Zero Operating System Volume One, the Kernel DRAFT VERSION 1

Tuomo Petteri Venäläinen

June 22, 2012

Contents

Part I

Preface

Chapter 1

Goals

POSIX

Zero kernel aims to implement a POSIX-compatible Unix-like system. POSIX-compliance will be provided at library level, where the base kernel's system call and other interfaces will provide the required functionality.

Software Development

As a software developer, I find it crucial that an operating system has good, modern tools for software development. I will eventually look into porting GNU and other tools such as the GNU C Compiler (GCC), the GNU Debugger (gdb), and the GNU Profiler (gprof), as well as experiment with new functionality such as graphical frontends for them.

Multimedia

Another goal of Zero is to provide good support for multimedia; both playback and production. Such functionality would include libraries and APIs for audio, video, still images, and so forth. One idea is to port software such as SDL early on to provide support for applications such as audio and video editors as well as games.

Research and Education

I see it as an important goal of the project to provide an open source system for use in research and education. The system should capture some of the original, elegant philosophy of early Unix. The kernel should be [relatively] easy to modify for targeted use. I have chosen a relatively liberal '2-clause BSD' license which projects such as the FreeBSD operating system use. I chose it over the GNU General Public License (GPL) for easier adoption to commercial projects such as smart phones.

Portability

The plan is to make the kernel portable to different platforms, and provide support for IA-32, X86-64, as well as ARM from early on.

Chapter 2

Overview

2.1 Multithread Kernel

Zero will be a multithread kernel, designed for multiprocessor and multicore systems from the ground up. Zero will have a few separate threads and processes for certain system functionality. For example, the page daemon might run as its own thread, where init will most likely be a separate process, spawning new processes as its children.

2.2 Multiuser System

Zero will be a multitasking, multiprocessing kernel. Zero shall support virtual memory as a required part of a multiuser system to provide protection and separation on per-process basis.

2.3 Networking

Even though networking is not one of the first things to do, I plan to support IPv4 and IPv6 with the BSD Unix sockets API.

2.4 User Interfaces

Zero will have both command line and graphical interfaces. The graphical interface is going to be a network-enabled, event-based system; I will probably support the X Window System as well to let us run the myriad of existing applications [also known as clients] in circulation.

2.5 C Library

The kernel will be distributed with other software such as a C library with support for functionality from standards like ISO C, POSIX, X/Open and other relevant standards.

At the time of writing this, some machine dependent parts of the C library such as setjmp(), longjmp(), and alloca() are implemented for IA-32, X86-64, and ARM. This choice of platforms reflects most modern desktop and embedded systems, with plans to make it easy to port the kernel and accompanying software to other platforms as well.

2.6 Zero Library

Zero implements functionality for both the kernel and user environment in a library called Zero. Currently, there's support for mutexes using atomic compare and swap; these mutexes tend to be faster than those implemented as a part of the POSIX Thread Library (pthread). I will see about implementing the ISO C11 API for multithread functionality.

Part II

Zero Kernel

Chapter 3

Build Environment

3.1 System Parameters

Some system parameters are declared for the programmers through <**zero/-param.h**>. These declarations include sizes of certain types, pages, and cache-lines. The following is the file <**zero/ia32/param.h**> which gets included on IA-32 systems.

<zero/param.h>

```
#ifndef __ZERO_IA32_PARAM_H__
#define __ZERO_IA32_PARAM_H__

#define CHARSIZE     1
#define SHORTSIZE    2
#define INTSIZE      4
#define LONGSIZE     4
#define LONGSIZELOG2 2
#define LONGLONGSIZE 8
#define PTRSIZE      4
#define PTRBITS      32
#define PAGESIZELOG2 12

#define CLSIZE       32
#define PAGESIZE     (1L << PAGESIZELOG2)

#endif /* __ZERO_IA32_PARAM_H__ */
```

3.2 Compiler Attributes

<zero/cdecl.h> declares attributes specific to the C compiler in use. Some of these attributes are specified in ways of both ISO C standards as well as compiler-specific ones which often predate standardisation by the ISO C Committee. The following is the file **<zero/cdecl.h>**.

<zero/cdecl.h>

```
#ifndef __ZERO_CDECL_H__
#define __ZERO_CDECL_H__

/* size for 'empty' array (placeholder) */
#if defined(__STDC_VERSION__) && (__STDC_VERSION__ >= 199901L)
#   define EMPTY
#else
#   define EMPTY   0
#endif

/* align variables, aggregates, and tables to boundary of a */
#define ALIGNED(a)  __attribute__ ((__aligned__(a)))
/* pack aggregate fields and table items */
#define PACK()      __attribute__ ((__packed__))
/*
 * AMD64 passes first six arguments in rdi, rsi, rdx, rcx, r8, and r9; the rest
 * are pushed to stack in reverse order
 *
 * IA-32 can pass up to 3 register arguments in eax, edx, and ecx
 */
#define REGARGS(n)  __attribute__ ((regparm(n)))
#if defined(__i386__) || defined(__i486__) || defined(__i586__) || defined(__i686__)
#define FASTCALL    REGARGS(3)
/* pass all arguments on stack for assembly-link */
#define ASMLINK     __attribute__ ((regparm(0)))
#endif
#define NORET       __attribute__ ((noreturn))

#define likely(x)   __builtin_expect(!!(x), 1)
#define unlikely(x) __builtin_expect(!!(x), 0)

#endif /* __ZERO_CDECL_H__ */
```

Chapter 4

Bootstrap

4.1 GRUB Support

Multiboot

Zero uses the Grand Unified Boot Loader (GRUB) to start the system. Using GRUB requires a multiboot header; the following is the file **kern/ia32/boot.h** which shows the multiboot header and parameters in C syntax (see **struct mboothdr**).

kern/ia32/boot.h

```
#ifndef ___IA32_BOOT_H__
#define ___IA32_BOOT_H__

#if !defined(__ASSEMBLY__)
#include <stdint.h>
#include <kern/conf.h>
#endif

#if !defined(__ASSEMBLY__)
/* RAM-size in bytes */
#define grubmemsz(hdr) ((1024 + (hdr)->himem) << 10)
#endif

#define MBMAGIC     0x1BADB002
#define MBPAGEALIGN (1 << 0)
#define MBMEMINFO   (1 << 1)
#define MBFLAGS     (MBPAGEALIGN | MBMEMINFO)
#define MBCHKSUM    (-(MBMAGIC + MBFLAGS))

/* flags to select fields to fill */
#define GRUBMEM  (1 << 0)     /* lomem, himem */
#define GRUBDEV  (1 << 1)     /* bootdev */
#define GRUBCMD  (1 << 2)     /* cmdline */
#define GRUBMOD  (1 << 3)     /* modcnt, modadr */
#define GRUBSYM1 (1 << 4)     /* symbols */
#define GRUBSYM2 (1 << 5)
#define GRUBMAP  (1 << 6)     /* maplen, mapadr */
#define GRUBDRV  (1 << 7)     /* drvlen, drvadr */
#define GRUBCONF (1 << 8)     /* conftab */
#define GRUBLDR  (1 << 9)     /* bootldr */
#define GRUBAPM  (1 << 10)    /* apmtab */
#define GRUBVBE  (1 << 11)    /* VBE video extensions */

#if !defined(__ASSEMBLY__)
/* header structure to use in C code */
struct mboothdr {
    int32_t  flags;
    int32_t  lomem;
    int32_t  himem;
    int32_t  bootdev;
    int32_t  cmdline;
    int32_t  modcnt;
    int32_t  modadr;
    int32_t  syms[3];
    uint32_t maplen;
    uint32_t mapadr;
    uint32_t drvlen;
    uint32_t drvadr;
    uint32_t conftab;
    uint32_t bootldr;
    uint32_t apmtab;
    uint32_t vbectlinfo;
    uint32_t vbemodeinfo;
    uint32_t vbemode;
    uint32_t vbeseg;
    uint32_t vbeofs;
    uint32_t vbelen;
};
#endif

#define KERNSTKTOP   0x00080000
#define KERNSTKSIZE  8192

/* segment IDs */
#define NULLSEG      0
#define TEXTSEG      1
#define DATASEG      2
#define TSSSEG       3
#define UTEXTSEG     4
#define UDATASEG     5
#define CPUSEG       6
#define NGDT         7

/* segment selectors */
#define NULLSEL      0x0000
#define TEXTSEL      0x0008
#define DATASEL      0x0010
#define TSSSEL       0x0018
#define UTEXTSEL     0x0020
#define UDATASEL     0x0028
#define CPUSEL       0x0030

/* page size in bytes */
#define NBPG         4096

/* CR0 control register bits */
#define CR0PE        0x00000001
#define CR0WP        0x00010000
#define CR0PG        0x80000000

#if defined(__ASSEMBLY__)
#define SEG_EXEC     0x8
//#define SEG_GROWDOWN 0x4
#define SEG_CONFORM  0x4
#define SEG_WRITE    0x2
#define SEG_READ     0x2
#define SEG_ACCESS   0x1
#define SEG_ASM_NULL                                                        \
    .word 0, 0;                                                           \
    .byte 0, 0, 0, 0
#define SEG_ASM(type, base, lim)                                          \
    .word (((lim) >> 12) & 0xffff), ((base) & 0xffff);                    \
    .byte (((base) >> 16) & 0xff), (0x90 | (type)),                       \
        (0xc0 | (((lim) >> 28) & 0xf)), (((base) >> 24) & 0xff)
#endif

#endif /* ___IA32_BOOT_H__ */
```

TODO: use memory map provided by GRUB.

Bootstrap Code

The following is our bootstrap code utilising GRUB with a multiboot header.

kern/ia32/boot.S

```
#define __ASSEMBLY__ 1
#include <kern/ia32/boot.h>

/* GRUB support */
_MBMAGIC     = MBMAGIC
_MBFLAGS     = MBFLAGS
_MBCHKSUM    = MBCHKSUM

/* globals */
.globl        _start, start
.globl        kernidt, kerngdt, kernpagedir, kerniomap

.text        32

/* kernel entry */
_start:
start:
        cli                // disable interrupts
        jmp        _start2        // flush CPU pipeline

.align        4

/* MULTIBOOT header; must be in first 8 kilobytes of kernel image */
mboothdr:
.long        _MBMAGIC
.long        _MBFLAGS
.long        _MBCHKSUM

        /* kernel startup */
_start2:
        /* set kernel stack up */
        movl        $KERNSTKTOP, %esp
        movl        %esp, %ebp
        call        kmain        // call kmain

.align        NBPG

/* IDT; interrupt descriptor table; shared between processors */
kernidt:
        .space        NBPG
/* per-CPU GDTs; kernel segment descriptor tables */
kerngdt:
        .space        NBPG
/* page directory index page */
kernpagedir:
        .space        NBPG
/* kernel I/O protection bitmap */
kerniomap:
        .space        8192
```

4.2 System Initialisation

Assembly Routines

The following code is **kern/ia32/setup.S**, which implements assembly routines for system initialisation. I will refer to parts of this code later in the related subsections of this chapter.

kern/ia32/setup.S

```
#define __ASSEMBLY__ 1
#include <kern/conf.h>
#include <kern/ia32/boot.h>

.globl        gdtinit, pginit, idtset, gdtptr, idtptr
.extern        kernpagedir

.text        32

.align        4

        /*
         * initialize segmentation.
         * - load segment registers DS, ES, and FS with the DATA selector
         * - GS is per-CPU segment
         * - load stack segment register SS with STK selector
         * - set code segment register CS up with a far jump
         */
gdtinit:
        lgdt        (gdtptr)
        movw        $DATASEL, %ax
        movw        %ax, %ds
        movw        %ax, %es
        movw        %ax, %fs
        movw        %ax, %ss
        movw        $CPUSEL, %ax
        movw        %ax, %gs
        ljmp        $TEXTSEL, $_gdtret
_gdtret:
        ret

        /*
         * initialize paging.
         * - load page directory physical address into CR3
         * - enable the PG-bit in CR0
         * - finish with a near jump
         */
pginit:
        movl        $kernpagedir, %eax
        movl        %eax, %cr3
        movl        %cr0, %eax
        orl         $CR0PG, %eax
        movl         %eax, %cr0
        jmp         _pgret
_pgret:
        ret

idtset:
        lidt         (idtptr)
        ret

gdtptr:
        .short         0x0000
        .long        0x00000000
idtptr:
        .short         0x0000
        .long        0x00000000
```

4.2.1 Segmentation

Kernel Segments

IA-32 architecture requires use of segmentation. Zero implements this in a simple way, relying on paging for protection and other tasks. The base segments are defined as follows.

