L4eX实验内核

参考手册

版本十2

系统体系结构组

计算机科学 Dept。

卡尔斯鲁厄技术研究所

(L4Ka 团队)

info@l4ka.org info@l4ka. org

文档修订版 7-6288f0536ce1

2011年10月13日

版权所有2001-2004年由系统架构组, 计算机科学系, Universitat¨卡尔斯鲁厄。

本规范 "按目前情况" 提供, 不作任何保证, 包括任何有关适销性、不侵权、适合任何特定用途的保证, 或任何建议的任何担保, 规格-阳离子或样品。

特此准许在任何媒体中为任何目的复制和分发本规范的逐字副本, 无需费或版税。没有权利创造修改或衍生物由这个执照授予。L4Ka 团队可能会对此进行更改规格在任何时间, 恕不另行通知。本文档的最新修订版可在<http://l4ka.org/>.

**内容**

|  |  |  |  |
| --- | --- | --- | --- |
| [关于本手册](#page5) | | | v |
|  | [介绍性发言](#page5). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | v |
|  | [了解此文档](#page6). . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . . . | | 我们 |
|  | [法](#page7). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 七 |
|  | [使用 API](#page8). . . . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 八 |
|  | [修订历史记录](#page9). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 九 |
| [1](#page13) | [基本内核接口](#page13) | | 1 |
|  | [1。1](#page14) | [内核界面页](#page14). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ... | 2 |
|  | [1。2](#page19) | [KERNELINterface](#page19). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 7 |
|  | [1。3](#page23) | [虚拟注册培训](#page23). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 11 |
| [2](#page25) | [线程](#page25) | | 13 |
|  | [2。1](#page26) | [threadid](#page26). . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 14 |
|  | [2。2](#page28) | [线程控制寄存器 (TCRs)](#page28). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . | 16 |
|  | [2。3](#page30) | [ExchangeREGISTERS](#page30). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 18 |
|  | [2。4](#page35) | [THREADC控制](#page35). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 23 |
| [3](#page39) | [调度](#page39) | | 27 |
|  | [3。1](#page40) | [时钟](#page40).. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. | 28 |
|  | [3。2](#page41) | [S系统C锁](#page41). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ... . . . . . . . . . | 29 |
|  | [3。3](#page42) | [时间](#page42). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 30 |
|  | [3。4](#page44) | [THREADS女巫](#page44). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 32 |
|  | [3。5](#page45) | [S安排](#page45). . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . | 33 |
|  | [3。6](#page49) | [抢占标志](#page49). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 37 |
| [4地址空间和映射](#page51) | | | 39 |
|  | [4。1](#page52) | [Fpage](#page52). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 40 |
|  | [4。2](#page54) | [Unmap](#page54). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. | 42 |
|  | [4。3](#page57) | [S步伐C控制](#page57)... . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. | 45 |
| [5](#page61) | [ipc](#page61) |  | 49 |
|  | [5。1](#page62) | [消息和消息寄存器 (夫人)](#page62). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 50 |
|  | [5。2](#page67) | [MapItem](#page67). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ... . . . . . . . . . . . . . | 55 |
|  | [5。3](#page69) | [GrantItem](#page69). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 57 |
|  | [5。4](#page70) | [CtrlXferItem](#page70). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 58 |
|  | [5。5](#page71) | [StringItem](#page71). . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . | 59 |
|  | [5。6](#page74) | [字符串缓冲区和缓冲区寄存器 (BRs)](#page74). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 62 |
|  | [5。7](#page76) | [Ipc](#page76). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 64 |
| [6](#page83) | [杂项](#page83) | | 71 |
|  | [6。1](#page84) | [ExceptionHandler](#page84). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 72 |
|  | [6。2](#page85) | [警察标志](#page85). . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 73 |
|  | [6。3](#page86) | [PROCESSORC控制](#page86). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . | 74 |
|  | [6。4](#page88) | [M埃默里C控制](#page88). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 76 |
| [7](#page91) | [协议](#page91) | | 79 |
|  | [7。1](#page92) | [线程启动协议](#page92). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 80 |
|  | [7。2](#page93) | [中断协议](#page93). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 81 |
|  | [7。3](#page94) | [Pagefault 协议](#page94). . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . . . . . . | 82 |
|  | [7。4](#page95) | [抢占协议](#page95). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 83 |
|  | [7。5](#page96) | [异常协议](#page96). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 84 |
|  | [7。6](#page97) | [扩展控制传输协议](#page97). . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 85 |
|  | [7。7](#page101) | [Sigma0 RPC 协议](#page101). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. | 89 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 四 |  |  |  | 内容 |
|  | [7。8](#page104) |  | [一般引导](#page104). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . . . . 92 |
| [A](#page107) | [IA-32 接口](#page107) | | | 95 |
|  | [A. 1](#page108) | | [虚拟寄存器](#page108). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . . . . 96 |
|  | [A. 2](#page111) | | [Systemcalls](#page111). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. | . . . . . 99 |
|  | [A. 3](#page114) | | [内核功能](#page114). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . . . .102 |
|  | [A. 4](#page115) | | [IO 端口](#page115). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . | . . . . .103 |
|  | [A. 5](#page116) | | [空间控制](#page116). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . . . .104 |
|  | [一个.6](#page118) | | [缓存提示](#page118). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . . . .106 |
|  | [A. 7](#page119) | | [内存属性](#page119). . . . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . . . .107 |
|  | [A. 8](#page120) | | [异常消息格式](#page120). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . . . .108 |
|  | [9 IA-32 控制转移项目](#page121). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | | . . . . .109 |
|  | [A. 10](#page123) | | [处理器镜像](#page123). . . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . | . . . . .111 |
|  | [A. 11](#page124) | | [启动](#page124). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . . . .112 |
|  | [12支持硬件辅助虚拟化](#page125). . . . . . . . . . . . . . . . . . . . . . . . . . | | | . . . . .113 |
|  | [13 MSR-Fpage](#page130). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . | | | . . . . .118 |
| [B](#page133) | [AMD64 接口](#page133) | | | 121 |
|  | [B. 1](#page134) |  | [虚拟寄存器](#page134). . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . . . . . . | . . . . .122 |
|  | [B. 2](#page137) |  | [Systemcalls](#page137). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. | . . . . .125 |
|  | [B. 3](#page142) |  | [IO 端口](#page142). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . . . .130 |
|  | [B. 4](#page143) |  | [Cacheabili公司提示](#page143). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . . . .131 |
|  | [B. 5](#page144) |  | [内存属性](#page144). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . | . . . . .132 |
|  | [B. 6](#page145) |  | [异常消息格式](#page145). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . . . .133 |
|  | [B. 7](#page146) |  | [处理器镜像](#page146). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . . . .134 |
|  | [B. 8](#page147) |  | [启动](#page147). . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . | . . . . .135 |
| [C](#page149) | [PowerPC 接口](#page149) | | | 137 |
|  | [C. 1](#page150) |  | [虚拟寄存器](#page150). . . . . . . ... . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . . . .138 |
|  | [C. 2](#page152) |  | [Systemcalls](#page152). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. | . . . ..140 |
|  | [C. 3](#page156) |  | [内存属性](#page156). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . . . .144 |
|  | [C. 4](#page157) |  | [空间控制](#page157). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . . . .145 |
|  | [C. 5](#page158) |  | [异常消息格式](#page158). . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . | . . . . .146 |
|  | [C. 6](#page160) |  | [处理器镜像](#page160). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . . . .148 |
|  | [C. 7](#page161) |  | [启动](#page161). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . . . .149 |
|  | [C. 8](#page162) | [支持硬件加速虚拟化](#page162). . . . . . . . . . . . . . . . . . . . . . . . . | | . . . . .150 |
| [D](#page167) | [PowerPC64 接口](#page167) | | | 155 |
|  | [D. 1](#page168) | | [虚拟寄存器](#page168). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . . . .156 |
|  | [D. 2](#page170) | | [Systemcalls](#page170). . . . . . . . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . | . . . . .158 |
|  | [D. 3](#page175) | | [内存属性](#page175). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . . . .163 |
|  | [D. 4](#page176) | | [异常消息格式](#page176). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . . . .164 |
|  | [D. 5](#page178) | | [启动](#page178). . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . . . .166 |
| [E](#page179) | [通用 BootInfo](#page179) | | | 167 |
|  | [E. 1](#page180) |  | [通用 BootInfo](#page180). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . . . .168 |
|  | [E. 2](#page182) |  | [BootInfo 记录](#page182). . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . | . . . . .170 |
| [F](#page185) | [发展评论](#page185) | | | 173 |
|  | [F. 1](#page185) |  | [异常处理](#page185). . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . . . . . . . | . . . . .173 |
| [触发、类型和常量的表](#page187) | | | | 175 |
| [索引](#page195) | |  |  | 183 |

**关于本手册**

**介绍性发言**

**的目的此文档**

此 L4 参考手册可作为所有 L4 api 和 ABIs 的定义文档。首先, 它将 L4 微内核实现作为 api/abi 供应商, 并将代码生成器或库实现作为 api/abi 用户来处理。参考手册假定熟悉基本的 L4 概念和硬件架构。它的关键点是精确的定义, 而不是解释和例证。的

L4 系统程序员手册

旨在支持使用 L4 的程序员。它解释和说明基本概念, 并更详细地描述了如何 (以及为什么) 使用哪个函数等。

**维护**

该文件由以下 L4Ka 团队成员维护:

Dannowski (ud3@ira.uka.de ud3@ira. 乌卡) 约书亚 LeVasseur (jtl@ira.uka.de jtl@ira. 乌卡. de) Espen Skoglu(esk@ira.uka.de esk@ira 乌卡 de)

Volkmar Uhlig (volkmar@ira.uka.de volkmar@ira 乌卡) 1月 Stoess (stoess@ira.uka.de stoess@ira. 乌卡. de)

**学分**

本手册是基于 Jochen 利特克的最后草稿。它反映了他在 L4 微内核和系统研究方面的杰出工作。只有他系统设计的愿景使这项工作成为可能。Jochen 定义了近十年来微内核设计的艺术状态。我们感谢他的支持, 并努力以他的精神继续这项工作。

有助于提高这个参考人的贡献性征和 L4 接口来自许多人, 特别是来自艾伦 Au, 马库斯 Brinkmann, 菲利普逞, 凯文埃尔芬斯通, 布赖恩福特, 安德烈 Haeberlen, 赫尔曼 Hartig, ¨ Gernot 犹豫不决, 迈克尔 Hohmuth, 特伦耶格, 本莱斯利, Jork 失败者, ¨弗兰克我hnert, Yoonho 公园, 马克塞勒姆, 卡尔范罗伯特·范斯海克, 塞巴斯蒂安伯格, ¨ Cristan Szmajda, 哈维碰, 马库斯 Volp, ¨尼尔 Walfield, 亚当威金斯, 西蒙温伍德, 和吉恩 Wolter。

六 关于本手册

**文档历史记录**

|  |  |
| --- | --- |
| Jochen Liedtke | ??/??-06/01 |
| 由 L4Ka 团队审阅 | 06/01-09/01 |
| L4 开发者评论 | Q4/01 |
| 释放 | 01/02 |

**了解此文档**

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。其中一些可以在本文件的附录中找到。

在这篇文档中, 我们区分逻辑接口, 一般二进制 Interface、通用编程的面对面、方便的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, 一个对于32位和第二个64位处理器)。两个版本一起形成 L4 的一般二进制接口。

从纯粹的观点来看, 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两个更多的接口类来补充:

泛型编程接口

泛型编程接口定义逻辑接口的对象和泛型二进制接口作为伪 c++ 类。常规 c 的语言绑定大部分与 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.

方便的编程接口

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, 这是一个接口之上的微内核, 使最常见的操作更容易使用的程序员。

重要的是要明白, 方便和易用性, 而不是完整性, 是这个接口的标准。方便 progra体操接口支持程序员, 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) 的通用编程接口的操作ce.

显然, 方便的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方便编程接口。

实现备注: 虽然方便接口可以在通用编程接口之上完全实现, 即处理器独立的, 方便接口的者可以实现它的硬件依赖, 从而纳入任何优化, 成为可能通过特定的处理器特定的二进制接口。

最后一个接口类不是 t 的一部分通用 L4 API 规范。

关于本手册 七

特定于处理器的二进制接口

定义处理器特定的二进制接口。

**法**

**基本数据类型**

此参考手册描述了 L4 API 和 ABI32位和64位处理器。数据类型字表示32位处理器上的32位无符号整数和64位处理器上的64位无符号整数。Word64、Word32 和 Word16 表示与处理器类型无关的64、32和16位单词。

**特权线程**

某些系统调用只能由特权线程执行。任何属于同一地址空间的线程, 作为内核在启动时创建的初始线程之一 (请参见页[92)](#page104)被视为 privileged

**位字段**

位字段长度表示为下标(i = j)在这里我涉及到一个32位处理器和 j 到一个64位处理器。位字段下标我)指定对32位和64位处理器具有相同大小的位域。字节的fsets 被给作为

i = j 表示32位和64位处理器。如果指定单词的所有位字段只加32位, 则在64位处理器上剩余的上32位是未定义或忽略的。

**未定义、忽略和未更改**

–

输出参数或位字段可以是未定义的。对应的参数或字段由。它们对输出没有定义的值, 也就是说, 它们可能有任何值, 甚至可能无法访问。任何算法都依赖于未定义的参数或位字段的值为不正确。+ 无隐蔽通道。

输入参数或位字段可以被指定为忽略, 由-。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。-我它还用于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。

在特定于处理器的接口中, 寄存器有时定义为不变。这是 de-noted 的。

**向上兼容性**

下面的内容适用于将来的 API 版本和 sub-versions, 它们被指定为与当前版本的上行兼容。

输出参数和位字段.

当前定义为未定义 () 的字段可以指定为已定义。这些新定义的字段将只提供附加信息。如果系统调用与当前 API 中指定的方法完全相同, 则可以忽略它们。

八 关于本手册

输入参数和位域。

当前定义为忽略 (–) 的字段可以指定为已定义。但是, 这些字段的内容只与新定义的功能相关。这些领域将如果系统调用与此 API 中指定的 "旧" 语义一起使用, 则忽略。

**使用 API**

**命名**

程序员可以使用在通用和方便编程接口中定义的所有函数、类型和常量定义。手动。但是, 所有定义必须以字符串 "L4" 作为前缀, 类型名称必须包含 "t" 后缀 (例如, 使用 "L4 ipc ()" 和 "L4 MsgTag t", 而不是 "ipc ()" 和 "MsgTag")。当前仅为 c++ 和 c 定义接口。在如此我的情况下, 用于函数名的命名会导致 C 语言中的冲突。必须使用函数定义后括号中指定的替代名称来解决这些冲突。

**包含文件**

相关包含文件e 所需的定义和声明在通用和方便接口部分的开头指定。一般来说, 手册中每个章节都有一个包含文件。如果只需要基本的 L4 数据类型, 就可以包括 u唱<l4/类型。>.

关于本手册 九

**修订历史记录**

**修订版1**

初步修订。

**修订版2**

–阐明内核界面页和内核配置页的规范。

-UntypedWords 和 StringItems受体常数与函数 UntypedWords (MsgTag) 和 Strin-gItems (MsgTag) 函数声明发生碰撞。更名为 UntypedWordsAcceptor 和 StringItemsAcceptor。

–为 L4Ka 内核更改了内核 id。

–时间上的运算符的固定返回类型类型.

–将 fpages 中的 wrx 访问权限更改为执行。也改变了 W RX 参考位在 fpages 返回的 Unmap系统调用 RW 十

–重命名的 MsgBuffer 上运行的函数追加。

–地址空间删除现在执行, 删除AS 的最后一个线程。这使得创建和删除对称 (通过 ThreadControl)。之前, 所有线程, 但最后被删除的 ThreadControl, 最后由 SpaceControl。

–添加用于创建 ThreadIDs 和检索版本和 t 的功能hread 数字。固定大小的 MyLocalId 和 MyGlobalId TCRs。

–指定可用于用户线程的前三线程版本号专用于0,1, 并分别进行根任务。

-改变了在魔法领域的编码e 基普回0xE6 与以前版本的内核兼容。

–在内核界面页和内核配置页中更改了内存描述符 (例如, 专用内存), 以使用类型化描述符的数组, 而不是静态数目的 p重新定义的。

–为 PowerPC 接口添加了一个附录。

–添加 Niltag MsgTag 常量。

-MsgBuffer 结构的体积减小到32。

–将取消 () 和刷新 () 的单个 Fpage 和参数更改为通过值传递。

–更改了 ia32 内核功能字符串 "小" 到 "smallspaces"。

–添加了 ia64 接口的附录。

-更改了 ia32 IPC 和 LIPC ABI, 以便更好地适用于 sysenter/sysexit 和 gcc 的通用硬件。

–增加了 ProcDesc 的便利功能。

–指定包括要用于 API 的各个部分的 de 文件。

–允许特权线程访问 ia32 模型特定的寄存器。

–更改了系统调用链接的 ia64 ABI, 而pc而我ipc系统调用。

–新线程的 UTCB 位置现在显式pecified 由参数到 THREADC控制系统调用。

–添加了冲突函数名的 C 版本。

x 关于本手册

–为 fpages、映射项目、授予项目、字符串项和内核添加了许多方便功能接口页字段。

–在 "映射" 和 "授予项目" 中添加了 "发送基" 的说明。

–更改版本 x 2 和版本 4 API 的 subversion 编号。

–将 XferTimeout TCR 重命名为 XferTimeouts, 并拆分为单独的发送和接收超时。

–为每个体系结构特定的 TCR 节添加了两个线程特定的单词。这些词可以自由使用, 例如 IDL 编译器。

–将 L4Ka 内核的名称更改为正式名称。添加了 L4Ka::Strawberry。

-增加了 Alpha 和 MIPS64 的附录。

**修订版3**

–在内核界面页中对供应商字段的说明进行了澄清。

–添加了 NumMemoryDescriptors () 方便功能。

-阐明了 MemoryDescType () 函数的返回值。

–固定错误的等待超时规范 ()和 ReplyWait 超时 ()。

–在 E 中添加一个新的 h 标志来控制参数xchangeREGISTERS系统调用。h 标志控制是否应忽略 "恢复/暂停" 标志。

–将 TimePeriod () 的参数类型从 "int" 更改为 "Word64"。

–固定MsgTag 输入/输出的规格中的错字pc参数.

–添加评论pc系统-调用消息寄存器的读一次语义。

–将成员名称 "raw" 添加到声明为结构的所有 L4 类型。

–重命名的开始 () 和停止 () 函数开始 () 并停止 ()。

–描述未定义的 UTCB 内存区域的语义。

– PowerPC 上的前10消息寄存器现在定义为由物理寄存器支持。

– Alpha 上的前9消息寄存器现在定义为由物理 reg 支持isters

-固定先生0注册分配为 IA32 系统和适应系统相应地。

**修订版4**

-增加了 AMD64 的附录。

–更改 MIPS64 IpcABI 包含9消息寄存器。

–添加 S系统C锁调用为 MIPS64。

-澄清了在 IPC 传播期间, 中断线程可能是发起方线程的事实。

–添加了 SPARC v9 的附录。

–高内存描述符字段现在指定内存区域中的最后一个可寻址字节。

关于本手册 xi

**修订版5**

–代码 TCR 现在是一个通用占位符, 用于错误描述系统调用失败。

– M埃默里C控制现在返回一个结果参数。

–为各种系统调用定义的错误代码 (ExchangeREGISTERS, THREADC控制, S安排, S步伐C上-

–为错误代码值定义的方便定义。

-改变了 IA32 S系统C锁ABI 来修理 EDI 寄存器。

–指定地址空间的基普区域和 UTCB 区域不得重叠。

–对于 PowerPC 系统调用陷阱 "ption IPC, 使用消息标签-5, 并保留寄存器 LR。

– ExchangeREGISTERS系统调用不能再激活非活动线程。

–设置权限 () 的 Fpage 参数现在通过引用传递。

-固定的不一致的数量可缓冲区寄存器。

–将 void 重命名为 void、char 到 char 以及 bool 为 bool。

–启动 () 方便功能现在中止任何正在进行的 IPC 操作。

–在单个 fpage 上运行的取消 () 和冲洗 () 方便功能现在提供了统计修改过的 fpage 的美国比特。

-MIPS64 现在使用 k0 (26 美元) 的寄存器来存放 UTCB 地址。

–为 M 添加了两个新的内存类型埃默里C控制MIPS64。

–添加了通用 BootInfo 的附录。

-请明确指出, 不可能激活地址空间中未正确配置 S 的线程步伐C控制.

-增加了手臂的附录。

–如果使用64位内核, 则将内核接口页的第二个32位单词定义为0。

–更改 PowerPC 系统的 ABI 呼叫 Unmap和M埃默里C控制.

**修订版6**

–从 P 中移除控制参数ROCESSORC控制系统调用绑定和从 PROCESSORC控制阿尔法系统呼叫 ABI

–向 E 添加传递参数xchangeREGISTERS控制调用是否应传递线程的旧值。针对 MP 系统。

–添加和减去两个时钟值的运算符。

–指定0也理解 pagefault 协议, 和匿名0请求只将常规的内存视为可用。

–增加了 ARM 通用异常 IPC 消息格式。

–更改 MIPS64 调用异常 IPC 消息格式以更接近常规异常消息格式。

-澄清了 IPC 发送和收到.

–更改了 AMD64 和 IA32 特定 IO 端口映射接口。内核现在使用自定义 pagefault 标签将 IO pagefaults 传播到调页程序。

–更新了内核接口 pa 中 API 版本、内核 Id 和供应商的有效编码ge.

–在新线程开始执行的处理器上进行清楚。

第十二 关于本手册

– ProcessorNo 现在返回一个单词而不是 int。

–增加了读取 IO fpages 的功能。固定包含使用 IO fpages 的路径。

–定义, S安排如果调用线程驻留在与目标线程相同的地址空间中, 也允许系统调用。

–重新定义 IA32 内存属性的值, 以更好地对应于体系结构的默认页属性表 (PAT) 值s

**修订版7**

-已删除已停产的体系结构 IA-64、ARM、Alpha、MIPS、MIPS64、SPARC

–引入一个称为控制转移项 (CtrlXferItem) 的新项目, 它允许为 I 指定控件状态pc和 ExchangeREGISTERS系统调用建筑伸展依赖的方式。

–向 E 中的控制参数添加了三新标志 W、R 和 CxchangeREGISTERS系统调用。新的标志允许使用 CtrlXferItems 修改线程状态。

–为 pagefaults、异常添加了一组扩展协议,nd preeemptions, 它通过 CtrlXferItems 来检索和更新线程状态。

–增加了对 IA-32 和 PowerPC-32 处理器硬件辅助虚拟化的支持。

-引入的协议和空间控制扩展, 用于映射扩展的物理(64 位) 页面上的 IA-32 和 PowerPC-32

–修正了 S 的错误规范安排系统-调用 AMD-64 处理器。

**1章**

基本内核接口

2 内核界面页

**1.1 内核界面页** **[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。它不是由调页程序映射的, 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。 (请参阅页[7)](#page19).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | L4 版本部件 | | |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  | KernDescPtr | | |  |
| 供应商 |  | 内核版本 |  | KernelGenDate |  | 内核ID | | | | |  |
|  |  |  |  |  |  |  |  |  |  |  | ProcDescPtr | | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | InternalFreq |  | ExternalFreq | | | | |  |
|  |  |  |  |  | |  |  |  |  |  | MemDescPtr | | |  |
|  |  |  |  |  | | |  |  |  |  |  |
|  |  |  |  | MemoryDesc | | | | | | |  |
|  | |  |  |  |  |  |  |  |  |  | + F0/+1E0 | | |  |
|  | |  | |  |  |  |  |  |  |  |  |
| ~ |  | 调度系统SC |  | 线程切换SC |  | 系统时钟SC | | | | |  |
| 交换寄存器SC |  | 取消映射SC |  | LIPC SC |  | LPC SC | | | | | + E0/+ 1C0 | | |  |
|  |  |  |  |  |  |  |  |  |  |  | + D0/+ 1A0 | | |  |
| 内存控制psc |  | 进程控制psc |  | 线程控制psc |  | 空间控制psc | | | | |  |
|  |  |  |  |  |  |  |  |  |  |  | + C0 | /+180 | |  |
| ProcessorInfo |  | pageinfo |  | ThreadInfo |  | ClockInfo | | | | |  |
|  |  |  |  |  |  |  |  |  |  |  | + B0 | /+160 | |  |
| ProcDescPtr |  | BootInfo |  |  | ~ | | | | | |  |
| KipAreaInfo |  | UtcbInfo |  |  | ~ | | | | | | +A0 | /+140 | |  |
| ~ | | | | | | | | | | | +90 | /+120 | |  |
| ~ | | | | | | | | | | | +80 | /+100 | |  |
| ~ | | | | | | | | | | | +70/ | | + E0 |  |
| ~ | | | | | | | | | | | +60/ | | + C0 |  |
|  | | ~ |  | MemoryInfo |  | ~ | | | | | +50/ | | +A0 | AAAA0 |
| ~ | | | | | | | | | | | +40 | / | +80 |  |
| ~ | | | | | | | | | | | +30 | / | +60 |  |
| ~ | | | | | | | | | | | +20 | / | +40 |  |
| ~ | | | | | | | | | | | +10 | / | +20 |  |
| KernDescPtr |  | API 标志 |  | API 版本 |  | 0(0 = 32) | k' | 230 | "4" | 的' |  |  | +0 |  |

+ C/+18 +8/+10 +4/+8 +0

|  |  |
| --- | --- |
| 内核界面页 | 3 |

请注意, 此内核界面页面基本上是向上兼容版本2和十 x 0 的内核信息页面。在内核接口页对象的开头都标识了字符串 "L4uK"。

版本/id编号约定: 版本/副版本/subsubversion 号和 id/subid 号的最高有效位0表示正式版本/ id，并且在所有供应商中都是全球唯一的。将最高有效位设置为1的版本/ ID号码表示实验版本/ ID，并且可能仅在供应商的情况下才是唯一的。

API 描述

API 版本

API 标志

ee

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 版本(8) | |  | 颠覆(8) | |  | (16) | | | |  |
| 版本 |  | | | | |  |  |  |  |  |
| 颠覆 | | |  | |  |  |  |  |  |
|  |  |  |  |  | |  |  |  |  |  |
| 0x02 |  |  |  | 版本2 | |  |  |  |  |  |
| 0x83 | 0x80 | |  | 实验版本十0 | | | | | |  |
| 0x83 | 0x81 | |  | 实验版本十1 | | | | | |  |
| 0x84 | 冯智 | |  | 实验版本十 2 (修订版) | | | | | |  |
| 0x85 |  |  |  | 德累斯顿 L4秒 | |  |  |  |  |  |
| 0x86 | 冯智 | |  | NICTA N1 (修订版) | | | | | |  |
| 0x04 | 冯智 | |  | 版本 4 (修订版) | | | | | |  |
|  |  |  |  |  | |  |  |  | |  |
|  |  |  |  | (28 = 60) | |  | ww | ee | |  |

* 00: 小端,
* 01: 大端。

1. = 00:32 位 API,

= 01:64 位 API。

请注意, 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。

通过内核接口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 API 标志。

系统描述

ProcessorInfo

s

|  |  |  |
| --- | --- | --- |
| s(4) | (12 = 44) | 处理器1(16) |

由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。

处理器

可用系统处理器的数量。

|  |  |  |  |
| --- | --- | --- | --- |
| pageinfo | (7) |  |  |
| page-size mask(22 = 54) | R W X |  |

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。

4

R W X

ThreadInfo

t

内核界面页

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。对于rwx = 010, 只能控制正交的写入权限。处理器将隐式允许在任何映射页上读取和执行访问。对于执行 = 111, 可以设置所有三权限并研究t 独立。

|  |  |  |
| --- | --- | --- |
| UserBase(12) | SystemBase(12) | t(8) |

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

UserBase

用户线程可用的最低线程数 (请参见页[14)](#page26).前三线程号将分别用于初始线程0,1和根任务(请参见页[92)](#page104). 初始线程的版本号将等于1。

SystemBase

用于系统线程的最低线程数 (请参见页[14)](#page26).此值下面的线程号表示硬件中断。

ClockInfo

|  |  |
| --- | --- |
| SchedulePrecision(16) | ReadPrecision(16) |

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)。精度为时间段 (参见页[30)](#page42).

UtcbInfo

s

m

a

KipAreaInfo

s

BootInfo

|  |  |  |  |
| --- | --- | --- | --- |
| ~(10 /42) | s(6) | a(6) | m(10) |

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.

UTCB 大小乘数。

UTCB 位置必须与2a对齐。一个 UTCB 所需的总大小是2am

|  |  |
| --- | --- |
| (26 = 58) | s(6) |

内核接口页区域的大小为2s.

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)](#page104).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。

处理器说明

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 外部总线频率，单位kHz。

|  |  |
| --- | --- |
| 内核界面页 | 5 |

InternalFreq 内部处理器频率 (kHz)。

内核描述

KernDescPtr 指向包含4内核版本词 (如下所示) 的区域, 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)](#page114).零长度字符串 (即字符串

仅包含0个字符) 将终止功能说明的列表。

KernelDescPtr 的地址相对于内核接口页的基址。

KernelId

KernelGenDate

KernelVer

供应商

|  |  |  |
| --- | --- | --- |
| id(8) | subid8) | ~(16) |

可用于识别微内核。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| id | 了 |  | 内核 | |  | 供应商 | | | |
| 0 | 1 |  | L4/486 | |  | gmd | | | |
| 0 | 2 |  | L4/Pentium | |  | ibm | | | |
| 0 | 3 |  | L4/x86 | |  | 周 | | | |
| 1 | 1 |  | L4/Mips | |  | 新南 | | | |
| 2 | 1 |  | L4/Alpha | |  | TUD, 新南 | | | |
| 3 | 1 |  | 惨败 | |  | 可以 | | | |
| 4 | 1 |  | L4Ka::Hazelnut | |  | 周 | | | |
| 4 | 2 |  | L4Ka::P istachio | |  | 乌卡, 新南, NICTA | | | |
| 4 | 3 |  | L4Ka::Strawberry | |  | 周 | | | |
| 5 | 1 |  | NICTA::P istachio 嵌入式 | | | NICTA | | | |
|  |  |  | |  |  | |  |  |  |
|  | ~ | (16 = 48) | | | year-2000(7) | | 月(4) | 天(5) |  |
| 内核生成日期。 | | | | |  |  |  |  |  |
|  |  |  |  |  |  |  | | | |
|  | ver (8) |  |  | subver(8) |  | subsubver(16) | | |  |
|  |  |  |  |  |  |  |  |  |  |

可用于识别微内核的版本。请注意, 此内核版本不一定与 API 版本相关。

以"供应商" 字段的四最小有效字节指定标识内核供应商的字符串:

gmd" gmd

ibm" IBM 研究

新南" 新南威尔士大学, 悉尼

tud" 合作大学¨德累斯顿

周" 大学引用¨卡尔斯鲁厄 (TH)

编译" 澳大利亚国家信息和通信技术 (NICTA)

系统调用链接

sc

psc

用于正常系统调用的链接。

链接到特权的系统调用, 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。

6 内核界面页

内存描述

MemoryInfo

|  |  |
| --- | --- |
| MemDescPtr(16 = 32) | n(16 = 32) |

MemDescP tr

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理论描述符, 后面的描述符优先于较早的的.

* 内存描述符的数目。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| MemoryDesc | high=210 |  |  |  | ~(10) | | +4/+8 |  |
|  | (22 = 54) |  |  |  |
|  |  |  |  |  |  |  | +0 |  |
|  | low=210 | (22 = 54) | v | ~ | t(4) | 类型(4) |  |

高

低

v

类型

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。

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。

指示内存描述符是指物理内存 (v = 0) 还是虚拟内存 (v = 1)。

标识内存描述符的类型。

|  |  |
| --- | --- |
| 类型 | 描述 |
| 0x0 | 定义 |
| 0x1 | 常规存储器 |
| 0x2 | 保留内存 (即由内核保留) |
| 0x3 | 专用内存 (即, 用户无法使用的内存) |
| 0x4 | 共享内存 (即, 可供所有用户使用) |
| 0xe | 由引导加载程序定义 |
| 0xF | 依赖体系结构 |

t, 类型 = 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, 类型 = 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 (请参阅 "页面")。

t, 键入6 =0xE、型号6 =0xF

内存描述符的类型完全由类型字段定义。t 字段的内容未定义。

|  |  |
| --- | --- |
| KERNELINTERFACE | 7 |

**1。2 KERNELINterface** **[慢 Systemcall]**

|  |  |  |  |
| --- | --- | --- | --- |
| ! | void\* | Kernel interface page |  |
| Word | API Version |  |
|  | Word | API Flags |  |
|  | Word | KernelId |  |

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。

页面上描述了内核界面页面的结构[2。](#page14)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。它不是由调页程序映射的, 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。

任何线程都可以确定内核的地址。通过此系统调用接口页。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的内核接口页上使用唯一的地址。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。

自独立于 API 和 ABI 启用版本检测, 则保证当前的系统调用在所有 L4 版本中都能正常工作。systemcall 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。

**输出参数**

内核界面页

1及以上

基址(32 = 64)

内核界面页面地址, 总是页面对齐。0不有效地址。

|  |  |  |
| --- | --- | --- |
| 版本0及以下 | 0(32 = 64) |  |
|  |  |

较旧的版本 (2、x 0 等) 不包括内核接口页作为内核映射页。

未发送地址。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| API 版本 | 版本(8) | 颠覆(8) | (16) | | |  |  |
|  |  |  |
|  | 请参阅页面[3,](#page15)"内核界面" 页 | |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| API 标志 |  |  |  |  |  |  |  |
|  | (28 = 60) |  | ww | ee |  |  |
|  |  |  |  |  |

请参阅页面[3,](#page15)"内核界面" 页

8 KERNELINTERFACE

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| KernelId | id(8) | 了8) | (16) |  |
|  |  |

请参阅页面[5,](#page17)"内核界面" 页

**Pagefaults**

没有 pagefaults 会发生。

**泛型编程接口**

**系统调用函数:**

#include<l4/基普>

无效 \* KernelInterface 词&ApiVersion, ApiFlags, KernelId)

**方便的编程接口**

**派生函数:**

#include<l4/基普>

结构**M埃默里Desc**f 字原始 [2] g

结构**P民国Desc**f 字原始 [4] g

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 无效\* | KernelInterface () | | | [GetKernelInterface] |
|  |  | 传递指针到内核界面页。 | | |
| 单词 ApiVersion () | | |  |  |
| 字 ApiFlags | | () |  |  |
| 字 KernelId | | () |  |  |
| 无效 | KernelGenDate (KernelInterface, 词和年, 月, 日) | | | |
| 词 | KernelVersion | | (空 \* KernelInterface) | |
| 词 | KernelSupplier | | (空 \* KernelInterface) | |
|  |  | 提供 api 版本/api 标志/内核 Id/内核生成日期/内核版本/内核。 | | |
|  |  | 倍. | |  |
| 词 | NumProcessors | | (空 \* KernelInterface) | |
| 词 | NumMemoryDescriptors (空 \* KernelInterface) | | | |
|  |  | 提供系统/编号中的处理器数量内核中的内存描述符 | | |
|  |  | 界面页。 | | |
| 词 | PageSizeMask | | (空 \* KernelInterface) | |
| 词 | PageRights (空 \* KernelInterface) | | | |
|  |  | 为当前的内核/硬件体系结构提供支持的页面大小/页面权限。 | | |
| 词 | ThreadIdBits(空 \* KernelInterface) | | | |
| 词 | ThreadIdSystemBase | | | (空 \* KernelInterface) |

|  |  |  |
| --- | --- | --- |
| KERNELINTERFACE | | 9 |
| 词 | ThreadIdUserBase | (空 \* KernelInterface) |
|  | 为系统提供线程编号/最低线程编号的有效位数线程/最低 | |
|  | 用户线程的线程编号。 | |
| 词 | ReadPrecision (空 \* KernelInterface) | |
| 词 | SchedulePrecision | (空 \* KernelInterface) |
|  | 提供 S系统C锁读取 wakeups (s) 的精度/最大抖动。 | |
| 词 | UtcbAreaSizeLog2 | 无效\*KernelInterface) |
| 词 | UtcbAlignmentLog2 (空 \* KernelInterface) | |
| 词 | UtcbSize (空 \* KernelInterface) | |
|  | 提供所需的最小尺寸的 UTCB 面积/对齐要求 UTCBs/大小的罪孽- | |
|  | 看 UTCB | |
| 词 | KipAreaSizeLog2 | (空 \* KernelInterface) |
|  | 提供内核接口页面区域的大小。 | |
| 词 | BootInfo (空 \* KernelInterface) | |
|  | 提供 "引导信息" 字段的内容。 | |
| char\* | KernelVersionString (空 \* KernelInterface) | |
|  | 提供内核版本字符串。 | |
| char\* | 功能 (void \*KernelInterface, 字 num) | |
|  | 提供 numth 内核功能字符串, 如果 num 超过了利用的次数, 则为 null 指针 | |
|  | 功能字符串。 | |

MemoryDesc \* MemoryDesc (空 \* KernelInterface, 字 num)

提供 numth 内存描述符, 或者为 null如果 num 超过可用描述符的数量, 则为指针。

ProcDesc \* ProcDesc (空 \* KernelInterface, 字 num)

提供 numth 处理器描述符, 如果 num 超过系统的 pro-cessors 数, 则为 null 指针 (请参见 ProcessorInfo)。

**S支持功能:**

#include<l4/基普>

字 UndefinedMemoryType

字 ConventionalMemoryType

字 ReservedMemoryType

字 DedicatedMemoryType

字 SharedMemoryType

字 BootLoaderSpecificMemoryType

字 ArchitectureSpecificMemoryType

|  |  |  |
| --- | --- | --- |
| Bool IsVirtual(MemoryDesc & m) | | [IsMemoryDescVirtual] |
|  | 如果内存描述符指定虚拟内存区域, 则传递 true。 |  |
| 文字类型 | (MemoryDesc & m) | [MemoryDescType] |
| 低字 | (MemoryDesc & m) | [MemoryDescLow] |
| 字高 | (MemoryDesc & m) | [MemoryDescHigh] |

提供类型 (t 16 + 类型)、低限制和高内存区域限制。

10

KERNELINTERFACE

字 ExternalFreq (ProcDesc 和 p)

[ProcDescExternalFreq]

字 InternalFreq (ProcDesc 和 p)

[ProcDescInternalFreq]

提供处理器的外部频率/内部频率。

|  |  |
| --- | --- |
| 虚拟寄存器 | 11 |

**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。混合, 一些虚拟寄存器的硬件寄存器, 有些到内存也可以。虚拟寄存器访问的 ABI 取决于特定的处理器类型和 virtual-注册类型, 见附录[1,](#page108)??和[C. 1](#page150)具体的硬件详细信息。

有三类虚拟寄存器:

线程控制寄存器 (TCRs), 请参阅页面[16](#page28)

消息寄存器 (MRs), 请参阅页[50](#page62)

缓冲区寄存器 (BRs), 请参阅页[62](#page74)

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 (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) 是非法的, 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。

|  |  |  |
| --- | --- | --- |
| 话: | 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 提供任何间接地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。 |  |

**泛型编程接口**

#include<l4/留言>

|  |  |  |
| --- | --- | --- |
| 无效 | StoreMR | (int i, 词和 w) |
| 无效 | LoadMR | (int i, 字 w) |
|  |  | 提供/设置 MRi. |
| 无效 | StoreMRs | (int i; k, 单词和 [k] W) |
| 无效 | LoadMRs | (int i; k, 单词和 [k] W) |
|  |  | 商店/装载先生i:: 我 + k1内存。 |
| 无效 | StoreBR | (int i, 词和 w) |
| 无效 | LoadBR | (int i, 字 w) |
|  |  | 提供/设置 BR 的值i. |
| 无效 | StoreBRs | (int i, k, 词和 [k]) |
| 无效 | LoadBRs | (int i, k, 词和 [k]) |
|  |  | 商店或装载 BRi:: 我 + k1内存。 |
|  |  |  |

12 虚拟寄存器

**2章**

线程

14 threadid

**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 是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;线程的本地 ID 的范围仅作用于线程自己的地址空间。在不同的地址 spaces, 相同的本地线程 ID 可以识别不同的和不相关的线程。

请注意, 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 由一个字节组成,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 的值在内核中发布接口页 (参见页[4)](#page16).

|  |  |  |  |
| --- | --- | --- | --- |
| 全局线程 ID | 线程 no(18 = 32) | 版本(14 = 32) 6 = 0 (mod 64) |  |
|  |  |
| 全局中断 ID |  |  |  |
| intr no (18 = 32) | 1(14 = 32) |  |
|  |  |

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, 版本的最低6位不能全部是 0, 即 v mod 646 =0必须保持每版本五。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。操作系统的个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。它们由6最低位为0。

|  |  |  |  |
| --- | --- | --- | --- |
| 本地线程 ID | 本地 id/64(26 = 58) | 0 0 0 0 0 0 |  |
|  |  |

**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。ID *anylocalthread* 匹配驻留在相同地址空间的所有线程。

|  |  |  |
| --- | --- | --- |
| nilthread | 0(32 = 64) |  |
|  |  |

anythread

-1(32 = 64)

anylocalthread

|  |  |
| --- | --- |
| -1(26 = 58) | 0 0 0 0 0 0 |
|  |  |

|  |  |
| --- | --- |
| threadid | 15 |

**泛型编程接口**

#include<l4/螺纹 h>

结构**THREADID**f 字原始 g

ThreadId nilthread

ThreadId anythread

ThreadId anylocalthread

ThreadId GlobalId (Word threadno, 版本)

提供指示线程和版本号的线程 ID。

Word 版本 (ThreadId t)

字 ThreadNo (ThreadId t)

提供版本/线程指示的全局线程 ID 数。

**方便的编程接口**

#include<l4/螺纹 h>

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| bool | == (ThreadId 升, R) | | | | [IsThreadEqual] |
| Bool | ! = | (ThreadId l, r) | | | [IsThreadNotEqual] |
|  |  |  | 检查线程 id 是否匹配或不同。将本地 id 与全局 id 进行比较的结果总是表示不匹配, 即使 IDs 引用同一线程也是如此。 | | |
| Bool | SameThreads (ThreadId l, r) | | | |  |
|  |  |  | f GlobalId (l) == GlobalId (r) g | |  |
|  |  |  | Check if thread IDs refer to the same thread. Also works if one ID is local and the other is | | |
|  |  |  | global. | |  |
| Bool | IsNilThread | | (ThreadId t) | |  |
|  |  |  | { t == nilthread } | |  |
| Bool | IsLocalId (ThreadId t) | | | |  |
| Bool | IsGlobalId | | (ThreadId t) | |  |
|  |  |  | Check if thread ID is a local/global one. | |  |
| ThreadId | | LocalId | (ThreadId t) | | [LocalIdOf] |
| ThreadId | | GlobalId (ThreadId t) | | | [GlobalIdOf] |
|  |  |  | Delivers the local/global ID of the specified local thread. Specifying a non-local thread delivers | | |
|  |  |  | nilthread (see EXCHANGEREGISTERS, page [18)](#page30). | |  |
| ThreadId MyLocalId | | | | () |  |
| ThreadId MyGlobalId | | | | () |  |
|  |  |  | Delivers the local/global ID of the currently running thread (see TCRs, page [16)](#page28). | | |
| ThreadId Myself | | | () |  |  |
|  |  |  | f MyGlobalId () g | |  |

16 THREAD CONTROL REGISTERS (TCRS)

**2.2 Thread Control Registers (TCRs)** **[Virtual Registers]**

TCRs are a fast mechanism to exchange relatively static control information between user thread and microkernel. TCRs are static non-transient per-thread registers.

TCR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。 TCR是每个线程的静态非暂态寄存器。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| VirtualSender/ActualSender (32/64) | | | R=W | see IPC |  |
|  |  |  |  | see IPC |  |
|  |  |  |  |  |
| IntendedReceiver (32/64)目标接收器 | | | R-only |  |
|  |  |  |  | see IPC |  |
|  |  |  |  |  |
| XferTimeouts (32/64) | | | R=W |  |
|  |  |  |  | see system-calls |  |
|  |  |  |  |  |
| ErrorCode (32/64) | | | R-only |  |
|  |  |  |  | see Scheduling |  |
|  |  |  |  |  |
|  |  | Preempt Flags (8) | R=W |  |
|  |  |  |  | see Miscellaneous |  |
|  |  |  |  |  |
|  |  | Cop Flags (8) | W -only |  |
|  | |  |  | see Miscellaneous |  |
|  | | |  |  |
| ExceptionHandler (32/64) | | | R=W |  |
|  | | |  | see Protocols |  |
|  | | |  |  |
| Pager (32/64) | | | R=W |  |
|  | | |  | see Threads |  |
| UserDefinedHandle (32/64) | | | R=W |  |
|  | | |  | see Miscellaneous |  |
|  | | |  |  |
| ProcessorNo (32/64) | | | R-only |  |
|  | | |  | see Threads, IPC |  |
|  | | |  |  |
| MyLocalId (32/64) | | | R-only |  |
|  | | |  | see Threads, IPC |  |
|  | | |  |  |
| MyGlobalId (32/64) | | | R-only |  |
|  |  |  |  |  |  |

MyGlobalId Global ID of the thread.

MyLocalId Local ID of the thread.

ProcessorNo 线程当前执行的处理器号.

UserDefinedHandle

This field can be freely set and read by user threads. It can, e.g., be used for storing a thread number, a pointer to an additional user thread control block, etc.

|  |  |
| --- | --- |
| THREAD CONTROL REGISTERS (TCRS) | 17 |

**Generic Programming Interface**

The listed generic functions permit user code to access TCRs independently of the processor-specific TCR model. All functions are user-level functions; the microkernel is not involved.

列出的通用功能允许用户代码访问TCR，而不依赖于特定于处理器的TCR模型。 所有功能都是用户级别的功能; 微内核不涉及

#include <l4/thread.h>

ThreadId MyLocalId ()

ThreadId MyGlobalId ()

Delivers the local/global ID of the currently running thread (see TCRs, page [16)](#page28).

ThreadId Myself ()

{ MyGlobalId ()}

Word ProcessorNo ()

Delivers the processor number the current thread is running on. Delivered value is a valid index into the processor description array (see Kernel Interface Page, page [4)](#page16).

Word UserDefinedHandle ()

void Set UserDefinedHandle (Word NewValue)

Delivers/sets the user defined handle of the currently running thread.

ThreadId Pager ()

void Set Pager (ThreadId NewPager)

Delivers/sets the pager for the currently running thread.

ThreadId ExceptionHandler ()

void Set ExceptionHandler (ThreadId NewHandler)

Delivers/sets the exception handler for the currently running thread.

void Set CopFlag (Word n)

void Clr CopFlag (Word n)

Sets/clears coprocessor flag cn.

Word ErrorCode ()

Delivers the error code of the last system-call.

Word XferTimeouts ()

void Set XferTimeouts (Word NewValue)

Delivers/sets the transfer timeouts for the currently running thread (see IPC, page [66)](#page78).

ThreadId IntendedReceiver ()

Delivers the intended receiver of last received IPC (see IPC, page [67)](#page79).

ThreadId ActualSender ()

Delivers the actual sender of the last propagated IPC (see IPC, page [66)](#page78).

void Set VirtualSender (ThreadId t)

Sets the virtual sender for the next deceiving IPC (see IPC, page [66)](#page78).

Code generators of IDL and other compilers are not restricted to the generic interface. They can use any processor-specific methods and optimizations to access TCRs.

18 EXCHANGEREGISTERS

**2.3 EXCHANGEREGISTERS** **[Systemcall]**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ThreadId | dest | ! | ThreadId | result |  |
| Word | control | Word | control |  |
| Word | SP |  | Word | SP |  |
| Word | IP |  | Word | IP |  |
| Word | FLAGS |  | Word | FLAGS |  |
| ThreadId | pager |  | ThreadId | pager |  |
| Word | UserDefinedHandle | | Word | UserDefinedHandle |  |

Exchanges or reads a thread’s FLAGS, SP, and IP hardware registers as well as pager and UserDefinedHandle TCRs. Furthermore, thread execution can be suspended or resumed. The destination thread must be an active thread (see page [23)](#page35) residing in the invoker’s address space.

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（参见第23页）。

Any IP, SP, or FLAGS modification changes the corresponding user-level registers of the addressed thread. In general, ongoing kernel activities are not influenced. However, a currently active IPC operation can be canceled or aborted. For details see the SR-bit specification below.

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。

Modifications of the pager TCR and the UserDefinedHandle TCR become immediately effective, whether the desti-nation thread executes in user mode or in kernel mode.

