_ERROR_CHECKING 变量，所有 Nucleus PLUS 服务错误检测逻辑被忽略（Bypassed）。这可以提升 Nucleus PLUS 服务运行性能。

当建立 Nucleus PLUS 库时，有几个有条件的编辑选项被有效应用。这些选项可以加到 MAKELIB.BAT 文件的编译命令里。这些文件驻留在 NUCLEUS 主文件夹里，包含所有建立 Nucleus 库所必须的命令。下面是对有条件的有效编译符号和他们对应的含义的定义：

编译符号	含义
NU_ENABLE_HISTROY	每个服务调用历史条目结果。这个符号可以被加到所有**C.C 文件的编译命令当中
NU_ENABLE_STACK_CHECK	使能堆栈检测。这个符号可以被加到所有**C.C 文件的编译命令当中
NU_ERROR_STRING	如果有致命系统错误发生建立一个 ASCII 错误信息。本选型只用于编译 ERD.C,ERI.C 和 ERC.C 时。
NU_NO_ERROR_CHECKING	忽略错误检测，和任何用户的应用程序代码

第三章　功能的描述

本章提供 Nucleus PLUS 设备的功能综述。第四章详细描述每个 Nucleus PLUS 服务。章节段

3.1　初始化
3.2　任务
3.3　任务通信
3.4　任务同步
3.5　定时器
3.6　内存管理
3.7　中断
3.8　输入/输出驱动器
3.9　系统诊断

3.1　初始化

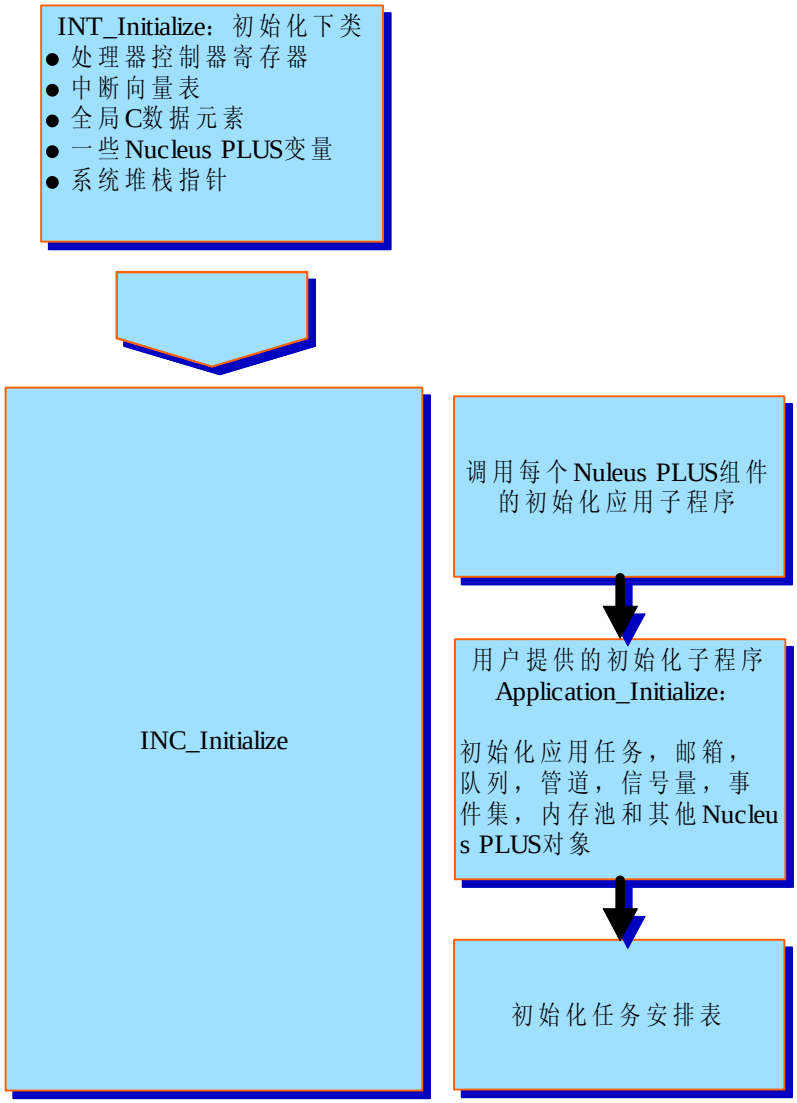
INT_Initialize 子程序在 Nucleus PLUS 系统中是最先运行的。对大多数的目标环境，硬件复位向量必须包含在 INT_Initialize 地址。

INT_Initialize 对所有的由目标决定（target-dependent）进行的初始化负责。由目标决定的初始化通常包括设置不同种类的处理器控制寄存器，中断向量表，全局的 C 数据元素，一些 Nucleus PLUS 变量，和系统堆栈指针。当 INT_Initialize 完成，控制转移到高级 Nucleus PLUS 初始化子程序 INC_Initialize 上。注意控制不会返回 INT_Initialize。

INC_Initialize 调用每个 Nucleus PLUS 组件的初始化子程序。在所有 Nucleus PLUS 初始化完成之后，INC_Initialize 调用用户供应的初始化子程序 Application_Initialize。

Application_Initialize 子程序负责定义初始化应用环境。初始化应用任务，邮箱，队列，管道，信号量，事件集，内存池和其他 Nucleus PLUS 对象都在子程序中被定义，这个子程序的格式在《开始》章节描述。

在 Application_Initialize 返回后，INC_Initialize 开始初始化任务调度表。



3.2 任务

任务就是目的明确的半独立程序段。大多数现代实时应用都要求多任务。另外，这些任务的重要等级经常变化。管理这些竞争、实时任务的运行是 Nucleus PLUS 的主要目的。

任务状态

每个任务都有五种状态：运行、就绪、挂起、中止、完成。下列列表描述了每个任务的状态：

状态	含义
运行	任务正在运行
就绪	任务就绪，但其他任务正在运行
挂起	等待服务请求完成之前任务休眠状态。当响应结束，任务转入等待状态
中止	任务被禁止。一旦进入这种状态，任务直到复位之前不能运行。
完成	任务完成并返回到初始入口子程序。一旦进入这种状态，任务直到复位前都不能运行
优先级	用户分配，范围 0～255，用来定义 Nulceus PLUS 任务的重要性，数字越小优先级越高

任务抢先

任务抢先就是当更高优先级的任务就绪时挂起低优先级的任务的行为。例如，假设一个优先级为 100 的任务正在运行。如果一个任务优先级为 20 的中断发生，优先级为 20 的任务在中断任务恢复前被运行。**任务抢先**也发生在低优先级任务调用 Nucleus PLUS 服务导致更高优先级任务就绪时。

任务抢先可以在个别任务基础上被禁止。当**任务抢先**禁止时，没有任务允许执行，直到运行的任务挂起，放弃控制或者使能**任务抢先**模式。无论任务抢先使能还是禁止，任务可以创建，在任务运行期间任务抢先可以被使能或禁止。

放弃

一种机制以类似圆知更鸟（知更鸟什么玩意？？？）方式被提供，用来解决一个任务和其他处于同一优先级且就绪运行的任务之间，任务共享处理器的问题。当一个任务请求这种服务时，所有其他具有同一优先级且就绪运行的任务在先前运行的任务恢复前运行。

时间片

时间片提供了处于同一优先级的任务共享处理器的另一种方案。符合最大定时器节拍（定时器中断）的时间片，可能在其他所有就绪且在同一优先级的任务有机会运行之前发生。时间片行为就好像一个没有请求的任务被放弃了。注意禁止任务抢先也可以禁止时间片。

动态创建

Nucleus PLUS 任务可以被动态创建或删除。一个应用程序所支持的任务数没有预设限制。每个任务请求控制块和堆栈。每个元素的内存由应用程序提供。

结论

挂起和恢复任务的处理时间是恒定的。它不受任务数的影响。另外，每个任务运行的方式不仅可以预料，也能保证。高优先级已就绪任务在低优先级已就绪任务之前运行。同等优先级就绪的任务按照就绪顺序执行。

堆栈检测

应用任务可以检测当前堆栈剩余的内存数量。这个功能也可以对最大堆栈使用保持跟踪。堆栈检测也可以禁止在 Nucleus PLUS 服务内部通过有条件的编译选项。

任务信息

应用任务可以获得激活任务的列表。每个任务的详细信息一个可以获得。这些信息包括任务名、当前状态、预定计数、优先级和堆栈参数。

优先级

任务优先级在任务创建期间定义。另外，支持动态任务优先级的更改。0 优先级的任务高于 255 优先级任务。Nucleus PLUS 在低优先级任务前运行更高优先级任务。同等优先级任务按照进入就绪状态的先后顺序运行。

注：给任务分配优先级时必须小心。如果一不小心，优先级可能导致任务饿死和系统超负荷。

一个任务只有在高优先级，就绪状态下才能运行。因此，处于同一优先级的任务且都进入就绪状态，所有低级任务不会运行。这种情况叫做“饿死”。有几种方法解决这个问题。第一，高优先级任务必须挂起给低优先级任务一个运行的机会。连续运行的任务必须有一个想过的低优先级。另一种解决“饿死”的技术就是逐渐提升饿死任务的优先级。

如果任务优先级运用的不正确的话，会招致大量的附加消耗。考虑一下带三个任务 A，B，C 的系统。每个任务由同样的处理过程构成，在无限循环中等待/发送信息。任务 A 从中断服务子程序（ISR）等待信息然后发送信息到 B。任务 B 从任务 A 中等待信息然后发送到任务 C，任务 C 等待任务 B 的信息然后发送计数器递增一位。在这个简单系统开始运行（不管优先级），所有大额任务暂时运行，然后挂起等待信息。

如果所有的任务由相同的优先级，下列一套事件在 ISR 发送消息给 A 后发生：

任务 A 恢复
任务 A 发送消息给任务 B，使任务 B 进入就绪状态
任务 A 挂起等待其他信息
任务 B 恢复
任务 B 发送消息给任务 C，使任务 C 进入就绪状态
任务 B 挂起等待其他信息
任务 C 恢复
任务 C 给计数器递增
任务 C 挂起等待其他信息

现在假设任务 A 优先级低于任务 B，B 低于 C。下列事件在 ISR 发送消息给任务 A 后发生：

任务 A 恢复
任务 A 发送消息给任务 B，使任务 B 进入就绪状态
任务 A 放弃机会给高优先级任务 B
任务 B 恢复
任务 B 发送消息给任务 C，使任务 C 进入就绪状态
任务 B 放弃机会给高优先级任务 C
任务 C 恢复
任务 C 给计数器递增
任务 C 挂起等待其他信息
任务 A 再次恢复
任务 A 等待其他消息

应用程序在前一个的例子中运行是相同的，例如，两个任务发送消息，三个任务等待消息。然而在恢复和挂起任务中系统消耗是双倍的。也应该注意到在最后一个例子中任务 A 在发送消息和等待其他消息之间有延迟发生。

很明显，前一个例子系统只对优先级影响系统耗费做了说明。对响应外部事件的实时应用和分配处理事件到相关的更重要的任务的实时应用，不同的优先级是有必要的。然而，为了减少不必的系统消耗，应用程序中不同的优先级数应该最小化。

3.3 任务通信

Nucleus PLUS 为通信目的提供邮箱(mailbox),队列(queues),管道(pipes)。邮箱，队列，管道是独立的公共设备。任务之间和其他系统设备之间的联系由应用程序确定。这些通信设备之间主要的差别是数据通信的类型。

3.3.1 邮箱

邮箱为传输简单数据提供低消耗方案。每个邮箱可以保持 4 个 32 位字大小的单一消息。消息以（数）值方式发送和接受。一个发送消息要求拷贝消息到邮箱，一个接受消息要求从邮箱中把消息拷贝出来。

挂起

发送和接收邮箱服务为非条件的挂起、时间间隙挂起和无挂起提供配置。

任务在邮箱内可能因为几种原因挂起。一个试图从空邮箱中接受消息的任务可以被挂起。同样，一个试图发送消息至满邮箱的任务可以被挂起。当邮箱能够确保任务请求时挂起的任务恢复。例如，假设一个任务在邮箱等待接受消息时挂起。当一个消息发送到邮箱时，挂起的任务就恢复了。

多任务能在一个邮箱上挂起。依靠创建邮箱的任务可以以 FIFO 或是优先级次序被挂起。如果邮箱支持 FIFO 挂起，任务按他们挂起的顺序恢复。另外，如果邮箱支持优先级挂起，任务从高优先级到低优先级顺序恢复。

广播

邮箱的消息可以被广播。这种服务类似于发送请求，除了所有从邮箱等待消息的任务改成了等待广播消息。

动态创建

Nucleus PLUS 邮箱可以动态创建和删除。一个应用程序在邮箱数量上没有预先限制。每个邮箱需要一个控制块。控制块的内存由应用程序提供。

结论

发送和接受邮箱消息的处理时间请求为常量。然而，按优先级顺序挂起任务所需的处理时间受当前在邮箱上挂起的任务数影响。

邮箱信息

应用程序任务可以获得活动邮箱的列表。每个邮箱的详细信息也也可以获得。这些信息包括邮箱名、挂起类型、信息是否出现、第一个任务等待。

3.3.2 队列

队列提供了传输多个消息的机制。消息以数值形式发送接受。一个发送消息要求拷贝消息到队列，一个接收消息要求从队列上拷贝消息。消息可以放在对列的前端或队列的后端。

消息尺寸

一列消息包括一个或多个 32 位字。固定的和长度可变的消息都支持。消息类型的格式在队列创建的时候定义。长度可变的消息队列需要为队列中的每个消息消耗一个附加的 32 位字。另外，接收消息要求在长度可变消息队列指定最大消息尺寸，在固定长度消息队列指定合适的消息尺寸上有同样的要求。

挂起

发送和接收队列服务为非条件的挂起、时间间隙挂起和无挂起提供配置。

任务能在队列内因为几种原因挂起。一个试图从空队列中接受消息的任务可以被挂起。同样，一个试图发送消息至满队列的任务可以被挂起。当队列能够确保任务请求时挂起的任务恢复。例如，假设一个任务在队列等待接受消息时挂起。当一个消息发送到队列时，挂起的任务就恢复了。

多任务能在一个队列上挂起。依靠创建队列的任务可以以 FIFO 或是优先级次序被挂起。如果队列支持 FIFO 挂起，任务按他们挂起的顺序恢复。另外，如果队列支持优先级挂起，任务从高优先级到低优先级顺序恢复。

广播

队列的消息可以被广播。这种服务类似于发送请求，除了所有从队列等待消息的任务改成了等待广播消息。

动态创建

Nucleus PLUS 队列可以动态创建和删除。一个应用程序在队列数量上没有预先限制。每个队列需要一个控制块。控制块的内存由应用程序提供。

结论

发送和接受队列消息请求的基本处理时间为常量。然而，按优先级顺序挂起任务所需的处理时间受当前在队列上挂起的任务数影响。

队列信息

应用程序任务可以获得活动队列的列表。每个队列的详细信息也也可以获得。这些信息

包括队列名、消息类型、挂起类型、消息出现的次数、第一个任务等待。

3.3.3 管道

管道提供了传输多个消息的机制。消息以数值形式发送接受。一个发送消息要求拷贝消息到管道，一个接收消息要求从管道上拷贝消息。消息可以放在管道的前端或队列的后端。

消息尺寸

一列管道消息包括一个或多个 32 位字。固定的和长度可变的消息都支持。消息类型的格式在管道创建的时候定义。长度可变的消息管道需要为管道中的每个消息消耗一个附加的 32 位字。另外，接收消息要求在长度可变消息管道指定最大消息尺寸，在固定长度消息管道指定合适的消息尺寸上有同样的要求。

挂起

发送和接收管道服务为非条件的挂起、时间间隙挂起和无挂起提供配置。

任务能在管道内因为几种原因挂起。一个试图从空管道中接受消息的任务可以被挂起。同样，一个试图发送消息至满管道的任务可以被挂起。当管道能够确保任务请求时挂起的任务恢复。例如，假设一个任务在管道等待接受消息时挂起。当一个消息发送到挂起时，挂起的任务就恢复了。

多任务能在一个管道上挂起。依靠创建管道的任务可以以 FIFO 或是优先级次序被挂起。如果管道支持 FIFO 挂起，任务按他们挂起的顺序恢复。另外，如果管道支持优先级挂起，任务从高优先级到低优先级顺序恢复。

广播

管道消息可以被广播。这种服务类似于发送请求，只是所有从队列等待消息的任务改成了等待广播消息。

动态创建

Nucleus PLUS 管道可以动态创建和删除。一个应用程序在管道数量上没有预先限制。每个管道需要一个控制块和一个管道数据区。每个内存由应用程序提供。

结论

发送和接受管道消息请求的基本处理时间为常量。然而，拷贝消息所需的时间与消息尺寸有关联。另外，按优先级顺序挂起任务所需的处理时间受当前在管道上挂起的任务数影响。

队列信息

应用程序任务可以获得活动管道的列表。每个管道的详细信息也可以获得。这些信息包括管道名、消息格式、挂起类型、消息出现的次数、第一个任务等待。

3.4 任务同步

Nucleus PLUS 提供信号量（semaphores），事件集（event groups）和信号（signals）解决信号同步问题。信号量和事件集都是独立的，公用的设备。任务和其他系统设备的联系由应用程序决定。信号（signals），换一种说法，与指定任务关联。

3.4.1 信号量（semaphores）

信号量提供了控制应用程序临界区运行的机制。Nucleus PLUS 提供了范围从 0～4294967294 的计算信号量。信号量两个基本操作是*获得*和*释放*。获得信号量请求消耗信号量，释放信号量请求增加了信号量。

信号量最普通的应用是资源配置。另外，带初始值信号量的创建可以用来指示事件。

挂起

获得信号量服务为无条件挂起、时间间隙挂起、无挂起提供配置。

一个的试图获得当前计数值为零信号量的任务可以被挂起。当释放信号量请求发生时，任务恢复是可能的。

多任务可以挂起，试图获得一个信号量。依靠信号量创建方式，任务既可以 FIFO 顺序也可以优先级顺序挂起。如果信号量支持 FIFO 挂起，任务按照它们试图获得信号量的顺序恢复。另外，如果信号量支持优先级挂起，任务从高优先级到低优先级顺序恢复。

死锁

两个任务或多个任务永远挂起试图获得两个或更多个信号量，死锁就涉及到这种状况。出现这种状况最简单的例子就是一个有两个任务和两个信号量的系统。假设第一个任务配第二个信号量和第二个任务配第一个信号量。现在假设第二个任务试图获得第二个信号量而第一个任务试图获得第一个信号量。一旦每个任务拥有其他任务所需的信号量，任务可能因为信号量永远挂起。

预防是处理死锁的最佳措施。这项技术把规则强加到应用程序使用的变量。例如，如果任务不允许一次占用超过一个信号量，死锁被保护。作为选择，如果任务在同一个次序获得多个信号量死锁可以被保护。获得信号量挂起时可选的空闲时间（Timeout）可以用来从死锁状态恢复。

优先级倒置

优先级倒置在高优先级任务请求低优先级任务使用的信号量引起挂起时发生。如果不同优先级任务共享相同的受保护的资源，这种情况不可避免。在这种情况下，在优先级倒置里，有限和可预料的时间量可接受。

然而，如果在优先级倒置状态，低优先级任务被中优先级任务占先，优先级倒置的时间量不确定。通过确保说有的使用相同的信号量任务有相同的优先级可以避免这种情况发生，至少在他们占用信号量的时候。

动态创建

Nucleus PLUS 信号量可以动态被创建和删除。应用程序可能拥有的信号量数没有预先限定。每个信号量需要一个控制块。控制块的内存由应用程序提供。信号量在创建时赋初值。

结论

获得和释放信号量要求的运行时间为常量。然而，以优先级顺序挂起一个任务所需的运行时间受当前在信号量上挂起的任务数影响。

信号量信息

应用程序任务可以获得激活的信号量列表。每个信号量的详细信息也可以得到。这些信息包括：信号量名、当前值、挂起类型、任务等待数、第一个等待任务。

3.4.2 事件集

事件集提供一个机制来描述一个指定系统事件的发生。事件由事件集中单个位来描述。这位叫事件标志。每个事件集有 32 个事件标志。

事件标志可以通过逻辑 AND/OR 组合被设置和清除。事件标志也可以以逻辑 AND/OR 组合接收。另外，事件标志可以在接收完后自动复位。

挂起

接收事件标志请求为无条件挂起、时间间隙挂起、无挂起提供选项。

试图接收一组没有出现的事件标志的任务，可以被挂起。当置位事件标志操作满足任务中事件请求组合时任务恢复。

动态创建

Nucleus PLUS 事件集可以动态创建和删除。应用程序可能拥有的事件集数没有预先

限定。每个事件集需要一个控制块。控制块的内存由应用程序提供。

结论

获得和释放信号量要求的运行时间为常量。然而，在事件集中置位事件标志所需的时间受事件集上挂起的任务数影响。

事件集信息

应用程序任务可以获得激活的事件集列表。每个事件集的详细信息也可以得到。这些信息包括：事件集名、当前事件标志、任务等待数、第一个等待任务。

信号（signals）

信号在某种程度上讲很相似。然而，他们在操作上有几个非常重要的差别。事件标志的用法天生就是同步的。直到指定的服务请求完成，任务不承认事件标志出现。信号以异步的方式运行。当信号出现，任务中断并且任务提前指定的信号处理子程序运行。每个任务可以处理 32 个信号。每个信号对应一个描述位。

信号处理子程序

任务信号处理子程序必须在任何信号运行之前被提供。信号处理子程序中的处理实际上和高级中断子程序有同样的强制性。倘若自挂起被禁止，基本上大多数 Nucleus PLUS 服务都是可以获得的。

使能信号子程序

任务缺省可以在所有信号禁止的情况下创建。个别信号可以被每个任务动态使能或禁止。

清除信号

当信号处理被调用信号自动清除。另外，当请求接收信号的恳求完成时信号被清除。注：在请求接收信号恳求时，任务不能挂起。

多信号

一旦信号处理子程序开始运行，任务的信号被清除。信号处理子程序不能被新信号中断。任何新信号的处理在当前信号处理完成后进行。在第一个信号被验证之前发送的同样的信号被放弃。

结论

至少在最坏的情况下，发送和接收信号要求的运行时间是常量。当然运行信号处理子程序所需的时间由应用程序指定。

3.5 定时器

大多数实时应用需要在按周期性的时间间隔运行。每个 Nucleus PLUS 任务都有一个内建定时器。这个定时器用来提供任务休眠和服务调用的时间。

节拍

节拍是所有 NucleusPlUS 定时器设备时间的基本单元。每一拍对应单个硬件定时器中断。实际的时钟节拍值是用户可编程的。

错误空白

大约一拍左右可以满足一个定时器请求。这是因为在定时器请求之后一节拍可以立即发生。因此，定时器请求的第一拍表示的是从零到硬件定时器中断率范围的真实时间。例如，在实际时间 n 和 n－l 拍之间一个 N 拍的下降请求实际时间值中止。

硬件请求

Nucleus PLUS 定时器服务需要硬件提供的周期性的定时器中断。没有中断，定时器设备不运行。然而，其他的 Nucleus PLUS 设备在没有定时器设备时不会激活。

连续时钟

Nucleus PLUS 维持一个连续的技术节拍时钟。这个时钟的最大值为 4294967294。

时钟在到达节拍大值后自动复位。
这个连续时钟为应用程序的使用专门保留。它可以在若任何时间由应用程序读出或写入。

3.5.2 任务时钟

每个任务都有一个内建定时器。这个定时器为任务休眠请求和挂起时间间隙请求而准备。

3.5.3 应用时钟

Nucleus PLUS 为应用程序提供可编程定时器。这些定时器在他们到时时运行指定的用户提供子程序。用户提供时间到子程序作为一个高级中断服务子程序运行。因此，自挂起请求被禁止。另外，运行必须保持最小化。

定时器重新置初值

当一个定时器定时时间到时，指定的定时时间到时子程序开始运行。在运行结束之后，定时器既是静止的又可以重新置初始值。如果定时器置初始值为零，在初始化到时后还是禁止的。然而，如果定时器置初始值为非零，在时间间隔结束后重新置初值。

使能/禁止

应用程序定时器在创建期间可以被自动使能。另外，定时器可以被动态使能和禁止。

复位

初始化一个定时器的节拍、重新置初值的比率和一个定时器到时子程序可以被应用程序动态复位。

动态创建

Nucleus PLUS 应用程序定时器可以动态创建和删除。应用程序拥有的定时器数的没有预先限定。每个定时器请求一个控制块。控制快的内存由应用程序提供。

结论

创建、使能、禁止和修改应用程序定时器所需的处理时间是不变的。然而，运行用户提供的定时子程序所需的处理时间由定时子程序自身和同时结束的定时器数决定。

定时器信息

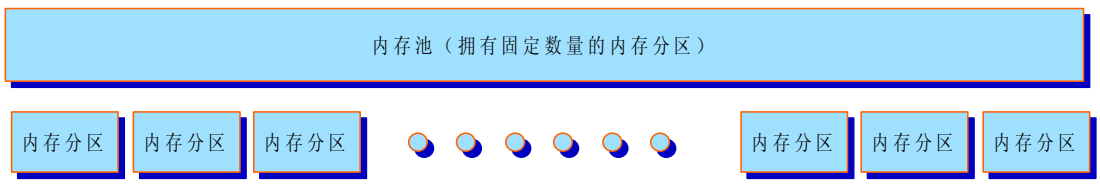
应用程序可以获得激活的定时器列表。每个定时器的详细信息也可以获得。这些信息包括定时器名称、状态、初始化节拍、重置值、保持节拍和到时计数。

3.6 内存管理

Nucleus PLUS 提供分区和动态内存管理设备。分区内存管理是确定性的但不是非常灵活。动态内存管理非常灵活但是有不确定性。大多数应用程序对两种类型的内存管理设备都有需要。

3.6.1 分区内存池

一个分区内存池包含一个指定固定尺寸的内存分区数。池的内存位置、池的字节数和在每个分区中的字节数都由应用程序决定。单个的分区从分区内存池中分配和收回。



从内存池的分配需要一些附加消耗来允许创建它的指针。看 4.10.2 章节，Nucleus PLUS 内部手册的分区内存数据结构在分区内存池头结构子章节之后，做了非常详细的阐述。

挂起

分配分区服务为无条件挂起、带时间间隙挂起和无挂起提供配置。

一个试图从一个空的池中分配一个区的任务可以被挂起。当分区返回到池中时任务才可能恢复。

一个分区内存池上的多任务可以被挂起。依据分区内存池的创建，任务可以以 FIFO 或是优先级顺序被挂起。如果分区内存池支持 FIFO 挂起，任务可以按照他们挂起的顺序恢复。同样，如果分区内存池支持优先级挂起，任务按照从高优先级到低优先级顺序恢复。

动态创建

Nucleus PLUS 分区内存池可以动态创建和删除。应用程序中拥有的分区内存池数量没有预先限定。每个分区内存池需要一个控制块和一个分区的内存区指针。控制块和分区的内存由应用程序提供。

结论

自从搜索完全被禁止以来，分配和释放分区所需的处理是快速和恒定的。然而，以优先级顺序挂起任务所需的处理时间受荡起在分区内存池上挂起的任务数影响。

分区信息

应用程序可以获得激活的分区内存池的列表。每个分区的详细信息也可以获得。这些信息包括分区内存池名称、起始池地址、总分区、分区大小、保持分区、挂起任务数、第一个挂起任务的身份。

3.6.2 动态内存池

一个动态内存池包含一个用户指定的字节数。内存在池中的位置由应用程序决定。

Nucleus PLUS 为动态内存池提供可变长度的分配和释放服务。分配以最适合的方式（first_fit_manner）运行。例如，满足要求的第一个有效内存被分配。如果分配的块比要求的大的多，没有使用的内存返回到动态内存池。先前释放的块在分配搜索期间又从新合并了。

从内存池分配要求附加消耗来允许它的指针结构。看章节 4.11.2 的 Nucleus PLUS 动态内存数据结构内部手册，在动态内存池头结构子章节之后，做了详细阐述。

挂起

分配动态内存服务为无条件挂起、延时挂起和无挂起提供配置。

一个试图从当前没有足够有效内存的池中分配动态内存的任务会被挂起。当有足够的先前分配内存返回池中时，任务的恢复变得可能。

在单个动态内存池上的多任务会被挂起。依据动态内存池的创建方式，任务即可以 FIFO 顺序挂起也可以优先级顺序挂起。如果动态内存池支持 FIFO 挂起，任务以他们被挂起的顺序恢复。另外，如果动态内存池支持优先级挂起，任务从高优先级到低优先级顺序恢复。

动态创建

Nucleus PLUS 动态内存池可以动态创建和删除。应用程序拥有的动态内存池数没有预先限定。每个动态内存池需要一个控制块和一个指向实际动态内存区的指针。控制块和内存区的内存由应用程序提供。

