L4 eXperimental Kernel

Reference Manual

Version X.2

分析文档

刘伟浩

2017.12.31

目录

[关于手册 2](#_Toc503985593)

[手册理解 2](#_Toc503985594)

[第一章 基本内核接口 4](#_Toc503985595)

[1.1 内核接口页 [数据结构] 4](#_Toc503985596)

[1.2 KernelInterface [slow 系统调用] 11](#_Toc503985597)

[1.3 虚拟寄存器 [虚拟寄存器] 12](#_Toc503985598)

[第二章 线程 13](#_Toc503985599)

[2.1 ThreadId [数据类型] 13](#_Toc503985600)

[2.2 线程控制寄存器（TCRs） [虚拟寄存器] 15](#_Toc503985601)

[2.3 ExchangeRegisters [系统调用] 17](#_Toc503985602)

[2.4 ThreadControl [特权系统调用] 22](#_Toc503985603)

[第三章 调度 25](#_Toc503985604)

[3.1 Clock [数据类型] 25](#_Toc503985605)

[3.2 SystemClock [系统调用] 25](#_Toc503985606)

[3.3 Time [数据类型] 25](#_Toc503985607)

[3.4 ThreadSwitch [系统调用] 26](#_Toc503985608)

[3.5 Schedule [系统调用] 27](#_Toc503985609)

[3.6 抢占标志 [线程控制寄存器的一个字段] 29](#_Toc503985610)

[第四章 地址空间和映射 31](#_Toc503985611)

[4.1 Fpage [数据类型] 31](#_Toc503985612)

[4.2 Unmap [系统调用] 32](#_Toc503985613)

[4.3 SpaceControl [特权系统调用] 34](#_Toc503985614)

[第五章 IPC 36](#_Toc503985615)

[5.1 消息和消息寄存器（MRs） [虚拟寄存器] 36](#_Toc503985616)

[5.2 MapItem [数据类型] 38](#_Toc503985617)

[5.3 GrantItem [数据类型] 39](#_Toc503985618)

[5.4 CtrlXferItem [数据类型] 40](#_Toc503985619)

[5.5 StringItem [数据类型] 41](#_Toc503985620)

[5.6 字符缓冲和缓冲寄存器（BRs） [虚拟寄存器] 42](#_Toc503985621)

[5.7 Ipc [系统调用] 43](#_Toc503985622)

[第六章 其他 50](#_Toc503985623)

[6.1 异常处理程序 [TCR一个字段] 50](#_Toc503985624)

# 关于手册

## 手册理解

这个L4参考手册是所有L4 API和ABI的定义文档。 阅读本参考手册需要读者对L4的基本概念和硬件体系结构有深入了解。本手册的重点是精确的定义，而不是解释和说明。

此 L4 参考手册定义了所有32位和64位计算机的通用 API。因此, 通用参考手册是独立于特定处理器体系结构。它还辅以处理器特定的 ABI 规范。

在这篇文档中, 我们区分逻辑接口, 通用二进制接口、通用编程接口、便捷编程接口和特定于处理器的二进制接口。

* **逻辑接口**。逻辑接口定义所有概念和逻辑对象, 如系统调用操作、逻辑数据对象、数据类型及其语义。总之, 它们构成了逻辑 L4 API。
* **通用二进制接口**。大多数数据类型和一般数据对象的二进制表示是独立于特定处理器定义的 (尽管有两个不同的版本, 一个针对32位一个针对64位处理器)。两个版本一起形成 L4 的一般二进制接口。

逻辑接口加上通用二进制接口可以被看作是硬件的一个完整的规范独立的 L4 微内核接口。但是, 出于易用性和标准化的原因, 所提到的两个基本接口由另外两个接口类来补充:

* **通用编程接口**。通用编程接口定义逻辑接口的对象和通用二进制接口作为伪 c++ 类。与常规 c 语言绑定的语言大部分与 c++ 相同。对于 c 语言导致函数命名冲突的情况,函数名在方括号中给出。目前, 只指定 API 的 c 和 c++ 版本。其他语言接口的具体语法将被保留。稍后, 所有语言绑定都将包括在泛型编程接口.
* **便捷编程接口**。此接口不是严格意义上的 L4 微内核规范的一部分。它的所有数据类型和程序都可以使用通用编程接口实现。严格的说, 这是一个微内核之上的接口, 使最常见的操作对程序员来说更容易使用。重要的是要明白, 这个接口的目标是方便和易用性, 而不是完整性。它们提供的操作包括大约95% 的微内核功能。对于余下的 5%, 程序员必须使用基本 (不那么方便) 的通用编程接口来实现。显然, 便捷编程接口不是强制性的。因此, 从极简的角度来看, 不需要将它包含在通用的 L4 规范中。

然而, 由于标准化的原因, 因而软的可移植性洁具, 每一个完整的 L4 语言绑定必须包括整个便捷编程接口。

虽然便捷编程接口完全可以在通用编程接口之上实现, 即处理器独立的, 但是便捷编程接口的实现者也可以实现它的硬件依赖, 从而纳入任何优化, 成为可能针对特定的处理器而定制的二进制接口。

手册说明

* **基本数据类型**。基本数据类型是 word64，word32，word16，分别代表数据大小是 64、32、16位，并且与处理器无关；
* **特权线程**。某些系统调用只能由特权线程执行。任何与内核在启动时创建的初始线程属于同一地址空间的线程, 被视为特权线程。
* **位域**。位字段长度表示为下标（i/j），其中 i 是针对32位处理器，j 是针对64位处理器。如果仅仅只有下标 i 则表示32位和64位处理器有相同的位域。字节偏移是以 正负i/正负j 来分别表示32位处理器和64位处理器。如果所指定的字节，所有位字段都是只加32位，那么对于64位机剩下的32位，则是未定义或者是被忽略的。
* **符号“ ~ ”** 。输出参数或位字段可以是未定义的，对应的未定义参数或字段由 ~ 标记。它表示输出是没有被定义的值, 也就是说, 它们可能有任何值, 也甚至可能无法访问。任何依赖于未定义的参数或位字段的算法都视为不正确。
* **符号“ - ”** 。输入参数或位字段可以被指定为忽略, 由 - 表示。此类参数或字段可以保留任何值, 而不会影响被调用的服务。- 还可以定义可用于其他信息的位域。例如, fpage 外延包含一些被忽略的位, 它们用于某些系统调用中的访问控制位。
* **符号“ ≡ ”** 。在特定于处理器的接口中, 寄存器有时定义为不变。这时候就用 ≡ 表示。
* **向上兼容性**

1. 输出参数和位字段

当前被定义为 ~ 未定义的参数或者位字段可以被指定为已定义的。这些新定义的字段将只提供额外附加信息。如果系统调用与当前API中指定的方法是一样的，那么就可以忽略这些新定义的东西。

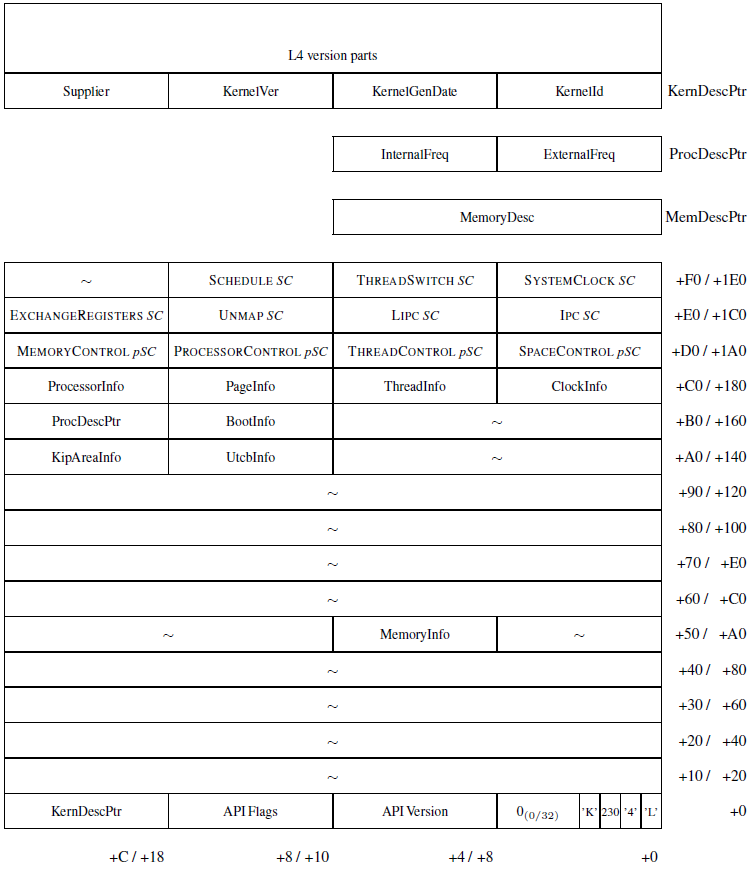
1. 输入参数与位字段

当前被定义为 - 忽略的字段可以指定为已定义的。但是这些字段的内容只与新定义的功能相关。如果这些API和指定用旧语义的系统调用一同使用，那么这些域将会被忽略。

# 第一章 基本内核接口

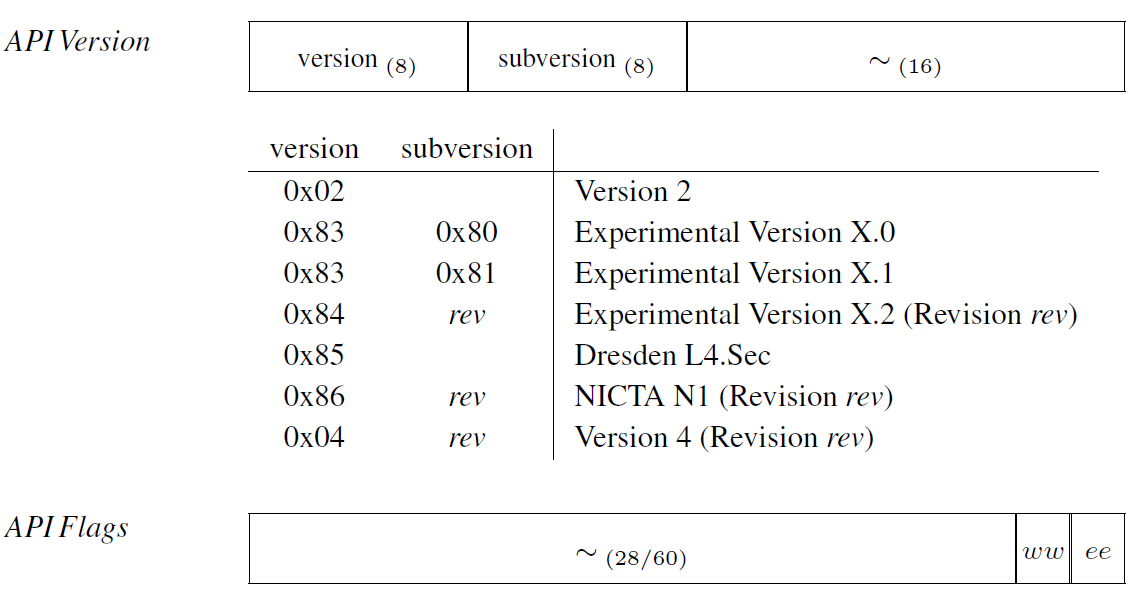
## 1.1 内核接口页 [数据结构]

内核界面页包含 API 和内核版本数据、系统描述符包括内存描述符和系统调用链接。页的其余部分未定义。

该页面是一个微内核对象。是的在地址空间创建时, 通过微内核直接映射到每个地址空间。它不是由pager映射的, 不能映射或授予另一个地址空间, 也不能被取消映射。新地址空间的创建者可以指定需要映射内核接口页的位置。此地址将在该地址空间的生存期内保持不变。任何线程都可以通过KERNELINTERFACE这个系统调用获取内核接口页的地址。

如上图所示，右边的偏移是以字节为单位，一个字节不论32位还是64位机都是8位长度。每一行看做四等分，则每一份都是一个字，只是32位机的字是32位，64位机的字是64位。

**API描述部分**

其中

ee = 00 : 小端模式

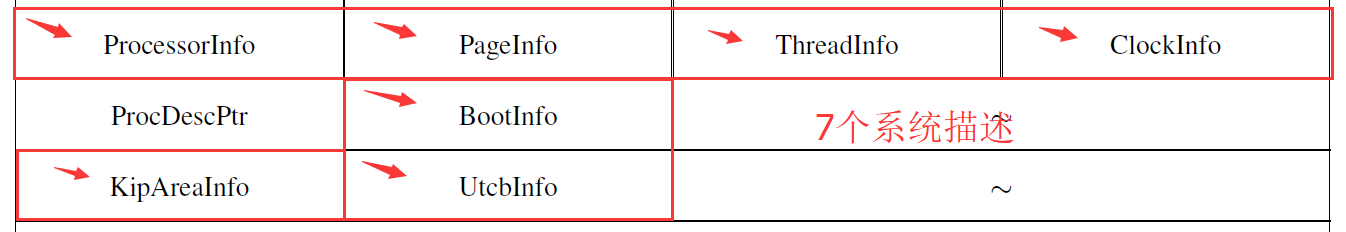
ee = 01 : 大端模式

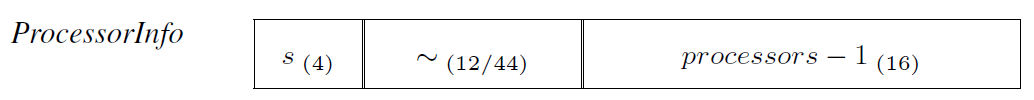
ww = 00 : 32位API

ww = 01 : 64位API

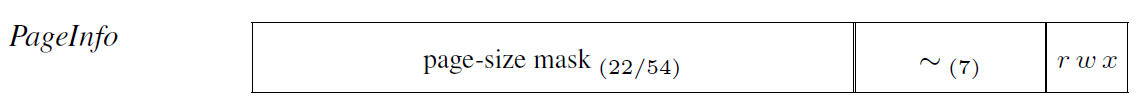
注意, 此字段不能直接用于区分大端模式和小端模式, 因为 ee 字段驻留在两个模式的不同字节中。此外, API 标志的偏移地址对于32位和64位模式是不同的。总之, 直接检查内核接口页不足以安全地区分32/64 位模式和小/大端模式。