ID	Segment	Protection	Purpose
0	NULL	no access	NULL/zero segment required by the CPU
1	TEXT	read + execute	kernel code segment
2	DATA	read + write	kernel data segment
3	STK	read + write + grow-down	kernel stack
4	TSS	read + write	kernel task state segment
5	UTEXT	read + execute + user	user code segment
6	UDATA	read + write	user data segment
7	CPU	read + write	per-CPU data

4.2.2 Paging

Page Structures

IA-32 uses two-level page structure; a single page for page directory whose entries point to page tables, the entries of which in turn point to pages.

Recursive Paging

I use Jolitz-style recursive paging, which is covered in depth in the chapter **Memory Management**.

4.2.3 Interrupt Initialisation

Interrupt Vector

Interrupt vector, or **IDT** (interrupt descriptor table), sets up handler functions for interrupts. This table is initialised in **kern/ia32/trap.c** and activated in **kern/ia32/setup.S**.

kern/ia32/trap.c

```
#include <stdint.h>
#define __KERNEL__ 1
#include <kern/signal.h>
#include <kern/conf.h>
#include <zero/param.h>
#include <zero/types.h>
#include <kern/ia32/boot.h>
#include <kern/ia32/trap.h>
#include <kern/ia32/io.h>

extern void picinit(void);
extern void idtset(void);

extern void trapde(void);
extern void trapdb(void);
extern void trapnmi(void);
extern void trapbp(void);
extern void trapof(void);
extern void trapbr(void);
extern void trapud(void);
extern void trapnm(void);
extern void trapdf(void);
extern void trapts(void);
extern void trapnp(void);
extern void trapss(void);
extern void trapgp(void);
extern void trappf(void);
extern void trapmf(void);
extern void trapac(void);
extern void trapmc(void);
extern void trapxf(void);

extern void irqtimer0(void);
extern void irqtimer(void);
extern void irqkbd(void);
extern void irqmouse(void);
#if (SMP)
extern void irqerror(void);
extern void irqspurious(void);
#endif
```

```
extern void syscall(void);

#if (SMP)
extern volatile long   mpmultiproc;
#endif
extern uint64_t        kernidt[];
extern struct m_farptr idtptr;

#if 0
uint8_t trapsigmap[NIDT] ALIGNED(PAGESIZE) = {
    SIGFPE,     // DE
    0,          // DB
    0,          // NMI
    SIGTRAP,    // BP
    0,          // OF
    SIGBUS,     // BR
    SIGILL,     // UD
    SIGILL,     // NM
    0,          // DF
    0,          // RES1
    0,          // TS
    SIGSEGV,    // NP
    SIGSTKFLT,  // SS
    SIGSEGV,    // GP
    0,          // PF
    0,          // RES2
    SIGFPE,     // MF
    SIGBUS,     // AC
    SIGABRT,    // MC
    SIGFPE      // XF
};
#endif

//long interrbits = INTERRBITS;

void
idtinit(uint64_t *idt)
{
    trapsetgate(&idt[TRAPDE], trapde, TRAPSYS);
    trapsetgate(&idt[TRAPDB], trapdb, TRAPSYS);
    trapsetgate(&idt[TRAPNMI], trapnmi, TRAPSYS);
    trapsetgate(&idt[TRAPBP], trapbp, TRAPSYS);
    trapsetgate(&idt[TRAPOF], trapof, TRAPSYS);
    trapsetgate(&idt[TRAPBR], trapbr, TRAPSYS);
    trapsetgate(&idt[TRAPUD], trapud, TRAPSYS);
    trapsetgate(&idt[TRAPNM], trapnm, TRAPSYS);
    trapsetgate(&idt[TRAPDF], trapdf, TRAPSYS);
    trapsetgate(&idt[TRAPTS], trapts, TRAPSYS);
    trapsetgate(&idt[TRAPNP], trapnp, TRAPSYS);
    trapsetgate(&idt[TRAPSS], trapss, TRAPSYS);
    trapsetgate(&idt[TRAPGP], trapgp, TRAPSYS);
    trapsetgate(&idt[TRAPPF], trappf, TRAPSYS);
    trapsetgate(&idt[TRAPMF], trapmf, TRAPSYS);
    trapsetgate(&idt[TRAPAC], trapac, TRAPSYS);
    trapsetgate(&idt[TRAPMC], trapmc, TRAPSYS);
    trapsetgate(&idt[TRAPXF], trapxf, TRAPSYS);
    /* system call trap */
    trapsetgate(&idt[TRAPSYSCALL], syscall, TRAPUSER);
    /* IRQs */
    trapsetgate(&idt[trapirqid(IRQTIMER)], irqtimer0, TRAPUSER);
    trapsetgate(&idt[trapirqid(IRQKBD)], irqkbd, TRAPUSER);
    trapsetgate(&idt[trapirqid(IRQMOUSE)], irqmouse, TRAPUSER);
#if (SMP)
    trapsetgate(&idt[trapirqid(IRQERROR)], irqerror, TRAPUSER);
    trapsetgate(&idt[trapirqid(IRQSPURIOUS)], irqspurious, TRAPUSER);
#endif
#if 0
    trapsetgate(&idt[TRAPV86MODE], trapv86, TRAPUSER);
#endif
    /* initialize interrupts */
    idtptr.lim = NIDT * sizeof(uint64_t) - 1;
    idtptr.adr = (uint32_t)idt;
    idtset();

    return;
}

void
trapinit(void)
{
    idtinit(kernidt);
    picinit();  // initialise interrupt controllers
    /* mask timer interrupt, enable other interrupts */
    outb(0x01, 0x21);
    outb(0x00, 0xa1);
//    pitinit();  // initialise interrupt timer

    return;
}
```

idtset()

Our interrupt vector is enabled with the **idtset** function in **kern/ia32/setup.S** shown earlier in this chapter.

4.3 Multiprocessor Bootstrap

I'm working on multiprocessor scheduling and execution for my operating system project called Zero.

The following is **currently unworking code** for our IA-32 multiprocessor startup sequence. **Please help me fix this code**.

kern/ia32/mpentry.S

```
/*
 * this code relies on the caller to wait for the CPU to be marked active
 * - it's not reentrant
 *
 * mpentry is located at 0x9f000
 * GDT is located at MPGDT
 * call stack is located at MPENTRYSTK
 * processor kernel stack is located at MPSTKSIZE + cpu->apicid * MPSTKSIZE
 *
 * call stack
 * ----------
 *
 * 0x9e000        - stack top
 * 0x9dffc        - APIC ID
 * 0x9dff8        - processor kernel stack address
 * 0x9dff4        - page directory address
 */

#include <kern/conf.h>

#if (SMP)

#define __ASSEMBLY__ 1
#include <kern/ia32/boot.h>
#include <kern/ia32/mp.h>

.text        16

mpentry:
        /* disable interrupts */
        cli
        /* set data segment */
        movw        $0x9e00, %ax
        movw        %ax, %ds
        /* set up segmentation */
        .byte        0x66
        lgdt        (MPGDT)
//        .byte        0x66, 0x0f, 0x01, 0x15, 0x00, 0xe0, 0x09, 0x00
        /* switch to protected mode */
        movl        %cr0, %eax
        orl        $CR0PE, %eax
        movl        %eax, %cr0

.text        32
_mpentry32:
        /* initialise %cs */
//        .byte        0x66
        ljmp        $TEXTSEL, $(_mpflush - mpentry + 0x9f000)
_mpflush:
        /* load other segment registers */
        movw        $DATASEL, %ax
        movw        %ax, %ds
        movw        %ax, %es
        movw        %ax, %fs
        movw        %ax, %gs
        /* set the stack up */
//        movw        $STKSEL, %ax
        movw        %ax, %ss
        movl        $(MPENTRYSTK - 8), %esp
//        movl        $_mpdone, %ecx
        /* load PDBR and enable paging */
        popl        %eax                        // page directory address
        movl        %eax, %cr3
        movl        %cr0, %eax
        orl        $CR0PG, %eax
        movl        %eax, %cr0
        popl        %eax                        // kernel stack address
        popl        %ebx                        // APIC ID
        /* initialise processor kernel stack */
        movl        %eax, %esp
        /* the stack has the APIC ID as an argument */
        pushl        %ebx
        call        mpmain
//        jmp        *%ecx
//_mpdone:
//        ret

#endif /* SMP */
```

4.4 Linker Script

Physical Memory

The linker script describes the physical memory layout of our loaded kernel image. We use GNU linker script syntax to achieve this goal. The following is the linker script for the IA-32 kernel. The segments are discussed in more detail in the **Memory Management** chapter.

