操作系统内核设计
前序知识

ZJUNIX

目录

调用规范

- 调用规范的概念
- MIPS32架构通用寄存器
- MIPS32架构调用规范

调用规范的概念

- 调用规范规定了子程序如何从调用者接收参数、存放本地变量以及返回结果
- 参数、本地变量及返回结果可存放在寄存器、堆栈或其他存储器中
- MIPS32架构的通用寄存器较多，因此该架构上的调用规范对寄存器的使用作出了详细的规定
  - 这只是软件层面上的规范，而MIPS32架构上几乎所有的寄存器都是相同的
  - 其他架构的调用规范的寄存器使用可能与硬件架构相关

MIPS32架构通用寄存器

- MIPS32架构共有32个通用寄存器
- 其中2个寄存器有指定的用途：
  - 0号寄存器($zero)的值永远是0，向其中写入的数据会被丢弃
  - 31号寄存器($ra)在执行jal等指令进行函数调用时用于保存返回地址；该寄存器也可以像其他通用寄存器一样使用
- 除此之外，MIPS32架构上其他通用寄存器都是等同的

MIPS32架构调用规范

- 最常见的是O32调用规范
- 用gcc编译产生的代码使用O32调用规范
- 另外还有N32调用规范，与O32相差不大，这里不作介绍

Registers for O32 Calling Convention

Name	Number	Use	Callee must preserve?
$zero	$0	constant 0	N/A
$at	$1	assembler temporary	No
$v0–$v1	$2–$3	values for function returns and expression evaluation	No
$a0–$a3	$4–$7	function arguments	No
$t0–$t7	$8–$15	temporaries	No
$s0–$s7	$16–$23	saved temporaries	Yes
$t8–$t9	$24–$25	temporaries	No
$k0–$k1	$26–$27	reserved for OS kernel	N/A
$gp	$28	global pointer	Yes (except PIC code)
$sp	$29	stack pointer	Yes
$fp	$30	frame pointer	Yes
$ra	$31	return address	N/A

MIPS32架构调用规范

- 前4个参数通过$a0~$a3传递
- 如果有更多的参数，则通过堆栈传递
- 被调用者需要保护$s0~$s7，以及$gp, $sp, $fp寄存器，在函数返回时须保证这些寄存器的值与进入函数时相同
- 返回值存放在$v0寄存器中
- 如有需要，可以用$v1再传递一个返回值

Registers for O32 Calling Convention

Name	Number	Use	Callee must preserve?
$zero	$0	constant 0	N/A
$at	$1	assembler temporary	No
$v0–$v1	$2–$3	values for function returns and expression evaluation	No
$a0–$a3	$4–$7	function arguments	No
$t0–$t7	$8–$15	temporaries	No
$s0–$s7	$16–$23	saved temporaries	Yes
$t8–$t9	$24–$25	temporaries	No
$k0–$k1	$26–$27	reserved for OS kernel	N/A
$gp	$28	global pointer	Yes (except PIC code)
$sp	$29	stack pointer	Yes
$fp	$30	frame pointer	Yes
$ra	$31	return address	N/A

MIPS32架构调用规范

名称	序号	说明
$zero	0	常数0
$at	1	汇编器临时寄存器，用于保存某些伪指令展开后的中间数据
$v0-$v1	2-3	函数返回值
$a0-$a3	4-7	函数参数
$t0-$t7	8-15	临时寄存器
$s0-$s7	16-23	被调用者需保护的临时寄存器
$t8-$t9	24-25	临时寄存器
$k0-$k1	26-27	为操作系统内核保留的寄存器（应用程序不应该使用）
$gp	28	全局指针，一般指向数据段中间，作为快速访问数据段的指针
$sp	29	堆栈指针
$fp	30	帧指针
$ra	31	返回地址

特权资源

- 特权资源的作用
- MIPS32内核态与用户态
- MIPS32虚拟内存
- MIPS32异常机制

特权资源的作用

- 操作系统运行至少需要用户态和内核态两种特权级别
  - 用户态不能访问内核态的资源，反之则可以
- 各应用程序运行的虚拟地址空间需要相互隔离
- 用户态程序与操作系统内核需要通过异常机制进行沟通

