计算机系统综合实验
总结报告
操作系统部分

2011011278 计14 武祥晋
(第二组 同组成员: 刘亚宁 秦同)

简介

本次计算机系统综合实验中, 要求在 ThinPad II 上实现精简的 MIPS32 指令系统, 然后移植教学操作系统 uCore 在其上运行, 并修改交叉编译器 Decaf/C0, 使之生成的汇编代码与操作系统一起编译后, 在操作系统下运行.

本次实验中我负责的是操作系统的部分, 移植操作系统课程中完成的 x86 上 uCore, 使之能在 ThinPad II 和 qemu 上运行, 同时支持 C0 编译器生成的汇编代码. 以下第 2 节简要列举 uCore Lab 1~8 的主要改动, 第 3 节对实验中核心内容作详细介绍, 第 4 节总结参考资料与工具, 第 5 节为实验收获与课程建议.

各实验主要修改

格式约定: 各 Lab 会列举 x86 版本中相对于前一个 Lab 新增的内容, 之后针对每条介绍移植过程中的改动, 其中的核心内容会在第 3 节详细介绍. **粗体**表示核心修改, *斜体*表示小的对应修改, 下划线表示直接移植, 无效果表示与前一个 Lab 相同.

Lab1

- **boot**: 删除, 详见“启动流程”.
- *libs*: 使用 cp0.h 和 mips32s.h 代替 x86.h, 在 defs.h 中定义宏 xstr, 在 stdarg.h 修正 va_end 的定义.
- **tools**: 删除 sign.c 和 vector.c. 修改 kernel.ld, 详见“启动流程”, “异常处理”.
- **Makefile**: 手动填写 GCCPREFIX 和 QEMU 的值, 修改 CFLAGS 和 LDFLAGS, 删除 bootblock 和 UCOREIMG 部分, 修改伪目标 qemu 和 debug. 详见“工具链”, “启动流程”.
- *kern/debug*: 修改 read_eip 的内联汇编, 取消 print_stackframe 的实现. 其余内容未详细阅读检查.
- **kern/driver**: 详见“硬件中断”.
- **kern/init**: 详见“启动流程”.
- kern/libs: 直接移植.
- **kern/mm**: 去除段寄存器初始化, 暂无内容. 详见“内存管理”.
- **kern/trap**: 详见“异常处理”.

Lab2

- *boot*: 删除了内存探测, 直接认定物理地址 end ~ 8M 全部可用.
- *libs*: 开关中断实现原子读写, 直接移植 list.h, 忽略了空白符的不同.
- *tools*: 删除了 boot.ld 与 kernel_nopage.ld, 删除不打算移植的 grade.sh.

- *Makefile*: 增加 kern/sync, 改进伪目标 qemu. debug, gdb.
- kern/debug: 没有增加多余的头文件和函数声明.
- kern/driver: 移植串口读写前后关开中断.
- kern/init: 无需重新设置段寄存器.
- kern/libs: 忽略了空白符的不同.
- *kern/mm*: 去掉对内存探测结果的读取, 修改内存布局常量, 其他直接移植.
- kern/trap: 忽略了空白符的不同.
- *kern/sync*: eflags 的 IF 位改为 Status 的 IE 位.

Lab3

- boot: 已删除.
- libs: 直接移植 rand.c 和 stdlib.h.
- tools: 无修改.
- *Makefile*: 增加 kern/fs, 不使用 swap.img.
- kern/debug: 无修改.
- **kern/driver**: 删除了 ide 的实现. 详见“硬盘与 Flash”.
- kern/init: 增加各初始化函数.
- kern/libs: 无修改.
- **kern/mm**: 增加了 TLB 缺失处理. 详见“内存管理”.
- *kern/trap*: 对缺页异常的处理改为 TLB 缺失处理.
- kern/sync: 无修改.
- **kern/fs**: 删除 fs.h, 修改 swapfs 实现. 详见“硬盘与 Flash”.

Lab4

- boot: 已删除.
- libs: 直接移植 hash.c 和 stdlib.h.
- tools: 无修改.
- Makefile: 增加 kern/process, kern/schedule.
- kern/debug: 无修改.
- kern/driver: 无修改.
- kern/init: 增加 proc_init, cpu_idle.
- kern/libs: 无修改.
- kern/mm: 直接移植 kmalloc.
- *kern/trap*: 增加 forkrets. 修正了异常处理返回前的栈切换.
- kern/sync: 无修改.
- kern/fs: 无修改.
- **kern/process**: 修改寄存器使用, 详见“进程与系统调用”.
- kern/schedule: 直接移植.