通过KERNELINTERFACE系统调用来启用安全模式检测.安全模式检测可以在寄存器中传递 *APIFlags*。

**系统描述部分**

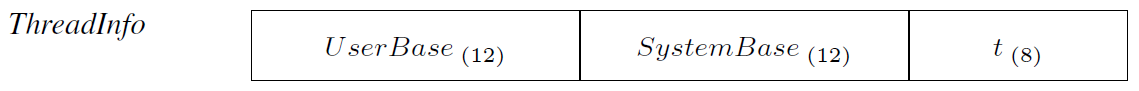
s 单个处理器描述占用的区域大小为2s。第一个处理器的描述字段的位置由 ProcDescPtr 表示。后续处理器的描述字段位于前一个之后。

processors 可用系统处理器的数量



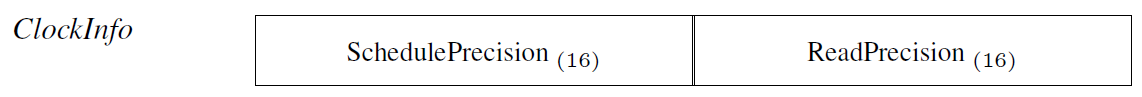
page-size mask 如果位 k-10 的page-size mask字段 (整个word的第 k 位) 设置为1，硬件和内核支持大小为2k的页。如果该位是0，硬件和/或内核不支持大小为2k的页。注意即使2k不是硬件支持的页面大小，大小为2k 的fpages也可以使用。有关支持的硬件页面大小的信息只是一个性能提示。

r w x 标识可以独立于其他访问权限设置的受支持的访问权限 (读取、写入、执行)。一个1位的信号可以在映射页上设置和重置该权限。处理器将隐式允许在任何映射页上读取和执行访问。



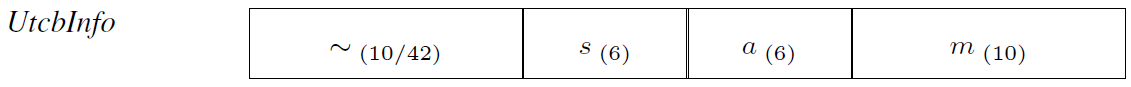
t 有效的线程号位数。"线程编号" 字段可能较大, 但仅位 0…t-1 对内核是有效的。较高的位必须都是0。

*UserBase* 用户线程可用的最低线程号 (请参见页14).前三线程号将分别用于初始线程σ0, σ1和root task(请参见页92)。这些初始线程的版本号将等于1。

*SystemBase* 用于系统线程的最低线程号 (请参见页14).此值下面的线程号表示硬件中断。

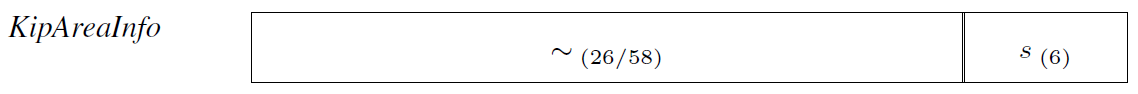
ReadPrecision 指定通过SYSTEMCLOCK系统调用读取系统时钟可以检测到的最小时间差不等于0。 基本上，这就是读取系统时钟时候的精度。

SchedulePrecision 指定基于唤醒时间的调度线程激活的最大抖动 (±):（假定没有更高或同等优先级的线程处于活动状态并启用计时器中断)。精度以时间段形式给出 (参见页30).



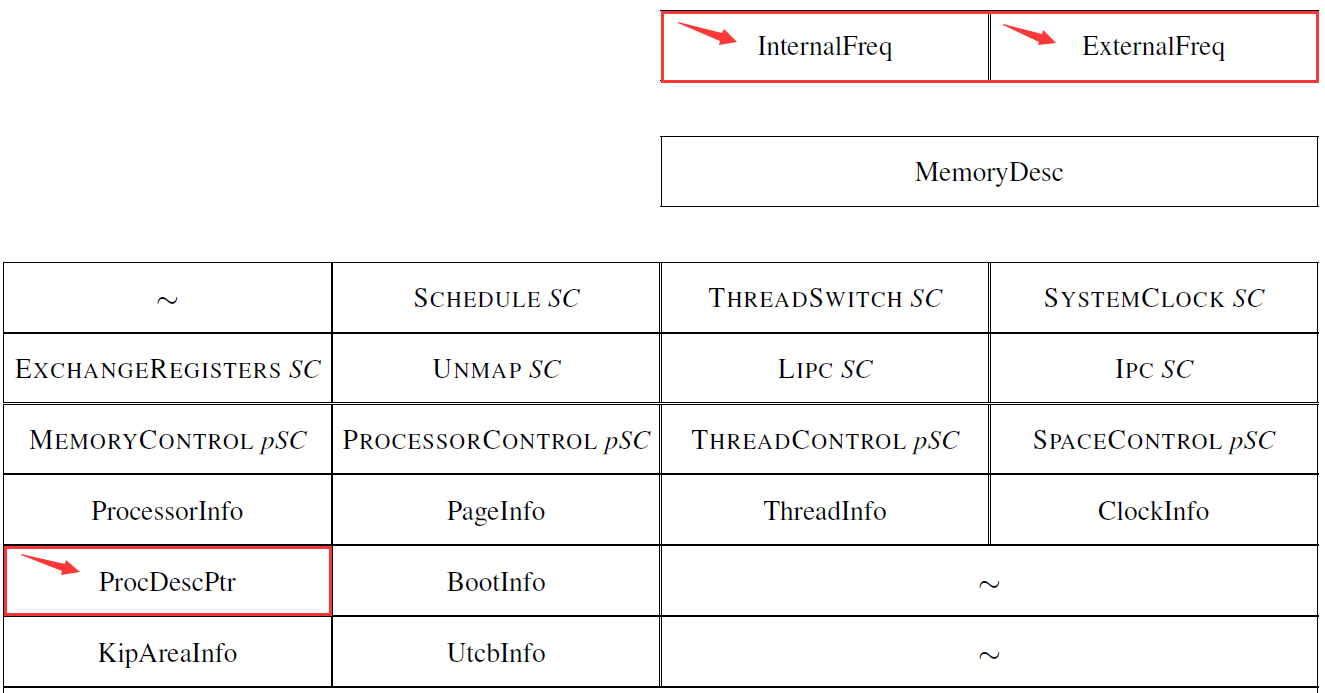
s 地址空间中的 UTCB 区域的最小大小是2s。UTCB 区域的大小限制了线程的总数量 k ,且满足式子2a mk ≤ 2s.

m UTCB大小的乘数。

a UTCB 位置必须与2a对齐。一个 UTCB 所需的总大小是2am 。

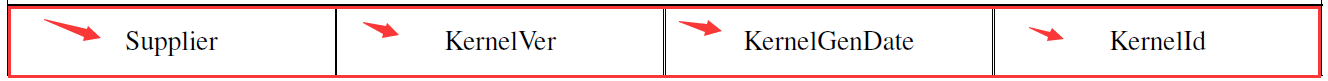
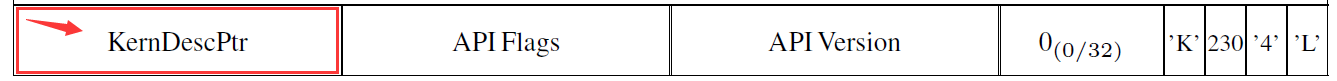
s 内核接口页的大小是2s

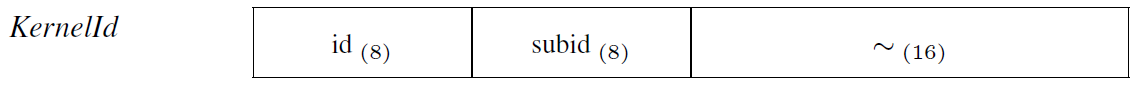
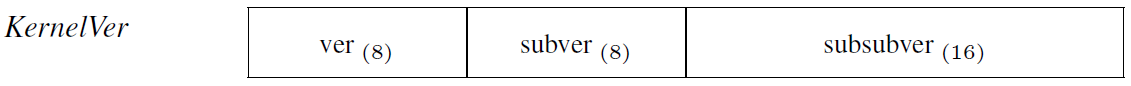
*Bootinfo* 在内核初始化之前, 引导加载程序可以将任意值写入内核配置页的 BootInfo 字段中 (请参见页92).初始化后的代码, 例如, 根服务器可以稍后从内核界面页读取该字段。它的值既不改变也不被内核解释。这是一个通过内核初始化传递系统信息的通用方法。

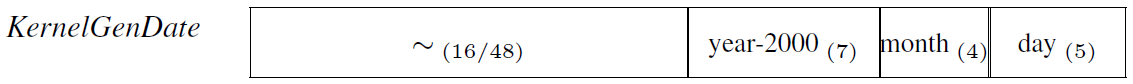
**处理器描述部分**

ProcDescPtr 指向一个数组, 其中包含每个系统处理器的说明。ProcessorInfo域包含数组的维度。ProcDescPtr作为相对于内核接口页的基址的地址提供。

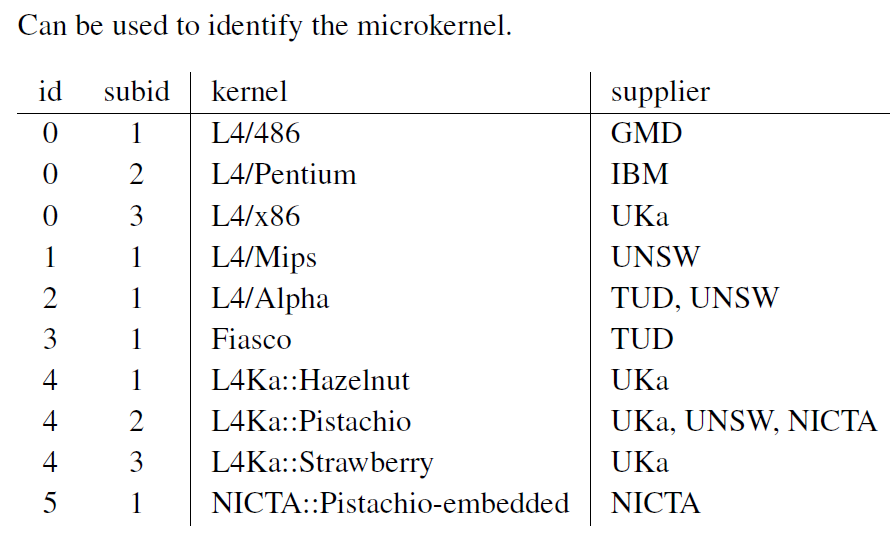
ExternalFreq/InternalFreq 外部总线/内部处理器频率，单位kHz。

**内核描述部分**

*KernDescPtr* 指向包含4个内核版本字的区域（后面有说明）, 该区域后跟一些0终止的明文字符串。第一个明文字符串标识当前内核, 后跟进一步可选的内核特定版本信息。其余的明文字符串标识依赖内核功能的结构 (见附录3).零长度字符串 (即字符串仅包含一个字符0) 将终止功能说明的列表。KernelDescPtr 给出的是相对于内核接口页基址的地址。

 用于识别微内核。

内核生成日期。

 可用于识别微内核的版本。注意, 此内核版本不一定与 API 版本相关。

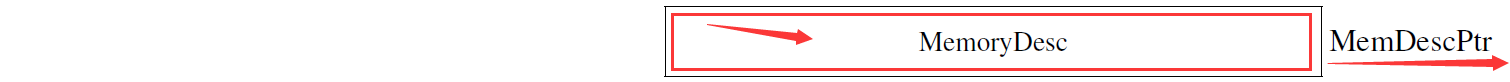
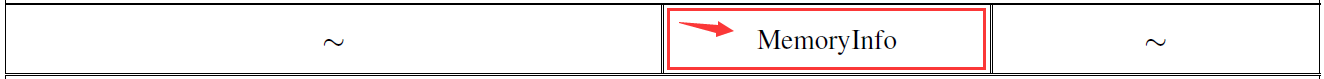
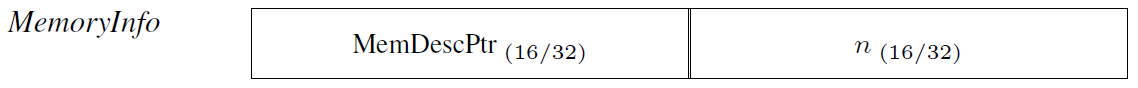
*Supplier* 以"供应商" 字段的四个最小有效字节指定标识内核供应商的字符串:

**系统调用链接**

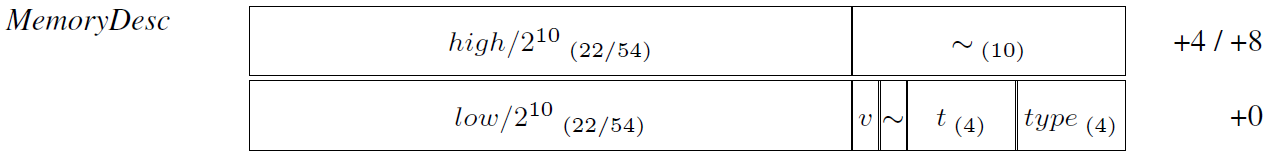
*SC* 一般系统调用。

*pSC* 特权系统调。例如, 只能由特权线程执行的系统调用

系统调用链接指定应用程序如何调用当前微内核的系统调用。系统调用链接的解释是 ABI 特定的, 但通常会根据内核接口页的基址来寻址,在那个地址内核提供了系统调用存根的位置。