MIPS32内核态与用户态

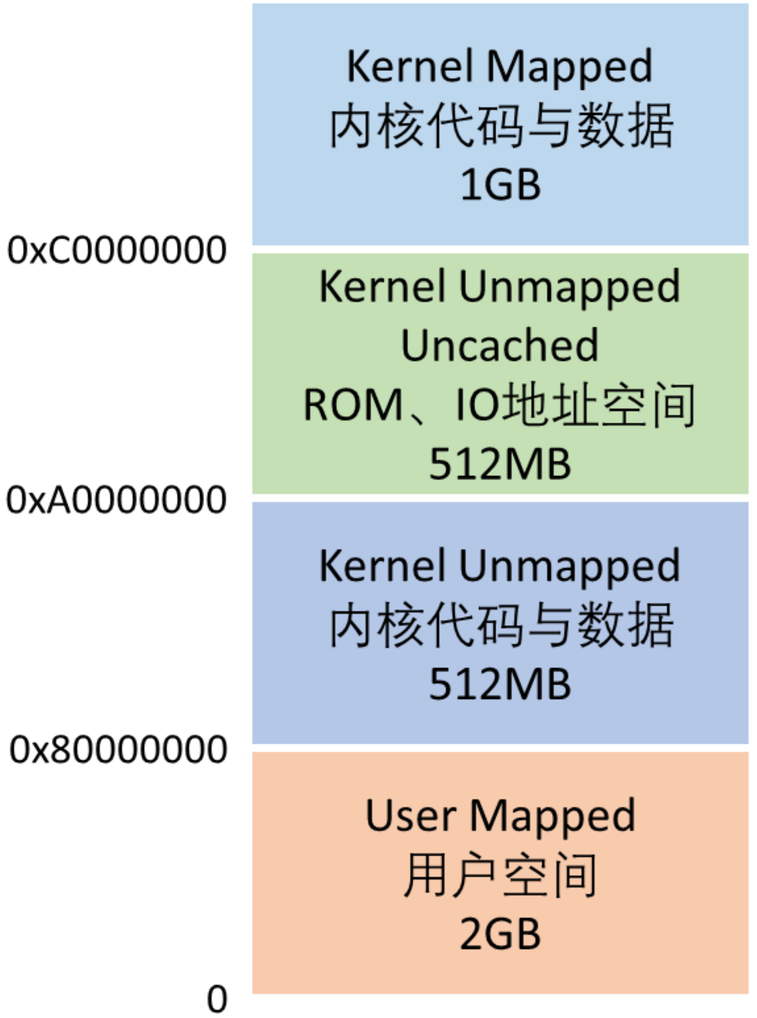
用户态：

- 可访问虚拟地址空间的低2GB空间
- 不能使用COP0(协处理器0)特权指令
  - 尝试执行这些指令会产生异常，并转到操作系统内核

内核态：

- 可访问完整的4GB虚拟地址空间
- 可以使用全部COP0(协处理器0)特权指令

MIPS32虚拟内存

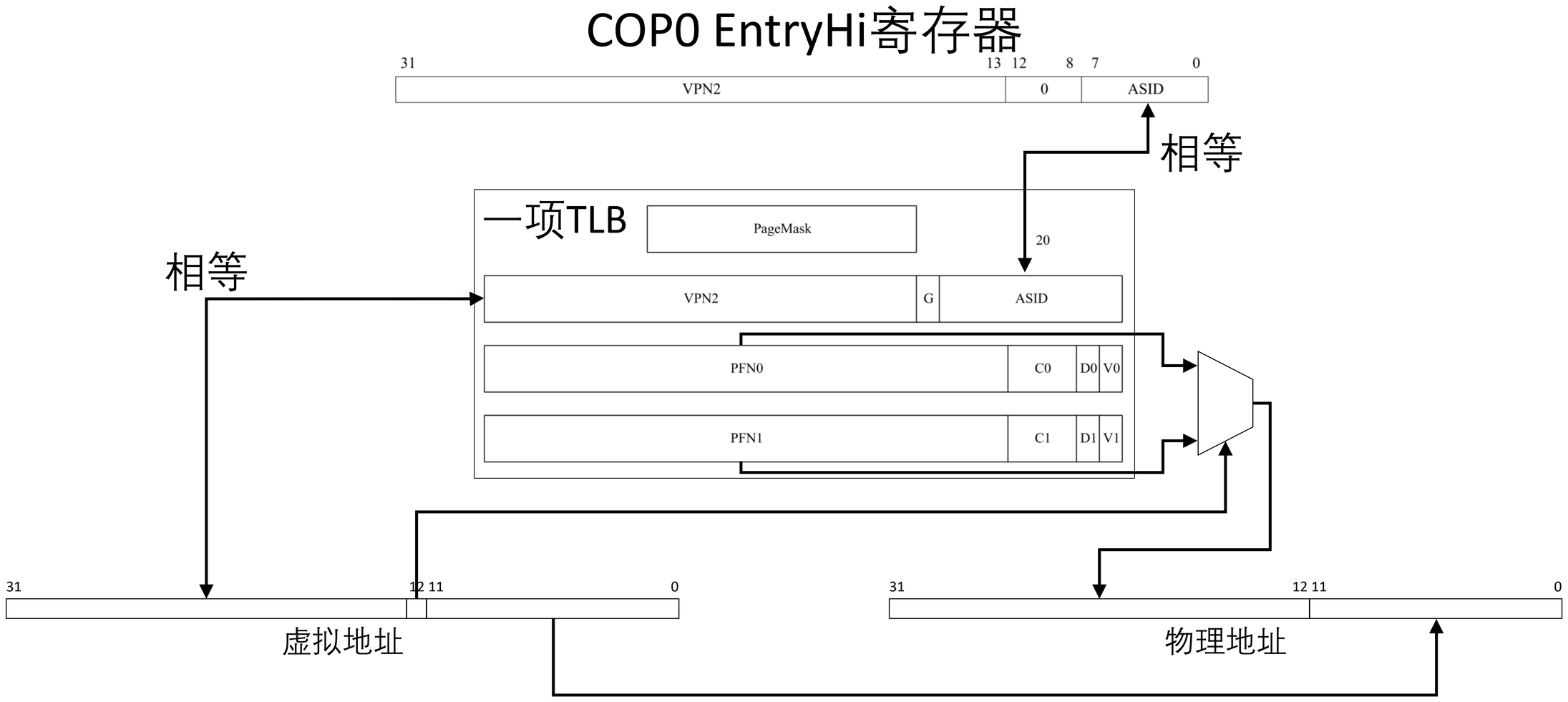

Kernel Mapped
内核代码与数据
1GB
0xC0000000
Kernel Unmapped
Uncached
ROM、IO地址空间
512MB
0xA0000000
Kernel Unmapped
内核代码与数据
512MB
0x80000000
User Mapped
用户空间
2GB
0

MIPS32虚拟内存

- MIPS32架构使用TLB进行虚拟地址翻译
- 页大小可变，最小4KB，最大256MB
- 一个TLB项映射两个物理页面
- TLB与COP0寄存器的ASID域需相等，操作系统内核进行进程调度时修改ASID，即可实现不同应用程序的地址空间隔离
- 以页大小为4KB为例，地址翻译流程如下：