结论

从一个动态内存池分配内存是天生具有不确定性。很大程度上是因为池中可能存在的内存碎片。最适合（first-fit）的运算法则主要是线性搜索，结果最坏的情况就是存在大量的碎片。然而，内存的存储单元分配是不变的。以优先级顺序挂起的任务所需的

处理时间受当前动态内存池挂起任务数量的影响。

动态内存池信息

应用程序任务可以获得激活的动态内存持列表。每个动态内存池列表的详细信息也可以获得。这些信息包括动态内存池名、池起始地址、总尺寸大小、空闲字节数、任务挂起数、第一个挂起任务的特性。

3.7 中断

中断是为外部和内部事件提供立即响应的机制。当中断发生时，处理器挂起当前运行的程序，并且转移控制权到适当的中断服务子程序（ISR）。中断的强制运行是固有的、处理器指定的。

保护

中断对所有的实时内核造成了一个有趣的问题。Nucleus PLUS 也不例外。主要问题从中断服务子程序需要访问 Nucleus PLUS 这个事实滋生。表面上看好像不像是一个问题。然而，在被一个 ISR 同步访问的服务调用期间操作的数据结构需要得到保护。最简单的保护方法就是把服务期间的中断关在外面（关中断）。中断的快速响应是实时系统的基础。因此，关中断来保护数据结构不是个好办法。Nucleus PLUS 通过把应用程序的 ISRs 区分为低级到高级组件来处理保护问题。

低级中断

低级中断服务子程序（LISR）和正常的 ISR 一样运行，包括使用当前堆栈。Nucleus PLUS 在调用 LISR 之前保存上下文，在 LISR 返回之后恢复上下文。因此，LISRs 可以用 C 来写，也可以被其他的 C 子程序调用。然而，只有很少的一部分 Nucleus PLUS 服务可以被 LISR 访问。如果中断处理需要附加的 nucleus PLUS 服务，一个更高级的中断服务处理子程序（HISR）必须被激活。Nucleus PLUS 支持多个 LISRs 的嵌套。

高级中断

高级中断支持动态创建和删除。每个 HISR 由它自己的堆栈空间和控制块。每个的内存由应用程序提供。当然，HISR 必须在 LISR 激活之前被创建。

一旦 HISR 有自己的堆栈和控制块，如果它试图进入一个已经被访问 Nucleus PLUS 数据结构时就会被临时封锁。

HISRs 允许访问大多数的 Nucleus PLUS 服务，除了自挂起服务（self-suspension）。另外，一旦 HISR 不能在 Nucleus PLUS 服务时挂起，挂起参数必须总是设置为 NU_NO_SUSPEND。

HISR 有三个优先等级。在一个低优先级的 HISR 处理期间，如果一个更高优先等级的 HISR 被激活，低优先级的 HISR 以与任务抢先方式相同的方式抢先。相同优先级的 HISR 以他们最初激活的顺序运行。所有激活的 HISR 在正常任务调度恢复之前运行。

一个激活的计数器维护着每个 HISR。这个计数器用于确保每个 HISR 为每次激活运行一次。注意一个已经激活的 HISR 的每次附加触发都通过连续调用 HISR 来处理。

HISR 信息

应用程序任务可以获得激活 HISRs 的列表。每个 HISR 的详细信息也可以获得。这些信息包括 HISR 名、总预定计数、优先级和堆栈参数。

中断潜伏期

中断潜伏期是描述每个中断关闭时间值的术语。一旦 Nucleus PLUS 不依赖于关中断保护禁止同步 ISR 访问，中断潜伏期是很短和恒定的。事实上，在一些 Nucleus PLUS 端口，中断关闭只能超过几个指令周期。

应用程序中断锁定

应用程序提供禁止和使能中断的能力。一个中断通过应用程序关闭保持到应用程序解锁。

直接向量访问

Nucleus PLUS 提供直接设置中断向量的能力。ISRs 直接下载到向量表被允许用来存储和恢复使用的寄存器。因此，直接进入向量表的 ISRs 通常用汇编语言编写。这样的 ISRs，倘若遵循某种惯例，可以触发一个 LISR 和/或一个 HISR。

3.8 输入/输出驱动器

大多数的实时应用程序需要不同外围设备的输入输出。这些输入输出的管理通常由一个 I/O 器件驱动器来完成。

通用接口

Nucleus PLUS 为初始化、赋值、释放、输入、输出、状态和中止请求提供一个标准的 I/O 驱动器接口。这个接口由通用控制结构来实现。每个驱动器有一个唯一入口。控制结构确定被请求的服务和所有必须的参数。如果指定的驱动器需要附加参数，控制结构提供一个机制为它连接一个追加的控制结构。有一个标准接口使能应用程序来解决多种多样类似或不一样的外围器件。

驱动器内容

一个 I/O 驱动器通常处理初始化、分配、释放、输入、输出、状态和中止请求过程。如果 I/O 驱动器由中断驱动，中断处理子程序也是必须的。

Nucleus PLUS 设备可以被 I/O 驱动器使用。队列、管道和信号量通常可以被 I/O 驱动器利用。

保护

除了大多数 Nucleus PLUS 服务的可用性之外，I/O 驱动器也提供服务保护内部数据结构不被同步高优先级 ISR 访问。低优先级 ISR 同步访问的保护通过禁止适当的中断来保护。

挂起

I/O 驱动器可以被系统中不同的线程调用。如果一个 I/O 驱动器被一个任务线程调用，与其他 Nucleus PLUS 设备相关的挂起设备有效。另外，一个服务被提供用来挂起和清除 HISR 同步保护。

动态创建

Nucleus PlUS I/O 驱动器可以动态创建和删除。一个应用程序拥有的 I/O 驱动器数量没有预先限定。每个 I/O 驱动器需要一个控制块。控制块内存由应用程序提供。创建和删除驱动器子程序并没有真正的调用驱动器。完成不同的调用进行初始化和中止驱动器。

驱动器信息

应用程序任务可以获得激活的 I/O 驱动器的列表。指定的驱动器的详细信息也可获得。

3.9 系统诊断

Nucleus PLUS 提供给应用程序任务几种提高系统故障的诊断能力的设备。

错误管理

如果一个致命的系统错误出现，处理转到通用错误处理子程序。默认情况下，这个程序准备一个 ASCII 错误消息并暂停系统。然而，附加的错误处理可以由应用程序开发人员加入。

系统历史

Nucleus PLUS 为各种系统活动提供一个环形的事件日志。应用程序任务和 HISR 可以进入这些日志。Nucleus PLUS 服务有一个有条件的编辑选项，用来使能每次服务

请求发生时进入历史日志。历史日志的每次进入包含有关服务和调用的信息。

版本信息

RLD_Release_String 是一个全局 C 字符串，包含当前 Nucleus PLUS 软件版本和版权信息。这个字符串在目标系统中的检查提供对在下面的 Nucleus PLUS 快速鉴定。

许可信息

LID_License_String 是一个全局 C 字符串，包含用户许可信息，包括用户的序列号。

第四章 Nucleus PLUS 服务

本章分类和为每个 Nucleus PLUS 服务提供一个完整的描述。每个服务描述包含一个例子代码片断。

章节段

4.1 任务控制服务
4.2 任务通信服务
4.3 任务同步服务
4.4 定时器服务
4.5 内存服务
4.6 中断服务
4.7 I/O 驱动器服务
4.8 开发服务
4.9 服务定义

4.1 任务控制服务

任务控制服务用来控制应用程序任务的运行。下列服务运行任务控制函数：

```
NU_Create_Task            NU_Delete_Task
NU_Resume_Task            NU_Suspend_Task
NU_Terminate_Task         NU_Reset_Task
NU_Change_Time_Slice      NU_Change_Priority
NU_Change_Preemption      NU_Sleep
NU_Relinquish             NU_Check_Stack
NU_Current_Task_Pointer   NU_Established_Tasks
NU_Task_Pointers
NU_Task_Information
```

4.2 任务通信服务

任务通信服务提供对 Nucleus PLUS 邮箱、队列、管道的访问。下面的服务适合任务通信：

```
NU_Broadcast_To_Mailbox        NU_Create_Mailbox
NU_Delete_Mailbox              NU_Established_Mailboxs
NU_Mailbox_Information         NU_Mailbox_Pointers
NU_Receive_From_Mailbox        NU_Reset_Mailbox
NU_Send_To_Mailbox             NU_Broadcast_To_Queue
NU_Create_Queue                NU_Established_Queues
NU_Delete_Queue                NU_Queue_Pointers
NU_Queue_Information           NU_Reset_Queue
NU_Receive_From_Queue          NU_Send_To_Queue
NU_Send_To_Front_of_Queue      NU_Create_Pipe
NU_Broadcast_To_Pipe           NU_Established_Pipes
NU_Delete_Pipe                 NU_Pipe_Pointers
```

NU_Pipe_Information NU_Reset_Pipe
NU_Receive_From_Pipe NU_Send_To_Pipe
NU_Send_To_Front_of_Pipe

4.3 任务同步服务

任务同步服务提供访问 Nucleus PLUS 信号量、事件标志、信号。下列服务适合任务同步：

NU_Create_Semaphore NU_Delete_Semaphore
NU_Established_Semaphores NU_Obtain_Semaphore
NU_Release_Semaphore NU_Reset_Semaphore
NU_Semaphore_Information Nu_Semaphore_Pointers
NU_Create_Event_Group NU_Delete_Event_Group
NU_Established_Event_Groups NU_Retrieve_Events
NU_Event_Group_Pointers NU_Receive_Signals
NU_Event_Group_Information NU_Send_Signals
NU_Register_Signal_Handler NU_Set_Events
NU_Control_Signals

4.4 定时器服务

定时器服务提供访问 Nucleus PLUS 定时器资源的接口。下列服务适合定时器资源：

NU_Control_Timer NU_Create_Timer
NU_Delete_Timer NU_Established_Timers
NU_Reset_Timer NU_Retrieve_Clock
NU_Set_Clock NU_Timer_Information
NU_Timer_Pointers

4.4 内存服务

内存服务提供访问 Nucleus PLUS 固定和可变长短内存管理设备接口。下列服务适合内存管理：

NU_Allocate_Partition NU_Create_Partition_Pool
NU_Deallocate_Partition NU_Delete_Partition_Pool
NU_Established_Partition_Pools NU_Partition_Pool_Information
NU_Partition_Pool_Pointers NU_Create_Memory_Pool
NU_Allocate_Memory NU_Delete_Memory_Pool
NU_Deallocate_Memory NU_Memory_Pool_Information
NU_Established_Memory_Pools NU_Memory_Pool_Pointers

4.6 中断服务

中断服务提供访问 Nucleus PLUS 中断管理资源的入口。下列服务适合中断设备：

NU_Activate_HISR NU_Control_Interrupts
NU_Local_Control_Interrupts NU_Create_HISR
NU_Current_HISR_Pointer NU_Delete_HISR
NU_Established_HISRs NU_HISR_Information
NU_HISR_Pointers NU_Protect
NU_Register_LISR NU_Setup_Vector
NU_Unprotect

4.7 I/O 驱动器服务

I/O 驱动器服务提供访问 Nucleus PLUS 设备驱动器的标准接口。下列服务适合 I/O 驱动器：

NU_Create_Driver	NU_Delete_Driver
NU_Driver_Pointers	NU_Established_Drivers
NU_Request_Driver	NU_Resume_Driver
NU_Suspend_Driver	

4.8 开发服务

开发服务提供访问 Nucleus PLUS 开发支持设备的接口。下列服务适合开发支持：

NU_Disable_History_Saving	NU_Enable_History_Saving
NU_License_Information	NU_Make_History_Entry
NU_Release_Information	NU_Retrieve_History_Entry

4.9 服务定义

每个 Nucleus PLUS 服务作为本章的保留部分进行描述。服务的描述按字母排序。

包含文件（Inclde File）

为了使用 Nucleus PLUS 服务，NUCLEUS.H 文件必须被包含。

编辑选项

默认情况下，检查提供给 Nucleus PLUS 服务的参数是否有错。错误检查可以在应用程序 C 文件中编辑期间通过定义变量 NU_NO_ERROR 为 CHECKING 来禁止。看目标编译器文档来决定如何在编辑期间定义变量。

标准数据类型

Nucleus PLUS 定义几个标准的数据类型。抛开平台，这些数据类型在尺寸和性能上保持不变。因此，Nucleus PLUS 服务可以以同样的方式在所有的目标环境上运行。下类数据类型由 Nucleus PLUS 定义：

数据类型	含义
UNSIGNED	32 位无符号整型
SIGNED	32 位带符号整型
OPTION	很容易操作的最小数据类型，通常为一个无符号字符
DATA ELEMENT	和上一个 OPTION 数据类型相同
UNSIGNED_CHAR	8 位无符号字符
CHAR	8 位字符
STATUS	与目标编译器符号整型数据类型相同
INT	整型数据类型，与正常的机器上的字的尺寸对应
VOID	等于目标编译器 void 的数据类型

描述格式

描述格式用来描述每个 Nucleus PLUS 服务在本章中的一致性。描述每一节的摘要讨论格式如下：

函数原型

本章包含服务完全的 C 定义。所有的 Nucleus PLUS 服务适合 ANSI C。

描述

本章包括一个段落描述服务的操作。另外，服务的参数在本章也进行了详细的描述。

返回值

无论如何，在服务中，本章定义了值的返回。

任务改变

本章指出服务是否能够导致任务环境的改变。如果任何任务挂起或恢复可以导致调有服务，本章列出‘是’。否则，如果服务不能挂起或恢复任何任务，本章列出‘否’。注：如果‘是’在本章列出，其他的任务可能在服务返回调用程序前运行。

允许调用

本节定义了在 Nucleus PLUS 服务可以被调用时线程的运行。虽然大多数的服务可以从不同线程的运行被调用，挂起也只能在任务线程调用的服务上被指定。其他线程的运行包括：应用程序初始化（Application_Initialize），信号处理（signal Handling），低优先级中断服务子程序（LISR），和高优先级中断服务子程序（HISR）

分类

本章定义了普通的分类，一个服务术语在本章前一节定以的类别。

相关

本章列举了其他一些在讨论中非常相近的服务。

例子

每个服务由下面的样例源代码片断描述。

NU_Active_HISR

函数原型

```
STATUS  NU_Activeate_HISR (NU HISR *hisr);
```

描述

本服务激活由 hisr 指针指向的 HISR。如果指定的 HISR 正在运行，这次激活请求在当前运行结束之前不会处理。对每次激活请求，HISR 运行一次。

返回值

这个服务的完成状态定义如下：

状态	含义
NU_SUCCESS	指示服务成功的完成
NU_INVALID_HISR	指示 HISR 指针非法

任务改变(Tasking Change)

无

允许调用(allowed from)

LISR

分类

中断服务

相关

NU_Create_HISR,NU_Delete_HISR,NU_HISR_Information

例子

```
        NU_HISR Operator_Input;
        STATUS status;
        /* 激活以前创建的 Operator_Input 输入 HISR，控制块由
Operator_Input 输入*/
        status = NU_Activate_HISR(&Operator_Input);
```

NU_Allocate_Memory

函数原型

```
STATUS NU_Allocate_Memory(NU_MEMORY_POOL *pool,
                          VOID **return pointer,
                          UNSIGNED size,
                          UNSIGNED suspend);
```

描述

此项服务从指定的动态内存池分配一个内存块。参数定义如下：

参数	含义
pool	动态内存池指针
return_pointer	指向调用内存指针的指针。请求成功时，分配的块地址被放置到调用内存指针里。
size	从动态内存池里分配时指定的字节数
Suspend	如果内存请求数量非法，指定调用任务是否挂起

下列挂起配置有效：

NU_NO_SUSPEND

不管无论请求是否成功服务立即返回。注：如果服务从非任务线程调用，这是唯一有效的配置。

NU_SUSPEND

调用任务一直挂起到请求内存有效。

时间间隔值（1 ～ 4294967293）

调用任务挂起直到请求内存有效或指定数量的定时器节拍到时，无论哪一个发生在先。

返回值

除了内存指针，本服务返回一个完整的状态。本服务完整状态定义如下：

状态	含义
NU_SUCCESS	表明服务成功完成
NU_INVALID_POOL	表示动态内存池非法
NU_INVALID_POINTER	表示返回指针为空
NU_INVALID_SIZE	表示非法的大小请求
NU_INVALID_SUSPEND	表示试图从非任务线程挂起
NU_NO_MEMORY	表示内存请求不能立即满足
NU_TIMEOUT	表示请求的内存甚至在指定的时间间隔挂起后仍然很难获得
NU_POOL_DELETED	任务挂起时动态内存池删除

任务切换

允许

允许来源

应用程序初始化（Application_Initialize）,高级中断服务子程序（HISR）,信号处理函数（Signal Handler）,任务（task）

类别

内存服务

相关

NU_Deallocate_Memory,NU_Memory_Pool_Information

实例

```
        NU_MEMORY_POOL Pool;
```

```
VOID *memory_ptr;
STATUS status;
/*从内存池控制块“Pool”分配300字节的内存块。如果请求内存非法，无条件挂起调用任务。假设“Pool” 已经在 Nucleus PLUS NU_Create_Memory_Pool服务调用时提前创建。*/
status = NU_Allocate_Memory(&Pool,
                            &memory_ptr,
                               300,
                            NU_SUSPEND);
/*在这里，status表示服务请求是否成功*/
```

NU_Allocate_Partition

函数原型

```
STATUS NU_Allocate_Partition(NU_PARTITION_POOL,*pool,
                        VOID   **return pointer,
                        UNSIGNED  suspend);
```

描述

本服务从指定的内存分区池分配一个内存分区。注内存分区的大小在内存分区池创建时定义。本服务的参数详细定义如下：

参数	含义
pool	指向内存分区池的指针
return_pointer	指向调用内存指针。请求成功时，分配内存分区的地址放置到调用内存指针中
suspend	如果没有有效的内存分区，指定调用任务是否挂起

下列挂起配置是有效的

NU_NO_SUSPEND

不管请求是否被满足，服务立即返回。注：如果服务从非任务线程调用，这是唯一有效的配置。

NU_SUSPEND

调用任务挂起直到内存分区有效。

时间间隔值（1 ～ 4294967293）

调用任务挂起直到一个内存分区有效，或者直到指定数量的时钟节拍到时。

返回值

除了内存指针，本服务返回一个完整的状态值。详细定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_POOL	表示内存分区池指针非法
NU_INVALID_POINTER	表示返回指针为空
NU_INVALID_SUSPEND	表示试图从非任务线程挂起
NU_NO_PARTITION	表示内存分区请求不能立即满足
NU_TIMEOUT	表示甚至挂起到指定的时间间隔之后都没有有效的内存分区
NU_POOL_DELETED	任务挂起时分区内存池被删除

任务改变

可以

允许来源

应用程序初始化，高优先级中断，信号处理器，任务

类型

内存服务

相关

NU_Create_Partition_Pool,NU_Deallocate_Partition,NU_Partition_Pool_Information

实例

```
NU_PARTITION_POOL   Pool;
VOID   *memory_ptr;
STATUS status;
/*用内存分区池控制块“Pool”分配一个内存分区。如果没有可用分区，无条件挂起调用任务。假设“Pool”在 Nuleus PLUS NU_Create_Partion_Pool 服务调用时提前创建*/
status = NU_Allocate_Partion( &Pool,
                              &memory_ptr,
                              NU_SUSPEND);
/*在这里，status 表示服务请求成功与否*/
```

NU_Broadcast_To_Mailbox

函数原型

```
STATUS NU_Broadcast_To_MailBox(NU_MAILBOX *mailbox,
                               VOID   *message,
                               UNSIGNED  Suspend);
```

描述

这项服务广播一个消息到所有正在等待指定的邮箱消息的任务。如果没有任务在等待，消息只是简单的放到邮箱中。每个消息等于 0 到 4 个无符号数据元素。此项服务的参数详细定义如下：

参数	含义
mailbox	邮箱指针
message	广播消息指针
suspend	如果邮箱已经包含有消息，指定是否挂起调用任务

下列挂起配置有效：

NU_NO_SUSPEND

不管请求是否满足服务立即返回。注：如果服务从一个非任务线程被调用时只有此选项有效。

NU_SUSPEND

调用任务挂起直到消息被拷贝入邮箱。

时间间隔值（1 ～ 4294967293）

调用任务挂起直到消息拷贝入邮箱或者直到指定的定时器节拍到时。

返回值

此项服务的完全状态如下定义：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_MAILBOX	表示邮箱指针无效
NU_INVALID_POINTER	表示邮箱指针为空
NU_INVALID_SUSPEND	表示任务试图从非任务线程挂起
NU_MAILBOX_FULL	表示消息不能被立即放置到邮箱中，因为邮箱中已经有消息了
NU_TIMEOUT	表示甚至在挂起指定的时间间隔值之后邮箱仍然不能接收消息，
NU_MAILBOX_DELETED	表示当任务挂起时邮箱被删除
NU_MAILBOX_RESET	表示当任务挂起时邮箱复位

任务改变

可以

允许调用

应用程序初始化，HISR，信号处理器，任务

类别

任务通信服务

相关

NU_Send_To_MailBox,NU_Receive_From_MailBox,NU_Mailbox_Information

实例

```
NU_MAILBOX    Mailbox;
UNSIGNED      message[4];
STATUS        status

/*建立一个消息发送到邮箱。“消息”内容没有意义*/
message[0]    = 0x00001111;
message[1]    = 0x22223333;
message[2]    = 0x44445555;
message[3]    = 0x66667777;
/*发送消息至邮箱控制块“Mailbox”。如果邮箱已经包含一个消息，挂起20 个定时器节拍。假设“Mailbox”在先前通过调用 Nucleus PLUS NU_Create_Mailbox 服务创建*/

status = NU_Broadcast_To_Mailbox( &Mailbox,
                                  &message[0],
                                  20);

/*在这里，status 表示服务请求成功与否*/
```

NU_Broadcast_To_Pipe

函数原型

```
STATUS NU_Broadcast_To_Pipe(  NU_PIPE *pipe,
                              VOID   *message,
                              UNSIGNED  size;
                              UNSIGNED  suspend);
```

描述

此项服务广播一个消息到从指定的管道等待消息的所有任务。如果没有任务在等待，消息只是放到管道末端。管道可以保存多条消息。根据管道的创建，管道消息由定长或不定长字节组成。此项服务的消息详细定义如下：

参数	含义
Pipe	管道指针
Message	广播消息指针
Size	指定消息的字节数。如果管道支持变长消息，此参数必须等于或小于管道支持的消息长度。如果管道支持定长消息，此参数必须与管道支持的消息长度相同。
Suspend	如果管道中没有足够的空间保留消息，指定是否挂起调用的任务。下列挂起配置有效：

NU_NO_SUSPEND

不管请求是否满足服务立即返回。注：如果服务从一个非任务线程被调用时只有此选项有效。

NU_SUSPEND

调用任务挂起直到消息被拷贝入管道。

时间间隔值（1 ～ 4294967293）

调用任务挂起直到消息拷贝入管道或者直到指定的定时器节拍到时。

返回值

此项服务的所有状态定义如下：

状态	含义
NU_SUCCESS	表示任务成功完成
NU_INVALID_PIPE	表示管道指针无效
NU_INVALID_POINTER	表示消息指针为空
NU_INVALID_SIZE	表示消息大小的指定与管道创建时指定的大小不一致。
NU_INVALID_SUSPEND	表示试图从非任务线程挂起
NU_PIPE_FULL	表示因为没有足够的有效空间，消息不能立即放到管道中。
NU_TIMEOUT	表示管道甚至在挂起指定的时间间隔之后仍然不能接收消息。
NU_PIPE_DELETED	任务挂起时管道被删除
NU_PIPE_RESET	任务挂起时管道被复位

任务更改

可以

允许调用

应用程序初始化，HISR，信号处理器，任务

类别

任务通信服务

相关

NU_Send_To_Pipe,NU_Send_To_Frong_Of_Pipe,NU_Receive_From_Pipe,
NU_Pipe_Information

实例

```
NU_PIPE   Pipe;
UNSIGNED_CHAR    message[4];
STATUS    status;

/*建立一个四字节消息发送到管道。“message”内容没有特殊意义*/

message[0] = 0x01;
message[1] = 0x23;
message[2] = 0x45;
message[3] = 0x67;

/*发送消息至管道控制块“Pipe”。即使在管道没有足够空间存放消息也不挂起。假设“Pipe”在先前已经调用 Nucleus PLUS NU_Create_Pipe 服务创建*/

status = NU_Broadcast_To_Pipe(&Pipe,
                              &message[0],
                              4,
                              NU_NO_SUSPEND);

/*在这里，status 表示服务请求成功与否*/
```

NU_Broadcast_To_Queue

函数原型

```
STATUS    NU_Broadcast_To_Queue( NU QUEUE *queue,
                                 VOID   *message,
                                 UNSIGNED  size,
                                 UNSIGNED  suspend);
```

描述

此项服务广播一个消息到从指定的队列等待消息的所有任务。如果没有任务在等待，消息只是放到队列末端。队列可以保存多条消息。根据管道的创建，管道消息由定长或不定长字节组成。此项服务的消息详细定义如下：

参数	含义
queue	队列指针

message 广播消息指针
size 指定消息的字节数。如果队列支持变长消息，此参数必须等于或小于队列支持的消息长度。如果队列支持定长消息，此参数必须与队列支持的消息长度相同。
suspend 如果队列中没有足够的空间保留消息，指定是否挂起调用的任务。下列挂起配置有效：

NU_NO_SUSPEND

不管请求是否满足服务立即返回。注：如果服务从一个非任务线程被调用时只有此选项有效。

NU_SUSPEND

调用任务挂起直到消息被拷贝入队列。

时间间隔值（1 ～ 4294967293）

调用任务挂起直到消息拷贝入队列或者直到指定的定时器节拍到时。

返回值

此项服务的所有状态定义如下：

状态	含义
NU_SUCCESS	表示任务成功完成
NU_INVALID_QUEUE	表示队列指针无效
NU_INVALID_POINTER	表示消息指针为空
NU_INVALID_SIZE	表示消息大小的指定与队列创建时指定的大小不一致。
NU_INVALID_SUSPEND	表示试图从非任务线程挂起
NU_QUEUE_FULL	表示因为没有足够的有效空间，消息不能立即放到队列中。
NU_TIMEOUT	表示队列甚至在挂起指定的时间间隔之后仍然不能接收消息。
NU_QUEUE_DELETED	任务挂起时队列被删除
NU_QUEUE_RESET	任务挂起时队列被复位

任务更改

可以

允许调用

应用程序初始化，HISR，信号处理器，任务

类别

任务通信服务

相关

NU_Send_To_Queue,NU_Send_To_Frong_Of_Queue,NU_Receive_From_Queue,
NU_Queue_Information

实例

```
NU_QUEUE  Queue;
UNSIGNED  Message[4];
STATUS    status;

/*建立一个四字节消息发送到队列。“message”内容没有特殊意义
```

```
*/

message[0] = 0x01;
message[1] = 0x23;
message[2] = 0x45;
message[3] = 0x67;

/*发送消息至队列控制块“Queue”。如果队列已满，挂起直到请求被满足。假设“Queue”在先前已经调用 Nucleus PLUS NU_Create_Queue 服务创建*/
status = NU_Broadcast_To_Queue(   &Queue,
                                  &message[0],
                                  4,
                                  NU_SUSPEND);

/*在这里，status 表示服务请求成功与否*/
```

NU_Change_Preemption

函数原型

```
OPTION    NU_Change_Preemption(OPTION   preempt);
```

描述

此项服务改变当前运行任务的占先状态。如果占先参数包含 NU_NO_PREEMPT，调用任务的占先被禁止。

另外，如果占先参数包含 NU_PREEMPT，调用任务的占先被使能。注：禁止占先同样也禁止了正在调用的任务关联的任何时间片。

返回值

返回先前的占先状态（不是 NU_NO_PREEMPT 就是 NU_PREEMPT）。

任务更改

NO

允许调用

任务

类别

任务控制服务

相关

NU_Create_Task,NU_Change_Priority,NU_Change_Time_Slice

实例

```
OPTION    old_preempt;
/*禁止当前任务占先状态*/
old_Preempt = NU_Change_Preemption(NU_NO_PREEMPT);
/*恢复先前的占先状态*/
NU_Change_Preemption(old_preempt);
```

NU_Change_Priority

函数原型

```
OPTION    NU_Change_Priority(NU_TASK *Task,
                             OPTION new_Priority);
```

描述

此项服务改变指定任务的优先级为包含新优先级的值。优先级为从 0 到 255 范围的数值。数字越小任务优先级越高。

返回值

此项服务返回调用程序先前的优先级。

任务更改

yes

允许调用

应用程序初始化，HISR，信号处理器，任务

类别

任务控制服务

相关

NU_Create_Task,NU_Change_Preemption,NU_Change_Time_Slice

实例

```
NU_TASK Task_Ptr;
OPTION old_priority;

/*更改任务控制块“Task”的优先级为 10。假设“Task”在先前已经调用 Nucleus PLUS NU_Create_Task 服务创建*/

old_priority = NU_Change_Priority (&Task,10);

/*恢复*/
NU_Change_Priority(&Task,old_priority);
```