调页程序的TCR和UserDefinedHandle TCR的修改无论目标线程是以用户模式还是以内核模式执行，都会立即生效。

**Input Parameters**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| dest | Thread ID of the addressed thread. This may be a local or a global ID. However, the addressed | | | | | |  |
|  | thread must reside in the current address space. Using a local thread ID might be substantially | | | | | |  |
|  | faster in some implementations. 被寻址线程的线程标识。 这可能是本地或全球ID。 但是，所提到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。 | | |  |  |  |  |
|  |  |  |  |  |  |  |  |
| control |  |  |  |  |  |  |  |
|  | 0 (20=52) |  | W R C d h p u f i s S R H |  |  |  |
|  |  |  |  |  |  |
| CtrlXferItems MR i |  |  |  |  |  | MR cw0+:::+cwn+1 |  |
|  | Write CtrlXferItem nw | | |  |  |
|  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  | . |  |  | . | |  |
|  | . | |  |  | . | |  |
|  | . | |  |  | . | |  |
|  |  | Write CtrlXferItem 0 | |  |  | MR cw0+1 |  |
|  |  |  | |  |  | MR cr0+:::+crn+1 |  |
|  |  |  | |  |  |  |
|  |  | Read CtrlXferItem nr | |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  | . |  |  | . | |  |
|  | . | |  |  | . | |  |
|  | . | |  |  | . | |  |
|  |  | Read CtrlXferItem 0 | |  |  | MR c0+1 |  |
|  |  |  | | |  | MR nc |  |
|  |  |  | | |  |  |
|  |  | CtrlXferConfItem nc (32) | | |  |  |
|  |  |  |  |  |  |  |  |
|  |  | . |  |  | . | |  |
|  | . | |  |  | . | |  |
|  | . | |  |  | . | |  |
|  |  | CtrlXferConfItem 0 (32) | | |  | MR 0 |  |
|  |  |  |  |  |  |  |  |

|  |  |
| --- | --- |
| EXCHANGEREGISTERS | 19 |

W R C

C = 1

W，R和C标志是指基于扩展的控制转移项目的状态操作。 C标志配置内核为内核生成的消息使用扩展的控制传输协议（见第7.6节）。 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。 所有控制转移项目都被传递到调用线程的消息寄存器中。

调用者请求扩展状态在内核生成的消息上发送。 控制传输配置项CtrlXferConfItem（见

下面），每个故障指定扩展状态。 这些项目从MR0开始。

|  |  |  |  |
| --- | --- | --- | --- |
| idmask (20=52) | fault (8) | 1 1 0 C | MR i |

fault

内核消息生成的故障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。 标识符是架构特定的。 如果掩码中的位n被设置为1，则内核将把CtrlXferItem编号n附加到消息上。 请注意，内核不允许选择特定CtrlXferItem的单个寄存器是不允许的; 而是总是包含特定项目的全部内容。

C

R = 1

* = 1

连续标志c除了最后一个CtrlXferConfItem外都被设置。

The continuation flag c is set for all but the last CtrlXferConfItem .

调用者请求扩展状态从目标线程读取。 扩展状态在调用者的消息寄存器中传递，作为一组控制传输项目。 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MR 0开始。tye将继续标志c设置为除最后一个CtrlXferConfItem之外的所有标志。 请注意，每次读取CtrlXferConfItem必须提供足够的空间来覆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k个寄存器，则该项目必须包含用于k个寄存器的字）。

The caller requests extended state to be written from the destination thread. Extended state is passed in the caller’s message registers, as a set of control transfer items. If R = 1, the first write CtrlXferItem follows directly after the last read item. Else, if C = 1, the first write CtrlXferItem follows directly after the last CtrlXferConfItem . Else, C = 0, write starts at MR 0. The continuation flag c is set for all but the last CtrlXferConfItem .

h p u f i s

S R

S = 0

S = 1

R = 0

R = 1

The s-flag refers to the SP register, i to IP, f to FLAGS, u to the *UserDefinedHandle* TCR, p to the *pager* TCR, and h to the H-flag. If a flag is set to 1, the register/state is overwritten by the corresponding input parameter. Otherwise, the corresponding input parameter is ignored and the register/state is not modified.

Controls whether the addressed thread’s ongoing IPC opereration should be canceled/aborted through the system call or not. 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。

An IPC operation of the addressed thread that is currently waiting to send a message or is sending a message will continue as usual. SP, IP or FLAGS modifications are delayed until the IPC operation terminates.

An IPC operation of the addressed thread that is currently waiting to send a message will be canceled. An IPC operation that is currently sending a message will be aborted.

An IPC operation of the addressed thread that is currently waiting to receive a message or is receiving a message will continue as usual. SP, IP or FLAGS modifications are delayed until the IPC operation terminates.

An IPC operation of the addressed thread that is currently waiting to receive a message will be canceled. An IPC operation that is currently receiving a message will be aborted.

H Halts/resumes the thread if h = 1. Ignored for h = 0. 如果h = 1，则暂停/恢复线程。忽略h = 0。

H = 0 No effect if the thread was not halted. Otherwise, thread execution is resumed.

|  |  |  |  |
| --- | --- | --- | --- |
| 20 |  | EXCHANGEREGISTERS |  |
|  | H = 1 | User-level thread execution is halted. Note that ongoing IPCs and other kernel operations are |  |
|  | not affected by H. (See SR for also aborting active IPC.) 用户级线程执行被暂停。 请注意，正在进行的IPC和其他内核操作  不受H的影响（参见SR也放弃活动的IPC）。 |  |
|  |  |  |
| d |  | If d = 1 the result parameters (IP, SP, FLAGS, UserDefinedHandle, pager, control) are delivered. |  |
|  | If d = 0 the return values are undefined. |  |
|  |  |  |
|  |  |  |  |
| SP |  | The current user-level stack pointer is set to SP if s = 1. Ignored for s = 0. 当s = 1时，当前用户级堆栈指针被设置为SP。s = 0时忽略。 |  |
|  |  |  |  |
| IP |  | The current user-level instruction pointer is set to IP if i = 1. Ignored for i = 0. |  |
|  |  |  |  |
| FLAGS |  | Sets the user-level processor flags of the thread if f = 1. Ignored for f = 0. The semantics of |  |
|  |  | the FLAGS word depends on the processor type. 如果f = 1，则设置线程的用户级处理器标志。f = 0时忽略。标志字的语义取决于处理器类型。 |  |
|  |  |  |  |

UserDefinedHandle

Sets the thread’s UserDefinedHandle TCR if u = 1. Ignored for u = 0.

pager Sets the thread’s pager TCR if p = 1. Ignored for p = 0.

**Output Parameters**

result 不= nilthread, input parameter dest was a local thread ID

global thread ID of the addressed thread. EXCHANGEREGISTERS succeeded.

result 不= nilthread, input parameter dest was a global thread ID

local thread ID of the addressed thread. EXCHANGEREGISTERS succeeded.

result = nilthread Operation failed. The ErrorCode TCR indicates the reason for the failure.

ErrorCode [TCR] Set if result = nilthread. Undefined if result 不= nilthread.

|  |  |  |
| --- | --- | --- |
| = 2 | Invalid thread. The dest parameter specified an invalid thread ID, an inactive thread, or a thread |  |
| within a different address space. 无效的线程。 dest参数指定了不同的地址空间内的无效线程ID，不活动线程或线程。 |  |
|  |  |

control

H

|  |  |
| --- | --- |
| 0 (29=61) | S R H |

The control parameter is only valid if d = 1 and undefined otherwise.

控制参数仅在d = 1时有效，否则为undefined。

Reports whether the addressed thread was halted (H = 1) or not (H = 0) when EXCHANGE-REGISTERS was invoked. Note that this output control bit is independent of the input parameter control.

|  |  |  |  |
| --- | --- | --- | --- |
| SR |  | Reports whether the addressed thread was within an IPC operation when EXCHANGEREGIS- |  |
|  | TERS was invoked. A value of 0 reports that the addressed thread was not within a send phase |  |
|  |  |  |
|  |  | (S = 0) or not within a receive phase (R = 0), respectively. Note that these output control bits |  |
|  |  | are independent of the input parameter control. |  |
|  | R = 1 | Operation was executed while the addressed thread was within the receive phase of an IPC |  |
|  | operation. Iff the input control word had R = 1 the IPC operation was canceled or aborted. |  |
|  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| EXCHANGEREGISTERS | | | 21 |  |
|  | S = 1 | Operation was executed while the addressed thread was within the send phase of an IPC opera- | |  |
|  | tion. Iff the input control word had S = 1 the IPC operation was canceled or aborted. |  |  |
|  |  |  |  |
|  |  |  |  |  |
| SP |  | Old user-level stack pointer of the thread, if d = 1 and undefined for d = 0. |  |  |
|  |  |  |  |  |
| IP |  | Old user-level instruction pointer of the thread, if d = 1 and undefined for d = 0. |  |  |
|  |  |  | |  |
| FLAGS |  | Old user-level flags of the thread, if d = 1 and undefined for d = 0. The semantics of this word | |  |
|  |  | is processor specific. |  |  |
|  |  |  |  |  |

UserDefinedHandle

Old content of thread’s UserDefinedHandle TCR, if d = 1 and undefined for d = 0.

pager Old content of thread’s pager TCR, if d = 1 and undefined for d = 0.

**Pagefaults**

No pagefaults will happen.

**Generic Programming Interface**

**System-Call Function:**

#include <l4/thread.h>

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)

**Convenience Programming Interface**

**Derived Functions:**

#include <l4/thread.h>

ThreadId

GlobalId

(ThreadId t)

{ if (IsLocalId (t)) ExchangeRegisters (t,0,–. . . ) else t }

[GlobalIdOf]

Delivers global ID of specified local thread. Specifying a non-local thread delivers nilthread.

ThreadId

LocalId

(ThreadId t)

{ if (IsGlobalId (t)) ExchangeRegisters (t,0,–. . . ) else t }

[LocalIdOf]

Delivers local ID of specified local thread. Specifying a non-local thread delivers nilthread.

Word

UserDefinedHandle

(ThreadId t)

[UserDefinedHandleOf]

void

Set UserDefinedHandle (ThreadId t, Word handle) [Set UserDefinedHandleOf]

Delivers/sets the user defined handle of specified local thread. Result of specifying a non-local thread is undefined.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 22 |  |  |  |  |  | EXCHANGEREGISTERS |
| ThreadId Pager | | | (ThreadId t) |  |  | [PagerOf] |
| void | Set Pager | | (ThreadId t, p) |  |  | [Set PagerOf] |
|  |  |  | Delivers/sets the pager for specified local thread. Result of specifying a non-local thread is | | | |
|  |  |  | undefined. |  |  |  |
| void | Start | (ThreadId t) | |  |  |  |
| void | Start | (ThreadId t, Word sp, ip) | | |  | [Start SpIp] |
| void | Start | (ThreadId t, Word sp, ip, flags) | | | | [Start SpIpFlags] |
|  |  |  | 继续执行指定的本地线程（如果暂停）。 中止任何正在进行的IPC操作。 根据功能参数可以选择修改堆栈指针，指令指针和处理器标志。 指定非本地线程的结果是未定义的。 | | | |
|  |  |  |  | | | |
|  |  |  |  | | | |
| ThreadState | | Stop (ThreadId t) | |  |  |  |
| ThreadState | | Stop (ThreadId t, Word& sp, ip, flags) | | | | [Stop SpIpFlags] |
|  |  |  | Halt execution of specified local thread and return its current thread state. Do not abort any on- | | | |
|  |  |  | going IPC operation. Optionally return（可选择返回） thread’s stack pointer, instruction pointer, and processor | | | |
|  |  |  | flags in output parameters. Result of specifying a non-local thread is undefined. | | | |
| ThreadState | | AbortReceive\_and\_ stop | | | (ThreadId t) |  |
| ThreadState | | AbortReceive\_ and\_stop | | | (ThreadId t, Word& sp, ip, flags) | [AbortReceive and stop SpIpFlags] |
|  |  |  | As stop (), except any ongoing IPC receive operation is immediately aborted. | | | |
| ThreadState | | AbortSend and stop | | (ThreadId t) | |  |
| ThreadState | | AbortSend and stop | | (ThreadId t, Word& sp, ip, flags) | | [AbortSend and stop SpIpFlags] |
|  |  |  | As stop (), except any ongoing IPC send operation is immediately aborted. | | | |
| ThreadState | | AbortIpc and stop | | (ThreadId t) | |  |
| ThreadState | | AbortIpc and stop | | (ThreadId t, Word& sp, ip, flags) | | [AbortIpc and stop SpIpFlags] |
|  |  |  | As stop (), except any ongoing IPC send or receive operations are immediately aborted. | | | |
|  |  |  |  |  |  |  |

**Support Functions:**

#include <l4/thread.h>

struct **THREADSTATE** f Word raw g

Bool ThreadWasHalted (ThreadState s)

Bool ThreadWasSending (ThreadState s)

Bool ThreadWasReceiving (ThreadState s)

Bool ThreadWasIpcing (ThreadState s)

Query the thread state returned from one of the stop () functions.

Word ErrorCode ()

Word ErrInvalidThread

|  |  |
| --- | --- |
| THREADCONTROL | 23 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **2.4 THREADCONTROL** | **[Privileged Systemcall]** | |  |  |
| ThreadId | dest | ! | Word result |  |
| ThreadId | SpaceSpecifier |  |  |
| ThreadId | scheduler |  |  |  |
| ThreadId | pager |  |  |  |
| void\* | UtcbLocation |  |  |  |

A privileged thread, e.g., the root server, can delete and create threads through this function. It can also modify the global thread ID (version field only) of an existing thread. 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）。

Threads can be created as active or inactive threads. Inactive threads do not execute but can be activated by active threads that execute in the same address space. 线程可以创建为活动或不活动的线程。非活动线程不执行，但可以由在相同地址空间中执行的活动线程激活。

An actively created thread starts immediately by executing a short receive operation from its pager. (An active thread must have a pager.) The activeted thread expects a start message (MsgTag and two untyped words) from its pager. Once it receives the start message, it takes the value of MR 1 as its new IP, the value of MR 2 as its new SP, and then starts execution at user level with the received IP and SP. The new thread will execute on the same processor where the activating ThreadControl was invoked. 主动创建的线程立即启动，通过执行其传呼机的短接收操作。 （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开始消息（MsgTag和两个无类型的单词）。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，然后在接收到的IP和SP的用户级开始执行。新的线程将在调用激活的ThreadControl的同一处理器上执行

Interrupt threads are treated as normal threads. They are active at system startup and can not be deleted or migrated into a different address space (i.e., SpaceSpecifier must be equal to the interrupt thread ID). When an interrupt occurs the interrupt thread sends an IPC to its pager and waits for an empty end-of-interrupt acknowledgment message (MR 0=0). Interrupt threads never raise pagefaults. To deactivate interrupt message delivery the pager is set to the interrupt thread’s own ID. 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，中断线程将IPC发送到其调页程序，并等待空的中断确认消息（MR 0 = 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。

|  |  |
| --- | --- |
|  | **Input Parameters** |
|  |  |
| dest | Addressed thread. Must be a global thread ID. Only the thread number is effectively used |
|  | to address the thread. If a thread with the specified thread number exists, its version bits are |
|  | overwritten by the version bits of dest id and any ongoing IPC operations are aborted. Otherwise, |
|  | the specified version bits are used for thread creations, i.e., a thread creation generates a thread |
|  | with ID dest. |
|  |  |

SpaceSpecifier 不= nilthread, dest not existing

Creation. The space specifier specifies in which address space the thread will reside. Since address space do not have own IDs, a thread ID is used as SpaceSpecifier. Its meaning is: the new thread should execute in the same address space as the thread SpaceSpecifier.

The first thread in a new address space is created with SpaceSpecifier = dest. This operation implicitly creates a new empty address space. Note that the new address space is created with an empty UTCB and KIP area. The space creation must therefore be completed by a SPACECON-TROL operation before the thread(s) can execute.

创建。 空格说明符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。 请注意，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SPACECON-TROL操作来完成空间创建。

SpaceSpecifier 不= nilthread, dest exists

Modification Only. The addressed thread dest is neither deleted nor created. Modifications can change the version bits of the thread ID, the associated scheduler, the pager, or the associated address space, i.e., migrate the thread to a new address space.

只有修改。 被寻址的线程dest既不被删除也不被创建。 修改可以改变线程ID，相关调度器，调页程序或相关地址空间的版本位，即将线程迁移到新的地址空间。

SpaceSpecifier = nilthread, dest exists

Deletion. The addressed thread dest is deleted. Deleting the last thread of an address space implicitly also deletes the address space.

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

scheduler 不= nilthread

Defines the scheduler thread that is permitted to schedule the addressed thread. Note that the scheduler thread must exist when the addressed thread starts executing.

定义允许调度编址线程的调度程序线程。 请注意，调度程序线程在寻址的线程开始执行时必须存在。

24 THREADCONTROL

scheduler = nilthread

The current scheduler association is not modified. This variant is illegal for a creating THREAD-CONTROL operation. 当前调度程序关联不被修改。 创建THREAD-CONTROL操作时，这种变体是非法的。

pager 不= nilthread The pager of dest is set to the specified thread. If dest was inactive before, it is activated.

pager = nilthread The current pager association is not modified.

If used with a creating THREADCONTROL operation, dest is created as an inactive thread.

UtcbLocation 不= -1 The start address of the UTCB of the thread is set to UtcbLocation. Upon thread activation the UTCB must fit entirely into the UTCB area of the configured address space, and must be prop-

erly aligned according to the UtcbInfo field of the kernel interface page. It is the application’s responsibility to ensure that UTCBs of multiple threads do not overlap. Changing the UtcbLo-cation of an already active thread is an illegal operation. Note that since a newly created space has an empty UTCB area, it is not possible to activate a thread in an address space which has not been properly configured with SPACECONTROL.

线程的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。 请注意，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 The UTCB location is not modified.

UtcbInfo [KernelInterfacePage Field]

Permits to calculate the appropriate page size of the UTCB area fpage and specifies the size and alignement of UTCBs. Note that the size restricts the total number of threads that can reside in an address space.

允许计算UTCB区域fpage的相应页面大小，并指定UTCB的大小和对齐。 请注意，大小限制了可以驻留在地址空间中的线程总数。

|  |  |  |  |
| --- | --- | --- | --- |
| ~(10=42) | s (6) | a (6) | m (10) |

|  |  |  |
| --- | --- | --- |
| s | The minimal area size for an address space’s UTCB area is 2s. The size of the UTCB area limits |  |
| the total number of threads k to 2amk 2s. |  |
|  |  |

* UTCB size multiplier.

a The UTCB location must be aligned to 2a. The total size required for one UTCB is 2am.

**Output Parameters**

|  |  |
| --- | --- |
| result | The result is 1 if the operation succeeded, otherwise the result is 0 and the ErrorCode TCR |
|  | indicates the failure reason.  如果操作成功，则结果为1，否则结果为0，并且ErrorCode TCR指示失败原因。 |
|  |  |

ErrorCode [TCR]

= 1

= 2

= 3

= 4

= 6

Set if result = 0. Undefined if result 不=0.

No privilege. Current thread does not have have privilege to perform the operation.

没有特权。 当前线程没有执行操作的权限。

Unavailable thread. The dest parameter specified a kernel thread or an unavailable interrupt thread.

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。

Invalid space. The SpaceSpecifier parameter specified an invalid thread ID, or activation of a thread in a not yet initialized space.

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。

Invalid scheduler. The scheduler paramter specified an invalid thread ID, or was set to nilthrad for a creating THREADCONTROL operation.

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。

Invalid UTCB location. UtcbLocation lies outside of UTCB area, or attempt to change the UtcbLocation for an already active thread.

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。

|  |  |
| --- | --- |
| THREADCONTROL | 25 |

* 8Out of memory. Kernel was not able to allocate the resources required to perform the operation.
* 内存不足。 内核无法分配执行操作所需的资源。

**Pagefaults**

No pagefaults will happen.

**Generic Programming Interface**

**System-Call Function:**

#include <l4/thread.h>

Word ThreadControl (ThreadId dest, SpaceSpecifier, Scheduler, Pager, void\* UtcbLocation)

**Convenience Programming Interface**

**Derived Functions:**

#include <l4/thread.h>

Word AssociateInterrupt (ThreadId InterruptThread, InterruptHandler)

{ ThreadControl (InterruptThread, InterruptThread, nilthread, InterruptHandler, -1)}

Associate a handler thread with the specified interrupt source.

将处理程序线程与指定的中断源相关联。

Word DeassociateInterrupt (ThreadId InterruptThread)

{ ThreadControl (InterruptThread, InterruptThread, nilthread, InterruptThread, -1)}

Remove association between the specified interrupt source and any potential handler thread.

删除指定的中断源和任何潜在的处理程序线程之间的关联。

**Support Functions:**

Word ErrorCode ()

Word ErrNoPrivilege

Word ErrInvalidThread

Word ErrInvalidSpace

Word ErrInvalidScheduler

Word ErrUtcbArea

Word ErrNoMem

26 THREADCONTROL

**Chapter 3**

Scheduling

28 CLOCK

**3.1 Clock** **[Data Type]**

On both 32-bit and 64-bit processors, the system clock is represented as a 64-bit unsigned counter. The clock measures time in 1 s units, independent of the processor frequency. Although the clock base is undefined, it is guaranteed that the counter will not overflow for at least 1,000 years.

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。 虽然时钟基址是未定义的，但保证计数器不会溢出至少1000年。

**Generic Programming Interface**

#include <l4/schedule.h>

struct **CLOCK** { Word64 raw }

|  |  |  |
| --- | --- | --- |
|  | **Convenience Programming Interface** |  |
| #include <l4/schedule.h> | |  |
| Clock | + (Clock l, r) | [ClockAdd] |
| Clock + (Clock l, Word64 r) | | [ClockAddUsec] |
| Clock | + (Clock l, int r) |  |
| Clock | * (Clock l, r) | [ClockSub] |
| Clock - (Clock l, Word64 r) | | [ClockSubUsec] |
| Clock | * (Clock l, int r) |  |
|  | Adds/subtracts a number of s to/from a clock value. Delivers new clock value. Does not | |
|  | modify the old clock value. 在时钟值上添加/减去一定数量的s。 提供新的时钟值。 不要修改旧的时钟值。 |  |
| Bool | < (Clock l, r) | [IsClockEarlier] |
| Bool | > (Clock l, r) | [IsClockLater] |
| Bool | <= (Clock l, r) |  |
| Bool | >= (Clock l, r) |  |
| Bool | == (Clock l, r) | [IsClockEqual] |
| Bool | ! = (Clock l, r) | [IsClockNotEqual] |
|  | Compares two clock values. |  |
|  |  |  |

|  |  |
| --- | --- |
| SYSTEMCLOCK | 29 |

**3.2 SYSTEMCLOCK** **[Systemcall]**

! Clock clock

Delivers the current system clock. Typically, the operation does not enter kernel mode.

**Pagefaults**

No pagefaults will happen.

**Generic Programming Interface**

**System-Call Function:**

#include <l4/schedule.h>

Clock SystemClock ()

30 TIME

**3.3 Time** **[Data Type]**

Time values are used to specify send/receive timeouts for IPC operations (see page [65)](#page77) and time quanta for scheduling (see page [33)](#page45). The unit for time periods as well as for time points is 1 s. Clock ticks thus happen every s.

Relative time values specify a time period. Time periods are encoded as un-normalized 16-bit floating-point numbers. (Note that for easier handling the mantissa can have leading 0-bits.) The shortest non-zero time period that can be specified is 1 s, the longest finite period slightly exceeds 610 hours. Two special periods frequently used for timeouts are 0 and 1, a never ending period. The values 0 and 1 have special encodings.

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）。 时间单位和时间单位是1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，尾数可以具有前0位）。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1，这是一个永不停歇的时期。 值0和1有特殊的编码。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| time period: | 0 | e (5) | m (10) | = | 2em s |  |
|  |  |
|  |  |  |  |  | |  |
|  |  |  | 0 (16) | = 1 | |  |
|  | |  |  |  |  |  |
|  | 0 | 1 (5) | 0 (10) | = | 0 |  |

Absolute time values specify a point in time. They are only valid for a limited period, at maximum 67 seconds.

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。

|  |  |
| --- | --- |
| time point: |  |
| 1 e (4) c | m (10) |

For a semantical description of time-point values, we use Clock to denote the current clock value in s, x[i] to denote

bit i of x, and x[i;j] to denote the number consisting of bits i to j of x. Then, the time-point value (c; m; e) specifies the point

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微s，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

:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 8 | 2 | e |  | m + Clock[63;e+9] |  | 2 | 10 |  |  |  | if | Clock[e+10] | = c |  |
| t = | e | 10 | 10 |  |  |
|  | : | 2 |  |  |  |  | 2 |  | + 2 |  | if | Clock[e+10] | 6 |  |
|  | < |  |  | m + Clock[63;e+9] |  |  |  |  | = c |  |

Absolute time values are thus the more precise the nearer in the future they are.

因此绝对时间值在未来越近越精确。

Absolute time values with maximal precision become invalid just after the clock has reached the specified point in time. The validity interval can be expanded, but only by reducing the precision. In general, a time-point value (c; m; e) that is constructed when the current clock value is C0 is valid from C0 up to

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，但只能降低精度。 通常，当前时钟值为C0时构成的时间点值（c; m; e）从C0到

C0 + (210 1) 2e

Therefore, a time-point value that should remain valid for 10 ms can have a precision of 10 s whereas a value that should remain valid for an entire second can only have a precision of 1 ms. In general, a precision of 0.1% of the required validity interval can be achieved.

因此，应该保持有效10毫秒的时间点值可以具有10微秒的精度，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，可以达到所需有效期间的0.1％的精度。

**Generic Programming Interface**

#include <l4/schedule.h>

struct **TIME** f Word16 raw g

Time Never

Time ZeroTime

Time TimePeriod (Word64 microseconds)

|  |  |
| --- | --- |
| TIME | 31 |

Time TimePoint (Clock at)

|  |  |  |
| --- | --- | --- |
|  | **Convenience Programming Interface** |  |
| #include <l4/schedule.h> | |  |
| Time + (Time l, Word r) | | [TimeAddUsec] |
| Time + = (Time l, Word r) | | [TimeAddUsecTo] |
| Time(Time l, Word r) | | [TimeSubUsec] |
| Time = (Time l, Word r) | | [TimeSubUsecFrom] |
|  | Adds/subtracts a number of microseconds to/from a time value. |  |
| Time + (Time l, r) | | [TimeAdd] |
| Time + = (Time l, r) | | [TimeAddTo] |
| Time(Time l, r) | | [TimeSub] |
| Time = (Time l, r) | | [TimeSubFrom] |
|  | Adds/subtracts a time period to/from a time value. The result of adding/subtracting a time point | |
|  | is undefined. |  |
| Bool | > (Time l, r) | [IsTimeLonger] |
| Bool | >= (Time l, r) |  |
| Bool | < (Time l, r) | [IsTimeShorter] |
| Bool | <= (Time l, r) |  |
| Bool | == (Time l, r) | [IsTimeEqual] |
| Bool | ! = (Time l, r) | [IsTimeNotEqual] |
|  | Compares two time values. The result of comparing a time period with a time point, or vice | |
|  | 反之, 则是未定义的。 |  |
|  |  |  |

32 THREADSWITCH

**3.4 THREADSWITCH** **[Systemcall]**

ThreadId dest ! void

The invoking thread releases the processor (non-preemptively) so that another ready thread can be processed.

调用线程释放处理器（非预占），以便可以处理另一个就绪线程。

**Input Parameter**

dest = nilthread

dest 不= nilthread

Processing switches to an undefined ready thread which is selected by the scheduler. (It might be the invoking thread.) Since this is “ordinary” scheduling, the thread gets a new timeslice.

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

If dest is ready, processing switches to this thread. In this “extraordinary” scheduling, the invok-ing thread donates its remaining timeslice to the destination thread. (This one gets the donation in addition to its ordinarily scheduled timeslices, if any.)

If the destination thread is not ready or resides on a different processor, the system call operates as described for dest = nilthread.

如果dest准备就绪，则处理切换到此线程。 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，除了通常的计划时间外，这个人还可以获得捐赠）  
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。

**Pagefaults**

No pagefaults will happen.

**Generic Programming Interface**

**System-Call Function:**

#include <l4/schedule.h>

void ThreadSwitch (ThreadId dest)

**Convenience Programming Interface**

**Derived Functions:**

#include <l4/schedule.h>

void Yield ()

{ ThreadSwitch (nilthread)}

Switch processing to a thread selected by the scheduler.

|  |  |
| --- | --- |
| SCHEDULE | 33 |

**3.5 SCHEDULE** **[Systemcall]**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ThreadId | dest | ! | Word | result |  |
| Word | time control | Word | time control |  |
| Word | processor control |  |  |  |  |
| Word | prio |  |  |  |  |
| Word | preemption control |  |  |  |  |

The system call can be used by schedulers to define the priority, timeslice length, and other scheduling parameters of threads. Furthermore, it delivers thread states.

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。

The system call is only effective if the calling thread resides in the same address space as the destination thread’s scheduler (see thread control, page [23)](#page35).

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（请参见线程控制，第23页）。

**Input Parameters**

dest Destination thread ID. The destination thread must be existent (but can be inactive) and the

current thread must reside in the same address space as the destination thread’s scheduler (see thread control). Otherwise, the destination thread is not affected.

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。

All further input parameters have no effect if the supplied value is -1, ensuring that the corresponding internal thread variable is not modified. The following description always refers to values 不= 1. 如果提供的值为1，则所有其他输入参数都不起作用，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。

time control

ts len

total quantum

|  |  |
| --- | --- |
| ts len (16) | total quantum (16) |
|  |  |

New timeslice length for the destination thread. The timeslice length is specified as a time period (see page [30)](#page42). Absolute time values and the value 0 are illegal. A timeslice length of 1, however, can be specified. In that case, the thread never experiences a preemption due to exhausted time slice. The specified value is always rounded up to the nearest possible timeslice length. In particular, a time period of 1 s results in the shortest possible timeslice.

Writing the timeslice length initializes the current quantum with the new length. After the quan-tum is exhausted, the thread is preempted while the quantum is reloaded with ts len for the next timeslice.

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（请参见第30页）。 绝对时间值和值0是非法的。 然而，可以指定时间片长度为1。 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秒的时间段将导致尽可能短的时间片。

写入时间片长度将使用新长度初始化当前量程。 在量子耗尽之后，线程被抢占，而量子在下一个时间片被重新加载ts len。

Defines the total quantum for the thread. Exhaustion of the total quantum results in an RPC to the thread’s scheduler (i.e., the current thread). (Re)writing the total quantum re-initializes the quantum, independent of the already consumed total quantum. The total quantum is specified as a time period (see page [30)](#page42). Absolute time values are illegal. A total quantum of 1 can be specified.

定义线程的总量子。 总量的耗尽导致RPC到线程的调度器（即当前线程）。 （Re）写总量子重新初始化量子，独立于已经消耗的总量子。 总量程被指定为一个时间段（请参阅第30页）。 绝对时间值是非法的。 总量子1可以被指定。

prio

|  |  |
| --- | --- |
| 0 (24=56) | prio (8) |

New priority for destination thread. Must be less than or equal to current thread’s priority.

preemption control

|  |  |  |
| --- | --- | --- |
| 0 (8=40) | sensitive prio (8) | maximum delay (16) |

|  |  |
| --- | --- |
| 34 | SCHEDULE |
| sensitive prio | Preemptions by threads that run on a priority lower or equal to this sensitive prio will, (a) if |
|  | the delay-preemption flag is set, be delayed until the thread executes a thread switch (nilthread) |
|  | system call; and (b) if the signal-preemption flag is set, raise a preemption fault to the exception |
|  | handler. |
|  | No preemption delays or signaling will occur if preempted by a thread having a higher priority |
|  | than sensitive prio, regardless of the state of the delay-preemption and signal-preemption flags.  （a）如果延迟抢占标志被设置，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。  不管延迟抢占和信号抢占标志的状态如何，优先级高于敏感prio的线程都不会抢占延迟或信号。 |
| maximum delay | The maximum time in s a pending preemption can be delayed in the destination thread. The |
|  | value 0 effectively disables preemption delay.  在等待抢占中的最大时间可以在目标线程中被延迟。 值0有效地禁用了抢占延迟。 |
|  |  |

|  |  |  |
| --- | --- | --- |
| processor control | processor number (16) |  |
| 0 (16/48) |  |

processor number Specifies the processor number to which the thread should be migrated. The processor number must be valid, i.e., smaller than the total number of processors (see kernel interface page at page [3)](#page15). Otherwise, the parameter is ignored. The first processor number is denoted as 0.

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。

**Output Parameters**

result

tstate =

0

1

2

3

4

5

6

7

|  |  |
| --- | --- |
| ~(24=56) | tstate (8) |

Thread state:

Error. The operation failed completely. The ErrorCode TCR indicates the reason for the failure.

Dead. The thread is unable to execute or does not exist.

Inactive. The thread is inactive/stopped.

Running. The thread is ready to execute at user-level.

Pending send. A user-invoked IPC send operation currently waits for the destination (recipient) to become ready to receive. 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。

Sending. A user-invoked IPC send operation currently transfers an outgoing message. 用户调用的IPC发送操作当前正在传送一个传出消息。

Waiting to receive. A user-invoked IPC receive operation currently waits for an incoming mes-sage.

Receiving. A user-invoked IPC receive operation currently receives an incoming message.

ErrorCode [TCR]

= 1

= 2

= 5

Set if lower 8 bits of result = 0. Undefined if lower 8 bits of result 不=0.

No privilege. Current thread is not the scheduler of the destination thread.

The dest parameter specified an invalid thread ID.

Invalid parameter. The specified time-slice length, total quantum, priority, or processor number was invalid.

|  |  |  |  |
| --- | --- | --- | --- |
| time control | rem ts (16) | rem total (16) |  |
|  |  |

SCHEDULE

35

rem ts

rem total

Remainder of the current timeslice. 当前时间片的剩余部分。

Remaining total quantum of the thread. 线程的剩余总量。

**Pagefaults**

No pagefaults will happen.

**Generic Programming Interface**

**System-Call Function:**

#include <l4/schedule.h>

Word Schedule (ThreadId dest, Word TimeControl, ProcessorControl, prio, PreemptionControl, Word& old TimeControl)

**Convenience Programming Interface**

**Derived Functions:**

#include <l4/schedule.h>

Word Set Priority (ThreadId dest, Word prio)

f Schedule (dest, -1, -1, prio, -1) g

Word Set ProcessorNo (ThreadId dest, Word ProcessorNo)

f Schedule (dest, -1, ProcessorNo, -1, -1) g

Word Timeslice (ThreadId dest, Time & ts, Time & tq)

Delivers the remaining timeslice and total quantum of the given thread.

Word Set Timeslice (ThreadId dest, Time ts, Time tq)

f Schedule (dest, ts \* 216 + tq, -1, -1, -1) g

Word Set PreemptionDelay (ThreadId dest, Word sensitivePrio, Word maxDelay) f Schedule (dest, -1, -1, -1, SensitivePrio \* 216 + MaxDelay) g

**Support Functions:**

Word ErrorCode ()

Word ErrNoPrivilege

Word ErrInvalidThread

36 SCHEDULE

Word ErrInvalidParam

|  |  |
| --- | --- |
| PREEMPT FLAGS | 37 |

**3.6 Preempt Flags** **[TCR]**

The preemption flags TCR controls asynchronous preemptions (timeslice exhausted or activation of a higher-priority thread including device interrupts).

抢占标志TCR控制异步抢占（时间片已耗尽或激活包括设备中断在内的高优先级线程）。

Preempt Flags

s = 0

s = 1

d = 0

d = 1

* = 0
* = 1

I d s ~

The ds-flags are used to control the microkernel. User threads can set/reset them. The I-flag signals an event to the user. It is set by the microkernel and typically read/reset by the user.

ds-flags用于控制微内核。 用户线程可以设置/重置它们。 I标志向用户发信号通知事件。 它由微内核设置，通常由用户读取/重置。

Asynchronous preemptions are not signaled to the exception handler.

异步抢占不会通知异常处理程序。

Asynchronous preemptions are signaled as preemption faults to the exception handler. If d = 0 this happens immediately. Otherwise, it is delayed until the thread continues execution after the preemption.

异步抢占被作为抢占故障发送给异常处理程序。 如果d = 0，则立即发生。 否则，它将被延迟，直到线程在抢先后继续执行。

All asynchronous preemptions happen immediately. If they are signaled as preemption faults (s = 1), this happens after the preemption took place, i.e., when the thread gets reactivated. 所有异步抢占立即发生。 如果它们被指示为抢占故障（s = 1），则这发生在抢先发生之后，即线程被重新激活之后。

Asynchronous preemptions are delayed if the priority of the preemptor is lower or equal than the sensitive priority for the current thread. (The sensitive priority is set by the scheduler, see page [34.)](#page46) A delayed preemption does not interrupt the current thread immediately but is post-poned until the current thread invokes a systemcall thread switch (nilthread). However, a pend-ing preemption must not be delayed for longer than the maximum delay that was set by the thread’s scheduler. Such a preemption-delay overflow resets the d bit and is signaled to the exception handler.

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34页。）延迟抢占不会立即中断当前线程，而是在当前线程调用systemcall线程切换（nilthread）之前进行后置。 但是，抢占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。 这种抢占延迟溢出重置d位，并通知异常处理程序。

No asynchronous preemption is pending.

没有异步抢占正在等待。

An asynchronous preemption is currently pending, i.e., the thread should as soon as possible reset the d-flag and invoke thread switch. Invoking thread switch re-enables the maximum delay for the next delayed asynchronous preemption.

Invoking thread switch is not required if no asynchronous preemption is pending (I = 0) after the user thread has reset the d-flag.

异步抢占当前正在等待，即线程应尽快重置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。

**Generic Programming Interface**

#include <l4/schedule.h>

Bool EnablePreemptionFaultException ()

Bool DisablePreemptionFaultException ()

Sets/resets the s-flag and delivers the old s-flag value (true = set).

Bool DisablePreemption ()

Bool EnablePreemption ()

Sets/resets the d-flag and delivers the old d-flag value (true = set).

Bool PreemptionPending ()

Resets the I-flag and delivers the old I-flag value (true = set).

38 PREEMPT FLAGS

**Chapter 4**

Address Spaces and Mapping

40 FPAGE

**4.1 Fpage** **[Data Type]**

Fpages (Flexpages) are regions of the virtual address space. An fpage consists of all pages mapped actually in this region sans kernel mapped objects, i.e., kernel interface page and UTCBs. Fpages have a size of at least 1 K. For specific processors, the minimal fpage size may be larger; e.g., a Pentium processor offers a minimal page size of 4 K while the Alpha processor offers smallest pages of 8 K. Fpages smaller than the minimal page size are treated as nilpages. The kernel interface page (see page [3)](#page15) specifies which page sizes are supported by the hardware/kernel. An fpage of size 2s has a 2s-aligned base address b, i.e., b 0 (mod 2s), where s 10 for all architectures.

Mapped fpages are considered inseparable objects. That is, if an fpage is mapped, the mapper can not later partially unmap the mapped page; the whole fpage must be unmapped in a single operation. The mappee can, however, separate the fpage and map fpages (objects) of smaller size. Partially unmapping an fpage might or might not work on some systems. The kernel will give no indication as to whether such an operation succeeded or not.

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UTCB。 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零页面。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2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。

映射的页面被认为是不可分离的对象。也就是说，如果一个fpage被映射，映射器不能稍后部分取消映射的映射页面;整个fpage必须在一个操作中被取消映射。然而，mappee可以将fpage和map的尺寸（对象）分开。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。

|  |  |  |  |
| --- | --- | --- | --- |
| fpage (b; 2s) | s (6) |  |  |
| b=210 (22/54) | 0 r w x |  |

Special fpage denoters describe the complete user address space and the nilpage, an fpage which has no base address and a size of 0:

特殊的fpage denoters描述了完整的用户地址空间和nilpage，一个没有基地址，大小为0的fpage：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| complete |  | 0 (22/54) |  | s = 1 (6) | 0 r w x |  |
|  |  |  |  |
| nilpage |  |  |  |  |  |  |
|  | 0 (32/64) |  |  |  |  |
|  |  |  |  |  |  |
| **Access Rights** |  |  |  |  |  |  |
| rwx | The rwx bits define the accessibility of the fpage: | | | | |  |
|  |  |  |  |  |  |
|  |  | r | readable | | |  |
|  |  | w | writable | | |  |
|  |  | x | executable | | |  |

A bit set to one permits the corresponding access to the newly-mapped/granted page provided that the mapper itself possesses that access right. If the mapper does not have the access right itself or if the bit is set to zero the mapped/granted page will not get the corresponding access right.

Note that processor architectures may impose restrictions on the access-right combinations. However, read-only (including execute), rwx = 101, and read/write/execute, rwx = 111, should be valid for any processor architecture. The kernel interface page (see page [3)](#page15) specifies which access rights are supported in the processor architecture.

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。

请注意，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。

**Generic Programming Interface**

#include <l4/space.h>

struct **FPAGE** f Word raw g

Word Readable

Word Writable

|  |  |
| --- | --- |
| FPAGE | 41 |
| Word eXecutable |  |
| Word FullyAccessible |  |
| Word ReadeXecOnly |  |
| Word NoAccess |  |
| Fpage Nilpage |  |
| Fpage CompleteAddressSpace |  |
| Bool IsNilFpage (Fpage f) |  |
| f f == Nilpage g |  |
| Fpage Fpage (Word BaseAddress, int FpageSize 1K) |  |
| Fpage FpageLog2 (Word BaseAddress, int Log2FpageSize < 64) |  |
| Delivers an fpage with the specified location and size. |  |
| Word Address (Fpage f) |  |
| Word Size (Fpage f) |  |
| Word SizeLog2 (Fpage f) |  |
| Delivers address/size of specified fpage. |  |
| Word Rights (Fpage f) |  |
| void Set Rights (Fpage& f, Word AccessRights) |  |
| Delivers/sets the access rights for the specified fpage. |  |
| Fpage + (Fpage f, Word AccessRights) | [FpageAddRights] |
| Fpage + = (Fpage f, Word AccessRights) | [FpageAddRightsTo] |
| Fpage(Fpage f, Word AccessRights) | [FpageRemoveRights] |
| Fpage = (Fpage f, Word AccessRights) | [FpageRemoveRightsFrom] |

Adds/removes specified access rights from fpage. Delivers new fpage value.

42 UNMAP

**4.2 UNMAP** **[Systemcall]**

Word control ! void

The specified fpages (located in MR 0:::) are unmapped. Fpages are mapped as part of the IPC operation (see page [64)](#page76).

指定的fpages（位于MR 0 :: :)未映射。 Fpages被映射为IPC操作的一部分（请参阅第64页）。

**Input Parameters**

control

k

f = 0

f = 1

|  |  |  |
| --- | --- | --- |
| 0 (25=57) | f | k (6) |

Specifies the highest MR k that holds an fpage to be unmapped. The number of fpages is thus k + 1.

指定保存fpage将被取消映射的最高MRk。 因此，数量是k + 1。

The fpages are unmapped recursively in all address spaces in which threads of the current ad-dress space have mapped them before. However, the fpages remain unchanged in the current address space.

在当前广告位空间的线程已经映射它们的所有地址空间中，递归地解除映射fpages。 但是，在当前的地址空间中，这些部分保持不变。

The fpages are unmapped like in the f = 0 case and, in addition, also in the current address space.

在f = 0情况下，fpages是未映射的，此外，还在当前地址空间中。

FpageList MR 0:::k

Fpage MR i

0rwx

=0111

=0010

=0000

Fpages to be processed.

|  |  |
| --- | --- |
| fpage (28=58) | 0 r w x |

Fpage to be unmapped. (The term unmapped is used even if effectively no access right is re-moved.) A nilpage specifies a no-op.

Fpage被取消映射。 （即使实际上没有重新移动访问权限，也使用术语“未映射”。）nilpage指定了no-op。

Any access bit set to 1 revokes the corresponding access right. A 0-bit specifies that the corre-sponding access right should not be affected. Typical examples:

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

Complete unmap of the fpage.

Partial unmap, revoke writability only. As a result, the fpage is set to read-only.

部分取消映射，只撤销可写性。 结果，fpage被设置为只读。

No unmap. This case is particularly useful if only dirty and accessed bits should be read and reset without changing the mapping.

没有未映射。 这种情况是特别有用的，如果只有脏和被访问的位应该被读取和重置而不改变映射。

**Output Parameters**

FpageList MR 0:::k The accessed status bits in the fpages are updated. fpages中的访问状态位被更新。

UNMAP

Fpage MR i

R = 0

R = 1

* = 0
* = 1
* = 0
* = 1

43

|  |  |
| --- | --- |
| fpage (28=58) | 0 R W X |

The status bits Referenced, Written, and eXecuted of all pages processed by the unmap operation are reset and the bitwise OR-ed old values of all the processed pages are delivered in MR 0:::k. For processors that do not differentiate between read access and execute access, the R and X bits are unified: either both are set or both are reset. Resetting status bits is not a recursive operation. However, the status bit values for pages within the current space will also reflect accesses performed on recursive mappings.

被取消映射操作处理的所有页面的引用，写入和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MR 0 ::: k中传递。 对于不区分读访问和执行访问的处理器，R和X位是统一的：既可以设置，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。

No part of the fpage has been Referenced after the last unmap operation (or after the initial map operation). This includes all recursively mapped pages.

Remark: The meaning of referenced slightly differs from read. Not being referenced means that not only no read access but that also no write and execute access occurred.

在最后的非映射操作之后（或在初始映射操作之后），fpage的任何部分都没有被引用。 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。

At least one page of the specified fpage (including all recursive mappings) has been referenced after the last unmap operation (or after the initial map operation). All in-kernel R bits are reset Remark: The meaning of referenced slightly differs from read. Write accesses and execute accesses also set the R bit.

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）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。

No part of the fpage has been written after the last unmap operation (or after the initial map operation), i.e., the fpage is clean. This includes all recursively mapped pages.

在最后的非映射操作之后（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。

At least one page of the specified fpage (including all recursive mappings) has been written after the last unmap operation (or after the initial map operation), i.e., the fpage is dirty. All in-kernel dirty bits are reset.

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。

No part of the fpage has been eXecuted after the last unmap operation (or after the initial map operation). This includes all recursively mapped pages.

在最后的非映射操作之后（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。

At least one page of the specified fpage (including all recursive mappings) has been executed after the last unmap operation (or after the initial map operation). All in-kernel X bits are reset. Remark: For processors that do not differentiate between read and execute accesses, the X bit is set to 1 iff R = 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。

**Pagefaults**

No pagefaults will happen.

**Generic Programming Interface**

**System-Call Function:**

#include <l4/space.h>

void Unmap (Word control)

**Convenience Programming Interface**

**Derived Functions:**

#include <l4/space.h>

Fpage Unmap (Fpage f) [UnmapFpage]

f LoadMR (0, f); Unmap (0); StoreMR (0, f); f g

void Unmap (Word n, Fpage& [n] fpages) [UnmapFpages]

f LoadMRs (0, n, fpages); Unmap (n 1); StoreMRs (0, n, fpages); g

Recursively unmaps the specified fpage(s) from all address spaces except the current one.

44

UNMAP

Fpage Flush (Fpage f)

f LoadMR (0, f); Unmap (64); StoreMR (0, f); f g

void Flush (Word n, Fpage& [n] fpages)

f LoadMRs (0, n, fpages); Unmap (64 + n 1); StoreMRs (0, n, fpages); g

[FlushFpages]

Recursively unmaps the specified fpage(s) from all address spaces, including the current one.

Fpage GetStatus (Fpage f)

f LoadMR (0, f FullyAccessible); Unmap (0); StoreMR (0, f); f g

Resets and delivers the status bits of the specified fpage.

Bool WasReferenced (Fpage f)

Bool WasWritten (Fpage f)

Bool WaseXecuted (Fpage f)

Checks the status bits of specified fpage. The specified fpage must be the output of an Unmap (), Flush (), or GetStatus () function.

|  |  |
| --- | --- |
| SPACECONTROL | 45 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **4.3 SPACECONTROL [Privileged Systemcall]** | | |  |  |  |
| ThreadId | SpaceSpecifier | ! | Word | result |  |
| Word | control | Word | control |  |
| Fpage | KernelInterfacePageArea | |  |  |  |
| Fpage | UtcbArea |  |  |  |  |
| ThreadId | Redirector |  |  |  |  |

A privileged thread, e.g., the root server, can configure address spaces through this function.

特权线程（例如根服务器）可以通过此功能配置地址空间。

**Input Parameters**

SpaceSpecifier Since address spaces do not have ids, a thread ID is used as SpaceSpecifier. It specifies the address space in which the thread resides. The SpaceSpecifier thread must exist although it may be inactive or not yet started. In particular, the thread may reside in an empty address space that is not yet completely created.

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

Specifies the fpage where the kernel should map the kernel interface page. The supplied fpage must have a size specified in the KipAreaInfo field of the kernel interface page, must fit entirely into the user-accessible part of the address space and must not overlap with the UTCB area (see below). Address 0 of the kernel interface page is mapped to the fpage’s base address. The value is ignored if there is at least one active thread in the address space.

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。

KipAreaInfo [KernelInterfacePage Field]

Permits calculation of the appropriate page size of the KernelInterface area fpage.

