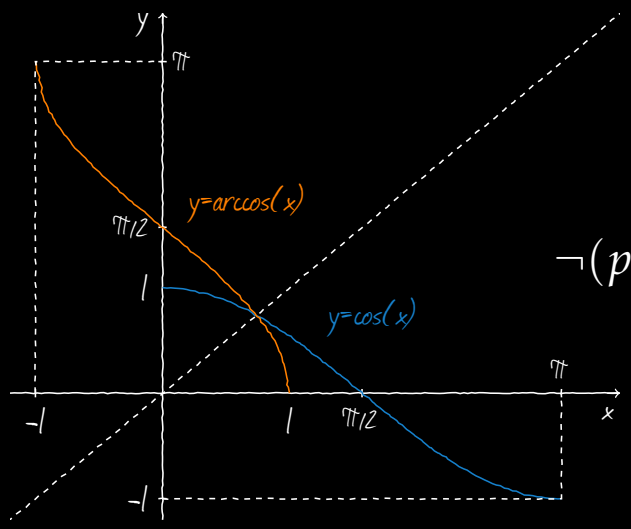
ZJUNIX

实验操作流程

浙江大学

2017.08.20

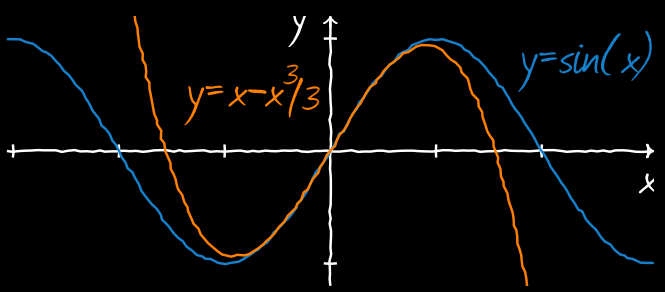
$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k}$$



$$\zeta_k = |a|^{1/n} \mathrm{e}^{i(\arg(a)+2k\pi)/n}$$

$$\mathrm{e}^{i\pi} + 1 = 0$$

$$\neg(p \vee q) \equiv (\neg p) \wedge (\neg q)$$



$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

目录

1.1 实验目的

- 理解 ZJUNIX 操作系统中对于异常的处理
- 编写异常机制初始化代码
- 编写系统调用初始化代码
- 测试系统调用的可用性

1.2 实验步骤

1.2.1 目录结构

代码 1.1: 实验 3 目录结构

```
exp3
├── arch
│   ├── Makefile
│   └── mips32
│       ├── arch.c
│       ├── arch.h
│       ├── exc.c
│       ├── exc.h
│       ├── intr.c
│       ├── intr.h
│       ├── Makefile
│       └── start.s
├── config
│   ├── debug.h
│   ├── flags.conf
│   ├── kernel.ld
│   ├── make.global
│   ├── submake.global
│   └── tools.conf
├── include
│   ├── init_place_holder.h
│   └── zjunix
│       └── syscall.h
├── kernel
│   ├── init.c
│   ├── Makefile
│   └── syscall
│       ├── Makefile
│       ├── syscall4.c
│       ├── syscall4.h
│       └── syscall.c
├── LICENSE
└── Makefile
```

```
└── README.md
```

1.2.2 新加入的代码

代码 1.2: arch/mips32/exc.h

```h
#ifndef _EXC_H
#define _EXC_H

typedef void (*exc_fn)(unsigned int, unsigned int, unsigned int*);

extern exc_fn exceptions[32];

void do_exceptions(unsigned int status, unsigned int cause, unsigned int* sp);
void register_exception_handler(int index, exc_fn fn);
void init_exception();

#endif
```