NU_Change_Time_Slice

函数原型

```
UNSIGNED  NU_Change_Time_Slice(NU_TASK *Task,
                               UNSIGNED  time_slice);
```

描述

此项服务改变指定任务时间片为包含时间片的数值。如果时间片包含值为 0，任务时间片禁止。

返回值

此项服务返回调用的先前时间片值。

任务更改

NO

允许调用

应用程序初始化，HISR，信号处理器，任务

类型

任务控制服务

相关

NU_Create_Task,NU_Change_Priortiy,NU_Change_Preemption

实例

```
    NU_TASK Task;
    UNSIGNED old_time_slice;

    /*改变任务控制块“Task”的时间片为35个定时器节拍。假设“Task”先前已经调用Nucleus PLUS NU_Create_Task服务创建*/
    old_time_slice = NU_Change_Time_Slice(&Task,35);
    /*恢复*/
    NU_Change_Time_Slice(&Task,old_time_slice);
```

NU_Check_Stack

函数原型

```
    UNSIGNED NU_Check_Stack(VOID);
```

描述

此项服务检查调用的堆栈使用情况。如果剩余的空间数少于保存调用所需变量空间，堆栈溢出条件出现并控制权将不能返回到调用。如果一个堆栈溢出条件没有出现，服务返回堆栈保持的空闲字节数。另外，此项服务保持跟踪有效堆栈空间的最小值。

译者注：‘调用（caller）’为名词，表示调用此项函数的主体。

返回值

此项服务返回当前调用使用的堆栈有效字节数。

任务改变

NO

允许调用

HISR，信号处理器，任务

类型

任务控制服务

相关

NU_Create_Task,NU_Create_HISR

实例

```
    UNSIGNED remaining;
    /*检查当前堆栈的溢出条件情况。保存空闲堆栈字节数至“remaining”*/
    remaining = NU_Check_Stack();
```

NU_Control_Interrupts

函数原型

```
INT NU_Control_Interrupts(INT new_level);
```

描述

此项服务根据新等级指定的值使能或禁止中断。中断以任务独立方式禁止和使能。因此，通过此项服务禁止的中断保持禁止直到后来调用此项服务使能中断。新等级值与处理器有关。然而，NU_DISABLE_INTERRUPTS 和 NU_ENABLE_INTERRUPTS 可以分别用来禁止所有中断和使能所有中断。

返回值

此服务返回使能中断以前的等级。

任务改变

NO

允许调用

LISR、HISR、信号处理函数、任务

类别

中断服务

相关

NU_Setup_Vector,NU_Register_LISR,NU_Create_HISR,NU_Delete_HISR

举例

```
INT    old_level;       /*旧的中断等级*/
/*临时停止所有中断*/
old_level = NU_Control_Interrupts(NU_DISABLE_INTERRUPTS);
/*恢复先前的中断停止等级*/
NU_Control_Interrupts(old_level);
```

NU_Control_Signals

函数原型

```
UNSIGNED NU_Control_Signals(UNSIGNED signal_enable_mask);
```

描述

此项服务使能和/或禁止调用任务的信号。每个任务可以有 32 个信号有效。每个信号由 signal_enable_mask 一个位描述。信号 0 由 0 位描述，信号 31 由 31 描述。在 signal_enable_mask 中置位使能相应信号，清除位禁止相应信号。注：signal_enable_mask 在任务创建期间清除。

返回值

服务返回前一次信号使能/静止标志。

任务更改

NO

允许调用

任务

类别

任务同步服务

相关

NU_Send_Signals,NU_Receive_Signal,NU_Register_Signal_Handler

举例

```
UNSIGNED old_signal_mask;/*先前定义的信号*/
/*临时停止所有当前任务的信号量*/
old_signal_mask = NU_Control_Signals(0);
/*恢复先前的信号*/
NU_Control_Signals(old_signal_mask);
```

NU_Control_Timer

函数原型

STATUS NU_Control_Timer(NU_TIMER *timer,OPTION enable);

描述

此项服务使能或禁止应用程序指针指定的定时器。作为使能参数的有效值是 NU_ENABLE_TIMER 和 NU_DISABLE_TIMER。

返回值

此项服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_TIMER	表示定时器指针非法
NU_INVALID_ENABLE	表示使能参数非法

任务更改

NO

允许调用

应用程序初始化，HISR，信号处理函数，任务

类别

定时器服务

举例

```
NU_TIMER    Timer;
STATUS      status;

/*禁止定时器控制模块 Timer*，假设 Timer 已经在先前调用 Nucleus PLUS 中
NU_Create_Timer 服务被创建/

status = NU_Control_Timer(&Timer,NU_Disable_Timer);

/*这里 status 表示服务请求是否成功*/
```

NU_Create_Driver

函数原型

```
STATUS NU_Create_Driver(   NU_DRIVE *driver,
                           CHAR *name,
                           VOID (*driver_entry)
                           (NU_DRIVER *,
                           NU_DRIVER_REQUEST*));
```

描述

此项服务创建一个输入/输出驱动器。注：此项服务不调用驱动器。此项服务的参数详细定义如下：

参数	含义
driver	用户提供驱动器控制块的指针。注：所有针对驱动器的并发请求都需要这个指针。
name	驱动器 8 字节名字指针。名字不需以零结尾。
driver_entry	指定驱动器的函数入口指针。注：函数必须与描述接口一致。

返回值

此项服务完成状态如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_DRIVER	表示驱动器指针为空或已经被应用
NU_INVALID_POINTER	表示入口指针为空

任务更改

NO

允许调用

应用程序初始化（Application_Initialize）,HISR,信号处理器，任务

类别

IO 驱动器服务

相关

NU_Delete_Driver,NU_Established_Drivers,NU_Driver_Pointers

举例

```
/*假设驱动器的控制块“Driver”已经在全局数据结构中定义。这是几种分配控制块的方法之一*/
NU_DRIVER    Driver;
/*假设 status 在本地定义*/
STATUS    status;
/* 创建一个驱动器，函数入口为“Driver_Entry”，注意在这之后 NU_Request_Driver 必须被调用来真正初始化 I/O 驱动器*/
status = NU_Create_Driver(&Driver,”any name”,Driver_Entry);

/*这里 status 表示服务是否成功*/
```

NU_Create_Event_Group

函数原型

```
STATUS NU_Create_Event_Group(NU_EVENT_GROUP *group,CHAR *name);
```

描述

此项服务创建一个事件标志集。每个事件标志集包含 32 个事件标志。所有事件标志都初始化为 0。此项服务的参数定义如下：

参数	含义
group	用户提供事件标志集控制块指针。注：所有针对事件标志集相关的请求都需要这个指针
name	事件集 8 字符名字。名字不能为空

返回值

此项服务完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_GROUP	表示事件集控制块指针为空或已经被应用。

任务更改

NO

允许调用

应用程序初始化（Application_Initialize），HISR,信号处理器，任务

类型

任务通信服务

举例

```
/*假设事件集控制块“Events”已经定义为全局数据结构。这是分配控制块方法之一*/
NU_EVENT_GROUP   Events;
/*假设状态在本地定义*/
STATUS status;/*事件集创建状态 */
/*创建一个事件标志集*/
status = NU_Create_Event_Group(&Events,”any name”);
/*在这里 status 表示服务成功是否成功*/
```

NU_Create_HISR

函数原型

```
STATUS NU_Create_HISR(  NU_HISR   *hisr,
                        CHAR *name,
                        VOID (*hisr_entry)(VOID),
                        OPTOIN priority,
                        VOID   *stack_pointer,
                        UNSIGNED  stack_size);
```

描述

此项服务创建一个高级中断服务子程序（HISR）。HISRs 允许被大多数的 Nucleus PLUS 服务调用，不像低级中断服务子程序（LISRs）。此项服务参数详细定义如下：

参数	含义
hisr	用户提供的 HISR 控制块指针。注：所有关于这项 HISR 的请求都需要这个指针。
name	8 字节 HISR 名称。名字不能为空
hisr_entry	指定 HISR 的函数入口点
priority	有三个 HISR 优先级（0－2）。优先级 0 为最高
stack_pointer	HISR 的堆栈区指针。每个 HISR 有它自己的堆栈区。注意 HISR 堆栈已经被调用者分配过。
stack_size	HISR 堆栈的字节数

返回值

此项服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_HISR	表示 HISR 控制块指针为空或已经被应用
NU_INVALID_ENTRY	表示 HISR 入口指针为空
NU_INVALID_PRIORITY	表示 HISR 优先级为空
NU_INVALID_MEMORY	表示堆栈指针为空
NU_INVALID_SIZE	表示堆栈尺寸太小

任务更改

NO

允许调用

应用程序初始化（Application_Initialize）,HISR,信号处理器,任务

类别

中断服务

相关

NU_Delete_HISR,NU_Established_HISRs,NU_HISR_Pointers,NU_HISR_Information

举例

```
/*假设 HISR 控制块“HISR”定义为全局数据结构。这是分配控制块的方法之一*/

NU_HISR  HISR
/*假设 status 定义为局部变量*/

STATUS   status;/*HISR 创建 status*/

/*创建一个 HISR。注意 HISR 入口函数为“HISR_Entry”和“stack_pointer”指向一个 400 字节预先分配内存块*/

status = NU_Create_HISR(&HISR,”any name”,
                        HISR_Entry,2,stack_pointer,400);
            /*在这里 status 表示服务成功是否成功*/
```

NU_Create_Mailbox

函数原型

```
STATUS NU_CreateMailbox(    NU_MAILBOX *mailbox,
                            CHAR   *name,
                            OPTION suspend_type);
```

描述

本服务创建一个任务通信邮箱。一个邮箱可以保存一条消息。邮箱消息大小等于四个UNSIGNED 数据类型。（译者：UNSIGNED 为 32 位）。此项服务参数详细定义如下：

参数	含义
mailbox	用户供应邮箱控制块指针。注：所有针对邮箱并发请求都需要这个指针。
name	指向邮箱的 8 字符名称。名字不能为空。
Suspend－type	指定邮箱任务挂起模式。这个参数的有效配置为 NU_FIFO 和 NU_PRIORITY,分别表示先进先出和优先级顺序任务挂起

返回值

服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_MAILBOX	表示邮箱控制块指针为空或已经被应有
NU_INVALID_SUSPEND	表示 suspend_type 参数无效

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务通信服务

相关

NU_Delete_Mailbox,NU_Established_Mailboxs,NU_Mailbox_Pointer, NU_Mailbox_Information

举例

```
/*假设邮箱控制块“Mailbox”在全局数据结构中定义。这是分配控制块的方法之一*/

NU_MAILBOX    Mailbox;

/*假设状态在本地定义*/

STATUS status;/*邮箱创建状态*/

/*创佳一个邮箱以 FIFO 方式管理任务挂起*/

status = NU_Create_Mailbox(&Mailbox,”any name”,NU_FIFO);

/*status 表示服务成功与否*/
```

NU_Create_Memory_Pool

函数原型

```
STATUS NU_Create_Memory_Pool( NU_MEMORY_POOL *pool,
                              CHAR *name,
                              VOID *start_address,
                              UNSIGNED  pool_size,
                              UNSIGNED  min_alloction,
                              OPTION suspend_type);
```

描述

此项服务在调用指定的内存区内创建动态内存池，此项服务参数详细定义如下：

参数	含义
pool	用户供应内存池控制块指针。注：所有针对内存池并发请求都需要此指针。
name	内存池 8 字符名称的指针。不能为空
start_address	内存池起始地址
pool_size	内存池大小
min_alloction	指定内存池中每个分配的最小字节数
suspend_type	指定内存池挂起类型。此参数有效配置为 NU_FIFO 和 NU_PRIORITY，分别表示先入先出和优先级顺序挂起。

返回值

此项服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表明服务成功完成。
NU_INVALID_POOL	表明内存池控制块指针为空或已经被使用。
NU_INVALID_MEMORY	表示由起始地址指定的内存区无效。
NU_INVALID_SIZE	表示内存池的尺寸和/或最小分配尺寸非法。
NU_INVALID_SUSPEND	表示挂起类型参数无效。

任务更改

NO

允许调用

Application_Initialize,HISR,Signal_Handler,task

类型

内存服务

相关

NU_Delete_Memory_Pool,NU_Established_Memory_Pools,
NU_Memory_Pool_Pointers,NU_Memory_Pool_Information

举例

```
/*假设动态内存控制块“Pool”在全局数据结构中定义。这是分配控制块的方法之一*/

NU_MEMORY_POOL   Pool;

/*假设状态在本地定义*/

STATUS status;
```

/*创建一个 4000 字节动态内存池，起始地址在 0xA000。最小分配尺寸为 30 字节。任务挂起以优先级顺序。*/

```
status = NU_Create_Memory_Pool(   &Pool,
                                  “any name”,
                                  (VOID *)0xA000,
                                  4000,
                                  30,
                                  NU_PRIORITY);
```

/*在这里 status 表示服务成功与否*/

NU_Create_Partition_Pool

函数原型

```
STATUS NU_Create_Partition_Pool(  NU_PARTITION_POOL *pool,
                                  CHAR *name,
                                  VOID *start_address,
                                  UNSIGNED pool_size,
                                  UNSIGNED partition_size,
                                  OPTION suspend_type);
```

描述

此项服务从一个由调用指定的内存区中创建一个固定尺寸的内存分区池。参数详细定义如下：

参数	含义
pool	用户供应分区池控制块指针。注：针对这个分区池的并发请求需要这个指针。
name	分区池 8 字符指针。不能为空
pool_size	指定内存区总字节数
Partition_size	为每个池中的每个分区指定字节数。对每个区来说这是一个少量的消耗。这个消耗需要两个数据指针使用。
suspend_type	指定内存池挂起类型。此参数有效配置为 NU_FIFO 和 NU_PRIORITY，分别表示先入先出和优先级顺序挂起。

返回值

此项服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表明服务成功完成。
NU_INVALID_POOL	表明分区池控制块指针为空或已经被使用。
NU_INVALID_MEMORY	表示由起始地址指定的内存区无效。
NU_INVALID_SIZE	表示分区的尺寸为 0 或者比总分区内存还大。
NU_INVALID_SUSPEND	表示挂起类型参数无效。

任务更改

NO

允许调用

Application_Initialize,HISR,Signal_Handler,task

类型

内存服务

相关

NU_Delete_Memory_Pool,NU_Established_Memory_Pools,
NU_Partition_Pool_Pointers,NU_Partition_Pool_Information

举例

```
/*假设动态内存控制块“Pool”在全局数据结构中定义。这是分配控制块的方法之一*/

NU_PARTITION_POOL   Pool;

/*假设状态在本地定义*/

STATUS status;

/*在一个起始地址为 0xB000，2000 字节内存区中，创建一个 40 字节内存分区的分区内存池。最小分配尺寸为 30 字节。任务挂起以 FIFO 顺序。*/

status = NU_Create_Partition_Pool(   &Pool,
                                     “any name”,
                                     (VOID *)0xB000,
                                     2000,
                                     40,
                                     NU_FIFO);
/*在这里 status 表示服务成功与否*/
```

NU_Create_Pipe

函数原型

```
STATUS NU_Create_Pipe(  NU_Pipe *pipe,
                        CHAR *name,
                        VOID *start_address,
                        UNSIGNED pipe_size,
                        OPTION message_type,
                        UNSIGNED message_size,
                        OPTION suspend_type);
```

描述

此项服务创建一个消息管道。管道创建支持固定和可变尺寸的消息管理。管道消息是面向字节的。参数详细定义如下：

参数	含义
Pipe	用户提供管道控制块指针。注：所有针对管道的并发请求都需要这个指针

	道的并发请求都需要这个指针。
Name	管道 8 字符名字指针。名字不能为空。
Start_address	指定管道起始地址
Pipe_size	管道中的字节总数
Message_type	指定管道管理的消息类型。NU_FIXED_SIZE 表示管道管理定长消息。注：定长消息只能用于消息尺寸均匀分布的管道区域。 NU_VARIABLE_SIZE 表示管道管理变长尺寸消息。注：每个变长尺寸的消息在管道内需要一个附加的 UNSIGNED 数据类型的消耗。附加填充字节对一个消息必不可少，为了确保下一个变长尺寸的消息 UNSIGNED 队列。
Message_size	如果管道支持定长消息，这个参数指定每个消息的精确长度。另外，如果管道支持变长消息，这个参数表示最大消息尺寸。所有尺寸都是字节数。
Suspend_type	指定管道挂起类型。此参数有效配置为 NU_FIFO 和 NU_PRIORITY，分别表示先入先出和优先级顺序挂起。

返回值

此项服务完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成。
NU_INVALID_PIPE	表示管道控制模块指针为空或已经被应用
NU_INVALID_MEMORY	表示指定起始地址的内存区或尺寸参数无效
NU_INVALID_MESSAGE	表示消息类型参数无效
NU_INVALID_SIZE	表示指定的消息尺寸大于管道尺寸或为 0
NU_INVALID_SUSPEND	表示挂起类型参数无效

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类型

任务通信服务

相关

NU_Delete_Pipe,NU_Establish_Pipes,NU_Pipe_Pointers,
NU_Pipe_Information,NU_Reset_Pipe

举例

```
/*假设管道控制块“Pipe”作为全局数据结构定义。这是分配控制块方法之一*/

NU_PIPE   Pipe;
/*假设 status 在本地定义*/
```

```
STATUS status;

/*在一个由“start”变量指定起始地址，大小为 1500 字节的内存区创建一个管道。管道支持定长，20 字节消息。管道以任务优先级顺序挂起任务*/

status = NU_Create_Pipe(   &Pipe,
                           “any name”,
                           start,
                           1500,
                           NU_FIXED_SIZE,
                           20,
                           NU_PRIORITY);
/*这里，status 表示服务成功与否*/
```

NU_Create_Queue

函数原型

```
   STATUS NU_Create_Queue( NU_QUEUE  *queue,
                           Char *name,
                           VOID *start_address,
                           OPTION messge_type,
                           UNSIGNED message_size,
                           OPTION suspend_type);
```

描述

此服务创建一个消息队列。队列创建支持定长和变长消息的管理。队列消息由一个或多个 UNSIGNED 数据元素组成。参数具体定义如下：

参数	含义
queue	用户提供队列控制块指针。注：所有针对队列的并发服务都需要这个指针。
Name	指向队列的 8 字符名称。不能为空
start_address	指定队列的起始地址。注：此地址必须与 UNSIGNED 数据访问正确对应。
Queue_size	指定对列中 UNSIGNED 数据元素的总数
message_type	指定队列管理的消息类型。NU_FIXED_SIZE 表示队列管理定长消息。注：定长消息只能用于消息尺寸均匀分布的队列区域。 NU_VARIABLE_SIZE 表示队列管理变长尺寸消息。注：每个变长尺寸的消息在队列内需要一个附加的 UNSIGNED 数据类型的消耗。附加填充字节对一个消息必不可少，为了确保下一个变长尺寸的消息 UNSIGNED 队列。
Message_size	如果队列支持定长消息，这个参数指定每个消息的精确长度。另外，如果队列支持变长消息，这个参数表示最大消息尺寸。所有尺寸都是字节数。
Suspend_type	指定队列挂起类型。此参数有效配置为 NU_FIFO 和 NU_PRIORITY，分别表示先入先出和优先级顺序挂起。

返回值

此项服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成。
NU_INVALID_QUEUE	表示队列控制模块指针为空或已经被应用
NU_INVALID_MEMORY	表示指定起始地址的内存区或尺寸参数无效
NU_INVALID_MESSAGE	表示消息类型参数无效
NU_INVALID_SIZE	表示指定的消息尺寸大于管道尺寸或为 0
NU_INVALID_SUSPEND	表示挂起类型参数无效

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务通信服务

相关

NU_Delete_Queue,NU_Established_Queues,NU_Queue_Pointers,
NU_Queue_Pointers,NU_Queue_Information,NU_Reset_Queue

举例

```
/*假设队列控制块“Queue”定义为全局数据结构。这是分配控制块的几种方法之一*/

NU_QUEUE  Queue;

/*假设状态在本地定义*/

STATUS status;

/*创建一个起始地址由“start 指定，1000 个 UNSIGNED 数据类型的队列。支持变长消息，最大消息尺寸为 20，队列任务挂起按照 FIFO 顺序”*/

status = NU_Create_Queue(  &Queue,
                           “any name”,
                           start,
                           1000,
                           NU_VARIABLE_SIZE,
                           20,
                           NU_FIFO);
/*此处 status 代表服务成功与否*/
```

NU_Create_Semaphore

函数原型

```
STATUS NU_Create_Semaphore(   NU_SEMAPHORE *semaphore,
```

```
                                    CHAR   *name,
                                        UNSIGNED initial_count,
                                        OPTION suspend_type);
```

描述

此项服务创建一个计数信号量。信号量值范围 0～4294967294。参数详细定义如下：

参数	含义
semaphore	用户提供信号量控制块指针。注：所有针对信号量并发请求都需要这个指针。
name	信号量 8 字符指针。不能为空
initial_count	指定信号量的初始值
supend_type	指定任务挂起类型。此参数有效配置为 NU_FIFO 和 NU_PRIORITY，分别表示先入先出和优先级顺序挂起。

返回值

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_SEMAPHORE	表示信号量控制块指针为空或已经被应用
NU_INVALID_SUSPEND	表示挂起类型参数无效

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务同步服务

相关

NU_Delete_Semaphore,NU_Established_Semaphores,
NU_Semaphore_Pointers,NU_Semaphore_Information

举例

```
/*假设信号量控制块”Semaphore”定义为全局数据结构。这是分配控制块的几种方法之一*/
NU_SEMAPHORE Semaphore;

/*假设 status 在本地定义*/
STATUS status；
/*创建一个信号量，初始化值为 1，优先级顺序任务挂起*/

status = NU_Create_Semaphore( &Semaphore,
                              “any name”,
                              1,
                              NU_PRIORITY);
                    /*此处 status 表示服务成功与否*/
```

NU_Create_Task

函数原型

```
STATUS NU_Create_Task(  NU_TASK *task,
                        CHAR *name,
                        VOID (*task_entry)(UNSIGNED)
                        (VOID *)
                        UNSIGNED argc,VOID *argv,
                        VOID *stack_address,
                        UNSIGNED stack_size,
                        OPTION priority,
                        UNSIGNED time_slice,
                        OPTION preempt,
                        OPTION auto_start);
```

描述

此项服务创建一个应用程序任务。参数详细定义如下：

参数	含义
task	用户提供的任务控制块指针。注：所有针对这个任务的并发请求都需要这个指针。
name	任务 8 字节指针。名称不能为空
task_entry	指定任务函数入口
argc	一个 UNSIGNED 数据类型，可以用来传递初始化信息到任务
argv	指针，传递初始化信息到任务
stack_address	分配任务堆栈区的起始内存地址位置
stack_size	指定堆栈的字节数
Priority	在 0～255 之间指定优先级值。数值越低，任务级别越高。
time_slice	表示中止运行任务的定时器节拍最大值。0 值表示禁止任务时间片。
preempt	此配置的有效参数为 NU_PREEMPT 和 NU_NO_PREEMPT。NU_PREEMPT 表示任务占先有效，NU_NO_PREEMPT 表示任务占先无效。注：如果任务占先无效，时间片禁止
auto_start	此配置有效参数为：NU_NO_START 和 NU_START。NU_START 表示在任务创建后把任务放置到就绪状态。NU_NO_START 表示任务在创建之后处于休眠状态。参数为 NU_NO_START 的任务稍后必须恢复。

返回值

下面是服务完全的返回状态：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_TASK	表示任务控制块指针为空

NU_INVALID_ENTRY	表示入口函数指针为空
NU_INVALID_MEMORY	表示 stack_address 指定的内存区为空
NU_INVALID_SIZE	表示指定的堆栈尺寸不够大
NU_INVALID_PRIORITY	表示指定的优先级无效
NU_INVALID_PREEMPT	表示占先参数无效。这个错误发生在连同无占先配置一起时间片被指定。
NU_INVALID_START	表示 auto_start 参数无效

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务控制服务

相关

NU_Delete_Task,NU_Established_Tasks,NU_Task_Pointers
NU_Task_Information,NU_Reset_Task

举例

/*假设任务控制模块“Task“定义为全局数据结构。这是分配控制块的几种方法之一*/

```
NU_TASK Task;

/*假设 status 在本地定义*/

STATUS status;

/*创建一个入口函数为“task_entry“,堆栈指针为”stack_ptr“,堆栈尺寸为 2000 字节的任务。注：下列附加参数： argc and argv (0,NULL)，优先级为 200，时间片为 15 个定时器节拍，占先有效，自动启动*/

status = NU_Create_Task(   &Task,
                           “any name”,
                           task_entry,
                           0,
                           NULL,
                           Stack_ptr,
                           2000,
                           200,
                           15,
                           NU_PREEMPT,
                           NU_START);
```

NU_Create_Timer

函数原型

```
STATUS NU_Create_Timer( &NU_TIMER *timer,
                        CHAR *name,
                        VOID (*expiration routine)(UNSIGNED),
                        UNSIGNED id,
                        UNSIGNED initial_time,
                        UNSIGNED reschedule_time,
                        OPTION enable);
```

描述

此项服务创建应用程序定时器。指定期满子程序在每个定时器每次到期时运行。应用到期子程序应该避免任务挂起配置。到期子程序挂起可以导致其他需要时钟的应用程序延时。此项服务的参数详细定义如下：

参数	含义
timer	用户供应的定时器控制块指针。注：所有针对定时器的相关请求都需要这个指针。
name	定时器 8 字节名字指针。不能为空
expiration_routine	在时钟到期时指定应用程序子程序运行
id	一个由到时子程序提供的 UNSIGNED 数据类型。参数用于帮助识别使用同一个到时子程序的定时器。
initial_time	为定时器子程序指定时钟节拍初始值
reschedule_time	在第一次到时后指定时钟节拍到时值。如果参数为 0 的话，定时器只到时一次。
enable	有效配置参数为：NU_ENABLE_TIMER 和 NU_DISABLE_TIMER。NU_ENABLE_TIMER 在定时器创建后激活它。配置为 NU_DISABLE_TIMER 的定时器必须在之后调用 NU_Control_Timer 来使能。

返回值

服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功调用
NU_INVALID_TIMER	表示时钟控制块指针为空或已经被应用
NU_INVALID_FUNCTION	表示到时函数指针为空
NU_INVALID_ENABLE	表示 enable 参数无效

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

定时器服务

相关

NU_Delete_Timer,NU_Established_Timers,NU_Timer_Pointers,
NU_Timer_Information,NU_Reset_Timer

举例

```
/*假设时钟控制模块“Timer”定义为全局数据结构。这是分配控制块的方法之一*/

NU_TIMER Timer;

/*假设 status 在本地定义*/

STATUS status;

/*创建一个定时器，到时函数为“timer_expire”,ID 为 0，初始化到时值为 23 个时钟节拍。在初始化到时值之后，定时其没 5 个时钟节拍到时。注意定时器在创建期间使能*/

status = NU_Create_Timer(  &Timer,
                           “any name”,
                           timer_expire,
                           0,
                           23,
                           5,
                           NU_ENABLE_TIMER);
/*这里 status 表示服务成功与否*/
```

NU_Current_HISR_Pointer

函数原型

```
NU_HISR *NU_Current_HISR_Pointer(VOID);
```

描述

此项服务返回当前运行 HISR 的指针。如果调用没有 HISR，返回值为 NU_NULL。

任务更改

NO

允许调用

HISR，LISR

类别

中断服务

相关

NU_Established_HISRs,NU_HISR_Pointers,NU_HISR_Information

举例

```
NU_HISR *HISR_ptr;
/*获得当前运行的 HISR 指针*/
HISR_ptr = NU_Current_HISR_Pointer();
```

NU_Current_Task_Pointer

函数原型

```
NU_TASK *NU_Current_Task_Pointer(VOID);
```

描述

此项服务返回当前激活的任务指针。如果当前没有任务运行，返回一个 NU_NULL。如果一个 HISR 是激活的线程，且可以在 HISR 完成后恢复的任务被中断，返回值仍然为 NU_NULL。

任务更改

NO

允许调用

LISR,Signal Handler,task

类别

任务控制服务

相关

```
NU_TASK *task_ptr;
/*获得当前激活任务的指针*/
task_ptr = NU_Current_Task_Pointer();
```

NU_Deallocate_Memory

函数原型

```
STATUS NU_Deallocate_Memory(VOID *memory);
```

描述

此项服务把 memory 指向的内存块收回到动态内存池。

返回值

服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_POINTER	表示内存块指针为空，或当前没有分配，或无效

任务更改

YES

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

内存服务

相关

NU_Allocation_Memory,NU_Memory_Pool_Information

举例

```
STATUS status;

/*收回 memory 指针指定的内存块*/

status = NU_Deallocate_Memory(memory);

/*在这里 status 表示服务成功与否*/
```