允许计算KernelInterface区域fpage的适当页面大小。

|  |  |
| --- | --- |
| ~(2不=58) | s (6) |

* The size of the kernel interface page area is 2s.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| UtcbArea | Specifies the fpage where the kernel should map the UTCBs of all threads executing in the | | | | |
|  | address space. The fpage must fit entirely into the user-accessible part of an address space and | | | | |
|  | must not overlap with the KIP area. The fpage size has to be at least the smallest supported | | | | |
|  | hardware-page size. In fact, the size of the UTCB area restricts the maximum number of threads | | | | |
|  | that can be created in the address space. See the kernel interface page for the space and alignment | | | | |
|  | that is required for UTCBs. | |  |  |  |
|  | The value is ignored if there is at least one active thread in the address space.  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。 如果地址空间中至少有一个活动线程，则忽略该值。 | | | | |
|  |  |  |  |  |  |
| UtcbInfo [KernelInterfacePage Field] | |  |  |  |  |
|  | Permits to calculate the appropriate page size of the UTCB area fpage and specifies the size and | | | | |
|  | alignment of UTCBs. Note that the size restricts the total number of threads that can reside in | | | | |
|  | an address space.  允许计算UTCB区域fpage的相应页面大小，并指定UTCB的大小和对齐方式。 请注意，大小限制了可以驻留在地址空间中的线程总数。 | | | |  |
|  |  |  |  |  |  |
|  | ~(10=42) | s (6) | a (6) | m (10) |  |

|  |  |  |
| --- | --- | --- |
| s | The minimal area size for an address space’s UTCB area is 2s. The size of the UTCB area limits |  |
| the total number of threads k to 2amk 2s. |  |
|  |  |

* UTCB size multiplier.

|  |  |
| --- | --- |
| 46 | SPACECONTROL |
| a | The UTCB location must be aligned to 2a. The total size required for one UTCB is 2am.  UTCB位置必须对齐到2a。 一个UTCB所需的总大小是凌晨2am。 |
|  |  |

Redirector = nilthread

The current redirector setting for the specified space is not modified.

指定空间的当前重定向器设置不会被修改。

Redirector = anythread

All threads within the specified space are allowed to communicate with any thread in the system.

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 不= anythread, 不= nilthread

All threads within the specified address space are only allowed to send an IPC to a local thread or to a thread in the same address space as the specified redirector. All other send operations will be deflected to the redirector, the redirected bit (see page [67)](#page79) in the received message will be set, and the IntendedReceiver TCR will indicate the intended receiver of the message.

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，接收到的消息中的重定向位（请参见第67页）将被设置，而“有意接收器”TCR将指示消息的预期接收方。

|  |  |
| --- | --- |
| control | The control field is architecture specific (see Appendix [A.5)](#page116). It is undefined for some architec- |
|  | tures, but should for reasons of upward compatibility be set to zero.  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。 |
|  |  |

**Output Parameters**

|  |  |
| --- | --- |
| result | The result is 1 if the operation succeeded, otherwise the result is 0 and the ErrorCode TCR |
|  | indicates the failure reason. |
|  |  |

ErrorCode [TCR]

= 1

= 3

= 6

= 7

Set if result = 0. Undefined if result 不=0.

No privilege. Current thread does not have privilege to perform operation.

Invalid space. The SpaceSpecifier parameter specified an invalid thread ID.

Invalid UTCB area. Specified UTCB area too small (see UTCB info on page [4)](#page16) or not within user accessible virtual memory region (see Memory Descriptors on page [6)](#page18).

无效的UTCB区域。 指定的UTCB区域太小（请参见第4页的UTCB信息），或者不在用户可访问的虚拟内存区域内（请参阅第6页的内存描述符）。

Invalid KIP area. Specified KIP area too small (see KIP area info on page [4)](#page16) or not within user accessible virtual memory region (see Memory Descriptors on page [6)](#page18) or KIP area overlaps with UTCB area.

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 | Delivers the space control value that was effective for the thread when the operation was invoked. |
|  | The value is architecture specific.  在调用操作时提供对线程有效的空间控制值。 值是架构特定的。 |
|  |  |

**Pagefaults**

No pagefaults will happen.

**Generic Programming Interface**

**System-Call Function:**

#include <l4/space.h>

|  |  |
| --- | --- |
| SPACECONTROL | 47 |

Word SpaceControl (ThreadId SpaceSpecifier, Word control, Fpage KernelInterfacePageArea, UtcbArea, ThreadId Redirector, Word& old Control)

**Convenience Programming Interface**

**Support Functions:**

Word ErrorCode ()

Word ErrNoPrivilege

Word ErrInvalidSpace

Word ErrUtcbArea

Word ErrKipArea

48 SPACECONTROL

**Chapter 5**

IPC

50 MESSAGES AND MESSAGE REGISTERS (MRS)

**5.1 Messages And Message Registers (MRs)** **[Virtual Registers]**

Messages can be sent and received through the IPC system call (see page [64)](#page76). Basically, the sender writes a message into the sender’s message registers (MRs) and the receiver reads it from the receiver’s MRs. Each thread has 64 MRs, MR 0:::63. A message can use some or all MRs to transfer untyped words; it can include memory strings and fpages which are also specified using MRs.

MRs are virtual registers (see page [11),](#page23) but they are more transient than TCRs. MRs are read-once registers: once an MR has been read, its value is undefined until the MR is written again. The send phase of an IPC implicitly reads all MRs; the receive phase writes the received message into MRs.

The read-once property permits to implement MRs not only by special registers or memory locations, but also by general registers. Writing to such an MR has to block the corresponding general register for code-generator use; reading the MR can release it. Typically, code generated by an IDL compiler will load MRs just before an IPC system call and store them to user variables just afterwards.

消息可以通过IPC系统调用发送和接收（请参见第64页）。基本上，发送者将消息写入发送者的消息寄存器（MR），接收者从接收者的MR读取消息。每个线程有64个MR，MR 0 ::: 63。消息可以使用一些或全部MR来传送无类型的单词;它可以包括内存字符串和fpages，这些也是使用MR指定的。  
MR是虚拟寄存器（见第11页），但它们比TCR更为短暂。 MR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MR;接收阶段将接收到的消息写入MR。  
read-once属性允许不仅通过专用寄存器或存储器位置来执行MR，而且还可以通过通用寄存器来执行MR。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可以释放它。通常，由IDL编译器生成的代码将在IPC系统调用之前加载MR，并在之后将它们存储到用户变量中。

**Messages**

A message consists of up to 3 sections: the mandatory message tag, followed by an optional untyped-words section, followed by an optional typed-items section. The message tag is always held in MR 0. It contains message control information and the message label which can be freely set by the user. The kernel associates no semantics with it. Often, the message label is used to encode a request key or to define the method that should be invoked by the message.

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部分，后跟可选的键入项部分。 消息标签始终保存在MR 0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。 内核不与其相关联。 通常，消息标签用于对请求键进行编码或定义消息应该调用的方法。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| MsgTag [MR0] |  | label (16/48) | flags (4) | t (6) | u (6) |  |  |
|  |  |  |  |
| u | Number of untyped words following word 0. MR 1:::u hold the untyped words. u = 0 denotes | | | | | |  |
| a message without untyped words. | |  |  |  |  |  |
|  |  |  |  |  |  |
| t | Number of typed-item words following the untyped words or the message tag if no untyped | | | | | |  |
| words are present. The typed items use MR u+1:::u+t. A message without typed items has | | | | | |  |
|  |  |
|  | t = 0. | |  |  |  |  |  |
| flags | Message flags, see IPC systemcall, page [64.](#page76) | | |  |  |  |  |
| label | Freely available, often used to specify the request type or invoked method. | | | | | |  |

untyped words [MR1:::u]

The optional untyped-words section holds arbitrary data that is untyped from the kernel’s point of view. The data is simply copied to the receiver. The kernel associates no semantics with it.

可选的untyped-words部分保存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。

typed items [MRu+1:::u+t]

|  |  |  |  |
| --- | --- | --- | --- |
| The optional typed-items section is a sequence of items such as string items (page 59), map | | | |
| items (page [55),](#page67) grant items (page [57),](#page69) and (page [58)](#page70) ctrl transfer items.  可选的输入项目部分是字符串项目（页面59），地图项目（页面55），授予项目（页面57）和（第58页）ctrl转移项目等一系列项目。  键入的消息项目的类型编码在其第一个字的最低4位： | | |  |
| Typed message items have their type encoded in the lowermost 4 bits of their first word: | | |  |
| 0hhC | StringItem | see page [59](#page71) |  |
| 100C | MapItem | see page [55](#page67) |  |
| 101C | GrantItem | see page [57](#page69) |  |
| 110C | CtrlXferItem | see page [58](#page70) |  |
| 111C | Reserved |  |  |

The C bit signals whether the typed item is followed by another typed item (C = 1) or is the last one of the typed-item section (C = 0). The typed items must exactly fit into MR u+1:::u+t.

C位表示输入的项目后面是另一个输入项目（C = 1）还是输入项目部分（C = 0）的最后一个。 输入的项目必须完全符合MR u + 1 ::: u + t。

Note that C and t redundantly describe the message. This is by intention. The C bit allows efficient message parsing, whereas t + u can be used to store all MRs of a message to memory without parsing the complete message. Upon message sending, the C bits are completely ig-nored. The kernel will, however, ensure that the MRs on the receiver side will have the C bits set properly.

请注意，C和t冗余地描述消息。 这是有意的。 C位允许高效的消息解析，而t + u可以用来将消息的所有MR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MR将正确设置C位。

|  |  |
| --- | --- |
| MESSAGES AND MESSAGE REGISTERS (MRS) | 51 |

**Example Messages**

struct (label, Word [2] w)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Word w2 (32/64) | |  |  | MR 2 |  |
|  |  |  |  |  | MR 1 |  |
|  | Word w1 (32/64) | |  |  |  |
|  |  |  |  |  | MR 0 |  |
| label (16/48) |  | flags | t = 0 | u = 2 |  |

struct (label, MapItem m)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | MapItem m | |  |  |  | MR 1;2 |  |
|  |  |  | 1 0 0 0 |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  | MR 0 |  |
|  |  |  |  |  | |  |
| label (16/48) |  | flags | t = 2 | u = 0 | |  |

struct (label, Word w, StringItem s1; s2)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | StringItem s2 | |  |  |  | MR 4;5 |  |
|  |  |  | 0 h h 0 |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  | StringItem s1 | |  |  |  | MR 2;3 |  |
|  |  |  | 0 h h 1 |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  | MR 1 |  |
|  |  |  |  |  |  |  |
|  | Word w (32/64) | |  |  |  |  |
|  |  |  |  |  | | MR 0 |  |
| label (16/48) |  | flags | t = 4 | u = 1 | |  |

struct (label, Word [3] w, MapItem m, GrantItem g, StringItem s)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 52 |  |  | MESSAGES AND MESSAGE REGISTERS (MRS) | | | | |  |
|  |  |  |  |  |  |  |  |  |
|  |  | StringItem s | |  |  |  | MR 8;9 |  |
|  |  |  |  | 0 h h 0 |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  | GrantItem g | |  |  |  | MR 6;7 |  |
|  |  |  |  | 1 0 1 1 |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  | MapItem m | |  |  |  | MR 4;5 |  |
|  |  |  |  | 1 0 0 1 |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | MR 3 |  |
|  |  |  |  |  |  |  |  |
|  |  | Word w3 (32/64) | |  |  |  |  |
|  |  |  |  |  |  |  | MR 2 |  |
|  |  | Word w2 (32/64) | |  |  |  |  |
|  |  |  |  |  |  |  | MR 1 |  |
|  |  | Word w1 (32/64) | |  |  |  |  |
|  |  |  |  |  |  | | MR 0 |  |
|  | label (16/48) |  | flags | t = 6 | u = 3 | |  |

**Generic Programming Interface**

The listed generic functions permit user code to access message registers independently of the processor-specific MR model. All functions are user-level functions; the microkernel is not involved.

**MsgTag**

#include <l4/ipc.h>

struct **MSGTAG** { Word raw }

MsgTag Niltag

A message tag with no untyped or typed words, no label, and no flags.

Bool

==

(MsgTag l, r)

[IsMsgTagEqual]

Bool

! =

(MsgTag l, r)

[IsMsgTagNotEqual]

Compares all field values of two message tags.

Word Label (Msg Tag t)

Word UntypedWords (Msg Tag t)

Word TypedWords (Msg Tag t)

Delivers the message label, number of untyped words, and number of typed words, respectively.

|  |  |  |
| --- | --- | --- |
| MsgTag + (MsgTag t, Word label) | | [MsgTagAddLabel] |
| MsgTag + = (MsgTag t, Word label) | | [MsgTagAddLabelTo] |
|  | Adds a label to a message tag. Old label information is overwritten by the new label. | |
| MsgTag MsgTag | () |  |
| void Set MsgTag | (MsgTag t) |  |
|  | Delivers/sets MR 0. |  |
|  |  |  |

|  |  |
| --- | --- |
| MESSAGES AND MESSAGE REGISTERS (MRS) | 53 |

**Convenience Programming Interface**

**IDL-compiler generated Operations**

IDL code generators are not restricted to the generic interface for accessing MRs. Instead, they can use processor-specific methods and thus generate heavily optimized code for MR access.

IDL代码生成器不限于访问MR的通用接口。 相反，他们可以使用特定于处理器的方法，从而为MR访问生成大量优化的代码。

However, such processor-specific MR operations are not generally defined and should be used exclusively by processor-specific IDL code generators. All other programs must use the operations defined in this generic interface.

但是，这种特定于处理器的MR操作通常没有定义，应该由处理器特定的IDL代码生成器专门使用。 所有其他程序必须使用此通用接口中定义的操作。

**Msg**

#include <l4/ipc.h>

struct **MSG** { Word raw [64] }

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| void Put | (Msg& msg, Word l, int u, Word& [u] ut, int t, {MapItem, GrantItem, StringItem}& Items) | | | | [MsgPut] |
|  |  |  | Loads the specified parameters into the memory object msg. The parameters u and t respectively | | |
|  |  |  | indicate number of untyped words and number of typed words (i.e., the total size of all typed | | |
|  |  |  | items). It is assumed that the msg object is large enough to contain all items.  将指定的参数加载到内存对象消息中。 参数u和t分别指示无类型字词的数量和键入字词的数量（即，所有类型项目的总大小）。 假定msg对象足够大以包含所有项目。 | |  |
| void Get | (Msg& msg, Word& ut, fMapItem, GrantItem, StringItemg& Items) | | |  | [MsgGet] |
|  |  |  | Stores the msg object into the specified parameters. Type consistency between the message in | | |
|  |  |  | the memory object and the specified parameter list is not checked.  将msg对象存储到指定的参数中。 内存对象中的消息和指定的参数列表之间的类型一致性不被检查。 | |  |
| MsgTag MsgTag | | | (Msg& msg) |  | [MsgMsgTag] |
| void Set\_MsgTag | | | (Msg& msg, MsgTag t) | [Set MsgMsgTag] | |
|  |  |  | Delivers/sets the message tag of the msg object. |  |  |
| Word Label | | (Msg& msg) | |  | [MsgLabel] |
| void Set Label (Msg& msg, Word label) | | | | [Set MsgLabel] | |
|  |  |  | Delivers/sets the label of the msg object. |  |  |
| void Load | (Msg& msg) | | |  | [MsgLoad] |
|  |  |  | Loads message registers MR 0::: from the msg object. |  |  |
| void Store | (MsgTag t, Msg& msg) | | |  | [MsgStore] |
|  |  |  | Stores the message tag t and the current message beginning with MR 1 to the memory object | | |
|  |  |  | msg. The number of message registers to be stored is derived from t. | |  |
| void Clear (Msg& msg) | | | |  | [MsgClear] |
|  |  |  | Empties the msg object (i.e., clears the message tag). |  |  |
| void Append | | (Msg& msg, Word w) | | [MsgAppendWord] | |
| void Append | | (Msg& msg, MapItem m) | | [MsgAppendMapItem] | |
| void Append | | (Msg& msg, GrantItem g) | | [MsgAppendGrantItem] | |
| void Append | | (Msg& msg, StringItem s) | | [MsgAppendSimpleStringItem] | |
| void Append | | (Msg& msg, StringItem& s) | | [MsgAppendStringItem] | |
|  |  |  | Appends an untyped or a typed item to the msg object. Compound strings must always be | | |
|  |  |  | passed in by reference. A compound string passed by value will be treated as a simple string | | |
|  |  |  | (see page [59)](#page71). It is assumed that there is enough memory in the msg object to contain the new | | |
|  |  |  | item.  向msg对象附加无类型或类型的项目。 化合物字符串必须始终通过引用传入。 通过值传递的复合字符串将被视为简单字符串（请参见第59页）。 假定msg对象中有足够的内存来包含新项目。 | | |
| void Put | (Msg& msg, Word u, Word w) | | | [MsgPutWord] | |
|  |  |  | Puts an untyped word at untyped word position u (first untyped word has position 0) in the msg | | |
|  |  |  | object. It is assumed that the object contains at least u + 1 untyped words.  将一个无类型的单词放在msg对象中的无类型单词位置u（第一个无类型单词的位置为0）。 假定该对象至少包含u + 1个无类型单词。 | |  |

void Put (Msg& msg, Word t, MapItem m) [MsgPutMapItem]

|  |  |  |
| --- | --- | --- |
| 54 |  | MESSAGES AND MESSAGE REGISTERS (MRS) |
| void Put | (Msg& msg, Word t, GrantItem g) | [MsgPutGrantItem] |
| void Put | (Msg& msg, Word t, StringItem s) | [MsgPutSimplStringItem] |
| void Put | (Msg& msg, Word t, StringItem& s) | [MsgPutStringItem] |
| void Put | (Msg& msg, Word t, CtrlXferItem c) | [MsgPutCtrlXferItem] |
|  | Puts a typed item into the msg object, starting at typed word position t (first typed word has | |
|  | position 0). Compound strings must always be passed in by reference. A compound string | |
|  | passed by value will be treated as a simple string (see page [59)](#page71). It is assumed that that the object | |
|  | has enough typed words to contain the new item.  将输入的项目放入msg对象，从键入的单词位置t开始（第一个输入的单词具有位置0）。 化合物字符串必须始终通过引用传入。 通过值传递的复合字符串将被视为简单字符串（请参见第59页）。 假定对象有足够的单词来包含新项目。 | |
| Word Get | (Msg& msg, Word u) | [MsgWord] |
| void Get | (Msg& msg, Word u, Word& w) | [MsgGetWord] |
|  | Delivers the untyped words at position u. It is assumed that the object contains at least u + 1 | |
|  | untyped words.  在位置u提供无类型的单词。 假定该对象至少包含u + 1个无类型单词。 | |
| Word Get | (Msg& msg, Word t, MapItem& m) | [MsgGetMapItem] |
| Word Get | (Msg& msg, Word t, GrantItem& g) | [MsgGetGrantItem] |
| Word Get | (Msg& msg, Word t, StringItem& s) | [MsgGetStringItem] |
| Word Get | (Msg& msg, Word t, CtrlXferItem& c) | [MsgGetCtrlXferItem] |
|  | Delivers the typed item starting at typed word position t. It is assumed that the requested item | |
|  | is of the right size and type. Returns the size (in words) of the delivered item.  从键入的单词位置t开始输入已输入的项目。 假定请求的项目是正确的大小和类型。 返回交付项目的大小（单词）。 | |
|  |  |  |

**Low-Level MR Access**

#include <l4/ipc.h>

void StoreMR (int i, Word& w)

void LoadMR (int i, Word w)

Delivers/sets MR i.

void StoreMRs (int i; k, Word& [k] w)

void LoadMRs (int i; k, Word& [k] w)

Stores/loads MR i:::i+k 1 to/from memory.

|  |  |
| --- | --- |
| MAPITEM | 55 |

**5.2 MapItem** **[Data Type]**

An fpage (see page [40)](#page52) or IO fpage that should be mapped is sent to the mappee as part of a message. A map operation is a no-op within the same address space. The fpage is specified by a two-word descriptor:

应该映射的fpage（参见第40页）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。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| snd fpage (28=60) |  | 0 r w x | MR i+1 |  |
|  |  |  | MR i |  |
|  |  |  |  |
| snd base / 1024 (22/54) | 0 (6) | 1 0 0 C |  |

access rights rwx

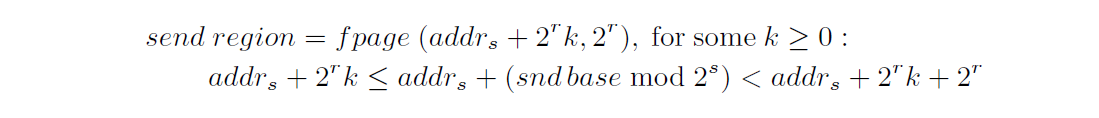
snd base

The effective access rights for the newly mapped page are calculated by bitwise AND-ing the access rights specified in the snd fpage and the access rights that the mapper itself has on that fpage. As such, the mapper can restrict the effective access rights but not widen them.

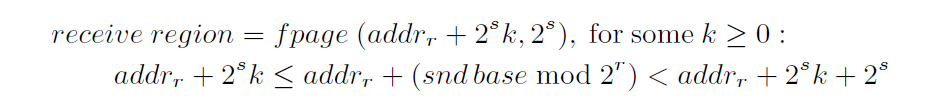
对新映射页面的有效访问权限是通过按位与在fnd页面中指定的访问权限和映射器本身在该页面上具有的访问权限进行计算来计算的。 因此，映射器可以限制有效的访问权限，但不能扩展它们。

The send base specifies the semantics of the map operation if the size of the snd fpage is larger or smaller than the window in which the receiver is willing to accept a mapping (see page [62)](#page74). If the size of the snd fpage, 2s, is larger than the receive window, 2r, the send base indicates which region of the snd fpage is transmitted. More precisely:

如果snd fpage的大小大于或小于接收者愿意接受映射的窗口（请参见第62页），则发送基数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，则发送基站指示发送snd fpage的哪个区域。 更确切地说



and where addrs is the base address of the snd fpage. If the size of the snd fpage, 2s, is smaller than the receive window, 2r, the send base indicates where in the receive window the snd fpage is mapped. More precisely:

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

and where addrr is the base address of the receive window.

Pages already mapped in the mappee’s address space that would conflict with new mappings are implicitly unmapped before new pages are mapped. For performance reasons extension of access rights is possible without prior unmapping, iff the very same mapping already exists. This is the case, when

已映射到mappee地址空间中的页面与新映射冲突的页面在新页面映射之前隐式地未映射。 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。 当是这样的话

* the mapper maps from the same address space as the existing mapping(映射器从与现有映射相同的地址空间映射); and
* the mapper maps from the same virtual source address as the existing mapping(映射器映射来自与现有映射相同的虚拟源地址); and
* the mapper maps to the same virtual destination address as the existing mapping(映射器映射到与现有映射相同的虚拟目标地址); and
* the object (physical address) is the same as the existing mapping.(对象（物理地址）与现有映射相同)

Access rights can not be revoked by mapping. The access rights of the resulting mapping are a bitwise OR of the existing and the new mapping’s access rights. Access rights are not extended recursively.

访问权限不能通过映射撤销。 结果映射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。

**Generic Programming Interface**

#include <l4/ipc.h>

struct **MAPITEM** f Word raw [2] g

MapItem MapItem (Fpage f, Word SndBase)

Delivers a map item with the specified fpage and send base.

56

MAPITEM

Bool MapItem (MapItem m)

Delivers true if map item is valid. Otherwise delivers false.

[IsMapItem]

Fpage SndFpage (MapItem m)

[MapItemSndFpage]

Word SndBase (MapItem m)

Delivers fpage/send base of map item.

[MapItemSndBase]

|  |  |
| --- | --- |
| GRANTITEM | 57 |

**5.3 GrantItem** **[Data Type]**

An fpage (see page [40)](#page52) or IO fpage that should be granted is sent to the mappee as part of a message. It is specified by a two-word descriptor:

应该被授予的fpage（参见第40页）或IO fpage作为消息的一部分发送到mappee。 它由一个双字描述符指定：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| snd fpage (28=60) |  | 0 r w x | MR i+1 |  |
|  |  |  | MR i |  |
|  |  |  |  |
| snd base / 1024 (22/54) | 0 (6) | 1 0 1 C |  |

access rights rwx

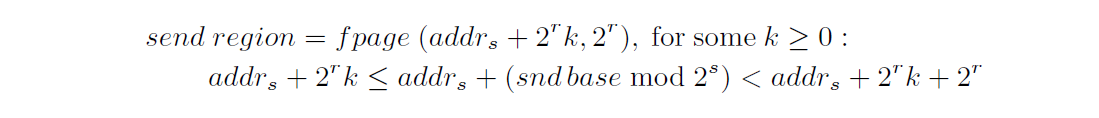
snd base

The effective access rights for the granted page are calculated by bitwise anding the access rights specified in the snd fpage and the access rights that the mapper itself has on that fpage. As such, the granter can restrict the effective access rights but not widen them.

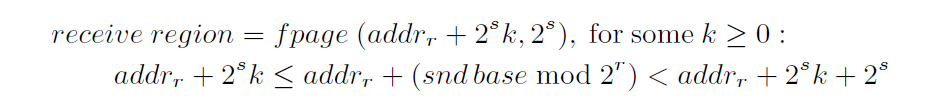
被授权页面的有效访问权限是通过按位访问权限来计算的，这些访问权限是在fnd页面中指定的访问权限以及映射器本身在fpage上的访问权限。 这样一来，大厨可以限制有效的访问权限，但不能扩大。

The send base specifies the semantics of the map operation if the size of the snd fpage is larger or smaller than the window in which the receiver is willing to accept a mapping (see page [62)](#page74). If the size of the snd fpage, 2s, is larger than the receive window, 2r, the send base indicates which region of the snd fpage is transmitted. More precisely:

如果snd fpage的大小大于或小于接收者愿意接受映射的窗口（请参见第62页），则发送基数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，则发送基站指示发送snd fpage的哪个区域。 更确切地说：



and where addrs is the base address of the snd fpage. If the size of the snd fpage, 2s, is smaller than the receive window, 2r, the send base indicates where in the receive window the snd fpage is mapped. More precisely:

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

and where addrr is the base address of the receive window.

Pages already mapped in the grantee’s address space that would conflict with new mappings are implicitly unmapped before new pages are mapped.

已经映射到被授权人的地址空间中与新映射冲突的页面在映射新页面之前隐式地未映射。

**Generic Programming Interface**

#include <l4/ipc.h>

struct **GRANTITEM** f Word raw [2] g

GrantItem GrantItem (Fpage f, Word SndBase)

Delivers a grant item with the specified fpage and send base.

Bool GrantItem (GrantItem g) [IsGrantItem]

Delivers true if grant item is valid. Otherwise delivers false.

Fpage SndFpage (GrantItem g) [GrantItemSndFpage]

Word SndBase (GrantItem g) [GrantItemSndBase]

Delivers fpage/send base of grant item.

58 CTRLXFERITEM

**5.4 CtrlXferItem** **[Data Type]**

A control transfer item specifies a control state such as instruction pointer, stack pointer, or general-purpose registers for the receiver of the message. The new values are automatically set by the kernel upon receiving the item. The contents of a control transfer item are architecture-specific. In general, a control transfer item is specified as follows:

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，如下指定控制转移项目

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | regn (32/64) | |  | MR i+n+2 |  |
|  |  |  |  |  |  |  |  |
|  |  | . |  |  | . | |  |
|  | . | |  |  | . | |  |
|  | . | |  |  | . | |  |
|  |  |  | reg1 (32/64) | |  | MR i+2 |  |
|  |  |  |  |  |  | MR i+1 |  |
|  |  |  | reg0 (32/64) | |  |  |
|  |  |  |  |  |  | |  |
|  |  | mask (20=52) |  | id (8) | 1 1 0 C | MR i |  |
| id | An identifier specifying the set of control transfer registers set by this item. Identifiers are | | | | | |  |
|  | architecture specific.  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 | A bitmask specifying the registers in the set to be modified.  指定要修改的集合中的寄存器的位掩码。 | | | |  |  |  |

**Generic Programming Interface**

#include <l4/ipc.h>

struct **CTRLXFERITEM** f Word raw [\*] g

Bool CtrlXferItem (CtrlXferItem& c) [IsCtrlXferItem]

Delivers true if control transfer item is valid. Otherwise delivers false.

如果控制权转移项目有效，则传递true。 否则传递错误。

void CtrlXferItemInit (CtrlXferItem& c, Word id)

Initializes the control transfer item with given id.

使用给定的ID初始化控制转移项目。

|  |  |
| --- | --- |
| STRINGITEM | 59 |

**5.5 StringItem** **[Data Type]**

A string item specifies a sequence of bytes in user space. No alignment is required, the maximal string size is 4 MB. In send messages, such a string is copied to the receiver buffer when transferring the message. String items are also used to specify receive buffers in buffer registers on the receiver’s side.

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。

**Simple String**

A simple string is a contiguous sequence of bytes.

一个简单的字符串是一个连续的字节序列。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | string ptr (32/64) |  |  |  |  | MR i+1 |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | |  |
|  | string length (22/54) | 0 |  | 0 (5) | 0 h h C | MR i |  |
| string ptr | The start address of the string to be sent or the start address of the buffer for receiving a string (no | | | | | |  |
|  | alignment restrictions). However, the string/buffer must fit entirely into the legally addressable | | | | | |  |
|  | user space.  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（不对齐限制）。 但是，字符串/缓冲区必须完全适合合法可寻址的用户空间。 | | | | | |  |
| string length | The length of the string to be sent or the size of the receive buffer. In the second case, strings | | | | | |  |
|  | up to (including) this length can be received. Maximum string length is 4 M bytes, even if the | | | | | |  |
|  | according field is 54 bits wide on 64-bit processors.  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。 | | | | | |  |
| h h | Cacheability hint. Except for hh = 00, the semantics of this parameter depends on the processor | | | | | |  |
| type (see Appendices [A.6](#page118) and ??).  缓存性提示。 除hh = 00外，此参数的语义取决于处理器类型（参见附录A.6和??）。 | | | | | |  |
|  |  |
| hh = 00 | Use the processor’s default cacheability strategy. Typically, cache lines are allocated for data | | | | | |  |
|  | read and written (assuming that the processor’s default strategy is write-back and write-allocate).  使用处理器的默认缓存策略。 通常情况下，缓存行分配给数据读取和写入（假设处理器的默认策略是回写和写分配）。 | | | | | |  |

**Compound String**

A compound string is a noncontiguous string that consists of multiple contiguous substrings which can be scattered around the entire user address space. The substrings must not overlap. For send and receive IPC operations, a compound string is handled as a single logical string. When sending such a string through IPC, the substrings are transferred as if they were one contiguous string (gather). On the receiver side, a compound string buffer is treated as one logical buffer. The corresponding received string is scattered among the compound buffer’s substrings.

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。

A compound string can be specified as a sequence of substrings where each substring has the form of a simple string except that the continuation flag c is set for all but the last substring. If j subsequent substrings have the same size, e.g., for equally sized buffers, a single length word can be used for all j substrings so that only j + 1 words instead of 2j words are required.

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被设置为连续标志c。 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。

|  |  |  |
| --- | --- | --- |
| length word | c j - 1 (5) 0 h h C |  |
| substring length (22/54) |  |

j

c = 0

c = 1

The type information 0hhC is only required for the first word of a string descriptor. The field is ignored for further length words in a compound-string descriptor.

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。

Number of subsequent string-ptr words. These string ptrs specify j substrings that have all the same substring length.

随后的字符串ptr字的数量。 这些字符串ptrs指定了具有全部相同子字符串长度的j个子字符串。

Continuation flag reset. The compound string descriptor ends with the jth string ptr word fol-lowing the current length word.

继续标志重置。 复合字符串描述符以第j个字符串ptr字结尾，后面跟随当前长度字。

Continuation flag set. The current length word and j string-ptr words are followed by (at least) one substring descriptor, i.e., another length word, etc.

延续标志设置。 当前长度字和j个字符串-ptr字之后是（至少）一个子字符串描述符，即另一个长度字等。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 60 |  |  |  |  |  | STRINGITEM |  |
| Example |  |  |  |  |  | MR i+j+2 |  |
| substringj+1 ptr (32/64) |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  | MR i+j+1 |  |
|  | substringj+1 length (22/54) | 0 | 0 (5) | 0 (4) |  |  |
|  |  |  |  |  |  | MR i+j |  |
|  | substringj ptr (32/64) |  |  |  |  |  |
|  | . |  |  |  | . | |  |
| . | |  |  |  | . | |  |
| . | |  |  |  | . | |  |
|  | substring1 ptr (32/64) |  |  |  |  | MR i+1 |  |
|  |  |  |  |  |  | MR i |  |
|  | substring1:::j length (22/54) | 1 | j - 1 (5) | 0 h h C |  |  |

**Generic Programming Interface**

#include <l4/ipc.h>

struct **STRINGITEM** f Word raw [\*] g

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Bool | StringItem | (StringItem& s) | | [IsStringItem] |
|  |  | Delivers true if string item is valid. Otherwise delivers false. | |  |
| Bool | CompoundString | | (StringItem& s) |  |
|  |  | Delivers the c-flag value (true = set).  判断是否是复合字符串，如果C = 1，是复合字符串 | |  |
| Word Substrings | | (StringItem& s) | |  |
| void\* Substring | | (StringItem& s, Word n) | |  |
|  |  | Delivers number of substrings/address of nth substring.  提供第n个子串的子串/地址的数目。 | |  |
| StringItem StringItem | | | (int size, void\* address) |  |
|  |  | Delivers a simple string item with the specified size and location.  提供具有指定大小和位置的简单字符串项目。 | |  |
| StringItem & + = (StringItem& dest, StringItem AdditionalSubstring) | | | | [AddSubstringTo] |
|  |  | Append substring to the string item. It is assumed that there is enough memory in the string item | | |
|  |  | to contain the new substring.  将字符串追加到字符串项目。 假定字符串项目中有足够的内存来包含新的子字符串。 | | |
| StringItem & + = (StringItem& dest, void\* AdditionalSubstringAddress) | | | | [AddSubstringAddressTo] |
|  |  | Append a new substring pointer to the string item. It is assumed that there is enough memory in | | |
|  |  | the string item to contain the new substring pointer.  追加一个新的子字符串指针到字符串项目。 假定字符串项中有足够的内存来包含新的子字符串指针。 | | |
|  |  |  |  |  |

**Convenience Programming Interface**

**Support Functions:**

#include <l4/ipc.h>

struct **CACHEALLOCATIONHINT** f Word raw g

CacheAllocationHint UseDefaultCacheLineAllocation

STRINGITEM 61

Bool == (CacheAllocationHint l, r) [IsCacheAllocationHintEqual]

Bool ! = (CacheAllocationHint l, r) [IsCacheAllocationHintNotEqual]

Compares two cache allocation hints.

CacheAllocationHint CacheAllocationHint (StringItem s)

Delivers the cache allocation hint of the string item.

StringItem + (StringItem s, CacheAllocationHint h)

[AddCacheAllocationHint]

StringItem + = (StringItem s, CacheAllocationHint h) [AddCacheAllocationHintTo]

Adds a cache allocation hint to a string item. An already existing hint is overwritten.

62 STRING BUFFERS AND BUFFER REGISTERS (BRS)

**5.6 String Buffers And Buffer Registers (BRs)** **[Pseudo Registers]伪寄存器**

For receiving messages that contain string items, the receiver has to specify appropriate string buffers. Such buffers are described by string items (see page [59)](#page71). A buffer can be contiguous (simple string) or non-contiguous (compound string).

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（参见第59页）。缓冲区可以是连续的（简单字符串）或不连续的（复合字符串）。

Such buffer descriptors are held in 33 per-thread Buffer Registers BR 0:::32. The number of buffer registers is sufficient to specify, for example, one compound buffer of 31 equally-sized sub-buffers. Up to 16 buffers can be specified provided that not more than 33 BRs are required.

这种缓冲区描述符保存在每个线程缓冲寄存器BR 0 ::: 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BR，就可以指定多达16个缓冲区。

When a message is received, the first message string item is copied into the first buffer string item which starts at BR 1; the next message string item is copied to the next buffer string item, etc. The list of buffer strings is terminated by having the C bit in the item type specifier of the last string zeroed.

当收到消息时，第一个消息字符串项被复制到BR 1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等等。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 are registers in the sense that they are per-thread objects and can only be addressed directly, not indirectly through pointers. BRs are static objects like TCRs, i.e., they keep their values until explicitly modified. BRs can be mapped to either special registers or to memory locations.

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Acceptor [BR0] |  | RcvWindow (28=60) | 0 0 c s |  |  |
|  |  |  |  |
|  |  |  |  |  |  |
|  | BR 0 specifies which typed items are accepted when a message is received.  BR 0指定收到消息时接受哪些类型的项目。 | | | |  |
| RcvWindow | Fpage (without access bits) that specifies the address-space window in which mappings and | | | |  |
|  | grants are accepted. Nilpage denies any mapping or granting; CompleteAddressSpace accepts | | | |  |
|  | any mapping or granting.  Fpage（无访问位），指定接受映射和授权的地址空间窗口。 Nilpage否认任何映射或授予; CompleteAddressSpace接受任何映射或授予。 | | | |  |
| c | Control transfer items are accepted iff c = 1.  如果c = 1，则接受控制转移项目。 | |  |  |  |
| s | StringItems are accepted iff s = 1.  StringItems被接受iff s = 1。 | |  |  |  |
|  |  |  |  |  |  |

buffer string items [BR1:::]

contain the valid buffer string items. Ignored if s = 0 in BR 0.

包含有效的缓冲区字符串项目。 BR 0中s = 0时忽略

**Generic Programming Interface**

The listed generic functions permit user code to access buffer registers independently of the processor-specific BR model.

All functions are user-level functions; the microkernel is not involved.

**Acceptor**

#include <l4/ipc.h>

struct **ACCEPTOR** f Word raw g

|  |  |  |
| --- | --- | --- |
| Acceptor | UntypedWordsAcceptor |  |
| Acceptor | StringItemsAcceptor |  |
| Acceptor | CtrlXferItemsAcceptor |  |
| Acceptor MapGrantItems (Fpage RcvWindow) | |  |
|  | Delivers an acceptor which allows untyped words, string items, or mappings and grants. | |
| Acceptor | + (Acceptor l, r) | [AddAcceptor] |
| Acceptor | + = (Acceptor l, r) | [AddAcceptorTo] |

Adds mappings/grants or string items to an acceptor. Adding a non-nil receive window will replace an existing window.

|  |  |  |
| --- | --- | --- |
| STRING BUFFERS AND BUFFER REGISTERS (BRS) | | 63 |
| Acceptor(Acceptor l, r) | | [RemoveAcceptor] |
| Acceptor = (Acceptor l, r) | | [RemoveAcceptorFrom] |
|  | Removes mappings/grants or string items from an acceptor. Removing a non-nil receive window | |
|  | will deny all mappings or grants, regardless of the size of the receive window. | |
| Bool | StringItems (Acceptor a) | [HasStringItems] |
| Bool MapGrantItems (受体 a) | | [HasMapGrantItems] |
|  | Checks whether string items/mappings are allowed. |  |
| Fpage RcvWindow (Acceptor a) | |  |
|  | Delivers the address space window where mappings and grants are accepted. Delivers nilpage | |
|  | if mappings or grants are not allowed. |  |
| void | Accept (Acceptor a) |  |
|  | Sets BR 0. |  |
| void | Accept (Acceptor a, MsgBuffer& b) | [AcceptStrings] |
|  | Sets BR 0 and loads the buffer description b into BR 1:::. |  |

Acceptor Accepted ()

Delivers BR 0.

**Convenience Programming Interface**

**MsgBuffer**

#include <l4/ipc.h>

struct **MSGBUFFER** f Word raw [32] g

|  |  |  |
| --- | --- | --- |
| void Clear (MsgBuffer& b) | | [MsgBufferClear] |
|  | Clears the message buffer (i.e., inserts a single empty string into it). | |
| void Append | (MsgBuffer& b, StringItem s) | [MsgBufferAppendSimpleRcvString] |
| void Append | (MsgBuffer& b, StringItem \* s) | [MsgBufferAppendRcvString] |
|  | Appends a string buffer to the message buffer. Compound strings must always be passed in by | |
|  | reference. A compound string passed by value will be treated as a simple string. It is assumed | |
|  | that there is enough memory in the message buffer object to contain the new string buffer. | |
|  |  |  |

**Low-Level BR Access**

#include <l4/ipc.h>

|  |  |  |
| --- | --- | --- |
| void | StoreBR | (int i, Word& w) |
| void | LoadBR | (int i, Word w) |
|  |  | Delivers/sets the value of BR i. |
| void | StoreBRs | (int i, k, Word& [k]) |
| void | LoadBRs | (int i, k, Word& [k]) |
|  |  | Stores/loads BR i:::i+k 1 to/from memory. |

Code generators of IDL and other compilers are not restricted to the generic interface. They can use any processor-specific methods and optimizations to access BRs.

64 IPC

**5.7 IPC** **[Systemcall]**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ThreadId | to | ! | ThreadId from |  |
| ThreadId | FromSpecifier |  |  |
| Word | Timeouts |  |  |  |

IPC is the fundamental operation for inter-process communication and synchronization. It can be used for intra- and inter-address-space communication. All communication is synchronous and unbuffered: a message is transferred from the sender to the recipient if and only if the recipient has invoked a corresponding IPC operation. The sender blocks until this happens or until a period specified by the sender has elapsed without the destination becoming ready to receive.

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。发件人阻塞直到发生这种情况，或直到发件人指定的时间段过去，而目的地不准备接收。

IPC can be used to copy data as well as to map or grant fpages from the sender to the recipient. For the description of messages see page [50.](#page62) A single IPC call combines an optional send phase followed by an optional receive phase. Which phases are included is determined by the parameters to and FromSpecifier. Transitions between send phase and receive phase are atomic.

IPC可用于复制数据以及映射或授予发件人到收件人的fpages。有关消息的说明，请参见第50页。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 operations are also controlled by MRs, BRs and some TCRs. RcvTimeout and SndTimeout are directly specified as system-call parameters. Each timeout can be 0, 1 (i.e., never expire), relative or absolute. For details on timeouts see page [30.](#page42)

IPC操作也受到MR，BR和一些TCR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,正无穷（即永不过期），相对或绝对。有关超时的详细信息，请参阅第30页。

**Variants** 变种

To enable implementation-specific optimizations, there exist two variants of the IPC system call. Functionally, both variants are identical. Transparently to the user, a kernel implementation can unify both variants or implement differently optimized functions.

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

* IPC

Default IPC function. Must always be used except if all criteria for using LIPC are fulfilled.

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。

IPC function that may be optimized for sending messages to local threads. Should be used whenever it is absolutely clear that in the overwhelming majority of all invocations

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

* a send phase is included; and

包含发送阶段

* the destination thread is specified as a local thread ID; and

目标线程被指定为本地线程ID

* a receive phase is included; and

包含接收阶段

* the destination thread runs on the same processor; and

目标线程在同一个处理器上运行

* the RcvTimeout is 1, and

RcvTimeout是1

* the IPC includes no map/grant operations.

IPC不包含地图/授权操作

**Input Parameters**

|  |  |
| --- | --- |
| to = nilthread | IPC includes no send phase. |
| to 不= nilthread | Destination thread; IPC includes a send phase  目标线程; IPC包括一个发送阶段 |
|  |  |

FromSpecifier = nilthread

IPC includes no receive phase.

|  |  |
| --- | --- |
| IPC | 65 |

FromSpecifier = anythread

IPC includes a receive phase. Incoming messages are accepted from any thread (including hardware interrupts).

IPC包括一个接收阶段。 来自任何线程（包括硬件中断）的消息都被接受。

FromSpecifier = anylocalthread

IPC includes a receive phase. Incoming messages are accepted from any thread that resides in the current address space.

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受。

FromSpecifier 不= nilthread, 不= anythread, 不= anylocalthread

Ipc includes a receive phase. Incoming messages are accepted only from the specified thread.

(Note that hardware interrupts can be specified.)

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。）

Timeouts

RcvTimeout

SndTimeout

|  |  |
| --- | --- |
| SndTimeout (16) | RcvTimeout (16) |
|  |  |

The receive phase waits until either a message transfer starts or the RcvTimeout expires. Ignored for send-only IPC operations.

For relative receive timeout values, the receive timeout starts to run after the send phase has successfully completed. If the receive timeout expires before the message transfer has been started IPC fails with “receive timeout”. A pending incoming message is received if the timeout period is 0.

接收阶段等待，直到消息传输开始或RcvTimeout到期。 忽略只发送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。

If the send timeout expires before the message transfer could start the IPC operation fails with “send timeout”. A send timeout of 0 ensures that IPC happens only if the addressed receiver is ready to receive when the send IPC operation is invoked. Otherwise, IPC fails immediately, i.e., without blocking.

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。

MsgTag [MR0]

u

t

p=0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| label (16/48) | 0 (3) | p | t (6) | u (6) |

Message head of the message to be sent. Only the upper 16/48 bits are freely available. The lower 16 bits hold the SndControl parameter. It describes the message to be sent and contains some control bits; ignored if no send phase.

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。

Number of untyped words following word 0. MR 1:::u hold the untyped words. u = 0 denotes a message with no untyped words.

在单词0后面的无类型单词的数量。MR 1 :::ü保持无类型的单词。 u = 0表示没有无类型单词的消息。

Number of words holding typed items that follow the untyped words (or the message tag if no untyped words are present). The typed items use MR u+1 and following MRs, potentially up to MR 63. t = 0 denotes a message without typed items.

持有输入类型的项目的单词的数量（或者是message tag如果没有无类型的单词存在）。 键入的项目使用MR u + 1和后面的MR，可能直到MR 63. t = 0表示没有类型项目的消息。

Normal (unpropagated) send operation. The recipient gets the original sender’s id.

正常（未传送）发送操作。 收件人获取原始发件人的ID。

|  |  |
| --- | --- |
| p=1 | Propagating send operation. The VirtualSender TCR specifies the id of the originator thread. (i.e., the thread to send the message on behalf of). If originator thread and current sender, or current sender and receiver reside in the same address space, propagation is always permitted. Otherwise, IPC occurs unpropagated. Propagation is also allowed if the originator thread is an interrupt thread waiting (closed) for the current thread, or if the current sender is a redirector for the originator thread (or there exists a chain of redirectors from the originator to the current sender).  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。  If propagation is permitted, the receiver receives the originator’s id instead of the current sender’s id, the p bit in the receiver’s MsgTag is set, and the current sender’s id is stored in the receiver’s ActualSender TCR. If the originator thread is waiting (closed) for a reply from the current sender, the originator’s state is additionally modified so that it now waits for the new receiver instead of the current sender.  如果允许传播，接收者接收发起者的id而不是当前发送者的id，接收者的MsgTag中的p位被设置，并且当前发送者的id被存储在接收者的ActualSender TCR中。如果发起者线程正在等待（关闭）来自当前发送者的回复，则发起者的状态被另外修改，以便它现在等待新的接收者而不是当前的发送者。 |
| label | Freely available, often used to specify the request type or invoked method, respectively.  自由可用，通常分别用于指定请求类型或调用的方法。 |
| [MR1:::u] | Untyped words to be sent. Ignored if no send phase. |
| [MRu+1:::u+t] | Typed items to be sent. Ignored if no send phase. |
|  |  |

66 IPC

XferTimeouts [TCR]

|  |  |
| --- | --- |
| XferTimeout Snd (16) | XferTimeout Rcv (16) |
|  |  |

Once a message transfer has been started, the time for transferring the message is roughly bounded by the minimum of sender’s and receiver’s XferTimeout. “Roughly” means that xfer timeouts are only checked when message copy raises a pagefault in the sender’s or in the re-ceiver’s address space. Copying data and mapping/granting is assumed to take no time. A relative transfer timeout always refers to the beginning of the message transfer (actually when the first page fault is raised). Logically, at that point it is transferred into an absolute timeout which then is used as send and receive timeout for the first and all subsequent page-fault RPCs in the message transfer.

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。

If the effective transfer timeout expires during the message transfer, IPC fails with “xfer timeout” (on both sides). Additional information specifies whether the page fault was in the receiver’s or in the sender’s address space and which part of the message was already transferred. Each thread has two transfer timeouts. One for the send phase and one for the receive phase.

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）失败。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。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Acceptor [BR0] |  | RcvWindow (28=60) | 0 0 c s |  |  |
|  |  |  |  |
|  |  |  |  |  |  |
|  | BR 0 specifies which typed items are accepted when a message is received.  BR 0指定收到消息时接受哪些类型的项目。 | | | |  |
| RcvWindow | Fpage (without access bits) that specifies the address-space window in which mappings and | | | |  |
|  | grants are accepted. Nilpage denies any mapping or granting; CompleteAddressSpace accepts | | | |  |
|  | any mapping or granting.  Fpage（无访问位），指定接受映射和授权的地址空间窗口。 Nilpage否认任何映射或授予; CompleteAddressSpace接受任何映射或授予。 | | | |  |
| s | StringItems are accepted iff s = 1.  StringItems被接受iff s = 1。 | |  |  |  |
| c | CtrlXferItems are accepted iff c = 1.  如果c = 1，则接受CtrlXferItems。 | |  |  |  |

buffer string items [BR1:::]

contain the valid buffer string items. Ignored if s = 0 in BR 0.