kern/ia32/kern.lds

```
/*
 * load low kernel at 1M physical
 * DMA buffers (8 * 128K) at 3M physical
 * locate high kernel at 3G virtual
 */

MPENTRY                = 0x9f000;
HICORE                = 1M;
DMABUF                = 4M;
DMABUFSIZE        = 4M;
PAGETAB                = 8M;
PAGESIZE        = 4K;
PAGETABSIZE        = 4M;
VIRTBASE        = 0xc0000000;

OUTPUT_FORMAT("elf32-i386")

ENTRY(_start)

SECTIONS {
    . = HICORE;

    /* identity-mapped low kernel segment */
    .boot : AT(HICORE) {
            boot.o(.text)
            boot.o(.data)
            boot.o(.bss)
        setup.o(.text)
        setup.o(.data)
        main.o(.text)
        trap.o(.text)
        trap.o(.bss)
        isr.o(.text)
        isr.o(.data)
        pic.o(.text)
        seg.o(.text)
        vm.o(.text)
        util.o(.text)
            mp.o(.text)
            mp.o(.data)
            mp.o(.bss)
        _eboot = .;
        . = ALIGN(PAGESIZE);
    }

    /* multiprocessor bootstrap */
    .mp : AT(HICORE + SIZEOF(.boot)) {
        _mpentry = .;
        mpentry.o(.text)
        _emp = .;
    }

    . = DMABUF;

    /* DMA buffers (below 16 megabytes) */
    .dma : AT(DMABUF) {
        _dmabuf = .;
        . += DMABUFSIZE;
        _edmabuf = .;
    }

    . = PAGETAB;

    /* identity-mapped set of page tables */
    .ptab : AT(PAGETAB) {
        _pagetab = .;
          . += PAGETABSIZE;
        _epagetab = .;
    }

    . = VIRTBASE;

    /* sections mapped by virtual addresses */

    /* read-only segment; code and some data */
    .text : AT(PAGETAB + SIZEOF(.ptab)) {
        _text = .;
        _textvirt = PAGETAB + SIZEOF(.ptab);
        *(.text*)
        *(.rodata*)
        *(.rodata.*)
        *(.eh*)
        _etext = _text + SIZEOF(.text);
        _etextvirt = _textvirt + SIZEOF(.text);
        . = ALIGN(PAGESIZE);
    }

    /* read-write data segment; initialised global structures */
    .data : AT(PAGETAB + SIZEOF(.ptab) + SIZEOF(.text)) {
            _data = _etext;
        _datavirt = _etextvirt;
        *(.data)
        . = ALIGN(PAGESIZE);
    }

    /* bss segment; runtime-allocated, uninitialised data */
    .bss : AT(PAGETAB + SIZEOF(.ptab) + SIZEOF(.text) + SIZEOF(.data)) {
        _bss = _data + SIZEOF(.data);
        _bssvirt = _etextvirt + SIZEOF(.text);
        *(.bss)
        *(COMMON)
        _ebss = _bss + SIZEOF(.bss);
        _ebssvirt = _bssvirt + SIZEOF(.bss);
        . = ALIGN(PAGESIZE);
    }

    _kernsize = PAGETAB + PAGETABSIZE + SIZEOF(.text) + SIZEOF(.data) + SIZEOF(.bss);
}
```

# Chapter 5

Interrupt Management

5.1 General Information

Interrupts by Nature

Interrupts are asynchronous events indicating things such as hardware and software errors as well as I/O (input and output) operations.

Hardware and Software

Interrupts can be triggered by hardware (e.g. user or device I/O request) as well as software (e.g. division by zero).

Terminology

Traditional Unix terminology calls internal interrupts, i.e. those occurring as the result of event internal to CPU, **traps**. External interrupts, i.e. requests from devices, are called **interrupts**. I chose to use more traditional PC terminology of **interrupt requests** (**IRQs**) for external interrupts. When necessary, I refer to traps and interrupt requests collectively as interrupts.

Traps vs. Interrupt Requests

A **noteworthy difference** between traps and interrupt requests is that the CPU **triggers interrupt handlers right off on traps**. However, with **interrupt requests**, their **disposition may be postponed** if other higher priority interrupts are being dealt with.

Interrupt Service Routines

Interrupts are managed by hooking special handler routines, called interrupts service routines, to them. The mechanism to do this is CPU dependent and is described for the IA-32 platform later in this chapter.

5.2 Special Interrupts

5.2.1 System Call Interrupt

Library Support

System call interrupts are software generated ones used to request the kernel for services. Zero C library shall implement a library-level interface to this kernel functionality. The system call interface is described elsewhere in this book in a chapter called **System Call Interface**.

System Call Implementation

It is noteworthy that there exist other ways of implementing system calls; these ways tend to be used to speed system call processing up. On later IA-32 implementations, this may be done with the machine instructions **SYSENTER** and **SYSEXIT**.

The 'slow' way of doing system calls with interrupts is implemented in the name of tradition as well as to support older hardware.

5.2.2 Page Fault Exception

Details

Page faults are interrupts of special interest as the means to implement kernel page daemon. A page fault is generated every time a page not in physical memory is accessed. The kernel can then do its magic. This exception pushes an error code with the fault address and a few flag bits (**system/user**, **read-/write**, and **present** flags). The kernel bases its [virtual] memory management based on this error code and its paging algorithm. Virtual memory is covered in more depth in the **Memory Management** chapter.

5.3 Interrupt List

Traps and IRQs

This section lists standard IA-32 traps, i.e. CPU exceptions as well as hardware IRQs (interrupt requests).

Note that interrupt numbers **0x14 through 0x20** are **reserved**.

Notes

- The **Page Fault** exception (interrupt) (0x0e) is of special interest to implementors of kernel-level page management
- Some interrupts push an error-code on stack, other's don't; I take care to take this in account in our interrupt handler/service routines

- The **NMI**, i.e. non maskable interrupt, is a result of a hardware failure (like a memory parity error), or watchdog timer which can be used to detect kernel lock-ups

- **Faults** leave the EIP point at the faulting instruction

- **Traps** leave the EIP point at the instruction right after the one that caused the interrupt

- **Aborts** may not set the return instruction pointer right, so should be acted on by shutting the process down

Mnemonic	Number	Class	Error Code	Explanation
DE	0x00	fault	no	Divide Error
DB	0x01	fault/trap	no	Reserved
NMI	0x02	interrupt	no	Non Maskable Interrupt
BP	0x03	trap	no	Breakpoint
OF	0x04	trap	no	Overflow
BR	0x05	fault	no	BOUND Range Exceeded
UD	0x06	fault	no	Invalid (Undefined) Opcode
NM	0x07	fault	no	No Math Coprocessor
DF	0x08	fault	0	Double Fault
RES1	0x09	fault	no	Coprocessor Segment Overrun (reserved)
TS	0x0a	fault	yes	invalid TSS (task state segment)
NP	0x0b	fault	yes	Segment Not Present
SS	0x0c	fault	yes	Stack-Segment Fault
GP	0x0d	fault	yes	General Protection
PF	0x0e	fault	yes	Page Fault
RES2	0x0f	N/A	no	Intel Reserved
MF	0x10	fault	yes	x87 FPU Floating-Point Error (Math Fault)
AC	0x11	fault	0	Alignment Check
MC	0x12	abort	no	Machine Check
XF	0x13	fault	no	SIMD Exception

5.4 Signal Map

Trap Signals

The table below describes mapping of traps to software signals to be delivered to the corresponding process.

Interrupt	Signal
DE	SIGFPE
DB	not mapped
NMI	not mapped
BP	SIGTRAP
OF	not mapped
BR	SIGBUS
UD	SIGILL
NM	SIGILL
DF	not mapped
RES1	not mapped
TS	not mapped
NP	SIGSEGV
SS	SIGSTKFLT
GP	SIGSEGV
PF	not mapped
RES2	not mapped
MF	SIGFPE
AC	SIBGUS
MC	SIGABRT
XF	SIGFPE

5.5 Interrupt Requests (IRQs)

5.5.1 IRQ Map

Interrupt controllers can be programmed to map IRQs as the system wishes; our kernel maps interrupt requests conventionally to interrupts **0x20 through 0x2f**.

Interrupt	Function	Number	Brief
0x20	Timer	0	interrupt timer
0x21	Keyboard	1	keyboard interface
0x22	Cascade	2	Tied to IRQs 8-15
0x23	COM2/COM4	3	serial ports #2 and #4
0x24	COM1/COM3	4	serial ports #1 and #3
0x25	Printer Port 2	5	parallel port #2, often soundcard
0x26	Floppy Drive	6	floppy drive interface
0x27	Printer Port 1	7	parallel port #1
0x28	Real-Time Clock	8	clock
0x29	IRQ2 Substitute	9	
0x2c	Mouse	12	PS/2 mouse interface
0x2d	FPU	13	floating point coprocessor
0x2e	IDE Channel 0	14	disk controller #1
0x2f	IDE Channel 1	15	disk controller #2

Chapter 6

Thread Scheduler

6.1 Scheduler Classes

Zero scheduler has several different scheduler classes encapsulated into the table below.

Class	Characteristis	Example
RT	real-time, fixed priority	optrec
INTACT	interactive, short CPU bursts	terminal session
BATCH	batch processing, CPU-intensive	compiler instance
IDLE	executed when system is idle	zeroing pages

Interval Tasks

In addition to these, Zero runs certain services every few timer interrupts to achieve tasks such as high-speed screen synchronisation. These interval tasks should be fast to execute not to degrade speed of other threads.

6.2 Thread Priorities

Zero supports 32 priorities per scheduler class, hence having a total of 128 run queues to pick threads to run from.

6.3 Timer Interrupts

Interval tasks run using the standard PIT (programmable interrupt timer). Per-CPU thread scheduling is done with local APIC timer interrupts.

6.3.1 PIT/8253 Timer

The following header file is a part of our driver for the 8253 timer chip (also known as the PIT, short for programmable interrupt timer).

```
#ifndef __IA32_PIT_H__
#define __IA32_PIT_H__

void pitinit(void);
void pitsleep(long msec, void (*func)(void));

/* support package for the 8253 timer chip */

#define PITHZ       (1193182L / HZ)
#define PITCMD      (PITDUALBYTE | PITWAVEGEN)

/* only use channel 0 */
#define PITCHAN0    0x40 /* data port */
#define PITCTRL     0x43 /* mode/command register */
/* modes/commands */
#define PITTERMCNT  0x00 /* interrupt on terminal count */
#define PITONESHOT  0x02 /* hardware re-triggerable one shot */
#define PITRATEGEN  0x04 /* rate generator */
#define PITWAVEGEN  0x06 /* square wave generator */
#define PITSOFTSTB  0x08 /* software triggered strobe */
#define PITHARDSTB  0x0a /* hardware triggered strobe */
#define PITLOBYTE   0x10
#define PITHIBYTE   0x20
#define PITDUALBYTE 0x30
#define PITREADBACK 0xc0

#define pitsethz(hz)                                                    \
    do {                                                                \
        long _hz = 1193182L / (hz);                                     \
                                                                        \
        outb(_hz & 0xff, PITCHAN0);                                     \
        outb(_hz >> 8, PITCHAN0);                                       \
    } while (0)

#endif /* __IA32_PIT_H__ */
```

The following code implements our PIT initialisation.

```
#include <stdint.h>
#include <zero/asm.h>
#include <kern/conf.h>
#include <kern/ia32/io.h>
#include <kern/ia32/trap.h>
#include <kern/ia32/pit.h>
#include <kern/ia32/pic.h>

extern uint64_t  kernidt[];
extern void     *irqvec[];
extern void      irqtimer(void);
volatile long    irqtimerfired;

void
pitinit(void)
{
    uint64_t *idt = kernidt;

    trapsetgate(&idt[trapirqid(IRQTIMER)], irqtimer, TRAPUSER);
    /* enable all interrupts */
    outb(0x00, PICMASK1);
    outb(0x00, PICMASK2);
    /* initialise timer */
    outb(PITCMD, PITCTRL);
    pitsethz(HZ);

    return;
}

/*
 * sleep for msec milliseconds, then call trigger func
 * only to be used before the scheduler is enabled
 */
void
pitsleep(long msec, void (*func)(void))
{
    long hz = 1000L / msec;

    /* enable timer interrupt, disable other interrupts */
    outb(~0x01, PICMASK1);
    outb(~0x00, PICMASK2);
    irqtimerfired = 0;
    irqvec[IRQTIMER] = func;
    outb(PITDUALBYTE | PITONESHOT, PITCTRL);
    pitsethz(hz);
    while (!irqtimerfired) {
        m_waitint();
    }
    /* enable all interrupts */
    outb(0x00, PICMASK1);
    outb(0x00, PICMASK2);

    return;
}
```

6.3.2 Local APIC Timers

Chapter 7

Memory Management

7.1 Segmentation

Zero mostly uses so-called 'flat memory model', where each segment maps the whole address space with access suitable for the use. The segments are set as follows.