NU_Deallocate_Partion

函数原型

```
STATUS NU_Deallocate_Partition(VOID *partition);
```

描述

此项服务收回 partition 指定的内存分区到相关池中。

返回值

服务完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_POINTER	表示内存分区指针为空，或当前为分配，或无效

任务更改

YES

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

内存服务

相关

NU_Allocate_Partition,NU_Parition_Pool_Information

举例

```
STATUS status;
/*收回 partition 指针指向的内存分区*/

status = NU_Deallocate_Partition(partition);

/*这里 status 表示服务是否成功*/
```

NU_Delete_Driver

函数原型

```
STATUS NU_Delete_Driver(NU_DRIVER *driver);
```

描述

此项服务删除一个先前创建的 I/O 驱动器。参数 driver 识别需要删除的 I/O 驱动器.指定驱动器所有的使用必须完全在调用这个服务前。典型的做法就是用中止请求来实现。

返回值

此项服务完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_DRIVER	表示驱动器指针非法

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler，task

类别

I/O 驱动器服务

相关

NU_Create_Driver,NU_Established_Drivers,
NU_Driver_Pointers

举例

```
NU_DRIVER Driver;
STATUS status;

/*删除“Driver”指定的驱动器控制块。假设“Driver”在先前已经调用 Nucleus PLUS
服务 NU_Create_Driver 时创建。*/

status = NU_Delete_Driver(&Driver);

/*这里，status 表示服务请求成功与否*/
```

NU_Delete_Event_Group

函数原型

```
STATUS NU_Delete_Event_Group(NU_EVENT_GROUP *group);
```

描述

此项服务删除一个先前定义的事件标志集。参数 group 用于识别需要删除的事件标志集。在这个事件集上挂起的任务恢复时返回适当的错误状态。在删除期间和删除之后，应用程序必须防止这个事件集的使用。

返回值

此项服务完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_GROUP	表示事件标志集指针无效

任务更改

YES

允许调用

Application_Initalize,HISR,Signal Handler,task

类别

任务同步服务

相关

NU_Create_Event_Group,NU_Established_Event_Groups,
NU_Event_Group_Pointers,NU_Event_Group_Information

举例

```
NU_EVENT_GROUP Group;
STATUS Status;

/*删除”Group”指定的事件标志集控制块。假设”Group”先前已经调用 Nucleus PLUS
服务 NU_Create_Event_Group 服务创建*/

status = NU_Delete_Event_Group(&Group);

/*这里 status 表示服务请求是否成功*/
```

NU_Delete_HISR

函数原型

```
STATUS NU_Delete_HISR(NU_HISR *hisr);
```

描述

此服务删除一个先前创建的 HISR，参数 hisr 确定需要删除的 HISR。在删除期间和之后，应用程序必须防止 HISR 的使用。

返回值

此服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_HISR	表示 HISR 指针非法

任务更改

NO

允许调用

Application_Initialize,Signal Handler,task

类别

中断服务

相关

NU_Create_HISR,NU_Established_HISRs,NU_HISR_Pointers,
NU_HISR_Information

举例

```
NU_HISR Hisr;
STATUS status;

/*删除指针"Hisr"指定的 HISR 控制块。假设"Hisr"再先前已经调用 Nulceus PLUS
服务 NU_Create_HISR 创建*/

status = NU_Delete_HISR(&Hisr);

/*这里，status 表示服务请求是否成功*/
```

NU_Delete_Mailbox

函数原型

```
STATUS NU_Delete_Mailbox(NU_MAILBOX *mailbox);
```

描述

此服务删除先前创建的邮箱。参数 mailbox 识别需要删除的邮箱。在这个邮箱上的任务挂起恢复时返回相应的错误状态。在删除期间和之后，应用程序必须防止这个邮箱的使用。

返回值

此项服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_MAILBOX	表示邮箱指针无效

任务更改

YES

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务通信服务

相关

NU_Create_Mailbox,NU_Established_Mailbox,
NU_Mailbox_Pointers,NU_Mailbox_Information

举例

```
NU_MAILBOX Mailbox；
STATUS status;

/*删除“Mailbox”指定的邮箱控制块。假设“Mailbox”已经在先前调用 Nucleus PLUS 服务 NU_Create_Mailbox 创建*/

status = NU_Delete_Mailbox(&Mailbox);

/*这里 status 表示服务请求成功与否*/
```

NU_Delete_Memory_Pool

函数原型

```
STATUS NU_Delete_Memory_Pool(NU_Memory_Pool *pool);
```

描述

此项服务删除先前创建的动态内存池。参数 pool 确定需要删除的动态内存池。在这个动态内存池上挂起的任务恢复时返回适当的错误状态。在删除期间和之后，应用程序必须防止这个动态内存池的使用。

返回值

此项服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_POOL	表示动态内存池指针为空

任务更改

Yes

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

内存服务

相关

NU_Create_Memory_Pool,NU_Established_Memory_Pools,
NU_Memory_Pool_Pointers,NU_Memroy_Pool_Information

举例

```
NU_MEMORY_POOL Pool;
STATUS status;

/*删除 Pool 指定的内存池控制块。假设“Pool”再现前已经调用 Nucleus PLUS 服务 NU_Create_Memory_Pool 创建*/
```

```
status = NU_Delete_Memory_Pool(&Pool);

/*这里 status 表示服务请求成功与否*/
```

NU_Delete_Partition_Pool

函数原型

```
STATUS NU_Delete_Partition_Pool(NU_PARTITION_POOL *pool);
```

描述

此服务删除一个先前定义的内存分区池。参数 pool 确定需要删除的内存分区池。在这个内存分区池上挂起的任务恢复时返回适当的错误状态。在删除期间和之后，应用程序必须防止内存分区池的使用。

返回值

服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_POOL	表示内存分区池指针无效

任务更改

YES

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

内存服务

相关

NU_Create_Partition,NU_Established_Partition_Pools,
NU_Partition_Pool_Pointers,NU_Partition_Pool_Information

举例

```
NU_PARTITION_POOL Pool;
STATUS status;

/*删除 Pool 指定的分区池控制块，假设“Pool”在先前已经调用 Nucleus PLUS 服务
NU_Create_Partition_Pool 创建*/

status = NU_Delete_Partition(&Pool);

/*这里，status 表示服务请求成功与否*/
```

NU_Delete_Pipe

函数原型

```
STATUS NU_Delete_Pipe(NU_PIPE *Pipe);
```

描述

此服务删除一个先前定义的消息管道。参数 Pipe 确定需要删除的消息管道。在这个管道上挂起的任务恢复时返回适当的错误状态。在删除期间和之后，应用程序必须防止管道的使用。

返回值

此项服务完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_PIPE	表示管道指针非法

任务更改

Yes

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务通信服务

相关

NU_Create_Pipe,NU_Established_Pipes,NU_Pipe_Pointers,
NU_Pipe_Information,NU_Reset_Pipe

举例

```
NU_PIPE Pipe;
STATUS status;

/*删除 Pipe 指定的管道控制块。假设“Pipe”在先前已经调用 Nucleus PLUS 服务
NU_Create_Pipe 创建*/

status = NU_Delete_Pipe(&Pipe);

/*这里 status 表示服务请求成功与否*/
```

NU_Delete_Queue

函数原型

```
STATUS NU_Delete_Queue(NU_QUEUE *Queue);
```

描述

此服务删除一个先前定义的消息队列。参数 Queue 确定需要删除的消息队列。在这个队列上挂起的任务恢复时返回适当的错误状态。在删除期间和之后，应用程序必须防止队列的使用。

返回值

此项服务完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_QUEUE	表示队列指针非法

任务更改

Yes

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务通信服务

相关

NU_Create_Queue,NU_Established_Queues,NU_Queue_Pointers,
NU_Queue_Information,NU_Reset_Queue

举例

```
NU_QUEUE Queue;
STATUS status;

/*删除 Queue 指定的队列控制块。假设“Queue”在先前已经调用 Nucleus PLUS 服务
NU_Create_Queue 创建*/

status = NU_Delete_Queue(&Queue);

/*这里 status 表示服务请求成功与否*/
```

NU_Delete_Semaphore

函数原型

```
STATUS NU_Delete_Semaphore(NU_SEMAPHORE *Semaphore);
```

描述

此服务删除一个先前创建的信号量。参数 Semaphore 确定需要删除的信号量。在这个这个上挂起的任务恢复时返回适当的错误状态。在删除期间和之后，应用程序必须防止信号量的使用。

返回值

此项服务完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_SEMAPHORE	表示信号量指针非法

任务更改

Yes

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务同步服务

相关

NU_Create_Semaphore,NU_Established_Semaphores,
NU_Semaphore_Pointers,NU_Semaphore_Information,

举例

```
NU_SEMAPHORE Semaphore;
STATUS status;

/*删除 Semaphore 指定的信号量控制块。假设“Semaphore”在先前已经调用 Nucleus
PLUS 服务 NU_Create_Semaphore 创建*/

status = NU_Delete_Semaphore(&Semaphore);

/*这里 status 表示服务请求成功与否*/
```

NU_Delete_Task

函数原型

```
STATUS NU_Delete_Task(NU_TASK *Task);
```

描述

此服务删除一个先前定义的任务。参数 Task 确定需要删除的任务。在这个任务上挂起的任务恢复时返回适当的错误状态。在删除期间和之后，应用程序必须防止任务的使用。

返回值

此项服务完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_TASK	表示任务指针非法
NU_INVALID_DELETE	表示任务处于一个未完成或未终止状态

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务通信服务

相关

NU_Create_Task,NU_Established_Tasks,NU_Task_Pointers,
NU_Task_Information,NU_Reset_Task

举例

```
NU_TASK Task;
STATUS status;

/*删除 Task 指定的任务控制块。假设“Task”在先前已经调用 Nucleus PLUS 服务
NU_Create_Task 创建*/

status = NU_Delete_Task(&Task);

/*这里 status 表示服务请求成功与否*/
```

NU_Delete_Timer

函数原型

```
STATUS NU_Delete_Timer(NU_TIMER *Timer);
```

描述

此服务删除一个先前定义的应用程序定时器。参数 Timer 确定需要删除的定时器。注：指定的定时器必须先于此项服务请求时禁止。在删除期间和之后，应用程序必须防止定时器的使用。

返回值

此项服务完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_TIMER	表示定时器指针非法

NU_NOT_DISABLED 表示指定的定时器不能禁止

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

定时器服务

相关

NU_Create_Timer,NU_Established_Timers,NU_Timer_Pointers,
NU_Timer_Information,NU_Reset_Timer

举例

```
NU_TIMER Timer;
STATUS status;

/*删除 Timer 指定的定时器控制块。假设“Timer”在先前已经调用 Nucleus PLUS 服务 NU_Create_Timer 创建*/

status = NU_Delete_Timer(&Timer);

                    /*这里 status 表示服务请求成功与否*/
```

NU_Disable_Histroy_Saving

函数原型

```
VOID NU_Disable_Histroy_Saving(VOID);
```

描述

此项服务禁止内部历史存储。这个服务经常用于在准备检查历史纪录时禁止历史存储。

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,Task

类别

开发服务

相关

NU_Enable_History_Saving,NU_Retrieve_History_Entry

举例

```
/*禁止历史存储*/
NU_Disable_History_Saving();
```

NU_Driver_Pointers

函数原型

```
UNSIGNED NU_Driver_Pointers(  NU_DRIVER **pointer_list,
                              UNSIGNED maximum_pointers);
```

描述

此项服务为系统中所有已建立的 I/O 驱动器建立一个连续的指针列表。注：已经删除的 I/O

驱动器不再被认为已建立。参数 pointer_list 为建立指针列表的位置，maximum_pointers 表示表的最大尺寸。服务返回列表的指针实际数量。另外，列表按照从最旧到最新排序。

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

I/O 驱动器服务

相关

NU_Create_Driver,NU_Delete_Driver,NU_Established_Drivers

举例

```
/*定义一个可以容纳 20 个 I/O 驱动器指针的阵列*/

NU_DRIVER *Pointer_Array[20];
UNSIGNED number;

/*获得当前激活的 I/O 驱动器列表（最大为 20）*/

number = NU_Driver_Pointers(&Pointer_Array[0],20);

/*这里，number 包含列表中指针的实际数量*/
```

NU_Enable_Histroy_Saving

函数原型

```
VOID NU_Enable_History_Saving(VOID);
```

描述

此项服务使能内部历史存储。

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

开发服务

相关

NU_Disable_History_Saving,NU_Retrieve_History_Entry,
NU_Make_History_Entry

举例

```
/*使能内部历史存储*/
NU_Enable_History_Saving();
```

NU_Established_Drivers

函数原型

```
UNSIGNED NU_Established_Drivers(VOID);
```

描述

此项服务返回已建立的 I/O 驱动器数量。所有创建的 I/O 驱动器都被认为已经建立。删除的 I/O 驱动器不在被认为是已经建立。

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

I/O 驱动器服务

相关

NU_Create_Driver,NU_Delete_Driver,NU_Driver_Pointers

举例

```
UNSIGNED total_drivers;
/*获得 I/O 驱动器总数*/
total_drivers = NU_Established_Drivers();
```

NU_Established_Event_Groups

函数原型

```
UNSIGNED NU_Established_Event_Groups(VOID);
```

描述

此项服务返回已建立的事件标志集数量。所有创建的事件标志集都被认为已经建立。删除的事件标志集不再被认为是已经建立。

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务同步服务

相关

NU_Create_Event_Group,NU_Delete_Event_Group,
NU_Event_Group_Pointers,NU_Event_Group_Information

举例

```
UNSIGNED total_Event_Groups;
/*获得事件标志集总数*/
      total_Event_Groups = NU_Established_Event_Groups();
```

NU_Established_HISRs

函数原型

```
UNSIGNED NU_Established_HISRs(VOID);
```

描述

此项服务返回已建立的 HISR 数量。所有创建的 HISR 都被认为已经建立。删除的 HISR 不在被认为是已经建立。

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

中断服务

相关

NU_Create_HISR,NU_Delete_HISR,
NU_HISR_Pointers,NU_HISR_Information

举例

```
UNSIGNED total_HISRs;
/*获得 HISR 总数*/
            total_HISRs = NU_Established_HISRs();
```

NU_Established_Mailboxes

函数原型

```
UNSIGNED NU_Established_Mailboxes(VOID);
```

描述

此项服务返回已指定的邮箱数量。所有创建的邮箱都被认为已经建立。删除的邮箱不在被认为是已经建立。

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务通信服务

相关

NU_Create_Mailbox,NU_Delete_Mailbox,
NU_Mailbox_Pointers,NU_Mailbox_Information

举例

```
UNSIGNED total_Mailboxes;
/*获得邮箱总数*/
         total_Mailboxes = NU_Established_Mailboxes();
```

NU_Established_Memory_Pools

函数原型

```
UNSIGNED NU_Established_Memory_Pools(VOID);
```

描述

此项服务返回已指定的动态内存池数量。所有创建的动态内存池都被认为已经建立。删除的动态内存池不在被认为是已经建立。

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

内存服务

相关

NU_Create_Memory_Pool,NU_Delete_Memory_Pool,
NU_Memory_Pool_Pointers,NU_Memory_Pool_Information

举例

```
UNSIGNED total_Memory_Pools;
/*获得动态内存池总数*/
      total_Memory_Pools = NU_Established_Memory_Pools();
```

NU_Established_Partition_Pools

函数原型

```
UNSIGNED NU_Established_Partition_Pools(VOID);
```

描述

此项服务返回已指定的分区内存池数量。所有创建的分区内存池都被认为已经建立。删除的分区内存池不在被认为是已经建立。

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

内存服务

相关

NU_Create_Partition_Pool,NU_Delete_Partition_Pool,
NU_Partition_Pool_Pointers,NU_Partition_Pool_Information

举例

```
UNSIGNED total_Partition_Pools;
/*获得分区内存池总数*/
   total_Partition_Pools = NU_Established_Partition_Pools();
```

NU_Established_Pipes

函数原型

```
UNSIGNED NU_Established_Pipes(VOID);
```

描述

此项服务返回已指定的管道数量。所有创建的管道都被认为已经建立。删除的管道不在被认为是已经建立。

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务通信服务

相关

NU_Create_Pipe,NU_Delete_Pipe,
NU_Pipe_Pointers,NU_Pipe_Information,NU_Reset_Pipe

举例

```
UNSIGNED total_Pipes;
/*获得管道总数*/
            total_Pipes = NU_Established_Pipes();
```

NU_Established_Queues

函数原型

```
UNSIGNED NU_Established_Queues(VOID);
```

描述

此项服务返回已指定的队列数量。所有创建的队列都被认为已经建立。删除的队列不在被认为是已经建立。

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务通信服务

相关

NU_Create_Queue,NU_Delete_Queue,
NU_Queue_Pointers,NU_Queue_Information,NU_Reset_Queue

举例

```
UNSIGNED total_Queues;
/*获得队列总数*/
            total_Queues = NU_Established_Queues();
```

NU_Established_Semaphores

函数原型

```
UNSIGNED NU_Established_Semaphores(VOID);
```

描述

此项服务返回已指定的信号量数量。所有创建的信号量都被认为已经建立。删除的信号量不在被认为是已经建立。

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务通信服务

相关

NU_Create_Semaphore,NU_Delete_Semaphore,
NU_Semaphore_Pointers,NU_Semaphore_Information

举例

```
UNSIGNED total_Semaphores;
/*获得信号量总数*/
        total_Semaphores = NU_Established_Semaphores();
```

NU_Established_Tasks

函数原型

```
UNSIGNED NU_Established_Tasks(VOID);
```

描述

此项服务返回已指定的任务数量。所有创建的任务都被认为已经建立。删除的任务不在被认为是已经建立。

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务控制服务

相关

NU_Create_Task,NU_Delete_Task,
NU_Task_Pointers,NU_Task_Information,NU_Reset_Task

举例

```
UNSIGNED total_Tasks;
/*获得任务总数*/
            total_Tasks = NU_Established_Tasks();
```

NU_Established_Timers

函数原型

```
UNSIGNED NU_Established_Timers(VOID);
```

描述

此项服务返回已指定的定时器数量。所有创建的定时器都被认为已经建立。删除的定时器不在被认为是已经建立。

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

定时器服务

相关

NU_Create_Timer,NU_Delete_Timer,
NU_Timer_Pointers,NU_Timer_Information,NU_Reset_Timer

举例

```
UNSIGNED total_Timers;
/*获得定时器总数*/
            total_Timers = NU_Established_Timers();
```

NU_Event_Group_Information

函数原型

```
STATUS NU_Event_Group_Information(   NU_EVENT_GROUP *group,
                                     CHAR *name,
                                     UNSIGNED *event_flags,
                                     UNSIGNED *tasks_waiting,
                                     NU_TASK **first_task);
```

描述

此服务返回关于指定事件标志集的多种信息。参数详细定义如下。

参数	含义
group	事件标志集指针
name	事件标志集 8 字符区域指针
event_flags	保留当前事件标志集的变量指针
tasks-waiting	保留等待事件标志集任务数的变量指针
first_task	任务指针的指针。第一个挂起任务的指针放在这个任务指针里。

返回值

服务完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_GROUP	表示事件标志集指针无效

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务同步服务

相关

NU_Create_Event_Group,NU_Delete_Event_Group
NU_Established_Event_Groups,NU_Event_Group_Pointers

举例

```
NU_EVENT_GROUP Group;
CHAR Group_name[8];
UNSIGNED event_flags;
UNSIGNED tasks_suspended;
NU_TASK *first_task;
STATUS status;

/*获得 Group 指定的事件标志集控制块信息。假设“Group”在先前已经调用 Nucleus PLUS 服务 NU_Create_Event_Group 创建*/

status = NU_Event_Group_Information( &Group,
                                     group_name,
                                     &event_flags,
```

```
                                        &tasks_suspended,
                                        &first_task);
        /*如果 status 等于 NU_SUCCESS，其他的信息是精确的*/
```

NU_Event_Group_Pointers

函数原型

```
UNSIGNED NU_Event_Group_Pointers( NU_EVENT_GROUP *pointer_list,
                                  UNSIGNED maximum_Pointers);
```

描述

服务建立系统中所有已建立的事件标志集的指针连续列表。注：已经删除的事件标志集不再被认为是已建立。参数 pointer_list 指向建立指针列表的位置，maximum_pointers 表示列表的最大尺寸。这个服务返回列表的实际大小值。另外，列表按最旧到最新成员排序。

任务更改

no

允许调用

Application_Initialize,HISR,Signal Handler,task

相关

NU_Create_Event_Group,NU_Delete_Event_Group,
NU_Established_Event_Groups,NU_Event_Group_Information

举例

```
/*定义一个能容纳 20 个事件标志集指针的阵列*/
NU_EVENT_GROUP *Pointer_Array[20];
UNSIGNED number;

/*获得当前活动的事件标志集指针列表（最大值为 20）*/

number = NU_Event_Group_Pointers(&Pointer_Array[0],20);

/*这里，number 包含了列表中指针数量*/
```

NU_HISR_Information

函数原型

```
STATUS NU_HISR_Information(   NU_HISR *hisr,
                              Char *name,
                              UNSIGNED *scheduled_cout,
                              DATA ELEMENT *priority,
                              VOID **stack_base,
                              UNSIGNED *stack_size,
                              UNSIGNED *minimum_stack);
```

描述

此项服务返回关于指定 HISR 的多种信息。参数详细定义如下：

参数	含义
hisr	HISR 指针
name	HISR 名称 8 字符目标区指针

Scheduled_count	保存这个 HISR 被调度的总次数的变量指针
Priority	保存 HISR 优先级的变量指针
Stack_base	保存原堆栈指针的指针。
Stack_size	保存 HISR 的堆栈总尺寸的标量指针
Minimum_stack	HISR 运行期间保存有效堆栈检测最小值的指针

返回值

服务完成状态如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_HISR	表示 HISR 指针非法

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

中断服务

相关

NU_Create_HISR,NU_Delete_HISR,NU_Established_HISRs,
NU_HISR_Pointers

举例

```
NU_HISR Hisr;
CHAR hisr_name[8];
UNSIGNED activations;
DATA ELEMENT priority;
VOID *stack_base;
UNSIGNED stack_size;
UNSIGNED minimu_stack;
STATUS statsu;

/*获得关于“Hisr”指定 HISR 控制块的信息。假设“Hisr”在先前已经调用 Nucleus PLUS
服务 NU_Create_HISR 创建*/
status = NU_HISR_Information( &Hisr,
                              hisr_name,
                              &activations,
                              &priority,
                              &stack_base,
                              &stack_size,
                              &maximum_stack);
/*如果 status 为 NU_SUCCESS,其他信息都是精确的*/
```

NU_HISR_Pointers

函数原型

```
UNSIGNED NU_HISR_Pointers( NU_HISR **pointer_list,
                           UNSIGNED maximum_pointers);
```

描述

此项服务建立一个连续的系统中所有已建立的 HISRs 指针的列表。注：已经删除 HISRs 不再被任务已建立。参数 pointer_list 指向用于建立指针列表的位置，maximum_pointers 表示列表的最大尺寸。这项服务返回列表指针的数量。另外，列表按最新到最旧成员排序。

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

中断服务

相关

NU_Create_HISR,NU_Delete_HISR,NU_Established_HISRs,
NU_HISR_Information

举例

```
/*定义一个能容纳 20 个 HISR 指针的阵列*/
NU_HISR *Pointer Array[20];
UNSIGNED number;

/*获得当前激活的 HISR 指针列表（最大为 20）*/
number = NU_HISR_Pointers(&Pointer_Array[0],20);
/*这里，number 包含了列表指针实际数值*/
```

NU_License_Information

函数原型

```
CHAR *NU_License_Information(VOID);
```

描述

此服务返回一个包含消费者序列号和一个简单产品描述的字符串指针。字符串为 ASCII 格式，以 NULL 结束。

任务更改

NO

允许调用

Application_Initialize,LISR,HISR,Signal Handler,task

类别

开发服务

相关

NU_Release_Information

举例

```
CHAR *license_string;
/*获得用户许可字符串指针*/
license_string = NU_License_Information();
```

NU_Local_Control_Interrupts

函数原型

```
INT NU_Local_Control_Interrupts(INT new_level);
```

描述

此服务根据 new_level 指定的值使能或禁止中断。中断被使能和禁止依赖于子程序方式。因此，一个被这项服务禁止的中断保持禁止一直到 new_level 的下一个返回值是由处理器决定。然而，NU_DISABLE_INTERRUPTS 和 NU_ENABLE_INTERRUPTS 分别可以用来禁止和使能所有中断。

返回值

此项服务返回使能中断的先前的等级。

任务更改

NO

允许调用

LISR,HSIR,Signal Handler,task

类别

中断服务

相关

NU_Setup_Vector,NU_Register_LISR,NU_Create_HISR,
NU_Delete_HISR

举例

```
INT old_level;/*旧中断级别*/

/*临时禁止所有中断*/
old_level = NU_Local_Control_Interrupts(NU_DISABLE_INTERRUPTS);

return;   /*或中断返回*/
```

NU_Mailbox_Information

函数原型

```
STATUS NU_Mailbox_Information(NU_MAILBOX *mailbox,
                              CHAR *name,
                              OPTION *suspend_type,
                              DATA ELEMENT *message_present,
                              UNSIGNED *tasks_waiting,
                              NU_TASK **first_task);
```

描述

此项服务返回有关指定邮箱的多种信息。参数详细定义如下：

参数	含义
mailbox	邮箱指针
name	邮箱名 8 字符目标区域指针
Suspend_type	保存任务挂起类型的变量指针。有效任务挂起类型为：NU_FIFO 和 NU_PRIORITY
Message_present	如果一个消息在邮箱出现，NU_TRUE 值放入这个参数指向的变量。另外，如果邮箱为空，NU_FALSE 值放入这个变量。
Tasks_waiting	保存等待邮箱的任务数量的变量指针

First_task	指向任务指针的指针。第一个挂起的任务指针放入这个任务指针。

返回值

服务完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_MAILBOX	表示邮箱指针无效

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务通讯服务

相关

NU_Create_Mailbox,NU_Delete_Mailbox,
NU_Established_Mailbox,NU_Mailbox_Pointers

举例

```
NU_MAILBOX Mailbox;
CHAR mailbox_name[8];
OPTION suspend_type;
DATA ELEMENT message_present;
UNSIGNED task_suspended;
NU_TASK *first_task;
STATUS status;

/*获得“Mailbox”指定邮箱控制块的信息。假设“Mailbox”在先前已经调用 Nucleus
PLUS 服务 NU_Create_Mailbox 创建*/

status = NU_Mailbox_Information( &Mailbox,
                                 mailbox_name,
                                 &suspend_type,
                                 &message_present,
                                 &tasks_suspended,
                                 &first_task);
/*如果 status 等于 NU_SUCCESS,其他信息是精确的。*/
```

NU_Mailbox_Pointers

函数原型

```
UNSIGNED NU_Mailbox_Pointers( NU_MAILBOX **pointer_list,
                              UNSIGNED maximum_pointers);
```

描述

此服务建立一个系统中所有已建立的邮箱的指针连续列表。注：已经删除的邮箱不再被认为已经建立。参数 pointer_list 指向用于建立指针列表的位置，maximum_pointers 表示列表最大尺寸。这个服务返回列表指针实际数量。另外，列表按最旧到最新成员排序。

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务通信服务

相关

NU_Create_Mailbox,NU_Delete_Mailbox
NU_Established_Mailboxes,NU_Mailbox_Information

举例

```
/*定义一个能容纳 20 个邮箱指针的阵列*/
NU_MAILBOX *Pointer_Array[20];
UNSIGNED number;
/*获得当前激活的邮箱指针列表（最大 20 个）*/
number = NU_Mailbox_Pointers(&Pointer_Array[0],20);

/*这里，number 包含了列表中指针的实际数量*/
```

NU_Make_History_Entry

函数原型

```
VOID NU_Make_History_Entry(   DATA ELEMENT id,
                              UNSIGNED param1,
                              UNSIGNED param2,
                              UNSIGNED param3);
```

描述

如果历史纪录性能被使能，此服务在系统历史记录中制造一个入口。否则，这个服务什么都不做。

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

开发服务

相关

NU_Enable_History_Saving,NU_Disable_History_Saving,
NU_Retrieve_History_Entry

举例

```
/*在历史纪录中建立一个入口，参数为 1，2，3。*/
NU_Make_History_Entry(1,2,3);
```

NU_Memory_Pool_Information

函数原型

```
STATUS NU_Memory_Pool_Information(   UN_MEMORY_POOL *pool,
                                     CHAR *name,
                                     VOID **start_address,
                                     UNSIGNED * pool_size,
                                     UNSIGNED * min_allocation,
                                     UNSIGNED *available,
                                     OPTION *suspend_type,
                                     UNSIGNED *tasks_waiting,
                                     NU_TASK **first_task);
```