包含有效的缓冲区字符串项目。 BR 0中s = 0时忽略。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **Output Parameters** | | |  |  |  |  |  |
|  |  |  |  |  | |  | |  |  |
| from | Thread ID of the sender from which the IPC was received. Thread IDs are delivered as local | | | | | | | |  |
|  | thread IDs iff they identify a thread executing in the same address space as the current thread. It | | | | | | | |  |
|  | does not matter whether the sender specified the destination as local or global id. | | | | | | | |  |
|  | Only defined for IPC operations that include a receive phase.  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。 | | | | | | | |  |
|  |  |  |  |  |  |  |  |  |  |
| MsgTag [MR0] |  |  |  |  |  |  |  |  |  |
|  | label (16/48) | E X r p |  | t (6) |  | u (6) |  |  |
|  |  |  |  |  |  |
|  | If the IPC operation included a receive phase, MR 0 contains the message tag of the received | | | | | | | |  |
|  | message. The upper 16/48 bits contain the user-specified label. The lower bits describe the | | | | | | | |  |
|  | received message, contain the error indicator, and the cross-processor IPC indicator. | | | | | | | |  |
|  | MR 0 is defined even if the IPC operation did not include a receive phase. In the send-only case, | | | | | | | |  |
|  | MR 0 returns the error indicator.  如果IPC操作包括接收阶段，则MR 0包含接收到的消息的消息标记。 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，MR 0也被定义。 在发送情况下，MR 0返回错误指示符。 | | | | | | | |  |
| u | Number of untyped words following word 0. u = 0 means no untyped words. For IPC opera- | | | | | | | |  |
| tions without receive phase, u = 0 is delivered.  单词0之后的无类型单词的数量。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被传送。 | | | | | | | |  |
|  |  |
| t | Number of received words that hold typed items. t = 0 means no typed items. For IPC opera- | | | | | | | |  |
| tions without receive phase, t = 0 is delivered.  保存输入项目的收到字数。 t = 0表示没有输入的项目。 对于没有接收阶段的IPC操作，传送t = 0。 | | | | | | | |  |
|  |  |
| p | Propagated IPC. If reset (p = 0) the IPC was not propagated. If set (p = 1) the IPC was propa- | | | | | | | |  |
|  | gated and the FromSpecifier indicates the originator thread’s id. The ActualSender specifies the | | | | | | | |  |
|  | id of the thread which performed the propagation.  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。 | | | | | | | |  |

IPC

r

X

E

67

Redirected IPC. If reset (r = 0) the IPC was not a redirected one. If set (r = 1) the IPC was redirected to the current thread, and the IntendedReceiver TCR specifies the id of the thread supposed to receive the message.

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，IntendedReceiver TCR指定应该接收消息的线程的ID。

Cross-processor IPC. If reset (X = 0) the received IPC came from a thread running on the same processor as the receiver. If set (X = 1) the received IPC was cross-processor. For IPC operations without receive phase, X = 0 is delivered.

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。

Error indicator. If reset (E = 0) the IPC operation terminated successful.

If set (E = 1) IPC failed. If the send phase was successful but a receive timeout occurred afterwards, or if a message could only be partially transferred, the entire IPC fails. The error code and additional information can be retrieved from the ErrorCode TCR. The fields label, t, and u are valid if the error code signals a partially received message.

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 | Label of the received message. For IPC operations without receive phase, the label is 0. |
| [MR1:::u] | Untyped words that have been received. Undefined if no receive phase. |
| [MRu+1:::u+k] | Typed items that have been received. Undefined if no receive phase. |
|  |  |

|  |  |
| --- | --- |
| ErrorCode [TCR] |  |
| x (28=56) | e (3) p |

Only defined if the error indicator E in MR 0 is set. IPC failed, i.e., was not correctly completed. The x field depends on the error code, see below. The p field specifies whether the error occurred during send or receive phase. If the error occurred during the receive phase the send phase (if any) was completed successfully before. If the error occurred during the send phase, the receive phase (if any) was skipped.

仅在MR 0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 Specifies whether the error occurred during the send phase (p = 0) or the receive phase (p = 1).

errors 1, 2,3

e = 1

e = 2

e = 3

|  |  |  |
| --- | --- | --- |
| ~(28=60) | e (3) | p |

Error happened before a partner thread was involved in the message transfer. Therefore, the error is signaled only to the thread that invoked the failing IPC operation.

在伙伴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。

Timeout.

From is undefined in this case.

Non-existing partner. If the error occurred in the send phase, to does not exist. (Anythread as a destination is illegal and will also raise this error.) If the error occurred in the receive phase, FromSpecifier does not exist. (FromSpecifier = anythread is legal, and thus will never raise this error.)

Canceled by another thread (system call exchange registers).

errors 4,5,6,7

offset

e = 4

|  |  |  |
| --- | --- | --- |
| offset (28=60) | e (3) | p |

A partner thread is already involved in the IPC operation, and the error is therefore signaled to both threads.

IPC操作中已经涉及到合作伙伴线程，因此将错误信号发送给两个线程。

The message transfer has been started and could not be completed. The offset identifies exactly the number of bytes that have been been transferred successfully so far through string items.

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。

Message Overflow.

A message overflow can occur (1) if a receiving buffer string is too short, (2) if not enough buffer string items are present, and (4) if a map/grant of an fpage fails because the system has not enough page-table space available. The offset in conjunction with the received MRs permits sender and receiver to exactly determine the reason.

（1）接收缓冲串太短，（2）缓冲区串项目不足，（4）fpage的映射/授权由于系统没有足够的分页功能而失败， 可用的表空间。 与收到的MR一起的偏移允许发送者和接收者准确地确定原因。

e = 5 Xfer timeout during page fault in the invoker’s address space.

在调用者的地址空间中的页面错误期间Xfer超时。

e = 6 Xfer timeout during page fault in the partner’s address space.

在伙伴地址空间的页面错误期间，Xfer超时。

e = 7 Aborted by another thread (system call exchange registers).

由另一个线程（系统调用交换寄存器）中止。

68 IPC

**Pagefaults**

Three different types of pagefault can occur during ipc: pre-send, post-receive, and xfer pagefaults. Only xfer pagefault are critical from a security point of view. Fortunately, messages without strings will never raise xfer pagefaults and need thus no special pagefault provisions:

在ipc期间可能发生三种不同类型的页面错误：预发送，接收后和xfer页面错误。 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页面错误，因此不需要特殊的页面错误规定：

Pre-send pagefaults

happen in the sender’s context before the message transfer has really started. The destination thread is not involved; in particular, it is not locked. Therefore, the destination thread might receive another message or time out while the sender’s pre-send pagefault is handled. Send and transfer timeouts do not control pre-send pagefaults. Pre-send pagefaults are uncritical from a security point of view, since only the sender’s own pager is involved and only the sender could suffer from its potential misbehavior.

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。 预先发送的页面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

happen in the receiver’s context after the message has been transferred. The sender thread is no longer involved, especially, it is no longer locked. Consequently, post-receive pagefault are not subject to send and transfer timeouts. Like pre-send pagefaults, post-receive pagefaults are also uncritical from a security perspective since only the receiver and its pager are involved.

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。 就像预发送页面错误一样，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传呼机。

Xfer pagefaults happen while the message is being transferred and both sender and receiver are involved. There-fore, xfer pagefaults are critical from a security perspective: If such a pagefault occurs in the receiver’s space, the sender may be starved by a malicious receiver pager. An xfer pagefault in the sender’s space and a malicious sender pager may starve the receiver. As such, xfer pagefaults are controlled by the minimum of sender’s and receiver’s xfer timeouts.

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页面错误是至关重要的：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页面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。

However, xfer pagefaults can only happen when transferring strings. Send mes-sages without strings or receive messages without receive string buffers are guaranteed not to raise xfer pagefaults.

但是，xfer页面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保证不会引发xfer页面错误。

**Generic Programming Interface**

**System-Call Function:**

#include <l4/ipc.h>

MsgTag Ipc (ThreadId to, FromSpecifier, Word Timeouts, ThreadId& from)

MsgTag Lipc (ThreadId to, FromSpecifier, Word Timeouts, ThreadId& from)

Note that message registers have read-once semantics and that returning the message tag implies reading MR 0. The contents of the message tag is therefore lost if the application does not implicitly store the return value of IPC or LIPC .

**Convenience Programming Interface**

**Derived Functions:**

#include <l4/ipc.h>

MsgTag Call (ThreadId to)

f Call (to, never, never) g

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| IPC |  |  |  | 69 |
| MsgTag Call | (ThreadId to, Time SndTimeout, RcvTimeout) | | | [Call Timeouts] |
|  |  |  | f Ipc (to, to, Timeouts (SndTimeout, RcvTimeout), –) g |  |
| MsgTag Send | (ThreadId to) | | |  |
|  |  |  | f Send (to, never) g |  |
| MsgTag Send | (ThreadId to, Time SndTimeout) | | | [Send Timeout] |
|  |  |  | f Ipc (to, nilthread, Timeouts (SndTimeout, –), –) g |  |
| MsgTag Reply | (ThreadId to) | | |  |
|  |  |  | f Send (to, ZeroTime) g |  |
| MsgTag Receive | | (ThreadId from) | |  |
|  |  |  | f Receive (from, never) g |  |
| MsgTag Receive | | (ThreadId from, Time RcvTimeout) | | [Receive Timeout] |
|  |  |  | f Ipc (nilthread, from, Timeouts (–, RcvTimeout), –) g |  |
| MsgTag Wait | (ThreadId& from) | | |  |
|  |  |  | f Wait (never, from) g |  |
| MsgTag Wait | (Time RcvTimeout, ThreadId& from) | | | [Wait Timeout] |
|  |  |  | f Ipc (nilthread, anythread, Timeouts (–, RcvTimeout), from) g |  |
| MsgTag ReplyWait | | | (ThreadId to, ThreadId& from) |  |
|  |  |  | f ReplyWait (to, never, from) g |  |
| MsgTag ReplyWait | | | (ThreadId to, Time RcvTimeout, ThreadId& from) | [ReplyWait Timeout] |
|  |  |  | f Ipc (to, anythread, Timeouts (TimePeriod(0), RcvTimeout), from) g |  |
| void Sleep (Time t) | | | |  |
|  |  |  | f Set MsgTag (Receive (MyLocalId, t)) g |  |
| MsgTag Lcall | (ThreadId to) | | |  |
|  |  |  | f Lipc (to, to, Timeouts (never, never), –) g |  |

MsgTag LreplyWait (ThreadId to, ThreadId& from)

f Lipc (to, anylocalthread, Timeouts (TimePeriod (0), never), from) g

**Support Functions:**

#include <l4/ipc.h>

Bool IpcSucceeded (MsgTag t)

Bool IpcFailed (MsgTag t)

Delivers the state of the error indicator (the E bit of MR 0).

Bool IpcPropagated (MsgTag t)

Bool IpcRedirected (MsgTag t)

Bool IpcXcpu (MsgTag t)

Checks if the IPC was propagated/redirected/cross cpu.

Word ErrorCode ()

ThreadId IntendedReceiver ()

70 IPC

ThreadId ActualSender ()

Delivers the error code/intended receiver TCR/actual sender.

void Set Propagation (MsgTag& t)

Sets the propagation bit.

void Set VirtualSender (ThreadId t)

Sets the virtual sender TCR.

Word Timeouts (Time SndTimeout, RcvTimeout)

Delivers a word containing both timeout values.

**Chapter 6**

Miscellaneous

72 EXCEPTIONHANDLER

**6.1 ExceptionHandler**异常处理程序 **[TCR]**

An exception handler thread can be installed to receive exception IPCs.

可以安装异常处理程序线程来接收异常IPC。

ExceptionHandler

不=nilthread

=nilthread

Specifies the exception handler thread. When a thread raises an exception the kernel sends an exception IPC message on the thread’s behalf to the thread’s exception handler thread and waits for a response from the exception handler containing the instruction pointer where the thread should continue execution in MR 1. The format of the exception IPC message is architecture specific.

The architectural registers of the faulting thread, BR 0, TCRs, and the MRs containing the ex-ception message are preserved.

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。 异常IPC消息是体系结构特定的。  
故障线程的结构寄存器，BR 0，TCR和包含该消息的MR被保留。

No exception handler is specified. If an exception is raised the thread is halted and not scheduled anymore. nilthread is the default value for newly created threads.

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。 nilthread是新创建的线程的默认值。

**Generic Programming Interface**

#include <l4/thread.h>

ThreadId ExceptionHandler ()

void Set ExceptionHandler (ThreadId new)

Delivers/sets the exception handler TCR.

|  |  |
| --- | --- |
| COP FLAGS | 73 |

**6.2 Cop Flags** **[TCR]**

The coprocessor flags TCR helps the kernel to optimize thread switching for some hardware architectures.

协处理器标志TCR帮助内核优化某些硬件体系结构的线程切换。

Cop Flags

c7 : : : c0

By resetting a ci-bit to 0, a thread tells the system that it no longer needs coprocessor i. If the kernel finds ci = 0, it concludes that registers and state of coprocessor i do not have to be saved. However, the kernel ensures that the coprocessor can not be used as a covert channel between different address spaces.

Once a thread has reset bit ci it must set ci to 1 before it issues the next operation on coprocessor i. Otherwise, coprocessor registers and state might be arbitrarily modified while using it.

Note that the ci-bits are write-only. Reading them results in an undefined value. Upon thread creation, all ci-bits are set to 1.

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。

**Generic Programming Interface**

#include <l4/thread.h>

void Set CopFlag (Word n)

void Clr CopFlag (Word n)

Sets/clears coprocessor flag cn.

74 PROCESSORCONTROL

**6.3 PROCESSORCONTROL** **[Privileged Systemcall]**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Word | ProcessorNo | ! | Word result |  |
| Word | InternalFrequency |  |  |
| Word | ExternalFrequency |  |  |  |
| Word | voltage |  |  |  |

Control the internal frequency, external frequency, or voltage for a system processor.

控制系统处理器的内部频率，外部频率或电压。

|  |  |
| --- | --- |
|  | **Input Parameters** |
|  |  |
| ProcessorNo | Specifies the processor to control. Number must be a valid index into the processor descriptor |
|  | array (see Kernel Interface Page, page [4)](#page16).  指定要控制的处理器。 数字必须是处理器描述符数组的有效索引（请参阅内核接口页面，第4页）。 |

All further input parameters have no effect if the supplied value is -1, ensuring that the corresponding value is not modified. The following description always refers to values 不= -1.

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

InternalFrequency Sets internal frequency for processor to the given value (in kHz).

ExternalFrequency

Sets external frequency for processor to the given value (in kHz).

|  |  |
| --- | --- |
| voltage | Sets voltage for processor to the given value (in mV). A value of 0 shuts down the processor. |
|  |  |
|  | **Output Parameters** |
|  |  |
| result | The result is 1 if the operation succeeded, otherwise the result is 0 and the ErrorCode TCR |
|  | indicates the failure reason. |
|  |  |
| ErrorCode [TCR] | Set if result = 0. Undefined if result 不=0. |

* 1 No privilege. Current thread does not have privilege to perform operation.

没有特权。 当前线程没有执行操作的权限。

Note that the active internal and external frequency of all processors are available to all threads via the kernel interface page.

请注意，所有处理器的活动内部和外部频率均可通过内核接口页面供所有线程使用。

**Pagefaults**

No pagefaults will happen.

|  |  |
| --- | --- |
| PROCESSORCONTROL | 75 |

**Generic Programming Interface**

**System-Call Function:**

#include <l4/misc.h>

Word ProcessorControl (Word ProcessorNo, InternalFrequency, ExternalFrequency, voltage)

**Convenience Programming Interface**

**Support Functions:**

Word ErrorCode ()

Word ErrNoPrivilege

76 MEMORYCONTROL

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **6.4 MEMORYCONTROL** | **[Privileged Systemcall]** | |  |  |
| Word | control | ! | Word result |  |
| Word | attribute0 |  |  |
| Word | attribute1 |  |  |  |
| Word | attribute2 |  |  |  |
| Word | attribute3 |  |  |  |

Set the page attributes of the fpages (MR 0:::k) to the attribute specified with the fpage.

将fpages（MR 0 ::: k）的页面属性设置为fpage指定的属性。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Input Parameters** |  |  |  |
|  |  |  |  |  |
| control |  |  |  |  |
| 0 (2不=58) | k (6) |  |  |
|  |  |  |

* Specifies the highest MR k that holds an fpage to set the attributes. The number of fpages is thus k + 1.

指定拥有fpage的最高MR k来设置属性。 因此，数量是k + 1。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| attributei | Specifies the attribute to associate with an fpage. | | The semantics of the attributei values are | | | |  |
|  | hardware specific, except for the value 0 which specifies default semantics.  指定要与fpage关联的属性。 属性值的语义是 硬件特定的，除了指定默认语义的值0之外。 | | | | | |  |
|  |  |  |  |  |  |  |  |
| FpageList MR 0:::k | Fpages to be processed.要处理的页面 | |  |  |  |  |  |
| Fpage MR i |  |  |  |  |  |  |  |
|  | fpage (28=60) |  | 0 0 | a (2) |  |  |
|  |  |  |  |  |
|  | Fpage to change the attributes. A nilpage specifies a no-op.  Fpage改变属性。 一个nilpage指定一个no-op。 | | | | | |  |
| a | selects attributea to be set as the fpages memory attributes.  选择属性设置为fpages内存属性。 | | | | | |  |

**Output Parameters**

|  |  |
| --- | --- |
| result | The result is 1 if the operation succeeded, otherwise the result is 0 and the ErrorCode TCR |
|  | indicates the failure reason. |
|  |  |
| ErrorCode [TCR] | Set if result = 0. Undefined if result 不=0. |

* 1No privilege. Current thread does not have privilege to perform operation.
* 5Invalid parameter. Invalid or unsupported memory attribute.

**Pagefaults**

No pagefaults will happen.

|  |  |
| --- | --- |
| MEMORYCONTROL | 77 |

**Generic Programming Interface**

**System-Call Function:**

#include <l4/misc.h>

Word MemoryControl (Word control, Word& attributes[4])

Word DefaultMemory

**Convenience Programming Interface**

**Derived Functions:**

#include <l4/misc.h>

Word Set PageAttribute (Fpage f, Word attribute)

f Word attributes[4]; attributes[0] = attribute; Set Rights(f, 0); LoadMR (0, f); MemoryControl (0, &attributes); g

Word Set PagesAttributes (Word n, Fpage& [n] fpages, Word& [4] attributes)

f LoadMRs (0, n, fpages); MemoryControl (n 1, attributes); g

**Support Functions:**

Word ErrorCode ()

Word ErrNoPrivilege

Word ErrInvalidParam

78 MEMORYCONTROL

**Chapter 7**

Protocols

80 THREAD START PROTOCOL

**7.1 Thread Start Protocol** **[Protocol]**

Newly created active threads start immediately by receiving a message from its pager. The received message contains the initial instruction-pointer and stack-pointer for the thread.

新创建的活动线程立即从其传呼机接收消息开始。 收到的消息包含线程的初始指令指针和堆栈指针。

From Pager

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Initial SP (32/64) | | | | MR 2 |  |
|  |  |  |  |  | MR 1 |  |
|  | Initial IP (32/64) | | | |  |
|  |  |  |  |  | MR 0 |  |
|  |  |  |  |  |  |
| 0 (16/48) |  | 0 (4) | t = 0 (6) | u = 2 (6) |  |

|  |  |
| --- | --- |
| INTERRUPT PROTOCOL | 81 |

**7.2 Interrupt Protocol** **[Protocol]**

Interrupts are delivered as an IPC call to the interrupt handler thread (i.e., the pager of the interrupt thread). The interrupt is disabled until the interrupt handler sends a re-enable message.

中断作为IPC调用被传递给中断处理程序线程（即中断线程的调页程序）。 中断被禁用，直到中断处理程序发送重新启用消息。

中断处理过程中不能被再中断

From Interrupt Thread

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| -1 (12/44) | 0 (4) | 0 (4) | t = 0 (6) | u = 0 (6) | MR 0 |

To Interrupt Thread

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 (16/48) | 0 (4) | t = 0 (6) | u = 0 (6) | MR 0 |

82 PAGEFAULT PROTOCOL

**7.3 Pagefault Protocol** **[Protocol]**

A thread generating a pagefault will cause the kernel to transparently generate a pagefault IPC to the faulting thread’s pager. The behavior of the faulting thread is undefined if the pager does not exactly follow this protocol.

生成页面错误的线程将导致内核透明地为错误线程的调页程序生成页面错误IPC。 如果调页程序不完全遵循这个协议，那么错误线程的行为是不确定的。

To Pager

rwx

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | faulting user-level IP (32/64) | | | | | MR 2 |  |
|  |  |  |  |  |  | MR 1 |  |
|  |  |  |  |  |  |  |
|  | fault address (32/64) | | | | |  |
|  |  |  |  |  |  | MR 0 |  |
|  |  |  |  |  |  |  |
| 2 (12/44) |  | 0 r w x | 0 (4) | t = 0 (6) | u = 2 (6) |  |

The rwx bits specify the fault reason:

* read fault

w write fault

x execute fault

A bit set to one reports the type of the attempted access. On processors that do not differentiate between read and execute accesses, x is never set. Read and execute accesses will both be reported by the r bit.

设置为1的位报告尝试访问的类型。 在不区分读取和执行访问的处理器上，x从不设置。 读取和执行访问都将由r位报告。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Acceptor [BR0 | ] | 0 (22/54) | s = 1 (6) | 0 0 0 0 | BR 0 |  |
|  |  |  |

The acceptor covers the complete user address space. The kernel accepts mappings or grants into this region on behalf of the faulting thread. The received message is discarded.

接受者覆盖了完整的用户地址空间。 内核代表错误线程接受映射或授予该区域。 收到的消息被丢弃。

From Pager

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | MapItem / GrantItem | | | | MR 1;2 |  |
|  |  |  |  |  | MR 0 |  |
|  |  |  |  |  |  |
| 0 (16/48) |  | 0 (4) | t = 2 (6) | u = 0 (6) |  |

|  |  |
| --- | --- |
| PREEMPTION PROTOCOL | 83 |

**7.4 Preemption Protocol抢占协议** **[Protocol]**

From Preempted Thread

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Clock =2(32/64) (32/64) | | | | | MR 2 |  |
|  |  |  |  |  |  | MR 1 |  |
|  |  |  |  |  |  |  |
|  | Clock mod2(32/64) (32/64) | | | | |  |
|  |  |  |  |  |  | MR 0 |  |
|  |  |  |  |  |  |  |
| -3 (12/44) |  | 0 (4) | 0 (4) | t = 0 (6) | u = 2 (6) |  |

The preemption message contains the system clock when the thread was preempted. The pre-emption message is sent with relative timeout 0. If the message can not be delivered (e.g., due to timeouts) the message is dropped.

占先消息包含线程被抢占时的系统时钟。 抢占消息以相对超时0发送。如果抢占消息不能被传送（例如，由于超时），则消息被丢弃。

84 EXCEPTION PROTOCOL

**7.5 Exception Protocol异常协议** **[Protocol]**

The exception IPC contains a label, the faulting instruction pointer, and additional architecture specific exception words. The reply from the exception handler contains a label, an instruction pointer where the faulting thread is resumed, and an optional number of additional architecture specific words.

Note that the stack pointer is not explicitly specified to allow architecture specific optimizations.

异常IPC包含标签，错误指令指针以及其他体系结构特定的异常字。 来自异常处理程序的回复包含一个标签，一个指令指针，指向故障线程恢复的地方，另外还有一些可选的附加体系结构特定字。  
请注意，未明确指定堆栈指针以允许特定于架构的优化。

To Exception Handler

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | exception word k 1 (32/64) | | | | |  |  | MR k+1 |  |
|  |  | . |  |  |  |  |  |  | . | |  |
|  | . | |  |  |  |  |  |  | . | |  |
|  | . | |  |  |  |  |  |  | . | |  |
|  |  |  | exception word 0 (32/64) | | | |  |  |  | MR 2 |  |
|  |  |  |  |  |  |  |  |  |  | MR 1 |  |
|  |  |  |  | IP (32/64) | | |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  | MR 0 |  |
|  |  | label (12/44) |  | 0 (4) | 0 (4) |  | t = 0 (6) | u = k (6) |  |  |
| k | Number of exception words. | | | |  |  |  |  |  |  |  |
| label | specifies the exception type. | | | |  |  |  |  |  |  |  |

* -4 System exceptions are defined for all architectures. 系统异常是为所有体系结构定义的。
* -5 Architecture specific exceptions. 架构特定异常。

From Exception Handler

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | exception reply word k 1 (32/64) | | | | |  |  | MR k+1 |  |
|  |  | . |  |  |  |  |  | . | |  |
|  | . | |  |  |  |  |  | . | |  |
|  | . | |  |  |  |  |  | . | |  |
|  |  | exception reply word 0 (32/64) | | | | |  |  | MR 2 |  |
|  |  |  |  |  |  |  |  |  | MR 1 |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  | IP (32/64) | |  |  |  |  |  |
|  |  | |  |  |  |  |  |  | MR 0 |  |
|  |  | 0 (16/48) |  | 0 (4) | t = 0 (6) |  | u = k (6) |  |  |
| k | Number of exception reply words. | | | |  |  |  |  |  |  |
| IP | Location where execution is resumed in the faulting thread.定位在失败线程中恢复执行的位置 | | | | | | | | |  |

|  |  |
| --- | --- |
| EXTENDED CONTROL TRANSFER PROTOCOL | 85 |

**7.6 Extended Control Transfer Protocol** **[Protocol]**

**扩展的控制传输协议**

To facilitate building L4-based virtualization solutions, the kernel can be configured to include extended control trans-fer state for kernel-generated messages, that is, for thread startups, pagefaults, exceptions, preemptions, and all other, architecture-specific types of messages.

为了便于构建基于L4的虚拟化解决方案，可以将内核配置为包含内核生成消息的扩展控制传输状态，即针对线程启动，页面错误，异常，抢占以及所有其他体系结构特定的类型消息。

**Configuring default control transfer state**

配置默认控制传输状态

By default, the kernel will use the protocols specified in the previous sections. Upon request, the kernel will switch to an extended protocol based on control transfer items. Requests to enable/disable the extended protocol are performed using the EXCHANGEREGISTERS system call and appropriate control transfer configuration items CtrlXferConfItem (CtrlXferConfItem , see Section [2.3)](#page30).

默认情况下，内核将使用前面部分中指定的协议。 根据请求，内核将根据控制传输项目切换到扩展协议。 使用EXCHANGEREGISTERSTS系统调用和相应的控制传输配置项CtrlXferConfItem（CtrlXferConfItem，参见2.3节）来执行启用/禁用扩展协议的请求。

**CtrlXfer Item based kernel message protocol**

CtrlXferItem基于内核消息协议

**Extended Thread Start Protocol扩展线程启动协议**

Newly created active threads start immediately by receiving a message from its pager. The received message contains one or more control transfer items:

新创建的活动线程立即从其传呼机接收消息开始。 收到的消息包含一个或多个控制转移项目：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| From Pager |  | CtrlXferItem n | |  |  | MR c0+:::+cn+1 |  |
|  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  | . |  |  | . | | |  |
| . | |  |  | . | | |  |
| . | |  |  | . | | |  |
|  |  | CtrlXferItem 0 | |  |  | MR c0+1 |  |
|  |  |  |  |  |  |  |  |
|  | |  |  |  |  |  |  |
|  | 0 (16/48) |  | 0 (4) |  | u = 0 (6) | MR 0 |  |

**Extended Pagefault Protocol**

To Pager

CtrlXferItem n

.

.

.

CtrlXferItem 0

faulting user-level IP (32/64)

fault address (32/64)

MR c0+:::+cn+1

.

.

.

MR c0+1

MR 2

MR 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 (16/48) | 0 (4) |  | u = 2 (6) | MR 0 |

86 EXTENDED CONTROL TRANSFER PROTOCOL

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Acceptor [BR0 | ] | 0 (22/54) | s = 1 (6) | 0 0 1 0 | BR 0 |  |
|  |  |  |

The acceptor covers the complete user address space and accepts all control transfer items. The kernel accepts mappings or grants into this region on behalf of the faulting thread, and sets the thread state based upon the control transfer items enclosed in the reply message. The received message is discarded.

接受者覆盖完整的用户地址空间并接受所有的控制转移项目。 内核代表错误线程接受映射或授予该区域，并根据回复消息中包含的控制传输项目设置线程状态。 收到的消息被丢弃。

From Pager

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | CtrlXferItem n | | | |  | MR c0+:::+cn+3 |  |
|  |  |  |  |  |  |  |  |  |
| . |  |  |  |  | . | |  |
| . | |  |  |  |  | . | |  |
| . | |  |  |  |  | . | |  |
|  |  | CtrlXferItem 0 | | | |  | MR c0+3 |  |
|  |  |  |  |  |  |  | MR 1;2 |  |
|  |  |  |  |  |  |  |  |
|  |  | MapItem / GrantItem | | | |  |  |
|  |  |  |  |  |  |  |  |  |
|  | |  |  |  |  |  |  |  |
|  | 0 (16/48) |  | 0 (4) | t= 2 + | u = 0 (6) |  | MR 0 |  |

**Extended Exception Protocol**

To Exception Handler

CtrlXferItem n

.

.

.

CtrlXferItem 0

exception error code (32/64)

exception number (32/64)

MR c0+:::+cn+1

.

.

.

MR c0+1

MR 2

MR 1

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 0 (16/48) | 0 (4) |  | | u = | | 2 (6) | MR 0 |  |
| Acceptor [BR0 | ] |  |  |  |  |  |  |  | BR 0 |  |
| 0 (22/54) |  |  | s = 1 (6) |  | 0 | 0 1 0 |  |
|  |  |  |  |  |  |

The acceptor covers the complete user address space and accepts all control transfer items. The kernel accepts mappings or grants into this region on behalf of the faulting thread, and sets the thread state based upon the control transfer items enclosed in the reply message. The received message is discarded.

From Exception Handler

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EXTENDED CONTROL TRANSFER PROTOCOL | | |  |  |  |  |  |  |  | 87 | |  |
|  |  |  |  |  |  |  |  |  |  |  | MR c0+:::+cn+3 |  |
|  |  |  | CtrlXferItem n | |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | . |  |  |  |  |  |  |  | . | |  |
|  | . | |  |  |  |  |  |  |  | . | |  |
|  | . | |  |  |  |  |  |  |  | . | |  |
|  |  |  | CtrlXferItem 0 | |  |  |  |  |  |  | MR c0+3 |  |
|  |  |  | |  | |  |  |  |  |  | MR 1;2 |  |
|  |  |  | |  | |  |  |  |  |  |  |
|  |  | MapItem / GrantItem | | | | | |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | |  |  |  |  |  |  |  |  |  |  |
|  |  | 0 (16/48) |  | 0 (4) | t | t= 2 + | | u = 0 (6) | |  | MR 0 |  |
| **Extended Preemption Protocol** | | |  |  |  |  |  |  |  |  |  |  |
| To Scheduler |  |  |  |  |  |  |  |  |  |  | MR c0+:::+cn+1 |  |
|  |  | CtrlXferItem n | |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | . |  |  |  |  |  |  |  | . | |  |
|  | . | |  |  |  |  |  |  |  | . | |  |
|  | . | |  |  |  |  |  |  |  | . | |  |
|  |  |  | CtrlXferItem 0 | |  |  |  |  |  |  | MR c0+1 |  |
|  |  |  | | | | |  |  |  |  | MR 2 |  |
|  |  |  | | | | |  |  |  |  |  |
|  |  | Clock =2(32/64) (32/64) | | | | | |  |  |  |  |
|  |  |  | | | | |  |  |  |  | MR 1 |  |
|  |  |  | | | | |  |  |  |  |  |
|  |  | Clock mod2(32/64) (32/64) | | | | | |  |  |  |  |
|  |  |  |  | |  |  |  |  |  |  |  |  |
|  |  | |  | |  |  |  |  |  |  |  |  |
|  |  | 0 (16/48) |  | 0 (4) |  | t = | | u = 2 (6) | |  | MR 0 |  |
| Acceptor [BR0 | ] |  |  |  |  |  |  |  |  |  | BR 0 |  |
| 0 (22/54) |  |  |  |  | s = 1 (6) |  | 0 0 1 0 |  |  |
|  |  |  |  |  |  |  |  |  |

The acceptor covers the complete user address space and accepts all control transfer items. The kernel accepts mappings or grants into this region on behalf of the faulting thread, and sets the thread state based upon the control transfer items enclosed in the reply message. The received message is discarded.

From Scheduler

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | CtrlXferItem n | | | |  | MR c0+:::+cn+3 |  |
|  |  |  |  |  |  |  |  |
| . |  |  |  |  | . | |  |
| . |  |  |  |  | . | |  |
| . |  |  |  |  | . | |  |
|  | CtrlXferItem 0 | | | |  | MR c0+3 |  |
|  |  |  |  |  |  | MR 1;2 |  |
|  |  |  |  |  |  |  |
|  | MapItem / GrantItem | | | |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| 0 (16/48) |  | 0 (4) | t= 2 + | u = 0 (6) |  | MR 0 |  |

88 EXTENDED CONTROL TRANSFER PROTOCOL

|  |  |
| --- | --- |
| SIGMA0 RPC PROTOCOL | 89 |

|  |  |  |
| --- | --- | --- |
| **7.7** | | **Sigma0 RPC protocol [Protocol]** |
|  | Sigma0 is the initial address space. Although it is not part of the kernel, its basic protocol is defined with the kernel. Specific | |
| Sigma0 implementations may extend this protocol. | | |
|  | The address space 0 is idempotent, i.e., all virtual addresses in this address space are identical to the corresponding | |

physical address. Note that pages requested from 0 continue to be mapped idempotently if the receiver specifies its complete address space as receive fpage.

0 gives pages to the kernel and to arbitrary tasks, but only once. The idea is that all pagers request the memory they need in the startup phase of the system so that afterwards 0 has exhausted all its memory. Further requests will then automatically be denied.

Sigma 0是初始地址空间。 尽管它不是内核的一部分，但它的基本协议是由内核定义的。 具体的Sigma 0实现可以扩展这个协议。  
地址空间Sigma0是幂等的，即该地址空间中的所有虚拟地址与相应的物理地址相同。 请注意，如果接收方将其完整地址空间指定为接收fpage，则从Sigma0请求的页面将继续被幂等映射。   
0给页面内核和任意任务，但只有一次。 这个想法是，所有的调页程序在系统的启动阶段都要求他们需要的存储器，以致于之后0已经耗尽了所有的存储器。 其他请求将自动被拒绝。

**Kernel Protocol**

To 0

requested fpage

s = 0

s 不= 0

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | ~(32/64) | | |  |  |  |  | MR 2 |  |
|  |  |  |  |  |  |  |  |  | MR 1 |  |
|  | requested fpage (32/64) | | | |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | |  |
| -6 (12/44) |  | 0 (4) | 0 (4) |  | t = 0 (6) | | u = 2 (6) | | MR 0 |  |
|  |  | | | |  |  |  |  | |  |
|  | -1 (22/54) | | | |  | s (6) |  | 0 r w x |  |  |

Kernel requests the amount of memory recommended by Sigma0 for kernel use (pagetable and other kernel-internal data).

内核请求Sigma0推荐的内存使用量（可分页和其他内核内部数据）。

Kernel requests an fpage of size 2s. The fpage can be located at an arbitrary position but must contain ordinary memory. If a free fpage of size 2s is available, it is granted to the kernel.

内核请求大小为2s的fpage。 fpage可以位于任意位置，但必须包含普通内存。 如果大小为2s的空闲页面可用，则授予内核。

rwx The rwx bits are ignored. Sigma0 always grants fpages with maximum access rights to the kernel.

rwx位被忽略。 Sigma0总是授予对内核最大访问权限的权限。

From 0

Kernel memory recommendation内核内存建议

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 (32/64) | |  |  | MR 2 |  |
|  |  |  |  |  | MR 1 |  |
|  | amount (32/64) | | | |  |
|  |  |  |  |  | MR 0 |  |
| 0 (16/48) |  | 0 (4) | t = 0 (6) | u = 2 (6) |  |

amount Amount of memory recommended for kernel use (in bytes).

建议内核使用的内存量（以字节为单位）。

Grant Response

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | GrantItem | | | | MR 1;2 |  |
|  |  |  |  |  | MR 0 |  |
|  |  |  |  |  |  |
| 0 (16/48) |  | 0 (4) | t = 2 (6) | u = 0 (6) |  |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 90 |  |  |  |  |  |  |  | SIGMA0 RPC PROTOCOL | |  |
| Grant Reject |  |  |  |  |  |  |  |  | MR 2 |  |
|  |  | nilpage (32/64) | | |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | MR 1 |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  | 0 (28=60) | |  |  |  |  | 1 0 1 0 |  |
|  | |  |  |  |  |  |  | | MR 0 |  |
|  | 0 (16/48) |  |  | 0 (4) |  | t = 2 (6) | u = 0 (6) | |  |
| **User Protocol** | |  |  |  |  |  |  |  |  |  |
| To 0 |  |  |  |  |  |  |  |  | MR 3 |  |
|  | high address (32/64) | | | |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  | | | |  |  | MR 2 |  |
|  |  |  |  | | | |  |  |  |
|  |  | requested attributes (32/64) | | | | |  |  |  |
|  |  |  |  | | |  |  |  | MR 1 |  |
|  |  |  |  | | |  |  |  |  |
|  |  | requested fpage (32/64) | | | |  |  |  |  |
|  |  |  |  | |  |  |  | | MR 0 |  |
|  |  |  |  | |  |  |  | |  |
|  | -6 (12/44) |  | 0 (4) | 0 (4) |  | t = 0 (6) | u = 2 (6) | |  |

requested fpage

b 不=-1

|  |  |  |  |
| --- | --- | --- | --- |
| b/210 (22/54) | s (6) | e | r w x |

0 deals with fpages of arbitrary size. A successful response from 0 contains an fpage of physically contiguous memory.

Sigma0涉及任意大小的fpages。 从Sigma0开始的成功响应包含fpage的物理连续内存。

Requests the specific fpage with base address b and size 2s. If the fpage is neither owned by the kernel nor by a user thread (not even partially), the requested fpage is mapped to the requestor’s address space and the fpage is marked as owned by the requesting thread (i.e., fpage is not marked as being owned by the address space in which thread resides). Any fpage not belonging to reserved memory (see page [93)](#page105) can be requested. If the requested fpage is already owned by the requestor only the page attributes are modified. No new mapping operation happens.

请求具有基地址b和大小是2s的特定fpage。 如果fpage既不属于内核也不属于用户线程（甚至不属于部分属性），则请求的fpage将映射到请求者的地址空间，并且fpage被标记为请求线程所拥有（即，fpage未被标记为 由线程所在的地址空间所拥有）。 可以请求任何不属于保留内存的fpage（请参见第93页）。 如果请求的fpage已经被请求者拥有，则只修改页面属性。 没有新的映射操作发生。

b = -1

e

Requests an fpage of size 2s but with arbitrary address. If a free fpage of size 2s is available, it is mapped to the requestor’s address space and marked as owned by the requesting thread (i.e., fpage is not marked as being owned by the address space in which thread resides). 0 is free to use the requested-attribute for choosing a best fitting page. Only fpages belonging to conventional memory (see page [93)](#page105) are considered free and handed out upon such anonymous requests.

请求大小为2s的fpage，但具有任意地址。 如果大小为2s的空闲页面可用，则将其映射到请求者的地址空间并被标记为由请求线程所拥有（即，fpage未被标记为由线程驻留的地址空间所拥有）。 Sigma0可以自由使用请求属性来选择一个最合适的页面。 只有属于传统内存（参见第93页）的fpages被认为是空闲的，并根据这些匿名请求分发。

Setting this bit to 1 instructs 0 to map an address longer than the usual address size of the system (e.g. a 64 bit address on a 32 bit system). In this case, the lowermost bits of the requested address are specified as ususal in the requested fpage field, while the highermost bits are specified in a separate message register (see high address below).

将该位设置为1将指示Sigma0映射比系统常用地址长度更长的地址（例如，32位系统上的64位地址）。 在这种情况下，所请求的地址的最低位被指定为所请求的fpage字段中的ususal，而最高位被指定在单独的消息寄存器中（见下面的高地址）。

rwx The rwx bits are ignored. Sigma0 always maps fpages with maximum access rights to the requestor.

requested attributes

= 0

不= 0

high address

The page is requested with default attributes.

该页面以默认属性请求。

The page is requested with some architecture dependent attributes.

该页面被请求与一些架构相关的属性。

this field contains the part of the requested address that didn’t fit in the requested fpage field. 0 concatenates this field with the base address of the requested fpage field and then tries to map the result into the requesters address space. Note that this field will not be included in the response’s MapItem.

This field is only read if the e bit is set to 1. If the e bit is 0, this field is ignored.

该字段包含所请求的地址部分，不符合请求的fpage字段。 Sigma0将此字段连接到所请求的fpage字段的基址，然后尝试将结果映射到请求者地址空间。 请注意，此字段不会包含在响应的MapItem中。  
只有当e位设置为1时才读取该字段。如果e位为0，则忽略该字段。

From Sigma0

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SIGMA0 RPC PROTOCOL | |  |  |  |  |  | 91 |  |
| Map Response |  |  |  |  |  |  | MR 1;2 |  |
|  | MapItem | |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |  | MR 0 |  |
|  | |  |  |  |  |  |  |
|  | 0 (16/48) |  | 0 (4) | t = 2 (6) | u = 0 (6) | |  |
| Map Reject |  |  |  |  |  |  | MR 2 |  |
|  | nilpage (32/64) | |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |  | MR 1 |  |
|  |  |  |  |  |  |  |  |
|  |  | 0 (28=60) |  |  |  | 1 0 0 0 |  |
|  | |  | |  |  | | MR 0 |  |
|  | 0 (16/48) |  | 0 (4) | t = 2 (6) | u = 0 (6) | |  |

0 responds with a map reject message if the page is reserved (i.e., kernel space) or already mapped to a different thread, or if memory is exhausted.

如果页面被保留（即，内核空间）或者已经被映射到不同的线程，或者如果内存耗尽，Sigma0响应一个映射拒绝消息。

**Pagefault Protocol**

0 also understands the pagefault protocol (see page [82)](#page94) and will convert pagefault requests into 0 user protocol requests. Further, only memory marked as conventional memory (see page [93)](#page105) can be requested using the pagefault protocol. Any non-conventional memory (including boot loader specific memory) must be requested explicitly using the regular 0 protocol.

Sigma0也理解页面错误协议（参见第82页），并将页面错误请求转换为Sigma0用户协议请求。 此外，只有使用pagefault协议才能请求标记为常规内存（请参见第93页）的内存。 任何非常规内存（包括引导加载程序特定内存）都必须使用常规Sigma0协议明确请求。

Incoming pagefault message

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | faulting user-level IP (32/64) | | | | | MR 2 |  |
|  |  |  |  |  |  | MR 1 |  |
|  |  |  |  |  |  |  |
|  | fault address (32/64) | | | | |  |
|  |  |  |  |  |  | MR 0 |  |
|  |  |  |  |  |  |  |
| -2 (12/44) |  | 0 r w x | 0 (4) | t = 0 (6) | u = 2 (6) |  |

Converted pagefault message

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 0 (32/64) | |  |  |  |  | MR 2 |  |
|  |  |  |  |  |  |  |  | MR 1 |  |
| fault address=210 | | (22/54) | |  | s (6) |  | 0 0 0 0 |  |
|  |  |  |  |  | |  | | MR 0 |  |
| -6 (12/44) |  | 0 (4) | 0 (4) | t = 0 (6) | | u = 2 (6) | |  |

* The minimum supported page size as defined by the PageInfo field in the kernel interface page (see page [3)](#page15).

由内核接口页面中的PageInfo字段定义的最小支持页面大小（请参见第3页）。

92 GENERIC BOOTING

**7.8 Generic Booting通用引导** **[Protocol]**

Machine-specific boot procedures are described on pages [112](#page124) ff.

After booting, L4 initializes itself. It generates the basic address space-servers 0, 1 and a root server which is intended to boot the higher-level system.

0, 1 and the root server are user-level servers and not part of the pure kernel. The predefined ones can be replaced by modifying the following table in the L4 image before starting L4. An empty area specifies that the corresponding server should not be started. Note, that 0 is a mandatory service. The kernel debugger kdebug is also not part of the kernel and can accordingly be replaced by modifying the table.

特定于计算机的引导过程在第112页及以后介绍。  
  
启动后，L4自行初始化。 它生成基本地址空间 - 服务器Sigma0,Sigma1和用于启动更高级别系统的根服务器。  
Sigma0,Sigma1和根服务器是用户级服务器，不是纯内核的一部分。 在启动L4之前，可以通过修改L4映像中的下表来替换预定义的映像。 空白区域指定不应启动相应的服务器。 请注意，Sigma0是强制服务。 内核调试器kdebug也不是内核的一部分，因此可以通过修改表来替换。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | MemoryDesc | | | | |  | MemDescPtr | | |  |
|  |  |  |  |  |  |  |  |  |  |  | +B0 | / +160 | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
| ~ |  | BootInfo | ~ | | | | | | | |  |
| ~ | | | | | | | | | | | +A0 | / +140 | |  |
| ~ | | | | | | | | | | | +90 | / +120 | |  |
| ~ | | | | | | | | | | | +80 | / +100 | |  |
| ~ | | | | | | | | | | | +70 / | | +E0 |  |
| ~ | | | | | | | | | | | +60 / | | +C0 |  |
| Kdebug.config1 |  | Kdebug.config0 |  | MemoryInfo |  | ~ | | | |  | +50 / | | +A0 |  |
| root server.high |  | root server.low |  | root server.IP |  | root server.SP | | | |  | +40 | / | +80 |  |
|  |  |  |  |  |  |  |  |  |  |  | +30 | / | +60 |  |
| Sigma1.high |  | Sigma 1.low |  | Sigma 1.IP |  | Sigma 1.SP | | | |  |  |
|  |  |  |  |  |  |  |  |  |  |  | +20 | / | +40 |  |
| Sigma 0.high |  | Sigma 0.low |  | Sigma 0.IP |  | Sigma 0.SP | | | |  |  |
|  |  |  |  |  |  |  |  |  |  |  | +10 | / | +20 |  |
| Kdebug.high |  | Kdebug.low |  | Kdebug.entry |  | Kdebug.init | | | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  | +0 |  |
| ~ | | |  | API Version |  | ~(0=32) | ’K’ | 230 | ’4’ | ’L’ |  |  |  |
| +C / +18 | | +8 / +10 |  | +4 / +8 | |  |  |  |  | +0 |  |  |  |  |

The addresses are offsets relative to the configuration page’s base address. The configuration page is located at a page boundary and can be found by searching for the magic “L4 K” starting at the load address. The IP and SP values however, are absolute addresses. The appropriate code must be loaded at these addresses before L4 is started.

地址是相对于配置页面的基地址的偏移量。 配置页面位于页面边界，可以从加载地址开始查找魔术“L4 K”。 然而，IP和SP值是绝对地址。 在启动L4之前，必须在这些地址加载适当的代码。

IP Physical address of a server’s initial instruction pointer (start).

服务器初始指令指针的物理地址（开始）。

SP Physical address of a server’s initial stack pointer (stack bottom).

服务器初始堆栈指针的物理地址（堆栈底部）

Kdebug.init Physical address of kdebug’s initialization routine.

kdebug初始化例程的物理地址。

|  |  |
| --- | --- |
| GENERIC BOOTING | 93 |

Kdebug.entry Physical address of kdebug’s exception handler entry point.

kdebug异常处理程序入口点的物理地址。

Kdebug.low Physical address of first byte of kernel debugger. Must be page aligned.

内核调试器的第一个字节的物理地址。 必须页面对齐。

Kdebug.high Physical address of last byte of kernel debugger. Must be the last byte in page.

内核调试器的最后一个字节的物理地址。 必须是页面中的最后一个字节。

Kdebug.config Configuration fields which can be freely interpreted by the kernel debugger. The specific seman-

tics of these fields are provided with the specific kernel debuggers.

内核调试器可以自由解释的配置字段。 这些字段的特定语义与特定的内核调试器一起提供。

BootInfo Prior to kernel initialization a boot loader can write an arbitrary value into this field. Post-

initialization code, e.g., a root server can later read the field. Its value is neither changed nor

interpreted by the kernel. This is the generic method for passing system information across

kernel initialization.

在内核初始化之前，引导加载程序可以在该字段中写入任意值。 后初始化代码（例如，根服务器）可以稍后读取该字段。 它的值既不被内核改变，也不被内核解释。 这是通过内核初始化传递系统信息的通用方法。

MemoryInfo

MemDescPtr

n

MemoryDesc

high

low

v

type

|  |  |
| --- | --- |
| MemDescPtr (16/32) | n (16/32) |

Location of first memory descriptor (as an offset relative to the configuration page’s base ad-dress). Subsequent memory descriptors are located directly following the first one. For memory descriptors that specify overlapping memory regions, later descriptors take precedence over ear-lier ones.

第一个内存描述符的位置（作为相对于配置页面的基本广告的偏移量）。 后续的内存描述符直接位于第一个之后。 对于指定重叠内存区域的内存描述符，后面的描述符优先于内存描述符。

Initially equals the number of available memory descriptors in the configuration page. Before starting L4 this number must be initialized to the number of inserted memory descriptors.

最初等于配置页面中可用内存描述符的数量。 在开始L4之前，这个数字必须被初始化为插入的内存描述符的数量。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| High/210 | (22/54) |  |  | ~(10) | | +4 / +8 |  |
|  |  |  |  |  |  | +0 |  |
| Low/210 | (22/54) | v | ~ | t (4) | type (4) |  |

Memory descriptors should be initialized before starting L4. The kernel may after startup insert additional memory descriptors or modify existing ones (e.g., for reserved kernel memory).

内存描述符在启动L4之前应该被初始化。 内核可能在启动之后插入额外的内存描述符或修改现有的描述符（例如，用于保留的内核内存）。

Address of last byte in memory region. The ten least significant address bits are all hardwired to 1.

内存区域中最后一个字节的地址。 十个最低有效地址位全部硬连线到1。

Address of first byte in memory region. The ten least significant address bits are all hardwired to 0.

内存区域中第一个字节的地址。 十个最低有效地址位全部硬连线到0。

Indicates whether memory descriptor refers to physical memory (v = 0) or virtual memory (v = 1).

指示内存描述符是指物理内存（v = 0）还是虚拟内存（v = 1）。

Identifies the type of the memory descriptor.

