

摘 要

本次操作系统的设计定制运行在基于NEXYS4开发板上开发的MIPS处理器及片上系统上。其中,内核采用了ucore操作系统作为设计的基础,ucore操作系统的原始设计面向的体系结构是MIPS32S,通过QEMU模拟器模拟。虽然MIPS32S体系结构与基于NEXYS4开发板的MIPS处理器及片上系统有很多相似之处,然而在实现细节上,两者存在很多的不同:MIPS32S有着完整的指令集,而片上系统的指令集不完全,甚至没有乘除法指令;MIPS32S的转译后备缓冲器(TLB)设计更加完整,而片上系统在这个环节上的设计较为简单;而且,QEMU模拟器模拟了标准的外部设备,地址规整,然而片上系统的外设具有自己的地址和特性。

这些体系结构上的差异都对操作系统的设计带来了直接的影响,从而提出了巨大的挑战:如何将ucore操作系统移植到片上系统,同时,有效利用模拟器的开发环境,使得在该模拟器上开发和调试的操作系统即能够在模拟的MIPS32S环境中运行,又能够通过调整编译参数的方法,不经源代码级的改动就能编译出直接运行在NEXYS4开发板上。

为了应对这些挑战,现提出了如下的方法:采用加减法指令和移位指令来替代乘除法指令;采用已有的硬件结构和相关指令代替原有的硬件结构实现缺页异常的处理;根据片上系统的外设的特性和地址设计和实现了定制的驱动。同时,为了解决调试上的困难,通过编译参数动态地设置宏的定义,实现不同情况下对不同代码块的编译。

为了验证操作系统设计的正确性,除了基本的shell和命令(如ls, cat等)外,本文还在原有的操作系统基础上开发了一个贪吃蛇的屏幕游戏,以及键盘的驱动,最终能够在所面对的片上系统上接上显示器和键盘,正确运行所有的命令和屏幕游戏。

关键词: MIPS 架构; 操作系统设计; NEXYS4 开发板; 驱动设计

Abstract

This operating system is reserved for the MIPS processor and System On Chip (SOC) which is built on the NEXYS4 board. We port ucore to this new architecture. However, ucore's originally oriented architecture is MIPS32S, which can only run in QEMU simulator. In spite of the fact that MIPS32S shares some features with the MIPS processor built on the NEXYS4 board. Nevertheless, there are also many differences: MIPS32S has all the instructions while SOC are short of many standard instructions, such as instructions used for multiplication and division; MIPS32S hold the prefect structure of translation lookaside buffer (TLB) while SOC has removed some unnecessary modules. What's more, QEMU simulator has standard devices while our devices have their own addresses and characteristics.

All the differences on architecture seriously affect the design of our operating system. The major challenges are: How to port ucore to the SOC. simultaneously, to make full use of the developing environment of simulator, the kernel should run and be debugged in the simulator and we need to set some parameters to compile different codes to acquire the kernel run on the NEXYS4 board instead of doing changes in the source codes.

In order to solve these challenges, we now puts forward many solutions: replace the instructions used for multiplication and division with the combinations of usable instructions; adapt the process of handling page fault to the new architecture instand of the formal one; design drivers for the devices with their addresses and features. To get over the difficulties in debug and tests. We also set the corresponding definitions by adjusting parameters to compile codes on different platforms.

To show that our system design works well, regardless of the basic programs and commands (e.g. ls, cat), we also design a VGA game named Retro Snaker on our operating system, including a keyboard device. Finally, we are able to connect the VGA and keyboard to our SOC, and execute all commands and games.

Key Words: MIPS; operating system design; NEXYS4 board; driver design

目 录

摘要.....	I
Abstract	II
1 绪论	1
1.1 选题背景和意义	1
1.2 国内外研究现状及发展趋势	1
1.3 研究内容	6
1.4 论文组织和结构	7
2 系统设计需求分析	8
2.1 片上系统和 QEMU 模拟器间的差异	8
2.2 基于片上系统的操作系统设计方案	9
2.3 本章总结	10
3 乘除法指令的替代与实现	11
3.1 乘法指令的替代与实现	11
2.2 除法指令的替代与实现	13
3.3 本章总结	15
4 缺页异常的处理	16
4.1 基于 NEXYS 硬件的缺页填充机制	16
4.2 中断和异常处理机制	22
4.3 片上系统的缺页异常处理	25
4.4 本章总结	26
5 片上系统的设备与驱动实现	27
5.1 UART16450 驱动的设计与实现驱动设计	29
5.2 PS/2 键盘驱动的设计与实现	32
5.3 VGA 驱动设计的设计与实现	33
5.4 本章总结	34
6 系统评测	35
6.1 系统平台搭建与测试环境配置	35
6.2 QEMU 模拟器中系统测试	37

6.3	NEXYS4 开发板的系统测试	38
6.4	上层应用测试	39
6.5	本章总结	41
7	总结与展望	42
7.1	总结与分析	42
7.2	对未来的展望	43
	致谢	44
	参考文献	45

1 绪论

1.1 选题背景和意义

本课题来源于综合能力培养课程建设,其最终目的是在一个 NEXYS4 开发板开发的 MIPS 处理器及 SOC (System On Chip, 片上系统) 上定制设计运行一个操作系统,而且这个操作系统具有自己的内存管理、进程调度以及文件系统,具备完善的中断异常处理机制,可正常执行一系列可执行文件和 Unix 常见命令。

在之前的系统课程设计中,大多数操作系统设计都是借由虚拟机的帮助来完成,其中包括 Bochs、QEMU 等等。但是历史的经验证明这大大减少了学生对真正的硬件底层、BSP (Board Support Package, 板级支持包) 以及驱动实现的认知。而完成这一套课程设计有利于帮助大学生更加深入地了解底层硬件与上层软件的工作方式,以及软硬件的协同设计思想。与此同时,这个操作系统又是比较轻量级的,比起冗杂的 Linux,这个系统更方便学生深入了解一个内核的组成,可谓麻雀虽小,五脏俱全。

1.2 国内外研究现状及发展趋势

就目前情况而言,国内外很少有学校具有一套从底层硬件开始,一直到上层应用这么完整的系统级实验案例。大多数都是和之前的系统课程相似,借用虚拟机的便利性来完成对简单内核的设计和调试,如 MIT 的 JOS,早期的 JOS 构建于 Bochs,后期随着 QEMU 的不断发展,最新版的 JOS 已经可以运行在 QEMU 这个平台上了。相类似的案例还有 xv6、哈佛的 OS161 教学操作系统。就 Linux 内核来说,其中包含太多东西,如繁杂的驱动文件,多种文件系统的支持,多种体系结构的支持,以及各种优化处理过程。学生将深入了解一个操作系统在底层硬件和上层应用间扮演的角色和一些核心机制的实现,然后设计一个可以在真正的硬件平台上运行的操作系统。

国内在这方面发展近期比较迅速,比较有名气的就是清华大学的 ucore 操作系统,基于国外的现状,整理出了这一套系统。ucore 现在也可以支持多种体系结构,如 x86、AMD64、ARM 等等。ucore 之前的版本都是运行在 QEMU 这个虚拟机当中。随着大学对操作系统、体系结构、编译原理、接口技术等专业课程的深入学

习,从而形成一套完善的体制,在 FPGA 上完成一个简单的片上系统的设计。文献[1]、[2]、[3]、[4]主要提出了如何在 FPGA 设计 MIPS 芯片以及片上系统。文献[5]描述了 MIPS 片上系统的一些软应用。文献[6]是 MIPS 官方给出的基于 MIPS 架构的片上系统使用说明。

本次设计选定 NEXYS4 开发板为设计基础,并使用了当前较为流行的 RISC (Reduced Instruction Set Computer, 精简指令集)之一的 MIPS (Microprocessor without Interlocked Piped Stages, 无内部互锁流水级的微处理器)指令集作为体系结构基础。不同于 x86 这种 CISC (Complex Instruction Set Computer, 复杂指令集), RISC 的优势是其所有指令的长度都是一致,而且指令周期也是相同的。根据这些特性, MIPS 中可以采用指令并行中的流水线技术。另一方面,实际生活中 MIPS 的应用也十分广泛,如路由器等,甚至有一些超级计算机也采用 MIPS 架构。为了尽可能提高此次设计系统的整体性能,片上系统采用 MIPS32 这种体系结构。文献[7]、[8]、[9]给出了 NEXYS4 开发板的一些开发指导以及部件说明。

和 x86 的寄存器命名不同, MIPS 提供了 32 个 GPR (GENERAL PURPOSE REGISTER)。通过总结文献[10]、[11]、[12],可以将它们归纳如下表 1-1 所示:

表 1-1 CPU 通用寄存器

REGISTER	NAME	USAGE
\$0	\$zero	常量 0
\$1	at	保留给汇编器
\$2-\$3	\$v0-\$v1	函数调用返回值
\$4-\$7	\$a0-\$a3	函数调用参数
\$8-\$15	\$t0-\$t7	临时变量
\$16-\$23	\$s0-\$s7	保存的寄存器
\$24-\$25	\$t8-\$t9	暂时的寄存器
\$26-\$27	\$k0-\$k1	中断和自陷程序使用
\$28	\$gp	全局指针
\$29	\$sp	堆栈指针
\$30	\$s8/\$fp	帧指针
\$31	\$ra	返回地址

通过上表可知在 MIPS 中,还存在一些寄存器被冠以特殊的名字,并且拥有特

殊的功能，但是这些都只是开发者之间约定俗成的规定，实际上这些寄存器都是完全等价的。

除了通常的逻辑运算之外，处理器都需要一些硬件部件来实现中断异常的处理以及其他功能。CP0 (Co-Processor 0, 协处理器编号 0) 能够获取 CPU (Central Processing Unit, 中央处理单元) 的状态信息, 并辅助它完成整个操作系统的功能。作为一个协处理器, 它控制了整个芯片系统的运行状况, 并负责处理 CPU 的异常状态, 其主要负责了以下工作:

1. CPU 的配置
2. 高速缓存控制
3. 异常/中断控制
4. 存储管理单元控制
5. 定时器、计数器等

同样地, 根据文献[13]、[14]、[15]对 CP0 这个硬件的描述, 和真实的 NEXYS4 开发板的设计的不同之处, 列出了 CP0 寄存器的设计信息。见表 1-2:

表 1-2 CP0 协处理器的寄存器

寄存器名称	CP0 寄存器编号	功能
SR	12	状态寄存器, 如 CPU 特权等级
Cause	13	导致异常或者中断的原因
EPC	14	异常程序计数器
Count	9	简单有用的高分辨率定时器, 频率约为 CPU 流水线频率的一半
Compare	11	
BadVaddr	8	导致最近的地址相关异常的程序地址
Context	4	存储器管理和地址转换硬件 (TLB) 的寄存器
EntryHi	10	
EntryLo0-1	2-3	
Index	0	
PageMask	5	
Random	1	
WatchLo	18.0	数据观察点工具
WatchHi	19.0	

上表总结了 CP0 中一些重要寄存器的功能, 这里所需要提到几点的就是:

1. 使用 Count 寄存器和 Compare 寄存器作为一个定时器, 来作为 MIPS 结构中的定时器模块。

2. 为了辅助页式地址转换的机制进行访问, MIPS 通过 CP0 的几个寄存器控制并管理地址转换硬件, 从而直接访问 Cache 的数据。

3. 异常或者中断处理时, 通过 Cause 来知道哪种类型的异常并进行处理, 处理结束后, 再通过 EPC 从异常中返回。

4. 一个十分重要的寄存器就是 BadVaddr 寄存器, 每当运行到一条非法指令的时候, 它都指出了指令发生错误的地方。这可以充分表达出 QEMU 模拟器和 NEXYS4 硬件之间关于指令集上的差异。更重要的是, 它同时也为缺页处理时提供了异常的页面信息。

5. 在 QEMU 模拟器中还拥有 Watch 的两个寄存器, 他们用于给出观察点的数据信息, 因此在内核启动时通常会初始化这两个寄存器。但是硬件设计上没有也不需要这两寄存器, 为了防止在硬件中对它们进行初始化而引发错误, 需要使用动态宏对它们加以区分。

6. NEXYS4 开发板设计的硬件上的 CP0 寄存器和 QEMU 的 CP0 寄存器存在一些差别, 之后在第 2 章提到这些差别以及设计内容。

操作系统是管理和控制计算机硬件与软件资源的计算机程序, 是可以直接运行在硬件机器上的最基本的系统软件, 任何其他软件都必须在操作系统的支持下才能运行, 操作系统为用户提供了用户接口和系统调用, 为多用户使用硬件资源提供了极大的便利。从早期 Unix 开始, 经历 Minux 的发展, 至 Linux、Windows、MacOS 这些桌面系统的成熟, 到现在 Android、IOS 等移动 OS 的出现, 操作系统一直在人类生活中扮演着不可或缺的角色。

首先, 在这里介绍该 kernel 的基本构成、各个组成模块之间的关系。为了兼容多体系结构的特点, 操作系统采用基于接近 Linux 的项目风格构建了本项目的目录结构。其结构如下表 1-3:

表 1-3 目录结构示意

目录名	负责内容
bootloader	Boot 程序, 载入内核程序
ht-mksfs	制作磁盘工具

kern-ucore	内核代码
\---arch	多体系结构目录
\--mips	对应体系结构的目录
\-debug	内核调试程序
\-driver	驱动程序
\-glue-ucore	内核头文件
\-init	内核 entry 入口以及 init 函数
\-libs	内核库文件
\-mm	内存管理和页式地址转换
\-process	进程结构和进程相关的函数调用
\-syscall	系统调用
\-trap	中断和异常处理
\---fs	文件系统
\---glue-ucore	一些公共头文件
\---libs	系统库文件
\---mm	虚拟内存分配检查
\---process	一些进程相关控制和信息获取的函数
\---schedule	进程调度算法
\---syscall	提供一些简单函数，如获取当前时钟
libs-user-ucore	用户运行库文件
user-ucore (和 apps 类似)	用户应用程序

结合上表重点分析比较重要的目录内容：

1. apps，放一些比较大的应用程序，如贪吃蛇游戏。
2. bootloader，boot 程序，将内核从磁盘载入内存。
3. ht-mksfs，MKSFS 程序，用于制作系统的磁盘文件。

4. kern-ucore 中，arch 是多体系结构体现的根本区别，其中可以参与编译的有 i386 和 mips 两种。在这些结构的目录下，driver 实现了驱动，mm 为物理内存、虚拟内存以及页式地址转换的部分，libs 为一个内核的库文件实现，process 则是进程管理的部分，trap 和 syscall 则分别代表中断异常处理和系统调用。而在二级目录中，和 arch 并列的，都是一个内核中多种体系结构公共的部分，他们更加偏向