描述

此服务返回关于指定动态内存池的多种信息。参数详细定义如下：

参数	含义
pool	动态内存池指针
name	保存动态内存池名字的 8 字符目标区域指针
start_address	指向保存动态内存池起始地址的内存指针
pool_size	指向保存动态内存池字节数的变量的指针
min_allocation	指向保存池中每个分配的最小字节数的变量的指针
available	指向保存池中可用字节数的变量的指针
suspend_type	指向保存任务挂起类型的变量的指针。有效任务挂起类型为：NU_FIFO 和 NU_PRIORITY
task_waiting	指向保存等待动态内存池的任务数量的变量的指针
first_task	指向任务指针的指针。第一个挂起的任务指针放入这个任务指针

返回值

此服务完成状态定义如下：

状态	含义
NU_SUCCCESS	表示服务成功完成
NU_INVALID_POOL	表示动态内存池指针无效

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

内存服务

相关

NU_Create_Memory_Pool,NU_Delete_Memory_Pool,
NU_Established_Memory_Pools,NU_Memory_Pool_Pointers

举例

```
NU_MEMORY_POOL Pool;
CHAR pool_name[8];
VOID *start_address;
UNSIGNED pool_size;
UNSIGNED min_allocation;
```

```
UNSIGNED available;
OPTION suspend_type;
UNSIGNED tasks_suspend;
NU_TASK *first_task;
STATUS status;

/*获得有关“Pool”指定的内存控制块的信息。假设“Pool”在先前已经调用 Nucleus PLUS
服务 NU_Create_Memory_Pool 创建*/

status = NU_Memory_Pool_Information( &Pool,
                                     pool_name,
                                     &start_address,
                                     &pool_size,
                                     &min_allocation,
                                     &available,
                                     &suspend_type,
                                     &tasks_suspended,
                                     &first_task);
/*如果 status 等于 NU_SUCCESS,其他信息是精确的*/
```

NU_Memory_Pool_Pointers

函数原型

```
UNSIGNED NU_Memory_Pool_Pointers( NU_MEMORY_POOL **pointer_list,
                                  UNSIGNED maximum_pointers);
```

描述

此服务建立一个系统中所有已建立的内存池的指针连续列表。注：已经删除的内存池不再被认为已经建立。参数 pointer_list 指向用于建立指针列表的位置，maximum_pointers 表示列表最大尺寸。这个服务返回列表指针实际数量。另外，列表按最旧到最新成员排序。

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

内存服务

相关

NU_Create_Memory_Pool,NU_Delete_Memory_Pool
NU_Established_Memory_Pools,NU_Memory_Pool_Information

举例

```
/*定义一个能容纳 20 个内存池指针的阵列*/
NU_MEMORY_POOL *Pointer_Array[20];
UNSIGNED number;
/*获得当前激活的内存池指针列表（最大 20 个）*/
number = NU_Memory_pool_Pointers(&Pointer_Array[0],20);
```

/*这里，number 包含了列表中指针的实际数量*/

NU_Obtain_Semaphore

函数原型

```
STATUS NU_Obtain_Semaphore(    NU_SEMAPHORE *semaphore,
                               UNSIGNED suspend);
```

描述

此项服务获得指定信号量的实例。一旦实例通过内部计数器实现，获得一个信号量转化为消耗该信号量，内部计数器减一。如果信号量计数器在这个调用之前为 0，服务不能满足。服务参数详细定义如下：

参数	含义
semaphore	需要获得的信号量的指针。
suspend	如果信号量不能被获得（当前为 0），指定正在调用的任务是否挂起。

下列挂起类型有效：

NU_NO_SUSPEND

不管请求是否满足，服务立即返回。注：如果服务从一个非任务线程被调用，这是唯一有效的配置。

NU_SUSPEND

正在调用的任务挂起直到信号量可以被获得。

时间间隔值（1～4294967293）

正在调用任务挂起知道信号量可以被获得或者指定的定时器节拍值到时。

返回值

此项服务完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_SEMAPHORE	表示信号量指针无效
NU_INVALID_SUSPEND	表示试图从一个非任务线程挂起
NU_UNAVAILABLE	表示信号量难以获得
NU_TIMEOUT	表示甚至在挂起指定的时间间隔后信号量仍然难以获得
NU_SEMAPHORE_DELETED	在任务挂起期间信号量被删除

任务更改

Yes

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务同步服务

相关

NU_Release_Semaphore,NU_Semaphore_Information

举例

```
NU_SEMAPHORE Semaphore;
STATUS status;
```

/*获得“Semaphore”指定信号量控制块的一个实例。如果信号量难以获得，挂起最大 20 个定时器时钟节拍。注：在同一信号量上多任务挂起顺序在信号量创建时确定。假设

“Semaphore”在先前已经调用 Nucleus PLUS 服务 NU_Create_Semaphore 创建*/

```
status = NU_Obtain_Semaphore(&Semaphore,20);

/*这里，status 表示服务请求成功与否*/
```

NU_Partition_Pool_Information

函数原型

```
STATUS NU_Partition_Pool_Information(NU_PARTITION_POOL *pool,
                                     CHAR *name,
                                     VOID **start_address,
                                     UNSIGNED *pool_size,
                                     UNSIGNED *partition_size,
                                     UNSIGNED *available,
                                     OPTION *suspend_type,
                                     UNSIGNED *tasks_waiting,
                                     NU_TASK **first_task);
```

描述

此服务返回关于指定分区内存池的多种信息。参数的详细定义如下：

参数	含义
pool	分区池指针
name	保存分区池名称的 8 字符目标区域指针
start_address	指向保存分区池起始地址内存指针的指针
pool_size	指向保存分区池总字节数的变量的指针
partition_size	指向保存每个内存分区的字节数的变量的指针
Available	指向保存池内可用分区数的变量的指针
allocated	指向保存已分配池分区的数量的变量的指针
suspend_type	指向保存任务挂起类型的变量的指针。有效的任务挂起类型为：NU_FIFO 和 NU_PRIORITY
tasks_waiting	指向保存等待分区池的任务数的变量的指针
first_task	执行任务指针的指针。第一个挂起任务的指针放入这个任务指针

返回值

此项服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_POOL	表示分区池指针无效

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

内存服务

相关

NU_Create_Partition_Pool,NU_Delete_Partition_Pool,

NU_Established_Partition_Pools,NU_Partition_Pool_Pointers

举例

```
NU_PARTITION_POOL Pool;
CHAR pool_name[8];
VOID *start_address;
UNSIGNED pool_size;
UNSIGNED partition_size;
UNSIGNED available;
UNSIGNED allocated;
UNSIGNED suspend_type;
UNSIGNED tasks_waiting;
UNSIGNED first_task;
STATUS status;

/*获得有关“Pool”指定的分区池控制块的信息。假设“Pool”在先前已经调用 Nucleus
PLUS 服务 NU_Create_Partition_Pool 创建*/

status = NU_Partition_Pool_Information( &Pool,
                                        pool_name,
                                        &start_address,
                                        &pool_size,
                                        &partition_size,
                                        &available,
                                        &allocated,
                                        &suspend_type,
                                        &tasks_suspended,
                                        &first_task);

/*如果 status 为 NU_SUCCESS,其他的信息是精确的*/
```

NU_Partition_Pool_Pointers

函数原型

```
UNSIGNED NU_Partition_Pool_Pointers(
                                    NU_PARTITION_POOL **pointer_list,
                                    UNSIGNED maximum_pointers);
```

描述

此项服务为系统中已经建立的内存分区池建立一个连续的指针列表，注：已经删除的内存分区池不再被认为已经建立。参数 pointer_list 指向用于建立指针列表的位置。Maximum_pointers 表示列表的最大尺寸。这个服务返回列表中指针的实际数量。另外，列表按最旧到最新成员顺序排序。

任务更改

NO

允许调用

Application_Initalize,HISR,Signal Handler,task

类别

内存服务

相关

NU_Create_Partition_Pool,NU_Delete_Partition_Pool,
NU_Established_Partition_Pools,NU_Partition_Pool_Information

举例

```
/*定义一个可以容纳 20 个内存分区池指针的阵列*/

NU_PARTITION_POOL *Pointer_Array[20];
UNSIGNED number;

/*获得当前激活的内存分区池指针列表（最大值为 20）*/

status = NU_Partititon_Pool_Pointers(&Pointer_Array[0],20);

/*这里，number 包含列表指针实际数量*/
```

NU_Pipe_Information

函数原型

```
STATUS NU_Pipe_Information(NU_PIPE *pipe,
                          CHAR *name,
                          VOID **start_address,
                          UNSIGNED *pipe_size,
                          UNSIGNED *available,
                          UNSIGNED *messages,
                          UNSIGNED *message_type,
                          UNSIGNED *suspend_size,
                          UNSIGNED *tasks_waiting,
                          NU_TASK **first_task);
```

描述

此项服务返回有关指定消息通讯管道的多种信息。此项服务的参数定义如下：

参数	含义
pipe	消息管道指针
name	保存消息管道名称的 8 字符目标区域的指针
start_address	指向保存管道其实地址的内存指针的指针
pipe_size	指向保存管道总字节数的变量的指针
available	指向保存管道可用字节数的变量的指针
messages	指向保存管道当前消息数量的变量的指针
message_type	指向保存管道支持的消息类型的变量的指针。有效消息类型为：NU_FIXED_SIZE 和 NU_VARIABLE_SIZE
message_size	指向保存每个消息字节数的变量的指针
suspend_type	指向保存任务挂起类型的变量的指针。有效任务挂起类型为：NU_FIFO 和 NU_PRIORITY
tasks_waiting	指向保存等待管道的任务数的变量的指针

first_task 指向任务指针的指针。第一个挂起的任务的指针放入这个任务指针

返回值

此项服务的完成状态定义如下：

参数	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_PIPE	表示管道指针无效

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务通讯服务

相关

NU_Create_Pipe,NU_Delete_Pipe,NU_Established_Pipes,
NU_Pipe_Pointers,NU_Reset_Pipe

举例

```
NU_PIPE Pipe;
CHAR pipe_name[8];
VOID *start_address;
UNSIGNED pipe_size;
UNSIGNED available;
UNSIGNED messages;
OPTION message_type;
UNSIGNED message_size;
OPTION suspend_type;
UNSIGNED tasks_suspended;
NU_TASK *first_task;
STATUS status;

/*获得 pipe 指定的消息管道控制块的信息。假设“Pipe“在先前已经调用 Nucleus PLUS
服务 NU_Create_Pipe 创建*/

status = NU_Pipe_Information( &Pipe,
                              pipe_name,
                              &start_address,
                              &pipe_size,
                              &available,
                              &messages,
                              &message_type,
                              &message_size,
                              &suspend_type,
                              &tasks_suspended,
                              &first_task);
/*如果 status 为 NU_SUCCESS,其他的信息就是精确的*/
```

NU_Pipe_Pointers

函数原型

```
UNSIGNED NU_Pipe_Pointers(
                                    NU_PIPE **pointer_list,
                                    UNSIGNED maximum_pointers);
```

描述

此项服务为系统中已经建立的管道建立一个连续的指针列表，注：已经删除的管道不再被认为已经建立。参数 pointer_list 指向用于建立指针列表的位置。Maximum_pointers 表示列表的最大尺寸。这个服务返回列表中指针的实际数量。另外，列表按最旧到最新成员顺序排序。

任务更改

NO

允许调用

Application_Initalize,HISR,Signal Handler,task

类别

任务通信服务

相关

NU_Create_Pipe,NU_Delete_Pipe,NU_Established_Pipes,
NU_Pipe_Information,NU_Reset_Pipe

举例

```
/*定义一个可以容纳 20 个管道指针的阵列*/

NU_PIPE *Pointer_Array[20];
UNSIGNED number;

/*获得当前激活的管道指针列表（最大值为 20）*/

status = NU_Partititon_Pool_Pointers(&Pointer_Array[0],20);

/*这里，number 包含列表指针实际数量*/
```

NU_Protect

函数原型

```
VOID NU_Protect(NU_PROTECT *protect_struct);
```

描述

此项服务启动临界数据结构的原始保护。因为 I/O 驱动器经常必须保护不被任务和 HISR 组件同步访问，比较典型的是这个服务为 I/O 驱动器内的数据结构保护而保留。正常的任务同步必须使用任务同步服务来实现。

注意下面的约束：

- ✧ 保护结构必须被应用程序初始化为零；
- ✧ 在服务被调用之后，只有下面的 Nucleus PLUS 服务有效：NU_Unprotect,NU_Suspend_Driver,NU_Resume_Driver。NU_Protect 嵌套调用是不允许的。

任务更改

Yes

允许调用

HISR,task,Signal Handler

类别

I/O 驱动器服务

相关

NU_Unprotect,NU_Suspend_Driver

举例

```
NU_PROTECT Protect_Struct;

/*初始化保护结构为"Protect_Struct"的临界段的保护。注：保护结构必须在应用之前清除*/

NU_Protect(&Protect_Struct);
```

NU_Queue_Information

函数原型

```
STATUS NU_Queue_Information(NU_QUEUE *Queue,
                           CHAR *name,
                           VOID *start_address,
                           UNSIGNED *Queue_size,
                           UNSIGNED *available,
                           UNSIGNED *messages,
                           UNSIGNED *message_type,
                           UNSIGNED *suspend_size,
                           UNSIGNED *tasks_waiting,
                           NU_TASK **first_task);
```

描述

此项服务返回有关指定消息通讯队列的多种信息。此项服务的参数定义如下：

参数	含义
Queue	消息队列指针
name	保存消息队列名称的 8 字符目标区域的指针
start_address	指向保存队列其实地址的内存指针的指针
Queue_size	指向保存队列总字节数的变量的指针
available	指向保存队列可用字节数的变量的指针
messages	指向保存队列当前消息数量的变量的指针
message_type	指向保存队列支持的消息类型的变量的指针。有效消息类型为：NU_FIXED_SIZE 和 NU_VARIABLE_SIZE
message_size	指向保存每个队列消息中 UNSIGNED 数据类型数的变量的指针。如果对列支持变长消息，这个值就是最大消息尺寸。
suspend_type	指向保存任务挂起类型的变量的指针。有效任务挂起类型为：NU_FIFO 和 NU_PRIORITY
tasks_waiting	指向保存等待队列的任务数的变量的指针

first_task 指向任务指针的指针。第一个挂起的任务的指针放入这个任务指针

返回值

此项服务的完成状态定义如下：

参数	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_QUEUE	表示队列指针无效

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务通讯服务

相关

NU_Create_Queue,NU_Delete_Queue,NU_Established_Queues,
NU_Queue_Pointers,NU_Reset_Queue

举例

```
NU_QUEUE Queue;
CHAR Queue_name[8];
VOID *start_address;
UNSIGNED Queue_size;
UNSIGNED available;
UNSIGNED messages;
OPTION message_type;
UNSIGNED message_size;
OPTION suspend_type;
UNSIGNED tasks_suspended;
NU_TASK *first_task;
STATUS status;

/*获得 Queue 指定的消息队列控制块的信息。假设“Queue“在先前已经调用 Nucleus
PLUS 服务 NU_Create_Queue 创建*/

status = NU_Queue_Information(    &Queue,
                                  Queue_name,
                                  &start_address,
                                  &Queue_size,
                                  &available,
                                  &messages,
                                  &message_type,
                                  &message_size,
                                  &suspend_type,
                                  &tasks_suspended,
                                  &first_task);
/*如果 status 为 NU_SUCCESS,其他的信息就是精确的*/
```

NU_Queue_Pointers

函数原型

```
UNSIGNED NU_Queue_Pointers(
                                    NU_QUEUE **pointer_list,
                                    UNSIGNED maximum_pointers);
```

描述

此项服务为系统中已经建立的队列建立一个连续的指针列表，注：已经删除的队列不再被认为已经建立。参数 pointer_list 指向用于建立指针列表的位置。Maximum_pointers 表示列表的最大尺寸。这个服务返回列表中指针的实际数量。另外，列表按最旧到最新成员顺序排序。

任务更改

NO

允许调用

Application_Initalize,HISR,Signal Handler,task

类别

任务通信服务

相关

NU_Create_Queue,NU_Delete_Queue,NU_Established_Queues,
NU_Queue_Information,NU_Reset_Queue

举例

```
/*定义一个可以容纳 20 个队列指针的阵列*/

NU_QUEUE *Pointer_Array[20];
UNSIGNED number;

/*获得当前激活的队列指针列表（最大值为 20）*/

status = NU_Partititon_Pool_Pointers(&Pointer_Array[0],20);

/*这里，number 包含列表指针实际数量*/
```

NU_Receive_From_Mailbox

函数原型

```
STATUS NU_Receive_From_Mailbox(   NU_MAILBOX *mailbox,
                                  VOID *message,
                                  UNSIGNED suspend);
```

描述

此项服务从指定的邮箱中接收消息。如果邮箱包含一个消息，立即从邮箱里移除且拷贝到指定的位置。邮箱消息尺寸等于 4 个 UNSIGNED 数据类型。此项服务的参数详细定义如下：

参数	含义
mailbox	邮箱指针
message	消息目的指针。注：消息目标必须至少为 4 个 UNSIGNED 数据类型大小。
suspend	如果邮箱为空，指定是否挂起正在调用的任务。

下面是挂起类型的有效选项：

NU_NO_SUSPEND

不管请求是否满足，服务立即返回。注：如果服务被非任务线程调用，这是唯一有效的配置。

NU_SUSPEND

正在调用的任务挂起知道消息有效。

时间间隔值（1~4294967293）

正在调用的服务挂起知道消息有效或知道指定的定时器节拍数到时。

返回值

此项服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_MAILBOX	表示邮箱指针无效
NU_INVALID_POINTER	表示消息指针为空
NU_INVALID_SUSPEND	表示试图从非任务线程挂起
NU_MAILBOX_EMPTY	表示邮箱为空
NU_TIMEOUT	表示尽管在挂起到指定超时值之后，邮箱仍然为空
NU_MAILBOX_DELETED	在任务挂起期间邮箱被删除
NU_MAILBOX_RESET	任务挂起期间任务被复位

任务更改

Yes

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务通信服务

相关

NU_Broadcast_To_Mailbox,NU_Send_To_Mailbox,
NU_Mailbox_Information

举例

```
NU_MAILBOX Mailbox;
UNSIGNED message[4];
STATUS status;

/*从”Mailbox”指定的邮箱控制块中接收一个消息。如果邮箱为空，挂起 20 个定时器节拍。注：在同一个邮箱上多任务挂起的顺序在邮箱创建时决定。假设”Mailbox”在先前已经调用 Nucleus PLUS 服务 NU_Create_Mailbox 创建*/

status = NU_Receive_From_Mailbox(&Mailbox,&message[0],20);

/*这里 status 表示服务请求成功与否。如果成功，”message”包含了接收的邮箱消息*/
```

NU_Receive_From_Pipe

函数原型

```
STATUS NU_Receive_From_Pipe(  NU_PIPE *Pipe,
                              VOID *message,
```

```
                                UNSIGNED size,
                                UNSIGNED *actual_size,
                                UNSIGNED suspend);
```

描述

此项服务从指定的管道中接收一个消息。如果管道包含一个或多个消息，立即从管道里移除前面的消息且拷贝到指定的位置。根据管道支持的消息类型，管道消息由定长和不定长字节组成。此项服务的参数详细定义如下：

参数	含义
Pipe	管道指针
message	消息目的指针。注：消息目标必须足够大能容纳 size 的字节数。
size	指定消息内的字节数。这个值必须对应于在管道创建时定义的消息尺寸。只适用于定义定长字节的管道；否则忽略。
actual_size	指向保存接收消息实际字节数的变量的指针。
Suspend	如果管道为空，指定是否挂起正在调用的任务。

下面是挂起类型的有效选项：

NU_NO_SUSPEND

不管请求是否满足，服务立即返回。注：如果服务被非任务线程调用，这是唯一有效的配置。

NU_SUSPEND

正在调用的任务挂起知道消息有效。

时间间隔值（1~4294967293）

正在调用的服务挂起知道消息有效或知道指定的定时器节拍数到时。

返回值

此项服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_PIPE	表示管道指针无效
NU_INVALID_POINTER	表示消息指针为空或者 actual_size 指针为空
NU_INVALID_SIZE	表示 size 参数不同于管道支持的消息尺寸。只适用于定义定长字节的管道。
NU_INVALID_SUSPEND	表示试图从非任务线程挂起
NU_PIPE_EMPTY	表示管道为空
NU_TIMEOUT	表示尽管在挂起到指定超时值之后，管道仍然为空
NU_PIPE_DELETED	在任务挂起期间管道被删除
NU_PIPE_RESET	任务挂起期间任务被复位

任务更改

Yes

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务通信服务

相关

NU_Broadcast_To_Pipe,NU_Send_To_Pipe,NU_Send_TO_Front_Of_Pipe,
NU_Pipe_Information

举例

```
NU_PIPE Pipe;
UNSIGNED message[4];
UNSIGNED actual_size;
STATUS status;

/*从"Pipe"指定的管道控制块中接收一个 4 字节，定长尺寸消息。即使管道为空也不挂起。
假设"Pipe"在先前已经调用 Nucleus PLUS 服务 NU_Create_Pipe 创建*/

status = NU_Receive_From_Pipe(&Pipe,
                              &message[0],
                              4,
                              &actual_size,
                              NU_NO_SUSPEND);

/*这里 status 表示服务请求成功与否。如果成功，"message"包含了接收的管道消息和
                    包含四个"actual_size"*/
```

NU_Receive_From_Queue

函数原型

```
STATUS NU_Receive_From_Queue( NU_QUEUE *Queue,
                              VOID *message,
                              UNSIGNED size,
                              UNSIGNED *actual_size,
                              UNSIGNED suspend);
```

描述

此项服务从指定的队列中接收一个消息。如果队列包含一个或多个消息，立即从队列里移除前面的消息且拷贝到指定的位置。根据队列支持的消息类型，队列消息由定长和不定长 UNSIGNED 数据类型组成。此项服务的参数详细定义如下：

参数	含义
Queue	队列指针
message	消息目的指针。注：消息目标必须足够大能容纳 size 的字节数。
size	指定消息内的 UNSIGNED 数据类型数。这个值必须对应于在队列创建时定义的消息尺寸。只适用于定义定长字节的队列；否则忽略。
actual_size	指向保存接收消息实际 UNSIGNED 数据类型数的变量的指针。
Suspend	如果队列为空，指定是否挂起正在调用的任务。

下面是挂起类型的有效选项：

NU_NO_SUSPEND

不管请求是否满足，服务立即返回。注：如果服务被非任务线程调用，这是唯一有效的配置。

NU_SUSPEND

正在调用的任务挂起知道消息有效。

时间间隔值（1~4294967293）

正在调用的服务挂起知道消息有效或知道指定的定时器节拍数到时。

返回值

此项服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_QUEUE	表示队列指针无效
NU_INVALID_POINTER	表示消息指针为空或者 actual_size 指针为空
NU_INVALID_SIZE	表示 size 参数不同于队列支持的消息尺寸。只适用于定义定长字节的队列。
NU_INVALID_SUSPEND	表示试图从非任务线程挂起
NU_QUEUE_EMPTY	表示队列为空
NU_TIMEOUT	表示尽管在挂起到指定超时值之后，队列仍然为空
NU_QUEUE_DELETED	在任务挂起期间队列被删除
NU_QUEUE_RESET	任务挂起期间任务被复位

任务更改

Yes

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务通信服务

相关

NU_Broadcast_To_Queue,NU_Send_To_Queue,
NU_Send_TO_Front_Of_Queue,NU_Queue_Information

举例

```
NU_QUEUE Queue;
UNSIGNED message[4];
UNSIGNED actual_size;
STATUS status;

/*从"Queue"指定的队列控制块中接收一个 4 个 UNSIGNED 数据类型消息。如果队列为空，
挂起知道请求被满足。假设"Queue"在先前已经调用 Nucleus PLUS 服务
NU_Create_Queue 创建*/

status = NU_Receive_From_Queue(&Queue,
                               &message[0],
                               4,
                               &actual_size,
                               NU_NO_SUSPEND);

/*这里 status 表示服务请求成功与否。如果成功，"message"包含了接收的队列消息*/
```

NU_Receive_Signals

函数原型

```
UNSIGNED NU_Receive_Signals(VOID);
```

描述

此项服务返回与正在调用任务相关联的每个信号的当前值。所有的信号作为服务调用结果自动清除。

任务更改

NO

允许调用

task

类别

任务同步服务

相关

NU_Control_Signals,NU_Register_Signal_Handler,
NU_Send_Signals

举例

```
UNSIGNED signals;
/*接收和清除当前任务的信号*/
signals = NU_Receive_Signals();
```

NU_Register_LISR

函数原型

```
STATUS NU_Register_LISR(   INT vector,
                           VOID (*list_entry)(INT),
                           VOID (**old_lisr)(INT));
```

描述

此项服务使被 vector 指定的中断向量和被 list_entry 指向 LISR 函数联合起来。在调用指定的 LISR 之前系统上下文自动保存并且在 LISR 返回之后恢复。因此，LISR 函数可以用 C 语言编写。然而，LISRs 只允许访问少量的 Nucleus PLUS 服务。如果和其他 Nucleus PLUS 的交流是必需的，一个高优先级的中断服务子程序（HISR）必须被 LISR 激活。

如果 lisr_entry 参数为 NU_NULL,指定向量的等级被清除。

警告:如果一个 LISR 用汇编语言编写，它必须遵循 C 编译器关于寄存器用法和返回机制的约定。请看你的关于 C-汇编交叉详细需求的编译器文档。

返回值

此项服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_VECTOR	表示指定的向量无效
NU_NOT_REGISTERED	表示向量当前没有注册且分别注册由 lisr_entry 指定 (Indicates the vector is not registered and de-registration was specified by lisr_entry.)
NU_NO_MORE_LISRS	表示已注册 LISRs 的最大数已经超出了。最大数可以在 NUCLEUS.H 中改动。注：这个返回值给出，Nucleus PLUS 库将需要重建。

任务更改

no

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

中断服务

相关

NU_Control_Interrupts,NU_Create_HISR,NU_Delete_HISR,
NU_Activate_HISR

举例

```
STATUS status;
VOID (*old_lisr)(INT);

/*
    关联变量 10 到 LIST 函数"LISR_example"。假设 LISR 函数在下面定义：
    void LISR_example(INT vector_number)
    {
        /*vector_number 包含实际的中断向量数*/
        /*Nucleus PLUS 服务调用，除了 NU_Activate_HISR 和其他几个，其他的都
不允许在这个函数中调用。*/
    }
*/

status = NU_Register_LISR(10,LISR_example,&old_lisr);

/*如果 status 为 NU_SUCCESS,LISR_example 在中断向量 10 出现时运行。
注："old_list"包含先前已注册的 LISR*/
```

NU_Register_Signal_Handler

函数原型

```
STATUS
NU_Register_Signal_Handler(VOID(*signal_handler)(UNSIGNED));
```

描述

此项服务为正在调用的任务注册一个 signal_handler 指定的信号处理函数。默认情况是，所有的信号在任务创建时都被禁止。不管 NU_Control_Signals 服务请求，信号保持禁止，直到任务的信号处理函数被注册。一个信号处理函数在任务上下文的顶部运行(A signal handler executes on top of the task context.)大多数的服务可以从一个信号处理函数被调用。然而，从信号处理函数调用的服务不能指定挂起。

返回值

服务完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功调用
NU_INVALID_TASK	表示提供的任务指针无效
NU_INVALID_POINTER	表示信号处理器指针为空

任务更改

NO

允许调用

task

类别

任务同步服务

相关

NU_Control_Signals,NU_Receive_Signals,NU_Send_Signals

举例

```
STATUS status;
/*注册函数"Signal_Handler"为任务处理函数。假设"Signal_Handler"已经在下面定义：
    void Signal_Handler(UNSIGNED signals)
    {
        /*处理 signals 出现的相关操作。注意处理和 HISRs 在不允许自挂起方面有同样的约束。*/
    }
        */

status = NU_Register_Signal_Handler(signal_Handler);

/*如果status为NU_SUCCESS,Signal_Handler每次被调用用来使能信号发送*/
```

NU_Release_Information

函数原型

```
CHAR *NU_Release_Information(VOID)
```

描述

此项服务返回Nucleus PLUS版权信息字符串指针。字符串以ASCII格式存在且NULL终止。

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

开发服务

相关

NU_License_Information

举例

```
CHAR *release_pointer;

/*指向 Nucleus PLUS 版权信息字符串*/
release_pointer = NU_Release_Information();
```

NU_Release_Semaphore

函数原型

```
STATUS NU_Release_Semaphore(NU_SEMAPHORE *semaphore);
```

描述

此项服务释放由参数 semaphore 指定的信号量的一个实例。如果有很多任务等待获得

同一个信号量，第一个等待的任务被给予信号量的这个实例。另外，如果没有任务等待这个信号量，内部计数器加一。

返回值

此项服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成。
NU_INVALID_SEMAPHORE	表示信号量指针无效