标识内存描述符的类型。

|  |  |
| --- | --- |
| Type | Description |
| 0x0 | Undefined |
| 0x1 | Conventional memory |
| 0x2 | Reserved memory (i.e., reserved by kernel) |
| 0x3 | Dedicated memory (i.e memory not available to user) 专用内存 |
| 0x4 | Shared memory (i.e., available to all users) |
| 0xE | Defined by boot loader |
| 0xF | Architecture dependent结构依赖 |

t Identifies the precise type for boot loader specific or architecture dependent memory descriptors.

标识引导加载程序特定或体系结构相关的内存描述符的精确类型。

94 GENERIC BOOTING

type = 0xE

The type of the memory descriptor is dependent on the bootloader. The t field specifies the exact semantics. Refer to boot loader specification for more info.

内存描述符的类型取决于引导加载程序。 t字段指定确切的语义。 请参阅引导装载程序规范了解更多信息。

type = 0xF

The type of the memory descriptor is architecture dependent. The t field specifies the exact semantics. Refer to architecture specific part for more info (see page ??).

内存描述符的类型取决于体系结构。 t字段指定确切的语义。 有关更多信息，请参阅体系结构特定部分（请参阅第页）。

type 不= 0xE, type 不= 0xF

The type of the memory descriptor is solely defined by the type field. The content of the t field is undefined.

内存描述符的类型完全由类型字段定义。 t字段的内容是未定义的。

**Appendix A**

IA-32 Interface

96 VIRTUAL REGISTERS

**A.1 Virtual Registers** **[ia32]**

**Thread Control Registers (TCRs)**

TCRs are implemented as part of the ia32-specific user-level thread control block (UTCB). The address of the current thread’s UTCB will not change over the lifetime of the thread. Setting the UTCB address of an active thread via THREAD-CONTROL is similar to deletion and re-creation. There is a fixed correlation between the UtcbLocation parameter when invoking THREADCONTROL and the UTCB address. The UTCB address of the current thread can be loaded through a machine instruction

mov %gs:[0], %r

UTCB objects of the current thread can then be accessed as any other memory object. UTCBs of other threads must not be accessed, even if they are physically accessible. ThreadWord0 and ThreadWord1 are free to be used by systems software (e.g., IDL compilers). The kernel associates no semantics with these words.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | (32) | |  |  | UTCB address |  |
| . |  |  |  | . |  |  |
| . |  |  |  | . |  |  |
| . |  |  |  | . |  |  |
|  | ThreadWord 0 (32) | |  |  | –16 |  |
|  |  |  |  |  | –20 |  |
|  | ThreadWord 1 (32) | |  |  |  |
|  |  |  |  |  | –24 |  |
|  | VirtualSender/ActualSender (32) | |  |  |  |
|  |  |  |  |  | –28 |  |
|  | IntendedReceiver (32) | |  |  |  |
|  |  |  |  |  | –32 |  |
|  | XferTimeouts (32) | |  |  |  |
|  |  |  |  |  | –36 |  |
|  | ErrorCode (32) | |  |  |  |
|  |  |  |  |  | –40 |  |
| (16) |  | cop flags (8) | preempt flags (8) |  |  |
|  | ExceptionHandler (32) | |  |  | –44 |  |
|  |  | |  |  | –48 |  |
|  | Pager (32) | |  |  |  |
|  |  | |  |  | –52 |  |
|  | UserDefinedHandle (32) | |  |  |  |
|  |  | |  |  | –56 |  |
|  | ProcessorNo (32) | |  |  |  |
|  |  | |  |  | –60 |  |
|  | MyGlobalId (32) | |  |  |  |
|  |  |  |  |  |  |  |

|  |  |
| --- | --- |
| MyLocalId = UTCB address (32) | gs:[0] |
|  |  |

The TCR MyLocalId is not part of the UTCB. On ia32 it is identical with the UTCB address and can be loaded from memory location gs:[0].

|  |  |
| --- | --- |
| VIRTUAL REGISTERS | 97 |

**Message Registers (MRs)**

Memory-mapped MRs are implemented as part of the ia32-specific user-level thread control block (UTCB). The address of the current thread’s UTCB will not change over the lifetime of the thread. Setting the UTCB address of an active thread via THREADCONTROL is similar to deletion and re-creation. There is a fixed correlation between the UtcbLocation parameter when invoking THREADCONTROL and the UTCB address. The UTCB address of the current thread can be loaded through a machine instruction

mov %gs:[0], %r

UTCB objects of the current thread can then be accessed as any other memory object. UTCBs of other threads must not be accessed, even if they are physically accessible.

MR 0 is always mapped to a general register. MR 1 and MR 2 are mapped to general registers when reading a received message; in all other cases, MR 1 and MR 2 are mapped to memory locations. MR 3:::63 are always mapped to memory.

|  |  |  |
| --- | --- | --- |
| MR 0 | ESI |  |
|  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| MR 1 | (only for msg receive) | |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  | EBX |  |  |  |
| MR 2 |  |  |  |  |  |  |
| (only for msg receive) | |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  | EBP |  |  |  |
|  | |  |  |  |  |  |
| MR 1:::63 [UTCB fields] | | |  |  |  |  |
|  |  |  |  |  | |  |
|  |  |  | MR 63 (32) |  | +252 |  |
|  |  | . |  | . |  |  |
|  | . | |  | . |  |  |
|  | . | |  | . |  |  |
|  |  |  | MR 4 (32) |  | +16 |  |
|  |  |  | MR 3 (32) |  | +12 |  |
|  |  | MR 2 | (except for msg receive) (32) |  | +8 |  |
|  |  |  |  |  | UTCB address + 4 |  |
|  |  | MR 1 | (except for msg receive) (32) |  |  |

**Buffer Registers (BRs)**

BRs are implemented as part of the ia32-specific user-level thread control block (UTCB). The address of the current thread’s UTCB will not change over the lifetime of the thread. Setting the UTCB address of an active thread via THREAD-CONTROL is similar to deletion and re-creation. There is a fixed correlation between the UtcbLocation parameter when invoking THREADCONTROL and the UTCB address. The UTCB address of the current thread can be loaded through a machine instruction

mov %gs:[0], %r

UTCB objects of the current thread can then be accessed as any other memory object. UTCBs of other threads must not be accessed, even if they are physically accessible.

98

BR 0:::32 [UTCB fields]

(32)

.

.

.

BR 0 (32)

BR 1 (32)

.

.

.

BR 32 (32)

VIRTUAL REGISTERS

UTCB address

.

.

.

–64

–68

.

.

.

–196

**UTCB Memory With Undefined Semantics**

The kernel will associate no semantics with memory located at UTCB address. . . UTCB address + 3. The application can use this memory as thread local storage, e.g., for implementing the L4 API. Note, however, that the memory contents within this region may be overwritten during a system-call operating on message registers.

All undefined UTCB memory which is not covered by the above mentioned region may have kernel defined semantics.

|  |  |
| --- | --- |
| SYSTEMCALLS | 99 |

**A.2 Systemcalls** **[ia32]**

The system-calls which are invoked by the call instruction take the target of the calls from the system-call link fields in the kernel interface page (see page [2)](#page14). Each system-call link specifies an address relative to the kernel interface page’s base address. An application may use instructions other than call to invoke the system-calls, but must ensure that a valid return address resides on the stack.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **KERNELINTERFACE** | **[Slow Systemcall]** | | |  |  |  |  |
| – | EAX |  | KernelInterface ! |  | EAX | base address |  |
|  |  |  |
| – | ECX |  |  |  | ECX | API Version |  |
| – | EDX |  | lock: nop |  | EDX | API Flags |  |
| – | ESI |  |  | ESI | Kernel ID |  |
| – | EDI |  |  |  | EDI |  |  |
| – | EBX |  |  |  | EBX |  |  |
| – | EBP |  |  |  | EBP |  |  |
| – | ESP |  |  |  | ESP |  |  |
|  | | | |  |  |  |  |
| **EXCHANGEREGISTERS [Systemcall]** | | | |  |  |  |  |
| dest | EAX |  | Exchange Registers ! |  | EAX | result |  |
|  |  |  |
| control | ECX |  |  |  | ECX | control |  |
| SP | EDX |  | call ExchangeRegisters |  | EDX | SP |  |
| IP | ESI |  |  | ESI | IP |  |
| FLAGS | EDI |  |  |  | EDI | FLAGS |  |
| UserDefinedHandle | EBX |  |  |  | EBX | UserDefinedHandle |  |
| pager | EBP |  |  |  | EBP | pager |  |
| – | ESP |  |  |  | ESP |  |  |

“FLAGS” refers to the user-modifiable ia32 processor flags that are held in the EFLAGS register.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **THREADCONTROL** | **[Privileged Systemcall]** | | |  |  |  |  |
| dest | EAX |  | Thread Control ! |  | EAX | result |  |
|  |  |  |
| Pager | ECX |  |  |  | ECX |  |  |
| Scheduler | EDX |  | call ThreadControl |  | EDX |  |  |
| SpaceSpecifier | ESI |  |  | ESI |  |  |
| UtcbLocation | EDI |  |  |  | EDI |  |  |
| – | EBX |  |  |  | EBX |  |  |
| – | EBP |  |  |  | EBP |  |  |
| – | ESP |  |  |  | ESP |  |  |
|  | | | |  |  |  |  |
| **SYSTEMCLOCK [Systemcall]** | | | |  |  |  |  |
| – | EAX |  | SystemClock ! |  | EAX | clock 0. . . 31 |  |
|  |  |  |
| – | ECX |  |  |  | ECX |  |  |
| – | EDX |  | call SystemClock |  | EDX | clock 32. . . 63 |  |
| – | ESI |  |  | ESI |  |  |
| – | EDI |  |  |  | EDI |  |  |
| – | EBX |  |  |  | EBX |  |  |
| – | EBP |  |  |  | EBP |  |  |
| – | ESP |  |  |  | ESP |  |  |
|  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 100 |  |  |  |  |  |  | SYSTEMCALLS |  |
| **THREADSWITCH [Systemcall]** | | | | |  |  |  |  |
|  | dest | EAX |  | ThreadSwitch ! |  | EAX |  |  |
|  |  |  |  |
|  | – | ECX |  |  |  | ECX |  |  |
|  | – | EDX |  | call ThreadSwitch |  | EDX |  |  |
|  | – | ESI |  |  | ESI |  |  |
|  | – | EDI |  |  |  | EDI |  |  |
|  | – | EBX |  |  |  | EBX |  |  |
|  | – | EBP |  |  |  | EBP |  |  |
|  | – | ESP |  |  |  | ESP |  |  |
|  | | |  |  |  |  |  |  |
| **SCHEDULE [Systemcall]** | | |  |  |  |  |  |  |
|  | dest | EAX |  | Schedule ! |  | EAX | result |  |
|  |  |  |  |
|  | prio | ECX |  |  |  | ECX |  |  |
|  | time control | EDX |  | call Schedule |  | EDX | time control |  |
|  | processor control | ESI |  |  | ESI |  |  |
|  | preemption control | EDI |  |  |  | EDI |  |  |
|  | – | EBX |  |  |  | EBX |  |  |
|  | – | EBP |  |  |  | EBP |  |  |
|  | – | ESP |  |  |  | ESP |  |  |
|  |  |  |  |  |  |  |  |  |
| **IPC** | **[Systemcall]** |  |  |  |  |  |  |  |
|  | to | EAX |  | Ipc ! |  | EAX | from |  |
|  |  |  |  |
|  | Timeouts | ECX |  |  |  | ECX |  |  |
|  | FromSpecifier | EDX |  | call Ipc |  | EDX |  |  |
|  | MR 0 | ESI |  |  | ESI | MR 0 |  |
|  | UTCB | EDI |  |  |  | EDI |  |  |
|  | – | EBX |  |  |  | EBX | MR 1 |  |
|  | – | EBP |  |  |  | EBP | MR 2 |  |
|  | – | ESP |  |  |  | ESP |  |  |
|  |  |  |  |  |  |  |  |  |
| **LIPC** | **[Systemcall]** |  |  |  |  |  |  |  |
|  | to | EAX |  | Lipc ! |  | EAX | from |  |
|  |  |  |  |
|  | Timeouts | ECX |  |  |  | ECX |  |  |
|  | FromSpecifier | EDX |  | call Lipc |  | EDX |  |  |
|  | MR 0 | ESI |  |  | ESI | MR 0 |  |
|  | UTCB | EDI |  |  |  | EDI |  |  |
|  | – | EBX |  |  |  | EBX | MR 1 |  |
|  | – | EBP |  |  |  | EBP | MR 2 |  |
|  | – | ESP |  |  |  | ESP |  |  |
|  | |  |  |  |  |  |  |  |
| **UNMAP [Systemcall]** | |  |  |  |  |  |  |  |
|  | control | EAX |  | Unmap ! |  | EAX |  |  |
|  |  |  |  |
|  | – | ECX |  |  |  | ECX |  |  |
|  | – | EDX |  | call Unmap |  | EDX |  |  |
|  | MR 0 | ESI |  |  | ESI | MR 0 |  |
|  | UTCB | EDI |  |  |  | EDI |  |  |
|  | – | EBX |  |  |  | EBX |  |  |
|  | – | EBP |  |  |  | EBP |  |  |
|  | – | ESP |  |  |  | ESP |  |  |
|  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SYSTEMCALLS |  |  |  |  |  |  |  |  |  |  | 101 |  |
| **SPACECONTROL [Privileged Systemcall]** | | | | | | | | |  |  |  |  |
| SpaceSpecifier | EAX |  |  | Space Control ! | | | | |  | EAX | result |  |
|  |  |  |  |
| control | ECX |  |  |  |  |  |  |  |  | ECX | control |  |
| KernelInterfacePageArea | EDX |  |  | call SpaceControl | | | | |  | EDX |  |  |
| UtcbArea | ESI |  |  |  | ESI |  |  |
| Redirector | EDI |  |  |  |  |  |  |  |  | EDI |  |  |
| – | EBX |  |  |  |  |  |  |  |  | EBX |  |  |
| – | EBP |  |  |  |  |  |  |  |  | EBP |  |  |
| – | ESP |  |  |  |  |  |  |  |  | ESP |  |  |
|  | |  |  | |  |  |  |  |  |  |  |  |
| **PROCESSORCONTROL** | | **[Privileged Systemcall]** | | | | | | |  |  |  |  |
| ProcessorNo | EAX |  |  | Processor Control ! | |  |  |  |  | EAX | result |  |
|  |  |  |  |  |  |
| InternalFrequency | ECX |  |  |  |  |  |  |  |  | ECX |  |  |
| ExternalFrequency | EDX |  |  | call ProcessorControl | |  |  |  |  | EDX |  |  |
| voltage | ESI |  |  |  |  |  |  | ESI |  |  |
| – | EDI |  |  |  |  |  |  |  |  | EDI |  |  |
| – | EBX |  |  |  |  |  |  |  |  | EBX |  |  |
| – | EBP |  |  |  |  |  |  |  |  | EBP |  |  |
| – | ESP |  |  |  |  |  |  |  |  | ESP |  |  |
|  |  | | | |  | |  | |  |  |  |  |
| **MEMORYCONTROL** | **[Privileged Systemcall]** | | | | | | | |  |  |  |  |
| control | EAX |  |  | Memory Control ! |  | |  | |  | EAX | result |  |
|  |  |  | |  |  |
| attribute0 | ECX |  |  |  |  |  |  |  |  | ECX |  |  |
| attribute1 | EDX |  |  | call MemoryControl |  |  |  |  |  | EDX |  |  |
| MR 0 | ESI |  |  |  | |  | |  | ESI |  |  |
| UTCB | EDI |  |  |  |  |  |  |  |  | EDI |  |  |
| attribute2 | EBX |  |  |  |  |  |  |  |  | EBX |  |  |
| attribute3 | EBP |  |  |  |  |  |  |  |  | EBP |  |  |
| – | ESP |  |  |  |  |  |  |  |  | ESP |  |  |

102 KERNEL FEATURES

**A.3 Kernel Features** **[ia32]**

The ia32 architecture supports the following kernel feature descriptors in the kernel interface page (see page [5)](#page17).

String Feature

“smallspaces” Kernel has small address spaces enabled.

|  |  |
| --- | --- |
| IO PORTS | 103 |

**A.4 IO Ports** **[ia32]**

**IO Fpages**

On IA-32 processors, IO-ports are handled as fpages. IO fpages can be mapped, granted, and unmapped like memory fpages. Their minimal granularity is 1. An IO-fpage of size 2s0 has a 2s0-aligned base address p, i.e. p mod 2s0=0. An fpage with base port address p and size 2s0 is denoted as described below.

IO fpage (p; 2s0)

|  |  |  |  |
| --- | --- | --- | --- |
| p (16) | s’ (6) | s = 2 (6) | 0 1 1 0 |

IO-ports can only be mapped idempotently, i.e., physical port x is either mapped at IO address x in the task’s IO address space, or it is not mapped at all. There are no distinct rights associated with IO ports, i.e., a task can be granted either read- and write-access to an IO port, ore none at all.

**IO Pagefault Protocol**

A thread generating an IO port exception will cause the kernel to transparently generate an IO-pagefault IPC to the faulting thread’s pager. The behavior of the faulting thread is undefined if the pager does not exactly follow this protocol.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| To Pager |  | faulting user-level IP (32) | | | |  |  |  |  |  | MR 2 |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  | MR 1 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | faulting port (16) |  | size (6) | |  | s = 2 (6) | |  | 0 1 1 0 |  |
|  |  |  |  |  |  |  |  |  |  |  | MR 0 |  |
|  |  |  |  | |  |  | |  |  | |  |
|  |  | 8 (12) | 0 1 1 0 | 0 (4) |  | t = 0 (6) | |  | u = 2 (6) | |  |
| Acceptor [BR0 | ] |  |  |  |  |  |  |  |  |  | BR 0 |  |
| 0 (16) |  | 16 (6) | |  | s = 2 (6) | |  | 0 0 0 0 |  |
|  |  |  |  |  |  |

The acceptor covers the complete IO-address space. The kernel accepts mappings or grants into this region on behalf of the faulting thread. The received message is discarded.

**Generic Programming Interface**

#include <l4/arch.h>

Fpage IoFpage (Word BasePort, int FpageSize)

Fpage IoFpageLog2 (Word BasePort, int Log2FpageSize <= 16)

Delivers an IO fpage with the specified location and size.

Word IoFpagePort (Fpage f)

Word IoFpageSize (Fpage f)

Word IoFpageSizeLog2 (Fpage f)

Delivers port/size of specified IO fpage.

Bool IsIoFpage (Fpage f)

Delivers true if fpage is an IO fpage.

104 SPACE CONTROL

**A.5 Space Control** **[ia32]**

The SPACECONTROL system call has an architecture dependent control parameter to specify various address space char-acteristics. For ia32, the control parameter has the following semantics.

**Input Parameters**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| control | s 0 t | 0 (21) | small (8) |  |
|  |  |

* A value of 1 indicates the intention to change the small address space number for the specified address space. The small space number will remain unchanged if s = 0.
* A value of 1 instructs the kernel to add an entry to the translation table for extended mappings. This table allows mapping of memory addresses longer than 32 bits on 32-bit systems. The desired mapping is specified in the remaining parameters of the SpaceControl system call as follows: The redirector field must contain the highest 32 bits of the desired address, while the utcb area field must contain the lower 32 bits. The kip area field contains a regular fpage, which specifies a region of 32 bit addresses that should be mapped to a 64 bit address. If any address in this fpage is mapped to a thread, the address will be translated to the corresponding 64 bit address. If the mapping is successfull, the translation table entry is deleted.

small If s = 1, sets the small address space number for the specified address space. Small address

space numbers from 1 to 255 are available. A value of 0 indicates a regular large address space.

An assigned small space number is effective on all CPUs in an SMP system.

The position (pos) of the least significant bit of small indicates the size of the small space by the

following formula: size = 2pos 4 MB. After removing the least significant bit, the remaining

bits of small indicate the location of the space within a 512 MB region using the following

formula: location = small 2 MB. Setting the small space number fails if the specified region

overlaps with an already existing one.

The small field is ignored if s = 0, or if the kernel does not support small spaces (see Kernel

Features, page [102)](#page114).

**Output Parameter**

control

e

t

small

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| e | 0 | t | 0 (21) | small (8) |

Indicates if the change of small space number was effective (e = 1). Undefined if s = 0 in the input parameter.

Indicates if an entry was successfully added to the kernel’s translation table for extended map-pings.

The old value for the small space number. A value of 0 is possible even if the space has pre-viously been put into a small address space. An implicit change to small space number 0 can happen if a thread within the space accesses memory beyond the specified small space size.

**Generic Programming Interface**

#include <l4/space.h>

|  |  |
| --- | --- |
| SPACE CONTROL | 105 |

Word LargeSpace

Word SmallSpace (Word location, size)

Delivers a small space number with the specified location and size (both in MB). It is assumed that size = 2p 4 for some value p < 8.

106 CACHEABILITY HINTS

**A.6 Cacheability Hints** **[ia32]**

String items can specify cacheability hints to the kernel (see page [59)](#page71). For ia32, the cacheability hints have the following semantics.

1. = 00
2. = 01
3. = 10
4. = 11

Use the processor’s default cacheability strategy. Typically, cache lines are allocated for data read and written (assuming that the processor’s default strategy is write-back and write-allocate).

Allocate cache lines in the entire cache hierarchy for data read or written.

Do not allocate new cache lines (entire cache hierarchy) for data read or written.

Allocate only new L1 cache line for data read or written. Do not allocate cache lines in lower cache hierarchies.

**Convenience Programming Interface**

#include <l4/ipc.h>

CacheAllocationHint UseDefaultCacheLineAllocation

CacheAllocationHint AllocateNewCacheLines

CacheAllocationHint DoNotAllocateNewCacheLines

CacheAllocationHint AllocateOnlyNewL1CacheLines

|  |  |
| --- | --- |
| MEMORY ATTRIBUTES | 107 |

**A.7 Memory Attributes** **[ia32]**

The ia32 architecture in general supports the following memory attributes values.

|  |  |
| --- | --- |
| attribute | value |
| Default | 0 |
| Write Back | 1 |
| Write Through | 2 |
| Uncacheable | 4 |
| Write Combining | 5 |
| Write Protected | 8 |

Note that some attributes are only supported on certain processors. See the “IA-32 Intel Architecture Software Devel-oper’s Manual, Volume 3: System Programming Guide” for the semantics of the memory attributes and which processors they are supported on.

**Generic Programming Interface**

#include <l4/misc.h>

Word DefaultMemory

Word WriteBackMemory

Word WriteThroughMemory

Word UncacheableMemory

Word WriteCombiningMemory

Word WriteProtectedMemory

108 EXCEPTION MESSAGE FORMAT

**A.8 Exception Message Format** **[ia32]**

To Exception Handler

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | EAX (32) | |  |  | MR 12 |  |
|  |  |  |  |  | MR 11 |  |
|  | ECX (32) | |  |  |  |
|  |  |  |  |  | MR 10 |  |
|  | EDX (32) | |  |  |  |
|  |  |  |  |  | MR 9 |  |
|  | EBX (32) | |  |  |  |
|  |  |  |  |  | MR 8 |  |
|  | ESP (32) | |  |  |  |
|  |  |  |  |  | MR 7 |  |
|  | EBP (32) | |  |  |  |
|  |  |  |  |  | MR 6 |  |
|  | ESI (32) | |  |  |  |
|  |  |  |  |  | MR 5 |  |
|  | EDI (32) | |  |  |  |
|  |  |  |  |  | MR 4 |  |
|  | ErrorCode (32) | |  |  |  |
|  |  |  |  |  | MR 3 |  |
|  |  |  |  |  |  |
|  | ExceptionNo (32) | |  |  |  |
|  |  |  |  |  | MR 2 |  |
|  |  |  |  |  |  |
|  | EFLAGS (32) | |  |  |  |
|  |  |  |  |  | MR 1 |  |
|  |  |  |  |  |  |
|  | EIP (32) | |  |  |  |
|  |  |  |  |  | |  |
| 4= 5 (12/44) | 0 (4) | 0 (4) | t = 0 (6) | u = 12 (6) | MR 0 |  |

#PF (page fault), #MC (machine check exception), and some #GP (general protection), #SS (stack segment fault), and #NM (no math coprocessor) exceptions are handled by the kernel and therefore do not generate exception messages.

Note that executing an INT n instructions in 32-bit mode will always raise a #GP (general protection). The exception handler may interpret the error code (8n + 2, see processor manual) and emulate the INT n accordingly.

|  |  |
| --- | --- |
| IA-32 CONTROL TRANSFER ITEMS | 109 |

**A.9 IA-32 Control Transfer Items** **[ia32]**

General Purpose Register CtrlXferItem (id = 0)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | EAX (32) |  |  | MR i+9 |  |
|  |  |  |  | MR i+8 |  |
|  | ECX (32) |  |  |  |
|  |  |  |  | MR i+7 |  |
|  | EDX (32) |  |  |  |
|  |  |  |  | MR i+6 |  |
|  | EBX (32) |  |  |  |
|  |  |  |  | MR i+5 |  |
|  | ESP (32) |  |  |  |
|  |  |  |  | MR i+4 |  |
|  | EBP (32) |  |  |  |
|  |  |  |  | MR i+3 |  |
|  | ESI (32) |  |  |  |
|  |  |  |  | MR i+2 |  |
|  | EFLAGS (32) |  |  |  |
|  |  |  |  | MR i+1 |  |
|  |  |  |  |  |
|  | EIP (32) |  |  |  |
|  |  |  |  | |  |
|  | 0x3FF (20) | 1 (8) | 1 1 0 C | MR i |  |
|  |  |  |  |  |  |

Floating Point Register CtrlXferItem (id = 1)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | FPU register 128/512 (32) | | |  |  | FPU 25不=512 |  |
|  | . |  |  |  | . | |  |
|  | . |  |  |  | . | |  |
|  | . |  |  |  | . | |  |
|  | FPU register 0 (32) | | |  |  | FPU 0 |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | |  |
|  | (20) |  |  | 2 (8) | 1 1 0 C | MR i |  |
|  | . |  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

**Convenience Programming Interface**

#include <l4/ia32/arch.h>

struct **GPREGSCTRLXFERITEM** f Word raw [11] g

struct **FPUREGSCTRLXFERITEM** f Word raw [1] g

110

IA-32 CONTROL TRANSFER ITEMS

void Append

(Msg& msg, GPRegsCtrlXferItem c)

[MsgAppendGPRegsCtrlXferItem]

void Append

(Msg& msg, FPURegsCtrlXferItem c)

[MsgAppendFPURegsCtrlXferItem]

|  |  |
| --- | --- |
| PROCESSOR MIRRORING | 111 |

**A.10 Processor Mirroring** **[ia32]**

**Segments**

L4 uses a flat (unsegmented) memory model. There are only three segments available: user space, a read/write segment, user space exec, an executable segment, and utcb address, a read-only segment. Both user space and user space exec cover (at least) the complete user-level address space. Utcb address covers only enough memory to hold the UTCB address.

The values of segment selectors are undefined. When a thread is created, its segment registers SS, DS, ES and FS are initialized with user space, GS with utcb address, and CS with user space exec. Whenever the kernel detects a general protection exception and the segment registers are not loaded properly, it reloads them with the above mentioned selectors. From the user’s point of view, the segment registers cannot be modified.

However, the binary representation of user space and user space exec may change at any point during program exe-cution. Never rely on any particular value.

Furthermore, the LSL (load segment limit) machine instruction may deliver wrong segment limits, even floating ones.

The result of this instruction is always undefined.

**Debug Registers**

User-level debug registers exist per thread. DR0. . . 3, DR6 and DR7 can be accessed by the machine instructions mov n,DRx and mov DRx,r. However, only task-local breakpoints can be activated, i.e., bits G0. . . 3 in DR7 cannot be set. Breakpoints operate per thread. Breakpoints are signaled as #DB exception (INT 1).

Note that user-level breakpoints are suspended when kernel breakpoints are set by the kernel debugger.

**Model-Specific Registers**

All privileged threads in the system have read and write access to all the Model-Specific Registers (MSRs) of the CPU. Modification of some MSRs may lead to undefined system behavior. Any access to an MSR by an unprivileged thread will raise an exception.

112 BOOTING

**A.11 Booting** **[ia32]**

**PC-compatible Machines**

L4 can be loaded at any 16-byte-aligned location beyond 0x1000 in physical memory. It can be started in real mode or in 32-bit protected mode at address 0x100 or 0x1000 relative to its load address. The protected-mode conditions are compliant to the Multiboot Standard Version 0.6.

Start Preconditions

|  |  |  |
| --- | --- | --- |
|  | Real Mode | 32-bit Protected Mode |
|  |  |  |
| load base (L) | L 0x1000, 16-byte aligned | L 0x1000 |
| load offset (X) | X = 0x100 or X = 0x1000 | X = 0x100 or X = 0x1000 |
| Interrupts | disabled | disabled |
| Gate A20 |  | open |
| EFLAGS | I=0 | I=0, VM=0 |
| CR0 | PE=0 | PE=1, PG=0 |
| (E)IP | X | L + X |
| CS | L=16 | 0, 4GB, 32-bit exec |
| SS,DS,ES |  | 0, 4GB, read/write |
| EAX |  | 0x2BADB002 |
| EBX |  | P |
| hP + 0i |  | OR 1 |
| hP + 4i | n/a | below 640 K mem in K |
| hP + 8i |  | beyond 1M mem in K |
| all remaining registers & flags |  |  |
| (general, floating point, |  |  |
| ESP, xDT, TR, CRx, DRx) |  |  |
|  |  |  |

L4 relocates itself to 0x1000, enters protected mode if started in real mode, enables paging and initializes itself.

|  |  |
| --- | --- |
| SUPPORT FOR HARDWARE-ASSISTED VIRTUALIZATION | 113 |

**A.12 Support for Hardware-assisted Virtualization** **[ia32]**

In addition to its normal execution mode, L4 provides support for virtualization mode. Virtualization mode is largely common to L4’s normal execution model. However, in virtualization mode, threads have access to an extended ISA, and have restricted access to L4-specific features.

Hardware virtualization mode (HVM) is based on IA-32 virtualization hardware extensions: Intel VT-x or AMD Pacifica. Threads that execute in that mode have access to an extended architecture that includes the entire privileged instruction set (ideally, within the limits of the hardware facilities). Such a thread can be seen as a virtual CPU, which contains all of the state held by a physical CPU. In addition to the “normal” page faults and exceptions already handled by L4, HVM threads generate virtualization faults on all events that would be observable by the hardware connected to a physical CPU (and some events that would be internal to a physical CPU).

The virtualization extensions introduce new kernel feature strings:

|  |  |
| --- | --- |
| String | Feature |
| “x86-vmx” | Kernel has full virtualization support using Intel’s VT-x. |
| “x86-svm” | Kernel has full virtualization support using AMD’s Pacifica. |

**Extended Thread State**

An thread inside a HVM space represents a virtualized physical processor for the virtualization HVM space. It holds all privileged and unprivileged registers of the physical processor. VM-exits cause virtualization fault messages to efficiently manage critical instructions. Virtualization fault replies allow mapping memory into the HVM space and protocol items allow read/write access to the VCPU state. EXCHANGEREGISTERS grants asynchronous access by forcing virtualization faults.

**Address Space**

In hardware virtualization mode, the L4 execution and resource model is mapped onto a physical machine model. A thread that executes in HVM has access to the privileged part of the platform architecture and runs with an additional memory translation. Depending on the hardware support for double paging, L4 either utilizes the hardware features or provides a transparent translation of guest-virtual-to-host-physical translations, based on the guest’s virtual to physical, and the host’s virtual to physical mappings.

**SPACECONTROL**

The SPACECONTROL system call has an architecture dependent control parameter to specify various address space char-acteristics. For IA-32, the control parameter has the following semantics.

**Input Parameters**

**control**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| s | v | t | 0 (21) | small (8) |

* The v field denotes the virtualization mode for all threads in the address space. The v field can only be specified for inactive address spaces and is ignored for active address spaces. The availability of the virtualization

features is announced as a KIP feature string.

v=0 An address space with no virtualization support.

v=1 Hardware virtualization mode is the hardware assisted virtualization sup-

port for IA-32, either Intel’s VT-x or AMD’s Pacifica. In hardware virtu-

alization mode, the complete address space is empty and under control of

the pager thread. The thread’s state is extended by IA-32 processor state

including control registers, all segment selectors, debugging registers, etc.

**Output Parameters**

**control**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| e | v | t | 0 (21) | small (8) |

* Indicates if enabling the requested virtualization mode has succeeded (v = 1). Zero if v = 0 in the input parameter.

114 SUPPORT FOR HARDWARE-ASSISTED VIRTUALIZATION

**IA-32 HVM Control Transfer Items**

Control Register CtrlXferItem (id = 2)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| CR4 Read Shadow (32) | |  |  | MR i+8 |  |
|  |  |  |  | MR i+7 |  |
|  |  |  |  |  |
| CR4 Guest/Host Mask (32) | |  |  |  |
|  |  |  |  | MR i+6 |  |
|  |  |  |  |  |
| CR4 (32) | |  |  |  |
|  |  |  |  | MR i+5 |  |
| CR3 (32) | |  |  |  |
|  |  |  |  | MR i+4 |  |
| CR2 (32) | |  |  |  |
|  |  |  |  | MR i+3 |  |
| CR0 Guest/Host Mask (32) | |  |  |  |
|  |  |  |  | MR i+2 |  |
|  |  |  |  |  |
| CR0 Read Shadow (32) | |  |  |  |
|  |  |  |  | MR i+1 |  |
|  |  |  |  |  |
| CR0 (32) | |  |  |  |
|  |  |  |  | |  |
| 0x7F (20) |  | 4 (8) | 1 1 0 C | MR i |  |

Debug Register CtrlXferItem (id = 3)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DR7 (32) |  |  | MR i+6 |  |
|  |  |  | MR i+5 |  |
| DR6 (32) |  |  |  |
|  |  |  | MR i+4 |  |
| DR3 (32) |  |  |  |
|  |  |  | MR i+3 |  |
| DR2 (32) |  |  |  |
|  |  |  | MR i+2 |  |
| DR1 (32) |  |  |  |
|  |  |  | MR i+1 |  |
| DR0 (32) |  |  |  |
|  |  |  | |  |
| 0x3F (20) | 5 (8) | 1 1 0 C | MR i |  |

Code Segment Register CtrlXferItem (id = 4)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CS ATTR (32) |  |  | MR i+4 |  |
|  |  |  | MR i+3 |  |
|  |  |  |  |
| CS LIMIT (32) |  |  |  |
|  |  |  | MR i+2 |  |
|  |  |  |  |
| CS BASE (32) |  |  |  |
|  |  |  | MR i+1 |  |
|  |  |  |  |
| CS (32) |  |  |  |
|  |  |  | |  |
| 0xF (20) | 6 (8) | 1 1 0 C | MR i |  |

Stack Segment Register CtrlXferItem (id = 5)

SUPPORT FOR HARDWARE-ASSISTED VIRTUALIZATION 115

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SS ATTR (32) |  |  | MR i+4 |  |
|  |  |  | MR i+3 |  |
|  |  |  |  |
| SS LIMIT (32) |  |  |  |
|  |  |  | MR i+2 |  |
|  |  |  |  |
| SS BASE (32) |  |  |  |
|  |  |  | MR i+1 |  |
|  |  |  |  |
| SS (32) |  |  |  |
|  |  |  | |  |
| 0xF (20) | 7 (8) | 1 1 0 C | MR i |  |

Data Segment Register CtrlXferItem (id = 6)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DS ATTR (32) |  |  | MR i+4 |  |
|  |  |  | MR i+3 |  |
|  |  |  |  |
| DS LIMIT (32) |  |  |  |
|  |  |  | MR i+2 |  |
|  |  |  |  |
| DS BASE (32) |  |  |  |
|  |  |  | MR i+1 |  |
|  |  |  |  |
| DS (32) |  |  |  |
|  |  |  | |  |
| 0xF (20) | 8 (8) | 1 1 0 C | 先生i |  |

Extra Segment Register CtrlXferItem (id = 7)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ES ATTR(32) |  |  | MR i+4 |  |
|  |  |  | MR i+3 |  |
|  |  |  |  |
| ES LIMIT (32) |  |  |  |
|  |  |  | MR i+2 |  |
|  |  |  |  |
| ES BASE (32) |  |  |  |
|  |  |  | MR i+1 |  |
|  |  |  |  |
| ES (32) |  |  |  |
|  |  |  | |  |
| 0xF (20) | 9 (8) | 1 1 0 C | MR i |  |

F-Segment Register CtrlXferItem (id = 8)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| FS ATTR (32) |  |  | MR i+4 |  |
|  |  |  | MR i+3 |  |
|  |  |  |  |
| FS LIMIT (32) |  |  |  |
|  |  |  | MR i+2 |  |
|  |  |  |  |
| FS BASE (32) |  |  |  |
|  |  |  | MR i+1 |  |
|  |  |  |  |
| FS (32) |  |  |  |
|  |  |  | |  |
| 0xF (20) | 10 (8) | 1 1 0 C | MR i |  |

G-Segment Register CtrlXferItem (id = 9)

116 SUPPORT FOR HARDWARE-ASSISTED VIRTUALIZATION

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| GS ATTR (32) |  |  | MR i+4 |  |
|  |  |  | MR i+3 |  |
|  |  |  |  |
| GS LIMIT (32) |  |  |  |
|  |  |  | MR i+2 |  |
|  |  |  |  |
| GS BASE (32) |  |  |  |
|  |  |  | MR i+1 |  |
|  |  |  |  |
| GS (32) |  |  |  |
|  |  |  | |  |
| 0xF (20) | 11 (8) | 1 1 0 C | MR i |  |

Task Register CtrlXferItem (id = 10)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TR ATTR (32) |  |  | MR i+4 |  |
|  |  |  | MR i+3 |  |
|  |  |  |  |
| TR LIMIT (32) |  |  |  |
|  |  |  | MR i+2 |  |
|  |  |  |  |
| TR BASE (32) |  |  |  |
|  |  |  | MR i+1 |  |
|  |  |  |  |
| TR (32) |  |  |  |
|  |  |  | |  |
| 0xF (20) | 12 (8) | 1 1 0 C | MR i |  |

Local Descriptor Register CtrlXferItem (id = 11)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| LDTR ATTR (32) |  |  | MR i+4 |  |
|  |  |  | MR i+3 |  |
|  |  |  |  |
| LDTR LIMIT (32) |  |  |  |
|  |  |  | MR i+2 |  |
|  |  |  |  |
| LDTR BASE (32) |  |  |  |
|  |  |  | MR i+1 |  |
|  |  |  |  |
| LDTR (32) |  |  |  |
|  |  |  | |  |
| 0xF (20) | 13 (8) | 1 1 0 C | MR i |  |

Interrupt Descriptor Register CtrlXferItem (id = 12)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| IDTR ATTR (32) |  |  | MR i+3 |  |
|  |  |  | MR i+2 |  |
|  |  |  |  |
| IDTR LIMIT (32) |  |  |  |
|  |  |  | MR i+1 |  |
|  |  |  |  |
| IDTR BASE (32) |  |  |  |
|  |  |  |  |  |
|  |  |  | |  |
| 0x7 (20) | 14 (8) | 1 1 0 C | MR i |  |

Global Descriptor Register CtrlXferItem (id = 13)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| GDTR ATTR (32) |  |  | MR i+3 |  |
|  |  |  | MR i+2 |  |
|  |  |  |  |
| GDTR LIMIT (32) |  |  |  |
|  |  |  | MR i+1 |  |
|  |  |  |  |
| GDTR BASE (32) |  |  |  |
|  |  |  |  |  |
|  |  |  | |  |
| 0x7 (20) | 15 (8) | 1 1 0 C | MR i |  |

SUPPORT FOR HARDWARE-ASSISTED VIRTUALIZATION 117

Guest Non-Reg and Exception State CtrlXferItem (id = 14)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| IDT EEC (32) | |  |  | MR i+9 |  |
|  |  |  |  | MR i+8 |  |
|  |  |  |  |  |
| IDT INFO (32) | |  |  |  |
|  |  |  |  | MR i+7 |  |
|  |  |  |  |  |
| EXIT EEC (32) | |  |  |  |
|  |  |  |  | MR i+6 |  |
|  |  |  |  |  |
| EXIT INFO (32) | |  |  |  |
|  |  |  |  | MR i+5 |  |
|  |  |  |  |  |
| ENTRY ILEN (32) | |  |  |  |
|  |  |  |  | MR i+4 |  |
|  |  |  |  |  |
| ENTRY EEC (32) | |  |  |  |
|  |  |  |  | MR i+3 |  |
|  |  |  |  |  |
| ENTRY INFO (32) | |  |  |  |
|  |  |  |  | MR i+3 |  |
|  |  |  |  |  |
| PENDING DEBUG EXC (32) | |  |  |  |
|  |  | |  | MR i+2 |  |
|  |  | |  |  |
| INTERRUPTIBILITY STATE (32) | | |  |  |
|  |  |  |  | MR i+1 |  |
|  |  |  |  |  |
| ACTIVITY STATE (32) | |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  | |  |
| 0x3FF (20) |  | 16 (8) | 1 1 0 C | MR i |  |

Guest Execution Control CtrlXferItem (id = 15)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| EXC BITMAP (32) | |  |  | MR i+3 |  |
|  |  |  |  | MR i+2 |  |
|  |  |  |  |  |
| CPU EXEC CTRL (32) | |  |  |  |
|  |  |  |  | MR i+1 |  |
|  |  |  |  |  |
| PIN EXEC CTRL (32) | |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  | |  |
| 0x7f (20) |  | 17 (8) | 1 1 0 C | MR i |  |

Other Guest State CtrlXferItem (id = 16)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TPR THRESHOLD (32) | |  |  | MR i+9 |  |
|  |  |  |  | MR i+8 |  |
|  |  |  |  |  |
| VAPIC ADDRESS (32) | |  |  |  |
|  |  |  |  | MR i+7 |  |
|  |  |  |  |  |
| RDTSC OFS HIGH (32) | |  |  |  |
|  |  |  |  | MR i+6 |  |
|  |  |  |  |  |
| RDTSC OFS LOW (32) | |  |  |  |
|  |  |  |  | MR i+5 |  |
|  |  |  |  |  |
| DEBUGCTL MSR HIGH (32) | |  |  |  |
|  |  |  |  | MR i+4 |  |
|  |  |  |  |  |
| DEBUGCTL MSR LOW (32) | |  |  |  |
|  |  |  |  | MR i+3 |  |
|  |  |  |  |  |
| SYSENTER ESP MSR (32) | |  |  |  |
|  |  |  |  | MR i+2 |  |
|  |  |  |  |  |
| SYSENTER EIP MSR (32) | |  |  |  |
|  |  |  |  | MR i+1 |  |
|  |  |  |  |  |
| SYSENTER CS MSR (32) | |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  | |  |
| 0x1FF (20) |  | 18 (8) | 1 1 0 C | MR i |  |

118 SUPPORT FOR HARDWARE-ASSISTED VIRTUALIZATION

**Virtualization Fault Protocol**

The virtualization protocol is defined between a VCPU thread and its registered pager thread. It substitutes the page fault and exception protocol used for normal threads. Virtualization fault messages are sent to the pager on events related to virtualization that are not handled directly by the hardware or by the L4 microkernel. By default, the kernel will append the fault-specific state specified below when sending kernel messages. Like with the normal fault protocols (see Section [7,](#page91) the kernel will append additional control transfer items upon requests. Requests to add or remove control transfer items protocol are performed using the EXCHANGEREGISTERS system call and appropriate control transfer configuration items (see Section [2.3)](#page30).

Virtualization Fault

From Pager:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | guest address / instruction info (32) | | | | |  |  | MR 3 |  |  |
|  |  |  |  |  |  |  |  | MR 2 |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  | instruction length (32) | | |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | MR 1 |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  | 故障鉴定(32) | | |  |  |  |  |  |  |
|  | |  |  |  |  |  |  |  | MR 0 |  |
|  | |  |  |  |  |  |  | |  |
| 9 faultID (16) | | 0 (4) |  | 0 (4) |  | t = 0 (6) | u = 3 (6) | |  |
| fault ID | Implementation-specific fault identifiier. For Intel VT-x, the identifier cor- | | | | | | | | |  |
|  | 响应 VM 退出原因。 | | | | |  |  |  |  |  |
| 此类故障- | Additional information about the cause of exits. | | | | | |  |  |  |  |
| fication | The length of the faulting instruction. | | | | |  |  |  |  |  |
| 教学 |  |  |  |  |  |
| 长度 | Guest linear address / Additional information about the faulting instruction | | | | | | | | |  |
| operand |  |
| info | . |  |  |  |  |  |  |  |  |  |
| 值 | 对于读取错误,如果寄存器是 | | | | | | | | |  |
|  | VCPU 状态的一部分。 | | |  |  |  |  |  |  |  |

虚拟化故障回复

从调页程序:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | CtrlXferItem N | | | |  | 先生c0+::: + cn+3 |  |
|  |  |  |  |  |  |  |  |  |
|  | . |  |  |  |  | . | |  |
| . | |  |  |  |  | . | |  |
| . | |  |  |  |  | . | |  |
|  |  | CtrlXferItem 0 | | | |  | MR c0+3 |  |
|  |  |  |  |  |  |  | MR 1;2 |  |
|  |  |  |  |  |  |  |  |
|  |  | MapItem / GrantItem | | | |  |  |
|  |  |  |  |  |  |  |  |  |
|  | |  |  |  |  |  |  |  |
|  | 0 (148) |  | 0 (4) t | = 2 + Pci (6) | u = 0 (6) |  | MR 0 |  |

**A.13** **MSR-Fpage**

Access to processor’s model specific registers is controlled via fpages. The minimal granularity is 1. An MSR-fpage of size 2s0 has a 2s-aligned offset address sndbase + offset, i.e offset mod 2s=0.

**control**

A.13. MSR-FPAGE 119

|  |  |  |  |
| --- | --- | --- | --- |
| offset (16) | s0 (6) | s = 3 | v r w x |

* Allow read access to the specified MSRs.
* Allow write access to the specified MSRs.
* Ignored for mappings into non-HVM spaces. For mappings into HVM space v = 0 grants access to the system MSR. On v = 0 the kernel in-stalls a VCPU local MSRs which is transparently multiplexed.

|  |  |
| --- | --- |
| s’ | 2s0 is the size of the region. |
| 偏移 | 了set 指定 MSR 基址的最低16位。 |

120 SUPPORT FOR HARDWARE-ASSISTED VIRTUALIZATION

**Appendix B**

AMD64 Interface

122 VIRTUAL REGISTERS

**B.1 Virtual Registers** **[amd64]**

**Thread Control Registers (TCRs)**

TCRs are implemented as part of the amd64-specific user-level thread control block (UTCB). The address of the current thread’s UTCB will not change over the lifetime of the thread. Setting the UTCB address of an active thread via THREAD-CONTROL is similar to deletion and re-creation. There is a fixed correlation between the UtcbLocation parameter when invoking THREADCONTROL and the UTCB address. The UTCB address of the current thread can be loaded through a machine instruction

mov %gs:[0], %r

UTCB objects of the current thread can then be accessed as any other memory object. UTCBs of other threads must not be accessed, even if they are physically accessible. ThreadWord0 and ThreadWord1 are free to be used by systems software (e.g., IDL compilers). The kernel associates no semantics with these words.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | ThreadWord 0(64) | |  | UTCB 地址–32 |  |
|  | ThreadWord 1 (64) | |  | – 40 |  |
|  |  |  |  | – 48 |  |
|  | VirtualSender/ActualSender (64) | |  |  |
|  |  |  |  | – 56 |  |
|  | IntendedReceiver (64) | |  |  |
|  |  |  |  | – 64 |  |
|  | XferTimeouts (64) | |  |  |
|  |  |  |  | – 72 |  |
|  | 代码(64) | |  |  |
|  |  |  |  | – 80 |  |
| (48) |  | cop flags (8) | preempt flags (8) |  |
|  | ExceptionHandler (64) | |  | – 88 |  |
|  |  | |  | – 96 |  |
|  | Pager (64) | |  |  |
|  |  | |  | –104 |  |
|  | UserDefinedHandle (64) | |  |  |
|  |  | |  | –112 |  |
|  | ProcessorNo (64) | |  |  |
|  |  | |  | –120 |  |
|  | MyGlobalId (64) | |  |  |
|  |  |  |  |  |  |

|  |  |
| --- | --- |
| MyLocalId = UTCB 地址(64) | gs:[0] |
|  |  |

The TCR MyLocalId is not part of the UTCB. On amd64 it is identical with the UTCB address and can be loaded from memory location gs:[0].

**Message Registers (MRs)**

Memory-mapped MRs are implemented as part of the amd64-specific user-level thread control block (UTCB). The ad-dress of the current thread’s UTCB will not change over the lifetime of the thread. Setting the UTCB address of an active thread via THREADCONTROL is similar to deletion and re-creation. There is a fixed correlation between the UtcbLoca-tion parameter when invoking THREADCONTROL and the UTCB address. The UTCB address of the current thread can

|  |  |
| --- | --- |
| VIRTUAL REGISTERS | 123 |

be loaded through a machine instruction

mov %gs:[0], %r

UTCB objects of the current thread can then be accessed as any other memory object. UTCBs of other threads must not be accessed, even if they are physically accessible.