**内存描述部分**

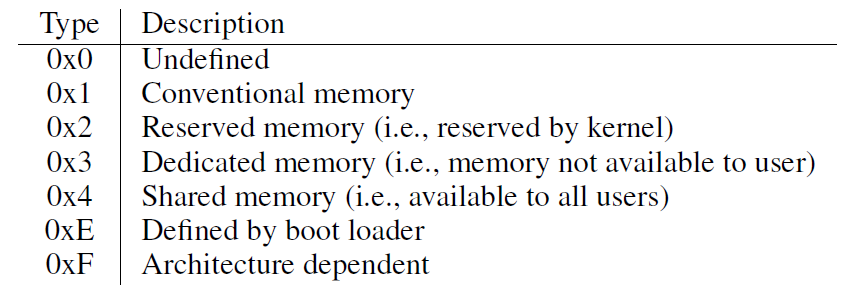
MemDescPtr 第一个内存描述符的位置(作为相对于内核接口页基址的偏移量)。后续内存描述符紧跟在第一个后面。对于指定重叠内存区域的存储描述符, 后面的描述符优先于较早的的。

n 内存描述符的数量。

high 内存区域中最后一个字节的地址。后十位地址位都是硬件置1。

low 内存区域中第一个字节的地址。后十位地址位都是硬件置0。

v 指示内存描述符是指物理内存 (v = 0) 还是虚拟内存 (v = 1)。

Type 标识内存描述符的类型：

*t,type=0xE*

内存描述符的类型依赖于引导程序。t 字段指定精确的语义。有关更多信息, 请参阅引导加载程序规范。

*t,type=0xF*

内存描述符的类型与体系结构相关。t 字段指定精确的语义。有关更多信息, 请参阅体系结构特定部件。

*t,type≠0xE, t,type≠0xE*

内存描述符的类型完全由type字段定义。t 字段的内容未定义。

## 1.2 KernelInterface [slow 系统调用]

该系统调用没有输入参数，返回的API version和API flags以及KernelId通过引用给形参赋值，函数的返回值是kernel interface page的基址。

函数形式**void\* KernelInterface（Word& ApiVersion，ApiFlags，KernelId）**

|  |  |
| --- | --- |
| 输入参数 | 输出 |
| 无 | void\* kernel interface page  Word ApiVersion  Word ApiFlags  Word KernelId |

各个形参的具体细节参看1.1节内容；便捷编程接口参看英文原版。

提供*kernel interface page, API version*和*API flags*的基址。后两个值是内核接口页中相应字段的副本。API 信息通过此系统调用使用寄存器传递(a) 以使在未来版本中的内核界面页面结构变化时候不受限制, (b) 能够确保内核的字节模式 (小/大) 和字宽 (32/64)的安全检测。

在第二页描述了kernel interface page。该页面是一个微内核对象。它通过微内核直接映射到地址空间创建时的每个地址空间。它不是由pager映射的, 不能映射或授予另一个地址空间, 也不能不要被取消映射。新地址空间的创建者可以指定必须映射内核接口页的地址。此地址将在该地址空间的生存期内保持不变。

任何线程都可以通过此系统调用确定内核接口页的地址。由于系统调用可能很慢, 因此极力建议将地址存储在静态变量中以供进一步使用。

对一个系统，还可以在所有地址空间中使用唯一一个内核接口页地址。然后, 在不使用当前系统调用的情况下, 可以通过固定绝对地址访问内核接口页。

除其他事项外, 该页还描述当前的 API、ABI 和微内核版本, 以便服务器或应用程序可以找出它是否以及如何在当前的微内核上运行。由于内核接口页还包含针对大多数其他系统调用的 API 和 ABI 特定的数据, 因此在使用其他系统调用之前通常需要该页的基址。

为了使版本检测能独立于 API 和 ABI,当前的系统调用在所有 L4 版本中都能正常工作。systemcall 的代码将永远不会更改, 并且在兼容处理器上将是相同的。(如果处理器向上兼容多个不兼容的处理器，内核应该为这个函数提供多个 systemcall 代码。)

针对KERNELINTERFACE系统调用衍生出来的其余内核接口，请查看英文原手册正文第 8 页。

## 1.3 虚拟寄存器 [虚拟寄存器]

虚拟寄存器由微内核实现。它们提供了一个快速接口, 用于在微内核和用户线程之间交换数据。虚拟寄存器是一个每个线程都拥有的静态对象。根据不同的特定的处理器类型, 它们可以映射到硬件寄存器或内存位置。也可以混合使用，即一部分虚拟寄存器映射到硬件寄存器，另一部分虚拟寄存器映射到内存上。 ABI 的虚拟寄存器访问取决于特定的处理器类型和虚拟寄存器类型, 更多硬件详细信息参看英文原版手册的附录。

一共有三种类型的虚拟寄存器：

* 线程控制寄存器（TCRs）
* 消息寄存器（MRs）
* 缓冲寄存器（BRs）

将非法值加载到虚拟寄存器、覆盖只读虚拟寄存器或访问在同一地址空间中的其他线程的虚拟寄存器是非法的(如果把虚拟寄存器映射到内存位置，这时候访问同一地址空间的中其他线程的虚拟寄存器是物理上可能的), 并且可能对当前地址空间的所有线程产生未定义的影响。然而, 因为虚拟寄存器不能跨地址空间访问, 它们从内核的角度来看是安全的：非法访问可以像任何其他编程 bug 一样只危害发端人的地址空间。

通常, 虚拟寄存器直接定位，而不能通过指针间接。

因此, 通用 API 不会提供任何的间接虚拟寄存器访问操作。然而，如果 ABI 允许, 处理器特定的代码生成器可能会使用间接访问技术。

针对虚拟寄存器的编程接口查看英文原手册正文第 11 页。

# 第二章 线程

## 2.1 ThreadId [数据类型]

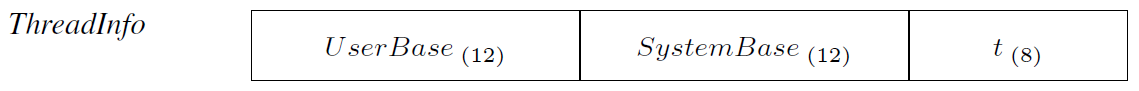
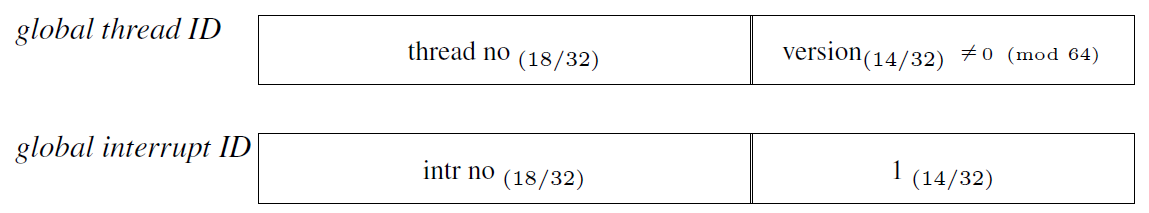
线程ID标识线程和硬件中断。线程 ID 可以是全局的或局部的。全局线程 ID在整个系统中是唯一的。它们独立于使用它们的地址空间来标识线程。每个地址空间都存在本地线程 ID；线程本地 ID 的范围仅作用于线程自己的地址空间。在不同地址空间, 相同的本地线程 ID 可以识别不同的和不相关的线程。

注意, 任何线程都有一个全局线程 ID。全局和本地线程 ID 都是以一个字节编码的。

**全局线程ID**

全局线程 ID 由一个字（32位机的字长是32位，64位机的字长是64位）组成,其中18 位 (32 位处理器) 或32位 (64 位处理器) 确定线程号，然后剩下的14位 (32 位处理器) 或32位 (64 位处理器) 可用于版本号。为了和局部ID区分开，表示版本号的字段的低6位中至少有一个必须是1。

用户线程编号可以在区间 [ UserBase，2t ) 内自由分配, 其中 t 表示线程 id 的上限。线程区间号 [ SystemBase，UserBase ) 是为L4内部线程保留的。硬件中断被视为硬件实现的线程。因此, 它们由线程 id 标识。它们对应的线程号在区间 [ 0, SystemBase ] 内。SystemBase、UserBase和 t 的值在内核接口页中。

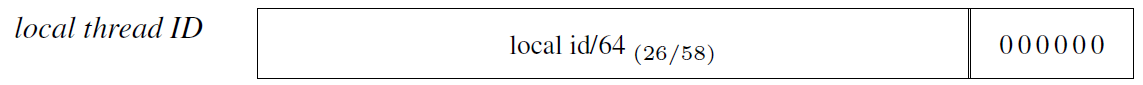


|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | SystemBase | | UserBase | | 2t |
| 硬件中断 | | 内部线程 | | 用户线程 | |

全局线程 ID 具有一个版本字段, 其内容可以由那些可以创建和删除线程的线程自由设置。但是, 版本的低6位不能全部是 0, 即每一个版本值v都必须满足 v mod 64 ≠ 0。对于硬件中断，版本字段始终是1。

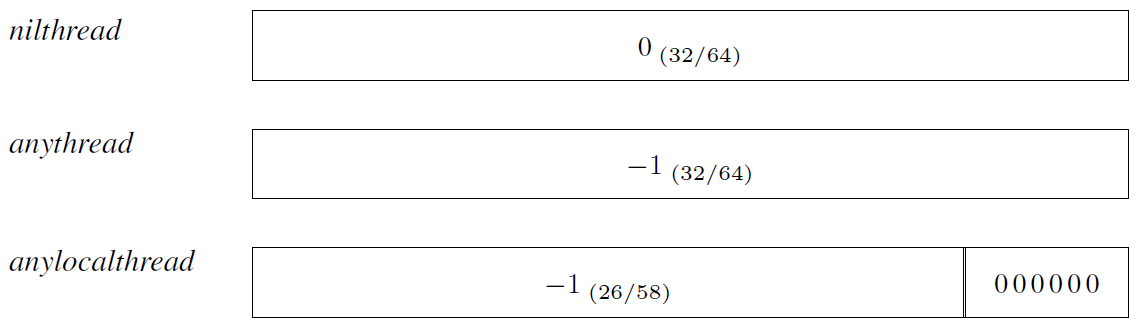
每当线程通过其全局线程 ID 访问时, 微内核就会检查版本字段。但是, 版本字段的语义不是由微内核定义的。操作系统的特性是自由使用这个字段用于任何用途。例如, 他们可以使用它来使线程 id 在时间上是唯一的。

**本地线程ID**

本地线程 ID 标识同一地址空间内的线程。通过设置线程ID最低6位都是0来标识本地线程。

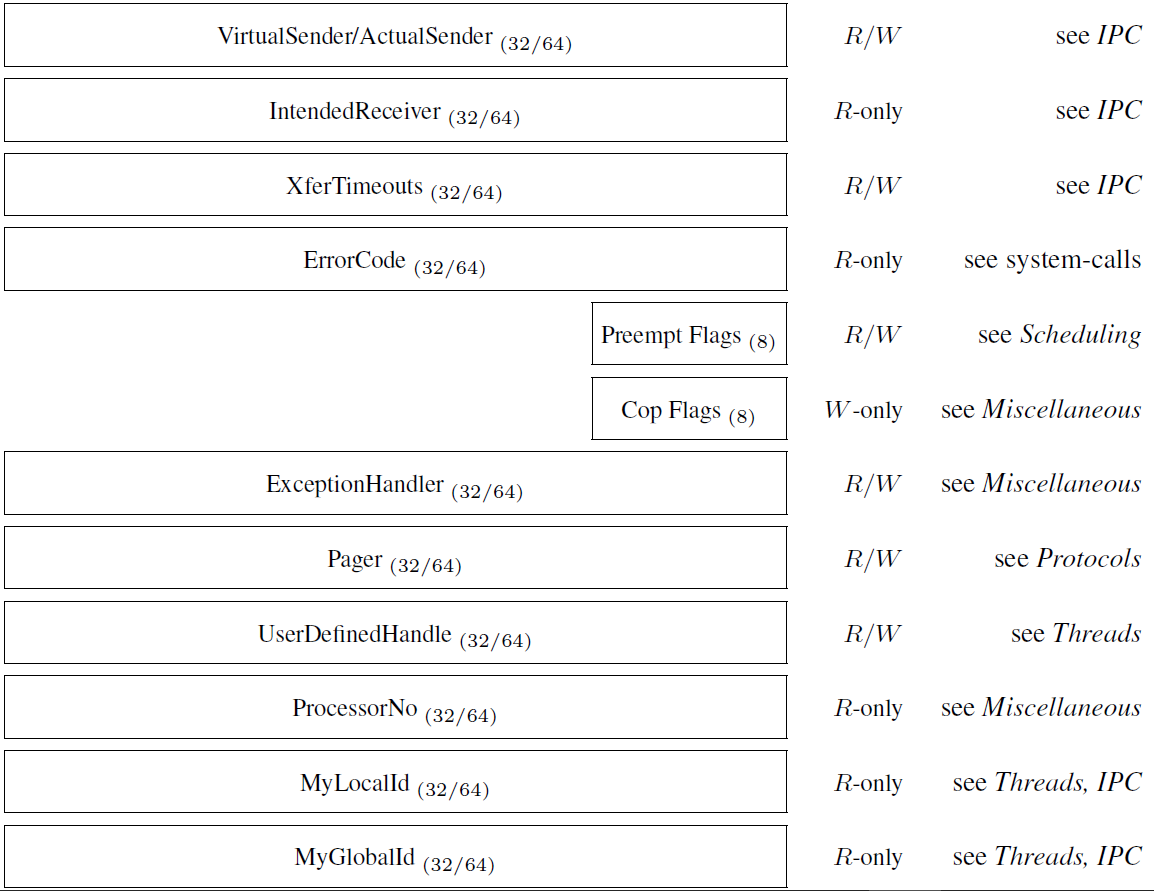
**特殊线程ID**

存在nilthread和两个通配符的特殊ID。线程ID：anythread，与任何给定线程ID匹配，包括所有的中断线程ID。线程ID：anylocalthread，匹配驻留在相同地址空间的所有线程。