Lab5

- boot: 已删除.
- libs: 直接移植 unistd.h 等.
- *tools*: 在 kernel.ld 中加入专用 binary 段, 要求放入的二进制用户程序必须 4 字节对齐. 直接移植 user.ld.
- Makefile: 直接移植用户态程序部分.
- kern/debug: 直接移植用户态程序部分.
- kern/driver: 无修改.

- kern/init: 无修改.
- kern/libs: 无修改.
- *kern/mm*: 用户栈从 0x80000000 向下, 其余直接移植.
- kern/trap: 增加 SYSCALL 处理. 增加用户态.
- *kern/sync*: 直接移植. 但由于与 atomic.h 相互包含问题, 公共部分放在 sync_base.h.
- kern/fs: 无修改.
- **kern/process**: 修改寄存器使用, 详见“进程与系统调用”.
- kern/schedule: 直接移植.
- *kern/syscall*: 修改寄存器使用, 详见“进程与系统调用”.
- **user/libs**: 修改寄存器使用, 增加 C0 需要的内置函数库. 详见“进程与系统调用”.

Lab6

- boot: 已删除.
- libs: 直接移植 skew_heap.h 等.
- tools: 无修改.
- Makefile: 无修改
- kern/debug: 无修改
- kern/driver: 无修改.
- kern/init: 增加 sched_init.
- kern/libs: 无修改.
- kern/mm: 无修改.
- kern/trap: 增加 run_timer_list.
- kern/sync: 无修改.
- kern/fs: 无修改.
- kern/process: 直接移植.
- kern/schedule: 直接移植.
- *kern/syscall*: 对应修改.
- *user/libs*: 对应修改.

Lab7

- boot: 已删除.
- libs: 从 atomic.h 移除 test_and_set_bit 等.
- tools: 无修改.
- Makefile: 无修改
- kern/debug: 无修改
- kern/driver: 无修改.
- kern/init: 无修改.
- kern/libs: 无修改.
- kern/mm: sem 代替 lock.
- kern/trap: 无修改.
- kern/sync: 直接移植.
- kern/fs: 无修改.
- kern/process: 直接移植.
- kern/schedule: 直接移植.
- *kern/syscall*: 对应修改.
- *user/libs*: 对应修改.

Lab8

- boot: 已删除.
- libs: 直接移植.
- tools: 直接移植.
- *Makefile*: 更改 swap.img 大小为 4M, 增加 kern/fs 各子目录.
- kern/debug: 无修改
- kern/driver: 获得系统时间.
- kern/init: 增加 fs_init.
- **kern/libs**: 增加除法, 其他直接移植. 详见“除法模拟”.
- kern/mm: 直接移植.
- **kern/trap**: 修改串口中断, 模拟除法指令. 详见“除法模拟”.
- kern/sync: 无修改.
- kern/fs: 直接移植.
- kern/process: 直接移植.
- kern/schedule: 直接移植.
- *kern/syscall*: 对应修改.
- *user/libs*: 对应修改.
- kern/fs/swap: 原 kern/fs/swapfs.
- kern/fs/vfs: 直接移植.
- **kern/fs/devs**: 详见“硬盘与 Flash”.
- kern/fs/sfs: 直接移植.

核心内容介绍

工具链

实验用到主要工具为 mips-linux-gnu-gcc 和 qemu-system-mipsel, 版本如下:

- gcc: Sourcery CodeBench Lite 2014.05-27
- qemu: Debian 2.0.0+dfsg-2ubuntu1.1

针对目标平台 4Kc, 需要的参数如下:

- CFLAGS += -EL -march=4kc # 原为 -m32
- LDFLAGS += -EL -m elf32ltsmip # 原为 -m elf_i386
- QEMUFLAGS += -cpu 4Kc -M malta -m 8M -nographic -kernel bin/kernel

启动流程

根据我们选择的模拟工具 qemu, 它的 MIPS 版本并没有 MBR 的设计, 而是要求传入参数 -kernel, 其为 ELF 格式, 使用 kseg0 地址段. qemu 启动后, 会从 0xbfc00000 执行一段 BIOS, 然后跳转到 ELF 的入口地址. 其中隐含了加载 ELF 的过程.

为了与之兼容, 我们直接使用 uCore 生成的 bin/kernel, 因此删去了 boot, tools/sign.c, bootblock, UCOREIMG. 同样的, 在 CPU 的 ROM 中写入一段初始化代码, 主要作用是从 Flash 偏移 0x0 的位置加载一个 ELF 文件, 然后跳转到其入口地址. 这段代码见与 ucore 同级的 bios 文件夹, 主要来自 boot/bootmain.c.