The first 8 message registers MR 0 through MR 7 are always mapped to general register. MR 8:::63 are always mapped to memory.

|  |  |  |  |
| --- | --- | --- | --- |
| MR 0:::7 | MR 7 | R15 |  |
|  |  |
|  |  | R14 |  |
|  | MR 6 |  |
|  |  | R13 |  |
|  | MR 5 |  |
|  |  | R12 |  |
|  | 先生4 |  |
|  |  | R10 |  |
|  | MR 3 |  |
|  |  | RBX |  |
|  | MR 2 |  |
|  |  | RAX |  |
|  | MR 1 |  |
|  |  | R09 |  |
|  | MR 0 |  |
|  |  |  |  |

MR 8:::63 [UTCB fields]

|  |  |  |
| --- | --- | --- |
| MR 63 (64) |  | + 504 |
| . | . |  |
| . | . |  |
| . | . |  |
| MR 10 (64) |  | + 80 |
| MR 9 (64) |  | + 72 |
| MR 8 (64) |  | UTCB address + 64 |

**Buffer Registers (BRs)**

BRs are implemented as part of the amd64-specific user-level thread control block (UTCB). The address of the current thread’s UTCB will not change over the lifetime of the thread. Setting the UTCB address of an active thread via THREAD-CONTROL is similar to deletion and re-creation. There is a fixed correlation between the UtcbLocation parameter when invoking THREADCONTROL and the UTCB address. The UTCB address of the current thread can be loaded through a machine instruction

mov %gs:[0], %r

UTCB objects of the current thread can then be accessed as any other memory object. UTCBs of other threads must not be accessed, even if they are physically accessible.

BR 0:::32 [UTCB fields]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 124 |  | VIRTUAL REGISTERS | |  |
|  |  |  | UTCB address –128 |  |
|  | BR 0 (64) |  |  |
|  | BR 1 (64) |  | –136 |  |
|  | . | . |  |  |
| . | | . |  |  |
| . | | . |  |  |
|  | BR 32 (64) |  | –384 |  |

|  |  |
| --- | --- |
| SYSTEMCALLS | 125 |

**B.2 Systemcalls** **[amd64]**

The system-calls which are invoked by the call instruction take the target of the calls the from system-call link fields in the kernel interface page (see page [2)](#page14). Each system-call link specifies an absolute address. An application may use instructions other than call to invoke the system-calls, but must ensure that a valid return address resides on the stack.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **KERNELINTERFACE** | **[Slow Systemcall]** | | |  |  |  |  |
| – | RAX |  | KernelInterface ! |  | RAX | base address |  |
|  |  |  |
| – | RCX |  |  |  | RCX | API Version |  |
| – | RDX |  | lock: nop |  | RDX | API Flags |  |
| – | RSI |  |  | RSI | Kernel ID |  |
| – | RDI |  |  |  | RDI |  |  |
| – | RBX |  |  |  | RBX |  |  |
| – | RBP |  |  |  | RBP |  |  |
| – | R08 |  |  |  | R08 |  |  |
| – | R09 |  |  |  | R09 |  |  |
| – | R10 |  |  |  | R10 |  |  |
| – | R11 |  |  |  | R11 |  |  |
| – | R12 |  |  |  | R12 |  |  |
| – | R13 |  |  |  | R13 |  |  |
| – | R14 |  |  |  | R14 |  |  |
| – | R15 |  |  |  | R15 |  |  |
| – | RSP |  |  |  | RSP |  |  |
|  |  |  |  |  |  |  |  |

**EXCHANGEREGISTERS** **[Systemcall]**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| dest | RAX | Exchange Registers ! | RAX | result |  |
| – | RCX |  | RCX |  |  |
| SP | RDX | call ExchangeRegisters | RDX | SP |  |
| control | RSI | RSI | control |  |
| pager | RDI |  | RDI | pager |  |
| – | RBX |  | RBX |  |  |
| – | RBP |  | RBP |  |  |
| IP | R08 |  | R08 | IP |  |
| FLAGS | R09 |  | R09 | FLAGS |  |
| UserDefinedHandle | R10 |  | R10 | UserDefinedHandle |  |
| – | R11 |  | R11 |  |  |
| – | R12 |  | R12 |  |  |
| – | R13 |  | R13 |  |  |
| – | R14 |  | R14 |  |  |
| – | R15 |  | R15 |  |  |
| – | RSP |  | RSP |  |  |

“FLAGS” refers to the user-modifiable amd64 processor flags that are held in the RFLAGS register.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 126 |  |  |  |  |  | SYSTEMCALLS |  |
| **THREADCONTROL** | **[Privileged Systemcall]** | | |  |  |  |  |
| – | RAX |  | Thread Control ! |  | RAX | result |  |
|  |  |  |
| – | RCX |  |  |  | RCX |  |  |
| scheduler | RDX |  | call ThreadControl |  | RDX |  |  |
| pager | RSI |  |  | RSI |  |  |
| dest | RDI |  |  |  | RDI |  |  |
| – | RBX |  |  |  | RBX |  |  |
| – | RBP |  |  |  | RBP |  |  |
| SpaceSpecifier | R08 |  |  |  | R08 |  |  |
| UTCBLocation | R09 |  |  |  | R09 |  |  |
| – | R10 |  |  |  | R10 |  |  |
| – | R11 |  |  |  | R11 |  |  |
| – | R12 |  |  |  | R12 |  |  |
| – | R13 |  |  |  | R13 |  |  |
| – | R14 |  |  |  | R14 |  |  |
| – | R15 |  |  |  | R15 |  |  |
| – | RSP |  |  |  | RSP |  |  |
|  |  |  |  |  |  |  |  |

**SYSTEMCLOCK** **[Systemcall]**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| – | RAX | SystemClock ! | RAX | clock |  |
| – | RCX |  | RCX |  |  |
| – | RDX | call SystemClock | RDX |  |  |
| – | RSI | RSI |  |  |
| – | RDI |  | RDI |  |  |
| – | RBX |  | RBX |  |  |
| – | RBP |  | RBP |  |  |
| – | R08 |  | R08 |  |  |
| – | R09 |  | R09 |  |  |
| – | R10 |  | R10 |  |  |
| – | R11 |  | R11 |  |  |
| – | R12 |  | R12 |  |  |
| – | R13 |  | R13 |  |  |
| – | R14 |  | R14 |  |  |
| – | R15 |  | R15 |  |  |
| – | RSP |  | RSP |  |  |
|  |  |  |  |  |  |

**THREADSWITCH** **[Systemcall]**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| – | RAX | ThreadSwitch ! | RAX |  |  |
| – | RCX |  | RCX |  |  |
| – | RDX | call ThreadSwitch | RDX |  |  |
| – | RSI | RSI |  |  |
| dest | RDI |  | RDI |  |  |
| – | RBX |  | RBX |  |  |
| – | RBP |  | RBP |  |  |
| – | R08 |  | R08 |  |  |
| – | R09 |  | R09 |  |  |
| – | R10 |  | R10 |  |  |
| – | R11 |  | R11 |  |  |
| – | R12 |  | R12 |  |  |
| – | R13 |  | R13 |  |  |
| – | R14 |  | R14 |  |  |
| – | R15 |  | R15 |  |  |
| – | RSP |  | RSP |  |  |
|  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| SYSTEMCALLS |  |  |  |  |  | 127 |  |
| **SCHEDULE [Systemcall]** | |  |  |  |  |  |  |
| – | RAX |  | Schedule ! |  | RAX | result |  |
|  |  |  |
| – | RCX |  |  |  | RCX |  |  |
| time control | RDX |  | call Schedule |  | RDX | time control |  |
| prio | RSI |  |  | RSI |  |  |
| dest | RDI |  |  |  | RDI |  |  |
| – | RBX |  |  |  | RBX |  |  |
| – | RBP |  |  |  | RBP |  |  |
| processor control | R08 |  |  |  | R08 |  |  |
| preemption control | R09 |  |  |  | R09 |  |  |
| – | R10 |  |  |  | R10 |  |  |
| – | R11 |  |  |  | R11 |  |  |
| – | R12 |  |  |  | R12 |  |  |
| – | R13 |  |  |  | R13 |  |  |
| – | R14 |  |  |  | R14 |  |  |
| – | R15 |  |  |  | R15 |  |  |
| – | RSP |  |  |  | RSP |  |  |
|  |  |  |  |  |  |  |  |

**IPC** **[Systemcall]**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| MR 1 | RAX | Ipc ! | RAX | MR 1 |  |
| – | RCX |  | RCX |  |  |
| FromSpecifier | RDX | call Ipc | RDX |  |  |
| to | RSI | RSI | from |  |
| UTCB | RDI |  | RDI |  |  |
| MR 2 | RBX |  | RBX | MR 2 |  |
| – | RBP |  | RBP |  |  |
| Timeouts | R08 |  | R08 |  |  |
| MR 0 | R09 |  | R09 | MR 0 |  |
| MR 3 | R10 |  | R10 | MR 3 |  |
| – | R11 |  | R11 |  |  |
| MR 4 | R12 |  | R12 | MR 4 |  |
| MR 5 | R13 |  | R13 | MR 5 |  |
| MR 6 | R14 |  | R14 | MR 6 |  |
| MR 7 | R15 |  | R15 | MR 7 |  |
| – | RSP |  | RSP |  |  |
|  |  |  |  |  |  |

**LIPC** **[Systemcall]**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| MR 1 | 獭 | Lipc ! | RAX | MR 1 |  |
| – | RCX |  | RCX |  |  |
| FromSpecifier | RDX | call Lipc | RDX |  |  |
| to | RSI | RSI | from |  |
| UTCB | RDI |  | RDI |  |  |
| MR 2 | RBX |  | RBX | MR 2 |  |
| – | RBP |  | RBP |  |  |
| Timeouts | R08 |  | R08 |  |  |
| MR 0 | R09 |  | R09 | MR 0 |  |
| MR 3 | R10 |  | R10 | MR 3 |  |
| – | R11 |  | R11 |  |  |
| MR 4 | R12 |  | R12 | MR 4 |  |
| MR 5 | R13 |  | R13 | MR 5 |  |
| MR 6 | R14 |  | R14 | MR 6 |  |
| MR 7 | R15 |  | R15 | MR 7 |  |
| – | RSP |  | RSP |  |  |
|  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 128 |  |  |  |  |  | SYSTEMCALLS |  |
| **UNMAP [Systemcall]** |  |  |  |  |  |  |  |
| MR 1 | RAX |  | Unmap ! |  | RAX | MR 1 |  |
|  |  |  |
| – | RCX |  |  |  | RCX |  |  |
| control | RDX |  | call Unmap |  | RDX |  |  |
|  | RSI |  |  | RSI |  |  |
| UTCB | RDI |  |  |  | RDI |  |  |
| MR 2 | RBX |  |  |  | RBX | MR 2 |  |
| – | RBP |  |  |  | RBP |  |  |
| – | R08 |  |  |  | R08 |  |  |
| 先生0 | R09 |  |  |  | R09 | MR 0 |  |
| MR 3 | R10 |  |  |  | R10 | MR 3 |  |
| – | R11 |  |  |  | R11 |  |  |
| MR 4 | R12 |  |  |  | R12 | MR 4 |  |
| MR 5 | R13 |  |  |  | R13 | MR 5 |  |
| MR 6 | R14 |  |  |  | R14 | MR 6 |  |
| MR 7 | R15 |  |  |  | R15 | MR 7 |  |
| – | RSP |  |  |  | RSP |  |  |
|  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **SPACECONTROL [Privileged Systemcall]** | | | |  |  |  |  |
| – | RAX |  | Space Control ! |  | RAX | result |  |
|  |  |  |
| – | RCX |  |  |  | RCX |  |  |
| KernelInterfacePageArea | RDX |  | call SpaceControl |  | RDX | control |  |
| control | RSI |  |  | RSI |  |  |
| SpaceSpecifier | RDI |  |  |  | RDI |  |  |
| – | RBX |  |  |  | RBX |  |  |
| – | RBP |  |  |  | RBP |  |  |
| UTCBArea | R08 |  |  |  | R08 |  |  |
| Redirector | R09 |  |  |  | R09 |  |  |
| – | R10 |  |  |  | R10 |  |  |
| – | R11 |  |  |  | R11 |  |  |
| – | R12 |  |  |  | R12 |  |  |
| – | R13 |  |  |  | R13 |  |  |
| – | R14 |  |  |  | R14 |  |  |
| – | R15 |  |  |  | R15 |  |  |
| – | RSP |  |  |  | RSP |  |  |
|  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **PROCESSORCONTROL** | | **[Privileged Systemcall]** | |  |  |  |  |
| – | RAX |  | Processor Control ! |  | RAX | result |  |
|  |  |  |
| – | RCX |  |  |  | RCX |  |  |
| ExternalFrequency | RDX |  | call ProcessorControl |  | RDX |  |  |
| InternalFrequency | RSI |  |  | RSI |  |  |
| ProcessorNo | RDI |  |  |  | RDI |  |  |
| – | RBX |  |  |  | RBX |  |  |
| – | RBP |  |  |  | RBP |  |  |
| voltage | R08 |  |  |  | R08 |  |  |
| – | R09 |  |  |  | R09 |  |  |
| – | R10 |  |  |  | R10 |  |  |
| – | R11 |  |  |  | R11 |  |  |
| – | R12 |  |  |  | R12 |  |  |
| – | R13 |  |  |  | R13 |  |  |
| – | R14 |  |  |  | R14 |  |  |
| – | R15 |  |  |  | R15 |  |  |
| – | RSP |  |  |  | RSP |  |  |
|  |  |  |  |  |  |  |  |

SYSTEMCALLS 129

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **MEMORYCONTROL** | **[Privileged Systemcall]** | | |  |  |  |  |
| MR 1 | RAX |  | Memory Control ! |  | RAX |  |  |
|  |  |  |
| attribute0 | RCX |  |  |  | RCX |  |  |
| control | RDX |  | call MemoryControl |  | RDX | result |  |
| attribute1 | RSI |  |  | RSI |  |  |
| UTCB | RDI |  |  |  | RDI |  |  |
| MR 2 | RBX |  |  |  | RBX |  |  |
| – | RBP |  |  |  | RBP |  |  |
| attribute2 | R08 |  |  |  | R08 |  |  |
| MR 0 | R09 |  |  |  | R09 |  |  |
| MR 3 | R10 |  |  |  | R10 |  |  |
| attribute3 | R11 |  |  |  | R11 |  |  |
| MR 4 | R12 |  |  |  | R12 |  |  |
| MR 5 | R13 |  |  |  | R13 |  |  |
| MR 6 | R14 |  |  |  | R14 |  |  |
| MR 7 | R15 |  |  |  | R15 |  |  |
| – | RSP |  |  |  | RSP |  |  |

130 IO PORTS

**B.3 IO Ports** **[amd64]**

**IO Fpages**

On AMD64 processors, IO-ports are handled as fpages. IO fpages can be mapped, granted, and unmapped like memory fpages. Their minimal granularity is 1. An IO-fpage of size 2s0 has a 2s0-aligned base address p, i.e. p mod 2s0=0. An fpage with base port address p and size 2s0 is denoted as described below.

IO fpage (p; 2s0)

|  |  |  |  |
| --- | --- | --- | --- |
| p (48) | s’ (6) | s = 2 (6) | 0 1 1 0 |

IO-ports can only be mapped idempotently, i.e., physical port x is either mapped at IO address x in the task’s IO address space, or it is not mapped at all. There are no distinct rights associated with IO ports, i.e., a task can be granted either read- and write-access to an IO port, ore none at all.

**IO Pagefault Protocol**

A thread generating an IO port exception will cause the kernel to transparently generate an IO-pagefault IPC to the faulting thread’s pager. The behavior of the faulting thread is undefined if the pager does not exactly follow this protocol.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| To Pager |  | faulting user-level IP (64) | | | |  |  |  |  |  | MR 2 |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  | MR 1 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | faulting port (48) |  | size (6) | |  | s = 2 (6) | |  | 0 1 1 0 |  |
|  |  |  |  |  |  |  |  |  |  |  | MR 0 |  |
|  |  |  |  | |  |  | |  |  | |  |
|  |  | 8 (44) | 0 1 1 0 | 0 (4) |  | t = 0 (6) | |  | u = 2 (6) | |  |
| Acceptor [BR0 | ] |  |  |  |  |  |  |  |  |  | BR 0 |  |
| 0 (48) |  | 16 (6) | |  | s = 2 (6) | |  | 0 0 0 0 |  |
|  |  |  |  |  |  |

The acceptor covers the complete IO address space. The kernel accepts mappings or grants into this region on behalf of the faulting thread. The received message is discarded.

**Generic Programming Interface**

#include <l4/amd64/specials.h>

Fpage IoFpage (Word BaseAddress, int FpageSize)

Fpage IoFpageLog2 (Word BaseAddress, int Log2FpageSize <= 16)

Delivers an IO fpage with the specified location and size.

|  |  |
| --- | --- |
| CACHEABILITY HINTS | 131 |

**B.4 Cacheability Hints** **[amd64]**

String items can specify cacheability hints to the kernel (see page [59)](#page71). For amd64, the cacheability hints have the following semantics.

1. = 00
2. = 01
3. = 10
4. = 11

Use the processor’s default cacheability strategy. Typically, cache lines are allocated for data read and written (assuming that the processor’s default strategy is write-back and write-allocate).

Allocate cache lines in the entire cache hierarchy for data read or written.

Do not allocate new cache lines (entire cache hierarchy) for data read or written.

Allocate only new L1 cache line for data read or written. Do not allocate cache lines in lower cache hierarchies.

**Convenience Programming Interface**

#include <l4/ipc.h>

CacheAllocationHint UseDefaultCacheLineAllocation

CacheAllocationHint AllocateNewCacheLines

CacheAllocationHint DoNotAllocateNewCacheLines

CacheAllocationHint AllocateOnlyNewL1CacheLines

132 MEMORY ATTRIBUTES

**B.5 Memory Attributes** **[amd64]**

The AMD64 architecture in general supports the following memory attributes values.

|  |  |
| --- | --- |
| attribute | value |
| Default | 0 |
| Uncacheable | 1 |
| Write Combining | 2 |
| Write Through | 5 |
| Write Protected | 6 |
| Write Back | 7 |

Note that some attributes are only supported on certain processors. See the “AMD64 Architecture Programmer’s Manual Volume 2: System Programming” for the semantics of the memory attributes and which processors they are supported on.

**Generic Programming Interface**

#include <l4/misc.h>

Word DefaultMemory

Word UncacheableMemory

Word WriteCombiningMemory

Word WriteThroughMemory

Word WriteProtectedMemory

Word WriteBackMemory

|  |  |
| --- | --- |
| EXCEPTION MESSAGE FORMAT | 133 |

**B.6 Exception Message Format** **[amd64]**

To Exception Handler

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | ErrorCode | |  |  | MR 20 |  |
|  |  |  |  |  | MR 19 |  |
|  |  |  |  |  |  |
|  | ExceptionNo | |  |  |  |
|  |  |  |  |  | MR 18 |  |
|  |  |  |  |  |  |
|  | RFLAGS | |  |  |  |
|  |  |  |  |  | MR 17 |  |
|  |  |  |  |  |  |
|  | RSP | |  |  |  |
|  |  |  |  |  | MR 16 |  |
|  |  |  |  |  |  |
|  | R11 | |  |  |  |
|  |  |  |  |  | MR 15 |  |
|  |  |  |  |  |  |
|  | R09 | |  |  |  |
|  |  |  |  |  | MR 14 |  |
|  |  |  |  |  |  |
|  | R08 | |  |  |  |
|  |  |  |  |  | MR 13 |  |
|  |  |  |  |  |  |
|  | RBP | |  |  |  |
|  |  |  |  |  | MR 12 |  |
|  |  |  |  |  |  |
|  | RDI | |  |  |  |
|  |  |  |  |  | MR 11 |  |
|  |  |  |  |  |  |
|  | RSI | |  |  |  |
|  |  |  |  |  | MR 10 |  |
|  |  |  |  |  |  |
|  | RDX | |  |  |  |
|  |  |  |  |  | MR 9 |  |
|  |  |  |  |  |  |
|  | RCX | |  |  |  |
|  |  |  |  |  | MR 8 |  |
|  |  |  |  |  |  |
|  | RAX | |  |  |  |
|  |  |  |  |  | MR 7 |  |
|  |  |  |  |  |  |
|  | R15 | |  |  |  |
|  |  |  |  |  | MR 6 |  |
|  |  |  |  |  |  |
|  | R14 | |  |  |  |
|  |  |  |  |  | MR 5 |  |
|  |  |  |  |  |  |
|  | R13 | |  |  |  |
|  |  |  |  |  | 先生4 |  |
|  |  |  |  |  |  |
|  | R12 | |  |  |  |
|  |  |  |  |  | MR 3 |  |
|  |  |  |  |  |  |
|  | R10 | |  |  |  |
|  |  |  |  |  | MR 2 |  |
|  |  |  |  |  |  |
|  | RBX | |  |  |  |
|  |  |  |  |  | MR 1 |  |
|  |  |  |  |  |  |
|  | RIP | |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  | |  |
| 4= 5 (44) | 0 (4) | 0 (4) | t = 0 (6) | u = 20 (6) | MR 0 |  |

#PF (page fault), #MC (machine check exception), and some #GP (general protection), #SS (stack segment fault), and #NM (no math coprocessor) exceptions are handled by the kernel and therefore do not generate exception messages.

Note that executing an INT n instructions in 32-bit mode will always raise a #GP (general protection). The exception handler may interpret the error code (8n + 2, see processor manual) and emulate the INT n accordingly.

134 PROCESSOR MIRRORING

**B.7 Processor Mirroring** **[amd64]**

**Segments**

L4 uses a flat (unsegmented) memory model. There are only three segments available: user space, a read/write segment, user space exec, an executable segment, and utcb address, a read-only segment. Both user space and user space exec cover (at least) the complete user-level address space. Utcb address covers only enough memory to hold the UTCB address.

The values of segment selectors are undefined. When a thread is created, its segment registers SS, DS, ES and FS are initialized with user space, GS with utcb address, and CS with user space exec. Whenever the kernel detects a general protection exception and the segment registers are not loaded properly, it reloads them with the above mentioned selectors. From the user’s point of view, the segment registers cannot be modified.

However, the binary representation of user space and user space exec may change at any point during program exe-cution. Never rely on any particular value.

Furthermore, the LSL (load segment limit) machine instruction may deliver wrong segment limits, even floating ones.

The result of this instruction is always undefined.

**Debug Registers**

User-level debug registers exist per thread. DR0. . . 3, DR6 and DR7 can be accessed by the machine instructions mov n,DRx and mov DRx,r. However, only task-local breakpoints can be activated, i.e., bits G0. . . 3 in DR7 cannot be set. Breakpoints operate per thread. Breakpoints are signaled as #DB exception (INT 1).

Note that user-level breakpoints are suspended when kernel breakpoints are set by the kernel debugger.

**Model-Specific Registers**

All privileged threads in the system have read and write access to all the Model-Specific Registers (MSRs) of the CPU. Modification of some MSRs may lead to undefined system behavior. Any access to an MSR by an unprivileged thread will raise an exception.

|  |  |
| --- | --- |
| BOOTING | 135 |

**B.8 Booting** **[amd64]**

**PC-compatible Machines**

L4 can be loaded at any 16-byte-aligned location beyond 0x1000 in physical memory. It can be started in real mode or in 32-bit protected mode at address 0x100 or 0x1000 relative to its load address. The protected-mode conditions are compliant to the Multiboot Standard Version 0.6.

Start Preconditions

|  |  |  |
| --- | --- | --- |
|  | Real Mode | 32-bit Protected Mode |
|  |  |  |
| load base (L) | L 0x1000, 16-byte aligned | L 0x1000 |
| load offset (X) | X = 0x100 or X = 0x1000 | X = 0x100 or X = 0x1000 |
| Interrupts | disabled | disabled |
| Gate A20 |  | open |
| EFLAGS | I=0 | I=0, VM=0 |
| CR0 | PE=0 | PE=1, PG=0 |
| (E)IP | X | L + X |
| CS | L=16 | 0, 4GB, 32-bit exec |
| SS,DS,ES |  | 0, 4GB, read/write |
| eax |  | 0x2BADB002 |
| EBX |  | P |
| hP + 0i |  | OR 1 |
| hP + 4i | n/a | below 640 K mem in K |
| hP + 8i |  | beyond 1M mem in K |
| all remaining registers & flags |  |  |
| (general, floating point, |  |  |
| ESP, xDT, TR, CRx, DRx) |  |  |
|  |  |  |

L4 relocates itself to 0x1000, enters protected mode if started in real mode, enables paging and initializes itself.

136 BOOTING

**Appendix C**

PowerPC Interface

138 VIRTUAL REGISTERS

**C.1 Virtual Registers** **[powerpc]**

**Thread Control Registers (TCRs)**

TCRs are mapped to memory locations. They are implemented as part of the PowerPC-specific user-level thread control block (UTCB). The address of the current thread’s UTCB is identical to the thread’s local ID, and is thus immutable. The UTCB address is provided in the general purpose register R2 at application start. The R2 register must contain the UTCB address for every system call invocation. UTCB objects of the current thread can be accessed as any other memory object. UTCBs of other threads must not be accessed, even if they are physically accessible. ThreadWord0 and ThreadWord1 are free to be used by systems software (e.g., IDL compilers). The kernel associates no semantics with these words.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | (32) | |  |  | UTCB address |  |
| . |  |  |  | . |  |  |
| . |  |  |  | . |  |  |
| . |  |  |  | . |  |  |
|  | ThreadWord 0 (32) | |  |  | –16 |  |
|  |  |  |  |  | –20 |  |
|  | ThreadWord 1 (32) | |  |  |  |
|  |  |  |  |  | –24 |  |
|  | VirtualSender/ActualSender (32) | |  |  |  |
|  |  |  |  |  | –28 |  |
|  | IntendedReceiver (32) | |  |  |  |
|  |  |  |  |  | –32 |  |
|  | XferTimeouts (32) | |  |  |  |
|  |  |  |  |  | –36 |  |
|  | ErrorCode (32) | |  |  |  |
|  |  |  |  |  | –40 |  |
| (16) |  | cop flags (8) | preempt flags (8) |  |  |
|  | ExceptionHandler (32) | |  |  | –44 |  |
|  |  | |  |  | –48 |  |
|  | Pager (32) | |  |  |  |
|  |  | |  |  | –52 |  |
|  | UserDefinedHandle (32) | |  |  |  |
|  |  | |  |  | –56 |  |
|  | ProcessorNo (32) | |  |  |  |
|  |  | |  |  | –60 |  |
|  | MyGlobalId (32) | |  |  |  |
|  |  |  |  |  |  |  |

|  |  |
| --- | --- |
| MyLocalId = UTCB address (32) | R2 |
|  |  |

The TCR MyLocalId is not part of the UTCB. On PowerPC it is identical with the UTCB address and can be loaded from register R2.

**Message Registers (MRs)**

Message registers MR 0 through MR 9 map to the processor’s general purpose register file. The remaining message registers map to memory locations in the UTCB. MR 10 starts at byte offset 40 in the UTCB, and successive message registers follow in memory.

VIRTUAL REGISTERS

MR 0:::9

MR 10:::63 [UTCB fields]

.

.

.

139

|  |  |
| --- | --- |
| MR 9 | R0 |
| MR 8 | R10 |
| MR 7 | R9 |
| MR 6 | R8 |
| MR 5 | R7 |
| MR 4 | R6 |
| MR 3 | R5 |
| MR 2 | R4 |
| MR 1 | R3 |
| MR 0 | R14 |
|  |  |

|  |  |  |
| --- | --- | --- |
| MR 63 (32) |  | +252 |
|  | . |  |
|  | . |  |
|  | . |  |
| MR 11 (32) |  | +44 |
| MR 10 (32) |  | UTCB address + 40 |

**Buffer Registers (BRs)**

The buffer registers map to memory locations in the UTCB. BR 0 is at byte offset -64 in the UTCB, BR 1 at byte offset -68, etc.

BR 0:::32 [UTCB fields]

|  |  |  |
| --- | --- | --- |
| (32) |  | UTCB address |
| . | . |  |
| . | . |  |
| . | . |  |
| BR 0 (32) |  | –64 |
| BR 1 (32) |  | –68 |
| . | . |  |
| . | . |  |
| . | . |  |
| BR 32 (32) |  | –196 |

**UTCB Memory With Undefined Semantics**

The kernel will associate no semantics with memory located at UTCB address. . . UTCB address + 39. The application can use this memory as thread local storage, e.g., for implementing the L4 API. Note, however, that the memory contents within this region may be overwritten during a system-call operating on message registers.

All undefined UTCB memory which is not covered by the above mentioned region may have kernel defined semantics.

140 SYSTEMCALLS

**C.2 Systemcalls** **[powerpc]**

The PowerPC system calls are invoked by changing the location of the instruction pointer to the location of the system call address, with the return address in the link-return (LR) register. The invocation may take place via any mechanism which changes the instruction pointer location. The precise locations of the system calls are stored in the kernel interface page (see page [2)](#page14).

The locations of the system calls are fixed during the life of an application, although they may change outside of the life of an application. It is not valid to prelink an application against a set of system call locations. The official locations are always provided in the kernel interface page.

The registers defined to survive across system-call invocations (unless otherwise noted) are: R1, R2, R30, R31, and the floating point registers. All other registers contain return values, are undefined, or may be preserved according to processor specific rules.

The R2 register must contain the UTCB pointer when invoking all system calls.

PowerPC uses one alternative system call invocation mechanism, for the KERNELINTERFACE system call. This system call is invoked via the ’tlbia’ instruction, and most registers are preserved across the function call.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **KERNELINTERFACE** | **[Slow Systemcall]** | | |  |  |  |  |
| UTCB | R2 |  | KernelInterface ! |  | R2 |  |  |
|  |  |  |
| – | R3 |  |  |  | R3 | KIP base address |  |
| – | R4 |  | tlbia |  | R4 | API Version |  |
| – | R5 |  |  | R5 | API Flags |  |
| – | R6 |  |  |  | R6 | Kernel ID |  |
| – | R7 |  |  |  | R7 |  |  |
| – | R8 |  |  |  | R8 |  |  |
| – | R9 |  |  |  | R9 |  |  |
| – | R10 |  |  |  | R10 |  |  |

For this system-call, all registers other than the output registers are preserved. The tlbia instruction encoding is 0x7c0002e4.

**EXCHANGEREGISTERS** **[Systemcall]**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| UTCB | R2 | Exchange Registers ! | R2 |  |  |
| dest | R3 |  | R3 | 结果 |  |
| control | R4 | call ExchangeRegisters | R4 | control |  |
| SP | R5 | R5 | SP |  |
| IP | R6 |  | R6 | IP |  |
| FLAGS | R7 |  | R7 | FLAGS |  |
| UserDefinedHandle | R8 |  | R8 | UserDefinedHandle |  |
| pager | R9 |  | R9 | pager |  |
| – | R10 |  | R10 |  |  |

“FLAGS” refers to the user-modifiable PowerPC processor flags that are held in the MSR register. See the PowerPC Processor Mirroring section (page [148)](#page160).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| SYSTEMCALLS |  |  |  |  |  | 141 |  |
| **THREADC控制** | **[特权Systemcall]** | | |  |  |  |  |
| Utcb | R2 |  | Thread Control ! |  | R2 |  |  |
|  |  |  |
| dest | R3 |  |  |  | R3 | result |  |
| SpaceSpecifier | R4 |  | call ThreadControl |  | R4 |  |  |
| Scheduler | R5 |  |  | R5 |  |  |
| Pager | R6 |  |  |  | R6 |  |  |
| UtcbLocation | R7 |  |  |  | R7 |  |  |
| – | R8 |  |  |  | R8 |  |  |
| – | R9 |  |  |  | R9 |  |  |
| – | R10 |  |  |  | R10 |  |  |
|  |  |  |  |  |  |  |  |

**SYSTEMCLOCK** **[Systemcall]**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| UTCB | R2 | SystemClock ! | R2 |  |  |
| – | R3 |  | R3 | clock 32. . . 63 |  |
| – | R4 | call SystemClock | R4 | clock 0. . . 31 |  |
| – | R5 | R5 |  |  |
| – | R6 |  | R6 |  |  |
| – | R7 |  | R7 |  |  |
| – | R8 |  | R8 |  |  |
| – | R9 |  | R9 |  |  |
| – | R10 |  | R10 |  |  |
|  |  |  |  |  |  |

**THREADS女巫** **[Systemcall]**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Utcb | R2 | ThreadSwitch! | R2 |  |  |
| dest | R3 |  | R3 |  |  |
| – | R4 | call ThreadSwitch | R4 |  |  |
| – | R5 | R5 |  |  |
| – | R6 |  | R6 |  |  |
| – | R7 |  | R7 |  |  |
| – | R8 |  | R8 |  |  |
| – | R9 |  | R9 |  |  |
| – | R10 |  | R10 |  |  |
|  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **SCHEDULE [Systemcall]** | |  |  |  |  |  |  |
| UTCB | R2 |  | Schedule ! |  | R2 |  |  |
|  |  |  |
| dest | R3 |  |  |  | R3 | result |  |
| time control | R4 |  | call Schedule |  | R4 | time control |  |
| processor control | R5 |  |  | R5 |  |  |
| prio | R6 |  |  |  | R6 |  |  |
| preemption control | R7 |  |  |  | R7 |  |  |
| – | R8 |  |  |  | R8 |  |  |
| – | R9 |  |  |  | R9 |  |  |
| – | R10 |  |  |  | R10 |  |  |
|  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 142 |  |  |  |  |  |  | SYSTEMCALLS |  |
| **IPC** | **[Systemcall]** |  |  |  |  |  |  |  |
|  | MR 9 | R0 |  | Ipc ! |  | R0 | MR 9 |  |
|  |  |  |  |
|  | – | R1 |  |  |  | R1 |  |  |
|  | UTCB | R2 |  | call Ipc |  | R2 |  |  |
|  | MR 1 | R3 |  |  | R3 | MR 1 |  |
|  | MR 2 | R4 |  |  |  | R4 | MR 2 |  |
|  | MR 3 | R5 |  |  |  | R5 | MR 3 |  |
|  | MR 4 | R6 |  |  |  | R6 | MR 4 |  |
|  | MR 5 | R7 |  |  |  | R7 | MR 5 |  |
|  | MR 6 | R8 |  |  |  | R8 | MR 6 |  |
|  | MR 7 | R9 |  |  |  | R9 | MR 7 |  |
|  | MR 8 | R10 |  |  |  | R10 | MR 8 |  |
|  | – | R11 |  |  |  | R11 |  |  |
|  | – | R12 |  |  |  | R12 |  |  |
|  | – | R13 |  |  |  | R13 |  |  |
|  | MR 0 | R14 |  |  |  | R14 | MR 0 |  |
|  | to | R15 |  |  |  | R15 |  |  |
|  | FromSpecifier | R16 |  |  |  | R16 | from |  |
|  | Timeouts | R17 |  |  |  | R17 |  |  |
|  |  |  |  |  |  |  |  |  |
| **LIPC** | **[Systemcall]** |  |  |  |  |  |  |  |
|  | MR 9 | R0 |  | Lipc ! |  | R0 | MR 9 |  |
|  |  |  |  |
|  | – | R1 |  |  |  | R1 |  |  |
|  | UTCB | R2 |  | call Lipc |  | R2 |  |  |
|  | MR 1 | R3 |  |  | R3 | MR 1 |  |
|  | MR 2 | R4 |  |  |  | R4 | MR 2 |  |
|  | MR 3 | R5 |  |  |  | R5 | MR 3 |  |
|  | MR 4 | R6 |  |  |  | R6 | MR 4 |  |
|  | 先生5 | R7 |  |  |  | R7 | MR 5 |  |
|  | 先生6 | R8 |  |  |  | R8 | MR 6 |  |
|  | MR 7 | R9 |  |  |  | R9 | MR 7 |  |
|  | MR 8 | R10 |  |  |  | R10 | 先生8 |  |
|  | – | R11 |  |  |  | R11 |  |  |
|  | – | R12 |  |  |  | R12 |  |  |
|  | – | R13 |  |  |  | R13 |  |  |
|  | 先生0 | R14 |  |  |  | R14 | 先生0 |  |
|  | 自 | R15 |  |  |  | R15 |  |  |
|  | FromSpecifier | R16 |  |  |  | R16 | 从 |  |
|  | 超时 | R17 |  |  |  | R17 |  |  |
|  | |  |  |  |  |  |  |  |
| **Unmap[Systemcall]** | |  |  |  |  |  |  |  |
|  | 先生9 | R0 |  | 取消! |  | R0 | 先生9 |  |
|  |  |  |  |
|  | – | R1 |  |  |  | R1 |  |  |
|  | Utcb | R2 |  | 呼叫取消 |  | R2 |  |  |
|  | 先生1 | R3 |  |  | R3 | 先生1 |  |
|  | 先生2 | R4 |  |  |  | R4 | 先生2 |  |
|  | 先生3 | R5 |  |  |  | R5 | 先生3 |  |
|  | 先生4 | R6 |  |  |  | R6 | 先生4 |  |
|  | 先生5 | R7 |  |  |  | R7 | 先生5 |  |
|  | 先生6 | R8 |  |  |  | R8 | 先生6 |  |
|  | 先生7 | R9 |  |  |  | R9 | 先生7 |  |
|  | 先生8 | R10 |  |  |  | R10 | 先生8 |  |
|  | – | R11 |  |  |  | R11 |  |  |
|  | – | R12 |  |  |  | R12 |  |  |
|  | – | R13 |  |  |  | R13 |  |  |
|  | 先生0 | R14 |  |  |  | R14 | 先生0 |  |
|  | 控制 | R15 |  |  |  | R15 |  |  |
|  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SYSTEMCALLS |  |  |  |  |  |  |  |  |  | 143 |  |
| **S步伐C控制[特权 Systemcall]** | | | | | | | |  |  |  |  |
| Utcb | R2 |  | 空间控制! | | | | |  | R2 |  |  |
|  |  |  |
| SpaceSpecifier | R3 |  |  |  |  |  |  |  | R3 | 结果 |  |
| 控制 | R4 |  | 呼叫 SpaceControl | | | | |  | R4 | 控制 |  |
| KernelInterfacePageArea | R5 |  |  | R5 |  |  |
| UtcbArea | R6 |  |  |  |  |  |  |  | R6 |  |  |
| 定向 | R7 |  |  |  |  |  |  |  | R7 |  |  |
| – | R8 |  |  |  |  |  |  |  | R8 |  |  |
| – | R9 |  |  |  |  |  |  |  | R9 |  |  |
| – | R10 |  |  |  |  |  |  |  | R10 |  |  |
|  | |  | |  |  |  |  |  |  |  |  |
| **PROCESSORC控制** | | **[特权 Systemcall]** | | | | | |  |  |  |  |
| Utcb | R2 |  | 处理器控制! | |  |  |  |  | R2 |  |  |
|  |  |  |  |  |
| 处理器编号 | R3 |  |  |  |  |  |  |  | R3 | 结果 |  |
| InternalFreq | R4 |  | 呼叫 ProcessorControl | |  |  |  |  | R4 |  |  |
| ExternalFreq | R5 |  |  |  |  |  | R5 |  |  |
| 电压 | R6 |  |  |  |  |  |  |  | R6 |  |  |
| – | R7 |  |  |  |  |  |  |  | R7 |  |  |
| – | R8 |  |  |  |  |  |  |  | R8 |  |  |
| – | R9 |  |  |  |  |  |  |  | R9 |  |  |
| – | R10 |  |  |  |  |  |  |  | R10 |  |  |
|  |  | | |  | |  | |  |  |  |  |
| **M埃默里C控制** | **[特权 Systemcall]** | | | | | | |  |  |  |  |
| 先生9 | R0 |  | 内存控制! |  | |  | |  | R0 |  |  |
|  |  | |  |  |
| – | R1 |  |  |  |  |  |  |  | R1 |  |  |
| Utcb | R2 |  | 呼叫 MemoryControl |  |  |  |  |  | R2 |  |  |
| 先生1 | R3 |  |  | |  | |  | R3 | 结果 |  |
| 先生2 | R4 |  |  |  |  |  |  |  | R4 |  |  |
| 先生3 | R5 |  |  |  |  |  |  |  | R5 |  |  |
| 先生4 | R6 |  |  |  |  |  |  |  | R6 |  |  |
| 先生5 | R7 |  |  |  |  |  |  |  | R7 |  |  |
| 先生6 | R8 |  |  |  |  |  |  |  | R8 |  |  |
| 先生7 | R9 |  |  |  |  |  |  |  | R9 |  |  |
| 先生8 | R10 |  |  |  |  |  |  |  | R10 |  |  |
| – | R11 |  |  |  |  |  |  |  | R11 |  |  |
| – | R12 |  |  |  |  |  |  |  | R12 |  |  |
| – | R13 |  |  |  |  |  |  |  | R13 |  |  |
| 先生0 | R14 |  |  |  |  |  |  |  | R14 |  |  |
| 控制 | R15 |  |  |  |  |  |  |  | R15 |  |  |
| 属性0 | R16 |  |  |  |  |  |  |  | R16 |  |  |
| 属性1 | R17 |  |  |  |  |  |  |  | R17 |  |  |
| 属性2 | R18 |  |  |  |  |  |  |  | R18 |  |  |
| 属性3 | R19 |  |  |  |  |  |  |  | R19 |  |  |

144 内存属性

**C. 3 内存属性** **powerpc]**

PowerPC 体系结构支持以下内存/缓存属性值, 用于 M埃默里C控制系统调用:

|  |  |
| --- | --- |
| 属性 | 值 |
| 默认 | 0 |
| 写入 | 1 |
| 回写 | 2 |
| 缓存抑制 | 3 |
| 启用缓存 | 4 |
| 内存-全局 (连贯) | 5 |
| 内存-本地 (不连贯) | 6 |
| 守卫 | 7 |
| 投机 | 8 |

默认属性启用回写、缓存和推测。只有当内核通过支持多个处理器进行编译时, 默认情况下才会启用内存一致性。

PowerPC 建筑的各种对内存/缓存属性使用的限制。有些组合是没有意义的 (例如, 将写操作与缓存抑制相结合), 或者不允许, 并会导致未定义的行为 (例如, 指令获取是 incompatible 与一些属性组合)。内存/缓存访问属性的精确语义在 "PowerPC 体系结构的32位实现的编程环境手册" 中进行了描述。

在禁用页的缓存之前,软件必须确保从缓存中刷新属于目标页的所有内存。

**泛型编程接口**

#include<l4/杂项 h>

字 DefaultMemory

字 WriteThroughMemory

字 WriteBackMemory

字 CachingInhibitedMemory

词CachingEnabledMemory

字 GlobalMemory

字 LocalMemory

字 GuardedMemory

字 SpeculativeMemory

|  |  |
| --- | --- |
| 空间控制 | 145 |

**C. 4 空间控制** **powerpc]**

的 S步伐C控制系统调用具有与结构相关的控制参数, 用于指定各种地址空间特点。对于 PowerPC, 控件参数具有以下语义。

**输入参数**

控制

t

|  |  |  |
| --- | --- | --- |
| 0(2) | t | 0(29) |

一个值1指示内核为扩展映射向转换表中添加项。此表允许映射32位系统上长于32位的内存地址。所需的映射在 SpaceCo 的其余参数中指定控制系统调用如下: 重定向器字段必须包含所需地址的最高32位, 而 utcb 区域字段必须包含较低的32位。"基普区域" 字段包含一个常规 fpage, 它指定32位地址的区域应该映射到64位地址。如果此 fpage 中的任何地址映射到一个线程, 则该地址将被转换为相应的64位地址。如果映射成功, 则删除转换表项。

**输出参数**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 控制 | 0(2) | t | 0(29) |  |
|  |  |

* 指示是否已成功将项添加到内核的转换表中以进行扩展映射。

146 异常消息格式

**C. 5 异常消息格式** **powerpc]**

**系统调用陷阱**

系统调用陷阱消息到异常处理程序

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 标志(32) | |  |  | 先生12 |  |
|  |  |  |  |  | 先生11 |  |
|  |  |  |  |  |  |
|  | sp(32) | |  |  |  |
|  |  |  |  |  | 先生10 |  |
|  | ip(32) | |  |  |  |
|  |  |  |  |  | 先生9 |  |
|  | R0(32) | |  |  |  |
|  |  |  |  |  | 先生8 |  |
|  | R10(32) | |  |  |  |
|  |  |  |  |  | 先生7 |  |
|  | R9(32) | |  |  |  |
|  |  |  |  |  | 先生6 |  |
|  | R8(32) | |  |  |  |
|  |  |  |  |  | 先生5 |  |
|  | R7(32) | |  |  |  |
|  |  |  |  |  | 先生4 |  |
|  | R6(32) | |  |  |  |
|  |  |  |  |  | 先生3 |  |
|  | R5(32) | |  |  |  |
|  |  |  |  |  | 先生2 |  |
|  | R4(32) | |  |  |  |
|  |  |  |  |  | 先生1 |  |
|  | R3(32) | |  |  |  |
|  |  |  |  |  | |  |
| -5(16 = 48) |  | 0(4) | t = 0(6) | 在 = 12(6) | 先生0 |  |

当用户代码执行PowerPC "sc" 指令, 内核将系统调用陷阱消息传递给异常处理程序。内核在处理 "sc" 指令时只保留部分用户状态。状态同样保存在 SVR4 PowerPC ABI 中, 用于函数调用。届e 非挥发性寄存器是 R1, R2, R13..。R31、CR2、CR3、CR4、LR 和寄存器。可变寄存器为 R0、R3..。R12, CR0, CR1, CR5..。CR7、中心和定影。线程虚拟寄存器也可能受到重创。

**一般陷阱**

异常处理程序的一般陷阱消息

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 异常消息格式 | |  |  |  |  | 147 |  |
|  |  |  |  |  |  | 先生6 |  |
|  |  | LocalID(32) | | | |  |
|  |  |  |  |  |  | 先生5 |  |
|  |  |  |  |  |  |  |
|  |  | 代码(32) | | | |  |
|  |  |  |  |  |  | 先生4 |  |
|  |  |  |  |  |  |  |
|  |  | ExceptionNo(32) | | | |  |
|  |  |  |  |  |  | 先生3 |  |
|  |  |  |  |  |  |  |
|  |  | 标志(32) | | | |  |
|  |  |  |  |  |  | 先生2 |  |
|  |  |  |  |  |  |  |
|  |  | sp(32) | | | |  |
|  |  |  |  |  |  | 先生1 |  |
|  |  | ip(32) | | | |  |
|  | |  |  |  |  | 先生0 |  |
|  | -5(16 = 44) |  | 0(4) | t = 0(6) | 在 = 6(6) |  |

内核在响应体系结构特定事件时合成异常消息。有些陷阱是由内核处理的, 因此不生成异常消息。内核保留所有用户状态, 包括线程虚拟寄存器。

148 处理器镜像

**C. 6 处理器镜像** **powerpc]**

内核将向所有用户级程序通过仿真: MFSPR PVR, MFSPR 和 MTSPR 的 DABR 和其他 cpu 特定的调试寄存器。

内核将仿效 MFSPR 和 MTSPR 的指令来代表 priv.edb 访问特定于 cpu 的性能监视器寄存器ileged 任务。性能监视器寄存器是全局的, 而不是每个线程的。

的 ExchangeREGISTERS系统调用访问处理器的标志。标志直接映射到 PowerPC MSR 寄存器。用户可以读取和修改以下位应用: LE, 是, SE, FE0, 和 FE1。内核还公开了其他 cpu 特定的位。

|  |  |
| --- | --- |
| 启动 | 149 |

**C. 7 启动** **powerpc]**

**苹果新世界兼容机器**

L4 必须在定义的物理位置加载到内存中由内核的精灵头。可以在启用或禁用虚拟寻址的同时启动它。L4 的执行必须从内核的 ELF 报头定义的入口点开始。

进入内核时, 支持寄存器文件参数的寄存器通过, R3–R10 根据 SVR4 ABI, 必须清除, 以向上兼容性, 除非如下所述。注册文件中的所有其他寄存器在内核条目中是未定义的。

内核可以使用 OpenFirmware 调试控制台 i/o。支持 OpenFirmOpenFirmware 虚拟模式客户端回叫地址必须在寄存器 R5 中传递给内核, OpenFirmware 必须准备好使用虚拟寻址处理客户端回调。在所有其他情况下, 注册 R5 必须为零。

启动 lo入场必须将 OpenFirmware 设备树复制到内存中, 并将其物理位置记录在内核接口页的内存描述符中。设备树的副本必须包含设备树节点的包句柄

150 支持硬件加速虚拟化

**C. 8 支持硬件加速虚拟化** **powerpc]**

除了正常的执行模式外, L4 还提供对虚拟化模式的支持。虚拟化模式很大程度上是常见的 L4's 正常 execution 模型。但是, 在虚拟化模式中, 线程可以访问扩展的 ISA, 并且限制了对 L4-specific 功能的访问。

硬件加速虚拟化模式 (HVM) 基于特权 PowerPC 指令的陷阱和模拟。届在该模式下执行的读取可以访问包含整个特权指令集的扩展体系结构 (理想情况下, 在硬件设施的限制范围内)。这样的线程可以看作是一个虚拟 CPU, 它包含所有的状态由物理 CPU 控制。除了 "正常" 页错误和已由 L4 处理的异常外, HVM 线程还会在所有事件上生成虚拟化故障, 这些错误将由连接到物理 CPU 的硬件 (以及一些事件是物理 CPU 的内部)。

虚拟化扩展引入了新的内核功能字符串:

字符串 功能

"powerpc-hvm" 内核具有虚拟化支持

**扩展线程状态**

HVM 空间中的线程表示虚拟化的物理虚拟化 HVM 空间的处理器。它拥有物理处理器的所有特权和未授权的寄存器。VM 退出会导致虚拟化故障消息, 从而有效地管理关键指令。虚拟化故障回复允许映射 m埃默里到 HVM 空间和协议项目允许读/写访问 VCPU 状态。exchangeREGISTERS通过强制虚拟化错误来授予异步访问。

**地址空间**

在硬件加速虚拟化模式中, L4 的执行和资源模型被映射到物理机器模型上。在 HVM 中执行的线程可以访问平台体系结构的特权部分, 并使用额外的内存转换进行运行。根据对双分页的硬件支持, L4 提供了一个基于来宾的 TLB 条目和主机的虚拟物理映射 (通过 L4 映射实现) 的 virtual-to-host-物理翻译的透明转换。

**S步伐C控制**

的 S步伐C控制系统调用具有结构依赖控制参数指定各种地址空间字符特点。对于 PowerPC, 控件参数具有以下语义。

**输入参数**

**控制**

|  |  |
| --- | --- |
| 0(31) | v |

* v 字段表示地址空间中所有线程的虚拟化模式。的 v字段只能为非活动地址空间指定, 并且在活动地址空间中被忽略。虚拟化的可用性

功能被宣布为一个基普特征字符串。

v=0 没有虚拟化支持的地址空间。

v=1 硬件虚拟化模式是对 PowerPC 的硬件加速虚拟化支持。在硬件虚拟化模式下, 完整的广告着装空间是空的, 在调页程序线程的控制下。线程的状态由特权 PowerPC 过程扩展sor.