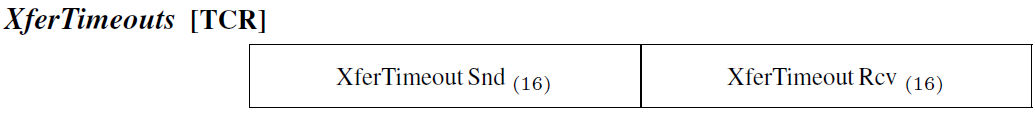
他们的格式如下：

针对线程ID的编程接口查看英文原手册正文第 15 页。

## 2.2 线程控制寄存器（TCRs） [虚拟寄存器]

TCRs是在用户线程和微内核之间交换相对静态控制信息的快速机制。TCRs是每个线程的静态非暂态寄存器。

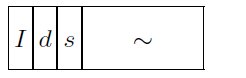
**VirtualSender** 当IPC阶段要发送的消息的消息头中的低16位控制位中的P字段等于1时候，表示该IPC是传输发送操作，而VirtualSender指定了发起者线程的ID。

**IntendedReceiver** 当一个IPC被标识为重定向IPC时候，该IPC被重定向到当前线程，并且IntendedReceiver这个字段用于指定应当接受消息的线程的ID。具体细节见IPC的msgtag部分。

在一个IPC中，一旦消息传输已经开始，传输消息的时间粗略地被发送者和接收者的XferTimeout的最小值所限制。 “粗略”意味着传输超时仅在发送者或接收者地址空间中的消息复制引起页面错误时才被检查。

每个线程有两个传输超时。一个用于发送阶段，一个用于接收阶段。具体细节见IPC部分。

**ErrorCode**



**Preempt Flags**

抢占标志，用于控制异步抢占。I标志位表示是否有异步抢占正在等待；s标志位控制异步抢占是否通知异常处理程序；d标志位控制抢占的发生方式，是所有抢占都立即响应还是基于优先级响应。具体细节见调度部分。

**Cop Flags** c0 ········ · · c7

协处理器标志，用于帮助内核优化某些硬件体系结构的线程切换。通过将ci位重置为0，一个线程告诉系统它不再需要协处理器i。 如果内核发现ci = 0，则得出结论：协处理器的寄存器和状态不必保存。 但是，内核确保协处理器不能用作不同地址空间之间的隐蔽通道。一旦一个线程复位了位ci，就必须将ci设置为1，然后才能在协处理器i上执行下一个操作。 否则，在使用协处理器寄存器和状态时可能会被任意修改。

请注意，ci位是只写的。 读取它们会导致一个未定义的值。 线程创建后，所有的ci位都被设置为1。有关Cop Flags的接口见6.2节。

**ExceptionHandler** 如果该字段不等于nilthread，指定异常处理程序线程。 当一个线程产生一个异常时，内核向线程的异常处理线程发送一个异常IPC消息，并等待来自包含指令指针的异常处理程序的响应，线程应该继续在MR1中执行；如果该字段等于nilthread，则线程没有指定异常处理程序，如果发生异常，则线程暂停，不再调度。

**Pager** 调页程序。也是一个进程，与其相关联的线程通信也是使用的MR寄存器。

**MyGlobalId** 线程的全局ID。

**MyLocalId** 线程的本地ID。

**ProcessorNo** 运行线程的处理器的编号。

**UserDefineHandle** 这个区域可以被用户线程随意设置和读取，比如用于存储线程号，存储附加线程控制块指针等等。

针对TCRs的编程接口查看英文原手册正文第 17 页。

## 2.3 ExchangeRegisters [系统调用]

函数形式 ThreadId ExchangeRegisters (ThreadId dest, Word control, sp, ip, flags, UserDefinedHandle, ThreadId pager, Word& old control, old sp, old ip, old flags, old UserDefinedHandle, ThreadId& old pager)

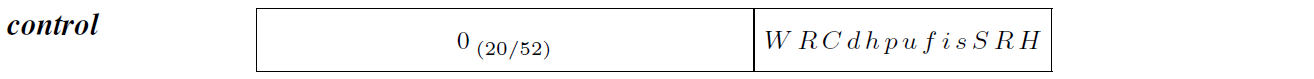
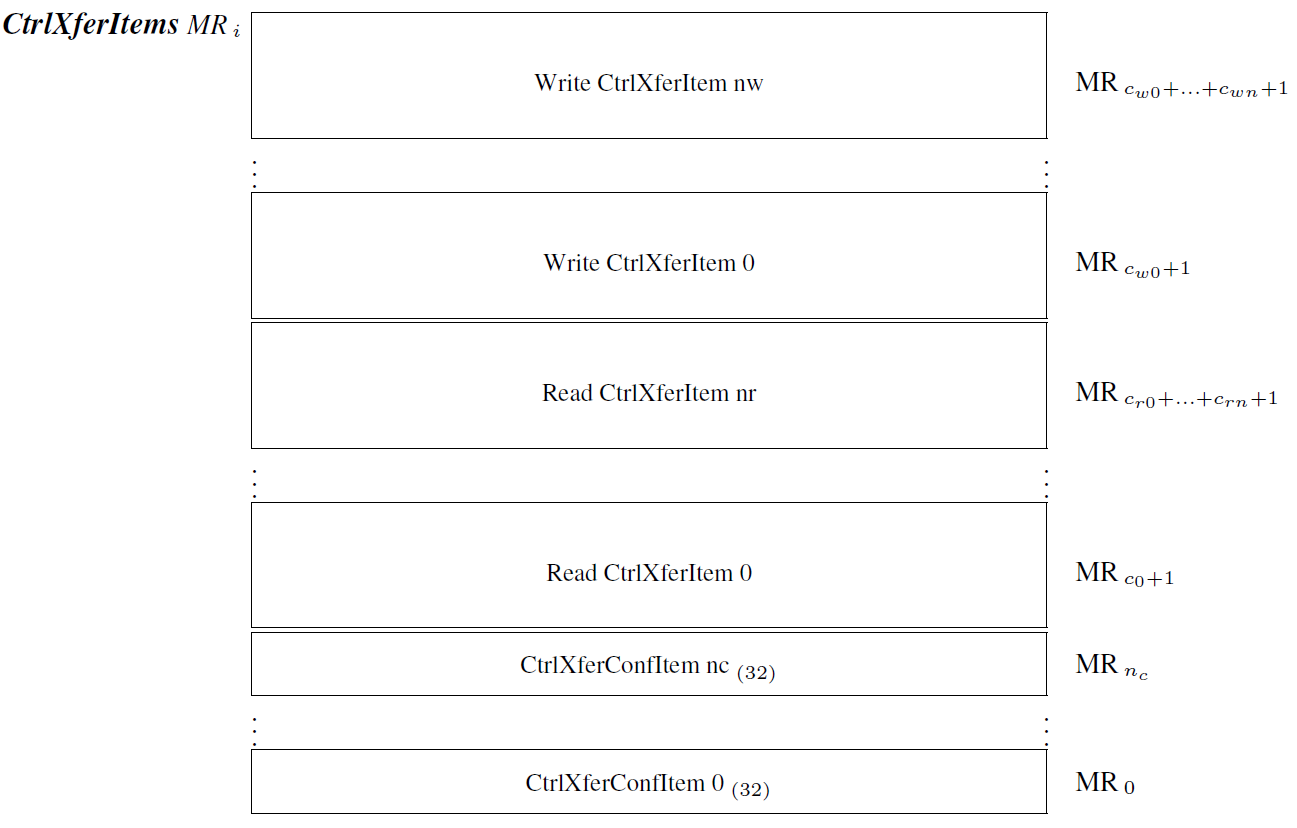
|  |  |
| --- | --- |
| 输入参数 | 输出 |
| ThreadId dest  Word control  Word SP  Word IP  Word FLAGS  ThreadId pager  Word UserDefinedHandle | ThreadId result  Word control  Word SP  Word IP  Word FLAGS  ThreadId pager  Word UserDefinedHandle |

该系统调用交换或读取线程的FLAGS，SP和IP硬件寄存器，以及调页程序和UserDefinedHandle TCR。而且，线程执行可以暂停或恢复。 目标线程必须是驻留在调用者地址空间中的活动线程。

任何IP，SP或FLAGS修改都会更改所寻址线程的相应用户级寄存器。一般来说，正在进行的内核活动不受影响。 但是，当前活动的IPC操作可以被取消或中止。有关详细信息，请参阅下面的SR位指定。

无论目标线程是以用户模式还是以内核模式执行，TCR中的调页程序（pager）字段和UserDefinedHandle字段的修改都会立即生效。

**输入参数**：

**dest** 被寻址线程的线程标识。这可能是本地或全局ID。但是，所指的线程必须驻留在当前地址空间中。在某些实现中使用本地线程ID可能会快得多。

W,R,C 就C标志位而言，针对内核生成消息，C标志位配置内核是否使用扩展控制传输协议。R和W标志允许使用控制传输项目直接读取和写入线程状态。所有控制转移项目（CtrlXferConfItem n）都被传递到调用线程的消息寄存器中。

 C = 1 调用者要求扩展状态在内核生成消息上发送。每一次错误发生，扩展状态位都要被指定，就像是控制传输配置项目CtrlXferConfItem一样。这些项目是从MR0开始。

fault 内核消息生成的错误的体系结构特定标识符。

idmask 指定发送消息时要包含的控制转移项目的位掩码。标识符是结构特定的。如果掩码中的位n被设置为1，则内核将把编号为n的CtrlXferItem附加到消息上。请注意，内核不保证为一个特定的CtrlXferItem选择单个寄存器不被允许; 相反，总是包含特定项目的全部内容。

C 每一个消息寄存器MRi中的最后一位C，是连续标志，除了最后一个CtrlXferItem外都被设置。

R = 1 调用者请求扩展状态从目标线程读取。 扩展状态作为一组控制传输项目在调用者的消息寄存器中传递。 如果C = 1，则要读取的第一个CtrlXferItem紧跟在最后一个CtrlXferConfItem之后。 如果C = 0，则读取项目从MR0开始。连续标志c被置位，除最后一个CtrlXferConfItem之外。注意，每次读取CtrlXferConfItem的时候，都必须提供足够的空间足以涵盖要读取的项目中的所有寄存器（即，如果项目的掩码指定要读取的寄存器有k个，则该项目的大小必须包含至少k个寄存器的字）。

W = 1 调用者请求扩展状态从目标线程写入。 扩展状态作为一组控制传输项目在调用者的消息寄存器中传递。 如果R = 1，则要写的第一个CtrlXferItem紧跟在最后一个读项目之后。否则如果C = 1，则第一个写项目紧跟在最后一个CtrlXferConfItem之后。否则如果C = 0，从MR0开始写。连续标志c被置位，除最后一个CtrlXferConfItem之外。

d 如果d = 0，返回结果是未定义；

如果d = 1，返回结果会提供调用该系统调用之前的各个参数所对应的值。

注意：不论d = 0还是1，与系统调用是否正确执行没有关系，d只决定是否返回旧值。

h p u f i s s表示SP寄存器；i表示IP寄存器；f表示FLAGS寄存器；u表示线程控制寄存器（TCR）的UserDefinedHandle字段；p表示TCR的调页函数（pager）字段；h指的是后文提到的H标志位。如果一个标志位被置1，那么他所对应的寄存器或者状态位就会被输入参数重写。否则的话忽略输入参数，并且所对应的寄存器或者是状态位不被修改。

SR 控制是否通过系统调用取消/中止所寻址的线程正在进行的IPC操作。

S = 0 所寻址的线程的，正在等待发送消息的IPC操作或者是正在发送消息的IPC操作会跟之前一样继续进行。SP、IP或标志修改将延迟到IPC操作终止。

S = 1 所寻址线程的正在等待发送消息的IPC操作会被取消；当前正在发送消息的IPC操作会被中止。

R = 0 所寻址的线程的，正在等待接收消息的IPC操作或者是正在接收消息的IPC操作会跟之前一样继续进行。SP、IP或标志修改将延迟到IPC操作终止。

R = 1 所寻址线程的正在等待接收消息的IPC操作会被取消；当前正在接收消息的IPC操作会被中止。

H 如果H = 1，则暂停/恢复线程；如果 H = 0，则忽略。

H = 0 如果线程没有停止，则没有影响。否则，线程执行将恢复。

H = 1 用户级线程执行被暂停。注意，正在进行的IPC和其他内核操作不受H的影响。

**SP** 如果输入参数control中的s标志位是1，那么当前用户级堆栈指针会被设置成该输入参数SP。如果s标志位是0，则忽略该参数。

**IP** 如果输入参数control中的i标志位是1，那么当前用户级指令寄存器指针会被设置成该输入参数IP。如果i标志位是0，则忽略该参数。

**FLAGS** 如果输入参数control中的f标志位是1，那么根据该参数来设置线程的用户级处理器标志。如果i标志位是0，则忽略该参数。FLAGS的语义取决于处理器类型。

**UserDefinedHandle** 如果输入参数control中的u标志位是1，那么根据该参数来设置线程TCR中的UserDefinedHandle字段。如果u标志位是0，则忽略该参数。

**pager** 如果输入参数control中的p标志位是1，那么根据该参数来设置线程TCR中的pager字段。如果p标志位是0，则忽略该参数。

**输出参数**：

**result** ≠ nilthread，并且输入参数dest是一个局部线程ID

被寻址的是全局线程ID，ExchangeRegisters系统调用成功。

**result** ≠ nilthread，并且输入参数dest是一个全局线程ID

被寻址的是局部线程ID，ExchangeRegisters系统调用成功。

**result** = nilthread

操作失败。TCR中的ErrorCode字段会表明出错原因。

**ErrorCode**

如果result = nilthread，则会被设置；否则如果result ≠ nilthread，则该字段未定义；

= 2，无效线程。dest参数指定了一个无效的线程ID、未活动的线程ID或者是指定了一个不在同一地址空间的线程。

该参数只有在输入参数的control参数d字段等于1的时候才是有效的，其他情况下都是未定义。

H 报告当ExchangeRegisters系统调用被调用的时候所寻址的线程是否是停止了。注意，这个输出参数control的H位是独立于输入参数control。

SR 报告当ExchangeRegisters系统调用被调用的时候所寻址的线程是否正在进行一个IPC操作。该字段的值为0表示所寻址线程没有在发送阶段或者是接收阶段。同样的，这个输出参数control的SR位是独立于输入参数control。