软件和算法的设计与实现。

5. lib-user-ucore, 用户级别的运行库。

6. user-ucore, 基本的系统软件。

然后, ucore 的 Makefile 目录树结构如下表 1-4:

表 1-4 Makefile 关系表

Makefile	根目录的 Makefile 文件
\---kern/Makefile.build	指导编译内核
\--kern/Makefile.subdir	编译子文件夹源码, 生成.o 模块
\-kern/Makefile	指定编译模块
\---user/Makefile	指导磁盘编译
\---apps/snake/Makefile	指导应用贪吃蛇编译

整个项目编译经过三个阶段: 配置、内核编译、磁盘生成。因为在 MIPS 结构中, 磁盘被作为一个 ramdisk 并加载到内核的 data 段之中, 故与 i386 架构编译时不同, 磁盘生成将被放在内核编译之前。现在借用编译 MIPS 架构进行整个过程的分析:

1. 整个项目的配置, 这一步主要配置编译时的体系结构、交叉编译器、模块选择或增减等项目设置。通过指令 `make ARCH = mips defconfig` 进行项目选项设置, 也可以通过 `menuconfig` 进入图形界面设置。

2. 磁盘生成, 这一步主要编译用户运行库和用户程序, 并制作成一个磁盘。通过指令 `make ARCH=mips sfsimg` 生成 ramdisk 文件镜像。

3. 编译内核, 并将上步生成的 ramdisk 通过 `ld` 载入到 kernel 的 data 段。通过指令 `make ARCH=mips` 即可生成最终的 kernel 镜像。

文献[16]、[17]采用的设计方式都是基于 Linux 的, 因此在系统设计上没有太多要求。而文献[18]详细描述了一般操作系统设计原理, 在页式地址转化一章中起到了极大的帮助作用。

1.3 研究内容

通过前一节的内容已经基本对 MIPS 架构的一般硬件结构有了基本的了解。但是, 实际上 QEMU 模拟器模拟的 MIPS32S 硬件和使用 NEXYS4 开发板设计的 MIPS32 硬件存在一些差异。针对这些差异, 为 NEXYS4 开发板定制操作系统。

1. 基于 NEXYS4 开发板设计的 MIPS32 位 CPU 使用了重新筛选的精简指令集, 但是编译器却在编译的时候会生成一些存在于标准指令集中, 但不存在与精简指令集中的指令。这些指令在硬件上执行时会触发非法指令异常, 从而导致整个系统的运行崩溃。

2. 基于 NEXYS4 开发板设计的定制外设有着独特的硬件结构, 因此, 需要在系统中设计对应的驱动来让操作系统可以通过外设正常与外界进行通讯。根据自己设计的硬件设备或者已存在的硬件设备, 以及外设统一编址的情况, 为操作系统添加外设驱动。由于硬件差异, 直接编译的内核并不能直接在开发板上执行, 因此, 编译的内核文件还需要进行预处理。

3. 片上系统上定制的外设结构同样和 QEMU 中的外设结构不同, 此部分主要在 NEXYS4 开发板上搭建硬件平台, 然后根据硬件平台和模拟器的架构差异对 kernel 进行修改。最后设计相应接口并根据统一编址的情况, 为操作系统添加外设驱动。

1.4 论文组织和结构

本文的章节安排和组织如下所示。

第 1 章: 介绍选题背景知识、国内外研究现状, 以及本文的研究内容。

第 2 章: 介绍和展示模拟器和真实硬件的指令集差别以及实现方案

第 3 章: 介绍和展示模拟器和真实硬件在缺页异常处理上的实现

第 4 章: 介绍和展示真实硬件的定制外设设计和驱动实现

第 5 章: 介绍和展示运行整个项目的各种测试结果

第 6 章: 总结本文的工作和提出未来的展望

2 系统设计需求分析

一台计算机的基本构成的要素为 CPU 和内存。CPU 提供了逻辑运算能力；内存提供了存储能力。同时，为了满足人机交互的需求，还需要设计一系列外设来满足计算机和外界的信息传递。因此，针对硬件结构进行操作系统设计通常都是从这三个角度进行考虑得到对应的设计方案。

2.1 片上系统和 QEMU 模拟器间的差异

对于 CPU 器件来说，最重要的就是指令集的异同。从文献[19]、[20]中可以知道 MIPS32 的标准指令集当中指令的内容和对应的逻辑功能。通过对片上系统的测试，和 QEMU 模拟器采用的标准 MIPS32 指令集相比，有以下不同之处：

1. 部分指令对应的机器码错误。如 sll 指令最后一位应该是 0，而片上系统中的 sll 指令最后一位为 1。

2. 部分指令缺失。为了提高 CPU 的运行频率，一些逻辑复杂的指令的执行周期远远大于其他指令的执行周期，如乘除法指令。在优化指令集的时候，将这部分复杂逻辑的指令从精简指令集中排出。

对于内存来说，主要考察作为页式地址装换的关键硬件 TLB（Translation Lookaside Buffer，快表）和 CP0 寄存器的差异进行比较：

1. TLB 中没有 PageMask 段的内容。
2. CP0 没有 PageMask 寄存器，无法对系统中的页面大小进行设置。
3. CP0 没有 Random 寄存器以及缺乏 TLB 随机刷新指令，无法对 TLB 进行随机刷新。
4. 在 TLB 缺失异常处理流程中，需要完成 TLB 缺失异常触发机制以及由内存页面向 TLB 进行填充的机制。

对于外设部分来说，两者相差很大，因此需要根据片上系统的设备协议，在操作系统中设计新的驱动，来完成操作系统与外界的交互过程，主要差异如下：

1. 串口设备的数据寄存器地址与状态寄存器的地址是重新编址，串口的协议也进行了一些变化。
2. QEMU 中不具备键盘外设，同时片上系统的键盘外设存在问题。
3. QEMU 中不具备 VGA（Video Graphics Array，视频图形阵列）外设，片上

系统存在该外设，需要在系统中完成设计。

2.2 基于片上系统的操作系统设计方案

在前一节中比较得出了基于 NEXYS4 的片上系统和 QEMU 模拟器之间的差异。这一节中主要探讨针对这些差异而采用的设计方案。

对于 CPU 指令集之间的差异，首先对片上系统进行了一些完善工作，将一些对应错误机器码的指令进行修改，使其可以正常工作。对于那些影响 CPU 工作性能的指令使用片上系统已有的速度比较快的指令通过算法进行组合并实现逻辑上的替代实现。为了保证高效性，除了对几种算法的空间复杂度和时间复杂度进行对比，同时还参考了文献[21]中的一些特殊情形下的快速算法。

对于 TLB 缺失异常的处理，首先需要设计如何在异常处理中设置对 TLB 缺失这类异常的处理。因为在异常处理中，还是需要根据 TLB 的构造来对其进行数据刷新，这需要了解片上系统中 TLB 的硬件结构。因为 CP0 中 Random 寄存器的缺失和 TLB 随机刷新指令的缺失，所以无法再通过随机刷新 TLB 的方式来刷新 TLB 的数据内容。但是 CP0 中仍然保留了 Index 寄存器以及指令集中仍然提供了按照 Index 寄存器的数值刷新 TLB 的功能指令。因此，根据提高的硬件机制和软件指令，可以设计一种 FIFO (First Input First Output, 先进先出) 的方式对 TLB 中的数据进行刷新。

对于外部设备的差异，首先要将设备按照设备类型分为三种类型，然后为每一种类型的设备完成驱动的设计，对于一些存在问题的硬件外设，则需要根据一些文献的协议描述，重新设计外部设备的硬件部分。对于串口，主要是完成驱动的实现，主要有：系统如何响应串口的输入事件，并进入到指定的驱动函数来使系统接收到外界的数据输入；用户如何通过系统调用这种自陷异常来达到将应用中的数据显示在串口上的方式。对于键盘，首先需要重新设计键盘的状态机，这一步根据文献[23]中对键盘协议描述进行硬件设计。然后，因为设计的键盘是罗技键盘，属于 104 按键类型的键盘，因此需要通过文献[24]中的字符和按键的转换表来实现键盘驱动处理的设计。对于 VGA 设备，VGA 虽然在片上系统中存在，但是在 QEMU 模拟器中却不存在，但是此时需要根据片上系统对 VGA 的设计部分完成定制的驱动。在设计中需要提到的是，VGA 设定的显存空间接近 1M 大小，如果使用系统调用完成设计，会导致系统运行速度变慢。这里采用一种与之前的

设计方式不同的设计方案。由于 I/O 部分的虚拟地址是不需要进行 TLB 映射和缓存就可以直接进行访问了。所以,用户也可以直接对 VGA 的虚拟地址进行写操作,从而完成输出过程。但是考虑到系统的安全性,在系统调用中提供了另外一种系统调用,这个系统调用利用 VGA 中额外设计的一个用于保存系统状态数据的数组来恢复被用户写过的 VGA 状态数组,从而使整个 VGA 的状态恢复到用户程序使用之前的状态。这样直接比起使用系统调用过程大大加快了使用 VGA 的应用程序的执行速度。

2.3 本章总结

本章主要从一台计算机的角度,从 CPU、内存、外设这三个部分对片上系统和 QEMU 模拟器进行对比。按照为片上系统定制操作系统的目的对每一个部分的需求进行了分析,并制定了设计方案和解决方案。

3 乘除法指令的替代与实现

为了加速 CPU 的执行频率,在硬件设计时将一些耗时较长的指令从原本的 MIPS32 指令集中移去,得到了一个精简版指令集。这个指令集不具有乘法指令、除法指令以及浮点计算指令等等。而这个精简后的指令集仍足以运行一个操作系统。但是操作系统又需要同时在 QEMU 模拟器和真实硬件上运行。因此可以使用了编译时添加额外宏定义的方式来选择编译两种不同运行方式下的内核。

在 Makefile 中添加可选参数 ON_FPGA,然后根据该选项开关选择添加编译宏 -DMACH_QEMU 或 -DMACH_FPGA。然后在进行编译的时候,即可将 QEMU 模拟器所对应的代码和 NEXYS4 硬件对应的代码通过宏定义的方式进行区分编译。

因为在精简后的指令集中不能支持原本指令集中的一些指令,因此在编译器进行代码编译时,多多少少会产生一些不能支持的指令。因此,需要通过改写原有的算法格式,使得编译器在编译时编译多条已经支持的指令,从而代替原有指令。

这一章节主要介绍如何通过软件的实现弥补 NEXYS4 开发板在硬件设计上的不足,并在此基础上保持对整个系统在运行上的高效性。

3.1 乘法指令的替代与实现

由于乘法指令在实现一些复杂算法过程时提供了极大的便利性。然而,在内核设计时,乘法指令同样扮演着不可或缺的角色。在人机交互过程中,除去字符设备可以直接输入输出字符外,系统还需要对一些数字的输入进行处理才可以得到正确的逻辑关系。在对系统进行调试时经常需要在系统中打印一些地址来获取调试信息,打印这些信息都是通过 `kprintf` 这个函数实现的。除此之外,还有一些内核函数也会涉及到乘法运算,以 `free` 程序为例:

系统剩余空间大小 = 空闲页面的数量 * 每个页面大小

所以,这些都离不开乘法指令的功能。除了进行系统开发和调试外,用户通常也需要在开发程序时进行一些乘法运算和格式化输出结果的功能。更重要的是,虽然为了提高 NEXYS4 硬件设计上的性能对指令集进行了一些精简,但是不能因此而导致整个系统在功能上产生缺陷。

于是就有了一种十分自然的解决方案。对一个形如 $n*m$ 的乘法计算,只需要

使用 $O(n)$ 条加上 m 的加法指令就可以完成乘法过程。但是这种方式存在以下三个缺陷:

1. 需要知道 n 和 m 中的某一个数字的数值。
2. 比起原来一条乘法指令, $O(n)$ 条指令会花去更多的空间
3. 运行需要花费 n 条指令的时间

然而, 还有一种可以通过对一条加 m 的指令循环 n 次进行计算的方法, 避免了之前方案中 1 和 2 的缺陷。但是这种方案仍然需要花费大量的时间, 作为一种基本而且常见的指令, 这种替代方式十分低效, 不符合硬件设计的初衷。

为了进一步优化系统性能, 只能采取了新的方案。因为任何一个数字在计算机之中都可以使用二进制表示, 所以只需要根据其中的一个数字的二进制表示对另一个数字进行移位和加法操作, 同样可以得到乘法的计算结果。这里以 $n*10$ 为例表示过程:

1. 10 的二进制表示为 1010
2. 从右往左扫描 10 的二进制形式的每一位
3. 如果该位为 0, 就不进行处理; 如果该位为 1, 则将结果加上 n 向左边移动该位的位数的数值。如: 10 的第 0 位为 0, 因此不做处理, 而第 1 位为 1, 所以将结果加上 $n \ll 1$ 的数值。
4. 当将该二进制数遍历一遍后, 结束该过程, 之前求和的结果即为答案。

因为在 MIPS32 位中, 一个数字的最长为 32 位, 因此最多遍历 32 位即可结束求积过程。因为所有的数字都可以像 10 一样完成二进制展开, 因此该方法对任何两个数字的乘积都是有效的。

算法 3.1 乘法指令的算法实现

输入: 乘数: n, m

输出: 乘积: $z = n*m$

```

1:   n = 0, m = 0, z = 0, i = 0
2:   while ( m != 0 ) { # 遍历 m 所有的二进制位
3:       if( m & 0x1) # 判断 m 的第 i 位是否为 1
4:           z += ( n << i) # z 加上 n 左移 i 位的数值
5:       m = m >> 1; # 指针移到 m 的 i+1 位
6:       i ++ ;}
7:   return z
    
```


这样通过加减指令和移位指令的组合,可以完全代替原来的乘法指令。而且,代替原本低效的通过循环加法实现乘法的方式,算法 3.1 仅仅使用常数时间即可达到相同的效果,从而达到高效设计的目的。

2.2 除法指令的替代与实现

在上一节中完成了乘法指令的替代与实现。但是,具备格式化输出功能的 `kprintf` 函数仍然无法正常工作。在输出一位数字的时候, `kprintf` 可以直接以字符形式输出数字结果;当输出多位数字的时候, `kprintf` 就需要对该数据的每一位数字按照字符的形式输出。为了满足这一要求,需要设计的一个函数 `printnum` 来负责 10 进制下高位数据的输出,其流程图如下:

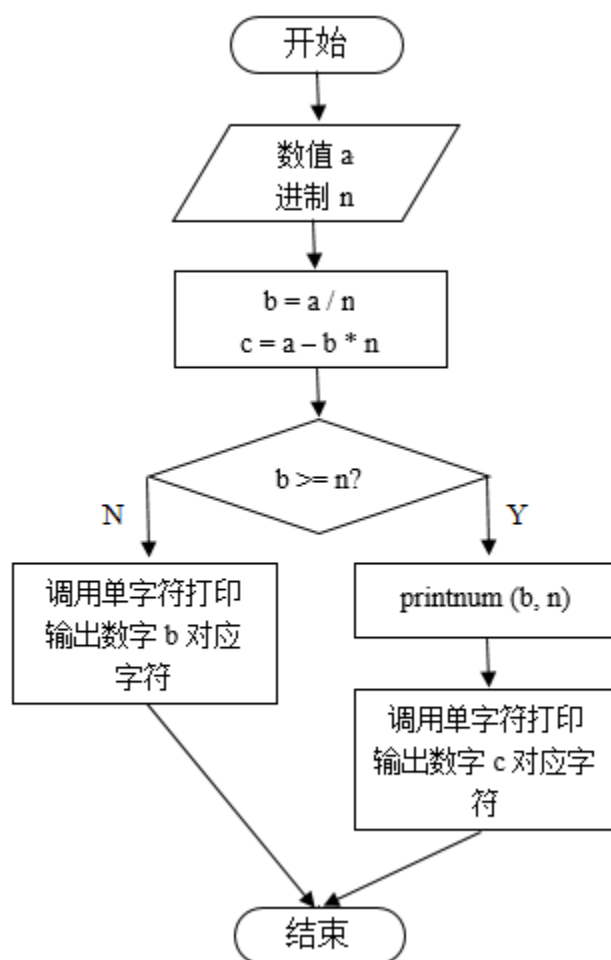


图 3-1 `printnum` 打印任意函数流程图

从图 3-1 中可以看出 `printnum` 在每次迭代中都会除以进制 `n` 和乘以进制 `n`, 来获取下一位的输出结果。虽然前一节中解决了乘法指令的替代问题,但是精简指令集中也没有包括除法指令,而类似于 `printnum` 这样的系统运行库函数中又存在

一些直接使用譬如 $a = b/c$ 这样的算式。这些算式将会被编译器直接翻译成 MIPS 中的除法指令，因而这些指令在开发板上不能直接运行。为了解决这个问题，同样可以通过移位指令和加减指令来逻辑上实现原有的除法指令的功能。特别的，针对关于 10 进制的输出中使用到了 $a = b/10$ 的除法，可以通过以下算法实现：

算法 3.2 除 10 指令的算法实现

输入：被除数：m

输出：商：m/10

```

1:    q = 0, r = 0
2:    q = (m >> 1) + (m >> 2)  # q = 3m/4
3:    q = q + (q >> 4)
4:    q = q + (q >> 8)
5:    q = q + (q >> 16)
6:    q = q >> 3
7:    r = m - ((q << 3) + (q << 1))
8:    q = q + ((r + 6) >> 4)
9:    return q

```

该算法来自文献[22]的描述，它十分巧妙的使用了移位指令和加减法指令达到了 $m/10$ 的功能。虽然该算法很巧妙，而且十分高效，仅仅用在对 10 进制数格式化输出当中是十分恰当的。但是对于用户程序来说还有明显的不足之处，因为在用户程序中，除数往往是未知的，而算法 3.2 又无法解决这个问题。因为除法是乘法的逆过程，类比算法 3.1，并得到了新的解决方案。

算法 3.3 除法指令的算法实现

输入：被除数：m, n

输出：商：m/n

```

1:    q = 0, x = bins(m), y = bins(n);  #bins 函数，获取该数字的二进制表示位数
2:    i = x - y;
3:    while(m >= n){
4:        if(m >= (n << i)) {m = m - (n << i); q++;}
5:        if(i == 0) break;  # 移位结束
6:        i--; q = q << 1; }  # 移动下标，进行下一位处理
7:    q = q << i;  # 将 q 按照剩下的位数进行左移位处理
8:    return q

```

这样虽然损失了一部分的性能,但是却实现了任意两个数字之间的除法。这样就可以完成通过移位指令和加法指令完成对除法指令的替代。而在系统设计过程中,需要根据具体的情况使用不同的算法方式尽可能地优化系统性能。考虑用户一般只会使用 10 进制输出,于是可以修改 `printnum` 函数的基数为 10,并使用了算法 3.2 尽可能提高系统性能。而算法 3.3 则被保留作为用户库函数中供给用户进行使用。

最后,在操作系统的一些函数和用户程序中也常常出现一种函数,那就是随机函数。然而在设计一些应用程序时常常用到随机数,如猜数字程序,如果没有随机数,那么每次用户运行这个游戏时都是程序内部决定好的结果,这样的程序就失去了娱乐性能。类似地,贪吃蛇程序中的苹果出现的位置也应该是随机出现的。如果每次程序运行时,苹果出现的时机都和前一局一样,时而久之,大家就会对此感觉乏味。由此可见,如果没有随机函数过程,很多程序都会出现设计上的不足。

早期的随机算法也是通过除法实现的。当然这种方式可以通过之前的指令替代的方式进行处理。但是这样对随机过程来说明显会有不足之处。因为,随机函数的被除数和除数都很大,而算法 3.3 对于这种除数非常大的情况来说,效率还是不算太高。因此,还有必要考虑对这个随机函数的算法进行完全的替换。这里改写后的随机函数使用一种常见的线性移位实现的算法,这种算法和 C 语言的 `random` 函数类似,即前面生成的随机数和下一个生成的随机数存在确定性关系,属于一种伪随机过程的实现。因此,还需要额外设计一个 `srand` 函数,将当前系统运行的 Ticks 传入随机数中作为 `seed`,这样可以提高随机函数的性能。

通过基本指令的替换以及基本库函数的实现,操作系统已经可以正常地执行了。一些库函数也足够帮助用户开发一些有趣的测试应用,大大增加了系统的可用性。

3.3 本章总结

本章主要解决了 QEMU 模拟器和片上系统之前的 MIPS 处理器中关于指令集差异的解决方案。通过片上系统中已经存在的指令,通过设计算法对加减指令和移位指令进行组合,从而完成乘除法指令的替代。

4 缺页异常的处理

操作系统和用户程序都存放在磁盘之中，而不是内存，采用这种方式的原因有很多，如：

1. 使用易失性存储介质作为内存，断电时导致存放的用户数据丢失
2. 本应存放在磁盘上的用户程序、数据内容远远超过内存的大小。

但是，为了保证程序的高效运行，又不能让 CPU 直接去磁盘寻找数据。因此，需要一种方案来主动将数据从磁盘载入内存之中，这就是缺页异常发生的来源。由于 NEXYS4 开发板设计的硬件架构与 QEMU 模拟器中有很大的差距，因此也导致了两种情形下对缺页异常处理方式的不同。

本章首先介绍基于 NEXYS 硬件设计的缺页填充机制，然后介绍在系统中如何触发并解决中断和异常，最后完成本系统对缺页异常的解决方案。

4.1 基于 NEXYS 硬件的缺页填充机制

在 NEXYS4 硬件中，CPU 主要通过 MMU（Memory Management Unit，内存管理器单元）部件完成虚实地址的转换。MMU 可以从物理内存里许多零散的页中创建连续的地址空间，使申请者能从一个充满固定大小页面的池里分配内存。MMU 是保障了系统和多用户程序的安全。因为在 MIPS32 中不存在实模式和保护模式的机制。但是作为替代，MMU 将整个虚拟地址空间划分为四段：

表 4-1 MIPS 虚拟内存地址分布

地址空间	访问方式	访问对象	功能
0xc0000000 - 0xffffffff	tlb-mapped	kernel	kseg0 的空间不够用则启用 High Memory 机制 使用 kseg2 的空间对 TLB 建立映射来访问
0xa0000000 - 0xbfffffff	unmapped uncache	kernel	I/O 外设访问 ROM 段程序的起址
0x80000000 - 0x9fffffff	unmapped cache	kernel	默认 kernel 使用的地址空间
0x00000000 - 0x7fffffff	tlb-mapped	user	默认 user 使用的地址空间

结合上表的内容，下面仔细罗列出这四段虚拟地址空间的内容：

1> [0x00000000, 0x7fffffff] (0~2G-1) KUSEG

这些地址是用户程序使用的地址。这些地址通常使用 MMU 进行页式地址转换。换句话说，除非 MMU 的机制被建立好，在 TLB 中存在对应的数据项，这 2G 地址是不可以进行的。在本次系统中，用户程序的起始地址皆为 0x10000000。

2> [0x80000000, 0x9fffffff] (2G~2.5G-1) KSEG0

这些地址通过映射的方式就可找到对应的物理地址。其方式为把最高位清零，然后把它们映射到物理地址低段 512M[0x00000000, 0x1FFFFFFF]。但是几乎全部的对这段地址的存取都会通过快速缓存。因此在 TLB 中存在这些数据对应的数据项之前，不能随便使用这段地址。通常一个没有 MMU 的系统会使用这段地址作为其绝大多数程序和数据存放位置。对于有 MMU 的系统，操作系统核心会存放在这个区域。

3> [0xa0000000, 0xbfffffff] (2.5G~3G-1) KSEG1

这些地址通过把最高 3 位清零的方式来映射到相应的物理地址上，与 KSEG0 映射的物理地址一样。但 KSEG1 是非缓存存取的。KSEG1 是唯一的在系统重启时能正常工作的地址空间。这也是为什么重新启动时的入口向量是 0xbfc00000。这个向量相应的物理地址是 0x1fc00000。将使用这段地址空间去存取初始化 ROM 程序。大多数人在这段空间使用 I/O 寄存器。这里展示出设计中使用的几个外设的地址映射表格：

表 4-2 外设统一编址分布

物理地址	虚拟地址	设备	附件说明
0x1fd003f8	0xbfd003f8	串口数据	可读写
0x1fd003fc	0xbfd003fc	串口状态	第 0 位判断是否可写 第 1 位判断是否可读
0x0f000000	0xaf000000	键盘	按键输入
0x0a000000	0xba000000	VGA	屏幕显示

4> [0xc0000000, 0xffffffff] (3G~4G-1) KSEG2

这段地址空间只能在核心态下使用并且要经过 MMU 的转换。在 MMU 设置好之前，不能存取这段区域。除非在设计一个控制物理内存超过 3G 的操作系统，一般来说你不需要使用这段地址空间。

通过各段部分的介绍可以知道 KSEG0 和 KSEG1 是可以不依赖 MMU，直接进行地址转换访问物理地址的。这也解释了为什么设计中的 kernel 是从 0x80000000 开始的（即位于 KSEG0 这一区间），且大部分外设的统一编址都位于 KSEG1 这一区间，如串口、键盘、VGA 等等。而对于 KUSEG 这一段地址，都是留给用户的应用程序使用的空间。

在了解了前面的四段地址的基础上，还需要对 MIPS 的内存转换机制进行详细说明，过程如图所示：

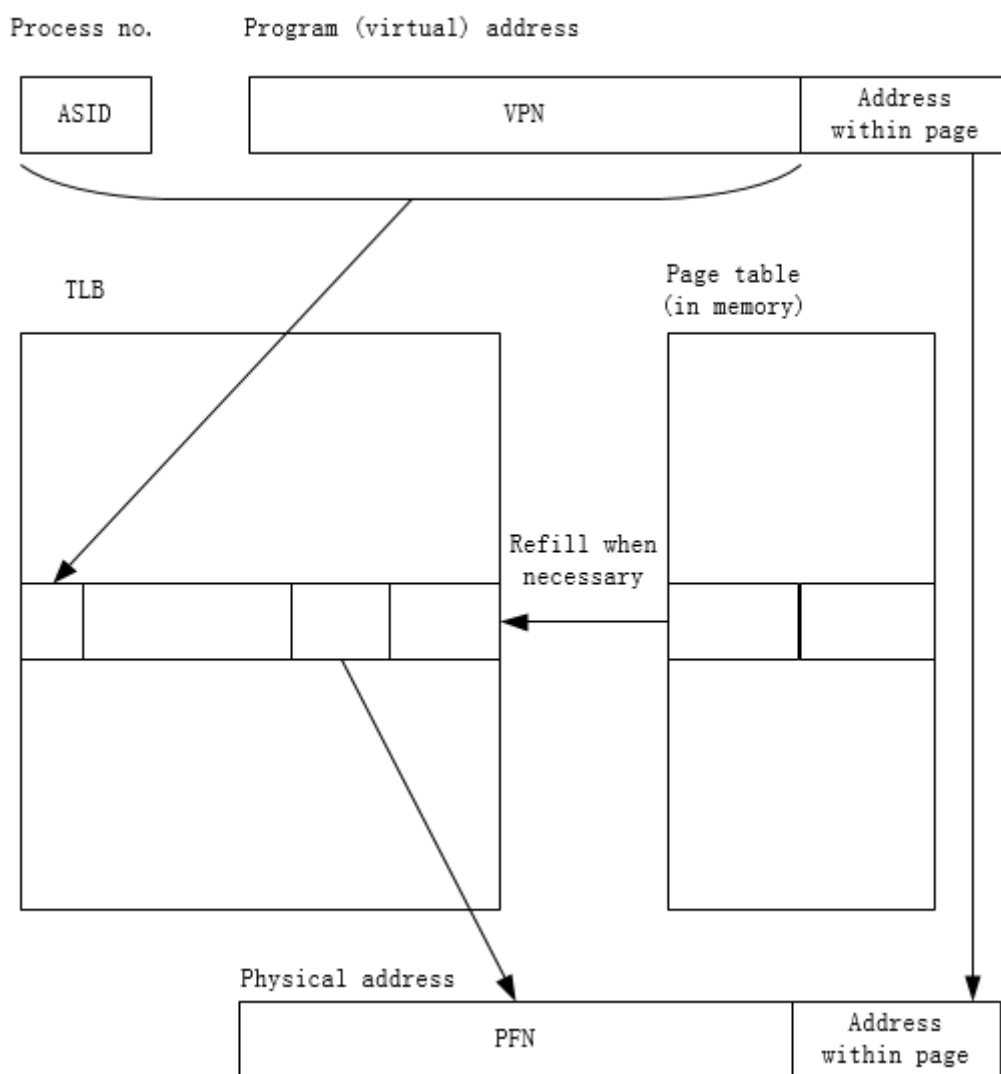


图 4-1 硬件-虚实地址转换机制

硬件的处理过程大致上是这样的：

1. 一个虚地址被分割为两部分，低半部分地址位（通常为 12 位）不经转译直接通过，因此转译结果总是落在一个页内（通常 4KB 大小）。因此这部分被称作页内偏移。