任务更改

YES

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务同步服务

相关

NU_Obtain_Semaphore,NU_Semaphore_Information

举例

```
NU_SEMAPHORE Semaphore;
STATUS status;

/*释放"Semaphore"指定的信号量控制块的一个实例。如果其他任务正在等待获得同一个信号量，这项服务的结果就是传输这个信号量的实例到第一个等待的任务。假设"Semaphore"在先前已经调用 Nucleus PLUS 服务 NU_Create_Semaphore 创建*/

status = NU_Release_Semaphore(&Semaphore);
```

NU_Relinquish

函数原型

```
VOID NU_Relinquish(VOID);
```

描述

此项服务允许所有其他同一优先级就绪任务有机会在调用任务再次运行之前执行。

任务更改

Yes

允许调用

task

类别

任务控制服务

相关

NU_Sleep,NU_Suspend_Task,NU_Resume_Task,
NU_Terminate_Task,NU_Reset_Task,NU_Task_Information

举例

```
/*允许其他同一优先级就绪任务在调用任务再次运行之前执行。
    NU_Relinquish();
```

NU_Request_Driver

函数原型

```
STATUS NU_Request_Driver(  NU_DRIVER *driver,
                           NU_DRIVER_REQUEST *request);
```

描述

此项服务发送一个指向由 driver 指定的 I/O 驱动器的请求的请求结构。标准 I/O 接口请求的定义可以在附录 C 中找到。

返回值

此项服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功初始化。请求结构中 nu_status 字段表示 I/O 请求的实际完成状态。
NU_INVALID_DRIVER	表示 I/O 驱动器指针无效
NU_INVALID_POINTER	表示 I/O 请求指针为空

任务更改

Yes

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

I/O 驱动器服务

相关

NU_Established_Drivers,NU_Driver_Pointers

举例

```
NU_DRIVER Driver;
NU_DRIVER_REQUEST request;
STATUS status;

/*对一个简单的 I/O 驱动器建立一个初始化请求*/

request.nu_function = NU_INITIALIZE;

/*发送初始化请求到"Driver"。假设"Driver"在先前已经调用 Nucleus PLUS
服务 NU_Create_Driver 创建。*/

status = NU_Request_Driver(&Driver,&request);

    /*如果 status 表示成功，驱动器接收请求。附加的 IO 驱动器指定状态在请
                求结构中可以用到*/
```

NU_Reset_Mailbox

函数原型

```
STATUS NU_Reset_Mailbox(NU_MAILBOX *mailbox);
```

描述

此项服务丢弃"mailbox"指定邮箱中当前一个信息。所有的在邮箱上任务的挂起恢复时返回适当的复位状态。

返回值

此项服务完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_MAILBOX	表示邮箱指针无效

任务更换

YES

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务通讯服务

相关

NU_Broadcast_To_Mailbox,NU_Send_To_Mailbox,
NU_Receive_From_Mailbox,NU_Mailbox_Information

举例

```
NU_MAILBOX Mailbox;
STATUS status;

/*复位邮箱控制块“Mailbox”。假设“Mailbox”已经在先前调用 Nucleus PLUS
服务 NU_Create_Mailbox 创建*/

status = NU_Reset_Mailbox(&Mailbox);
```

NU_Reset_Pipe

函数原型

```
STATUS NU_Reset_Pipe(NU_PIPE *Pipe);
```

描述

此项服务丢弃"Pipe"指定管道中当前所有消息。所有的在管道上任务的挂起恢复时返回适当的复位状态。

返回值

此项服务完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_PIPE	表示管道指针无效

任务更换

YES

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务通讯服务

相关

NU_Broadcast_To_Pipe,NU_Send_To_Pipe,
NU_Send_To_Front_Of_Pipe,NU_Receive_From_Pipe,
NU_Pipe_Information

举例

```
NU_PIPE Pipe;
STATUS status;

/*复位管道控制块“Pipe”。假设“Pipe”已经在先前调用 Nucleus PLUS 服务
NU_Create_Pipe 创建*/

                status = NU_Reset_Pipe(&Pipe);
```

NU_Reset_Queue

函数原型

```
STATUS NU_Reset_Queue(NU_QUEUE *queue);
```

描述

此项服务丢弃”queue”指定队列中当前所有消息。所有的在队列上任务的挂起恢复时返回适当的复位状态。

返回值

此项服务完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_QUEUE	表示队列指针无效

任务更换

YES

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务通讯服务

相关

NU_Broadcast_To_Queue,NU_Send_To_Queue,
NU_Receive_From_Queue,NU_Queue_Information

举例

```
NU_QUEUE Queue;
STATUS status;

/*复位队列控制块“Queue”。假设“Queue”已经在先前调用 Nucleus PLUS 服务
NU_Create_Queue 创建*/

                status = NU_Reset_Queue(&Queue);
```

NU_Reset_Semaphore

函数原型

```
STATUS NU_Reset_Semaphore( NU_SEMAPHORE *semaphore,
                           UNSIGNED initial_count);
```

描述

此项服务复位值为 initial_count，semaphore 指定的信号量。所有在此信号量挂起的任务恢复时返回适当的复位状态。

返回值

此项服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_SEMAPHORE	表示信号量指针无效

任务更改

Yes

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务同步服务

相关

NU_Obtain_Semaphore,NU_Release_Semaphore,
NU_Semaphore_Information

举例

```
NU_SEMAPHORE Semaphore;
STATUS status;

/*复位信号量控制块"Semaphore"。初始化计数值设置为 1。假设"Semaphore"在先前已经通过调用 Nucleus PLUS 服务 NU_Create_Semaphore 创建*/

status = NU_Reset_Semaphore(&Semaphore,1);
```

NU_Reset_Task

函数原型

```
STATUS NU_Reset_Task(NU_TASK *task,UNSIGNED argc,VOID *argv);
```

描述

此项服务复位先前已终止或结束的任务。注：在复位后，此项服务不恢复任务。NU_Resuem_Task 必须调用来真正的再次启动任务。此项服务的参数详细定义如下：

参数	含义
task	任务控制块指针
argc	一个 UNSIGNED 数据类型，可以用来传输信息到任务。
argv	一个指针，可以用来传输信息到任务。

返回值

此项服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功调用

NU_INVALID_TASK	表示任务指针无效
NU_NOT_TERMINATED	表示指定的任务不是处于终止或完成状态。只有处于终止或完成状态的任务可以被复位。

任务更改

No

允许调用

HISR,Signal Handler,task

类别

任务控制服务

相关

NU_Create_Task,NU_Delete_Task,NU_Terminate_Task,
NU_Resume_Task,NU_Suspend_Task,NU_Task_Information

举例

```
NU_TASK Task;
STATUS status;

/*复位先前已经终止的任务控制块"Task"。argc 和 argv 传递任务值为 0 和 NULL。
假设"Task"在先前已经调用 Nucleus PLUS 服务 NU_Create_Task 创建*/

status = NU_Reset_Task(&Task,0,NULL);
```

NU_Reset_Timer

函数原型

```
STATUS NU_Reset_Timer(  NU_TIMER *timer,
                        VOID ("expiration_routine")(UNSIGNED),
                        UNSIGNED initial_time,
                        UNSIGNED reschedule_time,
                        OPTION enable);
```

描述

此项服务用新的操作参数复位指定的定时器。注：定时器在此项服务调用之前必须禁止。此项服务的参数详细定义如下：

参数	含义
timer	定时器指针
expiration_routine	在定时器定时到时，指定应用子程序的运行
initial_time	指定定时器到时的定时器节拍的初始值
reschedule_time	为第一次到期后的到期指定定时器节拍数。如果此参数为零，定时器只到时一次
enable	此参数有效配置为：NU_ENABLE_TIMER 和 NU_DISABLE_TIMER。NU_ENABLE_TIMER 在定时器复位后立即激活它。NU_DISABLE_TIMER 使定时器禁止。定时器带 NU_DISABLE_TIMER 参数复位必须通过在下一次调用 NU_Control_Timer 来使能。

返回值

此项服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_TIMER	表示指定的定时器控制块指针无效
NU_INVALID_FUNCTION	表示定时到期函数指针为空
NU_INVALID_ENABLE	表示使能参数无效
NU_NOT_DISABLED	表示定时器当前为使能状态。它在被复位之前必须被禁止。

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

定时器服务

相关

NU_Create_Timer,NU_Delete_Timer,NU_Control_Timer,
NU_Timer_Information

举例

```
NU_TIMER Timer;
STATUS status;

/*复位定时器控制块"Timer"，初始化到时为 3 个定时器节拍，以后到时为 30 个定期节拍。新的到时子程序为"new_expire"。在定时器复位后自动使能。假设"Timer"在先前已经通过调用 Nucleus PLUS 服务 NU_Create_Timer 创建*/

status =
       NU_Reset_Timer(&Timer,new_expire,3,30,NU_ENABLE_TIMER);

                /*status 的内容表示服务成功与否*/
```

NU_Resume_Driver

函数原型

```
STATUS NU_Resume_Driver(NU_TASK *task);
```

描述

此项服务恢复一个先前通过 NU_Suspend_Driver 服务挂起的任务。典型的，此项服务和他的挂起副本都是应用于 I/O 驱动器的服务。参数 Task 指向恢复的任务。

返回值

此项服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_TASK	表示任务指针无效
NU_INVALID_RESUME	表示指定的任务没有通过调用 NU_Suspend_Driver 服务所以不处于挂起状态。

任务更改

Yes

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

I/O 驱动器服务

相关

NU_Suspend_Driver

举例

```
NU_TASK Task;
STATUS status;

/*恢复先前调用 NU_Suspend_Driver 服务挂起的任务控制块"Task"。假设"Task"
在先前已经调用 Nucleus PLUS 服务 NU_Create_Task 创建*/

status = NU_Resume_Driver(&Task);
```

NU_Resume_Task

函数原型

```
STATUS NU_Resume_Task(NU_TASK *task);
```

描述

此项服务恢复一个先前通过 NU_Suspend_Task 服务挂起的任务。另外，此项服务初始化一个在先前已经复位或创建但是没有自动启动的任务。

返回值

此项服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_TASK	表示任务指针无效
NU_INVALID_RESUME	表示指定的任务不处于无条件挂起状态。

任务更改

Yes

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务控制服务

相关

NU_Suspend_Task

举例

```
NU_TASK Task;
STATUS status;

/*恢复先前调用 NU_Suspend_Task 服务挂起的任务控制块"Task"。假设"Task"在
先前已经调用 Nucleus PLUS 服务 NU_Create_Task 创建*/
```

```
status = NU_Resume_Task(&Task);
```

NU_Retrieve_Clock

函数原型

```
UNSIGNED NU_Retrieve_Clock(VOID);
```

描述

此项服务返回连续递增定时器节拍计数器的当前值。每次定时器中断，计数器递增一次。

任务更改

NO

允许调用

Application_Initialize,LISR,HISR,Signal Handler,task

类别

定时器服务

相关

NU_Set_Clock

举例

```
UNSIGNED clock_value;

/*读取系统节拍时钟的当前值*/

clock_value = NU_Retrieve_Clock();
```

NU_Retrieve_Events

函数原型

```
STATUS NU_Retrieve_Events( NU_EVENT_GROUP *group,
                          UNSIGNED requested_events,
                          OPTION operation,
                          UNSIGNED *retrieved_events,
                          UNSIGNED suspend);
```

描述

此项服务从指定的事件标志集中找回指定的事件标志联合(combination)。如果联合(combination)出现，服务立即完成。此项服务参数详细定义如下：

参数	含义
group	事件标志集指针
requested_events	被请求的事件标志。置位表示相应的事件标志被请求。
operation	有四个操作配置有效：NU_AND,NU_AND_CONSUME,NU_OR,NU_OR_CONSUME。NU_AND 和 NU_AND_CONSUME 配置表示所有的已经被请求的事件标志都是需要的。NU_OR 和 NU_OR_CONSUME 配置表示已经被请求的事件标志中的一个或几个就够了。CONSUME 配置自动清除请求成功的事件标志。
retrieved_events	包含真正收回的事件标志。
suspend	如果被请求的事件标志联合无效，指定调用任务是否挂起。

下列挂起配置是有效的

NU_NO_SUSPEND

不管请求是否被满足，服务立即返回。注：如果服务从非任务线程调用，这是唯一有效的配置。

NU_SUSPEND

调用任务挂起直到内存分区有效。

时间间隔值（1 ～ 4294967293）

调用任务挂起直到一个内存分区有效，或者直到指定数量的时钟节拍到时。

返回值

此项服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_GROUP	表示事件标志集指针无效
NU_INVALID_POINTER	表示收回的事件标志指针为 NULL
NU_INVALID_OPERATION	表示 operation 参数无效
NU_INVALID_SUSPEND	表示试图从非任务线程挂起
NU_NOT_PRESENT	表示被请求的事件标志联合当前没有出现
NU_TIMEOUT	表示被请求的事件标志联合即使在挂起超时过后都未出现
NU_GROUP_DELETED	表示在任务挂起时，事件标志集删除

任务更改

Yes

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务通讯服务

相关

NU_Set_Events,NU_Event_Group_Information

举例

```
NU_EVENT_GROUP Group;
UNSIGNED actual_flags;
STATUS status;
```

/*从事件标志集控制块”Group”收回事件标志 7，2 和 1。注：所有的事件标志必须出现满足请求。如果他们没有出现，调用任务无条件挂起。当然，如果这个请求满足的时候，事件标志 7，2，1 被消灭。假设”Group”在先前已经调用 Nucleus PLUS 服务 NU_Create_Event_Group 创建*/

```
status = NU_Retrieve_Events(  &Group,
                              0x86,
                              NU_AND_CONSUME,
                              &actual_flags,
                              NU_SUSPEND);
```

NU_Retrieve_History_Entry

函数原型

```
STATUS NU_Retrieve_History_Entry( DATA ELEMENT *id,
                                  UNSIGNED *param1,
                                  UNSIGNED *param2,
                                  UNSIGNED *param3,
                                  UNSIGNED *time,
                                  NU_TASK **task,
                                  NU_HISR **hisr);
```

描述

此项服务返回一个系统历史记录最原始(Oldest)的入口。注：通常这是先于使用此项服务禁止历史存储的好注意。为了记录历史入口历史存储必须使能。默认情况，系统历史记录在启动时禁止。此项服务参数详细定义如下：

参数	含义
id	保存入口 ID 变量的指针。注：Nucleus PLUS 服务的 Ids 为大写服务名称，且末尾添加_ID。用户制造的入口有 NU_USER_ID 的 ID。
param1,2,3	指向保存第 1、2、3 历史参数入口的变量。
time	指向保存与此入口对应的系统时钟值的变量
task	指向保存制造入口的任务的指针的任务指针
hisr	指向保存制造入口的 HISR 指针的 HISR 指针

返回值

任务完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_END_OF_LOG	表示记录中没有更多的入口

任务更改

No

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

开发服务

相关

NU_Enable_History_Saving,NU_Disable_History_Saving

举例

```
DATA ELEMENT id;
UNSIGNED param1;
UNSIGNED param2;
UNSIGNED param3;
UNSIGNED time;
UNSIGNED *task;
NU_HISR *hisr;

/*假设系统历史纪录已经禁止。获得下一个最近的入口*/
```

```
status = NU_Retrieve_History_Entry(  &id,
                                     &param1,
                                     &param2,
                                     &param3,
                                     &time,
                                     &task,
                                     &hisr);
```

/*如果 status 为 NU_SUCCESS,提供的变量获得有效的信息。注：无论是 task 还是 hisr 必须为空。*/

NU_Semaphore_Information

函数原型

```
STATUS NU_Semaphore_Information(  NU_SEMAPHORE *semaphore,
                                  CHAR *name,
                                  UNSIGNED *current_count,
                                  OPTION *suspend_type,
                                  UNSIGNED *tasks_waiting
                                  NU_TASK **first_task);
```

描述

此项服务返回有关指定任务同步信号量的各种信息。参数详细定义如下：

参数	含义
Semaphore	同步信号量指针
name	信号量名 8 字符目标区域指针
current_count	指向保存当前信号量实例的变量
Suspend_type	保存任务挂起类型的变量指针。有效任务挂起类型为：NU_FIFO 和 NU_PRIORITY
Tasks_waiting	指向保存等待信号量的任务数量的变量指针
First_task	指向任务指针的指针。第一个挂起的任务指针放入这个任务指针。

返回值

服务完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_SEMAPHORE	表示信号量指针无效

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务同步服务

相关

NU_Create_Semaphore,NU_Delete_Semaphore,
NU_Established_Semaphore,NU_Semaphore_Pointers

举例

```
NU_SEMAPHORE Semaphore;
CHAR Semaphore_name[8];
UNSIGNED current_count;
OPTION suspend_type;
UNSIGNED task_suspended;
NU_TASK *first_task;
STATUS status;

/*获得“Semaphore”指定信号量控制块的信息。假设“Semaphore”在先前已经调用
Nucleus PLUS 服务 NU_Create_Semaphore 创建*/

status = NU_Semaphore_Information( &Semaphore,
                                   Semaphore_name,
                                   &current_count,
                                   &suspend_type,
                                   &tasks_suspended,
                                   &first_task);
   /*如果 status 等于 NU_SUCCESS,其他信息是精确的。*/
```

NU_Semaphore_Pointers

函数原型

```
UNSIGNED NU_Semaphore_Pointers(  NU_SEMAPHORE **pointer_list,
                                 UNSIGNED maximum_pointers);
```

描述

此服务建立一个系统中所有已建立的信号量的指针连续列表。注：已经删除的信号量不再被认为已经建立。参数 pointer_list 指向用于建立指针列表的位置，maximum_pointers 表示列表最大尺寸。这个服务返回列表指针实际数量。另外，列表按最旧到最新成员排序。

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务同步服务

相关

NU_Create_Semaphore,NU_Delete_Semaphore
NU_Established_Semaphorees,NU_Semaphore_Information

举例

```
/*定义一个能容纳 20 个信号量指针的阵列*/
NU_SEMAPHORE *Pointer_Array[20];
UNSIGNED number;
/*获得当前激活的信号量指针列表（最大 20 个）*/
number = NU_Semaphore_Pointers(&Pointer_Array[0],20);
```

/*这里，number 包含了列表中指针的实际数量*/

NU_Send_Signals

函数原型

```
STATUS NU_Send_Signals(NU_TASK *task,UNSIGNED signals);
```

描述

此项服务发送参数 signals 指示信号到参数 task 指示的任务。如果接收任务有任何指定使能的信号，一旦接收任务优先级允许，它的注册信号处理函数就立即运行。。每个任务有 32 个有效的信号，都由 signals 的每一位来描述。

有几个条件阻止接收任务信号处理函数被运行。如下所示：

- ✧ 接收任务处于完成或终止态；
- ✧ 接收任务无条件挂起(无论在创建之后没有启动或是被 NU_Suspend_Task 挂起)。如果是这种情况，信号处理函数不运行，直到任务恢复；
- ✧ 总是有优先级高于接收任务的任务就绪；
- ✧ 接收任务没有使能信号发送；
- ✧ 接收任务没有注册信号处理函数；

返回值

此项服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_TASK	表示任务指针无效

任务更改

Yes

允许调用

Application_Initialize,HISR,Signal Handler,task

相关

NU_Receive_Signals,NU_Control_Signals,
NU_Register_Signal_Handler

举例

```
NU_TASK Task;
STATUS status;
```

/*发送信号 1，7 和 31 到任务控制块”Task”。注意与位位置对应的信号。假设”Task”在先前已经调用 Nucleus PLUS 服务 NU_Create_Task 创建*/

```
status = NU_Send_Signals(&Task,0x80000082);
```

NU_Send_To_Front_Of_Pipe

函数原型

```
STATUS NU_Send_To_Front_Of_Pipe( NU_PIPE *pipe,
                                 VOID *message,
                                 UNSIGNED size,
                                 UNSIGNED suspend);
```

描述

此项服务防止消息至指定管道的前端。如果管道中有足够的空间放置消息，此项服务立即执行。依据管道支持的消息类型，管道消息由定长和不定长字节组成。此项服务的参数定义如下：

参数	含义
pipe	管道指针
message	发送消息指针
size	指定消息的字节数。如果管道支持变长消息，这个参数必须等于或小于管道支持的消息尺寸。如果管道支持定长消息，此参数必须正好等于管道支持的消息尺寸。
suspend	指定管道满时是否官气调用任务。

下列挂起配置是有效的

NU_NO_SUSPEND

不管请求是否被满足，服务立即返回。注：如果服务从非任务线程调用，这是唯一有效的配置。

NU_SUSPEND

调用任务挂起直到内存分区有效。

时间间隔值（1 ～ 4294967293）

调用任务挂起直到一个内存分区有效，或者直到指定数量的时钟节拍到时。

返回值

此项服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_PIPE	表示管道指针无效
NU_INVALID_POINTER	表示消息指针为 NULL
NU_INVALID_SIZE	表示消息尺寸和管道支持的消息尺寸有矛盾
NU_INVALID_SUSPEND	表示试图从非任务线程挂起
NU_PIPE_FULL	表示管道满
NU_TIMEOUT	表示即使在挂起超时之后，管道状态仍然为满
NU_PIPE_DELETED	任务在挂起期间管道被删除
NU_PIPE_RESET	任务在挂起期间管道被复位

任务更改

Yes

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务通讯服务

相关

NU_BroadCast_To_Pipe,NU_Receive_From_Pipe,
NU_Send_To_Pipe,NU_Pipe_Information

举例

```
NU_PIPE Pipe;
UNSIGNED_CHAR message[4];
```

```
STATUS status;

/*建立一个 4 字节发送消息。"message"内容没有意义*/

message[0] = 0x01;
message[1] = 0x02;
message[2] = 0x03;
message[3] = 0x04;

/*发送一个 4 字节，定长消息到管道控制块"Pipe"。即使管道为满也不挂起。假
设"Pipe"在先前已经调用 Nucleus PLUS 服务 NU_Create_Pipe 创建*/

status = NU_Send_To_Front_Of_Pipe(   &Pipe,
                                     &message[0],
                                     4,
                                     NU_NO_SUSPEND);

/*这里 status 表示服务请求成功与否。如果成功，"message"发送到"Pipe"*/
```

NU_Send_To_Front_Of_Queue

函数原型

```
STATUS NU_Send_To_Front_Of_Queue( NU_QUEUE *Queue,
                                  VOID *message,
                                  UNSIGNED size,
                                  UNSIGNED suspend);
```

描述

此项服务防止消息至指定队列的前端。如果队列中有足够的空间放置消息，此项服务立即执行。依据队列支持的消息类型，队列消息由定长和不定长 UNSIGNED 数据类型组成。此项服务的参数定义如下:

参数	含义
Queue	队列指针
message	发送消息指针
size	指定消息的 UNSIGNED 数据类型数。如果队列支持变长消息，这个参数必须等于或小于队列支持的消息尺寸。如果队列支持定长消息，此参数必须正好等于队列支持的消息尺寸。
suspend	指定队列满时是否官气调用任务。

下列挂起配置是有效的

NU_NO_SUSPEND

不管请求是否被满足，服务立即返回。注：如果服务从非任务线程调用，这是唯一有效的配置。

NU_SUSPEND

调用任务挂起直到内存分区有效。

时间间隔值（1 ～ 4294967293）

调用任务挂起直到一个内存分区有效，或者直到指定数量的时钟节拍到时。

返回值

此项服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_QUEUE	表示队列指针无效
NU_INVALID_POINTER	表示消息指针为 NULL
NU_INVALID_SIZE	表示消息尺寸和队列支持的消息尺寸有矛盾
NU_INVALID_SUSPEND	表示试图从非任务线程挂起
NU_QUEUE_FULL	表示队列满
NU_TIMEOUT	表示即使在挂起超时之后，队列状态仍然为满
NU_QUEUE_DELETED	任务在挂起期间队列被删除
NU_QUEUE_RESET	任务在挂起期间队列被复位

任务更改

Yes

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务通讯服务

相关

NU_BroadCast_To_Queue,NU_Receive_From_Queue,
NU_Send_To_Queue,NU_Queue_Information

举例

```
NU_QUEUE Queue;
UNSIGNED_CHAR message[4];
STATUS status;

/*建立一个 4 字节发送消息。”message”内容没有意义*/

message[0] = 0x00001111;
message[1] = 0x00002222;
message[2] = 0x00003333;
message[3] = 0x00004444;

/*发送一个 4 字节，定长消息到队列控制块”Queue”。挂起调用任务知道消息可以发
送。假设”Queue”在先前已经调用 Nucleus PLUS 服务 NU_Create_Queue 创建*/

status = NU_Send_To_Front_Of_Queue(  &Queue,
                                     &message[0],
                                     4,
                                     NU_SUSPEND);
```

/*这里 status 表示服务请求成功与否。如果成功，"message"发送到"Queue"*/

NU_Send_To_Mailbox

函数原型

```
STATUS NU_Send_To_Mailbox( NU_MAILBOX *mailbox,
                           VOID *message,
                           UNSIGNED suspend);
```

描述

此项服务放置消息到指定的邮箱。如果邮箱为空，消息立即拷贝到邮箱中。邮箱消息大小等于 4 个 UNSIGNED 数据类型。此项服务的参数详细定义如下：

参数	含义
mailbox	邮箱指针
message	发送消息指针
suspend	如果邮箱已经包含了一个消息，指定是否挂起调用任务

下列挂起配置是有效的

NU_NO_SUSPEND

不管请求是否被满足，服务立即返回。注：如果服务从非任务线程调用，这是唯一有效的配置。

NU_SUSPEND

调用任务挂起直到内存分区有效。

时间间隔值（1 ～ 4294967293）

调用任务挂起直到一个内存分区有效，或者直到指定数量的时钟节拍到时。

返回值

此项服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_MAILBOX	表示邮箱指针无效
NU_INVALID_POINTER	表示消息指针为 NULL
NU_INVALID_SUSPEND	表示试图从非任务线程挂起
NU_MAILBOX_FULL	表示邮箱满
NU_TIMEOUT	表示即使在挂起超时之后，邮箱状态仍然为满
NU_MAILBOX_DELETED	任务在挂起期间邮箱被删除
NU_MAILBOX_RESET	任务在挂起期间邮箱被复位

任务更改

Yes

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务通信服务

相关

NU_Broadcast_To_Mailbox,NU_Receive_From_Mailbox,
NU_Mailbox_Information

举例

```
    NU_MAILBOX Mailbox;
    UNSIGNED message[4];
    STATUS status;

    /*建立一个 4 个 UNSIGNED 变量的消息用于发送。"message"内容无意义*/
    message[0] = 0x00001111;
    message[1] = 0x00002222;
    message[2] = 0x00003333;
    message[3] = 0x00004444;

    /*发送消息至邮箱控制块"Mailbox"。挂起调用任务直到消息可以发送或者 25 个定
时器节拍到期。假设"Mailbox"在先前已经通过调用 Nucleus PLUS 服务
NU_Create_Mailbox 创建。*/

    status = NU_Send_To_Mailbox(Mailbox,&message[0],25);

    /*这里 status 表示服务请求是否成功。如果成功，"message"发送到"Mailbox"*/
```

NU_Send_To_Pipe

函数原型

```
STATUS NU_Send_To_Pipe( NU_PIPE *Pipe,
                        VOID *message,
                        UNSIGNED size,
                        UNSIGNED suspend);
```

描述

此项服务放置消息到指定的管道底端。如果管道有足够的空间保存消息，此项服务立即处理。根据管道支持的消息类型，管道消息包括定长或变长字节数。此项服务的参数详细定义如下：

参数	含义
Pipe	管道指针
message	发送消息指针
size	指定消息的字节数。如果管道支持变长消息，此项参数必须等于或小于管道支持的消息尺寸。如果管道支持定长消息，此参数必须正好等于管道支持的消息尺寸。
Suspend	如果管道已经包含了一个消息，指定是否挂起调用任务

下列挂起配置是有效的

NU_NO_SUSPEND

不管请求是否被满足，服务立即返回。注：如果服务从非任务线程调用，这是唯一有效的配置。

NU_SUSPEND

调用任务挂起直到内存分区有效。

时间间隔值（1 ～ 4294967293）

调用任务挂起直到一个内存分区有效，或者直到指定数量的时钟节拍到时。

返回值

此项服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_PIPE	表示管道指针无效
NU_INVALID_POINTER	表示消息指针为 NULL
NU_INVALID_SIZE	表示消息尺寸与管道支持的消息尺寸不匹配
NU_INVALID_SUSPEND	表示试图从非任务线程挂起
NU_PIPE_FULL	表示管道满
NU_TIMEOUT	表示即使在挂起超时之后，管道状态仍然为满
NU_PIPE_DELETED	任务在挂起期间管道被删除
NU_PIPE_RESET	任务在挂起期间管道被复位

任务更改

Yes

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务通信服务

相关

NU_Broadcast_To_Pipe,NU_Receive_From_Pipe,
NU_Pipe_Information,NU_Pipe_Information