R = 1 当寻址线程位于IPC操作的接收阶段时，执行操作。如果输入参数control的R = 1，IPC操作被取消或中止。

S = 1 当寻址线程位于IPC操作的发送阶段时，执行操作。如果输入参数control的S = 1，IPC操作被取消或中止。

SP 如果输入参数control中的d字段等于1，则该参数的值是原来线程未调用ExchangeRegisters系统调用之前的SP寄存器的值，如果d等于0，输出参数SP的值未定义。

IP 如果输入参数control中的d字段等于1，则该参数的值是原来线程未调用ExchangeRegisters系统调用之前的IP寄存器的值，如果d等于0，输出参数IP的值未定义。

FLAGS 如果输入参数control中的d字段等于1，则该参数的值是原来线程未调用ExchangeRegisters系统调用之前的FLAGS寄存器的值，如果d等于0，输出参数FLAGS的值未定义。FLAGS的语义是处理器特定的。

UserDefinedHandle

如果输入参数control中的d字段等于1，则该参数的值是原来线程未调用ExchangeRegisters系统调用之前寻址线程TCR中UserDefinedHandle字段的值，如果d等于0，该参数的值未定义。

pager 如果输入参数control中的d字段等于1，则该参数的值是原来线程未调用ExchangeRegisters系统调用之前寻址线程TCR中pager字段的值，如果d等于0，该参数的值未定义。

针对ExchangeRegisters的编程接口查看英文原手册正文第 21 页。

## 2.4 ThreadControl [特权系统调用]

函数形式：Word ThreadControl (ThreadId dest, SpaceSpecifier, Scheduler, Pager, void\* UtcbLocation)

|  |  |
| --- | --- |
| 输入参数 | 输出 |
| ThreadId dest  ThreadId SpaceSpecifier  ThreadId scheduler  ThreadId pager  void\* UtcbLocation | Word result |

特权线程（例如根服务）可以通过此功能删除和创建线程。它还可以修改现有线程的全局线程ID（仅限版本字段）。

线程可以创建为活动或非活动的线程。非活动线程不运行，但可以由在相同地址空间中运行的活动线程激活。

主动创建的线程立即启动，方法是通过执行其调页程序的短接收操作。（一个活动线程必须有一个调页程序。）激活的线程期望从其调页程序获得启动消息（MsgTag和两个无类型的字）。一旦接收到启动消息，它将MR 1的值作为其新的IP，MR 2的值作为其新的SP，然后用这接收到的IP和SP在用户级开始执行线程。新的线程将在与调用了ThreadControl系统调用的处理器上运行。

中断线程被视为正常的线程。它们在系统启动时处于活动状态，不能删除或迁移到不同的地址空间（即，SpaceSpecifier必须等于中断线程ID）。当发生中断时，中断线程发送一个IPC到其调页程序，并等待空的中断结束确认消息（MR0 = 0）。中断线程永远不会引起页面错误。为了停用中断消息传送，调页程序被设置为中断线程自己的ID。

**输入参数**：

**dest** 寻址线程。必须是全局线程id。只有线程号才能有效地用于寻址线程。如果存在具有与dest指定线程号相同的线程，则其版本位将被dest id的版本位覆盖，并且任何正在进行的IPC操作都将被中止。否则，指定的版本位用于线程创建，即线程创建会生成一个id为dest的线程。

**SpaceSpecifier**≠nilthread，dest不存在

创建新的线程。SpaceSpecifier指定线程将驻留在哪个地址空间中。由于地址空间没有自己的ID，因此使用线程ID作为SpaceSpecifier。 其含义是：新线程应该在与线程SpaceSpecifier相同的地址空间中执行。

新地址空间中的第一个线程是使用SpaceSpecifier = dest创建的。 这个操作隐含地创建一个新的空的地址空间。注意，新地址空间是使用空的UTCB和KIP区域创建的。因此，在执行线程之前，必须通过SpaceControl操作来完成空间创建。

**SpaceSpecifier**≠nilthread，dest存在

只修改。被寻址的线程dest既不被删除也不被创建。可以修改寻址线程的版本字段，相关调度器，调页程序或相关地址空间。如，将线程迁移到新的地址空间。

**SpaceSpecifier** = nilthread，dest存在

删除。 被寻址的线程dest被删除。删除地址空间的最后一个线程也会隐式删除地址空间。

**scheduler**≠nilthread

定义允许调度被寻址线程的调度程序线程。注意，在被寻址线程开始执行时，调度程序线程必须存在。

**Scheduler** = nilthread

当前调度程序关联不被修改。ThreadControl在执行创建操作时，这种变体是非法的。

**pager**≠nilthread

dest所指线程的pager被设置为pager参数指定的线程。如果dest之前是非活动的，那么就激活它。

**Pager** = nilthread

当前的pager关联不被修改。

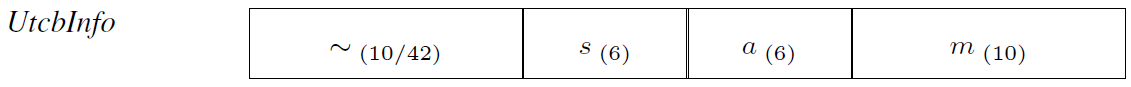
如果执行的是ThreadControl的创建线程操作，dest会被创建成一个非活动的线程。

**UtcbLocation**≠-1

线程UTCB的起始地址被设置为UtcbLocation。线程激活时，UTCB必须完全适合配置的地址空间的UTCB区域，并且必须根据内核接口页面的UtcbInfo字段正确对齐。确保多个线程的UTCB不重叠是应用程序的责任。更改已经激活的线程的UtcbLocation是非法的操作。注意，由于新创建的空间具有空的UTCB区域，因此无法在未使用SpaceControl正确配置的地址空间中激活线程。

**UtcbLocation** = -1

UTCB的位置不变。



s 地址空间中的 UTCB 区域的最小大小是2s。UTCB 区域的大小限制了线程的总数量 k ,且满足式子2a mk ≤ 2s.

m UTCB大小的乘数。

a UTCB 位置必须与2a对齐。一个 UTCB 所需的总大小是2am

**输出参数**：

**result** 如果操作成功，则结果为1，否则结果为0，并且TCR中的ErrorCode 指示失败原因。

**ErrorCode**

如果result = 1则是未定义，result = 0的时候才被设置。

= 1 没有特权。 当前线程没有执行操作的权限。

= 2 无法使用的线程。 dest参数指定了内核线程或不可用的中断线程。

= 3 无效的空间。 SpaceSpecifier参数指定了一个无效的线程ID，或者激活尚未初始化的空间中的线程。

= 4 无效的调度程序。 调度程序参数指定了无效的线程ID，或者设置为nilthrad来创建THREADCONTROL操作。

= 6 无效的UTCB位置。 UtcbLocation位于UTCB区域之外，或者尝试更改已经激活的线程的UtcbLocation。

= 8 内存不足。 内核无法分配执行操作所需的资源。

针对ThreadControl的编程接口查看英文原手册正文第 25 页。

# 第三章 调度

## 3.1 Clock [数据类型]

在32位和64位处理器上，系统时钟都表示为64位无符号计数器。 时钟以1 s为单位测量时间，与处理器频率无关。 虽然时钟基准时间未定义，但保证计数器不会溢出至少1000年。

与Clock数据类型有关的编程接口查看英文原手册正文第 28 页。

## 3.2 SystemClock [系统调用]

函数形式：Clock SystemClock ()

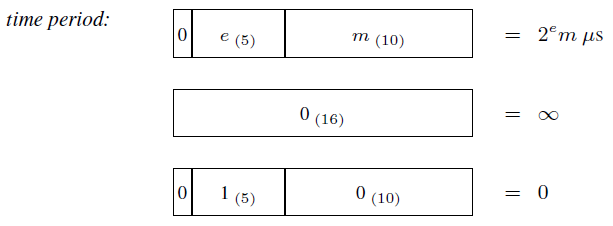
|  |  |
| --- | --- |
| 输入参数 | 输出 |
| 无 | Clock clock |

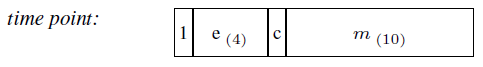
该系统调用提供当前的系统时钟，但是这个操作不会陷入到内核态。

针对SystemClock的编程接口查看英文原手册正文第 29 页。

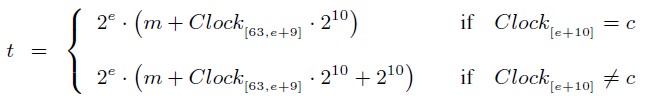
## 3.3 Time [数据类型]

时间值用于指定IPC操作的发送/接收超时（请参见第61页）和调度的时间量程（参见第33页）。时间单位是秒。时钟滴答因此每隔一秒发生。

相对时间值指定一个时间段。 时间段被编码为未标准化的16位浮点数。 （注意，为了更容易处理，尾数可以有前位0）。可以指定的最短非零时间段是1秒，最长有限期稍微超过610小时。 经常用于超时的两个特殊时期是0和∞，这是一个不会停止的时间段。值0和∞有特殊的编码。

绝对时间值指定一个时间点。它们只在有限的时间内有效，最多67秒。

对于时间点值的语义描述，我们使用Clock来表示当前的时钟值，单位是微秒，x[i]来表示x的位i，x[i,j]表示由x的位i到j构成的数。 然后，时间点值（c,m,e）指定该点:

因此绝对时间值在未来越近越精确。( Absolute time values are thus the more precise the nearer in the future they are.)

在时钟到达指定的时间点后，绝对时间值的精确度最高。有效区间可以扩大，但只能通过降低精度这种方式。通常，当前时钟值为C0时构成的时间点值（c，m，e）从C0到以下时间点是有效的：

因此，10毫秒的时间点值应该具有10微秒的精度以保持他的有效，而应该在整秒内保持有效的值只能具有1毫秒的精度。通常，可以达到所要求的有效区间精度为0.1％。

与Time数据类型有关的编程接口查看英文原手册正文第 30 页。

## 3.4 ThreadSwitch [系统调用]

函数形式：Void ThreadSwitch (ThreadId dest)

|  |  |
| --- | --- |
| 输入参数 | 输出 |
| ThreadId dest | void |

调用该系统调用的线程释放处理器（非预占），以便另一个就绪线程获取CPU。

**输入参数**：

**dest** = nilthread 处理切换到由调度程序选择的未定义就绪线程。（这可能是调用该系统调用的线程）。由于这是“普通”的调度，线程得到一个新的时间片。

**dest**≠nilthread 如果dest准备就绪，则处理器切换到此线程。在这个“特别”的调度中，调用线程将其剩余的时间片捐献给目标线程。（如果有的话，除了通常所计划的时间外，这个线程还可以获得调用线程的剩余时间）

如果目标线程没有准备好或驻留在不同的处理器上，则系统调用将按照dest = nilthread所述进行操作。

针对ThreadSwitch的编程接口查看英文原手册正文第 32 页。

## 3.5 Schedule [系统调用]

函数形式：void Schedule (ThreadId dest, Word TimeControl, ProcessorControl, prio, PreemptionControl, Word& old TimeControl)

|  |  |
| --- | --- |
| 输入参数 | 输出 |
| ThreadId dest  Word time control  Word processor control  Word prio  Word preemption control | Word result  Word time control |

调度程序可以使用系统调用来定义线程的优先级，时间片长度和其他调度参数。此外，它还提供线程状态。

系统调用仅在调用线程与目标线程的调度程序位于相同的地址空间时才有效（请参见ThreadControl()）。

**输入参数**：

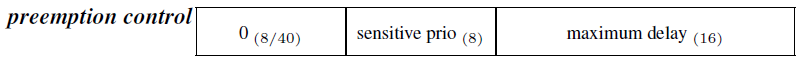
**dest** 目标线程ID。 目标线程必须存在（但可以不活动），并且当前线程必须与目标线程的调度程序（参见线程控制）位于相同的地址空间中。 否则，目标线程不受影响。

如果提供的值为-1，则所有其他输入参数都不起作用，以确保相应的内部线程变量不被修改。 以下描述总是指值不等于-1。

ts len 目标线程的新时间片长度。时间片长度被指定为一个时间段。 绝对时间值和0是非法的。然而，可以指定时间片长度为∞。在这种情况下，由于时间片耗尽，线程不会经历抢占。指定的值总是四舍五入到最接近的可能的时间片长度。特别是，1微秒的时间段将导致尽可能短的时间片。

写入时间片长度的操作会引起对当前时间片的初始化，初始化的时间片长度就是写入的新长度。在时间片长度耗尽之后，线程被抢占，而在下一个时间片到来时，时间片长度被重新加载为ts len。

total quantum 定义线程的总时间长度。时间总长度的耗尽将导致RPC到线程的调度器（即当前线程）。（重）写总时间长度会重新初始化总时间长度，但是不影响已经消耗了的时间量。总时间长度被指定为一个时间段。绝对时间值是非法的。总时间长度可以被指定为∞。