代码 1.3: arch/mips32/exc.c

```c
#include "exc.h"

#pragma GCC push_options
#pragma GCC optimize("O0")

exc_fn exceptions[32];

void do_exceptions(unsigned int status, unsigned int cause, unsigned int* sp) {
    int index = cause >> 2;
    index &= 0x1f;
    if (exceptions[index]) {
        exceptions[index](status, cause, sp);
    } else {
        while (1)
            ;
    }
}

void register_exception_handler(int index, exc_fn fn) {
    index &= 31;
    exceptions[index] = fn;
}

void init_exception() {
    // status 0000 0000 0000 0000 0000 0000 0000 0000
    // cause 0000 0000 1000 0000 0000 0000 0000 0000
    asm volatile(
```

```
        "mtc0 $zero, $12\n\t"
        "li $t0, 0x800000\n\t"
        "mtc0 $t0, $13\n\t");
}

#pragma GCC pop_options
```

代码 1.4: arch/mips32/intr.h

```
#ifndef _INTR_H
#define _INTR_H

typedef void (*intr_fn)(unsigned int, unsigned int, unsigned int*);

extern intr_fn interrupts[8];

void init_interrupts();
int enable_interrupts();
int disable_interrupts();
void do_interrupts(unsigned int status, unsigned int cause, unsigned int* sp);
void register_interrupt_handler(int index, intr_fn fn);

#endif
```

代码 1.5: arch/mips32/intr.c

```
#include "intr.h"
#include "arch.h"

#pragma GCC push_options
#pragma GCC optimize("O0")

intr_fn interrupts[8];

void init_interrupts() {
    // status 0000 0000 0000 0000 1001 1100 0000 0001
    // cause 0000 0000 1000 0000 0000 0000 0000 0000
    unsigned int t;
    asm volatile(
        "mfc0 $t0, $12\n\t"
        "ori $t0, $t0, 0x1\n\t"
        "move %0, $t0\n\t"
        "mtc0 $t0, $12"
        : "=r"(t));
}

int enable_interrupts() {
    int old = 0;
    asm volatile(
```

```
        "mfc0 $t0, $12\n\t"
        "andi %0, $t0, 0x1\n\t"
        "ori $t0, $t0, 0x1\n\t"
        "mtc0 $t0, $12"
        : "=r"(old));
    return old;
}

int disable_interrupts() {
    int old = 0;
    asm volatile(
        "mfc0 $t0, $12\n\t"
        "andi %0, $t0, 0x1\n\t"
        "li $t1, 0xfffffffe\n\t"
        "and $t0, $t0, $t1\n\t"
        "mtc0 $t0, $12"
        : "=r"(old));
    return old;
}

void do_interrupts(unsigned int status, unsigned int cause, unsigned int* sp) {
    int i;
    int index = cause >> 8;
    for (i = 0; i < 8; i++) {
        if ((index & 1) && interrupts[i] != 0) {
            interrupts[i](status, cause, sp);
        }
        index >>= 1;
    }
}

void register_interrupt_handler(int index, intr_fn fn) {
    index &= 7;
    interrupts[index] = fn;
    index = 1 << (index + 8);
    asm volatile(
        "mfc0 $t0, $12\n\t"
        "or $t0, $t0, %0\n\t"
        "mtc0 $t0, $12"
        : "=r"(index));
}

#pragma GCC pop_options
```

代码 1.6: include/zjunix/syscall.h

```
#ifndef _ZJUNIX_SYSCALL_H
#define _ZJUNIX_SYSCALL_H
```

```
typedef void(*sys_fn)(unsigned int, unsigned int, unsigned int, unsigned int);

extern sys_fn syscalls[256];

void init_syscall();
void syscall(unsigned int status, unsigned int cause, unsigned int* sp);
void register_syscall(int index, sys_fn fn);

#endif // ! _ZJUNIX_SYSCALL_H
```