举例

```
NU_PIPE Pipe;
UNSIGNED message[4];
STATUS status;

/*建立一个 4 个 UNSIGNED 变量的消息用于发送。”message”内容无意义*/
message[0] = 0x01;
message[1] = 0x02;
message[2] = 0x03;
message[3] = 0x04;
```

/*发送 4 字节消息至管道控制块”Pipe”。即使管道满也不挂起。假设”Pipe”在先前已经通过调用 Nucleus PLUS 服务 NU_Create_Pipe 创建。*/

```
status = NU_Send_To_Pipe(Pipe,&message[0],4,NO_NO_SUSPEND);

 /*这里 status 表示服务请求是否成功。如果成功，”message”发送到”Pipe”*/
```

NU_Send_To_Queue

函数原型

```
STATUS NU_Send_To_Queue(   NU_QUEUE *Queue,
                           VOID *message,
                           UNSIGNED size,
```

UNSIGNED suspend);

描述

此项服务放置消息到指定的队列底端。如果队列有足够的空间保存消息，此项服务立即处理。根据队列支持的消息类型，队列消息包括定长或变长 UNSIGNED 数据类型数。此项服务的参数详细定义如下：

参数	含义
Queue	队列指针
message	发送消息指针
size	指定消息的 UNSIGNED 数据类型数。如果队列支持变长消息，此项参数必须等于或小于队列支持的消息尺寸。如果队列支持定长消息，此参数必须正好等于队列支持的消息尺寸。
Suspend	如果队列已经包含了一个消息，指定是否挂起调用任务

下列挂起配置是有效的

NU_NO_SUSPEND

不管请求是否被满足，服务立即返回。注：如果服务从非任务线程调用，这是唯一有效的配置。

NU_SUSPEND

调用任务挂起直到内存分区有效。

时间间隔值（1 ～ 4294967293）

调用任务挂起直到一个内存分区有效，或者直到指定数量的时钟节拍到时。

返回值

此项服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_QUEUE	表示队列指针无效
NU_INVALID_POINTER	表示消息指针为 NULL
NU_INVALID_SIZE	表示消息尺寸与队列支持的消息尺寸不匹配
NU_INVALID_SUSPEND	表示试图从非任务线程挂起
NU_QUEUE_FULL	表示队列满
NU_TIMEOUT	表示即使在挂起超时之后，队列状态仍然为满
NU_QUEUE_DELETED	任务在挂起期间队列被删除
NU_QUEUE_RESET	任务在挂起期间队列被复位

任务更改

Yes

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务通信服务

相关

```
NU_Broadcast_To_Queue,NU_Receive_From_Queue,
NU_Queue_Information,NU_Queue_Information
```

举例

```
NU_QUEUE Queue;
UNSIGNED message[4];
```

```
    STATUS status;

    /*建立一个 4 个 UNSIGNED 变量的消息用于发送。"message"内容无意义*/
    message[0] = 0x00001111;
    message[1] = 0x00002222;
    message[2] = 0x00003333;
    message[3] = 0x00004444;

    /*发送 4 字节消息至队列控制块"Queue"。挂起调用任务直到消息可以发送。假
设"Queue"在先前已经通过调用 Nucleus PLUS 服务 NU_Create_Queue 创建。*/

    status = NU_Send_To_Queue(Queue,&message[0],4,NO_NO_SUSPEND);

    /*这里 status 表示服务请求是否成功。如果成功，"message"发送到"Queue"*/
```

NU_Set_Clock

函数原型

```
VOID NU_Set_Clock(UNSIGNED new_value);
```

描述

此项服务设置连续技术系统时钟为 new_value 指定值。

任务更改

No

允许调用

```
Application_Initialize,HISR,Signal Handler,task
```

类别

定时器服务

相关

```
    NU_Retrieve_Clock
```

举例

```
    /*设置系统时钟为 0*/
    NU_Set_Clock(0);
```

NU_Set_Events

函数原型

```
STATUS NU_Set_Events(   NU_EVENT_GROUP *group
                        UNSIGNED event_flags
                        OPTION operation);
```

描述

此项服务在指定的事件集中设置指定的事件标志。事件集的事件标志请求被这项服务满足时，任何等待这个事件集的任务恢复。此项服务的参数详细定义如下：

参数	含义
Group	事件标志集指针

event_flags	事件标志值
operation	有两个操作选项有效：NU_OR 和 NU_AND。NU_OR 导致指定的事件标志与当前事件集中的事件标志进行或操作。NU_AND 导致指定的事件标志与当前事件集中的事件标志进行与操作。

返回值

此项服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_GROUP	表示事件标志集指针无效
NU_INVALID_OPERATION	表示操作参数无效

任务更改

Yes

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务同步服务

相关

NU_Retrieve_Events,NU_Event_Group_Information

举例

```
NU_EVENT_GROUP Group;
STATUS status;

/*在事件集控制块"Group"中设置事件标志 7，2，1。假设"Group"在先前已经调用 Nucleus PLUS 服务 NU_Create_Event_Group 创建*/

status = NU_Set_Events(&Group,0x00000086,NU_OR);

/*如果 status 为 NU_SUCCESS，事件标志设置成功。*/
```

NU_Setup_Vector

函数原型

```
VOID *NU_Setup_Vector(INT vector,VOID *new);
```

描述

此项服务使用调用者（参数 new）自定义中断服务子程序代替 vector 指定的中断向量。服务返回先前中断向量内容。

警告：提供这个子程序的 ISRs 用汇编语言编写，负责存储和恢复任何使用的寄存器。Nucleus PLUS 一些端口有一些附加约束强加在这些 ISRs 上。请看附加指定目标信息的指定处理器端口笔记。

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

中断服务

相关

NU_Control_Interrupts,NU_Register_LISR

举例

```
VOID *old_vector;
/*放置命名为"asm_ISR"的汇编语言 ISR 到向量 5*/
old_vector = NU_Setup_Vector(5,asm_ISR);
```

NU_Sleep

函数类型

```
VOID NU_Sleep(UNSIGNED ticks);
```

描述

此项服务挂起正在调用任务至指定的定时器节拍数。

任务更改

Yes

允许调用

task

类别

任务控制服务

举例

```
/*休眠 20 个定时器节拍*/
NU_Sleep(20);
```

NU_Suspend_Driver

函数原型

```
STATUS NU_Suspend_Driver(  VOID (*terminate_routine)(VOID*),
                           VOID *information,
                           UNSIGNED timeout);
```

描述

此项服务从 I/O 驱动器中挂起正在调用任务。如果被指定，终止子程序允许驱动器清除任何在终止或超时处理期间与正在调用任务有关的内部结构。注：任何通过调用 NU_Portect 已建立的保护被此项服务清除。此服务的参数详细定义如下：

参数	含义
Terminate_routine	指向一个指定驱动器终止/超时子程序的指针（可选）。
information	指向终止/超时子程序（可选）必需的追加信息的指针。
timeout	挂起超时。NU_SUSPEND 表示无条件挂起。

返回值

此项服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成

NU_INVALID_SUSPEND 表示试图从一个运行的非任务线程调用子程序。

任务更改

Yes

允许调用

task

类别

I/O 驱动器服务

相关

NU_Resume_Driver,NU_Protect,NU_Unprotect

举例

```
/*此项服务有代表性的是，使用了 I/O 驱动器的内部函数（inside）在等待 I/O 时挂起当前任务。注：任何使用 NU_Protect 服务建立的保护可以使用此项服务清除。*/
NU_Suspend_Driver(NU_NULL,NU_NULL,0);
```

NU_Suspend_Task

函数原型

```
STATUS NU_Suspend_Task(NU_TASK *task);
```

描述

此项服务无条件挂起 task 指针指定的任务。如果任务已经处于挂起状态。即使在最初导致挂起的条件消失，此服务确保任务保留在挂起状态。NU_Resume_Task 必须用于恢复这种方式的任务挂起。

返回值

此项服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_TASK	表示任务指针无效

任务更改

Yes

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务控制服务

相关

NU_Resume_Task,NU_Terminate_Task,NU_Reset_Task

举例

```
NU_TASK Task;
STATUS status;

/*无条件挂起任务控制块"Task"。假设"Task"在先前已经调用 Nucleus PLUS 服务 NU_Create_Task 创建*/

status = NU_Suspend_Task(&Task);
```

NU_Task_Information

函数原型

```
STATUS NU_Task_Information(    NU_TASK *task,
                               CHAR *name,
                               DATA ELEMENT *task_status,
                               UNSIGNED *scheduled_count,
                               OPTION *priority,
                               OPTION *preempt,
                               UNSIGNED *time_slice,
                               VOID **stack_base,
                               UNSIGNED *stack_size,
                               UNSIGNED *minimum_stack);
```

描述

此项服务返回有关指定任务的各种信息。参数详细定义如下：

参数	含义
task	任务指针
name	任务名称 8 字符目标区指针
task_status	指向保存任务当前状态的变量的指针
scheduled_count	指向保存任务被调度的次数的变量的指针
priority	指向保存任务优先级变量的指针
time_slice	指向保存任务时间片值的变量的指针。值为 0 表示这个任务的时间片被禁止
stack_base	指向保存任务堆栈起始地址的内存指针的指针
size	指向保存任务堆栈总字节数的变量的指针
minimum_stack	指向保存任务堆栈剩余字节最小值的变量的指针

任务状态

下面列表总结了 task_status 参数的可能值。

参数值	任务状态
NU_READY	运行就绪
NU_PURE_SUSPEND	无条件挂起
NU_FINISHED	从入口函数返回
NU_TERMINATED	终止
NU_SLEEP_SUSPEND	休眠
NU_MAILBOX_SUSPEND	邮箱挂起
NU_QUEUE_SUSPEND	队列挂起
NU_PIPE_SUSPEND	管道挂起
NU_EVENT_SUSPEND	事件标志集挂起
NU_SEMAPHORE_SUSPEND	信号量挂起
NU_MEMORY_SUSPEND	动态内存池挂起
NU_PARTITION_SUSPEND	分区内存池挂起
NU_DRIVER_SUSPEND	从一个 110 驱动器请求挂起

返回值

此项服务的完成状态定义如下：

状态	含义

NU_SUCCESS 表示服务成功完成
NU_INVALID_TASK 表示任务指针无效

任务更改

No

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务控制服务

相关

NU_Create_Task,NU_Delete_Task,NU_Established_Tasks,
NU_Task_Pointers,NU_Reset_Task

举例

```
NU_TASK Task;
CHAR task_name[8];
DATA ELEMENT task_status;
UNSIGNED scheduled_count;
OPTION priority;
OPTION preempt;
UNSIGNED time_slice;
VOID *stack_base;
UNSIGNED stack_size;
UNSIGNED minimum_stack;
STATUS status;

/*获得有关任务控制块"Task"的信息。假设"Task"在先前已经调用 Nucleus PLUS
服务 NU_Create_Task 创建*/

status = NU_Task_Information( &task,
                              task_name,
                              &task_status,
                              &scheduled_count
                              &priority,
                              &preempt
                              &time_slice,
                              &stack_base,
                              &stack_size,
                              &minimum_stack);
/*如果 status 为 NU_SUCCESS,其他信息就是精确的*/
```

NU_Task_Pointers

函数原型

```
UNSIGNED NU_Task_Pointers( NU_TASK **pointer_list,
                           UNSIGNED maximum_pointers);
```

描述

此项服务为所有系统中已建立的任务建立一个连续指针列表。注：已经删除的任务不再被认为时已建立。参数指针 lisr 指向建立指针列表的位置，maximum_pointers 表示列表的最大尺寸。此项服务返回列表指针的实际数量。另外，列表按最旧到最新成员排序。

任务更改

NO

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

任务控制服务

相关

NU_Create_Task,NU_Delete_Task,NU_Established_Tasks,
NU_Task_Information,NU_Reset_Task

举例

```
/*定义能容纳 20 个任务指针的阵列*/
NU_TASK *Pointer_Array[20];
UNSIGNED number;

/*获得当前激活的任务指针列表（最大为 20）*/

number = NU_Task_Pointers(&Pointers_Array[0],20);

/*这里，number 包含列表指针的实际数量*/
```

NU_Terminate_Task

函数原型

```
STATUS NU_Terminate_Task(NU_TASK *task);
```

描述

此项服务终止 task 参数指定的任务。注 1：一个终止的任务不能再次运行，知道它被复位。注 2：当从一个信号处理函数调用这个函数时，信号处理函数正在运行的任务不能被终止。

返回值

服务的完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_TASK	表示任务指针无效

任务更改

yes

允许调用

HISR,Signal Handler,task

类别

任务控制服务

相关

NU_Suspend_Task,NU_Resume_Task,NU_Reset_Task,

NU_Task_Information

举例

```
NU_TASK Task;
STATUS status;

/*终止任务控制块"Task"，假设"Task"在先前已经调用 Nucleus PLUS 服务
NU_Create_Task 创建*/

status = NU_Terminate_Task(&Task);
```

NU_Timer_Information

函数原型

```
STATUS NU_Timer_Information(  NU_TIMER *timer,
                              CHAR *name,
                              OPTION *enable,
                              UNSIGNED *expirations,
                              UNSIGNED *id,
                              UNSIGNED *initial_time,
                              UNSIGNED *reschedule_time);
```

描述

此项服务返回有关指定的应用程序定时器的各种信息。此项服务的参数详细定义如下：

参数	含义
timer	应用程序定时器指针
name	定时器名称 8 字符目标区域指针
enable	指向保存定时器当前使能状态的变量的指针。NU_ENABLE_TIMER 或 NU_DISABLE_TIMER
expirations	指向保存定时器到时次数的变量的指针
id	指向保存用户提供的 id 号的变量的指针
initial_time	指向保存初始化定时器到时值的变量的指针
reschedule_time	指向保存定时器重新调用值的变量的指针

返回值

此项服务完成状态定义如下：

状态	含义
NU_SUCCESS	表示服务成功完成
NU_INVALID_TIMER	表示定时器指针无效

任务更改

No

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

定时器服务

相关

NU_Create_Timer,NU_Delete_Timer,NU_Established_Timers,

NU_Timer_Pointers,NU_Reset_Timer

举例

```
NU_TIMER Timer;
CHAR timer_name[8];
OPTION enable;
UNSIGNED expirations;
UNSIGNED id;
UNSIGNED initial_time;
UNSIGNED reschedule_time;
STATUS status;

/*获得有关定时器控制块"Timer"的信息。假设"Timer"在先前已经调用 Nucleus PLUS 服务 NU_Create_Timer 创建。*/

status = NU_Timer_Information(&Timer,
                              timer_name,
                              &enable,
                              &expiration,
                              &id,
                              &initial_time
                              &reschedule_time);
/*如果 status 为 NU_SUCCESS,其他信息就是精确的*/
```

NU_Timer_Pointers

函数原型

```
UNSIGNED NU_Timer_Pointers(   NU_TIMER **pointer_list,
                              UNSIGNED maximum_pointers);
```

描述

此项服务为系统中所有已建立应用程序定时器的指针建立一个连续的指针列表。注：已经删除的定时器不再被认为是已建立。参数 pointer_list 指向家里指针列表的位置,maximun_pointers 表示列表最大尺寸。此项服务返回列表中指针的实际数量。另外，列表按最旧到最新成员排序。

任务更改

No

允许调用

Application_Initialize,HISR,Signal Handler,task

类别

定时器服务

相关

NU_Create_Timer,NU_Delete_Task,NU_Established_Timers,
NU_Timer_Information,NU_Reset_Timer

举例

```
/*定义一个可以容纳 20 个定时器指针的阵列*/
```

```
NU_TIMER *Pointer_Array[20];
UNSIGNED number;

/*获得当前激活的定时器指针的列表（最大为 20）*/

number = NU_Timer_Pointers(&Pointer_Array[0],20);

/*这里，number 包含列表中指针的实际数量*/
```

NU_Unprotect

函数原型

```
VOID NU_Unprotect(VOID);
```

描述

此项服务解除以前调用 NU_Protect 已建立的临界数据结构的优先级保护。因为 I/O 驱动器金昌必须保护不被任务和 HISR 组件同时访问，这个服务在 I/O 驱动器中被保留使用。任务同步通过使用任务同步服务来实现。

注：如果保护已经被清除，必须小心避免调用这个子程序。

任务更改

Yes

允许调用

HISR,task

类别

I/O 驱动器服务

相关

NU_Protect,NU_Suspend_Driver

举例

```
/*解除先前调用 NU_Protect 产生的保护*/

NU_Unprotect();
```

第五章　扩展讨论

本章未不同的 Nucleus PLUS 题目提供一个扩展讨论。

章节信息

5.1 内存使用
5.2 自定义服务
5.3 执行线程
5.4 中断处理
5.5 I/O 驱动器

5.1 内存使用

Nucleus PLUS 给应用程序提供为每个系统对象指定内存利用的能力。系统对象包括任务，HISRs，队列、管道、邮箱、信号量、事件标志集、内存分区池、动态内存池和 I/O 驱动器。上面谈到的每个系统对象都需要一个控制结构。一些系统对象需要附加内存。例如，任务创建需要控制块内存和堆栈

所需内存。所有系统对象所需内存在它们创建的时候提供。注：Nucleus PLUS 期望所有的系统对象驻留在上电期间清除（初始化为 0）的内存区上。

这项技术最大的好处就是灵活性。例如，假设一个目标板安装一个有限的高速内存。可以通过在高速内存区增加放置任务控制块和堆栈使得高优先级任务的运行速度显著提升。系统的其他任务可以使用更充足但是速度更低的内存区。当然，其他系统对象的运行可以以同样的方法提升。

系统对象的内存分配有几种方法。最简单的方法就是使用全局 C 数据结构进行分配内存。另一种方法就是可以从动态内存池或是分区内存池动态分配内存。第三种方法时从目标系统的绝对物理区域分配内存。

使用全局 C 数据结构对系统对象分配内存是分配控制结构最简单的方法。下面是为每种类型的系统对象进行控制块分配的例子：

系统对象	例子
NU_TASK	Example_Task
NU_HIST	Example_HISR
NU_DRIVER	Example_Driver
NU_QUEUE	Example_Queue
NU_MAILBOX	Example_Mailbox
NU_PIPE	Example_Pipe
NU_EMAPHORE	Example_Semaphore
NU_EVENT_GROUP	Example_Event_Group
NU_PARTITION_POOL	Example_Psrtition_Pool
NU_MEMORY_POOL	Example_Memory_Pool

Example_*为驻留在全局 C 数据区的控制块。适当的控制块的指针被传递到相应创建的服务。堆栈、队列区、内存池区和其他系统对象区也可以作为全局 C 数据结构分配，然而相对于后面的方法它没有什么吸引人的地方。注：局部 C 数据结构分配也是合法的。在函数内定义的对象在函数返回时不再使用。

从 Nucleus PLUS 内存池给系统对象分配内存非常常见。内存池是系统对象自身的，因此可以被创建用来管理其他的内存区。

下面是一个从已经创建的动态内存池（System_Memory 是先前创建的内存池中一个全局 C 控制块）分配任务控制块和一个 1000 字节堆栈的例子：

```
NU_TASK   *Example_Task_Ptr;
VOID      *Example_Stack_Ptr;

/*为任务控制结构分配内存*/
NU_Allocate_Memory(&System_Memory,
                   (VOID **)&Example_Task_Ptr,
                   sizeof(NU_TASK),
                   NU_NO_SUSPEND);
/*为任务堆栈配置内存*/
NU_Allocate_Memory(&System_Memory,
                   &Example_Stack_Ptr,
                   1000,NU_NO_SUSPEND);
/* 任 务 创 建 调 用 由 Example_Task_Ptr 和
Example_Stack_Ptr 提供*/
```

最后，系统对象内存分配最后一种方法覆盖了目标板上的内存区。假设地址 0X200000 是一个 4096 字节的高速内存区。第一个例子在这块内存区创建一个动态内存池。第二个例子给高优先级的任务的任务块和一个 2000 字节堆栈在高速内存区上分配内存。

例一：

```
NU_MEMORY_POOL       System_Memory;
/*创建一个动态内存池管理地址在 0x200000 的高速内存*/
NU_Create_Memory_Pool(&System_Memory,SYSTEM
                  (VOID *)0x200000,4096,20,NU_FIFO);
```

例二：

```
NU_TASK   *Example_Task_Ptr;
VOID      *Example_Stack_Ptr;
CHAR      *High_Speed_Mem_Ptr;

/*放置起始地址在高速内存指针*/
High_Speed_Mem_Ptr = (CHAR *) 0x200000;

/*在开始处分配任务控制块*/
Example_Task_Ptr = (NU_TASK *) High_Speed_Mem_Ptr;

/*调整高速内存指针*/
High_Speed_Mem_Ptr = High_Speed_Mem_Ptr + sizeof(NU_TASK);

/*分配任务堆栈区*/
Example_Stack_Ptr = (VOID *) High_Speed_Mem_Ptr;

/*万一需要更多分配，调整高速内存区指针*/
High_Speed_Mem_Ptr = High_Speed_Mem_Ptr + 2000;

/*用 Example_Task_Ptr & Example_Stack_Ptr 调用创建任务*/
```

5.2 定制服务

Nucleus PLUS 服务可以被组合成专门的应用程序服务。例如，假设一个应用程序需要确认每个通过邮箱发送到其他任务的消息。通过在两个邮箱上操作创建发送/接收服务，这很容易实现。一个邮箱用来真正的传送数据，另一个邮箱用于消息确认。新的发送服务把消息放入第一个邮箱然后从第二个邮箱等待确认。新的接收服务从第一个又想等待消息并通过第二个邮箱发送确认。

不同的 Nucleus PLUS 服务也可以组合成指定的应用程序服务。例如，允许在多邮箱、队列和管道上挂起的新的服务可以通过使用计数信号量和期望的通信机制来实现。在服务当中，计数信号量初始化为 0。每次一个消息被放入一个通信对象，信号量增加（释放）。在接收请求中，信号量第一个被获得。如果信号量为 0，任何通信对象中都没有消息且可以挂起。另外，如果信号量可以获得，最少有一个消息出现。每个通信对象检查直到消息出现。

5.2.1 多个邮箱挂起例程

本节包含 C 源代码实例说明在多个邮箱上的挂起。虽然此例用的是邮箱，可以很容易的应用于其他的通信机制，包括队列和管道。下面是自定义多邮箱挂起服务的服务接口：

MM_Create	创建一个多邮箱对象
MM_Send	发送消息至多邮箱交换
MM_Receive	从一个挂起的邮箱接收消息

为了使用新的多邮箱服务，多邮箱挂起头文件必须包括。

下面显示了一个假设多邮箱挂起头文件。（注：Nucleus PLUS 软件中不包含下面的例子文件）。

```
#include <nucleus.h>
/*这是多邮箱挂起服务控制结构*/
typedef struct MMAIL_STRUCT
{
    NU_SEMAPHORE  message_count;
    NU_MAILBOX       mailbox_1;
    NU_MAILBOX       mailbox_2;
    NU_MAILBOX       mailbox_3;
}MMAIL;

/*定义多邮箱服务接口*/
STATUS = MM_Create(MMAIL *mmail_control);
STATUS = MM_Send(MMAIL *mmail_control,INT mailbox_id,
                 VOID *message,UNSIGNED suspend);
STATUS = MM_Receive(MMAIL *mmail_control,
                 VOID *message,UNSIGNED suspend);
```

下面是一个多任务挂起服务的 C 源程序例子。（注：Nucleus PLUS 软件中不包含下面例子。）

```
/*创建一个多任务控制块*/
STATUS MM_Create(MMAIL *mmail_control)
/*创建单片控制块*/
NU_Create_Semaphore(&(mmail_control ->message_count),
                 “MMCOUNT”,0,NU_FIFO);
NU_Create_Mailbox(&(mmail_control ->mailbox_0),
                 “MM_MB_0”,NU_FIFO);
NU_Create_Mailbox(&(mmail_control ->mailbox_1),
                 “MM_MB_1”,NU_FIFO);
NU_Create_Mailbox(&(mmail_control ->mailbox_2),
                 “MM_MB_2”,NU_FIFO);

/*返回一个成功状态，虽然状态应该从每个先前调用被检测。*/
```

```
    return(NU_SUCCESS);
    }

    /*发送一个消息到多邮箱对象*/
    STATUS MM_Send(MMAIL *mmail_control, INT Mailbox_id,
                   VOID   *message,
                   UNSIGNED  suspend)
    {
        STATUS status;
        /*决定哪个邮箱发送请求*/
        if (mailbox_id == 0)
        /*发送消息至邮箱 0*/
        status = NU_Send_To_Mailbox(&(mmail_control ->mailbox_0),
                                message,suspend);
        else if (mailbox_id == 1)
        /*发送消息到邮箱 1*/
        status = NU_Send_To_Mailbox(&(mmail_control ->mailbox_1),
                                message,suspend);
        else
        /*发送消息至邮箱 2*/
        status = NU_Send_To_Mailbox(&(mmail_control ->mailbox_2),
                                message,suspend);

        /*确定是否消息发送成功*/
        if (status == NU_SUCCESS)
            /*递增计数信号量*/
            NU_Release_Semaphore(&(mmail_control -> message_count));
            /*返回状态至调用*/
            return(status);
        }
        /*从多邮箱中一个接收消息*/
        STATUS MM_Receive(MMail *mmail_control,VOID *message,
                          UNSIGNED suspend)
        {
            STATUS status;
            /*获得消息信号量*/
            status       =       NU_Obtain_Semaphore(&(mmail_control
    ->message_count),suspend);
            /*确定消息是否出现*/
            if (status = NU_SUCCESS)
            {
                /*寻找第一个有效消息，从邮箱 0 开始*/
                status    =    NU_Receive_From_Mailbox(&(mmail_control
    ->mailbox_0),
```

```
                                        message,NU_NO_SUSPEND);
            /*确定下一个邮箱是否需要被查询*/
            if (status != NU_SUCCESS)


            /*检查邮箱 1*/
            status                                                  =
NU_Receive_From_Mailbox(&(mmail_control->mailbox_1),
                                        message,NU_NO_SUSPEND);
            else
            /*消息必须在邮箱 2 中*/
               status                                               =
NU_Receive_From_Mailbox(&(mmail_control->mailbox_2),

   message,NU_NO_SUSPEND);
            /*返回一个完整的状态到调用*/
            return(status);
        }
    }
```

5.3 执行线程

一个 Nucleus PLUS 应用程序总是八个可能运行线程中的一个。下面是所有那可能运行线程的列表。

初始化（Initialization）
系统错误（Ssytem Error）
.调度循环（Scheduling Loop）
.任务（Task）
.信号处理器（Signal Handler）
.用户中断处理子程序（User ISR）
.LISR
.HISR

5.3.1 初始化线程

初始化线程是系统运行的第一个线程。初始化线程的入口点为 INT_Initialize。在 Application_Initialize 函数返回之后，初始化线程中止，控制权转移到调度循环（Scheduling Loop）上。

5.3.2 系统错误线程

有几个可能发生的系统错误，它们中大多数在初始化期间被跟踪。然而，堆栈溢出的条件在任务和 HISR 运行时跟踪。这个线程在函数 ERC_System 错误调用时执行。默认情况，系统错误是致命的，因此在这个线程中控制暂停。看附录 B 中系统错误代码。

5.3.3 调度循环（Scheduling Loop）

调度循环发生入口在 TCT_Schedule。这个线程执行响应转移控制权到最高优先级的 HISR 上或任务就绪运行。当没有任务或 HISRs 就绪运行，控制暂停在一个 TCT_Schedule 简单的循环中。

5.3.4 任务线程

任务线程描述了应用程序的处理线程的多数情况。每个任务线程有它自己的堆栈。每个任务线程的入口在任务创建时指定。任务线程对完全访问 Nucleus PLUS 服务没有限制。

5.3.5 信号处理器线程

信号处理线程运行在相关任务线程的顶端。信号处理器线程对 Nucleus PLUS 服务访问有限制。主要的限制就是不允许自挂起。

5.3.6 用户 ISR 线程

用户中断服务子程序线程是典型的少量汇编语言子程序，可以直接挂到中断向量。此线程负责保存和恢复所有使用的寄存器。对此类型线程，Nucleus PLUS 服务完全不受限。实际上，C 函数也不受限，除了线程不能通过编译器保存和恢复所有使用的寄存器。

5.3.7 LISR 线程

低级中断服务子程序在 Nucleus PLUS 中注册。这样就允许 Nucleus PLUS 保存和恢复所有有需要的寄存器。因此 LISR 线程可以用 C 语言写。LISR 线程对 Nulceus PLUS 服务访问受限。最重要的就是 active_HER 服务。

下列服务从 LISRs 访问有效：

NU_Activate_HISR
NU_Local_Control_Interrupts
NU_Current_HISR_Pointer
NU_Current_Task_Pointer
NU_License_Information
NU_Retrieve_Clock