 目标线程的新优先级。 必须小于或等于当前线程的优先级。

sensitive prio 低优先级或者是相同优先级的线程发生抢占，（a）如果延迟抢占标志（后文的s标志）被设置，则被延迟直到线程执行线程切换（nilthread）系统调用; （b）如果信号抢占标志被置位，则引发异常处理程序的抢占错误，即抢占会产生一个抢占错误信号，发送给异常处理程序处理这次抢占。

如果是一个高优先级的线程发生抢占，则不管延迟抢占和信号抢占标志的状态如何，都不会有抢占延迟或抢占信号异常处理发生。

maximum delay 目标线程等待抢占的最大等待时间，单位是微秒。该值设置为0就可以禁用抢占延迟。

processor number

指定线程应该迁移到的处理器号。处理器号必须是有效的，即小于处理器的总数（参见第3页的内核接口页）。 否则，该参数将被忽略。第一个处理器号码表示为0。

**输出参数**：



tstate = thread state：

1. 出错。操作完全失败。TCR中的ErrorCode字段说明了出错的原因。
2. 死亡。该线程无法执行或者不存在。
3. 非活动。线程处于非活动状态或者是停止状态。
4. 运行。线程已经准备好运行在用户态下。
5. 等待发送。 当前用户调用的IPC发送操作等待目的地（接收者）准备好接收。
6. 正在发送。用户调用的IPC发送操作当前正在传送一个传出消息。
7. 等待接收。用户调用的IPC接收操作正在等待一个传入消息。
8. 正在接收。用户调用的IPC接收操作正在接收一个传入消息。

**ErrorCode**

= 1 没有特权。 当前线程没有执行操作的权限。

= 2 dest参数指定了一个无效的线程ID。

= 5 无效的参数。指定的时间片长度、时间总长度、优先级或者是处理器编号是无效的。

rem ts 当前时间片的剩余部分。

rem total 线程的剩余时间总量。

针对Schedule的编程接口查看英文原手册正文第 35 页。

## 3.6 抢占标志 [线程控制寄存器的一个字段]

**Preempt Flags**

抢占标志，用于控制异步抢占。I标志位表示是否有异步抢占正在等待；s标志位控制异步抢占是否通知异常处理程序；d标志位控制抢占的发生方式，是所有抢占都立即响应还是基于优先级响应。其中I标志由微内核设置，通常由用户读取/重置。

**s** = 0 异步抢占不会通知异常处理程序。

**s** = 1 异步抢占被作为抢占故障发送给异常处理程序。如果d = 0，则异步抢占立即发生。否则，如果抢占线程的优先级低于被抢占线程的优先级，异步抢占将被延迟，直到运行系统的调度函数。

**d** = 0 所有异步抢占立即发生。如果抢占被指定为抢占故障（即s = 1），则抢占发生之后才发送抢占故障到异常处理程序。

**d** = 1 如果抢占者的优先级低于或等于当前线程的敏感优先级，则异步抢占被延迟。 （敏感优先级由调度程序设置，请参见第27页。）延迟抢占不会立即中断当前线程，而是被延迟，直到在当前线程调用系统调用ThreadSwitch(nilthread)。但是，抢占的延迟时间不能超过线程调度程序设置的最大延迟时间。这种抢占延迟溢出会重置d位，并通知异常处理程序。

**I** = 0 当前没有异步抢占在等待。

**I** = 1 有一个异步抢占当前正在等待，例如线程应尽快重置d标志并调用线程切换。调用线程切换重新启用下一个延迟的异步抢占的最大延迟。

如果在用户线程重置d标志后没有异步抢占挂起（I = 0），则不需要调用线程切换。

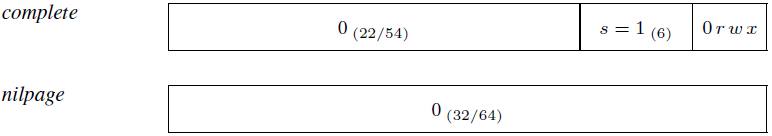
与Preempt Flags线程控制寄存器字段有关的编程接口查看英文原手册正文第 37 页。

# 第四章 地址空间和映射

## 4.1 Fpage [数据类型]

Fpages（灵活页面）是虚拟地址空间的区域。一个fpage由实际映射在这个区域中的所有页面组成，即没有内核映射的对象，即内核接口页面和UTCBs（An fpage consists of all pages mapped actually in this region sans kernel mapped objects, i.e., kernel interface page and UTCBs.）。Fpages的大小至少为1 K。对于特定的处理器，最小的fpage大小可能更大;例如奔腾处理器提供4K的最小页面大小，而Alpha处理器提供8K的最小页面。小于最小页面大小的页面被视为空页面。内核接口页面（参见第3页）指定硬件/内核支持哪些页面大小。大小为2s的fpage具有2s对齐的基地址b，即b≡0（mod 2s），其中对于所有体系结构，s≥10。

被映射的fpages被认为是不可分离的对象。也就是说，如果一个fpage被映射，之后映射器不能对该fpage部分取消映射；整个fpage必须在一个操作中被取消映射。然而，映射者（mappee）可以将fpage和map fpages（对象）分成更小的尺寸。部分取消映射fpage可能会或可能无法在某些系统上运行。内核将不会说明这样的操作是否成功。

特殊的fpage denoters描述了完整的用户地址空间和nilpage，其中complete fpage没有基地址，nilpage大小为0：

fpage访问权限

**rwx** r w x 位定义了fpage的访问权限

r 可读

w 可写

x 可执行

如果映射器本身具有访问权限，则设置为1的位允许对新映射/授权页面的相应访问。如果映射器本身没有访问权限，或者该位设置为零，映射/授权页面将不会获得相应的访问权限。

注意，处理器体系结构可能对访问权限组合施加限制。但是，对于任何处理器体系结构，只读（包括执行）rwx = 101和读/写/执行rwx = 111应该是有效的。内核接口页面（请参见第3页）指定处理器体系结构支持哪些访问权限。

与fpage数据类型有关的编程接口查看英文原手册正文第 40 页。

## 4.2 Unmap [系统调用]

函数形式：void Unmap (Word control)

|  |  |
| --- | --- |
| 输入参数 | 输出 |
| Word control | void |

指定的fpages（位于MR 0 · · ·)被取消映射。 Fpages被映射为IPC操作的一部分（请参阅Ipc系统调用）。被处理的页面是通过Ipc在MR中传递的。

**输入参数**：

k 指定保存将被取消映射的fpage的最高MRk。因此，将被取消映射的fpages数量是k + 1。

f = 0 在那些之前已经映射过这些fpages的所有地址空间中，这些fpages都会被递归的解除映射。但是，在当前的地址空间中，这部分的fpages保持不变。

f = 1 对于非当前地址空的所有其他映射了这些fpages的空间，解除映射的操作跟f = 0 是一样的，但是对于f = 1 的情况，当前地址空间中映射的fpages也会被解除映射。

**FpageList MR0 . . . k** 要处理的页面。

要被取消映射的Fpage。（即使实际上没有重新移动的访问权限，也使用术语“未映射”。）一个nilpage指定了一个空操作。

0 r w x 任何访问位设置为1都会撤消相应的访问权限。一个0位表示相应的访问权限不应受到影响。 典型例子：

=0111 将该fpage完全取消映射。

=0010 部分取消映射，只撤销可写性，这样的结果就是fpage被设置为只读。

=0000 没有取消映射。这种情况是特别有用的，如果只有脏（即被写过）和被访问的位应该被读取和重置而不改变映射。（This case is particularly useful if only dirty and accessed bits should be read and reset without changing the mapping.）

**输出参数**：

 **FpageList MR0 . . . k** fpage的访问状态位被更新。

所有执行取消映射操作处理的页面的引用、写入、执行的状态位被重置，并且所有处理的页面的按位“或”运算的旧值在MR0 . . . k中传递。对于不区分读访问权限和执行访问权限的处理器，R和X位是统一的：既可以同时被置位，也可以同时复位。重置状态位不是递归操作。但是，当前空间内页面的状态位值也将反映在递归映射上执行的访问。

**R** = 0 在最后的取消映射操作之后（或在初始映射操作之后），fpage的任何部分都没有被引用过。这包括所有递归映射的页面。

备注：引用的含义与读取略有不同。不被引用意味着不仅没有读取访问，而且也没有发生写入和执行访问。

**R** = 1 指定的fpage（包括所有递归映射）的至少一个页面在最后一次取消映射操作后（或初始映射操作之后）有被引用。 所有内核R位都被复位

备注：引用的含义与读取略有不同。 写访问和执行访问也设置R位。

**W** = 0 在最后的取消映射操作之后（或在初始映射操作之后）没有写入fpage的部分，即fpage是干净的。这包括所有递归映射的页面。

**W** = 1 指定的fpage中至少有一页（包括所有的递归映射）已经在最后的非映射操作之后（或在初始映射操作之后）被写入，即fpage是脏的。所有内核脏位都被复位。

**X** = 0 在最后的取消映射操作之后（或在初始映射操作之后），fpage的任何部分都没有被执行。 这包括所有递归映射的页面。

**X** = 1 指定的fpage（包括所有递归映射）至少有一个页面在最后一次取消映射操作之后（或在初始映射操作之后）被执行。 所有的内核X位都被复位。

备注：对于不区分读取和执行访问的处理器，如果R = 1，则X位设置为1。

针对Unmap的编程接口查看英文原手册正文第 43 页。

## 4.3 SpaceControl [特权系统调用]

函数形式：Word SpaceControl (ThreadId SpaceSpecifier,Word control, Fpage KernelInterfacePageArea, UtcbArea, ThreadId Redirector, Word& old Control)

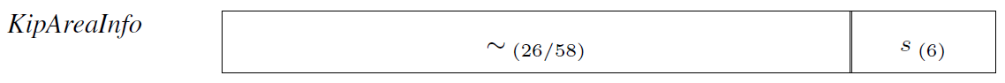
|  |  |
| --- | --- |
| 输入参数 | 输出 |
| ThreadId SpaceSpecifier  Word control  Fpage KernelInterfacePageArea  Fpage UtcbArea  ThreadId Redirector | Word result  Word control |

这是一个特权线程，比如根服务，可以通过这个系统调用来配置地址空间。

**输入参数**：

**SpaceSpecifier** 由于地址空间没有ID，所以线程ID被用作SpaceSpecifier。它指定了线程所在的地址空间。SpaceSpecifier线程必须存在，尽管它可能处于非活动状态或尚未启动。特别是，该线程可能驻留在尚未完全创建的空白地址空间中。

**KernelInterfacePageArea**

指定内核应该映射内核接口页面的fpage。提供的fpage必须具有在内核接口页面的KipAreaInfo字段中指定的大小，必须完全适合地址空间的用户可访问部分，并且不得与UTCB区域重叠（见下文）。内核接口页面的地址0被映射到fpage的基地址。 如果地址空间中至少有一个活动线程，则忽略该值。

s 内核接口页的大小是2s

**UtcbArea** 指定内核应该映射在地址空间中执行的所有线程的UTCB的fpage。 fpage必须完全适合地址空间的用户可访问部分，并且不得与KIP区域重叠。 fpage的大小必须至少是支持的最小硬件页面大小。 实际上，UTCB区域的大小限制了可以在地址空间中创建的最大线程数。 请参阅内核接口页面了解UTCB所需的空间和对齐方式。----UtcbInfo 字段

如果地址空间中至少有一个活动线程，则忽略该值

**Redirector** = nilthread

指定空间的当前重定向器设置不会被修改。

**Redirector** = anythread

允许指定空间内的所有线程与系统中的任何线程进行通信。

**Redirector**≠anythread ，≠nilthread

指定地址空间内的所有线程只允许将IPC发送到本地线程或与指定的重定向器处于相同地址空间的线程。所有其他的发送操作将被转移到重定向器，接收消息中的重定向位（请参见第67页）将被设置，而TCR中的“有意接收器”字段将指示消息的预期接收方。

**control** 控制字段是特定于体系结构的（见附录A.5）。对于某些体系结构，它是未定义的，但是应该为了向上兼容性而设置为零。

**输出参数**：

**result** 如果SpaceControl操作成功，则result结果是1，否则result的结果是0，并且TCR中的ErrorCode字段会指明出错原因。

**ErrorCode**

= 1 没有特权。 当前线程没有执行操作的权限。

= 3 无效的空间。 SpaceSpecifier参数指定了一个无效的线程ID。

= 6 无效的UTCB位置。指定的UTCB区域太小（请参见UtcbInfo），或者不在用户可访问的虚拟内存区域内（请参阅MemoryInfo的MemDescPtr字段）。

= 7 无效的KIP区域。指定的KIP区域太小（请参阅第4页上的KIP区域信息）或不在用户可访问的虚拟内存区域（请参阅第6页的内存描述符）或KIP区域与UTCB区域重叠。

**control** 调用操作时提供对线程有效的空间控制值。这个值是架构特定的。

针对SpaceControl的编程接口查看英文原手册正文第 46 页。

# 第五章 IPC

## 5.1 消息和消息寄存器（MRs） [虚拟寄存器]

消息可以通过IPC系统调用发送和接收（请参见Ipc系统调用）。基本上，发送者将消息写入发送者的消息寄存器（MRs），接收者从接收者的MRs读取消息。每个线程有64个MRs，MR0 . . . 63。消息可以使用一些或全部MRs来传送无类型的字数据;它可以包括由MRs指定的内存字符串和fpages。

MRs是虚拟寄存器（[见1.3](#_1.3__虚拟寄存器)），但它们有效时间比TCRs更为短暂。MRs是read-once寄存器：一旦MR被读取，其值不确定，直到MR被再次写入。IPC的发送阶段隐式读取所有的MRs;接收阶段将接收到的消息写入MRs。

read-once属性允许不仅通过专用寄存器或存储器位置来填充MRs，而且还可以通过通用寄存器来填充MRs。写入这样一个MR必须阻止相应的代码生成器使用通用寄存器；读MR之后就可以释放它了。通常，由IDL编译器生成的代码将在IPC系统调用之前加载MRs，并在Ipc之后将它们存储到用户变量中。