Flat Memory Model

Segment	Base	Limit	Permissions
NULL	0x00000000	0x00000000	none
TEXT	0x00000000	0xffffffff	RX
DATA	0x00000000	0xffffffff	RW
TSS	tss #coreid	sizeof(struct tss)	RW
UTEXT	0x00000000	0xffffffff	URX
DATA	0x00000000	0xffffffff	URW
CPU	cpu #coreid	sizeof(struct cpu)	RW

Permissions

- U stands for user permission
- R stands for read permission
- W stands for write permission
- X stands for execute permission

7.2 Virtual Memory

7.2.1 Page Structures

IA-32

On IA-32 architecture, we use Jolitz-style recursive page directory. This page directory maps our 4-megabyte table of page tables, which in turn map the individual pages. This simplifies our page address calculations and so makes the pager run faster.

Recursive Page Directory

The page directory is declared in **kern/ia32/boot.S**. It's just a single page of physical memory identity-mapped at the same virtual as physical address. A single page directory entry points to the page directory itself, creating a recursion when looking for page addresses.

Table of Page Tables

The page tables are located at 8 megabytes physical, identity mapped to 8 megabytes virtual. Effectively, this table is indexed with the page directory index and page table index of addresses to fetch addresses of the corresponding physical pages.

Diagram

Below you can see a simple visualisation of our page structure hierarchy.

```
    pagedir         pagetab
    -------         -------
    -------         -------
--->| #0  |-------->| PG0 |
|   -------         -------
|     ...             ...
|     ...             ...
|     ...             ...
|   -------         -------
|---| DIR |    ---->| ADR | \
    -------    |    -------  |
      ...      |    | ... |  --- last page of pagetab
    -------    |    -------  |
    |#1023|----|    | ADR | /
    -------         -------

    pagedir
    -------
    - 4096 bytes (one page)
    - 1024 page table entries pointing to pages in pagetab

    pagetab
    -------
    - 4 megabytes
    - flat table of page pointers
```

7.2.2 Identity Maps

Identity maps are mapped with their virtual addresses equal to their physical ones.

7.2.2.1 HICORE

The HICORE segment contains code and data for early kernel bootstrap and initialisation.

7.2.2.2 MP

The MP segment contains application processor startup code for multiprocessor systems.

7.2.2.3 DMA

The DMA segment contains DMA buffers for device I/O.

7.2.2.4 PAGETAB

The PAGETAB segment contains a flat table of page tables (containing page pointers).

7.2.2.5 KERNVIRT

The KERNVIRT segment is mapped at the virtual address 0xc0000000 (3 gigabytes) and contains kernel TEXT, DATA, and BSS segments.

7.2.2.6 Kernel Map

Address	Segment	Brief
0 M	LOCORE	low 1 megabyte
MPSTKSIZE	MPSTK0	processor 0 kernel-mode stack
1 M	HICORE	early kernel bootstrap
1 M + SIZEOF(.boot)	MP	multiprocessor startup code
4 M	DMA	DMA I/O buffers
8 M	PAGETAB	table of page tables
0xc0000000	KERNVIRT	kernel TEXT, DATA, and BSS segments

7.2.3 Memory Interface

This header file declares parts of our virtual memory manager.

kern/ia32/vm.h

```
#ifndef __IA32_VM_H__
#define __IA32_VM_H__

#include <stdint.h>
#include <kern/buf.h>

#define __KERNEL__ 1
#include <zero/mtx.h>
#define vmlklruq(pq)   mtxlk(&pq->lk, 0)
#define vmunlklruq(pq) mtxunlk(&pq->lk, 0)
#define vmlkbufq()     mtxlk(&vmbufq.lk, 0)
#define vmunlkbufq()   mtxunlk(&vmbufq.lk, 0)

extern uint32_t _kernpagedir[];

void *vmmapvirt(uint32_t *pagetab, void *virt, uint32_t size, uint32_t flags);
void  vmfreephys(void *virt, uint32_t size);

#define KERNVIRTBASE      0xc0000000U
#define vmphysadr(adr)    ((uint32_t)(adr) - KERNVIRTBASE)
```

```
#define vmpagedirnum(adr) ((uint32_t)(adr) >> PDSHIFT)
#define vmpagenum(adr)    ((uint32_t)(adr) >> PTSHIFT)
#define vmpageofs(adr)    ((uint32_t)(adr) & (PAGESIZE - 1))

#define vmpageid(adr)     ((uint32_t)(adr) >> PAGESIZELOG2)
#define vmbufid(adr)      ((uint32_t)(adr) >> BUFSIZELOG2)
#define vmisbufadr(adr)   (!((uint32_t)(adr) & (BUFSIZE - 1)))

#define vmpageadr(pg, pt)                                           \
    (((pg) - (pt)) << PAGESIZELOG2)

/* internal macros */

static __inline__ void
vmflushtlb(void *mp)
{
    __asm__ __volatile__ ("invlpg (%0)\n" : : "r" (mp) : "memory");
}

/* virtual memory parameters */
#define NPAGEMAX        (NPDE * NPTE)                   // # of virtual pages
#define NPDE            1024                            // per directory
#define NPTE            1024                            // per table
//#define PAGETAB         0x00700000U                     // physical address
#define PAGETABSIZE     (NPDE * NPTE * sizeof(uint32_t))
#define PDSHIFT         22
#define PTSHIFT         12
#define VMPDMASK        0xffc00000                      // top 10 bits
#define VMPTMASK        0x003ff000                      // bits 12..21
#define VMPGMASK        0xfffff000U                     // page frame; 22 bits

/* page structure setup */

/*
 * page flags
 */
/* standard IA-32 flags */
#define PAGEPRES     0x00000001U       // present
#define PAGEWRITE    0x00000002U       // writeable
#define PAGEUSER     0x00000004U       // user-accessible
#define PAGEBUSY     0x00000020U       // has been accessed
#define PAGEDIRTY    0x00000040U       // has been written to
#define PAGESUPER    0x00000080U       // 4M page
#define PAGEGLOBAL   0x00000100U       // global
#define PAGESYS1     0x00000200U       // reserved for system
#define PAGESYS2     0x00000400U       // reserved for system
#define PAGESYS3     0x00000800U       // reserved for system
/* custom flags */
#define PAGESWAPPED  PAGESYS1        // on disk
#define PAGEBUF      PAGESYS2        // buffer cache
```

```
#define PAGEWIRED    PAGESYS3        // wired
#define PAGESYSFLAGS (PAGESWAPPED | PAGEBUF | PAGEWIRED)

/* page fault management */

/* page fault exception */
#define NPAGEDEV     16
//#define pfdev(adr)  (((adr) & PFDEVMASK) >> 3)
#define pfadr(adr)   ((adr) & PFPAGEMASK)
#define PFPRES       0x00000001U        // page is present
#define PFWRITE      0x00000002U        // write fault
#define PFUSER       0x00000004U        // user fault
#define PFFLGMASK    0x00000007U
#define PFADRMASK    0xfffffff8U
#define PFPAGEMASK   0xfffff000U

struct vmpage {
    struct vmpage *prev;
    struct vmpage *next;
};

struct vmpageq {
    long          lk;
    struct vmpage *head;
    struct vmpage *tail;
};

#define VMBUFNREFMASK 0x07
#define vmsetbufnref(bp, npg)                                         \
    ((bp)->prev = (void *)((uint32_t)((bp)->prev)                     \
                           | (((npg) & VMBUFNREFMASK) << 4)),         \
     (bp)->next = (void *)((uint32_t)((bp)->next)                     \
                           | (((npg) >> 4) & VMBUFNREFMASK)))
#define vmgetbufnref(bp)                                              \
    (((uint32_t)((bp)->prev) & VMBUFNREFMASK)                         \
     | (((uint32_t)((bp)->next) & VMBUFNREFMASK) << 4))
#define vmgetprevbuf(bp)                                              \
    ((void *)((uint32_t)((bp)->prev) & ~VMBUFNREFMASK))
#define vmgetnextbuf(bp)                                              \
    ((void *)((uint32_t)((bp)->next) & ~VMBUFNREFMASK))
struct vmbuf {
    struct vmbuf *prev;
    struct vmbuf *next;
};

struct vmbufq {
    long          lk;
    struct vmbuf *head;
    struct vmbuf *tail;
};
```

```
#define vmaddpage(adr)                                                  \
    do {                                                                \
        struct vmpageq *_pageq = &vmpagelruq;                           \
        struct vmpage  *_pg = &vmpagetab[vmpageid(adr)];                \
        struct vmpage  *_head;                                          \
                                                                        \
        _pg->prev = NULL;                                               \
        vmlklruq(_pageq);                                               \
        _head = _pageq->head;                                           \
        _pg->next = _head;                                              \
        if (_head) {                                                    \
            _head->prev = _pg;                                          \
        } else {                                                        \
            _pageq->tail = _pg;                                         \
        }                                                               \
        _pageq->head = _pg;                                             \
        vmunlklruq(_pageq);                                             \
    } while (0)

#define vmrmpage(adr)                                                   \
    do {                                                                \
        struct vmpageq *_pageq = &vmpagelruq;                           \
        struct vmpage  *_pg = &vmpagetab[vmpageid(adr)];                \
        struct vmpage  *_tmp;                                           \
                                                                        \
        vmlklruq(_pageq);                                               \
        _tmp = _pg->prev;                                               \
        if (_tmp) {                                                     \
            _tmp->next = _pg->next;                                     \
        } else {                                                        \
            _tmp = _pg->next;                                           \
            _pageq->head = _tmp;                                        \
            if (_tmp) {                                                 \
                _tmp->prev = _pg->prev;                                 \
            } else {                                                    \
                _pageq->tail = _tmp;                                    \
            }                                                           \
            _pageq->head = _tmp;                                        \
        }                                                               \
        _tmp = _pg->next;                                               \
        if (_tmp) {                                                     \
            _tmp->prev = _pg->prev;                                     \
        } else {                                                        \
            _tmp = _pg->prev;                                           \
            _pageq->tail = _tmp;                                        \
            if (_tmp) {                                                 \
                _tmp->next = NULL;                                      \
            } else {                                                    \
                _pageq->head = _pageq->tail = _tmp;                     \
            }                                                           \
        }                                                               \
        vmunlklruq(_pageq);                                             \
    } while (0)

#define vmaddbuf(adr)                                                   \
    do {                                                                \
        struct vmbufq *_bufq = &vmbufq;                                 \
        struct vmbuf  *_hdrtab = vmbuftab;                              \
        struct vmbuf  *_buf = &_hdrtab[vmbufid(adr)];                   \
        struct vmbuf  *_head;                                           \
                                                                        \
        _buf->prev = NULL;                                              \
        vmlkbufq();                                                     \
        _head = _bufq->head;                                            \
        _buf->next = _head;                                             \
        if (_head) {                                                    \
            _head->prev = _buf;                                         \
        } else {                                                        \
            _bufq->tail = _buf;                                         \
        }                                                               \
        _bufq->head = _buf;                                             \
        vmunlkbufq();                                                   \
    } while (0)

#define vmrmbuf(adr)                                                    \
    do {                                                                \
        struct vmbufq *_bufq = &vmbufq;                                 \
        struct vmbuf  *_hdrtab = vmbuftab;                              \
        struct vmbuf  *_buf = &_hdrtab[vmbufid(adr)];                   \
        struct vmbuf  *_tmp;                                            \
                                                                        \
        vmlkbufq();                                                     \
        _tmp = _buf->prev;                                              \
        if (_tmp) {                                                     \
            _tmp->next = _buf->next;                                    \
        } else {                                                        \
            _tmp = _buf->next;                                          \
            _bufq->head = _tmp;                                         \
            if (_tmp) {                                                 \
                _tmp->prev = _buf->prev;                                \
            } else {                                                    \
                _bufq->tail = _tmp;                                     \
            }                                                           \
            _bufq->head = _tmp;                                         \
        }                                                               \
        _tmp = _buf->next;                                              \
        if (_tmp) {                                                     \
            _tmp->prev = _buf->prev;                                    \
        } else {                                                        \
            _tmp = _buf->prev;                                          \
            _bufq->tail = _tmp;                                         \
            if (_tmp) {                                                 \
                _tmp->next = NULL;                                      \
            } else {                                                    \
                _bufq->head = _bufq->tail = _tmp;                       \
            }                                                           \
        }                                                               \
        vmunlkbufq();                                                   \
    } while (0)

#define vmdeqpage(rpp)                                                  \
    do {                                                                \
        struct vmpageq *_pageq = &curproc->pagelruq;                    \
        struct vmpage  *_tail;                                          \
                                                                        \
        vmlklruq(_pageq);                                               \
        _tail = _pageq->tail;                                           \
        if (_tail) {                                                    \
            if (_tail->prev) {                                          \
                _tail->prev->next = NULL;                               \
            } else {                                                    \
                _pageq->head = NULL;                                    \
            }                                                           \
            _pageq->tail = _tail->prev;                                 \
        }                                                               \
        *(rpp) = _tail;                                                 \
        vmunlklruq(_pageq);                                             \
    } while (0)

#endif /* __IA32_VM_H__ */
```