代码 1.7: kernel/syscall/syscall.c

```
#include <exc.h>
#include <zjunix/syscall.h>
#include "syscall4.h"

sys_fn syscalls[256];

void init_syscall() {
    register_exception_handler(8, syscall);

    // register all syscalls here
    register_syscall(4, syscall4);
}

void syscall(unsigned int status, unsigned int cause, unsigned int* sp) {
    unsigned int a0, a1, a2, a3, v0;
    a0 = *(sp + 4);
    a1 = *(sp + 5);
    a2 = *(sp + 6);
    a3 = *(sp + 7);
    v0 = *(sp + 2) & 255;
    *(sp + 0) += 4;  // EPC
    if (syscalls[v0]) {
        syscalls[v0](a0, a1, a2, a3);
    }
}

void register_syscall(int index, sys_fn fn) {
    index &= 255;
    syscalls[index] = fn;
}
```

代码 1.8: kernel/syscall/syscall4.c

```
#include <arch.h>
#include <zjunix/syscall.h>
```

```
void syscall4(unsigned int a0, unsigned int a1, unsigned int a2, unsigned int a3)
     {
    *GPIO_LED = a0;
}
```

代码 1.9: kernel/syscall/syscall3.h

```
#ifndef _SYSCALL4_H
#define _SYSCALL4_H

void syscall4(unsigned int a0, unsigned int a1, unsigned int a2, unsigned int a3)
    ;

#endif  // ! _SYSCALL4_H
```

代码 1.10: kernel/syscall/Makefile

```
OBJS := syscall.o syscall4.o
DIRS :=

include $(SUB_MAKE_INCLUDE)
```

1.2.3 修改的代码

代码 1.11: arch/mips32/Makefile

```
# 新加入exc.o与intr.o
OBJS := arch.o start.o exc.o intr.o
DIRS :=

include $(SUB_MAKE_INCLUDE)
```