5.3.8 HISR 线程

高级中断服务子程序形成了 Nucleus PLUS 中断的第二部分。HISR 线程和任务线程一样被调度，也可以调用大多数 Nucleus PLUS 服务。然而，HISR 线程不允许自挂起请求。HISR 子程序入口点在 HISR 创建期间确定。

5.4 中断处理

Nucleus PLUS 既支持可控（managed）也支持不可控（unmanaged）ISRs。可控的 ISR 就是不需要存储和释放上下文信息，不可控 ISRs 对保存和恢复所有使用寄存器负全责。可控 ISRs 可以用 C 或汇编编程。不可控 ISRs 一般都用汇编。

5.4.1 可控（Managed）ISRs

可控 ISRs 在文中可以参照低优先级中断服务子程序（LISR）。LISRs 和传统 ISR 运行方式相同，除了所有上下文压栈和恢复由 Nucleus PLUS 管理。

下面是定义一个 LISR 函数并登记到 10 号向量的例子程序段：

```
VOID (*old_lisr) (INT);
VOID Example_LISR(INT vector);
INT Interrupt_Count = 0;

/*登记 LISR 为 10 号向量。先前登记的 LISR 返回到 old_lisr*/
NU_Register_LISR(10,Example_LISR,&old_lisr);

/*10 号向量 LISR 的实际定义*/
VOID   Example_LISR(INT vector)
```

```
    /*递增全局中断计数器*/
    Interrupt_Count++;
```

当中断 10 发生，Example_LISR 被调用，向量参数置为 10。中断处理包括递增全局变量，在 Example_LISR 返回时完成。非常值得注意的是 LISRs 对 Nucleus PLUS 服务访问极端受限。例如，如果任务必须作为中断 10 的结果被恢复，一个高优先级的中断服务子程序必须被 LISR 激活。

下面的例子就是在中断 10 发生时，恢复 Task_0_Ptr 指向的任务：

```
    extern NU_TASK *Task_0_Ptr;
    NU_HISR HISR_Control;
    CHAR   HISR_Stack[500];
    VOID   (*old_lisr) (INT);
    VOID   Example_LISR(INT vector);

    /*创建 HISR。此 HISR 由中断向量 10 的 LISR 激活*/
    NU_Create_HISR(&HISR Control,"EXMPHISR",
                   Example_HISR,2,HISR_Stack,500);

    /*登记 LISR 为中断向量 10。先前登记 LISR 返回 old_lisr 指定处*/

    /*中断向量 10 的 LISR 实际定义*/
    VOID Example_LISR(INT vector)
    {
        /*激活 Example_HISR 恢复 Task_0_Ptr 指定的任务。不允许从
LISR 调用大多数 Nucleus PLUS 服务*/
        NU_Activate_HISR(&HISR_Control);
    }
        /*Example_LISR 函数关联的 HISR 详细定义*/
    VOID Example_HISR(void)
    {
        /*恢复 Task_0_Ptr 指定的任务*/
        NU_Resume_Task(Task_0_Ptr);
    }
```

5.4.2 不可控 ISRs（Unmanage ISRs）

Nucleus PLUS 通过直接访问中断向量表（在大多数处理器架构中）支持不可控 ISRs。NU_Setup_Vector 服务可以用来联合一个指定的中断向量到不可控 ISR。二者选一，不可控 ISR 的地址可以直接放到 Nucleus PLUS 向量表中，在 INT.S 文件中定义。

不可控 ISRs 最典型的用法就是用于高频率中断。上下文信息（压栈资料）保存与恢复所消耗的资源数量与中断的频度成比例。当中断之间的时间与保存和恢复上下文信息所需时间取得相似成就时，就需要使用不可控 ISRs。例如，如果一个中断每 30uS 发生发生一次且可控中断需要 15uS 的消耗，在中断管理上，处理能力丢失了一半。

假设一个虚构的处理器有 32 个寄存器，命名为 r0～r31。现在假设每 30uS 有一个中断发生，此外，ISR 需要作的指示在一些内存映射区放置一个 1。下面就是一个满足要求的最小 ISR 例子：

```
Minimal_ISR:
Push r0   ;保存 r0
mov 1,r0  ;放置 1 到 r0
pop r0 ;弹出 r0
iret      ; 中断返回
```

如果这个虚拟的处理器中一个完全可控中断需要 15uS 保存和恢复所有 32 个寄存器，这个最小的 ISR 可能只需要 1uS。如果只消耗 1uS 的话一个 30uS 的中断完全是可行的。

不幸的是，不是所有的高频率的中断处理都如此简单。在很多情况下，这种中断依靠数据的实用性运行。最普通的处理这种情况的技术包括缓冲中断信息。为应用程序任务的处理，最小的 ISR 在全局内存区连续或者是周期性操作数据。创建最小 ISR，管理缓冲区数据且偶尔调用一下 Nucleus PLUS 也是一种可选的方法。

下面是上述的最小 ISR，偶然交叉调用 Nucleus PLUS（假设 LISR/HISR 提前定义）

```
Minimal ISR:
Push r0 ; 保存 rO
mov 1, r0; 放置1到 r0
mov r0, mem_map_loc ; 设置内存映射区

; 在这个区域进行缓冲器处理
; 查看缓冲溢出条件是否符合。如果符合，调用Nucleus PLUS唤醒
; 任务0
mov buffer_full, r0 ; 把缓冲区已满代码放入r0
cmp r0, 1 ; 如果缓冲区没有满，只是
jne Fast_Interrupt ; 运行快速中断

; 调用Nucleus PLUS上下文保存子程序
pop r0 ; 弹出到r0
call TCT_Interrupt_Context_Save
;
mov 10, r0 ; 放置向量号至r0
push r0 ; 压栈
call Example_LISR ;调用Example_LISR激活HISR，其实就是
                  ; 任务0
POP r0 ; 清除堆栈
;
; 恢复上下文，注意控制没有返回
jmp TCT_Interrupt_Context_Restore
Fast Interrupt:
pop r0 ; 恢复r0
```

```
        iret ; 中断返回
```

显然，上例是用汇编程序为一个虚拟处理器写的例子。这种中断处理的详细示例可以为给定的处理器进行移至。

5.5 I/O 驱动器

Nucleus PLUS 支持定制 I/O 驱动器。下面章节描述 Nucleus PLUS I/O 驱动器如何被使用，它们是如何构造的。有关指定驱动器的细节在不同的文档中分述。

5.5.1 运用 I/O 驱动器

Nucleus PLUS 提供一套基本 I/O 驱动器设备。这些设备帮助提供了一个与周边硬件支持无关的一致的驱动器接口。Nucleus PLUS 提供的基本 IO 驱动器设备如下：

- 创建 IO 驱动器
- 删除 IO 驱动器
- 请求 IO 驱动器

在 IO 驱动器可以使用之前必须被创建。这个由调用 Nucleus PLUS 中 NU_Create_Driver 服务完成。IO 驱动器的创建使得它被系统的其余部分所知。注：创建的 IO 驱动器在创建期间不能被访问。如果 IO 驱动器不再需要了可以被删除。Nucleus PLUS 服务 NU_Delete_Driver 执行此项功能。一个删除的 IO 驱动器不再被访问。

实际的驱动器请求

应用程序应用 NU_Request_Driver 服务请求创建驱动器。这个函数的主要目的是传送提供的驱动器请求结构到指定 IO 驱动器的入口函数。驱动器请求结构包含关于驱动器请求的所有信息。这样调和了 IO 驱动器不同的需求。例如，一个磁盘 IO 驱动器的输入请求通常不同于终端 IO 驱动器的输入请求。比较有代表性的，磁盘驱动器输入请求要求除了可读字节数和缓存指针还需要一个磁盘上的启动位置（扇区号）。一个终端驱动输入请求不需要任何偏移信息。

Nucleus PLUS 支持初始化（initialization）、**指派**（assign）、释放（release）、输入（input）、输出（output）、状态（status）和终止（terminate）IO 驱动器请求。当然，每个请求的参数依据真正的 IO 驱动器可能不同。

在 NU_Request_Driver 服务返回之后，实际 IO 请求的状态可能通过对请求结构中 nu_status 字段检查来决定。如果状态字包含 NU_SUCCESS,请求成功。如果请求无效，状态字的内容为 NU_INVALID_ENTRY。最后，如果在请求处理期间遇到一个 IO 错误，nu_status 字段被设置为 NU_IO_ERROR。附加错误信息可以由指定的 IO 驱动器添加。

初始化

一个初始化请求必须在 IO 驱动器创建之后和任何其他驱动器请求之前发生。请求用于初始化操作设备和内部设备控制结构。

下面是一个初始化请求的小代码段：

```
NU_DRIVER_REQUEST request;
STATUS status;

/*为一个简单的 IO 驱动器建立初始化请求。*/
request.nu_function = NU_INITIALIZE;
request.nu_timeout = NU_NO_SUSPEND;
/*发送初始化请求到预先创建的 IO 驱动器，指针为“driver”*/
status = NU_Request_Driver(driver,&request);
```

/*可变状态表示驱动器请求是否通过，请求中 nu_status 字段表示初始化请求的完成状态*/

指派

一个**指派**请求的发出市为了为了保护能够同步访问多任务的驱动器。举个例子，如果两个任务发送字符串到终端处理器，一次一个字节，在屏幕上字符串被混合成垃圾。在打印字符串之前如果每个任务获得唯一的访问驱动器入口，这个问题就解决了。

下面是**指派**请求的代码段：

```
NU_DRIVER_REQUEST request;
STATUS status;

/*为一个简单的 IO 驱动器建立一个指派请求*/
request.nu_function = NU_ASSIGN;
request.nu_timeout = NU_NO_SUSPEND;

/*发送指派请求到“drive”指向的驱动器*/
status = NU_Request_Driver (driver,&request);
```

/*可变状态表示驱动器请求是否通过，请求中 nu_status 字段表示**指派**请求的完全状态*/

释放

释放请求移除一个先前的指派。如果其他任务等待指派驱动器，指派传输到第一个等待的任务。下面是一个释放请求的小代码段：

```
NU_DRIVER_REQUEST request;
STATUS status;

/*为一个简单的 IO 驱动器建立释放请求*/
request.nu_function = NU_RELEASE;
request.nu_timeout = NU_NO_SUSPEND;
/*发送释放请求到“driver”指定的驱动器*/
status = NU_Request_Driver(driver,&request);
```

/*可变状态描述到驱动器的请求是否通过，请求中的 nu_status 字段表示释放请求的完全状态*/

输入

一个输入请求指示驱动器从相关设备获得确定数量的数据。下面是输入请求的一小段代码：

```
CHAR buffer[100];
NU_DRIVER_REQUEST request;
STATUS status;
```

```
            /*为一个简单 IO 驱动器建立输入请求*/
            request.nu_function = NU_INPUT;
            request.nu_timeout = NU_NO_SUSPEND:
            request.nu_request_info.nu_input.nu_buffer_ptr =
(VOID *)buffer;
            request.nu_request_info.nu_input.nu_request_size =
100;

            /*发送输入请求至“driver”指定驱动器*/
            status = NU_Request_Driver(driver,&request);
/*如果 status 和 request.nu_status 都成功，buffer 包含了实际数据*/
```

输出

一个输出请求指示驱动器发送指定数量的数据到相关设备。下面是输出请求的一小段代码：

```
CHAR   buffer[100];
NU_DRIVER_REQUEST   request;
STATUS    status;

/*给一个简单 IO 驱动器建立输出请求*/
request.nu_function = NU_OUTPUT;
request.nu_timeout = NU_NO_SUSPEND;
request.nu_request_info.nu_output.nu_buffer_ptr = (VOID
*)buffer;
request.nu_request_info.nu_output.nu_buffer_size = 100;
/*发送输出请求到“driver”指定的驱动器*/
status = NU_Request_Driver(driver,&request);

/*如果 status 和 request.nu_status 都成功,buffer 内容可以实际写出*/
```

状态

状态请求依赖于 IO 驱动器，是比较典型的。在驱动器控制结构中，驱动器名称总是有用的，字段名为 nu_driver。下面是状态请求的小代码段：

```
NU_DRIVER_REQUEST   request;
STATUS    status;

/*给一个简单的驱动器建立状态请求*/
request.nu_function = NU_STATUS;
/*发送状态请求至“driver”指定的驱动器*/
status = NU_Request_Driver(driver,&request);
/*如果 status 等于 NU_SUCCESS，驱动器成功调用，request.nu_status
的值和其他可能字段一起，依赖于驱动器的*/
```

终止

终止请求依赖于 IO 驱动器，是可配置的。一些驱动器在他们被删除和重新初始化之前需要终止请求。

下面是终止请求的一个小代码段：

```
NU_DRIVER_REQUEST   request;
STATUS    status;

/*给一个简单驱动器建立一个终止请求*/
request.nu_function = NU_TERMINATE;
/*发送终止请求值“driver”指定的驱动器*/
status = NU_Request_Driver(driver,&request);

/*如果 status 等于 NU_SUCCESS,驱动器成功终止。在这里，驱动器可以被删除和重新初始化。*/
```

5.5.2 驱动器执行

到这里，IO 驱动器信息干预了如何使用 IO 驱动器。本节覆盖了 IO 驱动器看上去象什么。

IO 驱动器基本上是一个带开关陈述的 C 函数。它们通常包括 LISR 和 HISR 中断处理器和自定义函数。所有 Nucleus PLUS IO 驱动器有一个类似于下面模板的入口函数：

```
VOID   Driver_Entry(NU_DRIVER *driver,NU_DRIVER_REQUEST
*request)
{
    /*根据请求进行处理*/
    switch(request -> nu_function)
    {
        case NU_INITIALIZE:
            /*初始化处理。注：“drive”中 nu_info_ptr 字段对驱动器的应
            用是有效的*/
        break;
        case NU_ASSIGN:
            / *指派处理*/
        break;
        case NU_RELEASE:
            /*释放处理*/
        break;
        case NU_INPUT:
            /*输入处理*/
        break;
        case NU_OUTPUT:
            /*输出处理*/
        break;
        case NU_STATUS;
```

```
            /*状态处理*/
        break;
        case NU_TERMINATE:
            /*终止处理*/
        break;
        default:
            /*错误请求处理*/
        break;
    }
    /*驱动器请求结束，返回调用*/
}
```

在驱动器中驱动控制结构（NU_DRIVER）有几个有效字段。

字段	含义
nu_info_ptr	驱动器详细信息指针。如果使用，此字段一般在为 IO 驱动器指定的追加结构类型初始化期间设置
nu_driver_name	一个八字节的名字，与 IO 驱动器相关的。

5.5.2.1 驱动器例程

下面的代码段描述了一个 MS－DOS 系统的小终端 IO 驱动器。驱动器支持单个字节输入和输出请求。注：驱动器只能从任务线程被访问。

```
#include <stdio.h>
#include “nucleus.h”
/*最小终端驱动器例程入口函数*/
VOID Terminal_Driver(NU_DRIVER *driver,
                     NU_DRIVER_REQUEST *request)
char *pointer

/*处理请求*/
switch(request -> nu_function)
{
    case NU_INITIALIZE:
        /*初始化不作任何事情*/
    break;
    case NU_INPUT:
        /*等待用户按键*/
        while(!kbhit())
        {
            /*休眠一个节拍允许其他任务运行*/
            NU_Sleep(1);
        }
        /*安装输入字符指针*/
```

```
                pointer = (char *)request ->
            nu_request_info.nu_input.nu_buffer_ptr;

                /*字符出现，把它读入到提供目的地址*/
                *pointer = (char)getch();

                /*成功完成*/
                request -> nu_status = NU_SUCCESS;
                break;

                case NU_OUTPUT:
                /*安装输出字符指针*/
                pointer = (char *)request ->
            nu_request_info.nu_output.nu_buffer_ptr;
                /*调用 putch 函数打印提供字符*/
                putch((int) *pointer);
                /*成功完成*/
                request -> nu_status = NU_SUCCESS;
                break;

                default:
                    /*错误请求处理*/
                    request -> nu_status = NU_INVALID_ENTRY;
                break;
        }
        /*结束驱动器请求，返回调用*/
```

第六章　样例系统

本章提供一个 Nucleus PLUS 系统样例的描述和完整代码列表。

章节

6.1 样例概况

6.2 样例系统

6.1　样例概况

样例系统在本章描述，包含了一个应用程序初始化（Application_Initialize）函数和六个任务。所有任务都在初始化期间创建。除了任务运行，任务通信和同步在系统中都有实例。

在样例系统代码列表中，数据结构咱第 5 和第 26 行之间定义。Nucleus PLUS 控制结构在第 7 和第 16 行之间定义。

Application_Initialize 在第 36 行开始，在第 93 行结束。

在这个例子中，所有系统对象（任务、队列、信号量和事件标志集）都在初始化期间创建。样例系统的任务在第 47 行和 80 行之间创建。通信队列在第 85 行创建。系统信号量在第 89 行创建。最后，系统事件标志集在第 92 行创建。注：由第一个有效内存参数指定开始地址的 20000 字节内存池在第 47 行首次创建。这个内存池用于分配所有的任务堆栈和真正的队列区。

任务 0 是系统启动后第一个运行的程序。这是因为任务 0 是系统中优先级最高（priority 1）的任务。任务 3 在任务 0 挂起后运行（priority 5）。任务 4 在任务 3 挂起后执行。认识到为什么任务 3 在任务 4 之前运行非常重要，尽管他们有相同的优先级。原因就是任务 3 创建且第一个启动（看 Application_Initialize）。同样优先级任务暗转他们就绪运行的顺序运行。在任务 4 挂起后，任务 5 运行（priority 7）。在任务 5 挂起后，任务 1 运行（priority 10）。最后任务 2 在任务 1 因为队列满条件挂起后运行（priority 10）。

任务 0 在第 101 和 126 行之间定义。和所有样例系统中的任务一样，任务 0 做一些预备的初始化之后开始执行一个无限循环。任务 0 无限循环的处理包括连续调用 NU_Sleep 和 NU_Set_Events。因为 NU_Sleep 的调用，任务 0 循环每 18 个定时器节拍运行一次。注：任务 5 在每次调用 NU_Set_Events 函数时进入就绪状态。一旦任务 5 优先级低于任务 0，在任务 0 再次运行 NU_Sleep 调用之前，任务 5 不会运行。

任务 1 在第 123 行和第 165 行之间定义。任务 1 连续发送单个 32 位消息到队列 0。当管道充满时，任务 1 挂起，直到队列 0 空间有效（有可用空间）。任务 1 的挂起允许任务 2 恢复运行。

任务 2 在第 181 和第 223 行之间定义。任务 2 连续从队列 0 接收单个 32 位消息。当队列为空时，任务挂起，直到队列 0 空间有效（有可用消息）。任务 1 的挂起允许任务 2 恢复运行。

任务 3 和任务 4 共享同样的指令代码。然而，每个任务有它自己的唯一堆栈。任务 3 和任务 4 在 232 和 262 行之间定义。每个任务竞争一个二进制的信号量。一旦信号量被获得，在再次释放信号量之前任务休眠 100 个时钟节拍。这种行为允许其他尝试获得同一信号量的任务运行和挂起。当信号量被释放时，等待信号量的挂起任务运行。

任务 5 在 267 和 292 行之间定义。此任务在一个死循环中等待事件标志被置位。被等待的事件标志被任务 0 置位。因此，任务 5 和任务 0 以同样的频率运行。

6.2 样例系统

下面时样例系统的源代码列表。注：左边的行号不是真正的文件中的一部分。这里的代码仅供参考。

```
1   /*包含必要的 Nucleus PLUS 文件*/
2   #include “nucleus.h”
3
4
5   /*定义应用程序数据结构*/
6
7   NU_TASK       Task_0;
8   NU_TASK       Task_1;
9   NU_TASK       Task_2;
10  ;
```

```
NU_TASK       Task_4;
NU_TASK       Task_5;
NU_QUEUE      Queue_0;
NU_SEMAPHORE  Semaphore_0;
NU_EVENT_GROUP   Event_Group_0;
NU_MEMORY_POOL   System_Memory;


/*分配全局计数器*/
UNSIGNED Task_Time;
UNSIGNED Task_2_message_received;
UNSIGNED Task_2_ invalid_messages;
UNSIGNED Task_1_messages_sent;
NU_TASK *Who_has_the_resource;
UNSIGNED Event_Detections;


/*定义引用函数的原型*/
void task_0(UNSIGNED argc,VOID *argv);
void task_1(UNSIGNED argc,VOID *argv);
void task_2(UNSIGNED argc,VOID *argv);
void task_3_and_4(UNSIGNED argc,VOID *argv);
void task_5(UNSIGNED argc,VOID *argv);


/*定义应用程序初始化子程序，初始化子程序决定初始化 Nucleus PLUS 应用程
序环境*/

void Application_Initialize(void *first_available_memeory)
{
   VOID *pointer;



 /*创建一个系统内存池将用于分配任务堆栈，队列区域等*/

   NU_Create_Memory_Pool(&System_Memory,”SYSTEM”,
                  first_available_memory,20000,50,NU_FIFO);

 /*创建系统中的每个任务*/

/*创建任务 0*/
NU_Allocate_Memory(&System_Memory,&pointer,1333,
NU_NO_SUSPEND);
```

```
54 NU_Create_Task(&Task_0,"TASK_0",task_0,0,NU_NULL,pointer,
55                              1000,1,20,NU_PREEMPT,NU_START);
56
57 /*创建任务 1*/
58 NU_Allocate_Memory(&System_Memory,&pointer,1000,
   NU_NO_SUSPEND);
59 NU_create_Task(&Task_1,"TASK_1",task_1,3,NU_NULL,pointer,
60                              1000,10,5,NU_PREEMPT,NU_START);
61
62 /*创建任务 2*/
63 NU_Allocate_Memory(&System_Memory,&pointer,1333,
   NU_NO_SUSPEND);
64 NU_Create_Task(&Task_2,"TASK_2",task_2,0,NU_NULL,pointer,
65                 1000,13,5,NU_PREEMPT,NU_START);
66
67 /*创建任务 3。注意任务 4 用的同一个指令区*/
68 NU_Allocate_Memory(&System_Memory,&pointer,1000,
   NU_NO_SUSPEND);
69 NU_Create_Task(&Task_3,"TASK_3",task3_and_4,0,NU_NULL,
70                pointer,1000,5,0,NU_PREEMPT,NU_START);
71
72 /*创建任务 4。注意任务 3 使用同样的指令区*/
73 NU_Allocate_Memory(&System_Memory,&pointer,1000,
   NU_NO_SUSPEND);
74 NU_Create_Task(&Task_4,"TASK_4",task_4,3,NU_NULL,pointer,
75                 1000,5,0,NU_PREEMPT,NU_START);
76
77 /*创建任务 5*/
78 NU_Allocate_Memory(&System_Memory,&pointer,1000,
   NU_NO_SUSPEND);
79 NU_Create_Task(&Task_5,"TASK_5",task_5,0,NU_NULL,pointer,
80                 1000,7,0,NU_PREEMPT,NU_START);
81
82 /*创建通信队列*/
83 NU_Allocate_Memory(&System_Memory,&pointer,
   100*sizeof(UNSIGNED),NU_NO_SUSPEND);
84 NU_Create_Queue(&Queue_0,"QUEUE_0",pointer,100,
   NU_FIXED_SIZE,1,NU_FIFO);
85
86 /*创建同步信号量*/
87 NU_Create_Semaphore(&Semaphore_0,"SEM_0",1,NU_FIFO);
88
89 /*创建事件标志集*/
90 NU_Create_Event_Group(&Event_Group_0,"EVGROUP0");
```

```
91
92
93
94  /*定义任务 0。任务 0 每 18 个时钟节拍递增一次 Task_Time 变量。另外，任务
95  0 置位任务 5 正在等待的事件标志集，每个循环一次反复*/
96
97
98
99
100
101 void task_0(UNSIGNED argc,VOID *argv)
102{
103
104 STATUS status;
105
106
107 /*访问 argc 和 argv 只是为了避免编辑警告*/
108 status = (STATUS) argc + (STATUS)argv;
109
110/*设置时钟为 0。这个时钟每 18 个系统定时器节拍为一拍。*/
111 Task_Time = 0;
112
113 while(1)
114 {
115
116 /*休眠 18 个定时器节拍。在 IND.ASM 中时钟节拍值可编程，且与目标系统的
117速度有关*/
118 NU_Sleep(18);
119
120 /*递增次数*/
121 Task_Time++;
122
123 /*设置事件标志来消除任务 5 的挂起*/
124 NU_Set_Events(&Event_Group_0,1,NU_OR);
125 }
126 }
127
128
129 /*定义队列发送任务。对列满条件和配置文件指定的时间片导致任务挂起。*/
130
131
132
133 void task_1(UNSIGNED argc,void *argv)
134 {
```

```
135
136 STATUS status;
137 UNSIGNED Send_Message;
138
139 /*访问 argc 和 argv 只是为了避免编辑警告*/
140 status = (STATUS)argc + (STATUS)argv;
141
142 /*初始化消息计数器*/
143 Task_1_messages_send = 0;
144
145 /*初始化消息内容。接收器将检查消息内容是否错误。*/
146
147 Send_Message = 0;
148
149 while(1)
150 {
151
152 /*发送消息到 Queue_0,任务 2 从里面读。注意如果目标队列充满，这个任务挂
153 起直到空间有效*/
154
155 status = NU_Send_To_Queue(&Queue_0,&Send_Message,1,
156                           NU_SUSPEND);
157
158 /*确定消息是否发送成功*/
159 if (status == NU_SUCCESS)
160 Task_1_message_sent++;
161
162 /*修改下一个发送消息的内容*/
163 Send_Message++;
164
165
166
167
168 /*定义队列接收任务。注意队列空条件和配置文件指定的时间片导致任务挂起。
169 */
170
180
181 void task_2(UNSIGNED argc,VOID *argv)
182
183
184 STATUS status;
185 UNSIGNED Receive_Message;
186 UNSIGNED received_size;
187 UNSIGNED message_expected;
```

```
188
189 /*访问 argc 和 argv 只是为了避免编辑警告*/
190 status = (STATUS)argc + (STATUS)argv;
191
192 /*初始化消息计数器*/
193 Task_2_message_received = 0;
194
195 /*初始化消息错误计数器*/
196 Task_2_invalid_messages = 0;
197
198 /*初始化消息内容为期望值*/
199 message_expected = 0;
200
201 while(1)
202
203
204 /*从 Queue_0 重新获得任务 1 写入的消息，注意如果源队列为空，这个任务挂
205 起知道有东西可用*/
206
207 status = NU_Receive_From_Queue(&Queue_0,&Receive_Message,1,
208                                  &received_size,NU_SUSPEND);
209
210                                  /*确定消息是否接收正确*/
211 if (status == NU_SUCCESS)
212 Task_2_message_received++;
213
214 /*检测消息内容是否与任务期望一致*/
215
216 if ((received_size !=) || (Receive_Message !=
217                                message_expected))
218 Task_2_invalid_messages++;
219
220 message_expected++;
221
222
223
224
225
226 /*Task_3_and_4 只想要单个资源。一旦其中一个任务获得资源，它将在释放
227 它之前保持 33 个时钟节拍。在这期间，其他任务挂起等待这个资源。注意任务 3
228 4 和都使用同样的代码区但是堆栈不一样。*/
229
230
231
```

```
232 void task_3_and_4(UNSIGNED argc,VOID *argv)
233 {
234
235 STATUS status;
236
237 /*访问 argc 和 argv 只是为了避免编辑警告*/
238 status = (STATUS)argc + (STATUS)argv;
239
240 /*循环分配和收回资源*/
241 while(1)
242 {
243
244  /*分配资源。挂起知道它变得有效*/
245  status =
246        NU_Obtain_Semaphore(&Semaphore_0,NU_NU_SUSPEND);
247  /*如果 status 为成功，显示这个任务拥有资源*/
248
249  if (status == NU_SUCCESS)
250  {
251
252   Who_has_the_resource = NU_Create_Task_Pointer();
253
254   /*休眠 100 个时钟节拍导致其他任务挂起等待资源*/
255
256   NU_Sleep(100);
257
258   /*释放信号量*/
259   NU_Release_Semaphore(&Semaphore_0);
260   }
261   }
262   }
263
264
265   /*定义等待任务 0 设置事件的任务*/
266
267   void task_5(UNSIGNED argc,VOID *argv)
268   {
269
270    STATUS status;
271    UNSIGNED event_group;
272
273
274    /*访问 argc 和 argv 只是为了避免编辑警告*/
275    status = (STATUS)argc + (STATUS)argv;
```

```
276
277   /*初始化事件跟踪计数器*/
278   Event_Detections = 0;
279
280   /*永远继续这个处理*/
281   while(1)
282   {
283
284    /*等待一个事件且消耗它*/
285   status = NU_Retrieve_Events(&Event_Group_3,1,
286           NU_OR_CONSUME,&event_group,NU_SUSPEND);
287
288  /*如果 status 成功，递增计数器*/
289  if (status == NU_SUCCESS)
290     Event_Detections++;
291  }
292  }
```