2. 高半部分地址位, 也就是 VPN (Virtual Page Number, 虚拟页号), VPN 由页目录和页表号组成, 然后会在前面拼接上当前运行进程的 ASID (Address Space ID, 地址空间标示符) 以形成一个独一无二的页首地址。因此也不必担心两个不同的进程的同一个虚拟地址会访问到同一个物理地址。

3. 在 TLB 中查找是否有一个本页的转译项在里面。 如果存在, 那么将得到对应的物理地址的高位, 最终得到可用地址。TLB 是一个做特殊用途的存储器件, 可以运用各种有效的方法来匹配地址。

这里的重点则是 TLB。TLB 是将程序的虚拟地址转换成访问存储器中的物理地址的硬件。因为 MIPS 的 CPU 的地址转换单位为页, 页内偏移是可以直接从虚拟地址传递到物理地址, 而虚拟页号和物理页号则需要通过查找页表来实现, 在 MIPS 中, 页表会被缓存到 TLB 当中, 即 CPU 是通过 TLB 将虚拟页号翻译成物理页号的。

TLB 中的每一项含有一个页的虚拟页号 VPN 和两个物理页号 PFN (Page Frame Number, 页帧号)。当程序给出一个虚拟地址, 该地址的 VPN 会和 TLB 中的每一个 TLB 进行比较, 如果匹配成功就再根据虚拟页号的奇偶性给出对应的 PFN, 并返回一组标志位让操作系统来确定某一页为只读或者是否进行高速缓存。

其结构如下图所示:

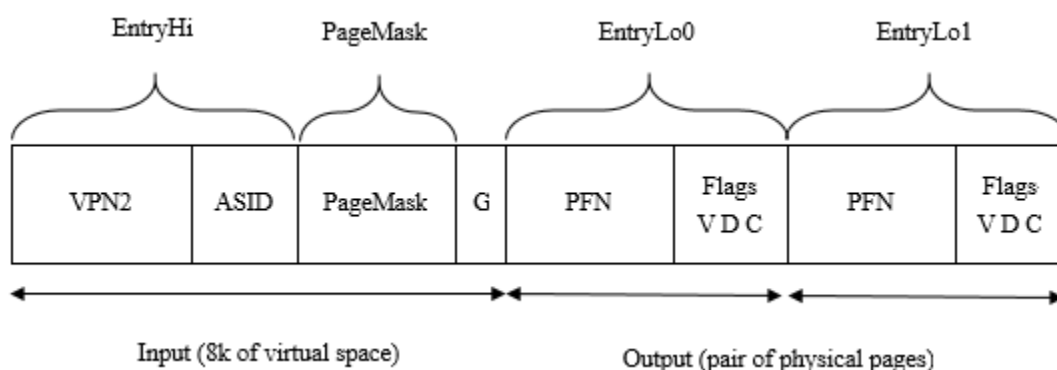


图 4-2 TLB 中的一项

上图为 TLB 中的一个数据项, 其可以容纳一对相邻的虚拟页面和对应的两个单独的物理地址。同时, 在图 4-2 中标出了加载和读取 TLB 表项时使用的 CP0 寄存器的名称。

为了顺利完成页式地址转换的过程, CP0 中有几个寄存器会辅助 TLB 进行地址转换, 下面列表说明:

表 4-3 用于虚实地址转换的 CP0 寄存器

寄存器助记符	CP0 寄存器号	功能
EntryHi	10	具备 VPN 和 ASID
EntryLo0	2	VPN 所映射到的物理页号, 以及对应物理页的存取权限
EntryLo1	3	
PageMask	5	用来创建能映射超过 4KB 的页的入口
Index	0	决定相应指令要读写的 TLB 表项
Random	1	这个伪随机值(实际上是一个自由计数的计数器)用来让 tlbwr 写入新的 TLB 入口到一个随机选择的位置。为那些使用随机替换的软件在陷入 TLB 重装入异常时的处理节省了时间
Context	4	这些是很有用的寄存器, 用来加速 TLB 重装入的过程。它们的高位可读写, 低位从不可转译的 VPN 中得来。寄存器的域这样布置使得如果您使用了合适的内存转译纪录的内存拷贝的安排方式, 那么紧跟在 TLB 重装入陷入后 Context 会装有一个指向用来映射触发异常地址的页表纪录的指针。XContext 在处理超过 32 位有效地址时做了相同的工作
XContext	20	

这里就 MIPS32 架构详细介绍一下几个常用寄存器, 并就 QEMU 模拟器和 NEXYS4 开发板之间的不同之处进行分析。

1. EntryHi

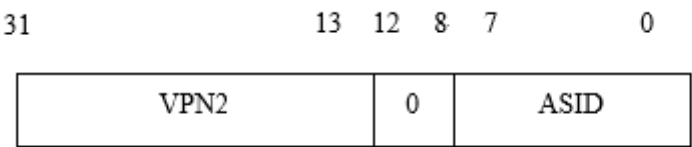


图 4-3 EntryHi 寄存器结构

VPN2 是除去页内偏移部分的虚拟地址, 一共有 20 位。ASID 是被用来保存操作系统当前的地址空间的标识。但是系统发生异常时, 该数据不会发生改变。因为处理完异常后, 执行的进程仍然可以对应该 ASID。总的来说, 这个寄存器就可以定位到某个进程的虚拟页号了。

2. PageMask

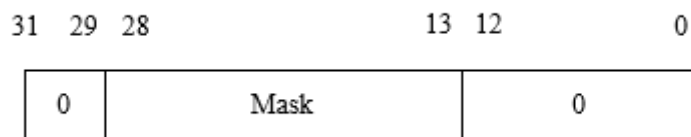


图 4-4 PageMask 寄存器结构

PageMask 是用来设置 TLB, 让它可以映射更大的页。这个寄存器在 QEMU 模拟器中存在, 但是在 NEXYS4 开发板中没有进行预留。但是这个寄存器的功能主要是设置映射的页面大小, 而系统常用的页面大小是 4K, 因此该寄存器的存在与否并不影响操作系统的运行。

3. EntryLo

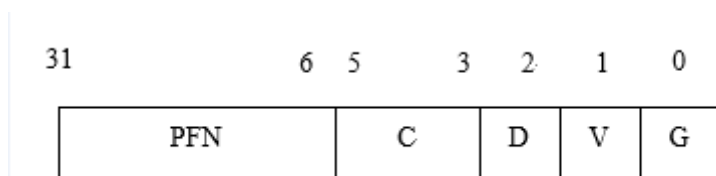


图 4-5 EntryLo 寄存器结构

PFN 是和 EntryHi 对应的地址转换项的物理地址的高位部分。C 是可以用来设置多处理器情况下的 cache 一致性算法, D 则代表常见脏位, V 代表该项是否有效, G 代表 global, 提供了一种所有进程共享的地址空间。

总结一下, 系统将用户进程的 ASID 和虚拟地址的 VPN 传给 CP0 的 EntryHi, 然后硬件就会自动从 TLB 中查找到对应的 EntryLo, 从而得到物理地址的 PFN 和相应物理地址的读写权限和高速缓存设置等信息, 从而用户可以获取进程中虚拟地址指向的指令或数据。

这里需要提到在 tlb_refill 的处理中, tlb_refill 这个函数的功能就是将存在于内存中的数据填充到 TLB 中, 从而 CPU 可能正常在 TLB 中使用虚拟地址匹配到对应的物理地址。但是, 事实上 QEMU 模拟器和 NEXYS4 真实硬件上存在不同之处。而在标准 CP0 当中存在一个 Random 寄存器。它的功能是生成一个这个伪随机值, 然后执行一个 tlbr 指令, 该指令的功能是选择一个新的 TLB 入口到一个随机选择的位置, 然后把数据写入到那个位置。这些机能为那些使用随机替换的软件在陷入 TLB 缺失异常后, 重装入 TLB 的处理节省了时间, 从而整体提升了效率。但

是, 在之前说过, 为了提高 CPU 的频率, NEXYS4 硬件中不再包含 Random 寄存器和 tlbwr 这条指令。因此, 操作系统不能使用这种随机写入页表项的方式来刷新 TLB 硬件中的数据。

为了解决这个问题, 可以换一种思维方式, 通过使用已有的 tlbwi 指令和 Index 寄存器来通过 FIFO 的方式实现 TLB 数据的刷新。处理算法如下:

算法 4.1 TLB 刷新算法的实现

输入: 发生 TLB 缺失异常的地址: BadAddr

输出: NULL

```

1:    #ifdef  MACH_QEMU
2:        write_cp0_register(BadAddr) #将一些信息写入 CP0 的辅助寄存器
3:        __asm__ __volatile__(“tlbwr”) #利用 Random 寄存器信息刷新 TLB
4:    #elif  defined  MACH_FPGA
5:        static int index = 0;
6:        write_cp0_register(BadAddr) #将一些信息写入 CP0 的辅助寄存器
7:        write_c0_index(index++)  #将 index 值写入 CP0 的 Index 寄存器
8:        __asm__ __volatile__( “tlbwi” ) #刷新由 index 指定的 TLB 项
9:    #endif
10:    return NULL
    
```

这个算法第一次展示出如何使用宏分支的方法来动态编译 QEMU 模拟器和 NEXYS4 硬件两种情况下各自的内核文件。代替原有的随机刷新 TLB 项的方式, 通过 Index 寄存器使用先进先出的方式刷新 TLB。虽然之先进先出的刷新方式比随机刷新的方式更可能增加 TLB 缺失异常的发生的次数, 但是考虑到精简指令集的设计可以提高整体 CPU 的频率性能并减少所有指令的运行周期, 这个设计仍能从整体提高系统的运行性能。

4.2 中断和异常处理机制

中断和异常是操作系统中极为重要的一部分。一个系统想要与外部进行有效通讯, 通常需要中断处理来达到目的; 类似地, 如果它想长时间的有效执行, 必须具备处理异常的能力。

每次在进入中断异常处理前, 系统都必须首先保护现场。因为在中断异常处理结束后, 系统是需要重新回到尚未执行完的任务中去。这一部分中, 主要是完

成两个部分：一个部分是保存现场和状态，主要是 CPU 寄存器和 CP0 寄存器；另一个是将中断向量号、异常处理号与对应的处理函数对应起来。

通过设计在硬件中可以允许发生的异常，归纳如下表 3-4 所示：

表 4-4 中断和异常处理号对应表

符号	编号	功能
EX_IRQ	0	Interrupt
EX_MOD	1	TLB Modify (write to read-only page)
EX_TLBL	2	TLB miss on load
EX_TLBS	3	TLB miss on store
EX_ADEL	4	Address error on load
EX_ADES	5	Address error on store
EX_SYS	8	Syscall
EX_BP	9	Breakpoint
EX_RI	10	Reserved (illegal) instruction
EX_CPU	11	Coprocessor unusable

相比通过设置中断门和异常门来实现系统对中断和异常的响应的方式，MIPS 主要是通过 CP0 给出的 CPU 的状态，然后按照上面的表格通过软件的方式进行处理。MIPS 的中断和异常的实现主要依赖于之前提到过的 CP0 的 Cause 和 EPC 来实现。这里，通过结合之前的 CP0 寄存器表格和异常处理号对应表，就可以轻松的完成这一部分的内容。

事实上，在本系统中，所有的外部中断和异常都被使用一个 handler 函数进行处理，从而整个中断作为一个分支被纳入到异常处理分发中。因此，这里提到的异常包括以下几个方面：

1. 外部事件。在有外部事件时，中断被用来引起 CPU 的注意。中断是唯一由 CPU 执行以外的事件引起的异常。当前只能通过 pic_disable 来使得中断变得无效。这里将存在的外部事件有键盘、串口和时钟等基本设备。
2. 内存翻译异常。当 CPU 没有根据虚拟地址找到与之对应的物理地址时，或者当程序试图写一个有写保护的页面时，会发生此种异常。
3. 程序或硬件检查出的错误。包括 CPU 执行到非法指令,在不正确的用户权限下执行的指令，地址对齐出错等等。

4. 系统调用和陷入。某些指令只是用来产生异常。它们提供了一种进入操作系统的安全机制。

关于异常处理的内容可以参考前面的异常处理号对应表 3-4。与常看到 CISC 硬件或微指令把 CPU 分派到不同的入口地址不同, MIPS 的协处理器 CP0 会通过 Cause 寄存器记住产生异常的原因,之后再内核中通过软件的方式派发到对应的异常处理函数进行解决。

下面使用流程图展示中断异常处理的设计流程:

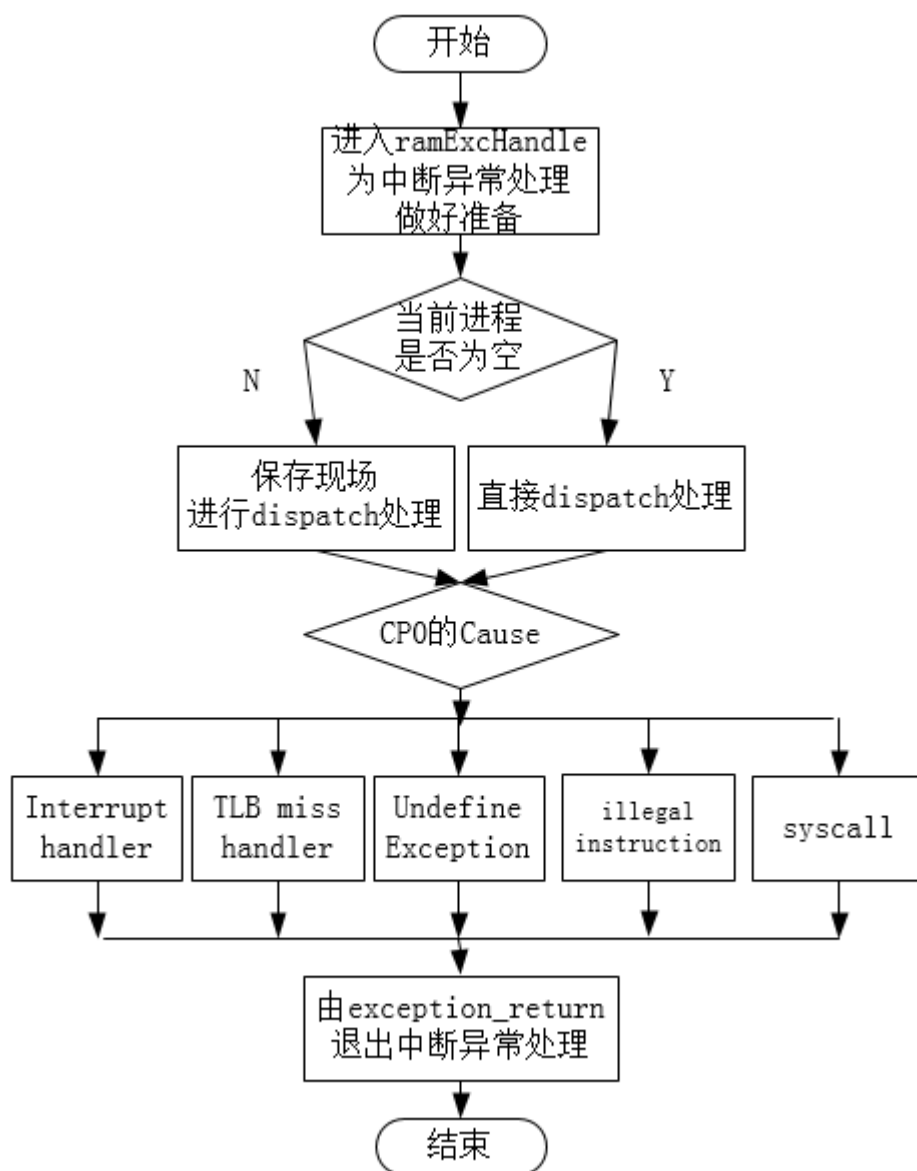


图 4-6 中断异常处理流程图

整个中断异常处理流程如下:

1. 发生中断或异常,首先进入通用处理入口 ramExcHandle, 这里的操作保存当前进程现场, 获取 CP0 寄存器信息等等, 为中断异常处理做好准备。

2. 根据 CP0 的 Cause 寄存器来决定中断处理的函数或者异常处理的函数，大体上可以划分为 5 类：中断处理、TLB 缺失处理、非法指令处理、系统调用、未定义异常。

3. 根据各自的原因进行中断和异常的处理，如外部中断则根据中断向量号来找到对应的处理函数；系统调用则根据系统调用号找到对应的内核函数进行处理；TLB 缺失则调用相应平台的缺页填充机制或者将磁盘数据搬运到内存中。

4. 处理完中断异常后，统一进入 exception_return 过程，这一步将恢复之前保护的线程，修改 CP0 的异常状态，并继续执行程序。

4.3 片上系统的缺页异常处理

从操作系统角度来看，地址转换是存储管理的一个主要特点。所谓地址转换就是将用户的逻辑地址转换成内存的物理地址。需要指出的是，地址转换是操作系统的地址变换机制自行完成的，不需用户进行干预，这样用户使用操作系统时，才方便而可靠。

地址变换的方式有很多，但是为了充分使用内存资源，系统使用最常用的页式地址转换机制。通过文献[7]的介绍，可以知道在 32 位的系统中，一般使用二级页表的方式来实现页式转换机制的。32 位划分为 10+10+12 位，其原理如下图所示：

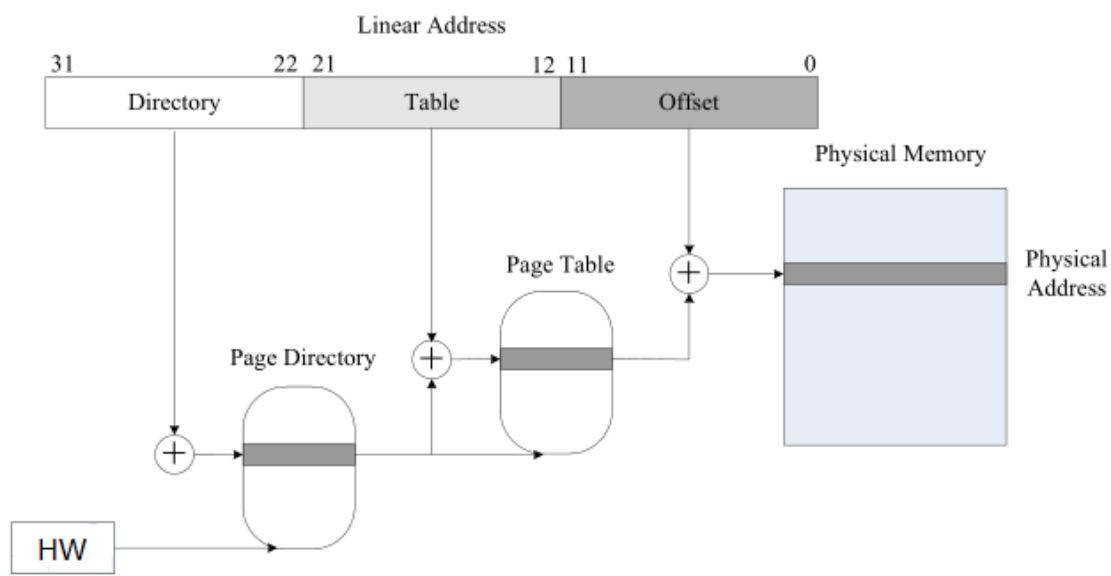


图 4-7 二级页表转换原理

表项中的高 20 位地址能够定位到内存的任何一个物理页面的首地址。如果是在页目录下，则找到的是下级页表项的物理页面首地址。如果是页表中找到就是数据的物理页面的首地址。该首地址结合虚拟地址的低十位的页内偏移，则可以

定位到数据在物理内存的存放位置。

在 MIPS 中 CPU 访问内存的一般过程为, CP0 中的一些寄存器会先帮助 CPU 访问 TLB (快表) 中的指定项内容, 然后再根据对应的 TLB 表项找到物理地址, 然后再通过物理地址进行访问。如果 TLB 中没有数据, 则会发生 TLB 缺失异常, 然后根据之前中断异常处理的设计流程知道, 此时会有对应的 TLB 缺失的 handler 函数处理 TLB 缺失异常, 然后处理完异常后的程序即可满足 CPU 在 TLB 表项中访问到物理地址。此时处理 TLB 缺失异常的过程有两种情况:

1. 需要访问的地址存在于在内存中, 但是其对应的内容不在 TLB 的每一项中。
2. 需要访问的地址根本就不在内存中, 此时该数据一定不会出现在 TLB 的任意一项中。

对于第一种情况, 系统只需要根据出现异常出现的地址, 然后再通过 `tlb_refill` 函数缺失的页面地址信息重新载入到 TLB 当中。

而第二种则是先将数据从磁盘载入到内存中, 然后再使用 `tlb_refill` 进行数据。在这种情况下, 系统首先需要获取出现异常出现的地址以及其相关权限信息, 如果因为权限信息表明该地址属于异常访问的地址时, 系统将会返回一个错误码。如果权限信息正常, 则系统会将磁盘中的数据载入到新分配的页面当中, 并将异常地址信息载入到 TLB 中。

4.4 本章总结

本章主要解决了 TLB 缺失异常处理的问题, TLB 的刷新关系到整个操作系统以及用户的程序能否正常执行。同时, 很多细节值得注意, 包括操作系统如何分辨不同的应用进程, 在不同的模拟平台下, 如何编译不同代码模块, 从而生成不同平台的内核文件。

5 片上系统的设备与驱动实现

设备，一般泛指外设。在一个计算机系统中，外设指的除去中央处理器和存储器的输入、输出设备的统称。外设对信息和数据起着传送、转输和存储的重要作用。同时，外围设备也是起到辅助功能的与计算机连接起来的设备，它们能扩充计算机系统，并使得计算机之间的交互成为可能。

本次设计中主要实现三种外设驱动，分别为串口、键盘和 VGA。其中：

1. 串口，是计算机最早和外界进行通讯的手段之一，是一种可读写设备。这里系统使用的串口为 UART16450，波特率为 115200，是一种使用异步串行方式进行通讯的串口。

2. 键盘，是计算机中一种比较常见的输入手段。根据开发板的条件限制，NEXYS4 开发板只能支持使用罗技系列的键盘进行驱动设计。

3. VGA，是一种比较常用的显示设备。这里 VGA 使用滚屏方式来进行内容的更新显示。这种更新方式比较容易理解，但同时也带来了效率上的问题。

在 ucore 中，设备分为三大类：只读设备、只写设备和读写设备。其中，系统对读设备采取中断方式进行交互处理，对写设备采取轮询的方式进行交互处理。开发一个驱动的时候，需要确定该设备对于操作系统来说所属的设备类型并针对其类型来确定如何让操作系统来控制 and 响应外部设备。

在中断方式中，首先需要注册外设的中断，然后系统就可以响应该设备的输入信息。当有外部输入时，系统首先会进入中断处理函数，然后使用设备的中断向量号来对应到设备的驱动处理函数，从而获取来自设备的数据并加工成可识别的有效字符。

在轮询方式中，则是通过系统调用的方式进行，系统将需要发送给设备的数据交付给设备进行处理。在这里，中断异常过程会描述得比较简明（直接按照对应的 case 处理）。但是对于用户来说，则需要通过系统调用这一自陷过程。详细的异常处理流程请看前一章节的内容。

首先来看设计一个外设的输入过程。设计一个驱动输入包含两个部分：驱动初始化和外来信号中断响应过程。

用图 5-1 展示系统中驱动输入的设计流程：

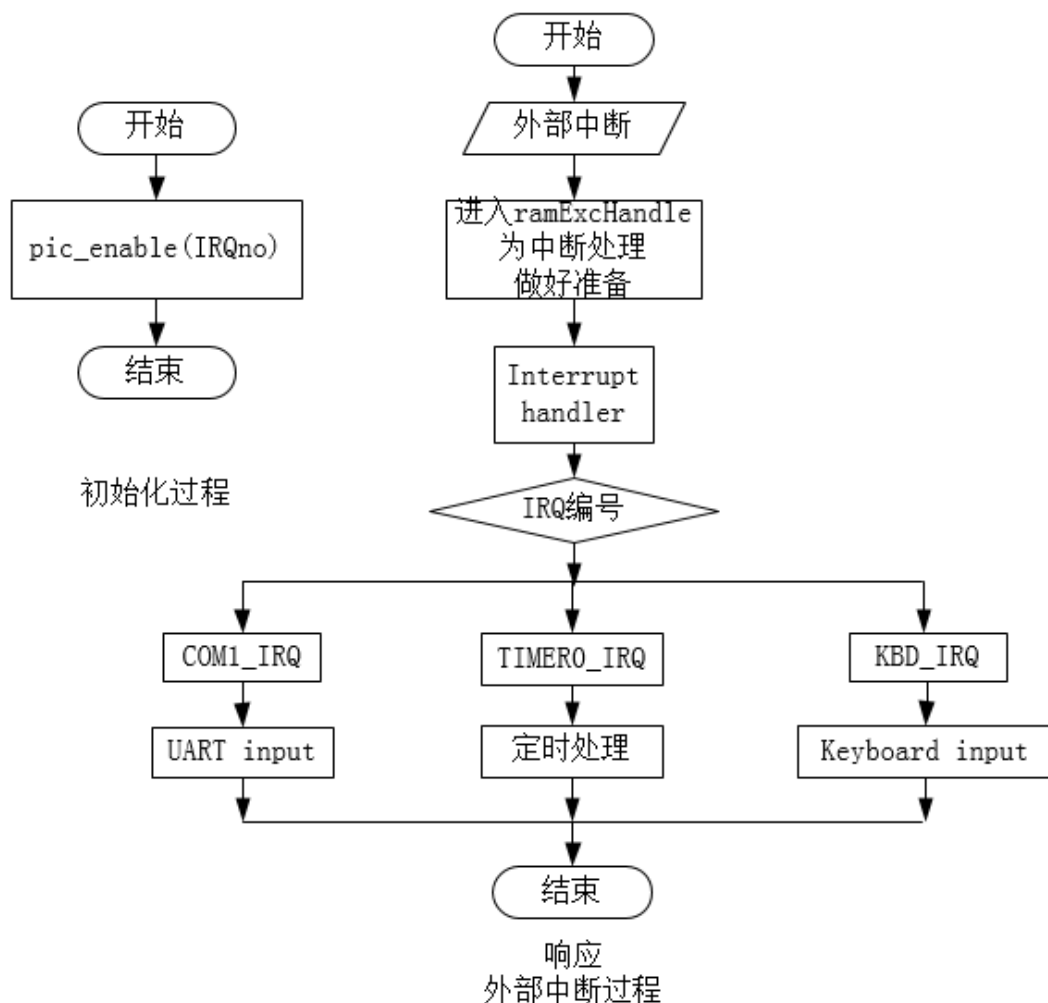


图 5-1 驱动输入流程图

从图 5-1 中可以看到，在设备初始化的过程中，会使用 `pic_enable` 和外设对应的中断向量号来让系统默认可以响应外设的中断。在操作系统运行之后，如果有外部数据到达外设的时候，就会产生该外设的中断，该中断信号会让操作系统进入 `Interrupt handler` 过程，然后根据 `IRQ` 编号即可进入到该外设的处理分支。处理结束后，便会从中断异常中退出，最终返回到中断发生前的用户程序。

然后考虑对外部设备的输出的设计。设计驱动的输出则主要考虑，分别处于内核态和用户态的程序如何通过设备输出有效信息。当然这对于内核来说十分简单，因为内核态的操作系统拥有最高的权限，它可以直接将数据写入设备即可。但是处于用户态的用户程序则不同，用户程序必须通过系统调用过程来间接使用设备，这样既可以协调多用户程序对外设的使用，又可以保证系统和用户程序的安全。