**输出参数**

**控制**

|  |  |
| --- | --- |
| 0(31) | v |

* 指示启用所请求的虚拟化模式是否成功 (v = 1)。如果输入参数中的 v = 0, 则为零。

|  |  |
| --- | --- |
| 支持硬件加速虚拟化 | 151 |

**PowerPC HVM 控制传输项**

探地雷达组 0 CtrlXferItem (id = 2)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| R15(32) |  |  | 先生i+16 |  |
| . |  | . | |  |
| . |  | . | |  |
| . |  | . | |  |
| R2(32) |  |  | 先生i+3 |  |
|  |  |  | 先生i+2 |  |
| R1(32) |  |  |  |
|  |  |  | 先生i+1 |  |
| R0(32) |  |  |  |
|  |  |  | |  |
| 0xFFFF(20) | 2(8) | 1 1 0 C | 先生i |  |

探地雷达组 1 CtrlXferItem (id= 3)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| R31(32) |  |  | 先生i+16 |  |
| . |  | . | |  |
| . |  | . | |  |
| . |  | . | |  |
| R18(32) |  |  | 先生i+3 |  |
|  |  |  | 先生i+2 |  |
| R17(32) |  |  |  |
|  |  |  | 先生i+1 |  |
| R16(32) |  |  |  |
|  |  |  | |  |
| 0xFFFF(20) | 3(8) | 1 1 0 C | 先生i |  |

探地雷达扩展 CtrlXferItem (id = 4)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ip(32) |  |  | 先生i+5 |  |
|  |  |  | 先生i+4 |  |
| 铬(32) |  |  |  |
|  |  |  | 先生i+3 |  |
| 中心(32) |  |  |  |
|  |  |  | 先生i+2 |  |
| 铬(32) |  |  |  |
|  |  |  | 先生i+1 |  |
| 定影(32) |  |  |  |
|  |  |  | |  |
| 0x1f(20) | 4(8) | 1 1 0 C | 先生i |  |

MMU CtrlXferItem (id = 6)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ip(32) |  |  | 先生i+5 |  |
|  |  |  | 先生i+4 |  |
| 铬(32) |  |  |  |
|  |  |  | 先生i+3 |  |
| 中心(32) |  |  |  |
|  |  |  | 先生i+2 |  |
| 铬(32) |  |  |  |
|  |  |  | 先生i+1 |  |
| 定影(32) |  |  |  |
|  |  |  | |  |
| 0x1f(20) | 6(8) | 1 1 0 C | 先生i |  |

152 支持硬件加速虚拟化

|  |  |
| --- | --- |
| 支持硬件加速虚拟化 | 153 |

**虚拟化故障协议**

虚拟化协议是在 VCPU 线程与其注册的调页程序线程之间定义的。它替代了用于普通线程的页错误和异常协议。虚拟化故障消息被发送到与 virtualizat 相关的事件的调页程序上不直接由硬件或 L4 微内核处理的离子。默认情况下, 内核将在发送内核消息时追加下面指定的故障特定状态。与正常的故障协议类似 (请参阅部分[7,](#page91)内核将根据请求追加附加的控制转移项。添加或删除控制传输项协议的请求将使用 ExchangeREGISTERS系统调用和适当的控制传输配置项 (请参见科n[2.3)](#page30).

虚拟化故障

从调页程序:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 来宾地址/说明信息(32) | | | | 先生3 |  |
|  |  |  |  | 先生2 |  |
|  |  |  |  |  |
| 指令长度(32) | | | |  |
|  |  |  |  | 先生1 |  |
|  |  |  |  |  |
| 故障鉴定(32) | | | |  |
|  |  |  |  | 先生0 |  |
|  |  |  |  |  |
| 9 faultID(16) | 0(4) | t = 0(6) | u = 3(6) |  |

|  |  |  |
| --- | --- | --- |
| 故障 ID | 特定于实现的故障 identifiier。对于英特尔 VT x, 标识符 |  |
|  | 响应 VM 退出原因。 |  |
| 此类故障- | 有关出口原因的其他信息。 |  |
| fication | 断裂指令的长度。 |  |
| 教学 |  |
| 长度 | 来宾线性地址/关于断裂指令的附加信息 |  |
| 操作 |  |
| 信息 | . |  |
| 值 | 对于读取故障, 如果寄存器是错误寄存器的虚值, 则为 |  |
|  | VCPU 状态的一部分。 |  |

虚拟故障回复

从调页程序:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | CtrlXferItem N | | | |  | 先生c0+::: + cn+3 |  |
|  |  |  |  |  |  |  |  |
| . |  |  |  |  | . | |  |
| . |  |  |  |  | . | |  |
| . |  |  |  |  | . | |  |
|  | CtrlXferItem 0 | | | |  | 先生c0+3 |  |
|  |  |  |  |  |  | 先生1; 2 |  |
|  |  |  |  |  |  |  |
|  | MapItem/GrantItem | | | |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| 0(148) |  | 0(4)t | = 2 +Pci (6) | u = 0(6) |  | 先生0 |  |

154 支持硬件加速虚拟化

**附录 D**

PowerPC64 接口

156 虚拟寄存器

**D. 1 虚拟寄存器** **[powerpc64]**

**线程控制寄存器 (TCRs)**

TCRs 映射到内存位置。它们是作为 ppc64-specific 用户级线程控制块 (UTCB) 的一部分实现的。当前线程的 UTCB 的地址与线程的本地 ID 相同, 因此是不可变的。设置通过 T 的活动线程的 UTCB 地址HREADC控制类似于删除和重新创造。当调用 T 时, UtcbLocation 参数之间有一个固定的关联HREADC控制和 UTCB 地址。UTCB 地址在 abi 线程中提供在应用程序启动时注册 r13。然后, 可以将当前线程的 UTCB 对象作为任何其他内存对象进行访问。不能访问其他线程的 UTCBs, 即使它们在物理上是可访问的。ThreadWord0 和 ThreadWord1 可以自由使用系统软件 (例如, IDL 编译器)。内核不将语义与这些词关联起来。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | ThreadWord 1(64) | |  | + 88 |  |
|  |  |  |  | + 80 |  |
|  | ThreadWord 0(64) | |  |  |
|  |  |  |  | +72 |  |
| (48) |  | 警察标志(8) | 抢占标志(8) |  |
|  | ProcessorNo(64) | |  | +64 |  |
|  |  | |  | +56 |  |
|  | VirtualSender/ActualSender(64) | |  |  |
|  |  | |  | +48 |  |
|  | IntendedReceiver(64) | |  |  |
|  |  | |  | +40 |  |
|  | 代码(64) | |  |  |
|  |  | |  | +32 |  |
|  | XferTimeouts(64) | |  |  |
|  |  | |  | +24 |  |
|  | UserDefinedHandle(64) | |  |  |
|  |  | |  | +16 |  |
|  | ExceptionHandler(64) | |  |  |
|  |  | |  | +8 |  |
|  | 调页程序(64) | |  |  |
|  |  | |  | UTCB 地址 |  |
|  | MyGlobalId(64) | |  |  |

|  |  |
| --- | --- |
| MyLocalId = UTCB 地址(64) | r13 |
|  |  |

TCR MyLocalId 不是 UTCB 的一部分。在 PowerPC64 上, 它与 UTCB 地址相同, 可以从寄存器 r13 加载。

**消息寄存器 (夫人)**

消息注册先生0通过先生9将处理器通用寄存器文件中的本地寄存器映射到 IPC 和 LIPC 调用, 否则它们位于 UTCB。剩余的消息将映射到 UTCB 中的内存位置。先生0开始于UTCB 中的字节偏移量 512, 后续的消息寄存器在内存中跟随。

虚拟寄存器

先生0::: 9

先生0:::63[UTCB 字段]

.

.

.

157

|  |  |
| --- | --- |
| 先生9 | r23 |
| 先生8 | r22 |
| 先生7 | r21 |
| 先生6 | r20 |
| 先生5 | r19 |
| 先生4 | r18 |
| 先生3 | r17 |
| 先生2 | r16 |
| 先生1 | r15 |
| 先生0 | r14 |
|  |  |

|  |  |  |
| --- | --- | --- |
| 先生63 (64) |  | +1016 |
|  | . |  |
|  | . |  |
|  | . |  |
| 先生0 (64) |  | UTCB 地址 + 512 |

**缓冲区寄存器 (BRs)**

缓冲区寄存器映射到 UTCB 中的内存位置。br0在字节偏移量248在 UTCB, BR1在字节偏移量256等。

br0:::32[UTCB 字段]

|  |  |  |
| --- | --- | --- |
| br32 (64) |  | +504 |
| . | . |  |
| . | . |  |
| . | . |  |
| br1 (64) |  | +256 |
| br0 (64) |  | UTCB 地址 + 248 |

**未定义语义的 UTCB 内存**

内核将不关联语义, 内存位于 UTCB 地址 + 80。..UTCB 地址 + 247。应用-阳离子可以将此内存用作线程本地存储, 例如, 用于实现 L4 API。但是请注意, 该区域内的内存内容可能被覆盖摩瑞亚在消息寄存器上运行的系统调用。

上述区域未覆盖的所有未定义的 UTCB 内存可能具有内核定义的语义。

158 SYSTEMCALLS

**D. 2 Systemcalls** **Powerpc64]**

由 bctrl 或指令调用的系统调用从内核接口页中的系统调用链接字段中调用目标 (请参见页[2)](#page14).每个系统调用链接值指定一个地址相对 to 内核接口页的基址。只要返回地址包含在 lr 中, 你就可以用任何指令向适当的目标调用系统调用。

系统调用的位置在 a 的生命期内固定应用, 尽管它们可能会在应用程序的生命之外发生变化。对一组系统调用位置 prelink 应用程序是无效的。官方地点总是在基普提供。

系统调用以下定义仅 specify 一般用途寄存器的上下文。除了 KERNELIN-接口系统调用时, 假定用户可访问状态寄存器的内容被划伤。floating-point 假定寄存器在系统调用中保留。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **KERNELINterface** | **[慢 Systemcall]** | | |  |  |  |  |
| – | r0... r2 |  | KernelInterface! |  | r0... r2 |  |  |
|  |  |  |
| – | r3 |  |  |  | r3 | 基普基址 |  |
| – | r4 |  | tlbia |  | r4 | API 版本 |  |
| – | r5 |  |  | r5 | API 标志 |  |
| – | r6 |  |  |  | r6 | 内核 ID |  |
| – | r7... r31 |  |  |  | r7... r31 |  |  |
| – | lr |  |  |  | lr |  |  |
| – | 中心 |  |  |  | 中心 |  |  |
| – | 铬 |  |  |  | 铬 |  |  |
| – | 定影 |  |  |  | 定影 |  |  |

对于此系统调用, 将保留除输出寄存器以外的所有寄存器。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **ExchangeREGISTERS** | | **[Systemcall]** | |  |  |  |  |
| – | r0 |  | 交换寄存器! |  | r0 |  |  |
|  |  |  |
| – | r1 |  |  |  | r1 |  |  |
| – | r2 |  | bctrl |  | r2 |  |  |
| dest | r3 |  |  | r3 | 结果 |  |
| 控制 | r4 |  |  |  | r4 | 控制 |  |
| sp | r5 |  |  |  | r5 | sp |  |
| ip | r6 |  |  |  | r6 | ip |  |
| 标志 | r7 |  |  |  | r7 | 标志 |  |
| UserDefinedHandle | r8 |  |  |  | r8 | UserDefinedHandle |  |
| 调页程序 | r9 |  |  |  | r9 | 调页程序 |  |
| islocal | r10 |  |  |  | r10 | islocal |  |
| – | r11, r12 | |  |  | r11, r12 |  |  |
| Utcb | r13 |  |  |  | r13 | Utcb |  |
| – | r14... r29 | |  |  | r14... r29 |  |  |
| – | r30, r31 | |  |  | r30, r31 |  |  |
| – | lr |  |  |  | lr |  |  |
| ExchangeRegisters | 中心 |  |  |  | 中心 |  |  |
| – | 铬 |  |  |  | 铬 |  |  |
| – | 定影 |  |  |  | 定影 |  |  |

"标志" 是指在 msr 寄存器中保存的用户可修改的 powerpc64 处理器标志。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| SYSTEMCALLS |  |  |  |  |  | 159 |  |
| **THREADC控制** | **[特权 Systemcall]** | | |  |  |  |  |
| – | r0 |  | 线程控制! |  | r0 |  |  |
|  |  |  |
| – | r1 |  |  |  | r1 |  |  |
| – | r2 |  | bctrl |  | r2 |  |  |
| dest | r3 |  |  | r3 | 结果 |  |
| 空间 | r4 |  |  |  | r4 |  |  |
| 程序 | r5 |  |  |  | r5 |  |  |
| 调页程序 | r6 |  |  |  | r6 |  |  |
| UtcbLocation | r7 |  |  |  | r7 |  |  |
| – | r8... r12 |  |  |  | r8... r12 |  |  |
| Utcb | r13 |  |  |  | r13 | Utcb |  |
| – | r14... r29 |  |  |  | r14... r29 |  |  |
| – | r30, r31 |  |  |  | r30, r31 |  |  |
| – | lr |  |  |  | lr |  |  |
| ThreadControl | 中心 |  |  |  | 中心 |  |  |
| – | 铬 |  |  |  | 铬 |  |  |
| – | 定影 |  |  |  | 定影 |  |  |
|  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **S系统C锁[Systemcall]** | |  |  |  |  |  |  |
| – | r0 |  | SystemClock! |  | r0 |  |  |
|  |  |  |
| – | r1 |  |  |  | r1 |  |  |
| – | r2 |  | bctrl |  | r2 |  |  |
| – | r3 |  |  | r3 | 时钟 |  |
| – | r4... r12 |  |  |  | r4... r12 |  |  |
| Utcb | r13 |  |  |  | r13 | Utcb |  |
| – | r14... r29 |  |  |  | r14... r29 |  |  |
| – | r30, r31 |  |  |  | r30, r31 |  |  |
| – | lr |  |  |  | lr |  |  |
| SystemClock | 中心 |  |  |  | 中心 |  |  |
| – | 铬 |  |  |  | 铬 |  |  |
| – | 定影 |  |  |  | 定影 |  |  |
|  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **THREADS女巫[Systemcall]** | | |  |  |  |  |
| – |  | | ThreadSwitch! |  |  |  |
| r0 |  | r0 |  |
| – | r1 |  |  | r1 |  |  |
| – | r2 |  | bctrl | r2 |  |  |
| dest | r3 |  | r3 |  |  |
| – | r4... r12 |  |  | r4... r12 |  |  |
| Utcb | r13 |  |  | r13 | Utcb |  |
| – | r14... r29 |  |  | r14... r29 |  |  |
| – | r30, r31 |  |  | r30, r31 |  |  |
| – | lr |  |  | lr |  |  |
| ThreadSwitch | 中心 |  |  | 中心 |  |  |
| – | 铬 |  |  | 铬 |  |  |
| – | 定影 |  |  | 定影 |  |  |
|  |  |  |  |  |  |  |

160 SYSTEMCALLS

**S安排** **[Systemcall]**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| – | r0 | 计划! | r0 |  |  |
| – | r1 |  | r1 |  |  |
| – | r2 | bctrl | r2 |  |  |
| dest | r3 | r3 | 结果 |  |
| 时间控制 | r4 |  | r4 | 时间控制 |  |
| 处理器控制 | r5 |  | r5 |  |  |
| 优先 | r6 |  | r6 |  |  |
| 抢占控制 | r7 |  | r7 |  |  |
| – | r8... r12 |  | r8... r12 |  |  |
| Utcb | r13 |  | r13 | Utcb |  |
| – | r14... r29 |  | r14... r29 |  |  |
| – | r30, r31 |  | r30, r31 |  |  |
| – | lr |  | lr |  |  |
| 计划 | 中心 |  | 中心 |  |  |
| – | 铬 |  | 铬 |  |  |
| – | 定影 |  | 定影 |  |  |
|  |  |  |  |  |  |

**Ipc** **[Systemcall]**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| – | r0 | ipc! | r0 |  |  |
| – | r1 |  | r1 |  |  |
| – | r2 | bctrl | r2 |  |  |
| 自 | r3 | r3 | 从 |  |
| FromSpecifier | r4 |  | r4 |  |  |
| 超时 | r5 |  | r5 |  |  |
| – | r6... r12 |  | r6... r12 |  |  |
| Utcb | r13 |  | r13 | Utcb |  |
| 先生0 | r14 |  | r14 | 先生0 |  |
| 先生1 | r15 |  | r15 | 先生1 |  |
| 先生2 | r16 |  | r16 | 先生2 |  |
| 先生3 | r17 |  | r17 | 先生3 |  |
| 先生4 | r18 |  | r18 | 先生4 |  |
| 先生5 | r19 |  | r19 | 先生5 |  |
| 先生6 | r20 |  | r20 | 先生6 |  |
| 先生7 | r21 |  | r21 | 先生7 |  |
| 先生8 | r22 |  | r22 | 先生8 |  |
| 先生9 | r23 |  | r23 | 先生9 |  |
| – | r24... r29 |  | r24... r29 |  |  |
| – | r30, r31 |  | r30, r31 |  |  |
| – | lr |  | lr |  |  |
| ipc | 中心 |  | 中心 |  |  |
| – | 铬 |  | 铬 |  |  |
| – | 定影 |  | 定影 |  |  |
|  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SYSTEMCALLS | |  |  |  |  |  |  | 161 |  |
| **Lipc** | **[Systemcall]** |  |  |  |  |  |  |  |  |
|  | – | r0 |  |  | Lipc! |  | r0 |  |  |
|  |  |  |  |
|  | – | r1 |  |  |  |  | r1 |  |  |
|  | – | r2 |  |  | bctrl |  | r2 |  |  |
|  | 自 | r3 |  |  |  | r3 | 从 |  |
|  | FromSpecifier | r4 |  |  |  |  | r4 |  |  |
|  | 超时 | r5 |  |  |  |  | r5 |  |  |
|  | – | r6... r12 |  |  |  |  | r6... r12 |  |  |
|  | Utcb | r13 |  |  |  |  | r13 | Utcb |  |
|  | 先生0 | r14 |  |  |  |  | r14 | 先生0 |  |
|  | 先生1 | r15 |  |  |  |  | r15 | 先生1 |  |
|  | 先生2 | r16 |  |  |  |  | r16 | 先生2 |  |
|  | 先生3 | r17 |  |  |  |  | r17 | 先生3 |  |
|  | 先生4 | r18 |  |  |  |  | r18 | 先生4 |  |
|  | 先生5 | r19 |  |  |  |  | r19 | 先生5 |  |
|  | 先生6 | r20 |  |  |  |  | r20 | 先生6 |  |
|  | 先生7 | r21 |  |  |  |  | r21 | 先生7 |  |
|  | 先生8 | r22 |  |  |  |  | r22 | 先生8 |  |
|  | 先生9 | r23 |  |  |  |  | r23 | 先生9 |  |
|  | – | r24..r29 |  |  |  |  | r24... r29 |  |  |
|  | – | r30, r31 |  |  |  |  | r30, r31 |  |  |
|  | – | lr |  |  |  |  | lr |  |  |
|  | Lipc | 中心 |  |  |  |  | 中心 |  |  |
|  | – | 铬 |  |  |  |  | 铬 |  |  |
|  | – | 定影 |  |  |  |  | 定影 |  |  |
|  |  |  |  |  |  |  |  |  |  |
| **Unmap** | **[Systemcall]** |  |  |  |  |  |  |  |  |
|  | – | r0 |  |  | 取消! |  | r0 |  |  |
|  |  |  |  |
|  | – | r1 |  |  |  |  | r1 |  |  |
|  | – | r2 |  |  | bctrl |  | r2 |  |  |
|  | 控制 | r3 |  |  |  | r3 |  |  |
|  | – | r4... r12 |  |  |  |  | r4... r12 |  |  |
|  | Utcb | r13 |  |  |  |  | r13 | Utcb |  |
|  | – | r14... r29 |  |  |  |  | r14... r29 |  |  |
|  | – | r30, r31 |  |  |  |  | r30, r31 |  |  |
|  | – | lr |  |  |  |  | lr |  |  |
|  | 取消 | 中心 |  |  |  |  | 中心 |  |  |
|  | – | 铬 |  |  |  |  | 铬 |  |  |
|  | – | 定影 |  |  |  |  | 定影 |  |  |
|  | | | | |  |  |  |  |  |
| **S步伐C控制[特权 Systemcall]** | | | | | |  |  |  |  |
|  | – | r0 |  |  | 空间控制! |  | r0 |  |  |
|  |  |  |  |  |
|  | – | r1 |  |  |  |  | r1 |  |  |
|  | – | r2 |  |  | bctrl |  | r2 |  |  |
|  | SpaceSpecifier | r3 |  |  |  | r3 | 结果 |  |
|  | 控制 | r4 |  |  |  |  | r4 | 控制 |  |
| KernelInterfacePageArea | | r5 |  |  |  |  | r5 |  |  |
|  | UtcbArea | r6 |  |  |  |  | r6 |  |  |
|  | 定向 | r7 |  |  |  |  | r7 |  |  |
|  | –r8... r12 | |  |  |  |  | r8... r12 |  |  |
|  | Utcb | r13 |  |  |  |  | r13 | Utcb |  |
|  | – | r14... r29 |  |  |  |  | r14... r29 |  |  |
|  | – | r30, r31 |  |  |  |  | r30, r31 |  |  |
|  | – | lr |  |  |  |  | lr |  |  |
|  | SpaceControl | 中心 |  |  |  |  | 中心 |  |  |
|  | – | 铬 |  |  |  |  | 铬 |  |  |
|  | – | 定影 |  |  |  |  | 定影 |  |  |
|  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 162 |  |  |  |  |  |  | SYSTEMCALLS |  |
| **PROCESSORC控制** | | **[特权Systemcall]** | | |  |  |  |  |
| – | r0 |  | 处理器控制! | |  | r0 |  |  |
|  |  |  |
| – | r1 |  |  |  |  | r1 |  |  |
| – | r2 |  | bctrl | |  | r2 |  |  |
| ProcessorNo | r3 |  |  | r3 | 结果 |  |
| InternalFreq | r4 |  |  |  |  | r4 |  |  |
| ExternalFreq | r5 |  |  |  |  | r5 |  |  |
| 电压 | r6 |  |  |  |  | r6 |  |  |
| – | r7... r12 | |  |  |  | r7... r12 |  |  |
| Utcb | r13 |  |  |  |  | r13 | Utcb |  |
| – | r14... r29 | |  |  |  | r14... r29 |  |  |
| – | r30, r31 | |  |  |  | r30, r31 |  |  |
| – | lr |  |  |  |  | lr |  |  |
| ProcessorControl | 中心 |  |  |  |  | 中心 |  |  |
| – | 铬 |  |  |  |  | 铬 |  |  |
| – | 定影 |  |  |  |  | 定影 |  |  |
|  |  | | |  |  |  |  |  |
| **M埃默里C控制** | **[特权 Systemcall]** | | | |  |  |  |  |
| – | r0 |  | 内存控制! |  |  | r0 |  |  |
|  |  |  |  |
| – | r1 |  |  |  |  | r1 |  |  |
| – | r2 |  | bctrl |  |  | r2 |  |  |
| 控制 | r3 |  |  |  | r3 | 结果 |  |
| 属性0 | r4 |  |  |  |  | r4 |  |  |
| 属性1 | r5 |  |  |  |  | r5 |  |  |
| 属性2 | r6 |  |  |  |  | r6 |  |  |
| 属性3 | r7 |  |  |  |  | r7 |  |  |
| – | r8... r12 | |  |  |  | r8... r12 |  |  |
| Utcb | r13 |  |  |  |  | r13 | Utcb |  |
| – | r14... r29 | |  |  |  | r14... r29 |  |  |
| – | r30, r31 | |  |  |  | r30, r31 |  |  |
| – | lr |  |  |  |  | lr |  |  |
| MemoryControl | 中心 |  |  |  |  | 中心 |  |  |
| – | 铬 |  |  |  |  | 铬 |  |  |
| – | 定影 |  |  |  |  | 定影 |  |  |

|  |  |
| --- | --- |
| 内存属性 | 163 |

**D. 3 内存属性** **[powerpc64]**

powerpc64 体系结构支持以下内存/缓存属性值, 用于 M埃默里C上-

控制系统调用:

属性 值

默认 0

缓存 1

一致 2

默认属性取决于平台, 而不是为所有处理器定义所有模式。

164 异常消息格式

**D. 4 消息异常格式** **Powerpc64]**

**系统调用陷阱**

系统调用陷阱消息到异常处理程序

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 标志(64) | |  |  | 先生12 |  |
|  |  |  |  |  | 先生11 |  |
|  |  |  |  |  |  |
|  | sp(64) | |  |  |  |
|  |  |  |  |  | 先生10 |  |
|  | ip(64) | |  |  |  |
|  |  |  |  |  | 先生9 |  |
|  | r0(64) | |  |  |  |
|  |  |  |  |  | 先生8 |  |
|  | r10(64) | |  |  |  |
|  |  |  |  |  | 先生7 |  |
|  | r9(64) | |  |  |  |
|  |  |  |  |  | 先生6 |  |
|  | r8(64) | |  |  |  |
|  |  |  |  |  | 先生5 |  |
|  | r7(64) | |  |  |  |
|  |  |  |  |  | 先生4 |  |
|  | r6(64) | |  |  |  |
|  |  |  |  |  | 先生3 |  |
|  | r5(64) | |  |  |  |
|  |  |  |  |  | 先生2 |  |
|  | r4(64) | |  |  |  |
|  |  |  |  |  | 先生1 |  |
|  | r3(64) | |  |  |  |
|  |  |  |  |  | |  |
| -5(44) |  | 0(4) | t = 0(6) | 在 = 12(6) | 先生0 |  |

当用户代码执行 PowerPC sc指令, 内核将系统调用陷阱消息传递给异常处理程序。内核在处理 sc 指令时只保留部分用户状态。根据64位的保存的寄存器集, 状态也同样保存PowerPC 精灵 ABI 的函数调用。

非挥发性寄存器有: r1、r2、r13、r31、CR2..。CR4

可变寄存器有: r0, r3, r12, LR, 定影, CR0, CR1, CR5..。CR7

线程虚拟寄存器也可能受到重创。

**一般陷阱**

异常处理程序的一般陷阱消息

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 异常消息格式 | |  |  | 165 | | |  |
|  |  |  |  |  |  | 先生7 |  |
|  |  | ErrorAddress(64) | |  |  |  |
|  |  |  |  |  |  | 先生6 |  |
|  |  |  |  |  |  |  |
|  |  | LocalID(64) | |  |  |  |
|  |  |  |  |  |  | 先生5 |  |
|  |  |  |  |  |  |  |
|  |  | 代码(64) | |  |  |  |
|  |  |  |  |  |  | 先生4 |  |
|  |  |  |  |  |  |  |
|  |  | ExceptionNo(64) | |  |  |  |
|  |  |  |  |  |  | 先生3 |  |
|  |  |  |  |  |  |  |
|  |  | 标志(64) | |  |  |  |
|  |  |  |  |  |  | 先生2 |  |
|  |  |  |  |  |  |  |
|  |  | sp(64) | |  |  |  |
|  |  |  |  |  |  | 先生1 |  |
|  |  | ip(64) | |  |  |  |
|  | |  |  |  |  | |  |
|  | -5(44) |  | 0(4) | t = 0(6) | 在 = 6 = 7(6) | 先生0 |  |

内核合成异常消息以响应特定于体系结构的事件。某些陷阱由内核处理, 因此不生成异常消息。提供错误地址的异常使用 ErrorAddress 寄存器并指定7非类型化单词, 否则只会有6非类型化单词发送.内核保留所有用户状态, 包括线程虚拟寄存器。

对于某些异常, 下面是通用陷阱 ExceptionNo 的值表:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 异常 | ExceptionNo | 代码 | 交付 | ErrorAddress |
| 系统复位 | 0x100 | - | 不 | - |
| 机器检查 | 0x200 | - | 不 | - |
| dsi | 0x300 | DSISR | 如果不分页相关 | 是的 |
| 的内容 | 0x400 | - | 如果不分页相关 | 不 |
| 中断 | 0x500 | - | 不 | 不 |
| 对 | 0x600 | DSISR | 是的 | 是的 |
| 程序 | 0x700 | - | 是的 | 是的 |
| FPU 不可用 | 0x800 | - | 不 | - |
| Decrementer | 0x900 | - | 不 | - |
| 系统打电话 | 0xc00 | - | 不 | - |
| 跟踪 | 0xd00 | - | 如果 kdb 不使用 | 不 |
| FPU 协助 | 0xe00 | - | 是的 | 不 |
| 性能 | 0xf00 | - | 是的 | 不 |
| 断点 | 0x1300 | - | 是的 | 不 |
| 软补丁 | 0x1500 | - | 是的 | 不 |
| 维护 | 0x1600 | - | 是的 | 不 |
| 仪器 | 0x2000 | - | 是的 | 不 |

注意, 并非所有这些例外将通过异常 IPC 来传递。一些将由内核处理。已传递的异常在上表的最后一列中指示。

166 启动

**D. 5 启动** **[powerpc64]**

**IBM OpenFirmware 机**

L4 必须在内核的 ELF 报头定义的物理位置加载到内存中。可以在启用或禁用虚拟寻址的同时启动它。L4 的执行必须从内核的 ELF 报头定义的入口点开始。

进入内核时,支持寄存器文件参数传递的寄存器 (根据开放电源 ABI R3–R10) 必须清除以使其更高的兼容性, 除非如下所述。注册文件中的所有其他寄存器在内核条目中是未定义的。

内核 may 使用 OpenFirmware 调试控制台 i/o。为了支持 OpenFirmware i/o, 必须在注册 R5 中将 OpenFirmware 虚拟模式客户端回叫地址传递给内核, 并且 OpenFirmware 必须准备好使用虚拟 addressin 处理客户端回调g？？？。在所有其他情况下, 注册 R5 必须为零。

引导加载程序必须将 OpenFirmware 设备树复制到内存, 并将其物理位置记录在内核接口页的内存描述符中。设备树的副本必须包括包装设备树节点的 e 句柄

**附录 E**

通用 BootInfo

168 通用 BOOTINFO

**E. 1 通用 BootInfo** **[数据结构]**

通用 BootInfo 结构包含特定于引导加载程序数据, 如加载的模块或文件, 系统表的位置等。数据结构可以位于内存中的任何位置, 但必须以单词大小对齐。

BootInfo 结构是一个纯启动加载程序特定的对象。即, 内核不协会ciate 任何语义及其内容。引导加载程序可以自由选择是否提供 BootInfo 结构。启动没有通用 BootInfo 结构的系统是完全有效的。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 第一 BootInfo 记录 | |  |  |  | 第一项 |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  | |
|  |  | |  | num项 |  | +10/+20 |
| 第一项 | 大小 |  | 版本 | 魔法 |  | BootInfo |
|  |  |  |  |  |  |  |
| + C/+18 | +8/+10 |  | +4/+8 |  | +0 |  |

bootinfo 结构的基址由内核接口页中的 bootinfo 字段指定 (请参见页[4)](#page16).请注意, BootInfo 字段指定的基址是物理地址。在虚拟内存上运行的应用程序必须通过其他方式确定 BootInfo 结构在其自己的地址空间中的位置。

BootInfo 描述

魔法

版本

大小

第一项

幻数0x14B0021D。魔术还决定了结构的设置 (即, 值0x1D02B014 表示字节是错误的)。BootInfo 结构的单词大小由内核接口中指定的单词大小定义e (请参阅页面[3)](#page15).

BootInfo 结构的 API 版本。本文档描述版本1。请注意, BootInfo 记录本身的任何更改都不会影响主 BootInfo 结构中的版本。这使 BootInfo 记录要添加或修改, 而不引入与解析 BootInfo 结构的程序的主要 incompatibili 关系。只有添加/修改的 BootInfo 记录类型才会受到更新的影响。

完整的 BootInfo st 的大小 (以字节为单位)ructure, 包括这些记录所引用的所有 BootInfo 记录和数据。

指向第一个 BootInfo 记录。第一个条目是相对于 BootInfo 结构本身的基址而提供的地址。

Num 条目 BootInfo 结构中的 BootInfo 记录数。

通用 BootInfo 记录

BootInfo 记录的确切结构由记录的类型确定。仅为所有 BootInfo 记录类型定义记录的三首字。

|  |  |  |
| --- | --- | --- |
| Offse下一页 | 版本 | 类型 |
|  |  |  |
| +8/+10 | +4/+8 | +0 |

类型 指定 BootInfo 记录的类型。

通用 BOOTINFO

版本

169

指定 BootInfo 记录类型的 API 版本。增加版本的BootInfo 记录类型也不需要增加主 BootInfo 版本。BootInfo 记录的更高版本保证向后兼容旧版本。

|  |  |
| --- | --- |
| 偏移下 | 中的偏移量 (以字节为单位)。下一个 BootInfo 记录。请注意, 偏移量可能因记录而异 |
|  | 记录, 甚至是同一类型的记录。这使引导加载程序具有可变长度 |
|  | 记录, 在记录之间放置数据, 或以其他方式对齐记录以方便实施信息. |
|  | 假定与某一记录类型的特定版本相关联的偏移量是 |
|  | 不断. |
|  |  |

**方便的编程接口**

#include<l4/bootinfo>

结构**BootR欧共体**f 字原始 [\*] g

Bool BootInfo 有效 (空 \* BootInfo)

检查指定的 BootInfo 结构是否有效 (即, 幻数和版本号是否正确)。

字 BootInfo 大小 (空 \* BootInfo)

提供 BootInfo 结构的大小 (以字节为单位)。假设 BootInfo specifies 一个有效的 BootInfo 结构。

BootRec \* BootInfo FirstEntry (空 \* BootInfo)

提供 BootInfo 结构的第一个 BootInfo 记录。假定 BootInfo 指定了一个有效的 BootInfo 结构。

单词 BootInfo 条目 (空 \* BootInfo)

提供 BootInfo 结构中的 BootInfo 记录数。假定 BootInfo 指定了一个有效的 BootInfo 结构。

文字类型 (BootRec \* BootRec) [BootRec 类型]

传递 BootInfo 记录的类型。

BootRec \* 下一页 (BootRec \* BootRec) [BootRec 下一页]

提供下一 BootInfo 记录。未定义 BootInfo 结构中最后一个 BootInfo 记录返回的值。

170 BOOTINFO 记录

**E. 2 BootInfo 记录** **Bootinfo]**

BootInfo 记录可以列出以任何顺序。本节列出当前定义的 BootInfo 记录。遇到未知 BootInfo 记录的程序可以跳过使用 "无处不在的偏移" 下一字段的记录。

简单模块 简单模块 BootInfo 记录指定二进制由引导加载程序加载到主存中的文件。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 命令关闭 | 大小 | +10/+20 |
|  |  |  |  |  |
| 开始 | 偏移下 | 版本 = 1 | 类型 = 0x1 |  |
|  |  |  |  |  |

开始

大小

命令关闭

+ C/+18 +8/+10 +4/+8 +0

加载的第一个字节的物理地址模块.

已加载模块的大小 (以字节为单位)。

与加载模块关联的命令行的地址, 如果不存在命令行, 则为0。Ad 礼服是相对于当前 BootInfo 记录的基址指定的。

简单可执行文件的 简单可执行文件 BootInfo 记录指定的可执行文件已加载到 主存储器并由引导加载程序重新定位。该记录只能用单个代码、数据和 bss 部分指定简单的可执行文件。

Pstart

Vstart

大小

初始 IP

标志

标签

命令关闭

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 命令关闭 | 标签 | 标志 | 初始 IP |  | +30/+60 |  |
|  |  |  |  |  | +20/+40 |  |
| 大小 | Vstart | Pstart | 数据. 大小 |  |  |
|  |  |  |  |  | +10/+20 |  |
| 数据. Vstart | 数据. Pstart | 文本. 大小 | 文本. Vstart |  |  |
|  |  |  |  |  |  |  |
| 文本. Pstart | 偏移下 | 版本 = 1 | 类型 = 0x2 |  |  |  |
|  |  |  |  |  |  |  |
| + C/+18 | +8/+10 | +4/+8 |  | +0 |  |  |

加载的可执行文件的代码/数据/bss 部分中的第一个字节的物理地址。

在代码/数据/bss 中的第一个字节的虚拟地址部分加载的可执行文件。

代码/数据/bss 部分的大小 (以字节为单位)。

加载的可执行文件的入口点的虚拟地址。

已加载可执行文件的标志 (由引导加载程序或应用程序定义)。请注意, 常规应用程序可能不ecessarily 对 "标志" 字段具有写权限。

自由可用的单词 (由引导加载程序或应用程序定义)。请注意, 常规应用-阳离子不一定对标签字段具有写权限。

命令行关联的地址d 带加载的可执行文件, 如果不存在命令行, 则为0。

地址是相对于当前 BootInfo 记录的基址指定的。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| BOOTINFO 记录 |  |  |  |  | 171 | |
| EFI 表 | EFI 表 BootInfo 记录指定位置和大小的 EFI 内存映射, 和 | | | | | |
|  | EFI 系统表的位置。 | |  |  |  |  |
|  |  |  |  |  |  |  |
|  | Memdesc 版本 | Memdesc 尺寸 | Memmap 尺寸 | Memmap |  | +10/+20 |
|  |  |  |  |  |  |  |
|  | systab | 偏移下 | 版本 = 1 | 类型 = 0x101 |  |  |
|  |  |  |  |  |  |  |
|  | + C/+18 | +8/+10 | +4/+8 |  | +0 |  |
| systab | efi 系统表的物理地址, 如果不存在 efi 系统表, 则为0。 | | | |  |  |
| Memmap | EFI 内存映射的物理地址。未定义如果 Memmap 大小 = 0。 | | | |  |  |

Memmap 尺寸 efi 内存映射的大小 (以字节为单位), 如果不存在 efi 内存映射, 则为0。

Memdesc 尺寸 EFI 内存映射中描述符项的大小 (以字节为单位)。未定义如果 Memmap 大小 = 0。

Memdesc 版本 EFI 内存映射中的描述符项的版本。未定义如果 Memmap 大小 = 0。

引导信息 引导信息 BootInfo记录指定引导头中第一个字节的位置。

|  |  |  |  |
| --- | --- | --- | --- |
| 引导地址 | 偏移下 | 版本 = 1 | 类型 = 0x102 |
|  |  |  |  |
| + C/+18 | +8/+10 | +4/+8 | +0 |

引导地址 引导头中第一个字节的物理地址。

**方便编程接口**

#include<l4/bootinfo>

Word BootInfo 模块

字 BootInfo SimpleExec

字 BootInfo EFITables

字 BootInfo 引导

Word 模块启动 (BootRec \* b)

Word 模块大小 (BootRec \* b)

提供开始和大小指定的引导模块。

char \* 模块命令 (BootRec \* b)

传递指定的引导模块的命令行, 如果命令行不存在, 则为0。

字 SimpleExec TextPstart (BootRec \* b)

字 SimpleExec TextVstart (BootRec \* b)

词SimpleExec TextSize (BootRec \* b)

字 SimpleExec DataPstart (BootRec \* b)

字 SimpleExec DataVstart (BootRec \* b)

字 SimpleExec DataSize (BootRec \* b)

字 SimpleExec BssPstart (BootRec \* b)

字 SimpleExec BssVstart (BootRec \* b)

172

BOOTINFO 记录

词

SimpleExec BssSize (BootRec \* b)

提供物理起始地址、虚拟起始地址和指定的代码/数据/bss 部分的大小可执行.

字 SimpleExec InitialIP (BootRec \* b)

为指定的可执行文件提供入口点的虚拟地址。

单词 SimpleExec 标志 (BootRec \* b)

空 SimpleExec 设置标志 (BootRec b, 字 w)

提供/设置 "标志" 字段。指定的可执行文件。

单词 SimpleExec 标签 (BootRec \* b)

空 SimpleExec 设置标签 (BootRec b, 字 w)

为指定的可执行文件提供/设置标签字段。

char \* SimpleExec 命令 (BootRec \* b)

传递指定的命令行可执行文件, 如果命令行不存在, 则为0。

字 EFI Systab (BootRec \* b)

提供 EFI 系统表, 如果系统表不存在, 则为0。

字 EFI Memmap (BootRec \* b)

字 EFI MemmapSize (BootRec \* b)

字 EFI MemdescSize (BootRec \* b)

在Rd EFI MemdescVersion (BOOTREC \* b)

提供 EFI 内存映射的位置、内存映射的大小、内存映射描述符项的大小以及内存映射描述符条目的版本。如果 EFI MemmapSize () 提供 0, 则其他返回值是未定义的。

字 MBI 地址 (BootRec \* b)

提供引导头中第一个字节的物理位置。

**附录 F**

发展评论

这些备注将设计过程从版本2到版本4。

**F. 1** **异常处理**

在 L4 中为异常处理确定的当前模型是将异常处理程序线程与系统中的每个线程关联 (请参见页[72)](#page84).之所以选择此模型, 是因为它允许我们一般不会在 API 中引入任何新的概念来处理异常。它也与当前页面错误处理模型非常相似。

另一个用于异常处理的模型是使用回调。使用此模型的指令指针回调函数和指向异常状态保存区域的指针与每个线程关联。捕获异常后, 内核将异常的原因存储到保存区域, 并将执行转移到异常回调函数。

它是 e晕船说, 回调模型可以比 ipc 模型更快, 因为回调模型可能只需要一个控制转移到内核中, 而 ipc 模型则需要至少两个。然而, 选择 IPC 模型是因为它引入了 no 新的机制进入内核, 我们目前还没有意识到任何真实的生活, nario 在那里额外的性能会使你获得很大的收益。证明这些说法是错误的, 存在着挑战。看到[http://l4hq.org/f联合国/](http://l4hq.org/fun/)为挑战的规则。