Chapter 8

I/O Operations

8.1 DMA Interface

This header file declares our DMA I/O interface.

kern/ia32/dma.h

```
#ifndef __IA32_DMA_H__
#define __IA32_DMA_H__

#include <stdint.h>
#include <kern/ia32/io.h>

#define dmagetbuf(chan) ((void *)(DMABUFBASE + (chan) * DMACHANBUFSIZE))

extern const uint8_t dmapageports[];

#define DMAIDLE         0
#define DMAREADOP       1
#define DMAWRITEOP      2

#define DMANCHAN        8
#define DMACHANBUFSIZE  (512 * 1024)
#define DMAIOBUFSIZE    65536

#define DMABUFBASE      0x00400000U
#define DMABUFSIZE      0x00400000U
//#define DMACBUFSIZE     (1U << DMACBUFSIZELOG2)
#define DMABUFNPAGE     (DMACHANBUFSIZE >> PAGESIZELOG2)
//#define DMACBUFSIZELOG2 17

/* 8237 DMA controllers */
#define DMA1BASE        0x00U
#define DMA2BASE        0xc0U
```

```
/* controller 1 registers */
#define DMA1CMD         0x08U   // command register
#define DMA1STAT        0x08U   // status register
#define DMA1REQ         0x09U   // request register
#define DMA1MASK        0x0aU   // single mask register bit
#define DMA1MODE        0x0bU   // mode register
#define DMA1CLRPTR      0x0cU   // clear LSB/MSB flip-flop
#define DMA1TEMP        0x0dU   // temporary register (not present in 82374)
#define DMA1RESET       0x0dU   // master clear/reset
#define DMA1CLRMASK     0x0eU   // clear mask register
#define DMA1MASKALL     0x0fU   // read/write all mask register bits (82374)

/* controller 2 registers */
#define DMA2CMD         0xd0U
#define DMA2STAT        0xd0U
#define DMA2REQ         0xd2U
#define DMA2MASK        0xd4U
#define DMA2MODE        0xd6U
#define DMA2CLRPTR      0xd8U
#define DMA2TEMP        0xdaU
#define DMA2RESET       0xdaU
#define DMA2CLRMASK     0xdcU
#define DMA2MASKALL     0xdeU

/* address registers */
#define DMAADR0         0x00U
#define DMAADR1         0x02U
#define DMAADR2         0x04U
#define DMAADR3         0x06U
#define DMAADR4         0xc0U
#define DMAADR5         0xc4U
#define DMAADR6         0xc8U
#define DMAADR7         0xccU

/* count registers */
#define DMACNT0         0x01U
#define DMACNT1         0x03U
#define DMACNT2         0x05U
#define DMACNT3         0x07U
#define DMACNT4         0xc2U
#define DMACNT5         0xc6U
#define DMACNT6         0xcaU
#define DMACNT7         0xceU

/* page registers for low byte (23-16) */
#define DMAPAGE0        0x87U
#define DMAPAGE1        0x83U
#define DMAPAGE2        0x81U
#define DMAPAGE3        0x82U
```

```
#define DMAPAGE5        0x8bU
#define DMAPAGE6        0x89U
#define DMAPAGE7        0x8aU
/* low byte page refresh */
#define DMAPAGEREFR     0x8fU

/* operation modes */
#define DMAREAD         0x44U
#define DMAWRITE        0x48U
#define DMAAUTOINIT     0x10U
#define DMAADRINCR      0x20U
#define DMASINGLE       0x40U
#define DMABLOCK        0x80U
#define DMACASCADE      0xc0U

#if 0
/*
 * channel structure
 * num    - channel number
 * buf    - I/O buffer
 * bufsz  - buffer size in bytes
  */
struct m_dmachan {
    long      num;
    void     *buf;
    size_t    bufsz;
};
#endif

#if 0
struct m_iodev {
    /* DMA channel number, -1 means PIO */
    long           dma;
    /*
     * iochans - device I/O channel table
     * niochan - number of channels in iochans
     */
    uint16_t      *iochans;
    long           niochan;
    /* system call interface */
    long          (*open)(char *, long, long);
    long long     (*seek)(long, long long, long);
    unsigned long (*read)(long, void *, unsigned long);
    unsigned long (*write)(long, void *, unsigned long);
}
#endif

#define DMACHANMASK    0x03
#define DMA2BIT        0x04
#define DMAMASKBIT     0x04
```

```
#define _isdma1(c)     (!(~((c) & DMACHANMASK)))

static __inline__ void
dmaunmask(uint8_t chan)
{
    if (_isdma1(chan)) {
        outb(DMACHANMASK, chan);
    } else {
        outb(DMA2MASK, chan & DMACHANMASK);
    }
}

static __inline__ void
dmamask(uint8_t chan)
{
    if (_isdma1(chan)) {
        outb(DMACHANMASK, chan | DMAMASKBIT);
    } else {
        outb(DMA2MASK, (chan & DMACHANMASK) | DMAMASKBIT);
    }
}

static __inline__ void
dmaclrptr(uint8_t chan)
{
    if (_isdma1(chan)) {
        outb(DMA1CLRPTR, 0);
    } else {
        outb(DMA2CLRPTR, 0);
    }
}

static __inline__ void
dmasetmode(uint8_t chan, uint8_t mode)
{
    if (_isdma1(chan)) {
        outb(DMA1MODE, mode | chan);
    } else {
        outb(DMA1MODE, mode | (chan & DMACHANMASK));
    }
}

static __inline__ void
dmasetpage(uint8_t chan, uint8_t page)
{
    if (_isdma1(chan)) {
        outb(dmapageports[page], chan);
    } else {
        outb(dmapageports[page], chan & 0xfe);
    }
}

static __inline__ void
dmasetadr(uint8_t chan, void *ptr)
{
    uint16_t port;
    uint32_t adr = (uint32_t)ptr;

    dmasetpage(chan, adr >> 16);
    if (_isdma1(chan)) {
        port = DMA1BASE + ((chan & DMACHANMASK) << 1);
        outb(port, adr & 0xff);
        outb(port, (adr >> 8) & 0xff);
    } else {
        port = DMA1BASE + ((chan & DMACHANMASK) << 2);
        outb(port, (adr >> 1) & 0xff);
        outb(port, (adr >> 9) & 0xff);
    }
}

/*
 * NOTE: the number of transfers is one higher than the initial count.
 */
static __inline__ void
dmasetcnt(uint8_t chan, uint32_t cnt)
{
    uint16_t port;

    cnt--;
    if (_isdma1(chan)) {
        port = DMA1BASE + ((chan & DMACHANMASK) << 1) + 1;
        outb(port, cnt & 0xff);
        outb(port, (cnt >> 8) & 0xff);
    } else {
        port = DMA1BASE + ((chan & DMACHANMASK) << 1) + 2;
        outb(port, (cnt >> 1) & 0xff);
        outb(port, (cnt >> 9) & 0xff);
    }
}

static __inline__ uint32_t
dmagetcnt(uint8_t chan)
{
    uint16_t port;
    uint32_t cnt;
    uint32_t tmp;

    if (_isdma1(chan)) {
        port = DMA1BASE + ((chan & DMACHANMASK) << 1) + 1;
    } else {
        port = DMA1BASE + ((chan & DMACHANMASK) << 1) + 2;
    }
    cnt = inb(port);
    tmp = inb(port);
    cnt++;
    cnt += tmp << 8;
    if (_isdma1(chan)) {

        return cnt;
    } else {

        return (cnt << 1);
    }
}

/* invalidate cache for buffers of chan */
static __inline__ void
dmainvlbuf(uint8_t chan)
{
    uint8_t *pg;
    long     n;

    pg = (uint8_t *)(DMABUFBASE + chan * DMACHANBUFSIZE);
    n = DMABUFNPAGE;
    while (n--) {
        vmflushtlb(pg);
        pg += PAGESIZE;
    }

    return;
}

#endif /* __IA32_DMA_H__ */
```

Chapter 9

System Calls

9.1 Process Interface

9.1.1 void halt(long flg);

The **halt** system call shuts the operating system down.

Arguments

- if flg argument is **HALT_REBOOT**, the system will be restarted

9.1.2 long exit(long val, long flg);

The **exit** system call exits a running process.