关于驱动输出的设计流程如图 5-2 所示：

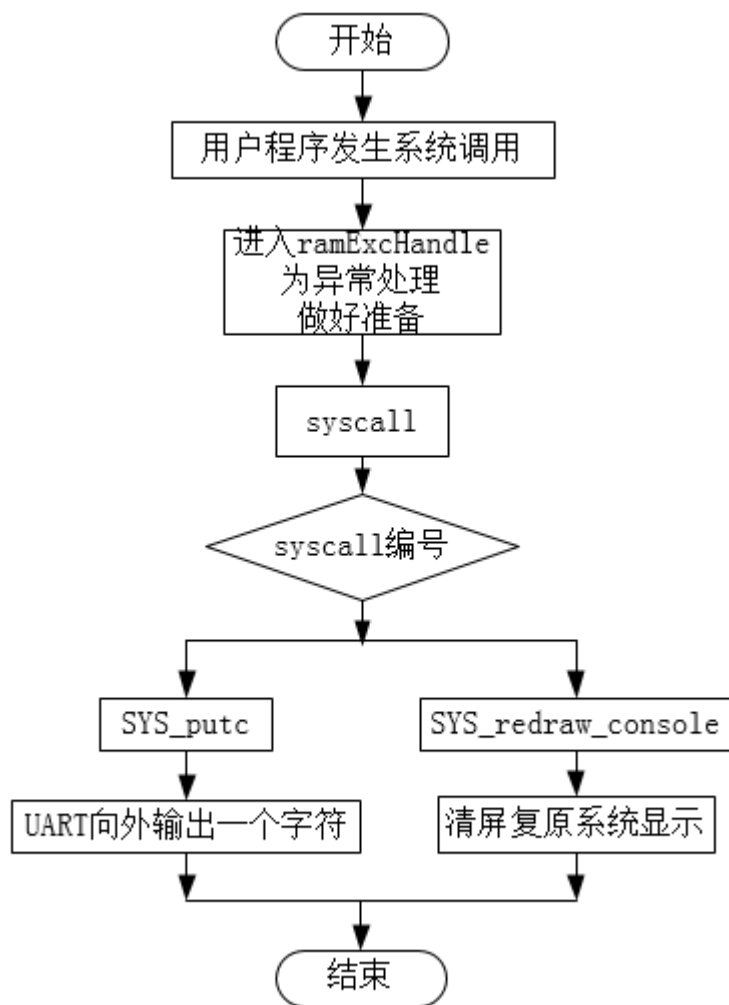


图 5-2 驱动输出的流程图

从操作系统提供的用户接口来看，用户进行系统调用一般都是从用户运行库当中调用对应的接口。在这些接口中包含了各自对应的系统调用号，使用同一的系统调用预处理接口，将系统调用号和用户传入的参数写出到约定俗称的寄存器中，然后通过 `syscall` 指令自陷入系统调用异常。

最后需要完成的就是各个设备如何处理接收到的数据（输入）和如何发送有效的数据（输出）的函数，这些函数往往都是针对该硬件具备的一些特性来完成。这些将在后面结合实例进行分析。

5.1 UART16450 驱动的设计与实现驱动设计

首先需要阅读硬件代码来获取串口相关的编址信息和中断向量号。不难发现，串口的数据寄存器的虚拟地址为 `0xbfd003f8`，控制寄存器的虚拟地址为 `0xbfd003fc`，中断向量号为 4。同时由上文的介绍知道串口的输入则是通过中断机制实现，而串

口的输出是通过轮询实现的。

首先来着手串口输入的实现。串口的输入指外界通过串口将数据传递给操作系统的过程，因为该过程对于操作系统来说是一个被动的过程，如果使用轮询的方式将会大大降低系统的效率，因此使用中断的方式来实现该过程。对应之前展示的驱动输入流程图，其具体流程如下：

1. 在初始化串口的时候，使用 `pic_enable` 和串口的中断向量号来打开系统对串口中断的响应。

2. 在 `trap` 中对中断进行处理的函数中，添加串口处理的分支并且使用中断向量号进行唯一性标记。从而在串口产生外部信号时由此进入到对从串口收到的数据进行处理函数当中。

3. 收到数据后，串口首先从数据编址处获取数据内容，然后将这个数据放入一个缓冲区中。该缓冲区是为了防止 CPU 读取数据和串口输入数据速率不一致导致的数据丢失现象。

4. 系统从缓存区中取出数据，作为来自用户的输入数据。

总体过程不太复杂，主要是先实现如何让系统响应串口的输入，然后就是通过缓冲区来防止数据的丢失。但是这里需要提到的是，在 NEXYS4 硬件设计中的 UART 为了加速，没有自带校验等功能的寄存器。考虑到和 QEMU 模拟器中的 UART 存在差异，这里使用了不同的处理算法：

算法 5.1 UART 处理输入数据的实现

输入：NULL

输出：UART 输入字符：c

```

1:    c = 0;
2:    COM_DATA = COM1 + 0x00; # 根据统一编址计算 UART 数据寄存器地址
3:    COM_SATA = COM1 + 0x04; # 根据统一编址计算 UART 状态寄存器地址
4:    read_enable = inw(COM_SATA) & 0x02; #获取 UART 的读状态
5:    if ((read_enable == 0) #判断 UART 是否为可读状态
6:        return -1; #该状态不可读
7:    c = inw(COM_DATA) & 0xFF; #可读，获取数据
8:    if (c == 127) #判断 c 是否为退格符
9:        c = '\b'; #退格符需要特殊处理，因为会将缓冲区回退一个字符
10:    return c
    
```

之后需要实现串口的输出，相比串口的输入，串口的输出过程则更加容易理解。因为此时系统是具有主动性的，即操作系统通过调用函数的方式将数据写到串口的数据寄存器上即可。但是为了使用户程序可以使用串口输出的功能，系统必须加入一个系统调用来实现这个功能。对应之前展示的驱动输出的流程图，这里通过用户的系统调用来描述其具体流程：

1.首先需要确认串口使用的系统调用号，根据一般规则，串口输出的系统调用号为 30。如果是内核中进行串口调用，则不需要通过系统调用过程，直接或间接调用串口输出函数即可。对于一般用户程序来说则需要进行系统调用，并设置对应的系统调用号（这里串口输出的系统调用号为 30）来进行调用。

2.在 trap 处理中，系统调用过程被当作是一种异常处理，其立场处理号为 8。在系统调用发生后，会根据相应传入的参数设置一些寄存器信息，然后根据对应的系统调用号来进行对应的系统调用过程（这里调用串口的输出函数），从而完成系统调用过程。

3.在串口发送数据的函数之中，首先要先循环访问控制位地址为 0xbfd003fc，从而确定串口现在是否处于忙状态。如果处于忙状态，则系统需要继续访问该位；如果处于非忙状态，则可以将需要发送的单字节数据写入到串口的数据寄存器中，完成这一次的打印内容。这里使用了不同的处理算法：

算法 5.2 UART 处理输出数据的实现

输入：UART 输出字符：c

输出：NULL

```

1:   write_enable = 0;
2:   COM_DATA = COM1 + 0x00; # 根据统一编址计算 UART 数据寄存器地址
3:   COM_SATA = COM1 + 0x04; # 根据统一编址计算 UART 状态寄存器地址
4:   while (write_enable == 0) #循环等待 UART 直到其可写
5:       write_enable = inw(COM1 + 0x04) & 0x01; # 获取串口写状态
6:   outb(COM_DATA, c & 0xFF); #将数据写入到 UART 的数据寄存器地址
7:   return NULL
    
```

至此，串口的输出过程基本上完成，相比之前这个过程更容易被理解。在实际系统操作中，串口的输出也是可以通过中断的方式来实现。但是限于硬件构造等等方面的原因，直接使用的轮询的方式实现。

5.2 PS/2 键盘驱动的设计与实现

首先确定键盘在 ucore 中为一个只读设备，由前面的设备分类知道来自键盘的数据将通过中断处理。

因为文献[24]已经给出了键盘数据和字符的关系对应表，因此具体的工作即是如何让 ucore 相应来自键盘的中断并将来自键盘的数据翻译成系统可识别的字符。

然后由于硬件中键盘需要按照协议设计硬件代码，键盘的统一编址和中断向量号，这里定义键盘在统一编址中的虚拟地址为 0xaf000000，其中断向量号为 6。这部分数据在 arch 相关的文件中声明。

在硬件设计中，参考文献[23]中描述的键盘协议，比较重要的命令有：

1. 0xFF，它用来复位键盘。
2. 0xF4，它用来启动键盘。
3. 0xFE，它用来进行数据的重新发送。

使用状态机的方式设计键盘的硬件架构，其状态机转化图设计如下：

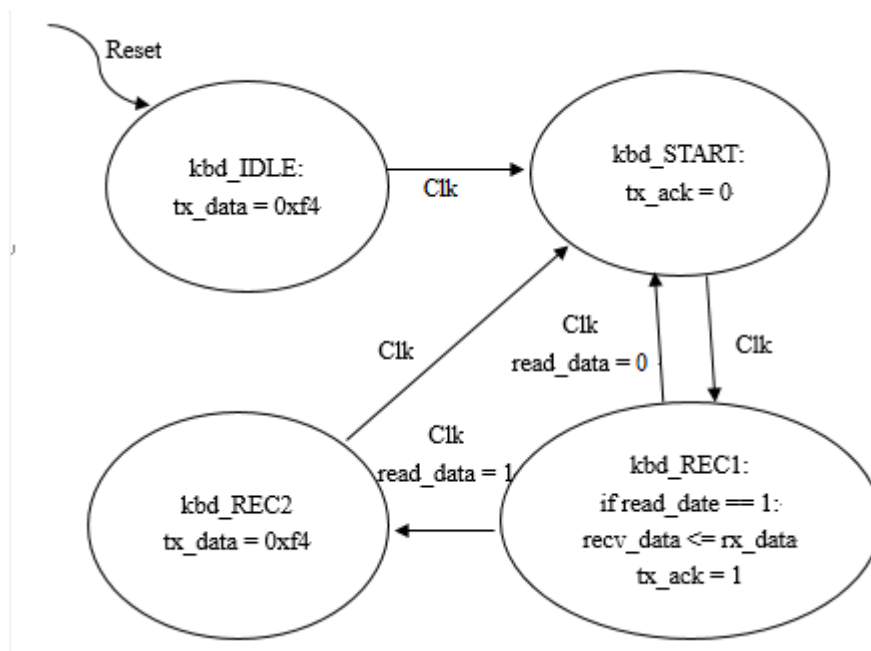


图 5-3 键盘状态机转移图

键盘驱动的软件设计上和串口输入实现处理类似的，在初始化的时候需要打开键盘的中断开关。通过给定的中断向量号和 pic_enable 函数就可以完成初始化过程。之后在 trap 中的中断处理函数中需要定义键盘中断情况的处理方式，这里则是使用了一个键盘的 handle 函数进行处理。这些步骤都和串口输入相类似。现在的重点则是键盘的 handle 函数的算法描述如下：

算法 5.3 键盘处理输入数据的实现

输入: NULL

输出: NULL

```

1:    c = *((int*)KEYBOARD); #从地址 0xaf000000 处获取来自键盘的数据 c
2:    if (c < 0 || c > 256) return; #判断该数据的合法性
3:    c = KEYCODE_MAP[c]; #通过文献[24]的关系对应表得到有效字符
4:    dev_stdin_write(c); #将字符写入系统的标准输入流
5:    return NULL

```

至此, 键盘的处理已经基本完成, 需要注意的是, 因为使用 CPU 频率远远慢于键盘扫描频率, 因此有可能需要通过两种方法来处理一次按键多现多个字符的情形:

1. 在硬件中降低键盘的时钟, 或者调整键盘控制模块的状态机。
2. 在中断处理后, 使用先关中断, 处理完后开中断的方式。

5.3 VGA 驱动设计的设计与实现

VGA 已经有硬件设计, 这里需要阅读对应 VGA 的硬件部分代码来得到相关的编址, 又因为 VGA 属于只写设备, 因为它的实现不会涉及到中断, 故而没有中断向量号。又因为使用开发板的 CPU 频率过低, 因此为了加速程序在开发板上程序运行速度, 对 VGA 驱动做了一定的改动。

类似地, 通过阅读 VGA 的硬件部分代码可以知道统一编址下其虚拟地址为 0xba000000。这里的 VGA 对应着一块固定的显存空间, 硬件部分的代码就是控制如何定时刷新 VGA, 使得其显示该部分内存的数据。

软件上将屏幕的 size 定义为 40*20。之前提到过, 为了加速 VGA 的现实过程, 实际上设置了两个大数组, 一个数组保存着系统显示的数据, 一个数组存放着当前 VGA 显示的数据。当用户程序想使用 VGA 指针时, 他可以直接通过 VGA 的统一编制地址来进行显存的数据的修改。当程序执行结束, 需要使用一个系统调用, 将原先保存系统数据的数组中的数据覆盖到当前 VGA 显示数据的数组中去。具体的过程会在测试一个简单的 VGA 应用--贪吃蛇中进行说明。

通常而言, 正常对外设的访问方式要求使用系统调用。然而由于 CPU 的频率较低, 而且 VGA 的滚屏刷新的方式比较慢。如果用户每使用一次 VGA 都需要经过系统调用, 则带来了很慢的反应速度。考虑到加速 VGA 应用程序, 因此只设计

了一个与 VGA 相关的系统调用，就是如何将用户程序使用的 VGA 状态还原到之前系统使用的状态。实现的原理之前也已经提到过，流程与串口输出类似，属于系统调用的方式，这里只说明其系统调用号为 242。

在这里简单的描述一些 VGA 的工作原理和初始化过程，并介绍其驱动的工作流程：

1.VGA 首先初始化数组信息，因为一开始没有数据，所以屏幕显示黑屏，数据内容置为-1。

2.通过 CHAR_FONT_BITMAP 表将一个字符转换成对应的可以显示在 VGA 上的具备字体的字符。这个转换字体后的数据会直接写出到显存的对应位置中，并通过硬件最终显示在屏幕上。

3.处理特殊字符和满行的情形，首先是如果遇到'\n'和'\r'字符时，此时显示需要进行换行。同样地，如果输出数据过多，超过一行的内容，也应该进行换行，并将接下来的数据接着显示。这里的换行则是通过滚屏的方式实现，和 JOS 系统设计的思想一致。

限于硬件的条件，现在的 VGA 构造上较为简单，但是基本上从原理上可以正常完成文字模式的工作。

5.4 本章总结

本章主要先对片上系统的键盘外设做了设计实现，然后通过各种外设的类型设计对应的驱动功能。对于操作系统而言，需要考虑是通过中断、或是异常处理的方式来收发外设信息。对于驱动功能而言，则需要通过外设硬件设计中使用的协议进行对应的设计。

6 系统评测

整个片上系统完成后,则需要关注这个系统的正确性和有效性。当一个操作系统部署完成后,即可开始测试一系列应用。本次上层应用测试包括了用户的 shell 程序、一些常见的 unix 命令,以及一个贪吃蛇的游戏应用。

经过测试,结果确定内核顺利启动并通过了启动过程中的一系列的 assert 函数,这些函数都从逻辑上证明了系统的正确性。通过将处理后的内核上载到 NEXYS4 开发板上,同样得到了正确的启动结果并执行了 shell 程序。之后,片上系统上还执行了其他用户程序,其执行结果同样符合预期,这充分证明该系统的有效性。