174 附录 F. 发展备注

**触发、类型和常量的表**

|  |  |  |
| --- | --- | --- |
|  | 使用的系统调用 | 页 |
|  |  |  |
| AbortIpc 和停止 (ThreadId t) ThreadState | ExchangeREGISTERS | [22](#page34) |
| AbortIpc 和停止 (ThreadId t, 字和 sp, ip, 旗) ThreadState | ExchangeREGISTERS | [22](#page34) |
| AbortReceive 和停止 (ThreadId t) ThreadState | ExchangeREGISTERS | [22](#page34) |
| AbortReceive并停止 (ThreadId t, 字和 sp, ip, 旗) ThreadState | ExchangeREGISTERS | [22](#page34) |
| AbortSend 和停止 (ThreadId t) ThreadState | ExchangeREGISTERS | [22](#page34) |
| AbortSend 和停止 (ThreadId t, 单词和 sp, ip, 标志)ThreadState | ExchangeREGISTERS | [22](#page34) |
| 接受 (承兑 a、MsgBuffer 和 b) 无效 | 没有– | [63](#page75) |
| 接受 (承兑 a) 无效 | 没有– | [63](#page75) |
| 接受 () 承兑 | 没有– | [63](#page75) |
| 受体数据类型 | – N/ | [62](#page74) |
| + (受体 l r) 受体 | 没有– | [62](#page74) |
| (受体 l, r)受体 | 没有– | [63](#page75) |
| ActualSender () ThreadId | 没有– | [17](#page29) |
| ActualSender () ThreadId | 没有– | [70](#page82) |
| 地址 (Fpage f) 字 | 没有– | [41](#page53) |
| AllocateNewCacheLines CacheAllocationHint 常量 | – N/ | [106](#page118) |
| AllocateNewCacheLines CacheAllocationHint 常量 | – N/ | [131](#page143) |
| AllocateOnlyNewL1CacheLines CacheAllocationHint 常量 | – N/ | [106](#page118) |
| AllocateOnlyNewL1CacheLines CacheAllocationHint 常量 | – N/ | [131](#page143) |
| anylocalthread ThreadId 常量 | – N/ | [15](#page27) |
| anythread ThreadId 常量 | – N/ | [15](#page27) |
| ApiFlags () 字 | 没有– | [8](#page20) |
| ApiVersion () 字 | 没有– | [8](#page20) |
| 附加 (MsgBuffer 和 b, StringItem \* s) 无效 | 没有– | [63](#page75) |
| 附加 (MsgBuffer 和 b, StringItem s) 无效 | 没有– | [63](#page75) |
| 附加 (msg 和味精, FPURegsCtrlXferItem c) 无效 | 没有– | [110](#page122) |
| 附加 (msg 和味精, GPRegsCtrlXferItem c) 无效 | 没有– | [110](#page122) |
| 附加 (msg 和味精, GrantItem g) 无效 | 没有– | [53](#page65) |
| 附加 (msg 和味精, MapItem m) 无效 | 没有– | [53](#page65) |
| 附加 (msg 和味精, StringItem s) 无效 | 没有– | [53](#page65) |
| 附加 (msg 和味精, StringItem) 无效 | 没有– | [53](#page65) |
| 附加 (msg 和味精, 字 w) 无效 | 没有– | [53](#page65) |
| ArchitectureSpecificMemoryType 字常量 | – N/ | [9](#page21) |
| AssociateInterrupt (ThreadId InterruptThread, InterruptHandler) 字 | 没有– | [25](#page37) |
| BootInfo EFITables 字常量 | – N/ | [171](#page183) |
| BootInfo 词条 (空 \* BootInfo) 词 | 没有– | [169](#page181) |
| BootInfo FirstEntry (空 \* BootInfo) BootRec \* | 没有– | [169](#page181) |
| BootInfo 模块字常量 | – N/ | [171](#page183) |
| BootInfo引导字常量 | – N/ | [171](#page183) |
| BootInfo SimpleExec 字常量 | – N/ | [171](#page183) |
| BootInfo 大小 (空 \* BootInfo) 字 | 没有– | [169](#page181) |
| BootInfo 有效 (void \* BootInfo) Bool | 没有– | [169](#page181) |
| BootInfo (空 \* KernelInterface) 字 | 没有– | [9](#page21) |
| BootLoaderSpecificMemoryType 字常量 | – N/ | [9](#page21) |
| BootRec 数据类型 | – N/ | [169](#page181) |
| CacheAllocationHint 数据类型 | – N/ | [60](#page72) |
| == (CacheAllocationHint l, r) Bool | 没有– | [61](#page73) |
| CacheAllocationHint (StringItem s) CacheAllocationHint | 没有– | [61](#page73) |
| CachingEnabledMemory 字常量 | – N/ | [144](#page156) |

|  |  |  |
| --- | --- | --- |
| 176 | 触发、类型和常量的表 | |
|  | 使用的系统调用 | 页 |
|  |  |  |
| CachingInhibitedMemory 字常量 | – N/ | [144](#page156) |
| 致电 (ThreadId to) MsgTag | Ipc | [68](#page80) |
| 电话 (ThreadId, 时间 SndTimeout, RcvTimeout) MsgTag | Ipc | [69](#page81) |
| 清除 (MsgBuffer 和 b) 无效 | 没有– | [63](#page75) |
| 清除 (msg 和味精) 无效 | 没有– | [53](#page65) |
| 时钟数据类型 | – N/ | [28](#page40) |
| (时钟 l, int r)时钟 | 没有– | [28](#page40) |
| + (时钟 l, int r) 时钟 | 没有– | [28](#page40) |
| (时钟 l, r) Bool | 没有– | [28](#page40) |
| (时钟 l, r) Bool | 没有– | [28](#page40) |
| == (时钟 l, r) Bool | 没有– | [28](#page40) |
| >> = (时钟 l, r) Bool | 没有– | [28](#page40) |
| >> (时钟 l, r) Bool | 没有– | [28](#page40) |
| (时钟 l, r)时钟 | 没有– | [28](#page40) |
| + (时钟 l, r) 时钟 | 没有– | [28](#page40) |
| (时钟 l, Word64 r)时钟 | 没有– | [28](#page40) |
| + (时钟 l, Word64 r) 时钟 | 没有– | [28](#page40) |
| Clr CopFlag (单词 n) 无效 | 没有– | [17](#page29) |
| Clr CopFlag (单词 n) 无效 | 没有– | [73](#page85) |
| CompleteAddressSpace Fpage 常量 | – N/ | [41](#page53) |
| CompoundString(StringItem s)bool | 没有– | [60](#page72) |
| ConventionalMemoryType 字常量 | – N/ | [9](#page21) |
| CtrlXferItem (CtrlXferItem 和 c) Bool | 没有– | [58](#page70) |
| CtrlXferItem 数据类型 | – N/ | [58](#page70) |
| CtrlXferItemInit (CtrlXferItem 和 c, 单词 id) 无效 | 没有– | [58](#page70) |
| CtrlXferItemsAcceptor 受体常量 | – N/ | [62](#page74) |
| DeassociateInterrupt (ThreadId InterruptThread) 字 | 没有– | [25](#page37) |
| DedicatedMemoryType 字常量 | – N/ | [9](#page21) |
| DefaultMemory 字常量 | – N/ | [107](#page119) |
| DefaultMemory 字常量 | – N/ | [132](#page144) |
| DefaultMemory 字const | – N/ | [144](#page156) |
| DefaultMemory 字常量 | – N/ | [77](#page89) |
| DisablePreemption () Bool | 没有– | [37](#page49) |
| DisablePreemptionFaultException () Bool | 没有– | [37](#page49) |
| DoNotAllocateNewCacheLines CacheAllocationHint 常量 | – N/ | [106](#page118) |
| DoNotAllocateNewCacheLines CacheAllocationHint 常量 | – N/ | [131](#page143) |
| 电喷 MemdescSize (BootRec \* b) 字 | 没有– | [172](#page184) |
| 电喷 MemdescVersion (BootRec \* b) 字 | 没有– | [172](#page184) |
| 电喷 Memmap (BootRec \* b) 字 | 没有– | [172](#page184) |
| 电喷 MemmapSize (BootRec \* b) 字 | 没有– | [172](#page184) |
| 电喷 Systab (BootRec \* b) 字 | 没有– | [172](#page184) |
| EnablePreemption () Bool | 没有– | [37](#page49) |
| EnablePreemptionFaultException () Bool | 没有– | [37](#page49) |
| ErrInvalidParam 字常量 | – N/ | [36](#page48) |
| ErrInvalidParam 字常量 | – N/ | [77](#page89) |
| ErrInvalidScheduler 字常量 | – N/ | [25](#page37) |
| ErrInvalidSpace 字常量 | – N/ | [25](#page37) |
| ErrInvalidSpace 字const | – N/ | [47](#page59) |
| ErrInvalidThread 字常量 | – N/ | [22](#page34) |
| ErrInvalidThread 字常量 | – N/ | [25](#page37) |
| ErrInvalidThread 字常量 | – N/ | [35](#page47) |
| ErrKipArea字常量 | – N/ | [47](#page59) |
| ErrNoMem 字常量 | – N/ | [25](#page37) |
| ErrNoPrivilege 字常量 | – N/ | [25](#page37) |
| ErrNoPrivilege 字常量 | – N/ | [35](#page47) |
| ErrNoPrivilege字常量 | – N/ | [47](#page59) |
| ErrNoPrivilege 字常量 | – N/ | [75](#page87) |
| ErrNoPrivilege 字常量 | – N/ | [77](#page89) |
| 代码 () 字 | 没有– | [17](#page29) |
| 代码 () 字 | 没有– | [22](#page34) |
| 代码 () 字 | 没有– | [25](#page37) |
| 代码 () 字 | 没有– | [35](#page47) |
| 代码 () 字 | 没有– | [47](#page59) |

|  |  |  |
| --- | --- | --- |
| 表的触发、类型和常量 |  | 177 |
|  | 使用的系统调用 | 页 |
|  |  |  |
| 代码 () 字 | 没有– | [69](#page81) |
| 代码 () 字 | 没有– | [75](#page87) |
| 代码 () 字 | 没有– | [77](#page89) |
| ErrUtcbArea 字const | – N/ | [25](#page37) |
| ErrUtcbArea 字常量 | – N/ | [47](#page59) |
| ExceptionHandler () ThreadId | 没有– | [17](#page29) |
| ExceptionHandler () ThreadId | 没有– | [72](#page84) |
| ExchangeRegisters (ThreadId, 文字控制, sp, ip, 旗帜, UserDe | ExchangeREGISTERS | [21](#page33) |
| finedHandle, ThreadId 调页程序, 单词和旧的控制, 老 sp, 旧的 ip, 旧的旗帜, |  |  |
| 老 UserDefinedHandle, ThreadId 老调页程序) ThreadId |  |  |
| 可执行字常量 | – N/ | [41](#page53) |
| ExternalFreq (ProcDesc 和 p) 字 | 没有– | [10](#page22) |
| 功能 (void \* KernelInterface, 字 num) 字符 \* | 没有– | [9](#page21) |
| 冲洗 (Fpage f)Fpage | Unmap | [44](#page56) |
| 冲洗 (字 n, Fpage 和 [n] fpages) 空 | Unmap | [44](#page56) |
| Fpage 数据类型 | – N/ | [40](#page52) |
| (Fpage f, 单词 AccessRights)Fpage | 没有– | [41](#page53) |
| +(Fpage f, 单词 AccessRights)Fpage | 没有– | [41](#page53) |
| FpageLog2 (词 BaseAddress, int Log2FpageSize 64) Fpage | 没有– | [41](#page53) |
| Fpage (词 BaseAddress, int FpageSize 1K) Fpage | 没有– | [41](#page53) |
| FPURegsCtrlXferItem 数据类型 | – N/ | [109](#page121) |
| FullyAccessible 字常量 | – N/ | [41](#page53) |
| 获取 (msg 和味精, 单词 t, CtrlXferItem 和 c) 字 | 没有– | [54](#page66) |
| 获取(味精和味精, 字 t, GrantItem 和 g)词 | 没有– | [54](#page66) |
| 获得 (味精, 味精, MapItem 和 m) 字 | 没有– | [54](#page66) |
| 获取 (msg 和味精, 单词 t, StringItem s) 词 | 没有– | [54](#page66) |
| 获取 (msg 和味精,单词和 ut, fMapItem, GrantItem, StringItemg 和项目) 无效 | 没有– | [53](#page65) |
| 获取 (msg 和味精, 单词 u) | 没有– | [54](#page66) |
| 获取 (msg 和味精, 单词 u, 单词和 w) 无效 | 没有– | [54](#page66) |
| GetStatus (平安险ge f) Fpage | 没有– | [44](#page56) |
| GlobalId (ThreadId t) ThreadId | ExchangeREGISTERS | [15](#page27) |
| GlobalId (ThreadId t) ThreadId | ExchangeREGISTERS | [21](#page33) |
| GlobalId (词 threadno, 版本)threadid | 没有– | [15](#page27) |
| GlobalMemory 字常量 | – N/ | [144](#page156) |
| GPRegsCtrlXferItem 数据类型 | – N/ | [109](#page121) |
| GrantItem 数据类型 | – N/ | [57](#page69) |
| GrantItem(Fpage f, 单词 SndBase)GrantItem | 没有– | [57](#page69) |
| GrantItem (GrantItem g) Bool | 没有– | [57](#page69) |
| GuardedMemory 字常量 | – N/ | [144](#page156) |
| 高 (MemoryDesc 和 m) 字 | 没有– | [9](#page21) |
| IntendedReceiver () ThreadId | 没有– | [17](#page29) |
| IntendedReceiver () ThreadId | 没有– | [69](#page81) |
| InternalFreq (ProcDesc 和 p) 字 | 没有– | [10](#page22) |
| IoFpageLog2 (字BaseAddress, int Log2FpageSize 16) Fpage | 没有– | [130](#page142) |
| IoFpageLog2 (词 BasePort, int Log2FpageSize = 16) Fpage | 没有– | [103](#page115) |
| IoFpagePort (Fpage f) 字 | 没有– | [103](#page115) |
| IoFpageSize (Fpage f) 字 | 没有– | [103](#page115) |
| IoFpageSizeLog2 (Fpage f) 字 | 没有– | [103](#page115) |
| IoFpage (词 BaseAddress, int FpageSize) Fpage | 没有– | [130](#page142) |
| IoFpage (词 BasePort,int FpageSize) Fpage | 没有– | [103](#page115) |
| IpcFailed (MsgTag t) Bool | 没有– | [69](#page81) |
| IpcPropagated (MsgTag t) Bool | 没有– | [69](#page81) |
| IpcRedirected (MsgTag t) Bool | 没有– | [69](#page81) |
| IpcSucceeded (MsgTag t) Bool | 没有– | [69](#page81) |
| Ipc (ThreadId, FromSpecifier, 单词超时, ThreadId 和来自) MsgTag | Ipc | [68](#page80) |
| IpcXcpu (MsgTag t) Bool | 没有– | [69](#page81) |
| IsGlobalId (ThreadId t) Bool | 没有– | [15](#page27) |
| IsIoFpage (Fpage f) Bool | 没有– | [103](#page115) |
| IsLocalId (ThreadId t) Bool | 没有– | [15](#page27) |
| IsNilFpage (Fpage f)bool | 没有– | [41](#page53) |
| IsNilThread (ThreadId t) Bool | 没有– | [15](#page27) |
| IsVirtual (MemoryDesc 和 m) Bool | 没有– | [9](#page21) |
| KernelGenDate (空 \* KernelInterface, 词和年, 月, 日) 无效 | 没有– | [8](#page20) |

|  |  |  |  |
| --- | --- | --- | --- |
| 178 | 触发、类型和常量的表 | | |
|  |  | 使用的系统调用 | 页 |
|  |  |  |  |
| KernelId () 字 |  | 没有– | [8](#page20) |
| KernelInterface () 无效 \* |  | KERNELINterface | [8](#page20) |
| KernelInterface (词和 ApiVersion, ApiFlags, KernelId) 无效 \* |  | KERNELINterface | [8](#page20) |
| KernelSupplier (空 \* KernelInterface) 字 |  | 没有– | [8](#page20) |
| KernelVersionString (void \*KernelInterface) char \* |  | 没有– | [9](#page21) |
| KernelVersion (空 \* KernelInterface) 字 |  | 没有– | [8](#page20) |
| KipAreaSizeLog2 (空 \* KernelInterface) 字 |  | 没有– | [9](#page21) |
| 标签 (msg 和味精) 字 |  | 没有– | [53](#page65) |
| 标签 (Msg 标签 t) 字 |  | 没有– | [52](#page64) |
| LargeSpace 字常量 |  | – N/ | [105](#page117) |
| Lcall (ThreadId) MsgTag |  | Lipc | [69](#page81) |
| Lipc (ThreadId,FromSpecifier, 单词超时, ThreadId 和来自) MsgTag | | Lipc | [68](#page80) |
| LoadBR (int i, 字 w) 无效 |  | 没有– | [11](#page23) |
| LoadBR (int i, 字 w) 无效 |  | 没有– | [63](#page75) |
| LoadBRs (int i, k, 词和[k]) 无效 |  | 没有– | [11](#page23) |
| LoadBRs (int i, k, 词和 [k]) 无效 |  | 没有– | [63](#page75) |
| LoadMR (int i, 字 w) 无效 |  | 没有– | [11](#page23) |
| LoadMR (int i, 字 w) 无效 |  | 没有– | [54](#page66) |
| LoadMRs (int i; k, 单词和 [k] w) 无效 |  | 没有– | [11](#page23) |
| LoadMRs (int i; k, 单词和 [k] w) 无效 |  | 没有– | [54](#page66) |
| 加载 (msg 和味精) 无效 |  | 没有– | [53](#page65) |
| LocalId(ThreadId t)threadid |  | ExchangeREGISTERS | [15](#page27) |
| LocalId (ThreadId t) ThreadId |  | ExchangeREGISTERS | [21](#page33) |
| LocalMemory 字常量 |  | – N/ | [144](#page156) |
| 低 (MemoryDesc 和 m) 字 |  | 没有– | [9](#page21) |
| LreplyWait (ThreadId, ThreadId) MsgTag |  | Lipc | [69](#page81) |
| MapGrantItems (受体 a) Bool |  | 没有– | [63](#page75) |
| MapGrantItems (Fpage RcvWindow) 受体 |  | 没有– | [62](#page74) |
| MapItem 数据类型 |  | – N/ | [55](#page67) |
| MapItem (Fpage f, 字 SndBase) MapItem |  | 没有– | [55](#page67) |
| MapItem (MapItem m) Bool |  | 没有– | [56](#page68) |
| MBI 地址(BootRec \* b)词 |  | 没有– | [172](#page184) |
| MemoryControl (词控制, 词和属性 [4]) 词 |  | M埃默里C控制 | [77](#page89) |
| MemoryDesc 数据类型 |  | – N/ | [8](#page20) |
| MemoryDesc (void \*KernelInterface, 字 num) MemoryDesc \* |  | 没有– | [9](#page21) |
| 模块命令 (BootRec \* b) 字符 \* |  | 没有– | [171](#page183) |
| 模块大小 (BootRec \* b) 字 |  | 没有– | [171](#page183) |
| 模块启动 (BootRec \* b)词 |  | 没有– | [171](#page183) |
| MsgBuffer 数据类型 |  | – N/ | [63](#page75) |
| Msg 数据类型 |  | – N/ | [53](#page65) |
| MsgTag 数据类型 |  | – N/ | [52](#page64) |
| == (MsgTag l, r) Bool |  | 没有– | [52](#page64) |
| MsgTag (味精和味精) MsgTag |  | 没有– | [53](#page65) |
| MsgTag () MsgTag |  | 没有– | [52](#page64) |
| + (MsgTag t, 单词标签) MsgTag |  | 没有– | [52](#page64) |
| + = (受体 l, r) 受体 |  | 没有– | [62](#page74) |
| = (受体 l r) 受体 |  | 没有– | [63](#page75) |
| != (CacheAllocationHint l, r) Bool |  | 没有– | [61](#page73) |
| != (时钟 l, r) Bool |  | 没有– | [28](#page40) |
| =(Fpage f, 单词 AccessRights)Fpage |  | 没有– | [41](#page53) |
| + = (Fpage f, 词 AccessRights) Fpage |  | 没有– | [41](#page53) |
| != (MsgTag l, r) Bool |  | 没有– | [52](#page64) |
| + = (MsgTag t, 单词标签) MsgTag |  | 没有– | [52](#page64) |
| + = (StringItem& dest, StringItem AdditionalSubstring) StringItem & | | 没有– | [60](#page72) |
| + = (StringItem, 空 \* AdditionalSubstringAddress) StringItem | | 没有– | [60](#page72) |
| + = (StringItem s, CacheAllocationHint h) StringItem |  | 没有– | [61](#page73) |
| != (ThreadId l, r) Bool |  | 没有– | [15](#page27) |
| != (时间 l, r) Bool |  | 没有– | [31](#page43) |
| = (时间 l, r) 时间 |  | 没有– | [31](#page43) |
| + = (时间 l, r) 时间 |  | 没有– | [31](#page43) |
| = (时间 l, 字 r) 时间 |  | 没有– | [31](#page43) |
| + = (时间 l, 词 r) 时间 |  | 没有– | [31](#page43) |
| MyGlobalId () ThreadId |  | 没有– | [15](#page27) |

|  |  |  |
| --- | --- | --- |
| 触发、类型和常量的表 |  | 179 |
|  | 使用的系统调用 | 页 |
|  |  |  |
| MyGlobalId () ThreadId | 没有– | [17](#page29) |
| MyLocalId () ThreadId | 没有– | [15](#page27) |
| MyLocalId () ThreadId | 没有– | [17](#page29) |
| 自己 () ThreadId | 没有– | [15](#page27) |
| 自己 () ThreadId | 没有– | [17](#page29) |
| 从不时间常量 | – N/ | [30](#page42) |
| 下一页 (BootRec \*BootRec) BootRec \* | 没有– | [169](#page181) |
| Nilpage Fpage 常量 | – N/ | [41](#page53) |
| Niltag MsgTag 常量 | – N/ | [52](#page64) |
| nilthread ThreadId 常量 | – N/ | [15](#page27) |
| NoAccess 字const | – N/ | [41](#page53) |
| NumMemoryDescriptors (空 \* KernelInterface) 字 | 没有– | [8](#page20) |
| NumProcessors (空 \* KernelInterface) 字 | 没有– | [8](#page20) |
| PageRights (空 \* KernelInterface) 字 | 没有– | [8](#page20) |
| 调页程序 () ThreadId | 没有– | [17](#page29) |
| 调页程序 (ThreadId t) ThreadId | ExchangeREGISTERS | [22](#page34) |
| PageSizeMask (空 \* KernelInterface) 字 | 没有– | [8](#page20) |
| PreemptionPending () Bool | 没有– | [37](#page49) |
| ProcDesc 数据类型 | – N/ | [8](#page20) |
| ProcDesc (空 \* KernelInterface, 字 num) ProcDesc \* | 没有– | [9](#page21) |
| ProcessorControl (词 ProcessorNo,InternalFrequency, ExternalFrequency, | 没有– | [75](#page87) |
| 电压) 字 |  |  |
| ProcessorNo () 字 | 没有– | [17](#page29) |
| 把 (味精和味精, 单词 l, int u, 单词和 [u] ut, int t, fMapItem, GrantItem, StringItem | 没有– | [53](#page65) |
| g 和项目) 无效 |  |  |
| 放 (味精、味精、CtrlXferItem c) 无效 | 没有– | [54](#page66) |
| 放 (味精、味精、GrantItem、g) 无效 | 没有– | [54](#page66) |
| 放 (味精, 味精, MapItem 米) 无效 | 没有– | [53](#page65) |
| 放 (味精、味精、StringItem) 无效 | 没有– | [54](#page66) |
| 把 (味精、味精、StringItem、s) 作废 | 没有– | [54](#page66) |
| 放 (味精, 单词 u, 字 w) 无效 | 没有– | [53](#page65) |
| RcvWindow (受体 a) Fpage | 没有– | [63](#page75) |
| 可读字常量 | – N/ | [40](#page52) |
| ReadeXecOnly 字常量 | – N/ | [41](#page53) |
| ReadPrecision (空 \* KernelInterface)词 | 没有– | [9](#page21) |
| 接收 (ThreadId) MsgTag | Ipc | [69](#page81) |
| 接收 (ThreadId, 时间 RcvTimeout) MsgTag | Ipc | [69](#page81) |
| 回复 (ThreadId) MsgTag | Ipc | [69](#page81) |
| ReplyWait (ThreadId, ThreadId) MsgTag | Ipc | [69](#page81) |
| ReplyWait (ThreadId, 时间 RcvTimeout, ThreadId) MsgTag | Ipc | [69](#page81) |
| ReservedMemoryType 字常量 | – N/ | [9](#page21) |
| 权利 (Fpage f) 字 | 没有– | [41](#page53) |
| SameThreads (ThreadId l, r) Bool | ExchangeREGISTERS | [15](#page27) |
| SchedulePrecision (空 \* KernelInterface) 字 | 没有– | [9](#page21) |
| 计划 (ThreadIdTimeControl, ProcessorControl, 奥斯陆, 抢占 | S安排 | [35](#page47) |
| 控制, 词和老 TimeControl) 词 |  |  |
| 发送 (ThreadId to) MsgTag | Ipc | [69](#page81) |
| 发送 (ThreadId 到, 时间 SndTimeout) MsgTag | Ipc | [69](#page81) |
| 设置 CopFlag (字 n) 无效 | 没有– | [17](#page29) |
| 设置 CopFlag (字 n) 无效 | 没有– | [73](#page85) |
| 设置 ExceptionHandler (ThreadId NewHandler) 无效 | 没有– | [17](#page29) |
| 设置 ExceptionHandler (ThreadId 新) 无效 | 没有– | [72](#page84) |
| 设置标签 (msg 和味精, 字标签) 无效 | 没有– | [53](#page65) |
| 设置 MsgTag (msg 和味精, MsgTag t) 无效 | 没有– | [53](#page65) |
| 设置 MsgTag (MsgTag t)无效 | 没有– | [52](#page64) |
| 设置 PageAttribute (Fpage f, 单词属性) | M埃默里C控制 | [77](#page89) |
| 设置调页程序 (ThreadId NewPager) 无效 | 没有– | [17](#page29) |
| 设置调页程序 (ThreadId t、p) 无效 | ExchangeREGISTERS | [22](#page34) |
| 设置 PagesAttributes (n、Fpage &gt; [n] fpages、&gt; [4] 属性) | M埃默里C控制 | [77](#page89) |
| 设置 PreemptionDelay (ThreadId, 单词 sensitivePrio, 单词 maxDelay) 字 | 没有– | [35](#page47) |
| 设置优先级 (ThreadId 目标, 单词奥斯陆) | 没有– | [35](#page47) |
| 设置 ProcessorNo (ThreadId, 单词 ProcessorNo) | 没有– | [35](#page47) |
| 设置传播 (MsgTag 和 t) 无效 | 没有– | [70](#page82) |

|  |  |  |  |
| --- | --- | --- | --- |
| 180 | 触发、类型和常量的表 | | |
|  |  | 使用的系统调用 | 页 |
|  |  |  |  |
| 设置权限 (Fpage 和 f, 单词 AccessRights) 无效 |  | 没有– | [41](#page53) |
| 设置时间 (ThreadId)时间 ts, 时间 tq) 字 |  | 没有– | [35](#page47) |
| 设置 UserDefinedHandle (ThreadId t, 字柄) 无效 |  | ExchangeREGISTERS | [21](#page33) |
| 设置 UserDefinedHandle (单词 NewValue) 无效 |  | 没有– | [17](#page29) |
| 设置 VirtualSender (ThreadId t) 无效 |  | 没有– | [17](#page29) |
| 设置 VirtualSender (ThreadId t) 无效 |  | 没有– | [70](#page82) |
| 设置 XferTimeouts (单词 NewValue) 无效 |  | 没有– | [17](#page29) |
| SharedMemoryType 字const |  | – N/ | [9](#page21) |
| SimpleExec BssPstart (BootRec \* b) 字 |  | 没有– | [171](#page183) |
| SimpleExec BssSize (BootRec \* b) 字 |  | 没有– | [172](#page184) |
| SimpleExec BssVstart (BootRec \* b) 字 |  | 没有– | [171](#page183) |
| SimpleExec 命令 (BootRec \* b) 字符 \* |  | 没有– | [172](#page184) |
| SimpleExec DataPstart (BootRec \* b) 字 |  | 没有– | [171](#page183) |
| SimpleExec DataSize (BootRec \* b) 字 |  | 没有– | [171](#page183) |
| SimpleExec DataVstart (BootRec \* b) 字 |  | 没有– | [171](#page183) |
| SimpleExec 旗 (BootRec \* b) 字 |  | 没有– | [172](#page184) |
| SimpleExec InitialIP (BootRec \* b) 字 |  | 没有– | [172](#page184) |
| SimpleExec 标签 (BootRec \* b) 字 |  | 没有– | [172](#page184) |
| SimpleExec 设置标志 (BootRec \* b, 字 w) 无效 |  | 没有– | [172](#page184) |
| SimpleExec 设置标签 (BootRec \* b, 字 w) 无效 |  | 没有– | [172](#page184) |
| SimpleExec TextPstart (BootRec \* b) 字 |  | 没有– | [171](#page183) |
| SimpleExec TextSize (BootRec \* b) 字 |  | 没有– | [171](#page183) |
| SimpleExec TextVstart (BootRec \* b) 字 |  | 没有– | [171](#page183) |
| 大小 (Fpage f) 字 |  | 没有– | [41](#page53) |
| SizeLog2 (Fpage f) 字 |  | 没有– | [41](#page53) |
| 睡眠 (时间 t) 无效 |  | Ipc | [69](#page81) |
| SmallSpace (词位置, 大小) 词 |  | 没有– | [105](#page117) |
| SndBase (GrantItem g) 字 |  | 没有– | [57](#page69) |
| SndBase (MapItem m) 字 |  | 没有– | [56](#page68) |
| SndFpage (GrantItem g) Fpage |  | 没有– | [57](#page69) |
| SndFpage(MapItem m)Fpage |  | 没有– | [56](#page68) |
| SpaceControl (ThreadId SpaceSpecifier, 文字控制, Fpage KernelInter- | | S步伐C控制 | [47](#page59) |
| facePageArea、UtcbArea、ThreadId 重定向器、单词和旧控件) | |  |  |
| SpeculativeMemory 字常量 |  | – N/ | [144](#page156) |
| 开始 (ThreadId t) 无效 |  | ExchangeREGISTERS | [22](#page34) |
| 开始 (ThreadId t, 单词 sp, ip, 标志) 无效 |  | ExchangeREGISTERS | [22](#page34) |
| 开始(ThreadId t, 单词 sp, ip) 无效 |  | ExchangeREGISTERS | [22](#page34) |
| 停止 (ThreadId t) ThreadState |  | ExchangeREGISTERS | [22](#page34) |
| 停止 (ThreadId t, 单词和 sp, ip, 标志) ThreadState |  | ExchangeREGISTERS | [22](#page34) |
| StoreBR (int i, 词和 w) 无效 |  | 没有– | [11](#page23) |
| StoreBR (int i, 词和 w) 无效 |  | 没有– | [63](#page75) |
| StoreBRs (int i, k, 词和 [k]) 无效 |  | 没有– | [11](#page23) |
| StoreBRs (int i, k, 词和 [k]) 无效 |  | 没有– | [63](#page75) |
| StoreMR (int i, 词和 w) 无效 |  | 没有– | [11](#page23) |
| StoreMR (int i, 词和 w) 无效 |  | 没有– | [54](#page66) |
| StoreMRs (int i; k, 单词和 [k] w) 无效 |  | 没有– | [11](#page23) |
| StoreMRs (int i; k, 单词和 [k] w) 无效 |  | 没有– | [54](#page66) |
| 存储 (MsgTag t、味精和味精) 无效 |  | 没有– | [53](#page65) |
| StringItem 数据类型 |  | – N/ | [60](#page72) |
| StringItem (int 大小, void \* 地址) StringItem |  | 没有– | [60](#page72) |
| StringItems (受体 a) Bool |  | 没有– | [63](#page75) |
| StringItemsAcceptor 受体常量 |  | – N/ | [62](#page74) |
| + (StringItem s,CacheAllocationHint h) StringItem |  | 没有– | [61](#page73) |
| StringItem (StringItem s) Bool |  | 没有– | [60](#page72) |
| 子字符串 (StringItem s) 单词 |  | 没有– | [60](#page72) |
| 子字符串 (StringItem 和 s, 单词 n)无效\* |  | 没有– | [60](#page72) |
| SystemClock () 时钟 |  | S系统C锁 | [29](#page41) |
| ThreadControl (ThreadId, SpaceSpecifier, 调度程序, 调页程序, 空 \* UtcbLoca | | THREADC控制 | [25](#page37) |
| ) 字 |  |  |  |
| ThreadIdBits (空 \* KernelInterface) 字 |  | 没有– | [8](#page20) |
| ThreadId 数据类型 |  | – N/ | [15](#page27) |
| == (ThreadId l, r) Bool |  | 没有– | [15](#page27) |
| ThreadIdSystemBase (空 \* KernelInterface) 字 |  | 没有– | [8](#page20) |

|  |  |  |
| --- | --- | --- |
| 触发、类型和常量的表 |  | 181 |
|  | 使用的系统调用 | 页 |
|  |  |  |
| ThreadIdUserBase (空 \* KernelInterface) 字 | 没有– | [9](#page21) |
| ThreadNo(ThreadId t)词 | 没有– | [15](#page27) |
| ThreadState 数据类型 | – N/ | [22](#page34) |
| ThreadSwitch (ThreadId) 无效 | THREADS女巫 | [32](#page44) |
| ThreadWasHalted (ThreadState s) Bool | 没有– | [22](#page34) |
| ThreadWasIpcing (ThreadState s) Bool | 没有– | [22](#page34) |
| ThreadWasReceiving (ThreadState s) Bool | 没有– | [22](#page34) |
| ThreadWasSending (ThreadState s) Bool | 没有– | [22](#page34) |
| 时间数据类型 | – N/ | [30](#page42) |
| = (时间 l, r) Bool | 没有– | [31](#page43) |
| (时间 l, r) Bool | 没有– | [31](#page43) |
| == (时间 l, r) Bool | 没有– | [31](#page43) |
| >> = (时间 l, r) Bool | 没有– | [31](#page43) |
| >> (时间 l, r) Bool | 没有– | [31](#page43) |
| (时间 l, r)时间 | 没有– | [31](#page43) |
| + (时间 l, r) 时间 | 没有– | [31](#page43) |
| (时间 l, 词 r)时间 | 没有– | [31](#page43) |
| + (时间 l, 字 r) 时间 | 没有– | [31](#page43) |
| 超时 (时间 SndTimeout, RcvTimeout) 单词 | 没有– | [70](#page82) |
| TimePeriod(Word64 微秒)时间 | 没有– | [30](#page42) |
| TimePoint (时钟) 时间 | 没有– | [31](#page43) |
| 时间 (ThreadId, 时间和 ts, 时间和 tq) 字 | 没有– | [35](#page47) |
| 文字 (BootRec \* BootRec) | 没有– | [169](#page181) |
| TypedWords (味精标签 t) 字 | 没有– | [52](#page64) |
| 文字 (MemoryDesc 和 m) | 没有– | [9](#page21) |
| UncacheableMemory 字常量 | – N/ | [107](#page119) |
| UncacheableMemory 字常量 | – N/ | [132](#page144) |
| UndefinedMemoryType 字常量 | – N/ | [9](#page21) |
| 取消 (Fpage f) Fpage | Unmap | [43](#page55) |
| 取消 (字控) 无效 | Unmap | [43](#page55) |
| 取消 (单词 n, Fpage 和 [n] fpages) 无效 | Unmap | [43](#page55) |
| UntypedWordsAcceptor 受体常量 | – N/ | [62](#page74) |
| UntypedWords (味精标签 t) 字 | 没有– | [52](#page64) |
| 我们eDefaultCacheLineAllocation CacheAllocationHint 常量 | – N/ | [106](#page118) |
| UseDefaultCacheLineAllocation CacheAllocationHint 常量 | – N/ | [131](#page143) |
| UseDefaultCacheLineAllocation CacheAllocationHint 常量 | – N/ | [60](#page72) |
| UserDefinedHandle (ThreadId t) 字 | ExchangeREGISTERS | [21](#page33) |
| UserDefinedHandle () 字 | 没有– | [17](#page29) |
| UtcbAlignmentLog2 (空 \* KernelInterface) 字 | 没有– | [9](#page21) |
| UtcbAreaSizeLog2 (空 \* KernelInterface) 字 | 没有– | [9](#page21) |
| UtcbSize (空 \* KernelInterface) 字 | 没有– | [9](#page21) |
| 版本 (ThreadId t) 单词 | 没有– | [15](#page27) |
| 等待 (ThreadId) MsgTag | Ipc | [69](#page81) |
| 等待 (时间 RcvTimeout, ThreadId) MsgTag | Ipc | [69](#page81) |
| WaseXecuted (Fpage f) Bool | 没有– | [44](#page56) |
| WasReferenced (Fpage f) Bool | 没有– | [44](#page56) |
| WasWritten (Fpage f) Bool | 没有– | [44](#page56) |
| 可写字常量 | – N/ | [40](#page52) |
| WriteBackMemory 字常量 | – N/ | [107](#page119) |
| WriteBackMemory 字常量 | – N/ | [132](#page144) |
| WriteBackMemory 字常量 | – N/ | [144](#page156) |
| WriteCombiningMemory 字常量 | – N/ | [107](#page119) |
| WriteCombiningMemory 字常量 | – N/ | [132](#page144) |
| WriteProtectedMemory 字常量 | – N/ | [107](#page119) |
| WriteProtectedMemory 字常量 | – N/ | [132](#page144) |
| WriteThroughMemory 字常量 | – N/ | [107](#page119) |
| WriteThroughMemory 字常量 | – N/ | [132](#page144) |
| WriteThroughMemory 字常量 | – N/ | [144](#page156) |
| XferTimeouts () 字 | 没有– | [17](#page29) |
| 屈服 () 无效 | THREADS女巫 | [32](#page44) |
| ZeroTime 时间常数 | – N/ | [30](#page42) |

182 触发、类型和常量的表

**索引**

* =,[15,](#page27)[28,](#page40)[31](#page43)

+,[28,](#page40)[31,](#page43)[41,](#page53)[52,](#page64)[61,](#page73)[62](#page74)

* =,[31,](#page43)[41,](#page53)[52,](#page64)[60–62](#page74),[28,](#page40)[31,](#page43)[41,](#page53)[63](#page75)

-(忽略),[七](#page7)=,[31,](#page43)[41,](#page53)[63](#page75)<,[28,](#page40)[31](#page43)<=,[28,](#page40)[31](#page43)

(未更改),[七](#page7)==,[15,](#page27)[28,](#page40)[31,](#page43)[52,](#page64)[61](#page73)>,[28,](#page40)[31](#page43)>=,[28,](#page40)[31](#page43)

(未定义),[七](#page7)

0, 请参见 sigma0

AbortIpc 和停止,[22](#page34)

AbortReceive 和停止,[22](#page34)

AbortSend 和停止,[22](#page34)

接受,[63](#page75)

接受,[63](#page75)

受体,[62](#page74)

ActualSender,[17,](#page29)[70](#page82)

地址,[41](#page53)

地址空间

创建/删除、[45](#page57)

初始,[89](#page101)

AllocateNewCacheLines,[106,](#page118)[131](#page143)

AllocateOnlyNewL1CacheLines,[106,](#page118)[131](#page143)

anylocalthread,[15](#page27)

anythread,[15](#page27)

ApiFlags,[8](#page20)

ApiVersion,[8](#page20)

追加,[53,](#page65)[63,](#page75)[110](#page122)

ArchitectureSpecificMemoryType,[9](#page21)

AssociateInterrupt,[25](#page37)

BootInfo,[9](#page21)

BootInfo EFITables,[171](#page183)

BootInfo 条目,[169](#page181)

BootInfo FirstEntry,[169](#page181)

BootInfo 模块,[171](#page183)

BootInfo 引导,[171](#page183)

BootInfo SimpleExec,[171](#page183)

BootInfo 大小,[169](#page181)

BootInfo 有效,[169](#page181)

启动, 92–94

amd64,[135](#page147)

ia32,[112](#page124)

powerpc,[149](#page161)

ppc64,[166](#page178)

BootLoaderSpecificMemoryType,[9](#page21)

BR, 请参阅缓冲寄存器

缓冲寄存器,[62](#page74)

amd64,[123–124](#page136)

ia32,[97–98](#page110)

powerpc,[139](#page151)

ppc64,[157](#page169)

缓存,[59,](#page71)[106,](#page118)[107,](#page119)[131,](#page143)[132,](#page144)[144,](#page156)[163](#page175)CacheAllocationHint,[61](#page73)CachingEnabledMemory,[144](#page156)CachingInhibitedMemory,[144](#page156)

打电话,[68,](#page80)[69](#page81)

明确,[53,](#page65)[63](#page75)

时钟,[28](#page40)

阅读,[29](#page41)

Clr CopFlag,[17,](#page29)[73](#page85)

CompleteAddressSpace,[41](#page53)

CompoundString,[60](#page72)

方便的编程接口,[六](#page6)

ConventionalMemoryType,[9](#page21)

协,[73](#page85)

ctrlxfer

ia32,[109](#page121)

ia32-hvm,[114](#page126)

powerpc-hvm,[151](#page163)

议定书,[85](#page97)

CtrlXferItem,[58](#page70)

CtrlXferItemInit,[58](#page70)

CtrlXferItemsAcceptor,[62](#page74)

DeassociateInterrupt,[25](#page37)调试寄存器,[111,](#page123)[134](#page146)DedicatedMemoryType,[9](#page21)DefaultMemory,[77,](#page89)[107,](#page119)[132,](#page144)[144](#page156)DisablePreemption,[37](#page49)DisablePreemptionFaultException,[37](#page49)不AllocateNewCacheLines,[106,](#page118)[131](#page143)

电喷 MemdescSize,[172](#page184)

电喷 MemdescVersion,[172](#page184)

电喷 Memmap,[172](#page184)

电喷 MemmapSize,[172](#page184)

电喷 Systab,[172](#page184)

EnablePreemption,[37](#page49)

EnablePreemptionFaultException,[37](#page49)

字节,[3](#page15)

ErrInvalidParam,[36,](#page48)[77](#page89)

ErrInvalidScheduler,[25](#page37)

ErrInvalidSpace,[25,](#page37)[47](#page59)

ErrInvalidThread,[22,](#page34)[25,](#page37)[35](#page47)

ErrKipArea,[47](#page59)

ErrNoMem,[25](#page37)

ErrNoPrivilege,[25,](#page37)[35,](#page47)[47,](#page59)[75,](#page87)[77](#page89)

代码,[17,](#page29)[22,](#page34)[25,](#page37)[35,](#page47)[47,](#page59)[69,](#page81)[75,](#page87)[77](#page89)

ErrUtcbArea,[25,](#page37)[47](#page59)

异常

处理,[72](#page84)

消息

amd64,[133](#page145)

ia32,[108](#page120)

|  |  |  |
| --- | --- | --- |
| 184 | 索引 |  |
| powerpc,[146](#page158) | 数据结构, 2-6 |  |
| ppc64,[164](#page176) | 检索, 7-10 |  |
| 议定书,[84](#page96) | KernelGenDate,[8](#page20) |  |
| ExceptionHandler,[17,](#page29)[72](#page84) | KernelId,[8](#page20) |  |
| ExchangeRegisters,[21](#page33) | KernelInterface,[8](#page20) |  |
| 可执行,[41](#page53) | KernelSupplier,[8](#page20) |  |
| ExternalFreq,[10](#page22) | KernelVersion,[8](#page20) |  |
| 功能,[9](#page21) | KernelVersionString,[9](#page21) |  |
| KipAreaSizeLog2,[9](#page21) |  |
| 冲,[44](#page56) | 标签,[52,](#page64)[53](#page65) |  |
| Fpage,[41](#page53) |  |
| fpage, 40–41 | LargeSpace,[105](#page117) |  |
| 映射,[64](#page76) | Lcall,[69](#page81) |  |
| 接收,[62](#page74) | Lipc,[68](#page80) |  |
| 取消,[40,](#page52)42–44 | lipc,[64](#page76) |  |
| FpageLog2,[41](#page53) | 负载,[53](#page65) |  |
| FullyAccessible,[41](#page53) | LoadBR,[11,](#page23)[63](#page75) |  |
| 通用二进制接口,[六](#page6) | LoadBRs,[11,](#page23)[63](#page75) |  |
| LoadMR,[11,](#page23)[54](#page66) |  |
| 通用 bootinfo, 167–172 | LoadMRs,[11,](#page23)[54](#page66) |  |
| 数据结构, 167–168 | 本地 ipc,[64](#page76) |  |
| 通用记录, 168–169 | 本地线程 ID,[14](#page26) |  |
| 通用编程接口,[六](#page6) | LocalId,[15,](#page27)[21](#page33) |  |
| 获取[53,](#page65)[54](#page66) | LocalMemory,[144](#page156) |  |
| GetStatus,[44](#page56) | 逻辑接口,[六](#page6) |  |
| 全局线程 ID,[14](#page26) | 低,[9](#page21) |  |
| GlobalId,[15,](#page27)[21](#page33) | LreplyWait,[69](#page81) |  |
| GlobalMemory,[144](#page156) | MapGrantItems,[62,](#page74)[63](#page75) |  |
| GrantItem,[57](#page69) |  |
| GuardedMemory,[144](#page156) | MapItem,[55,](#page67)[56](#page68) |  |
| 高,[9](#page21) | MBI Address, [172](#page184) |  |
| 内存描述符,[6,](#page18)93–94 |  |
| 包括文件、[八](#page8) | MemoryControl,[77](#page89) |  |
| MemoryDesc,[9](#page21) |  |
| IntendedReceiver,[17,](#page29)[69](#page81) | 消息寄存器, 50–51 |  |
| InternalFreq,[10](#page22) | amd64, 122–123 |  |
| 中断 | ia32,[97](#page109) |  |
| 协会,[23](#page35) | powerpc, 138–139 |  |
| 线程 ID,[14](#page26) | ppc64, 156–157 |  |
| 我 fpage,[103,](#page115)[130](#page142) | 消息 |  |
| IoFpage,[103,](#page115)[130](#page142) | 生成、50–54 |  |
| IoFpageLog2,[103,](#page115)[130](#page142) | 特定型号的寄存器,[111,](#page123)[134](#page146) |  |
| IoFpagePort,[103](#page115) | 模块命令,[171](#page183) |  |
| IoFpageSize,[103](#page115) | 模块大小,[171](#page183) |  |
| IoFpageSizeLog2,[103](#page115) | 模块启动,[171](#page183) |  |
| IPC, 64–70 | 先生, 请参阅消息寄存器 |  |
| 中止,[19](#page31) | MsgTag,[52,](#page64)[53](#page65) |  |
| 交叉 cpu,[67](#page79) | MyGlobalId,[15,](#page27)[17](#page29) |  |
| 传播,[65](#page77) | MyLocalId,[15,](#page27)[17](#page29) |  |
| ipc,[68](#page80) | 自己,[15,](#page27)[17](#page29) |  |
| IpcFailed,[69](#page81) |  |  |
| IpcPropagated,[69](#page81) | 从来,[30](#page42) |  |
| IpcRedirected,[69](#page81) | 下,[169](#page181) |  |
| IpcSucceeded,[69](#page81) | Nilpage,[41](#page53) |  |
| IpcXcpu,[69](#page81) | Niltag,[52](#page64) |  |
| IsGlobalId,[15](#page27) | nilthread,[15](#page27) |  |
| IsIoFpage,[103](#page115) | NoAccess,[41](#page53) |  |
| IsLocalId,[15](#page27) | NumMemoryDescriptors,[8](#page20) |  |
| IsNilFpage,[41](#page53) | NumProcessors,[8](#page20) |  |
| IsNilThread,[15](#page27) | 页 |  |
| IsVirtual,[9](#page21) |  |
|  | 访问权限、[4,](#page16)[40,](#page52)[55,](#page67)[57,](#page69)[82,](#page94)[90](#page102) |  |
| 内核功能,[5](#page17) | 改变,[42,](#page54)[55,](#page67)[57](#page69) |  |
| ia32,[102](#page114) | 检查,[43](#page55) |  |
| 内核界面页 | 属性,[90](#page102) |  |
| 位置,[45](#page57) | amd64,[132](#page144) |  |
| 内核界面页,10 | ia32,[107](#page119) |  |

索引

powerpc,[144](#page156)

ppc64,[163](#page175)

大小,[3](#page15)

pagefault

议定书,[82](#page94)

调页程序,[17,](#page29)[22](#page34)

调页程序,[82](#page94)

改变,[17,](#page29)[22,](#page34)[24](#page36)

PageRights,[8](#page20)

PageSizeMask,[8](#page20)

抢占,[34,](#page46)[37](#page49)

议定书,[83](#page95)

PreemptionPending,[37](#page49)

特权线程,[七](#page7)

ProcDesc,[9](#page21)

特定于处理器的二进制接口[七](#page7)

ProcessorControl,[75](#page87)

ProcessorNo,[16](#page28)

ProcessorNo,[17](#page29)

传播,[65](#page77)

把,[53,](#page65)[54](#page66)

RcvWindow,[63](#page75)

RDMSR,[111,](#page123)[134](#page146)

读,[40](#page52)

ReadeXecOnly,[41](#page53)

ReadPrecision,[9](#page21)

收到,[69](#page81)

定向,[46,](#page58)[65](#page77)

答复,[69](#page81)

ReplyWait,[69](#page81)

ReservedMemoryType,[9](#page21)

人权,[41](#page53)

SameThreads,[15](#page27)

Schedu的,[35](#page47)

SchedulePrecision,[9](#page21)

段,[111,](#page123)[134](#page146)

发送[69](#page81)

发送基地,[55](#page67)

敏感的奥斯陆,[34](#page46)

设置 CopFlag,[17,](#page29)[73](#page85)

设置 ExceptionHandler,[17,](#page29)[72](#page84)

设置标签,[53](#page65)

设置 MsgTag,[52,](#page64)[53](#page65)

设置 PageAttribute,[77](#page89)

设置调页程序,[17,](#page29)[22](#page34)

设置 PagesAttributes,[77](#page89)

设置 PreemptionDelay,[35](#page47)

设置优先级,[35](#page47)

设置 ProcessorNo,[35](#page47)

设置传播,[70](#page82)

设置权限,[41](#page53)

设置时间,[35](#page47)

设置 UserDefinedHandle,[17,](#page29)[21](#page33)

设置 VirtualSender,[17,](#page29)[70](#page82)

设置 XferTimeouts,[17](#page29)

SharedMemoryType,[9](#page21)

sigma0,[89](#page101)

协议, 89–91

SimpleExec BssPstart,[171](#page183)

SimpleExec BssSize,[172](#page184)

SimpleExec BssVstart,[171](#page183)

SimpleExec 命令,[172](#page184)

SimpleExec DataPstart,[171](#page183)

SimpleExec DataSize,[171](#page183)

SimpleExec DataVstart,[171](#page183)

185

SimpleExec 标志,[172](#page184)

SimpleExec InitialIP,[172](#page184)

SimpleExec 标签,[172](#page184)

SimpleExec 设置标志,[172](#page184)

SimpleExec 设置标签,[172](#page184)

SimpleExec TextPstart,[171](#page183)

SimpleExec TextSize,[171](#page183)

SimpleExec TextVstart, [171](#page183)

大小,[41](#page53)

SizeLog2,[41](#page53)

睡眠,[69](#page81)

小空间,[104](#page116)

SmallSpace,[105](#page117)

SndBase,[56,](#page68)[57](#page69)

SndFpage,[56,](#page68)[57](#page69)

SpaceControl,[47](#page59)

SpeculativeMemory,[144](#page156)

开始,[22](#page34)

停止,[22](#page34)

存储,[53](#page65)

StoreBR,[11,](#page23)[63](#page75)

StoreBRs,[11,](#page23)[63](#page75)

StoreMR,[11,](#page23)[54](#page66)

StoreMRs,[11,](#page23)[54](#page66)

StringItem,[60](#page72)

StringItems,[63](#page75)

StringItemsAcceptor,[62](#page74)

字符串, 59–61

接收,[62](#page74)

子,[60](#page72)

子,[60](#page72)

系统线程、[14](#page26)

系统线程、[69](#page81)

系统调用链接,[5](#page17)

amd64,[125](#page137)

ia32,[99](#page111)

powerpc, 140–143

ppc64,[158](#page170)

SystemBase,[4](#page16)

SystemClock,[29](#page41)

TCR, 请参见线程控制寄存器

线程

创建,[23](#page35)

停止,[19](#page31)

id,[14](#page26)

id, 15, 请参阅线程 id

migration, [34](#page46)

优先,[33](#page45)

特权,[七](#page7)

启动协议,[80](#page92)

国家,[22,](#page34)[34](#page46)

版本[14,](#page26)[23](#page35)

线程控制寄存器, 16–17

amd64,[122](#page134)

ia32,[96](#page108)

powerpc,[138](#page150)

ppc64,[156](#page168)

线程 ID,[14–15](#page27)

检索,[17,](#page29)[21](#page33)

ThreadControl,[25](#page37)

ThreadIdBits,[8](#page20)

ThreadIdSystemBase,[8](#page20)

ThreadIdUserBase,[9](#page21)

ThreadNo,[15](#page27)

ThreadSwitch,[32](#page44)

186 索引

ThreadWasHalted,[22](#page34)

届readWasIpcing,[22](#page34)

ThreadWasReceiving,[22](#page34)

ThreadWasSending,[22](#page34)

时间,[30–31](#page43)

时间量子,[33](#page45)

超时,[70](#page82)

TimePeriod,[30](#page42)

TimePoint,[31](#page43)

时间,[35](#page47)

时间,[33](#page45)

捐赠,[32](#page44)

类型,[9,](#page21)[169](#page181)

TypedWords,[52](#page64)

UncacheableMemory,[107,](#page119)[132](#page144)

UndefinedMemoryType,[9](#page21)

取消,[43](#page55)

UntypedWords,[52](#page64)

UntypedWordsAcceptor,[62](#page74)

向上兼容,[七](#page7)

UseDefaultCacheLineAllocation,[60,](#page72)[106,](#page118)[131](#page143)

用户,[4](#page16)

UserDefinedHandle,[16,](#page28)[20](#page32)

UserDefinedHandle,[17,](#page29)[21](#page33)

使用 API,[八](#page8)

Utcb

位置,[45](#page57)

大小,[4,](#page16)[24,](#page36)[45](#page57)

UtcbAlignmentLog2,[9](#page21)

UtcbAreaSizeLog2,[9](#page21)

UtcbSize,[9](#page21)

版本[15](#page27)

虚拟寄存器,[11](#page23)

等,[69](#page81)

是执行,[44](#page56)

WasReferenced,[44](#page56)

WasWritten,[44](#page56)

词,[七](#page7)

Word16,[七](#page7)

Word32,[七](#page7)

Word64,[七](#page7)

写,[40](#page52)

WriteBackMemory,[107,](#page119)[132,](#page144)[144](#page156)

WriteCombiningMemory,[107,](#page119)[132](#page144)

WriteProtectedMemory,[107,](#page119)[132](#page144)

WriteThroughMemory,[107,](#page119)[132,](#page144)[144](#page156)

WRMSR,[111,](#page123)[134](#page146)

XferTimeouts,[17](#page29)

产量,[32](#page44)

ZeroTime,[30](#page42)