因为缺少了 bootloader, kernel 此时不能依赖 $sp 的值, 即不能使用栈. 于是在进入 C 语言的 kern_init 之前要先执行 init/entry.S. 相比 x86 中设置各段寄存器, 这段代码只需设置 $sp. 其在 lab1 中也需要, 名为 init/boot.S.

硬件中断

这一部分主要在 kern/driver 中. 首先 intr 开关中断的内联汇编修改为对 Status 的 IE 置位, picirq 针对每个设备的开关中断修改为 Status 的 IM 置位.

时钟中断比 x86 简化很多, 但额外实现了 clock_intr 函数. 标准 MIPS 中, 每次时钟中断后要读取 Count, 加上时间间隔后存回 Compare. 这一过程在 clock_intr 实现, 需要在 trap 中调用.

我们的硬件实现中, 时钟中断后 Count 会清零, 所以此时直接设置 Compare 即可设置中断间隔.

由于对标准串口了解不足, 不懂在 qemu 中如何初始化串口以及设置中断, 仅在跟踪 mmon 的过程中发现向 0xb80003f8 写入可以输出, 于是 qemu 版本不支持串口中断, 也就不能从串口读入数据. 此后找到了串口编程方法, 但是没有付诸实践.

我们的硬件实现中, 串口数据地址在 0xbfd003f8, 串口控制地址在 0xbfd003fc. 仅当控制最低位为 1 时可写, 次低位为 1 时有数据可读. 无需软件做初始化设置.

去除了 uCore 中原有的并口, CGA, 键盘的支持.

异常处理

与 x86 的向量化中断不同, MIPS 意识到在实践中它们都会跳转到统一的入口, 因此 4Kc 约定一般的异常处理入口地址都是 0x80000180. 但是允许通过 Status 的 BEV 设置其到 kseg1 段, 以及 IV 设置异步中断到 0x80000200. 此外, TLB refill (不同于 TLB invalid) 的入口在 0x80000000, 以提高处理速度.

本次移植的 uCore 中, 删除了 tools/vector.c 和 kern/trap/vector.S, 且在 idt_init 中向 0x80000180 和 0x80000000 处写入一条跳转指令, 跳转到眞正的中断入口 __alltraps.

对应的, 修改了 trap.h 中 trapframe 的定义和 trapentry.S 中保存现场代码. 此外 x86 系统栈的切换由 CPU 读取 TSS 完成, 移植后用 kstack_sp 模拟 TSS. 在 lab4 之前, trapentry.S 中栈切换代码不支持多线程, 后来修正.

内存管理

与段寄存器有关的部分全部去除, 而且由于 MIPS 的约定, 使用 kseg0 也无需在页表加入 kva 的映射段. 内存布局仅是从 0xc0000000 改到了 0x80000000, 实际入口 0x80010000, 其下方为 idleproc 栈, 以及更低处的异常入口.

lab2 中设置完页表并不能使用, 因此某些测试要在 lab3 加入 TLB 后才能通过. TLB 的实现集中在 kern/mm/tlb.{c,h} 中, 首先提供与 cr3 类似的变量 cur_pgdir, 发现 TLB 缺失首先查找页表, 没有则触发并处理页缺失. 之后尝试加入 TLB.

加入时, 首先要查看与当前页配对的页是否已在 TLB 中, 若在则更新这一项, 否则通过 tlb_get_empty 获得一个 TLB 空位置写入. 通过 tlb_get_empty, tlb_init_callback, tlb_half_add_callback, tlb_half_del_callback 以及 tlb_del_callback, 可以实现不同的 TLB 替换算法.

修改页表后要更新 TLB, 在 x86 中由一句硬件指令完成, MIPS 中也要分为整条删除和半条删除分别考虑. 值得一提的是, 在 qemu 的模拟中, 仅将有效位设为 0 后若在 TLB 其他位置写入有效条目, 仍有可能触发 TLB invalid. 因此要写入不可能的值 ENTRYHI_INVALID.

为了维护 TLB, 在页表项中利用第 11 ~ 8 位记录了该页表项在 TLB 中位置, 第 7 位标志该页是否在 TLB 中.

硬盘与 Flash

原本的 uCore 中使用了 3 块硬盘, 分别用于 kernel, swap, disk0. 而 ThinPad II 上的非易失存储是 Flash. 因此对这三部分内容分别作了以下变通.

0x0 处直接存放 ELF 格式的 kernel, 详见“启动流程”.