6.1 系统平台搭建与测试环境配置

表 6-1 实验平台和工具信息

项目	信息说明
环境	Windows7 64 位旗舰 + Ubuntu 12.04 (VMware)
处理器	Intel(R) Core(TM) i7-3612QM CPU @ 2.10GHz (8 CPUs)
内存	8192MB RAM
工具	Xilinx ISE Design Suite 14.7 + Adept + ImgTrans + putty + 一系列 Linux 脚本
语言	Verilog + C + Java + Shell + Makefile

主要在 Windows7 中完成硬件模块设计、内核的上载操作以及硬件实验测试。ISE 是一款使用 Verilog 语言进行硬件设计的工具,网上可以下载 ISE 的安装包,然后进行安装即可。同时需要安装的还有 Adept 工具,这个工具的功能可以将 ISE 生成的 bit 文件下载到开发板中。

当使用 adept 将由 ISE 生成的 system.bit 文件下载到 NEXYS4 开发板中,可以得到如下一个小型 PC。

表 6-2 NEXYS4 硬件生成平台信息

项目	信息说明
处理器	自制 MIPS CPU @1.56MHz (1 CPU)
内存	8MB RAM
引导固件	4KB ROM
外设	UART16450 + PS/2 键盘 + VGA 文字模式

虽然该硬件配置不高,但是运行一个操作系统已经足够。

Windows 中还需要配置 ImgTrans 工具,ImgTrans 是之前使用 Java 开发的一款 NEXYS4 开发板和实验 PC 进行通讯的工具。因为通过表 5-2 知道 EXYS4 硬件生成的平台是具有 ROM 段程序的,一般的通讯工具是无法让该开发板和 PC 进行正常通讯。ImgTrans 针对该段 ROM 程序的协议进行开发,完美的满足了通讯的需求。为了兼容多平台执行,该工具使用 RXTX 库进行串口部分的开发。RXTX 是个提供串口和并口通信的开源 Java 类库。其配置步骤如下:

1. 将 rxtxParallel.dll、rxtxSerial.dll 拷贝到: 系统盘:\WINDOWS\system32 下。
2. 如果是在开发的时候(JDK),需要把 RXTXcomm.jar、rxtxParallel.dll、rxtxSerial.dll 拷贝到..\jre...\lib\ext 下; 如: D:\Program Files\Java\jre1.6.0_02\lib\ext
3. 而且需要把项目 1.右键->2.Preperities (首选项) ->3.Java Build Path ->4.Libraries -> 5.展开 RXTXcomm.jar

此时,基本上完成 Windows 的配置流程。

然后,需要转移到 Ubuntu 系统中进行一系列配置。主要在 Ubuntu 系统中完成内核编译过程、QEMU 模拟器测试和内核预处理过程。

首先,需要配置内核编译环境,在 shell 终端输入命令:

```
> sudo apt-get install build-essential kernel-package #编译工具
```

```
> sudo apt-get install libncurses5-dev # 运行 defconfig、menuconfig 所需的库
```

然后需要配置交叉编译器的环境,这个编译器可以通过 GCC 源码进行编译,这里为了方便直接使用了 mips-sde-gcc 这一整套工具链。只需要将这套工具链的 bin 目录添加到 home 目录下的.bashrc 下即可完成路径设置。然后需要安装 QEMU 模拟器,在 shell 中输入:

```
> sudo apt-get install zlib1g-dev libglib2.0-dev #编译 QEMU 所需的 zlib 和 glib
```

```
> sudo apt-get install libtool #libtool 有利于提供 QEMU 性能
```

```
> sudo apt-get install libpixman-1-deva #选择开启 pixman 编译
```

```
> sudo apt-get install libsdl1.2-dev #执行得到图形界面
```

```
> ./configure --target-list= mipsel-softmmu #选用 mipsel 体系结构的虚拟机
```

```
> make
```

不使用 make install 将 QEMU 进行安装,而是将生成的可执行文件的路径加入到 home 目录下的.bashrc 中,这样一样可以起到和安装一样的效果。QEMU 安装

好后即可开始模拟器中的测试。当编译了可以在硬件上执行的内核外，并不能直接使用 `ImgTrans` 进行内核上载，而是先用脚本对内核进行预处理。因为 QEMU 模拟器和 NEXYS4 硬件存在差异，内核开始一段的二进制数据是用来描述生成的二进制文件内容和格式，和内核代码并没有关系。在预处理脚本中使用 `xxd` 工具将该部分的内容剔除，并使用 `tr` 工具将所有的空白字符去掉，最终得到有效的二进制内核文件。

6.2 QEMU 模拟器中系统测试

Ubuntu12.04 中的 `ucore` 目录中输入以下命令：

> `ON_FPGA=n make arch=i386 defconfig` #或者使用 `menuconfig`

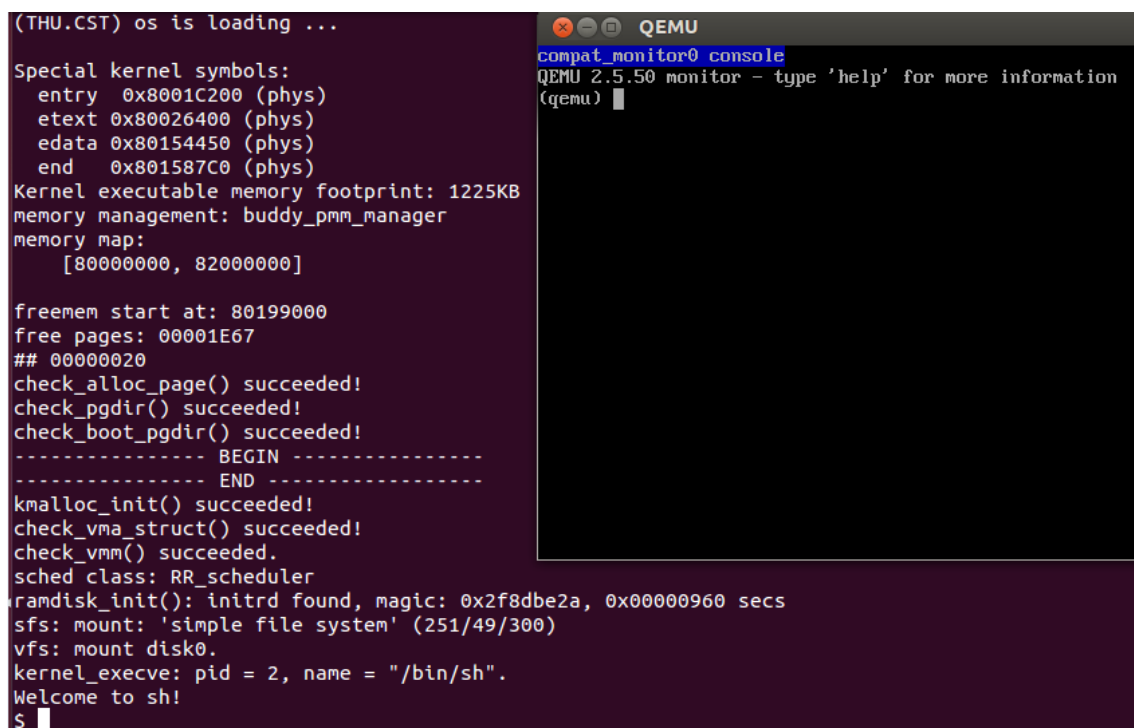
> `ON_FPGA=n make arch=mips sfsimg` # 编译磁盘

> `ON_FPGA=n make arch=mips` # 编译内核

至此，MIPS32 架构的 `ucore` 以及对应的文件系统已经完成编译，你可以使用 `ucore` 目录中自带的编译脚本简化操作过程。

然后在命令行执行 QEMU 虚拟机运行脚本：

> `./run_QEMU_mips.sh`



```
(THU.CST) os is loading ...
Special kernel symbols:
  entry 0x8001C200 (phys)
  etext 0x80026400 (phys)
  edata 0x80154450 (phys)
  end 0x801587C0 (phys)
Kernel executable memory footprint: 1225KB
memory management: buddy_pmm_manager
memory map:
[80000000, 82000000]

freemem start at: 80199000
free pages: 00001E67
## 00000020
check_alloc_page() succeeded!
check_pgdir() succeeded!
check_boot_pgdir() succeeded!
----- BEGIN -----
----- END -----
kmalloc_init() succeeded!
check_vma_struct() succeeded!
check_vmm() succeeded.
sched class: RR_scheduler
ramdisk_init(): initrd found, magic: 0x2f8dbe2a, 0x00000960 secs
sfs: mount: 'simple file system' (251/49/300)
vfs: mount disk0.
kernel_execve: pid = 2, name = "/bin/sh".
Welcome to sh!
$
```

图 6-2 QEMU-mipsel 执行 MIPS 版 kernel

6.3 NEXYS4 开发板的系统测试

1. Ubuntu12.04 中的 ucore 目录中输入以下命令:

> ON_FPGA=y make arch=mips defconfig #或者使用 menuconfig

> ON_FPGA=y make arch=mips sfsimg # 编译磁盘

> ON_FPGA=y make arch=mips # 编译内核

完成编译后, 你需要使用转换工具将内核文件进行上载预处理, 使用 tools 中 core.sh 脚本即可得到处理后的内核文件 kernelfile。

2. 搭建 MIPS32 架构的 CPU

先使用 USB 连接到计算机, 并使用 adept 将由 ISE 生成的 bit 文件下载到 NEXYS4 开发板上, 点击“program”完成 MIPS32 架构 CPU 平台的搭建。

3. 将 kernel 载入到开发板内存中

并查找到对应的 COM 口编号, 然后运行 ImgTrans 的 java 通讯工具, 根据前文的介绍和工具自带的使用说明将内核拷贝到开发板上:

> COM7 #测试 PC 上显示是 COM7

> upload kernelfile # 上载 kernel, 等待进度达到 100%, 表明内核上传成功

> run os

在进入内核后, 关闭 ImgTrans 工具并打开 putty, 以串口连接方式连接到 COM7, 即开始用户和内核的交互, 并进行应用程序的测试, 使用方法和 Unix 系统一致。

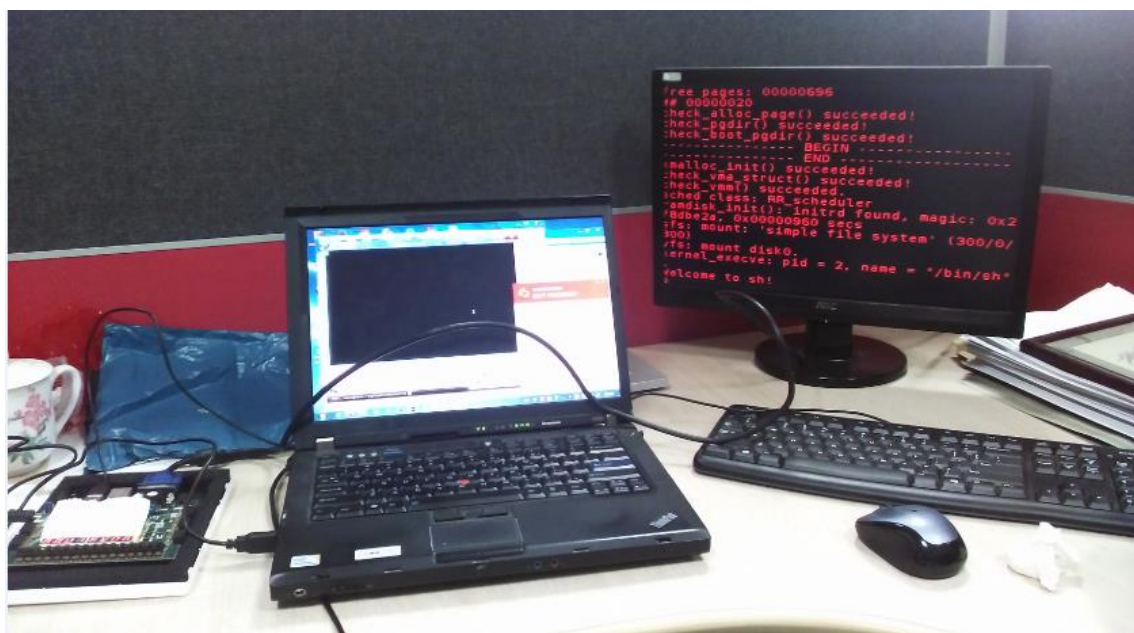


图 6-3 NEXYS4 开发板执行 kernel

6.4 上层应用测试

在操作系统中, 修改 proc.c 中的 user_main 函数, 调用 KERNEL_EXECVE 启动 shell 程序, 这样 shell 程序就成为操作系统启动后执行的第一个用户程序。这个程序用于接收来自于用户输入的指令字符串, 并对这些字符串进行解析, 最终运行这些指令, 命令执行结束后就会返回到 shell 程序中, 并重新等待用户的输入。

```
sched class: RR_scheduler
ramdisk_init(): initrd found, magic: 0x2f8dbe2a, 0x00000960 secs
sfs: mount: 'simple file system' (251/49/300)
vfs: mount disk0.
kernel_execve: pid = 2, name = "/bin/sh".
Welcome to sh!
$
```

图 6-4 kernel 中 shell 执行程序

除了正常的 shell 程序外, 还有一些常用的 Unix 命令, 如 cd、ls、cat 等等。cd 可以切换到指定目录。ls 程序则打印获取子目录和文件的相关权限信息。

在 shell 中输入命令:

\$ ls

```
$ ls
@ is [directory] 4(hlinks) 3(blocks) 1280(bytes) : @'.'
```

[d]	4(h)	3(b)	1280(s)	.
[d]	4(h)	3(b)	1280(s)	..
[d]	2(h)	7(b)	2304(s)	bin
[d]	3(h)	1(b)	768(s)	lib
[-]	1(h)	1(b)	13(s)	hello.txt

图 5-5 ls 执行程序

结束后, 依次输入命令:

\$ cd bin # 然后可以输入 ls 进行检验

```
$ cd bin
$ ls
@ is [directory] 2(hlinks) 7(blocks) 2304(bytes) : @'.'
```

[d]	2(h)	7(b)	2304(s)	.
[d]	4(h)	3(b)	1280(s)	..
[-]	1(h)	30(b)	121384(s)	num
[-]	1(h)	31(b)	124540(s)	cp
[-]	1(h)	33(b)	132015(s)	ls
[-]	1(h)	29(b)	118026(s)	echo
[-]	1(h)	32(b)	127233(s)	pwd
[-]	1(h)	34(b)	136304(s)	sh
[-]	1(h)	29(b)	118381(s)	cat