**消息**

一个消息最多由3个部分组成：必需的消息标记，后跟可选的无类型字，后跟可选的有类型项目。消息标签（message tag）始终保存在MR0中。消息标签包含消息控制信息和用户可自由设置的消息标签（message label）。内核不与其相关联。通常，消息标签用于对请求密钥进行编码或定义消息应该调用的方法。

u 在message tag之后的无类型字的数量。即MR1 . . . u保存的是无类型的字。u = 0表示这是一个没有无类型字的消息。

t 在无类型字之后跟着的有类型项目字的数量。如果该消息没有无类型字，则表示的是跟在message tag之后的有类型项目字的数量。MRu+1 . . . u+t保存的是有类型的项目字。如果该消息没有有类型的项目字，则t = 0。

flags 消息标记，见IPC系统调用。

label 自由可用，通常用于指定请求类型或调用的方法。

**无类型字[MR1 . . . u]** 可选的无类型字部分保存的是，从内核角度来看无类型的任意数据。数据只是复制到接收器。内核不与其相关联。

**有类型项目字[MRu+1 . . . u+t]**

可选的有类型项目字部分是字符串项目（string items），映射项目（map items），授予项目（grant items）和控制转移项目（ctrl transfer items）。

有类型项目字的类型编码在其第一个字的最低4位：

0hhC StringItem（[见5.5节](#_5.5__StringItem)）

100C MapItem（[见5.2节](#_5.2__MapItem)）

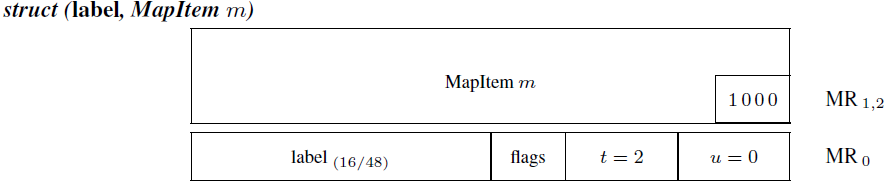
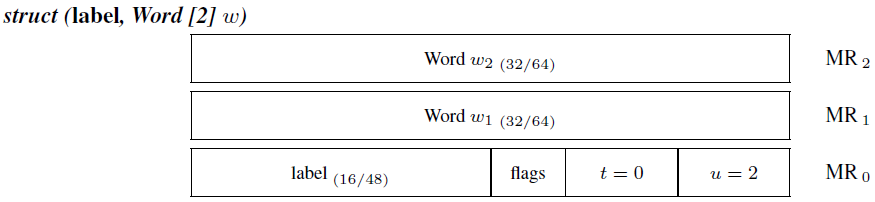
101C GrantItem（[见5.3节](#_5.3__GrantItem)）

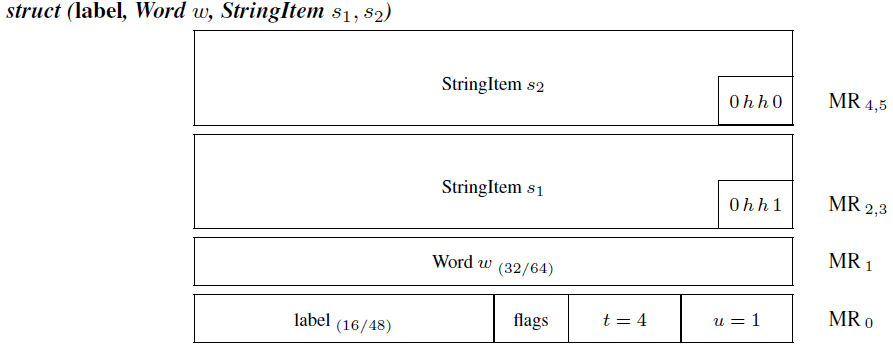
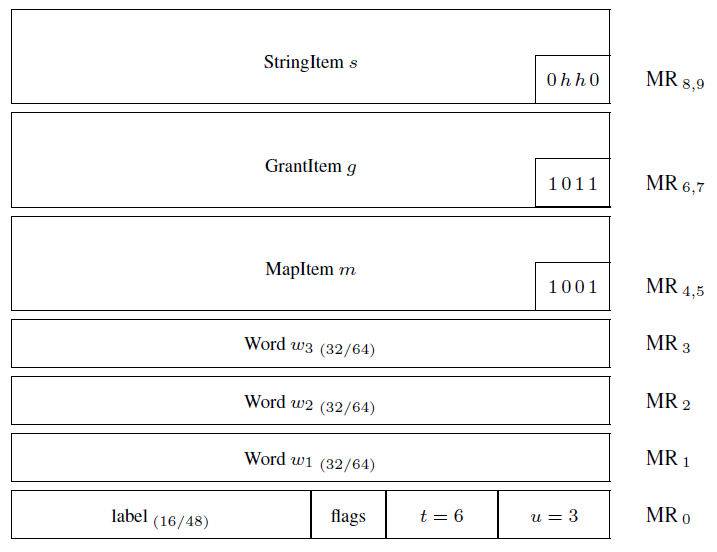
110C CtrlXferItem（[见5.4节](#_5.4__CtrlXferItem)）

111C 保留

C位表示当前有类型项目后面是另一个有类型项目（C = 1）还是有类型项目部分（C = 0）的最后一个。有类型项目必须完全符合MRu+1 . . . u+t。

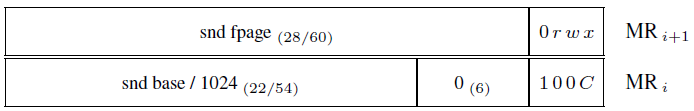
注意，C和t冗余地描述消息，这是有意的。C位保证了高效的消息解析，而t + u可以用来将一个消息的所有MRs存储到存储器而不解析完整的消息。在发送消息时，C位完全被忽略。然而，内核将确保接收器端的MRs将正确设置C位。

**消息举例**



跟消息相关的编程接口查看英文原手册正文第 52 页。

## 5.2 MapItem [数据类型]

应该被映射的fpage（[见4.1节](#_4.1__Fpage)）或IO fpage作为消息的一部分发送到mappee。映射操作在同一地址空间内是空操作。fpage是由一个双字描述符指定的。

**访问权限位 r w x** 对新映射页面的有效访问权限是通过snd fpage中指定的访问权限和映射器本身在该页面上具有的访问权限进行按位与计算来得到的。因此，映射器可以限制有效的访问权限，但不能扩展它们。

**snd base** 如果snd fpage的大小大于或小于接收者愿意接受映射的窗口（[见5.6节](#_5.6__字符缓冲和缓冲寄存器（BRs）)），则snd base将指定映射操作的语义。 如果snd fpage，2s的大小大于接收窗口2r，则snd base指示snd fpage要被发送到哪个区域。 更确切地说

addrs是snd fpage的基地址。如果snd fpage的大小2s小于接收窗口2r，则snd base指示在接收窗口中snd fpage被映射的位置。 更确切地说

其中addrr是接收窗口的基址。

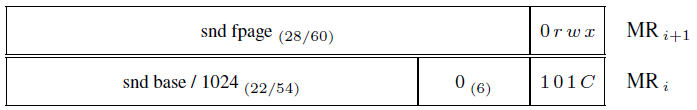
已映射到mappee地址空间中的页可能会与新映射的页冲突，发生冲突的旧页在新页映射之前会隐式地取消映射。出于性能原因，如果已经存在完全相同的映射，则可以在没有事先取消映射的情况下扩展访问权限。出现这种情况的四种可能：

* 映射器从与现有映射相同的地址空间映射
* 映射器映射来自与现有映射相同的虚拟源地址
* 映射器映射到与现有映射相同的虚拟目标地址
* 对象（物理地址）与现有映射相同

访问权限不能通过映射撤销。映射结果的访问权限是现有映射和新映射访问权限的按位或。访问权限不是递归的扩展。

与MapItem数据类型有关的编程接口查看英文原手册正文第 55 页。

## 5.3 GrantItem [数据类型]

应该被授予的fpage（[见4.1节](#_4.1__Fpage)）或IO fpage作为消息的一部分发送到mappee。映射操作在同一地址空间内是空操作。fpage是由一个双字描述符指定的。

**访问权限位 r w x** 对新映射页面的有效访问权限是通过snd fpage中指定的访问权限和映射器本身在该页面上具有的访问权限进行按位与计算来得到的。因此，映射器可以限制有效的访问权限，但不能扩展它们。

**snd base** 如果snd fpage的大小大于或小于接收者愿意接受映射的窗口（[见5.6节](#_5.6__字符缓冲和缓冲寄存器（BRs）)），则snd base将指定映射操作的语义。 如果snd fpage，2s的大小大于接收窗口2r，则snd base指示snd fpage要被发送到哪个区域。 更确切地说

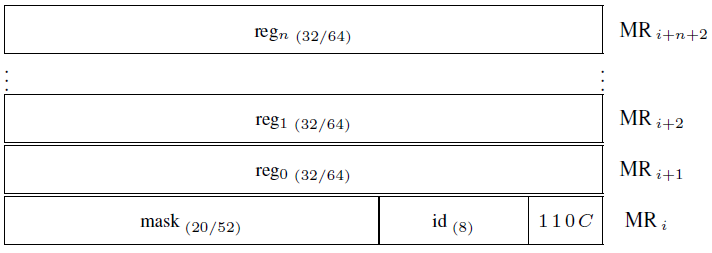
addrs是snd fpage的基地址。如果snd fpage的大小2s小于接收窗口2r，则snd base指示在接收窗口中snd fpage被映射的位置。 更确切地说

其中addrr是接收窗口的基址。

已映射到mappee地址空间中的页可能会与新映射的页冲突，发生冲突的旧页在新页映射之前会隐式地取消映射。

与GrantItem数据类型有关的编程接口查看英文原手册正文第 57 页。

## 5.4 CtrlXferItem [数据类型]

控制转移项目指定了消息接收者的控制状态，例如指令指针，堆栈指针或通用寄存器。 新值由内核在收到该项目后自动设置。控制转移项目的内容是特定于体系结构的。通常，控制转移项目的内容如下：

**id** 指定由该项目设置的一组控制传输寄存器的标识符。标识符是架构特定的。

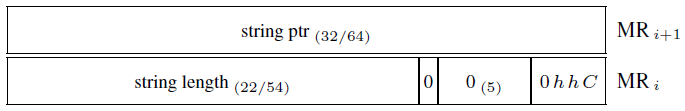
**mask** 指定要修改的集合中的寄存器的位掩码。

与CtrlXferItem数据类型有关的编程接口查看英文原手册正文第 58 页。

## 5.5 StringItem [数据类型]

字符串项指定用户空间中的字节序列。不需要对齐，最大字符串大小是4 MB。在发送消息时，这样的字符串在传送消息时被复制到接收方缓冲区。字符串项也用于指定接收方的缓冲寄存器中的接收缓冲区。

**简单字符串**

一个简单的字符串是一个连续的字节序列。

**string ptr** 要发送的字符串的起始地址或接收字符串的缓冲区的起始地址（没有对齐限制）。但是，字符串/缓冲区的地址必须在合法可寻址的用户空间中。

**string length** 要发送的字符串的长度或接收缓冲区的大小。在第二种情况下，可以接收到（包括）这个长度的字符串。即使64位处理器上的相应字段为54位宽，字符串的最大长度也是4 M字节。

**hh** 缓存性提示。除hh = 00外，此参数的语义取决于处理器类型。

hh = 00 使用处理器的默认缓存策略。通常情况下，cache分配给数据读取和写入（假设处理器的默认策略是写回和写分配）。

**复合字符串**

复合字符串是一个非连续的字符串，由多个连续的子字符串组成，可以分散在整个用户地址空间中。子串不能重叠。为了发送和接收IPC操作，复合字符串被处理为单个逻辑字符串。当通过IPC发送这样的字符串时，子字符串就像是一个连续的字符串（聚集）一样被传送。在接收端，复合字符串缓冲区被视为一个逻辑缓冲区。相应的收到的字符串分散在复合缓冲区的子字符串中。

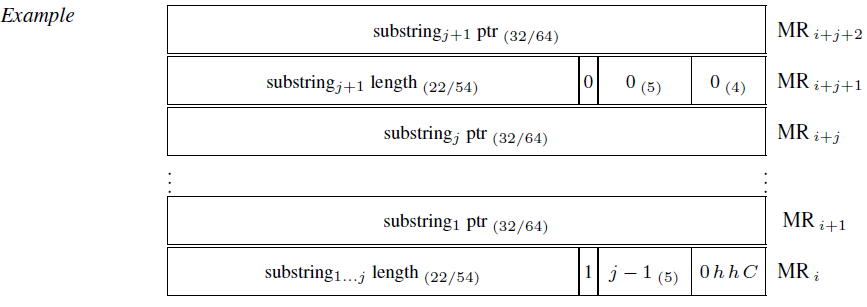
一个复合字符串可以被指定为一个子字符串序列，其中每个子字符串具有简单字符串的形式，除了最后一个子字符串外，其余子字符串的设连续标志c都被置位。如果后续的子字符串具有相同的大小，例如对于相同大小的缓冲区，则可以对所有的j个子字符串使用单个长度的字，从而只需要j + 1个字而不是2j个字。也就是只有一个string length字段，但是有个j个string ptr字段。

类型信息0hhC只对字符串描述符的第一个字是必需的。对于复合字符串描述符中的其他长度字，该字段将被忽略。（The field is

ignored for further length words in a compound-string descriptor.）

j 随后的string ptr的数量。这些字符串指针指定了具有全部相同子字符串长度的j个子字符串。

c = 0 continuation标志重置。表示这个字符串描述符描述的不是一个复合字符串，而后面跟的是一个当前字符串描述符substring length字段描述长度的字符串。

c = 1 continuation标志设置。其后的j个字都是相同substring length长度的字，而j个string ptr字之后是（至少）是一个子字符串描述符，即描述另一个长度字。