原计划 0x400000 ~ 0x800000 用作 swap. 同时考虑到了 Flash 的擦除特性. uCore 本身对置换位置的设计不太完善, 直接使用逻辑页号计算得到扇区号, 因此要求硬盘足够大, 且不支持多进程. 对于 Flash, 如果要求写入特定位置, 则必须先读入所有数据, 擦除后再写回. 但页替换本就发生在内存不足时, 无法读入数据. (1 个 Flash 块对应 32 个页)

目前在 swapfs 中实现了以下策略原型: 永远保留一块擦好的块 A, 当决定写入块 B 时, 写在块 A 对应偏移处, 并拷贝块 B 其余数据, 对应修改页表保证将来可读出. 最后擦除块 B. 这一策略要配合 Flash 空间分配算法, 就像物理页分配一样. 要实现内容较多而且超出了此次实验要求, 因此未实现. swapfs 的读写不做任何事.

实际测试结果显示, disk0 的 SFS 需要至少 4M 的空间. 由于不再需要 swap, 我们把它放在了 0x300000 ~ 0x700000 的位置. 但是没有实现 Flash 的擦除与写功能, 因此 disk0_write 只会 warn 但没有实际写入. 在 qemu 测试时, 由于不懂怎么模拟 Flash, 因此将 disk0 以二进制形式链接进 kernel 的 data 段.

简言之, 删除了 driver/ide 内的驱动, 而分别在其上一层 swapfs 和 dev_disk0 中实现了 Flash 的读取, 写入没有实现, 也不知道怎么在 qemu 中测试. 比较遗憾.

进程与系统调用

这一部分在原理上 x86 与 MIPS 差别不大, 只是寄存器的使用不同. 具体体现在如下方面.

kernel_thread_entry 的参数 fn 和 arg, x86 在 ebx 和 edx, 改为 a1 和 a0. 相应的, 调用 do_exit 传参也从压 eax 入栈变为移动 v0 到 a0.

进程的 cr3 由 pgdir 代替, tss 由 kstack_sp 代替.

trapframe 中, eip 由 EPC 代替, 无需段寄存器, 但要保留 gp, 在 Status 设好权限. 返回值 eax 由 v0 代替, 用户栈顶 esp 由 sp 代替, 开中断从 eflags 转为 Status.

context 中, 仅保存被调用者保存的寄存器(s0 ~ s8, gp), eip 由 ra 代替, esp 由 sp 代替.

系统调用中, 用 v0 传递调用号, a0 ~ a3 为 0 ~ 3 个参数, v1 为第 4 个参数, 返回值放在 v0. MIPS 版 gcc 的内联汇编不能像 x86 一样指定每一个寄存器的分配, 因此要实现在单独的汇编文件中. 此外, MIPS 中 EPC 指向的是 SYSCALL 指令, 处理前要加 4.

此外, 用户程序的入口汇编代码也做了对应修改. 为了支持 C0 编译器, 在 ulib 之外实现了内置库 intrinsic, 对应于 C0 中 machdesc/Intrinsic.java 中的函数.

除法模拟

对于 CPU 而言, 实现了 48 条指令, 没有除法. 对于 C0 编译器而言, 有 49 条指令可用, math 测例中包含了除法指令 div. 这一效果是在 uCore 的异常处理中实现的, 原理与系统调用类似.

CPU 发现未知指令后, 会抛出 RI 异常. uCore 读取 EPC 处指令译码, 并可从 trapframe 取得寄存器的值. 计算完成后将结果写回 trapframe 对应寄存器, 就好像指令执行成功一样.

未知指令避免

由于 CPU 支持指令有限, 要避免 uCore 以及 C0 结果用到未知指令. 具体而言, 首先要检测到未知指令, 然后进行变通. 检测过程如下:

```
objcopy -j .text -O binary a.elf a.text     # 提取 text 段到 a.text
python disas_r1.py a.text | grep unknown    # 查看未知指令偏移和编码
objdump -D -b binary -mmips a.text | less   # 对照寻找未知指令
```

确定未知指令位置与语义后, 就可进行针对性的替换. uCore 移植中主要遇到三类: 乘除指令, 条件执行指令, 半字写指令. 在 48 条指令中, 只有有符号数乘法, 半字读. 三者在 ucore 中出现分别是由于除法与 64 位乘法, 三元运算符“?:”的使用, uint16_t 类型变量. 很容易避免.

以上 disas_r1.py 为自行实现的小工具.