Arguments

- the val argument will be passed as return value to a shell
- if the flg argument has **EXIT_SYNC** bit set, synchronise stdin, stdout, and stderr
- if the flg argument has **EXIT_PROFRES** bit set, print statistics on resource usage

9.1.3 void abort(void);

The **abort** system call exits a process erraneously and writes out a core dump. This dump can later be used to analyse program execution with a debugger.

9.1.4 long fork(long flg);

The **fork** system call creates a copy of the running process and starts executing it.

Arguments

- if the flg argument has **FORK_COPYONWR** bit set, don't copy parent address space before pages are written on
- if the flg argument has **FORK_SHARMEM** bit set, share address space with parent (a'la Unix vfork())

9.1.5 long exec(char *file, char *argv[], char *env[]);

The **exec** system call executes **file**, passing it the runtime arguments in **argv** and environment in **env**.

Arguments

Both **argv** and **env** need to be terminated with NULL entries.

9.1.6 long throp(long cmd, long parm, void *arg);

The **throp** system call provides control of running threads, creation of new threads, and other thread functionality.

Arguments

cmd	Brief
THR_NEW	create new thread
THR_JOIN	join running thread (wait for exit)
THR_DETACH	detach running thread
THR_EXIT	exit thread
THR_MTXOP	mutex operation
THR_CLEANOP	cleanup; pop and execute cleanup functions
THR_KEYOP	create or delete thread
THR_CONDOP	condition operations; signal, broadcast, ...
THR_SYSOP	system attribute settings
THR_STKOP	stack operation
THR_RTOP	realtime thread settings
THR_SETATTR	set thread attributes

9.1.7 long pctl(long cmd, long parm, void *arg);

The **pctl** system call provides some process control operations.

Arguments

- if cmd is **PROC_WAIT**, wait for process termination
- if cmd is **PROC_USLEEP**, sleep for a given number of microseconds
- if cmd is **PROC_NANOSLEEP**, sleep for a given number of nanoseconds

PROC_WAIT

parm	Operation
PROC_WAITPID	wait for given process to terminate
PROC_WAITCLD	wait for a child process in a group to terminate
PROC_WAITGRP	wait for a child process in caller's group to terminate
PROC_WAITANY	wait for any child process to terminate

9.1.8 long sigop(long pid, long cmd, void *arg);

Arguments

cmd	Operation
SIG_WAIT	wait for a signal; pause()
SIG_SETFUNC	set signal handler; signal(), sigaction()
SIG_SETMASK	set signal mask; sigsetmask()
SIG_SEND	send signal to process; raise(), ...
SIG_SETSTK	configure signal stack; sigaltstack()
SIG_SUSPEND	suspend until signal received; sigsuspend()

SIG_SETFUNC

Set signal disposition.

arg	Operation
SIG_DEFAULT	initialise default signal disposition
SIG_IGNORE	ignore signal

SIG_SEND

Send signal to **pid**.

pid	Operation
SIG_SELF	send signal to self
SIG_CLD	send signal to child processes
SIG_GRP	send signal to process group
SIG_PROPCLD	propagate signal to child processes
SIG_PROPGRP	propagate signal to process group

SIG_PAUSE

Suspend process. Flag bits are passed in **arg**.

arg bit	Operation
SIG_EXIT	exit process on signal
SIG_DUMP	dump core on signal

9.2 Memory Interface

9.2.1 long brk(void *adr);

The **brk** system call adjusts process break, i.e. the current top of heap.

Arguments

- **adr** is the new break

9.2.2 void *map(long desc, long flg, struct memreg *reg);

struct memreg

```
struct memreg {
    void *base;
    long  ofs;
    long  len;
    long  perm;
};
```

Arguments

flg	Operation
MAP_FILE	map file (or /dev/zero for anonymous memory
MAP_ANON	map anonymous memory
MAP_NORMAL	default map behavior
MAP_SEQUENTIAL	sequential access
MAP_RANDOM	non-sequential access
MAP_WILLNEED	needed in the near future
MAP_DONTNEED	not needed soon
MAP_DONTFORK	don't share with forked child processes

9.2.3 long umap(void *adr, size_t size);

The **umap** system call unmaps file or anonymous memory.

9.2.4 long mhint(void *adr, long flg, struct memreg *arg);

The **mhint** system call is used to hint the kernel about the type of use for mapped regions.

Arguments

For values for the **flg** argument, see **map** earlier in this chapter.

9.2.5 void *bmap(long desc, size_t size, long flg);

The **bmap** system call is used to map memory from kernel buffer cache. This memory will be mapped to virtual address spaces of both the kernel and a user process to avoid copying memory for I/O requests.

Arguments

For values for the **flg** argument, see **map** earlier in this chapter.

9.3 Shared Memory

9.3.1 long shmget(long key, size_t size, long flg);

The **shmget** system call returns the shared memory identifier associated with **key**.

9.3.2 void *shmat(long id, void *adr, long flg);

The **shmat** system call attaches shared memory segment **id** to the address space of the calling process.

Arguments

- if **adr** is nonzero, it will be used as the base address for the attached segment; otherwise, a new virtual region shall be allocated
- **TODO: flg argument?**

9.3.3 long shmdt(void *adr);

The **shmdt** system call detaches a shared memory segment attached at **adr** in the calling process's address space.

9.3.4 long shmctl(long id, long cmd, void *arg);

The **shmctl** system call is used for controllling attributes of shared memory segments.

TODO

Part III

Base Drivers

Chapter 10

VGA Text Consoles

VGA text console is the base user interface for Zero before the advent of graphics drivers. There's always a place for separate consoles as command line interfaces, to display console messages from system, and so forth.

VGA color buffer is located at **0xb8000** (under one megabyte) and identity-mapped to the same region in kernel virtual address space. Hence drawing text becomes simple writing of character + attribute values into this memory region.

10.1 VGA Text Interface

This header file declares our VGA text interface.

kern/io/drv/pc/vga.h

```
#ifndef __IO_DRV_PC_VGA_H__
#define __IO_DRV_PC_VGA_H__

#include <stdint.h>
#include <kern/ia32/io.h>

void vgasyncscr(void);
void vgaputs(char *str);
void vgaputchar(int ch);

#define VGANCON        8
#define VGABUFSIZE     (1 << VGABUFSIZELOG2)
#define VGABUFSIZELOG2 12
#define VGACHARSIZE    2
#define VGABUFADR      0x000b8000U
#define VGACONBUFSIZE  (VGANCON * VGABUFSIZE)

/* interface macros */
#define vgasetfg(atr, c) ((atr) | (c))
#define vgasetbg(atr, c) ((atr)| ((c) < 4))
#define vgasetblink(ch)  ((ch) | VGABLINK)
#define vgasetfgcolor(c)                                               \
    do {                                                               \
        struct vgacon *_con;                                           \
        uint16_t      _atr;                                            \
                                                                       \
        _con = &_vgacontab[_vgacurcon];                                \
        _atr = con->chatr;                                             \
        con->chatr = vgasetfg(_atr, (c));                              \
    } while (0)
#define vgaputch(ch)                                                   \
    do {                                                               \
        struct vgacon *_con;                                           \
        uint16_t      *_buf;                                           \
        uint16_t      _atr;                                            \
                                                                       \
        _con = &_vgacontab[_vgacurcon];                                \
        _buf = (uint8_t *)VGABUFADR + _vgacurcon * VGABUFSIZE;         \
        _atr = con->chatr;                                             \
        _buf[con->w * con->x + con->y] = _vgamkch(ch, atr);            \
    } while (0)
#define vgaputch3(ptr, ch, atr)                                        \
    (*(ptr) = _vgamkch(ch, atr))
//    ((buf)[(x) * VGACHARSIZE + (y)] = _vgamkch(ch, atr))
```

```
#define vgaputc(buf, ch)                                                 \
    vgaputch(buf, ch, _vgaatrbuf[_vgaconid(buf)])
#define vgamoveto(x, y)                                                  \
    do {                                                                 \
        uint16_t _ofs = (x) * (y) * sizeof(uint16_t);                    \
                                                                         \
        outb(VGACURHI, VGACRTC);                                         \
        outb(_ofs >> 8, VGACRTC + 1);                                    \
        outb(VGACURLO, VGACRTC);                                         \
        outb(_ofs & 0xff, VGACRTC + 1);                                  \
    } while (0)

/* internal macros */
#define _vgaconid(buf)                                                   \
    (((uintptr_t)(buf) - VGABUFADR) >> VGABUFSIZELOG2)
#define _vgamkch(ch, atr)                                                \
    ((uint16_t)(ch) | ((uint16_t)(atr) << 8))

#define VGACRTC       0x03b4
#define VGACURHI      0x14
#define VGACURLO      0x15

/* text attributes */
#define VGACHARMASK   0x00ff
#define VGAFGMASK     0x0f00
#define VGABGMASK     0x7000
#define VGABLINK      0x8000
#define VGACATRMASK   0xff00

/* text colors */
#define VGABLACK      0x00
#define VGABLUE       0x01
#define VGAGREEN      0x02
#define VGACYAN       0x03
#define VGARED        0x04
#define VGAMAGENTA    0x05
#define VGABROWN      0x06
#define VGAWHITE      0x07
#define VGADARKGRAY   0x08
#define VGABBLUE      0x09
#define VGABGREEN     0x0a
#define VGABCYAN      0x0b
#define VGAPINK       0x0c
#define VGABMAGENTA   0x0d
#define VGAYELLOW     0x0e
#define VGABWHITE     0x0f

/* vga [text] console structure */
struct vgacon {
    uint16_t *buf;
    uint8_t   x;
    uint8_t   y;
    uint8_t   w;
    uint8_t   h;
    uint16_t  chatr;
    long      nbufln;   // number of buffered lines
    void     *data;     // text buffers
} PACK;

/* vga screen information */
struct vgainfo {
    void *fbuf;
    long  w;
    long  h;
    long  nbpp;
    long  fmt;
};

#endif /* __IO_DRV_PC_VGA_H__ */
```

10.2 VGA Console Interface

The following is our VGA text console driver.