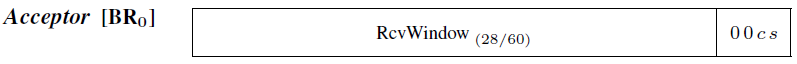
与StringItem数据类型有关的编程接口查看英文原手册正文第 60 页。

## 5.6 字符缓冲和缓冲寄存器（BRs） [虚拟寄存器]

为了接收包含字符串项目的消息，接收者必须指定适当的字符串缓冲区。这些缓冲区由字符串项目描述（[见5.5节](#_5.5__StringItem)）。缓冲区可以是连续的（简单字符串）或不连续的（复合字符串）。

这种缓冲区描述符保存在每个线程的33个缓冲寄存器BR0 . . . 32中。缓冲寄存器的数量足以指定例如31个同等大小的子缓冲器的一个复合缓冲器。只要不超过33个BRs，就可以指定多达16个缓冲区。

当收到消息时，第一个消息字符串项被复制到从BR1开始的第一个缓冲区字符串项中;下一个消息字符串项目被复制到下一个缓冲器字符串项目。缓冲器字符串列表通过使最后一个字符串的项目类型说明符中的C位归零而终止。

BRs是寄存器，它们是每个线程对象，只能被直接寻址，而不是间接通过指针。BRs是类似于TCRs的静态对象，即它们保持它们的值直到被明确修改。BRs可以映射到特殊寄存器或内存。

BR 0指定收到消息时接受哪些类型的项目。

RcvWindow Fpage（无访问权限位），指定接受映射和授权的地址空间窗口。 Nilpage拒绝任何映射或授予; CompleteAddressSpace接受任何映射或授予。

c 如果c = 1，则接受控制转移项目，CtrlXferItem。

s 如果s = 1，则接收字符串项目，StringItem。

**buffer string items[BR1. . . ]**

包含有效的缓冲区字符串项。如果BR0中的s = 0则忽略。

与BRs有关的编程接口查看英文原手册正文第 62 页。

## 5.7 Ipc [系统调用]

函数形式：MsgTag Ipc (ThreadId to, FromSpecifier, Word Timeouts, ThreadId& from)

|  |  |
| --- | --- |
| 输入参数 | 输出 |
| ThreadId to  ThreadId FromSpecifier  Word Timeouts | ThreadId from |

IPC是进程间通信和同步的基本操作。它可以用于地址空间内和地址空间间的通信。所有通信都是同步的和无缓冲的：当且仅当接收者已经调用相应的IPC操作时，才将消息从发送者传送给接收者。如果接收人没有准备好接收，则发件人阻塞；或者是发件人指定的时间段尚未过去，则发件人阻塞。

IPC可用于复制数据，以及映射或授予从发件人发往收件人的fpages。有关消息的说明，请[参见第5.1节](#_5.1__消息和消息寄存器（MRs）)。单个IPC调用结合了可选的发送阶段和可选的接收阶段。包含哪些阶段由参数to和FromSpecifier决定。发送阶段和接收阶段之间的转换是原子的。

IPC操作也受到MRs，BRs和一些TCRs的控制。 RcvTimeout和SndTimeout直接指定为系统调用参数。每个超时可以是0,∞（即永不过期），相对或绝对。有关超时的详细信息，请[见3.3节](#_3.3__Time)。

Ipc系统调用的变体：

为了实现特定于实现的优化，存在IPC系统调用的两种变体。在功能上，两种变体是相同的。对用户透明，内核实现可以统一两个变体或实现不同的优化功能。

Ipc 默认的IPC功能。 除非满足使用LIPC的所有标准，否则必须始终使用。

Lipc IPC功能可以优化发送消息到本地线程。在绝大多数的调用中绝对清楚的时候应该使用：

包含发送阶段

目标线程被指定为本地线程ID

包含接收阶段

目标线程在同一个处理器上运行

RcvTimeout是1

IPC不包含映射/授权操作

输入参数：

**to** = nilthread IPC没有发送阶段。

**to**≠nilthread to指定目标线程，IPC包括一个发送阶段。

**FromSpecifier** = nilthread

IPC没有接收阶段。

**FromSpecifier** = anythread

IPC包括一个接收阶段。来自任何线程（包括硬件中断）的消息都被接收。

**FromSpecifier** = anylocalthread

IPC包括一个接收阶段。传入消息可以从驻留在当前地址空间的任何线程接收。

**FromSpecifier**≠nilthread，≠anythread，≠anylocalthread

IPC包括一个接收阶段。传入消息只能从指定的线程接受。

（请注意，可以指定硬件中断。）

RcvTimeout 接收阶段等待，直到消息传输开始或RcvTimeout到期。 忽略只有发送阶段的IPC的操作。

对于相对接收超时值，发送阶段成功完成后，接收超时开始运行。 如果在消息传输开始之前接收超时到期，则IPC将以“接收超时”失败。如果超时时间为0，则收到待定的传入消息。

SndTimeout 如果在消息传输开始之前发送超时到期，则IPC操作失败，发送超时。发送超时为0确保IPC只有在调用发送IPC操作时被寻址的接收器准备好接收时才会发生。否则，IPC立即失败，即没有阻塞。

要发送的消息的消息头。只有高16/48位是随意访问的。低16位保存SndControl参数。它描述要发送的消息并包含一些控制位; 如果没有发送阶段则忽略。

u 在0号字（消息头）之后跟着的无类型字的数量。MR1 . . . u保存无类型的字。u = 0表示该消息没有无类型的字。

t 保存的有类型字的数量。如果u等于0，则消息头之后就是有类型字，否则就是在MRu之后是有类型的字，可以直到MR63，t = 0表示该消息没有有类型的字。

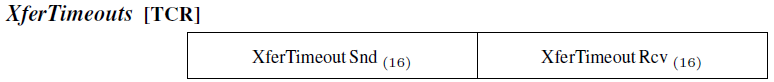
p = 0 正常（非传播）发送操作。收件人获取原始发件人的ID。

p = 1 传播发送操作。 VirtualSender TCR指定了发起者线程的ID。 （即，代表发送消息的线程）。如果发起者线程和当前发送者，或者当前发送者和接收者驻留在相同的地址空间中，则总是允许传播。否则，IPC发生未传播。如果发起者线程是等待（关闭）当前线程的中断线程，或者当前发送者是发起者线程的重定向器（或者存在从发起者到当前发送者的重定向器链），则传播也是允许的。

label 自由可用，通常用于指定请求类型或调用的方法。

**[MR1 . . . u]**  保存要发送的无类型字。如果没有发送阶段则忽略。

**[MRu+1 . . . u+t]** 保存要发送的有类型字。如果没有发送阶段则忽略。

一旦消息传输已经开始，传输消息的时间大致被发送者和接收者的XferTimeout的最小值所限制。 “粗略”意味着xfer超时仅在发送者或接收者地址空间中的消息复制引起页面错误时才被检查。复制数据和映射/授予被认为是不花时间的。相对传输超时始终是指消息传输的开始（实际上是在引发第一页错误时）。从逻辑上讲，在这一点上，它被转换成绝对超时，然后在消息传输中用作第一个和所有后续的页面错误RPC的发送和接收超时。

如果在消息传输期间有效的传输超时到期，则IPC以“xfer timeout”（双方）产生错误。附加信息指定页面错误是在接收者还是在发送者的地址空间中，以及指定消息的哪一部分已经被传送。每个线程有两个传输超时。一个用于发送阶段，一个用于接收阶段。

BR 0指定收到消息时接受哪些类型的项目。

RcvWindow Fpage（无访问权限位），指定接受映射和授权的地址空间窗口。 Nilpage拒绝任何映射或授予; CompleteAddressSpace接受任何映射或授予。

c 如果c = 1，则接受控制转移项目，CtrlXferItem。

s 如果s = 1，则接收字符串项目，StringItem。

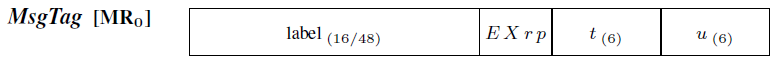
**buffer string items[BR1. . . ]**

包含有效的缓冲区字符串项。如果BR0中的s = 0则忽略。

输出参数：

**from** 从其收到IPC的发件人的线程标识。 线程ID作为本地线程ID传递，如果它们标识与当前线程在同一地址空间中执行的线程。 发件人是否将目的地指定为本地或全局ID无关紧要。

仅针对包含接收阶段的IPC操作进行定义。

如果IPC操作包括接收阶段，则MR0包含接收到的消息的消息头（message tag）。高16/48位包含用户指定的标签。低位描述接收到的消息，包含错误指示符和交叉处理器IPC指示符。

即使IPC操作不包括接收阶段，MR0也被定义。在发送情况下，MR0返回错误指示符。

u 字0之后的无类型字的数量。u = 0表示没有无类型的单词。对于没有接收阶段的IPC操作，u = 0。

t 接收到的字中有符号字的数量。t = 0表示没有有符号项目。对于没有接收阶段的IPC操作，t = 0。

p 传播IPC。 如果重置（p = 0）IPC不传播。如果设置（p = 1）IPC被传播并且FromSpecifier指示发起者线程的ID。 ActualSender指定执行传播的线程的ID。

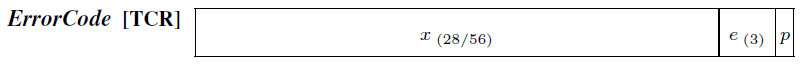
r 重定向的IPC。 如果重置（r = 0），IPC不是重定向的。 如果设置（r = 1），IPC被重定向到当前线程，TCR中的IntendedReceiver 字段指定应该接收消息的线程的ID。

X 跨处理器IPC。如果重置（X = 0），接收到的IPC来自与接收器在同一处理器上运行的线程。如果设置（X = 1），则接收到的IPC是交叉处理器。对于没有接收阶段的IPC操作，传送X = 0。

E 错误指示器。如果重置（E = 0），IPC操作终止成功。如果设置（E = 1）IPC失败。如果发送阶段成功，但之后发生接收超时，或者只能部分发送消息，则整个IPC都会失败。错误代码和附加信息可以从ErrorCode TCR中检索。如果错误代码表示部分接收到的消息，则标签，t和u字段有效。

label 接收到的消息的标签，如果IPC没有接收阶段，则label是0。

**[MR1 . . . u]**  保存接收到的消息的无类型字。如果没有接收阶段，则是未定义。

**[MRu+1 . . . u+t]**  保存接收到的消息的有类型字。如果没有接收阶段，则是未定义。

仅在MR0中的错误指示符E被设置时才被定义。IPC失败，即没有正确完成。x字段取决于错误代码，见下文。p字段指定发送或接收阶段发生的错误。如果在接收阶段发生错误，则发送阶段（如果有的话）之前已成功完成。如果在发送阶段发生错误，则跳过接收阶段（如果有的话）。

 p 指定错误是发生在发送阶段（p = 0）还是接收阶段（p = 1）。

在通信对方线程涉及消息传输之前发生错误。因此，错误信号只发送给调用失败的IPC操作的线程。

e = 1 超时。在这种情况下输出参数from未定义。

e = 2 不存在对方线程。如果错误发生在发送阶段，则不存在to。(任意线程作为目标都是非法的，也会引发此错误。)如果错误发生在接收阶段，则不存在FromSpecifier。(FromSpecifier =anythread是合法的，因此永远不会引发此错误。)

e = 3 被另一线程取消（系统调用改变寄存器）。

IPC操作已经涉及到通信对方线程，因此错误信号发送给两个线程。

offset 消息传输已经开始，无法完成。偏移量精确地标识到目前为止通过字符串项已成功传输的字节数。

e = 4 消息溢出。（1）接收缓冲字符串太短，（2）缓冲区字符串项目不足，（4）fpage的映射/授权由于系统没有足够的页表空间而失败。与收到的MRs一起的offset保证发送者和接收者准确地明确原因。

e = 5 在调用者的地址空间中的页面错误期间传输超时。

e = 6 在通信的对方线程地址空间的页面错误期间，传输超时。

e = 7 由别的线程中止。（系统调用交换寄存器）

**页出错**

在ipc期间可能发生三种不同类型的页面错误：预发送页出错（pre-send pagefaults），接收后页出错（post-receive pagefaults）和传输页错误（xfer pagefaults）。 从安全角度来看，只有xfer pagefault是至关重要的。幸运的是，没有字符串的消息永远不会引发xfer页错误，因此不需要特殊的页面错误规定：

**pre-send pagefaults ：** 在消息传输真正开始之前发生在发送者的上下文中。目标线程不涉及; 特别是没有锁定。因此，当发送者的预发送页错误正在被处理的时候，目标线程可能会收到另一个消息或超时处理。发送和传输超时不能控制预发送页面错误。预先发送的页错误从安全的角度来看是不加批判的，因为只涉及发送者自己的调页程序，只有发送者可能遭受其潜在的错误行为。

**Post-receive pagefaults ：** 消息传输完成后在接收方的上下文中发生。发件人线程不再涉及，特别是不再被锁定。因此，post-receive pagefault不会跟发送和传输超时有关系。 就像预发送页错误一样，从安全的角度来看，post-receive页面错误也是不重要的，因为只涉及到接收者和调页程序。

**Xfer pagefaults ：** 发生在发送消息的时候，发送者和接收者都受影响。因此，从安全角度来看，xfer页错误是至关重要的：如果在接收者的空间中发生这种页面错误，发送者可能会被恶意的接收方调页程序挨饿。发件人空间中的xfer页错误和恶意发件人调页程序可能会使接收者挨饿。因此，xfer页面错误由发送者和接收者的xfer超时的最小值控制。

但是，xfer页错误只能在传输字符串时发生。在没有字符串的情况下发送消息或者在没有接收的情况下接收消息字符串缓冲区一定不会引发xfer页错误。

针对Ipc的编程接口查看英文原手册正文第 68 页。

# 第六章 其他

## 6.1 异常处理程序 [TCR一个字段]