代码 1.12: arch/mips32/start.s

```
.extern init_kernel
.globl start
.globl exception
.extern kernel_sp
.extern exception_handler
.extern interrupt_handler

.set noreorder
.set noat
.align 2

exception:
    #TLB refill
```

```
    mfc0 $k0, $4
    lw $k1, 0($k0)
    mtc0 $k1, $2
    lw $k1, 4($k0)
    mtc0 $k1, $3
    lw $k1, 8($k0)
    mtc0 $k1, $10
    lw $k1, 12($k0)
    mtc0 $k1, $5
    nop #   CP0 hazard
    nop #   CP0 hazard
    tlbwr
    eret

.org 0x0180
    lui $k0, 0x8000
    sltu $k0, $sp, $k0
    beq $k0, $zero, exception_save_context
    move $k1, $sp
    la  $k0, kernel_sp
    j exception_save_context
    lw  $sp, 0($k0)

.org 0x0200
    lui $k0, 0x8000
    sltu $k0, $sp, $k0
    beq $k0, $zero, interrupt_save_context
    move $k1, $sp
    la  $k0, kernel_sp
    lw  $sp, 0($k0)

interrupt_save_context:
    addiu $sp, $sp, -128
    sw $at, 4($sp)
    sw $v0, 8($sp)
    sw $v1, 12($sp)
    sw $a0, 16($sp)
    sw $a1, 20($sp)
    sw $a2, 24($sp)
    sw $a3, 28($sp)
    sw $t0, 32($sp)
    sw $t1, 36($sp)
    sw $t2, 40($sp)
    sw $t3, 44($sp)
    sw $t4, 48($sp)
    sw $t5, 52($sp)
    sw $t6, 56($sp)
    sw $t7, 60($sp)
    sw $s0, 64($sp)
```

```
    sw $s1, 68($sp)
    sw $s2, 72($sp)
    sw $s3, 76($sp)
    sw $s4, 80($sp)
    sw $s5, 84($sp)
    sw $s6, 88($sp)
    sw $s7, 92($sp)
    sw $t8, 96($sp)
    sw $t9, 100($sp)
    sw $gp, 112($sp)
    sw $k1, 116($sp)
    sw $fp, 120($sp)
    sw $ra, 124($sp)
    mfc0 $a0, $12
    mfc0 $a1, $13
    mfc0 $a2, $14
    mfhi $t3
    mflo $t4
    sw $a2, 0($sp) # EPC
    sw $t3, 104($sp) # HI
    sw $t4, 108($sp) # LO

#   jump to do_interrupts
    move $a2, $sp
    addi $sp, $sp, -32
    jal do_interrupts
    nop
    addi $sp, $sp, 32

restore_context:
    lw $a2, 0($sp) # EPC
    lw $t3, 104($sp) # HI
    lw $t4, 108($sp) # LO
    mtc0 $a2, $14
    mthi $t3
    mtlo $t4
    lw $at, 4($sp)
    lw $v0, 8($sp)
    lw $v1, 12($sp)
    lw $a0, 16($sp)
    lw $a1, 20($sp)
    lw $a2, 24($sp)
    lw $a3, 28($sp)
    lw $t0, 32($sp)
    lw $t1, 36($sp)
    lw $t2, 40($sp)
    lw $t3, 44($sp)
    lw $t4, 48($sp)
    lw $t5, 52($sp)
```

```
    lw $t6, 56($sp)
    lw $t7, 60($sp)
    lw $s0, 64($sp)
    lw $s1, 68($sp)
    lw $s2, 72($sp)
    lw $s3, 76($sp)
    lw $s4, 80($sp)
    lw $s5, 84($sp)
    lw $s6, 88($sp)
    lw $s7, 92($sp)
    lw $t8, 96($sp)
    lw $t9, 100($sp)
    lw $gp, 112($sp)
    lw $k1, 116($sp)
    lw $fp, 120($sp)
    lw $ra, 124($sp)
    move $sp, $k1
    eret

exception_save_context:
    addiu $sp, $sp, -128
    sw $at, 4($sp)
    sw $v0, 8($sp)
    sw $v1, 12($sp)
    sw $a0, 16($sp)
    sw $a1, 20($sp)
    sw $a2, 24($sp)
    sw $a3, 28($sp)
    sw $t0, 32($sp)
    sw $t1, 36($sp)
    sw $t2, 40($sp)
    sw $t3, 44($sp)
    sw $t4, 48($sp)
    sw $t5, 52($sp)
    sw $t6, 56($sp)
    sw $t7, 60($sp)
    sw $s0, 64($sp)
    sw $s1, 68($sp)
    sw $s2, 72($sp)
    sw $s3, 76($sp)
    sw $s4, 80($sp)
    sw $s5, 84($sp)
    sw $s6, 88($sp)
    sw $s7, 92($sp)
    sw $t8, 96($sp)
    sw $t9, 100($sp)
    sw $gp, 112($sp)
    sw $k1, 116($sp)
    sw $fp, 120($sp)
```

```
    sw $ra, 124($sp)
    mfc0 $a0, $12
    mfc0 $a1, $13
    mfc0 $a2, $14
    mfhi $t3
    mflo $t4
    sw $a2, 0($sp) # EPC
    sw $t3, 104($sp) # HI
    sw $t4, 108($sp) # LO

# jump to do_exceptions
    move $a2, $sp
    addi $sp, $sp, -32
    jal do_exceptions
    nop
    addi $sp, $sp, 32

    j restore_context
    nop

.org 0x1000
start:
    lui $sp, 0x8100
    la $gp, _gp
    j init_kernel
    nop
```

代码 1.13: include/init_place_holder.h

```
// 去掉以下几行
void init_interrupts() {}
void init_exception() {}
void init_syscall() {}
```

代码 1.14: kernel/init.c

```
// 新加入如下内容
#include <exc.h>
#include <intr.h>
#include <zjunix/syscall.h>

void test_syscall4() {
    asm volatile(
        "li $a0, 0x00ff\n\t"
        "li $v0, 4\n\t"
        "syscall\n\t"
        "nop\n\t");
}
```

```
// 在init()中加入如下内容
void init() {
    ...
    // Init finished, write seg
    *GPIO_SEG = 0x11223344;
    test_syscall4();
    // Halt
    while (1);
}
```

1.3 预期结果

根据 init.c，在系统启动完成后，七段管会显示 11223344，同时，根据 test_syscall4()，通过调用 4 号系统调用，LED 的最低 8 位会被点亮。