kern/io/drv/pc/vga.c

```
#include <stdint.h>
#include <stddef.h>
#include <zero/cdecl.h>
#include <kern/util.h>
#include <kern/io/drv/pc/vga.h>

struct vgacon  _vgacontab[VGANCON] ALIGNED(PAGESIZE);
static long    _vgacurcon;

/* initialise 8 consoles */
void
vgainitcon(int w, int h)
{
    struct vgacon *con = _vgacontab;
    uint8_t       *ptr = (uint8_t *)VGABUFADR;
    long           l;

    for (l = 0 ; l < VGANCON ; l++) {
        bzero(ptr, PAGESIZE);
        con->buf = (uint16_t *)ptr;
        con->x = 0;
        con->y = 0;
        con->w = w;
        con->h = h;
        con->chatr = vgasetfg(0, VGAWHITE);
        con->nbufln = 0;
        /* TODO: allocate scrollback buffer */
        con->data = NULL;
        ptr += VGABUFSIZE;
        con++;
    }
    _vgacurcon = 0;
    vgamoveto(0, 0);
#if 0
    kprintf("VGA @ 0x%x - width = %d, height = %d, %d consoles\n",
            VGABUFADR, w, h, VGANCON);
#endif

    return;
}

void
vgaputs(char *str)
{
    struct vgacon *con;
    uint16_t      *ptr;
    int            x;
    int            y;
    int            w;
    int            h;
    uint8_t        ch;
    uint8_t        atr;

    con = &_vgacontab[_vgacurcon];
    x = con->x;
    y = con->y;
    w = con->w;
    h = con->h;
    atr = con->chatr;
    while (*str) {
        ptr = con->buf + y * w + x;
        ch = *str;
        if (ch == '\n') {
            if (++y == h) {
                y = 0;
            }
            x = 0;
        } else {
            if (++x == w) {
                x = 0;
                if (++y == h) {
                    y = 0;
                }
            }
            vgaputch3(ptr, ch, atr);
        }
        str++;
        con->x = x;
        con->y = y;
    }

    return;
}

void
vgaputs2(long conid, char *str)
{
    struct vgacon *con;
    uint16_t      *ptr;
    int            x;
    int            y;
    int            w;
    int            h;
    uint8_t        ch;
    uint8_t        atr;

    con = &_vgacontab[conid];
    x = con->x;
    y = con->y;
    w = con->w;
    h = con->h;
    atr = con->chatr;
    while (*str) {
        ptr = con->buf + y * w + x;
        ch = *str;
        if (ch == '\n') {
            if (++y == h) {
                y = 0;
            }
            x = 0;
        } else {
            if (++x == w) {
                x = 0;
                if (++y == h) {
                    y = 0;
                }
            }
            vgaputch3(ptr, ch, atr);
        }
        str++;
        con->x = x;
        con->y = y;
    }

    return;
}

void
vgaputchar(int ch)
{
    struct vgacon *con;
    uint16_t      *ptr;

    con = &_vgacontab[_vgacurcon];
    ptr = con->buf + con->w * con->x + con->y;
    *ptr = _vgamkch(ch, con->chatr);

    return;
}

void
vgasyncscr(void)
{
    ;
```

```
}
```

Chapter 11

PS/2 Keyboard and Mouse

11.1 Keyboard Driver

Here is a source file for our PS/2-connector or emulated USB keyboards.

kern/io/drv/pc/ps2/kbd.c

```
#include <kern/conf.h>

#if (PS2DRV)

#include <stdint.h>
#include <kern/util.h>
//#include <kern/event.h>
#include <kern/ia32/io.h>
#include <kern/ia32/trap.h>

/*
 * 19:49 <PeanutHorst>
 * { == curly brace   < == angle bracket   [ == bracket   ( == parentheses
 */

#include "kbd.h"
#include "keysym.h"

extern void *irqvec[];

void kbdinit_us(void);
void kbdint(void);

#if 0
/* modifier keys. */
static int32_t _mkeytabmod[KBD_NTAB];
#endif
```

```
static int32_t _modmask;
/* single-code values. */
static int32_t _mkeytab1b[KBD_NTAB];
/* 0xe0-prefixed values. */
static int32_t _mkeytabmb[KBD_NTAB];
/* release values. */
static int32_t _mkeytabup[KBD_NTAB];

#define kbdread(u8)                                                     \
    __asm__ ("inb %w1, %b0\n" : "=a" (u8) : "Nd" (KBD_PORT))
#define kbdsend(u8)                                                     \
    __asm__ ("outb %b0, %w1\n" : : "a" (u8), "Nd" (KBD_PORT))

#if 0
#define setkeymod(name)                                                 \
    (_mkeytabmod[name] = (name##_FLAG))
#endif
#define ismodkey(val)                                                   \
    (((val) & 0x80000000) && ((val) & 0xfffffff0) == 0xfffffff0)
#if 0
#define setkeymod(name)                                                 \
    (_mkeytabmod[name] = (1 << (-(name##_SYM))))
#endif
#define setkeycode(name)                                        \
    (((((name) >> 8) & 0xff) == KBD_UP_BYTE)                    \
     ? (_mkeytabup[name >> 16] = name##_SYM | KBD_UP_BIT)       \
     : ((((name) & 0xff) == KBD_PREFIX_BYTE)                    \
        ? (_mkeytabmb[name >> 8] = name##_SYM,                  \
           _mkeytabup[name >> 8] = name##_SYM | KBD_UP_BIT)     \
        : (_mkeytab1b[name] = name##_SYM,                       \
           _mkeytabup[name] = name##_SYM | KBD_UP_BIT)))

void
kbdinit(void)
{
    uint8_t u8;

    /* enable keyboard */
    kbdsend(KBD_ENABLE);
    do {
        kbdread(u8);
    } while (u8 != KBD_ACK);
    /* choose scancode set 1 */
    kbdsend(KBD_SETSCAN);
    do {
        kbdread(u8);
    } while (u8 != KBD_ACK);
    kbdsend(0x01);
    do {
        kbdread(u8);
    } while (u8 != KBD_ACK);
    kbdinit_us();
    kprintf("PS/2 keyboard with US keymap initialized\n");
    irqvec[IRQKBD] = kbdint;
    kprintf("PS/2 keyboard interrupt enabled\n");

    return;
}

void
kbdinit_us(void)
{
    /* modifiers. */
    setkeycode(KBD_LEFTCTRL);
    setkeycode(KBD_LEFTSHIFT);
    setkeycode(KBD_RIGHTSHIFT);
    setkeycode(KBD_LEFTALT);
    setkeycode(KBD_RIGHTALT);
    setkeycode(KBD_CAPSLOCK);
    setkeycode(KBD_NUMLOCK);
    setkeycode(KBD_SCROLLLOCK);

    /* single-byte keys. */

    setkeycode(KBD_ESC);
    setkeycode(KBD_1);
    setkeycode(KBD_2);
    setkeycode(KBD_3);
    setkeycode(KBD_4);
    setkeycode(KBD_5);
    setkeycode(KBD_6);
    setkeycode(KBD_7);
    setkeycode(KBD_8);
    setkeycode(KBD_9);
    setkeycode(KBD_0);
    setkeycode(KBD_MINUS);
    setkeycode(KBD_PLUS);
    setkeycode(KBD_BACKSPACE);

    setkeycode(KBD_TAB);
    setkeycode(KBD_q);
    setkeycode(KBD_w);
    setkeycode(KBD_e);
    setkeycode(KBD_r);
    setkeycode(KBD_t);
    setkeycode(KBD_y);
    setkeycode(KBD_u);
    setkeycode(KBD_i);
    setkeycode(KBD_o);
    setkeycode(KBD_p);
    setkeycode(KBD_OPENBRACKET);
    setkeycode(KBD_CLOSEBRACKET);

    setkeycode(KBD_ENTER);

    setkeycode(KBD_LEFTCTRL);

    setkeycode(KBD_a);
    setkeycode(KBD_s);
    setkeycode(KBD_d);
    setkeycode(KBD_f);
    setkeycode(KBD_g);
    setkeycode(KBD_h);
    setkeycode(KBD_i);
    setkeycode(KBD_j);
    setkeycode(KBD_k);
    setkeycode(KBD_l);
    setkeycode(KBD_SEMICOLON);
    setkeycode(KBD_QUOTE);

    setkeycode(KBD_BACKQUOTE);

    setkeycode(KBD_LEFTSHIFT);

    setkeycode(KBD_BACKSLASH);

    setkeycode(KBD_z);
    setkeycode(KBD_x);
    setkeycode(KBD_c);
    setkeycode(KBD_v);
    setkeycode(KBD_b);
    setkeycode(KBD_n);
    setkeycode(KBD_m);
    setkeycode(KBD_COMMA);
    setkeycode(KBD_DOT);
    setkeycode(KBD_SLASH);

    setkeycode(KBD_RIGHTSHIFT);

    setkeycode(KBD_KEYPADASTERISK);

    setkeycode(KBD_SPACE);

    setkeycode(KBD_CAPSLOCK);

    setkeycode(KBD_F1);
    setkeycode(KBD_F2);
    setkeycode(KBD_F3);
    setkeycode(KBD_F4);
    setkeycode(KBD_F5);
    setkeycode(KBD_F6);
    setkeycode(KBD_F7);
    setkeycode(KBD_F8);
    setkeycode(KBD_F9);
    setkeycode(KBD_F10);

    setkeycode(KBD_NUMLOCK);
    setkeycode(KBD_SCROLLLOCK);

    setkeycode(KBD_F11);
    setkeycode(KBD_F12);

    setkeycode(KBD_KEYPAD7);
    setkeycode(KBD_KEYPAD8);
    setkeycode(KBD_KEYPAD9);

    setkeycode(KBD_KEYPADMINUS2);

    setkeycode(KBD_KEYPAD4);
    setkeycode(KBD_KEYPAD5);
    setkeycode(KBD_KEYPAD6);

    setkeycode(KBD_KEYPADPLUS);

    setkeycode(KBD_KEYPADEND);
    setkeycode(KBD_KEYPADDOWN);
    setkeycode(KBD_KEYPADPGDN);

    setkeycode(KBD_KEYPADINS);
    setkeycode(KBD_KEYPADDEL);

    setkeycode(KBD_SYSRQ);

    /* dual-byte sequences. */

    setkeycode(KBD_KEYPADENTER);
    setkeycode(KBD_RIGHTCTRL);
    setkeycode(KBD_FAKELEFTSHIFT);
    setkeycode(KBD_KEYPADMINUS3);
    setkeycode(KBD_FAKERIGHTSHIFT);
    setkeycode(KBD_CTRLPRINTSCREEN);
    setkeycode(KBD_RIGHTALT);
    setkeycode(KBD_CTRLBREAK);
    setkeycode(KBD_HOME);
    setkeycode(KBD_UP);
    setkeycode(KBD_PGUP);
    setkeycode(KBD_LEFT);
    setkeycode(KBD_RIGHT);
    setkeycode(KBD_END);
    setkeycode(KBD_DOWN);
    setkeycode(KBD_PGDN);
    setkeycode(KBD_INS);
    setkeycode(KBD_DEL);

    /* acpi codes. */
    setkeycode(KBD_POWER);
    setkeycode(KBD_SLEEP);
    setkeycode(KBD_WAKE);
    setkeycode(KBD_POWERUP);
    setkeycode(KBD_SLEEPUP);
    setkeycode(KBD_WAKEUP);

    return;
}

/* keyboard interrupt handler. */
void
kbdint(void)
{
    int32_t isup = 0;
    int32_t val;
    uint8_t u8;

    val = 0;
    kbdread(u8);
    if (u8 == KBD_PAUSE_BYTE1) {
        /* pause/break. */
        kbdread(u8); /* 0x1d */
        kbdread(u8); /* 0x45 */
        kbdread(u8); /* 0xe1 */
        kbdread(u8); /* 0x9d */
        kbdread(u8); /* 0xc5 */
        u8 &= KBD_VAL_MSK;
        val = _mkeytab1b[u8];
    } else if (u8 != KBD_PREFIX_BYTE) {
        /* single-byte value. */
        if (u8 & KBD_UP_BIT) {
            /* release. */
            isup = 1;
            u8 &= ~KBD_UP_BIT;
            val = _mkeytabup[u8];
        } else {
            val = _mkeytab1b[u8];
        }
    } else {
        /* 0xe0-prefixed. */
        kbdread(u8);
        if (u8 == KBD_PRINT_BYTE2 || u8 == KBD_CTRLPAUSE_BYTE2) {
            /* print screen or ctrl-pause. */
            kbdread(u8); /* 0xe0 */
            kbdread(u8); /* 0x37 (prtsc) or 0xc6 (ctrl-pause) */
            val &= KBD_VAL_MSK;
            val = _mkeytabmb[u8];
        } else if (u8 == KBD_UP_BYTE) {
            kbdread(u8);
            val = _mkeytabup[u8];
#if (DEVEL)

            return;
#endif
        } else {
            if (u8 & KBD_UP_BIT) {
                isup = 1;
                u8 &= ~KBD_UP_BIT;
                val = _mkeytabup[u8];
            } else {
                val = _mkeytabmb[u8];
            }
        }
    }
    if (ismodkey(val)) {
        if (isup) {
            _modmask &= ~(1 << (-val));
        } else {
            _modmask |= 1 << (-val);
        }
    }

    return;
}

#endif /* PS2DRV */
```