ZJUNIX

MIPS32虚拟内存



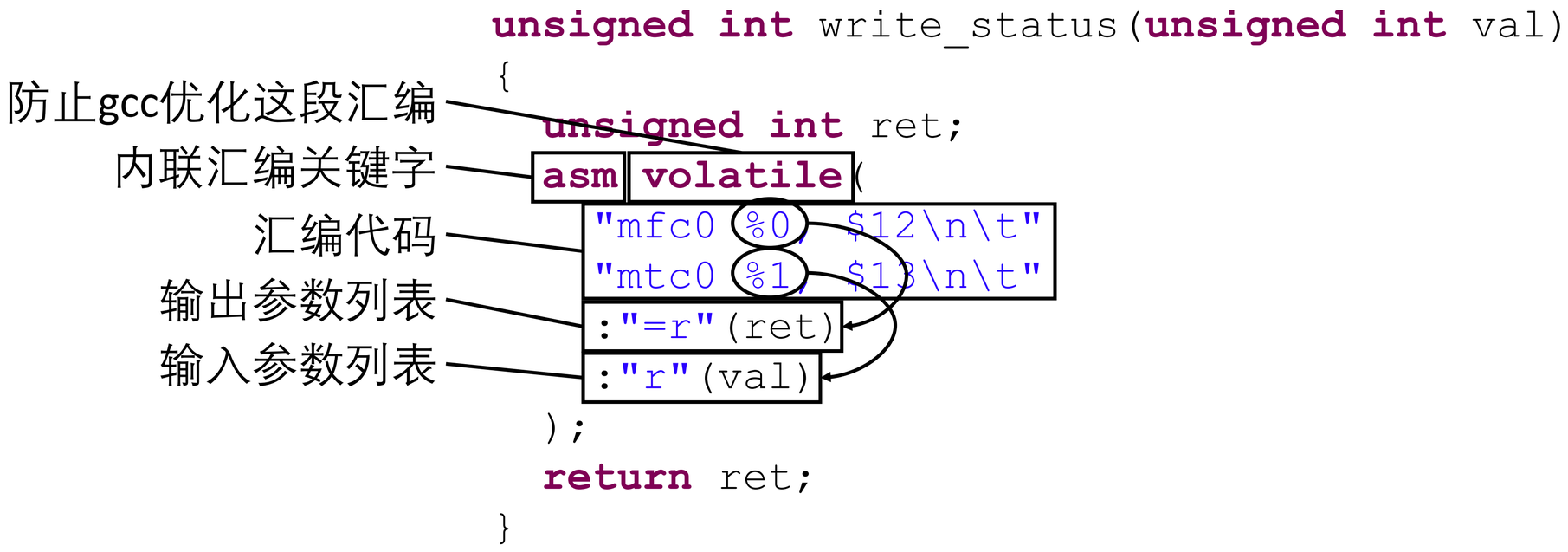
gcc内联汇编

- 内联汇编格式
- MIPS32汇编简介

内联汇编格式

- 内联汇编并非C语言标准所规定的语法，与各编译器实现相关
- 由于我们的操作系统内核使用gcc编译，因此这里介绍gcc内联汇编语法

MPIS内联（内嵌）汇编格式



MIPS32汇编简介

- MIPS32汇编的3种基本格式：

1. R型：寄存器-寄存器型

   add $t0, $s0, $s1：将$s0和$s1的值相加，结果存在$t0中

2. I型：寄存器-立即数型

   addi $t0, $s0, -32：将$s0的值减去32，结果存在$t0中

3. J型：跳转指令

   j func_1：跳转到func_1标签的地方执行

MIPS32汇编简介

- 我们的SoC平台所支持的指令：

指令类型(共87)	指令列表
算术运算(22)	add, addi, addiu, addu, sub, subu, slt, slti, sltiu, sltu, mul, mult, multu, madd, maddu, msub, msubu, div, divu, clo, clz, lui
逻辑运算(13)	and, andi, or, ori, xor, xori, nor, sll, sllv, srl, srlv, sra, srav
内存访问(12)	lb, lbu, lh, lhu, lwl, lwr, lw, sb, sh, swl, swr, sw
跳转(12)	beq, bne, blez, bgez, bltz, bgtz, bltzal, bgezal, j, jal, jr, jalr
移动(6)	movz, movn, mfhi, mthi, mflo, mtlo
陷入(12)	tge, tgei, tgeiu, tgeu, tlt, tlti, tltiu, tltu, teq, teqi, tne, tnei
COP0(7)	mfc0, mtc0, tlbr, tlbwi, tlbwr, tlbp, eret
其他(3)	syscall, break, cache

Makefile

- make程序
- Makefile规则

make程序

- make程序：是一个命令工具，是一个解释makefile中指令的命令工具
- make程序提供一种可以用于构件大规模工程的、强劲的而灵活的机制。

句法：	**make [选项] [目标] [宏定义]**