ThinPad II 与 qemu 双版本

经过 lab1 与 lab2 实践, 二者仅有 clock.c 和 console.c 实现不同, 因此从 lab3 开始加入了 clock_thin_c 和 clock_qemu_c 这样的替换文件, 以及 to_thin 和 to_qemu 脚本.

git 中保留的全部是 qemu 版本. 直到 lab8, 在 qemu 版本之后又多了一个 ThinPad II 最终版的提交.

综上, 当前提交的版本中, lab8 为最终片上版本, 可回退到 qemu 版本但无串口中断; lab3 ~ lab7 为 qemu 版本, 可用 to_thin 与 to_qemu 切换, 但可能存在在之后 lab 中修复过的隐患; lab1 与 lab2 为 ThinPad II 版本, 可用 git 回退到 qemu 版本.

uCore BUGS

lab3 的 check_swap 中有 7 个页没有被归还. 测试代码首先申请了 7 个页的空间, 1 个为二级页表, 1 用于 vma, 1 个用于 mm, 4 个用于页替换测试. 然后使用 nr_free_store 和 free_list_store 保存了当前状态. 测试结束后, 先释放了除页表外的 6 个页, 然后才从 nf_free_store 和 free_list_store 恢复状态, 回到缺少 7 个页的状态, 最后的检查 assert(count == 0 && total == 0) 被注释. 直到 lab8 也只是在恢复状态前多归还了页表. 理应先恢复状态, 再释放这 7 个页.

user/ls.c 的 getmode 中将常规文件标记为 -, 但 ls 函数解析时期望常规文件为 0, 导致输出为未知类型。任意修改一处使二者一致即可.

check_pgfault 没有给 vma 读权限但进行了读操作, 而 check_swap 同时给了 vma 读写权限. 幸而 check_pgfault 中 do_pgfault 由写操作触发, 否则引起错误.

lab3/kern/mm/pmm.c 的 kmalloc/kfree 中 1024*0124 疑为 1024*1024, 不过从 lab4 开始这两个函数被重新实现.

资料与工具

- https://sourcery.mentor.com/GNUToolchain/release2791

  x86 下生成 MIPS 的 gcc, 免去自己编译 gcc

- http://www.linux-mips.org/wiki/QEMU

  QEMU 的 MIPS 模拟情况, 缺少官方手册, 这个相对详细一点.

- http://www.linux-mips.org/wiki/QEMU

  QEMU 模拟的是 Malta 上的 4Kc, 课程提供了 4Kc 的手册, 这个是 Malta 手册.

- https://sourceware.org/binutils/docs-2.15/ld/Scripts.html#Scripts
  http://www.ibiblio.org/gferg/ldp/GCC-Inline-Assembly-HOWTO.html
  https://www.gnu.org/software/make/manual/html_node/Quick-Reference.html
  http://oreilly.com/catalog/make3/book/ch12.pdf

  分别是 ldscript, inline-asm, makefile 的手册, 以及一份 makefile 帮助.

- http://en.wikipedia.org/wiki/Minicom

  PC 上的终端, 用于与 ThinPad II 串口通信.

- http://www.hackersdelight.org/divcMore.pdf

  cprintf 输出十进制数时需要除以 10, 这是一种高速实现.

- http://retired.beyondlogic.org/serial/serial.htm

  标准串口编程方法, 比 ThinPad II 实现的串口复杂, 供 qemu 版参考.

- http://elinux.org/images/0/07/Intricacies_of_a_MIPS_Stack_Backtrace_Implementation.pdf
  http://elinux.org/images/6/68/ELC2008_-_Back-tracing_in_MIPS-based_Linux_Systems.pdf

  MIPS 中栈帧约束不像 x86 严格, 用于实现 backtrace 的参考资料.

- disas_r1.py

  查找对二进制指令文件中的未知指令, 可进一步修改成为反汇编器.

收获与建议

通过本次实验, 阅读并学习了许多 uCore 中之前忽略的内容, 更全面地了解了一个操作系统核心的组成. 并且, 在与下层 ThinPad II 以及上层 C0 编译器的对接中, 更细致地了解了 MIPS 体系架构.

本次实验中, 大量时间都花在了了解 qemu 模拟的硬件接口, 而且最后也没有得到很好的结果, 甚至读串口和 Flash 部分都是直接在 ThinPad II 上测试通过. 期间尝试了查找资料与阅读 Linux 源码的方法. 究其原因, 是在 CPU 指令集与操作系统两层之间还缺少其它硬件的相关知识. 以往的实验只涉及 CPU 和内存, 对总线, 外设, 驱动方面了解不够细致, 只有课上讲的所谓“思路”. 希望这一部分能在相关课程有所加强.