图 5-6 cd 执行程序

可以看到 ls 程序的结构和之前不同了。经过验证,该目录的显示结果的确是 bin 目录下的文件显示,这证明了 ls 和 cd 命令的正确性。

贪吃蛇游戏是一款十分经典的游戏,操作简单而且考察玩家的控制调度能力。通过控制蛇头方向吃苹果,使得蛇的身体变长,从而获得游戏积分。游戏中,玩家可以通过上下左右(w、s、a、d 四个键)控制蛇的移动方向,并且寻找苹果,每吃一个苹果就能得到一定的积分,并且使蛇的身子逐渐长,身子越长玩的难度就越大。虽然在这个游戏中没有墙的限制,但是蛇不能咬到自己的身体。一旦咬到自己的身体游戏就会结束。

这里有一点需要提到的是,作为一个应用程序是不应该直接访问到外设地址的,但是考虑到频繁的刷新会过于频繁,而 CPU 本身的频率过低,整个程序运行起来会比较卡顿。因此,考虑使用应用程序直接去写显存的方式进行,这样可以减少大量系统调用所花费的时间。但是这样会破坏系统原有的 VGA 的状态,为了避免这一情况的发生,在设计 VGA 驱动中使用了一个备份 Buffer 备份了系统的 VGA 状态,并在应用程序退出时通过系统调用,将备份 Buffer 中的数据回复到 VGA 的显存当中。

在 shell 中输入命令:

```
$ cd ..
```

```
$ snake
```

此时,系统会开始执行贪吃蛇应用,并进入游戏。

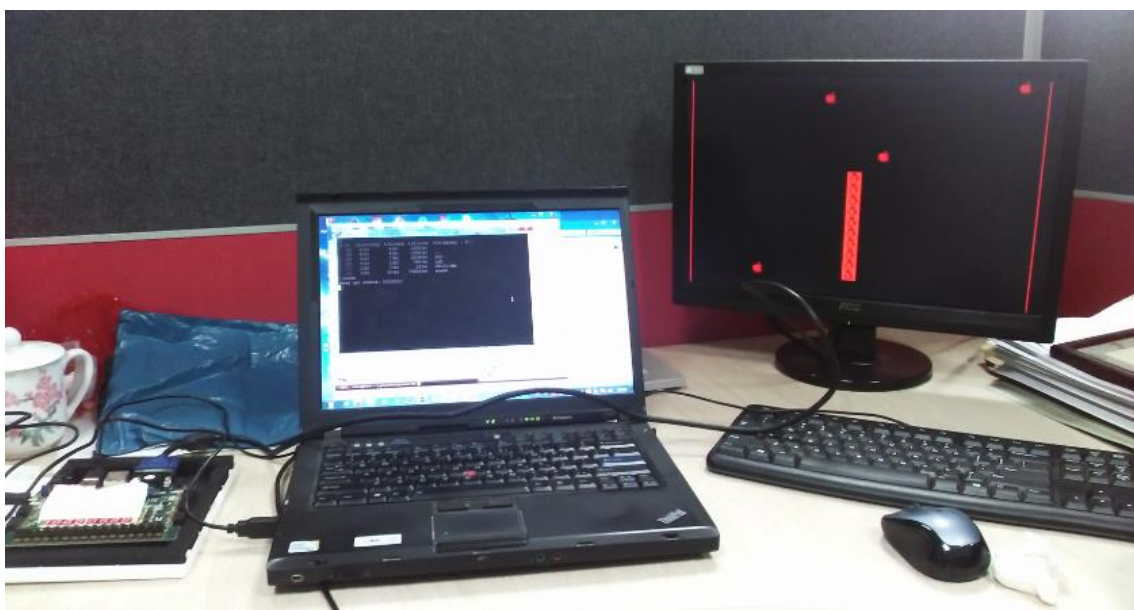


图 6-7 Retro Snaker 执行程序后

想结束游戏或者因为蛇咬到自己导致游戏结束时，按 q 键就可以退出游戏。

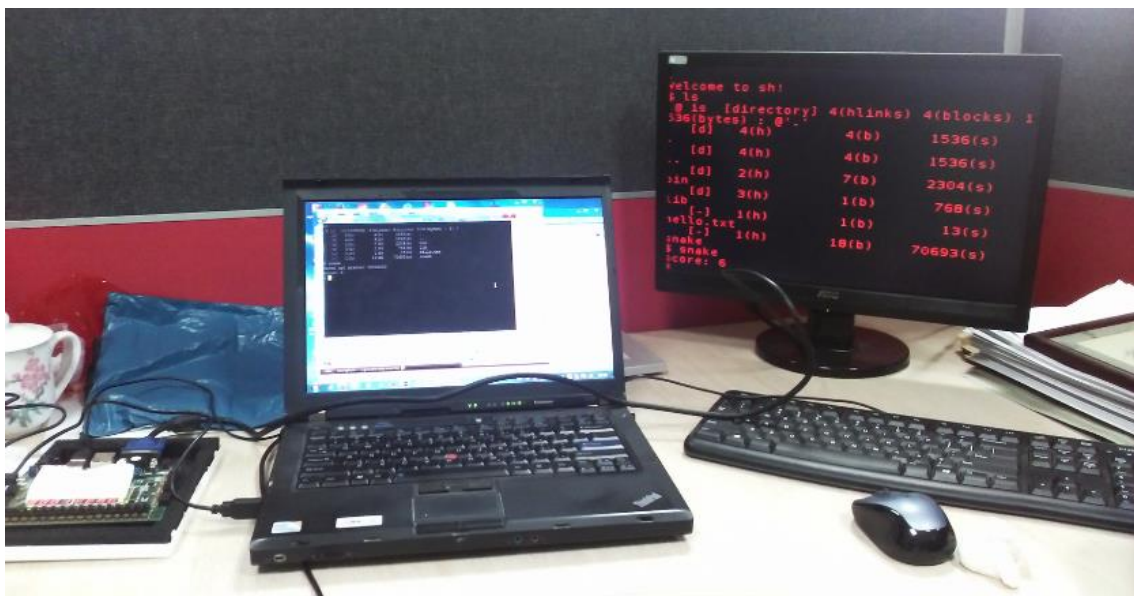


图 6-8 Retro Snaker 退出程序后

Retro Snaker 程序在退出时，会调用一个系统调用将 VGA 显示的内容恢复到用户程序使用 VGA 之前的系统状态，并继续完成剩下的输出内容。

6.5 本章总结

本节主要是配置了测试环境，并介绍了一些工具和脚本的使用，然后将测试的结果和预期进行比对。总体来说，程序执行结果符合系统设计阶段的预期，从而证明了系统的正确性与准确性。

7 总结与展望

本科毕业设计历时整个学期,在这段时间,通过严格规划了进度安排表,并定时定量完成指定的内容。设计初期,复习了之前学习过的 JOS 操作系统的设计部分和 MIPS 架构的处理器设计等内容,并参考了一系列的文献说明。在设计中期,已经基本完成了 MIPS 架构的移植内容和部分定制驱动的开发。在后期,则是完成剩下部分的驱动设计,并且解决了很多测试过程中出现的问题。

7.1 总结与分析

至此,一个基于 NEXYS4 开发板的定制操作系统基本构建完成。其中,硬件部分是基于 MIPS32 的体系结构。内核可以分为 i386、QEMU-mips、board-mips 三种方式各自进行编译,并生成对应情形下的内核文件。之后,还测试了一系列用户级别的应用程序,在实现一些基本功能的同时,更是测试了内核功能的正确性。回顾整个毕业设计过程,主要工作如下:

1. 从 CPU 的指令集角度,修正了片上系统不正确的指令,并使用高效的算法组合移位指令和加减指令来代替原有的一些低效指令。这为之后在该平台上执行更多的用户程序提供了很好的保障。

2. 从访存的角度,主要使用 FIFO 的 TLB 刷新方式代替原本的随机刷新方式。然后设计了异常处理程序来完成 TLB 缺失异常的处理。TLB 是 CPU 获得物理地址的关键硬件,该模块的实现决定了一个操作系统以及用户程序可以正常执行。

3. 从外设的角度,主要设计了键盘的硬件架构,然后为 UART16450、键盘、VGA 设计了驱动,并为 VGA 的驱动进行了一定程度的优化。这些外设的实现满足了操作系统、用户程序和用户之间的信息交互。

4. 在 Makefile 中添加了一些控制信息,用户通过设置这些信息可以方便地切换在 i386、QEMU-mips、board-mips 这三种平台上进行切换编译。这些实现都方便了测试上进行快速的切换编译。

5. 设计了针对 ROM 段程序协议的通讯工具,然后写了很多 Linux 脚本用户处理内核文件并构建响应环境。这一部分是极为重要的,因为脚本是用来对内核文件进行预处理,通讯工具是用来将处理后的内核上载到开发板上。两者缺一则会导致测试无法进行。

7.2 对未来的展望

当然,这一套系统仍存在一些不足。所以仍有望在将来对该套系统进行优化。大致方向考虑如下:

1. 从内核扩展和应用设计上考虑,通过 `Makefile` 的修改可以轻松添加一个其他的内核模块和用户应用。这为未来的工作提供了极大的方便性和可能性。比如可以通过在硬件上加入一个蓝牙串口 `pmod` 模块,然后在系统中添加一个驱动用于驱动马达,这样就可以完成一个蓝牙控制载板小车的设计,十分具备使用价值。

2. 通过软件方式实现标准的 `MIPS` 指令集,内核中留有 `EX_CPU` 这一异常号,用于处理保留指令,可以通过在这里使用软件的方式模拟那些因为低效没有纳入片上系统中的指令的实现。

3. 从片上系统优化角度来看,CPU 的频率为 `1.56MHz`。适当地优化 CPU 的性能有助于加快 CPU 的频率,从而加快程序的执行。而且外设部分驱动使用轮询的方式,从而导致本分程序执行时需要等待外设的输出结束后才能继续执行。

致谢

本次毕业设计能够顺利完成,首先要感谢我的指导老师邵志远老师。他学识渊博、治学严谨,平易近人。每当我出现疑问的时候,老师总是很及时地给出了耐心地指导。老师的建议让我学会了更多的专业知识和 Linux 使用技巧。在整个毕业设计期间,我和老师的定期反馈不但督促我按时完成计划的内容,同时也引导我解决了很多疑难问题。

然后,我还要感谢胡迪青老师。胡老师不但为我提供了基础的硬件架构,还在百忙之中抽出时间辅导我理解和修改该硬件架构。这为我按时完成整个毕业设计提高了极大的帮助。

我还要感谢答辩组的老师们。在他们的悉心指导下,我不仅得到了宝贵的建议,也学到了很多答辩技巧和表达。他们严谨的治学态度,一丝不苟的工作作风和诲人不倦的育人精神让我受益菲浅。

感谢计算机科学与技术学院所有曾经帮助过我的老师,他们给予了我精心的指导与帮助,使我获得了大量的知识,圆满完成了学业,在此我向老师们表示诚挚的敬意和衷心的感谢。

最后我要感谢我的家人和朋友,他们在背后默默地支持着我,包括物质和精神上的支持。每当我在进行研究时遇到挫折,他们都会来安慰我,鼓励我,他们是我完成毕业设计的坚强后盾。

参考文献

- [1] 潘滨. MIPS 4Kc CPU IP 核及其相关 SOC 的研究与设计,《西安科技大学》2014 年
- [2] 彭丹. 一种基于 MIPS 核的 32 位 SOC 的设计与实现,《武汉邮电科学研究院》, 2012
- [3] L Liu, B Zheng, H Shi. Design of 32-Bit RISC Microprocessor Based on FPGA. 《Journal of Data Acquisition & Processing》, 2011
- [4] RM Kubde, DB Bhoyar, RS Khedikar. Design of 32 bit (MIPS) RISC PROCESSOR using FPGA.ICWET '10 Proceedings of the International Conference and Workshop on Emerging Trends in Technology, Pages 936-939
- [5] 薛勃. 32 位 MIPS 处理器研究及其软硬件建模,《上海交通大学》2007 年
- [6] Imgttec. MIPSFPGA SOC Starter Tutorial. 2015 Imgttec official guidebook. [Http s://community.imgtec.com/forums/topic/mipsfpga-soc-advanced-starter-tutorial/](https://community.imgtec.com/forums/topic/mipsfpga-soc-advanced-starter-tutorial/)
- [7] Digilent. NEXYS4 DDR FPGA Board Reference Manual. 2015 NEXYS4 开发板自带
- [8] Digilent. NEXYS4 DDR FPGA Board Schdule 2015 NEXYS4 开发板自带
- [9] Digilent. NEXYS4-DDR 彩页 A4. 2015 NEXYS4 开发板自带
- [10] MIPS 研究网站 <https://imgtec.com/mips>
- [11] MIPS 文档 <https://imgtec.com/documentation/>
- [12] MIPS 维基百科 https://en.wikipedia.org/wiki/MIPS_Technologies
- [13] 斯威特曼. MIPS 体系结构透视. 机械工业出版社, 2008
- [14] Imgttec. MIPS_Overview-brochure. 2015 Imgttec official guidebook. <https://imgtec.com/?do-download=4408>
- [15] Imgttec. Introduction to the MIPS32 Architecture. 2015 Imgttec official guide book. <https://imgtec.com/?do-download=4278>
- [16] 朱嘉. 基于 MIPS32 平台的 Linux 操作系统移植,《单片机与嵌入式系统应用》2006 年 11 期
- [17] Imgttec. MIPSFPGA Linux System. 2015 Imgttec official guidebook. <http s://imgtec.com/tools/mips-tools/linux/>

- [18] 邵志远. 32 位操作系统实践. <http://grid.hust.edu.cn/zyshao/OSEngineering.htm>
- [19] Imgtec. MIPS32 Instruction Set Quick Reference. 2015 Imgtec official guide book <https://imgtec.com/?do-download=4275>
- [20] Imgtec. The MIPS32 Instruction Set Manual. 2015 Imgtec official guidebook.
- [21] Henry S. Warren, Jr. Hacker's Delight.
- [22] 卢仕昕, 尤凯迪, 韩军, 曾晓洋. MIPS 内存管理单元的设计与实现《计算机工程》 2010 年 21 期
- [23] 林树新, 吴朝晖. Linux 键盘驱动的移植分析及实现
- [24] 网络文档 PS2 接口协议 http://wenku.baidu.com/link?url=RRiCG_LaC3nYCoTh537y15vfz3DMAVWMCVDAwOhynjKVFVGQ190BPkMgQmRebtB712fVVnQ5VYXr_KEV-anzu6DJ6QyksDGQA4oPtAE_sgWhttps://imgtec.com/?do-download=4287