用途：	make工具根据名为makefile或Makefile的文件中指定的依赖关系对系统进行更新。[选项][目标][宏定义]可以以任意顺序指定。
常用选项/特性：	-d 显示调试信息 -f 文件 此选项告诉make使用指定文件作为依赖关系文件，而不是默认的makefile或Makefile，如果指定的文件名是"-"，那么make将从标准输入读入依赖关系。 -h 显示所有选项的帮助信息 -n 测试模式，并不真的执行任何命令，只是显示输出这些命令 -s 安静模式--不输出任何提示信息。

make程序

- make工具依赖一个特殊的，名为makefile的文件，这个文件描述了系统中各个模块之间的依赖关系。
- GNU make的主要功能是读进一个文本文件makefile并根据makefile的内容执行一系列的工作。
- makefile的默认文件名为GNUmakefile、makefile或Makefile，当然也可以在make的命令行中指定别的文件名。

Makefile

- Makefile是一个文本形式的数据库文件，其中包含一些规则来告诉make处理哪些文件以及如何处理这些文件。这些规则主要是描述哪些文件（称为target目标文件，不要和编译时产生的目标文件相混淆）是从哪些别的文件（称为dependency依赖文件）中产生的，以及用什么命令（command）来执行这个过程。
- make会对磁盘上的文件进行检查，如果目标文件的生成或被改动时的时间（称为该文件时间戳）至少比它的一个依赖文件还旧的话，make就执行相应的命令，以更新目标文件。目标文件不一定是最后的可执行文件，可以是任何一个中间文件并可以作为其他目标文件的依赖文件。

make程序

- Makefile规则的语法格式：

  目标文件列表：依赖文件列表

  <tab >命令列表

- 一个Makefile文件主要含有一系列的规则，每条规则包含以下内容。
  - “目标文件列表”，即make最终需要创建的文件，如可执行文件和目标文件；目标也可以是要执行的动作，如“clean”。
  - “依赖文件列表”，通常是编译目标文件所需要的其他文件。
  - “命令列表”，是make执行的动作，通常是把指定的相关文件编译成目标文件的编译命令，每个命令占一行，且每个命令行的起始字符必须为<Tab>字符。
- 除非特别指定，否则make的工作目录就是当前目录。“目标文件列表”是需要创建的二进制文件或目标文件，依赖文件列表是在创建“目标文件列表”时需要用到的一个或多个文件的列表，命令序列是创建“目标文件列表”文件所需要执行的步骤，比如编译命令。

THANK YOU