11.2 Mouse Driver

Here is a source file for our PS/2-connector mice.

kern/io/drv/pc/ps2/mouse.c

```
#include <kern/conf.h>

#if (PS2DRV)

#include <stdint.h>
#include <kern/util.h>
#include <zero/cdecl.h>
#include <kern/ia32/trap.h>

extern void *irqvec[];

void mouseint(void);

#define MOUSE_INPORT    0x60

/* state. */
#define MOUSE_LEFTBTN   0x01 /* left button flag. */
#define MOUSE_RIGHTBTN  0x02 /* right button flag. */
#define MOUSE_MIDDLEBTN 0x04 /* middle button flag. */
#define MOUSE_3BTNMSK   0x07 /* 3-button mask. */
#define MOUSE_BTN4      0x08 /* button 4 flag. */
#define MOUSE_BTN5      0x10 /* button 5 flag. */
#define MOUSE_XBTNMSK   0x18 /* extra button (4 & 5) mask. */
/* other data. */
#define MOUSE_XSIGN     0x10 /* x-movement sign. */
#define MOUSE_YSIGN     0x20 /* y-movement sign. */
#define MOUSE_XOVERFLOW 0x40 /* x-movement overflow. */
#define MOUSE_YOVERFLOW 0x80 /* y-movement overflow. */
/* extra byte. */
#define MOUSE_ZSIGN     0x08 /* z-movement sign. */
#define MOUSE_ZMSK      0x07 /* z-movement mask. */
#define MOUSE_5BTNMSK   0x30 /* extra button (4 & 5) mask. */
/* flags. */
#define MOUSE_WHEEL      0x01 /* scroll wheel flag. */
#define MOUSE_WHEEL5BTN 0x02 /* 5-button flag. */
#define MOUSE_WHEELMSK  0x03 /* intellimouse mask. */
struct mousestat {
    uint16_t flags;
    uint16_t state;
    uint32_t x;
    uint32_t y;
    uint32_t z;
    uint32_t xmax;
    uint32_t ymax;
    uint32_t zmax;
    int32_t shift;
} PACK();

static struct mousestat _mousestat ALIGNED(CLSIZE);

#define mouseread(u8)                                                           \
    __asm__("inb %w1, %b0" : "=a" (u8) : "i" (MOUSE_INPORT))

void
mouseinit(void)
{
    irqvec[IRQMOUSE] = mouseint;
    kprintf("PS/2 mouse interrupt enabled\n");

    return;
}

void
mouseint(void)
{
    uint32_t val;
    int32_t  xmov;
    int32_t  ymov;
    int32_t  zmov;
    int32_t  xtra;
    int32_t  shift;
    int32_t  tmp;
    uint8_t  msk;
    uint8_t  stat;
    uint8_t  u8;

    mouseread(msk);
    mouseread(u8);
    xmov = u8;
    mouseread(u8);
    ymov = u8;

    val = _mousestat.flags;
    zmov = 0;
    val &= MOUSE_WHEELMSK;                          /* scroll-wheel?. */
    stat = msk & MOUSE_3BTNMSK;                     /* button 1, 2 & 3 states. */
    if (val) {
        /* mouse with scroll-wheel, extra (4th) data byte. */
        mouseread(u8);
        xtra = u8;
        val &= MOUSE_WHEEL5BTN;                     /* 5-button?. */
        zmov = xtra & MOUSE_ZMSK;                   /* z-axis movement. */
        tmp = xtra & MOUSE_ZSIGN;                   /* extract sign bit. */
        if (val) {
            stat |= (xtra >> 1) & MOUSE_XBTNMSK;  /* button 4 & 5 states. */
        }
        if (tmp) {
            zmov = -zmov;
        }
    }
    _mousestat.state = stat;

    shift = _mousestat.shift;                     /* scale (speed) value. */

    val = _mousestat.x;
    tmp = msk & MOUSE_XOVERFLOW;
    if (tmp) {
        xmov = 0xff;
    } else if (shift > 0) {
        xmov <<= shift;
    } else {
        xmov >>= shift;
    }
    tmp = msk & MOUSE_XSIGN;
//      xmov |= tmp << 27; /* sign. */
    if (tmp) {
        xmov = -xmov;
    }
    if (xmov < 0) {
        _mousestat.x = (val < -xmov) ? 0 : (val + xmov);
    } else {
        tmp = _mousestat.xmax;
        _mousestat.x = (val < tmp - val) ? (val + xmov) : tmp;
    }

    val = _mousestat.y;
    tmp = msk & MOUSE_YOVERFLOW;
    if (tmp) {
        ymov = 0xff;
    } else if (shift > 0) {
        ymov <<= shift;
    } else {
        ymov >>= shift;
    }
    tmp = msk & MOUSE_YSIGN;
//      ymov |= tmp << 26; /* sign. */
    if (tmp) {
        ymov = -ymov;
    }
    if (ymov < 0) {
        _mousestat.y = (val < -ymov) ? 0 : (val + ymov);
    } else {
        tmp = _mousestat.ymax;
        _mousestat.y = (val < tmp - val) ? (val + ymov) : tmp;
```

```
    }

    if (zmov) {
        val = _mousestat.z;
        if (zmov < 0) {
            _mousestat.z = (val < -zmov) ? 0 : (val + zmov);
        } else {
            tmp = _mousestat.zmax;
            _mousestat.z = (val < tmp - val) ? (val + zmov) : tmp;
        }
    }

    return;
}

#endif /* PS2DRV */
```

Chapter 12

AC97 Audio Interface

I chose to develop drivers for AC97 because it's not only very wide-spread, but also well documented.

TODO: extend this chapter

Appendix A

Profiler Tools

Zero has simple tools for timing code execution both in microsecond-resolution wall clock time as well as CPU clock cycles.

This chapter shows the implementation of these profiling tools as well as examples of their use.

A.1 Wall Clock Profiler

```
#ifndef __ZERO_PROF_H__
#define __ZERO_PROF_H__

#include <stdint.h>
#include <sys/time.h>

#if defined(__x86_64__) || defined(__amd64__)
#include <zero/amd64/prof.h>
#elif defined(__i386__)
#include <zero/ia32/prof.h>
#elif defined(__arm__)
#include <zero/arm/prof.h>
#endif

#define tvcmp(tv1, tv2)                                                 \
    (((tv2)->tv_sec - (tv1)->tv_sec) * 1000000                          \
     + ((tv2)->tv_usec - (tv1)->tv_usec))
#define tvgt(tv1, tv2)                                                  \
    (((tv1)->tv_sec > (tv2)->tv_sec)                                    \
     || ((tv1)->tv_sec == (tv2)->tv_sec && (tv1)->tv_usec > (tv2)->tv_usec))

#define tvaddconst(tv, u)                                               \
  do {                                                                  \
      (tv)->tv_sec += (u) / 1000000;                                    \
      (tv)->tv_usec += (u) % 1000000;                                   \
                                                                        \
      if ((tv)->tv_usec >= 1000000) {                                   \
          (tv)->tv_sec++;                                               \
          (tv)->tv_usec -= 1000000;                                     \
      } else if ((tv)->tv_usec < 0) {                                   \
          (tv)->tv_sec--;                                               \
          (tv)->tv_usec += 1000000;                                     \
      }                                                                 \
  } while (FALSE)

#define PROFCLK(id)                                                     \
    struct timeval _tv##id[2]
#define profinitclk(id)                                                 \
    memset(&_tv##id, 0, sizeof(_tv##id))
#define profstartclk(id)                                                \
    gettimeofday(&_tv##id[0], NULL)
#define profstopclk(id)                                                 \
    gettimeofday(&_tv##id[1], NULL)
#define profclkdiff(id)                                                 \
    tvcmp(&_tv##id[0], &_tv##id[1])

#endif /* __ZERO_PROF_H__ */
```

A.2 Cycle Profiler

```
#ifndef __ZERO_IA32_PROF_H__
#define __ZERO_IA32_PROF_H__

#include <stdint.h>

union m_tickcnt {
    uint64_t u64val;
    uint32_t u32vals[2];
};

#define PROFTICK(id)                                                    \
    union m_tickcnt __tc##id[2]
#define profinittick(id)                                                \
    memset(&__tc##id, 0, sizeof(__tc##id))
#define profstarttick(id)                                               \
    _gettickcnt(&__tc##id[0])
#define profstoptick(id)                                                \
    _gettickcnt(&__tc##id[1])
#define proftickdiff(id)                                                \
    (__tc##id[1].u64val - __tc##id[0].u64val)
#define Z_PROF_TICK_FORMAT "%llu"

#define _gettickcnt(ptr)                                         \
    __asm__("rdtsc; movl %%eax, %0; movl %%edx, %1"              \
            : "=m" ((ptr)->u32vals[0]), "=m" ((ptr)->u32vals[1]) \
            :                                                    \
            : "eax", "edx");

#if 0
__inline__ uint64_t
_rdpmc(union _tickcnt *cnt, int id)
{
    __asm__("movl %0, %%ecx ; rdpmc ; mov %%eax, %1 ; mov %%edx, %2"
            : "=rm" (cnt->u32vals[0]), "=rm" (cnt->u32vals[1])
            : "rm" (id)
            : "eax", "edx");

    return (cnt->u64val);
}
#endif

#endif /* __ZERO_IA32_PROF_H__ */
```

A.3 Examples

A.3.1 Wall Clock Profiler

```
#include <zero/prof.h>
#include <stdio.h>

int
main(int argc, char *argv)
{
    PROFCLK(clk);

    profstartclk(clk);
    printf("hello world\n");
    profstopclk(clk);
    printf("%lu microseconds to print\n", profclkdiff(clk));
}
```

A.3.2 Cycle Profiler

```
#include <zero/prof.h>
#include <stdio.h>

int
main(int argc, char *argv)
{
    PROFCLK(clk);

    profstartclk(clk);
    printf("hello world\n");
    profstopclk(clk);
    printf("%lu microseconds to print\n", profclkdiff(clk));
}
```