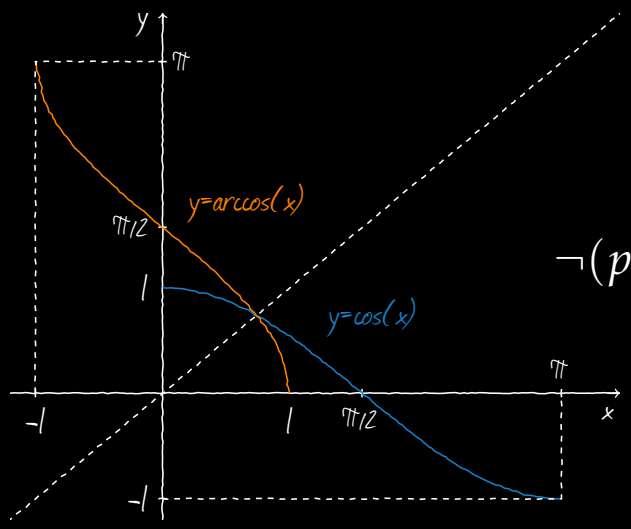
ZJUNIX

实验操作流程

浙江大学

2017.08.20

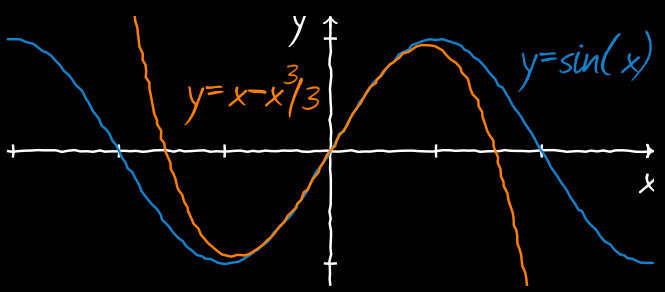
$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k}$$



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

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

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



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

目录

1.1 实验目的

1. 理解 ZJUNIX 操作系统启动与初始化的原理
2. 编写启动与初始化代码
3. 编译并链接生成内核镜像

1.2 实验步骤

这次实验是实际意义上的第一次实验，您可以从头开始搭建这次实验的工程，也可以直接进入本次实验的工程进行编译和修改

这次实验对应的工程是 exp2 目录下的工程，接下来我们会指导您从头建立本次实验的工程

- 新建您的实验 2 工程文件夹，比如 myexp2
- 从 exp1 中复制 config 文件夹与根 Makefile 到 myexp2 目录下
- 新建目录如??所示

代码 1.1: 实验 2 目录结构

```
| myexp2
    | config # 编译配置
    | arch   # 架构相关代码
        | mips32 # mips32架构
            | start.s   # 启动代码
            | arch.c    # 架构相关配置
            | arch.h    # 架构相关配置
            | Makefile  # 子Makefile
        | Makefile # 子Makefile
    | include
        | init_place_holder.h # 一些预留函数定义
    | kernel
        | init.c   # 启动初始化代码
        | Makefile # 子Makefile
    | Makefile # 根Makefile
```

- 在各个文件中填入以下内容

代码 1.2: arch/mips32/start.s

```
.globl exception
.globl start
.extern init_kernel

.set noreorder
.set noat
```

```
.align 2

exception:

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

代码 1.3: arch/mips32/arch.c

```c
#include "arch.h"

//  Virtual Memory
unsigned int* const CHAR_VRAM = (unsigned int*)0xbfc04000;
unsigned int* const GRAPHIC_VRAM = (unsigned int*)0xbfe0000;
unsigned int* const GPIO_SWITCH = (unsigned int*)0xbfc09000;     // switch read-
    only
unsigned int* const GPIO_BUTTON = (unsigned int*)0xbfc09004;     // button read-
    only
unsigned int* const GPIO_SEG = (unsigned int*)0xbfc09008;        // Seg R/W
unsigned int* const GPIO_LED = (unsigned int*)0xbfc0900c;        // LED R/W
unsigned int* const GPIO_PS2_DATA = (unsigned int*)0xbfc09010;   // PS/2 data
    register, R/W
unsigned int* const GPIO_PS2_CTRL = (unsigned int*)0xbfc09014;   // PS/2 control
    register, R/W
unsigned int* const GPIO_UART_DATA = (unsigned int*)0xbfc09018;  // UART data
    register, R/W
unsigned int* const GPIO_UART_CTRL = (unsigned int*)0xbfc0901c;  // UART control
    register, R/W
unsigned int* const GPIO_CURSOR = (unsigned int*)0xbfc09020;     // Cursor 8-bit
    frequency 8-bit row 8-bit col
unsigned int* const VGA_MODE = (unsigned int*)0xbfc09024;        // enable
    graphic mode
// kernel sp
volatile unsigned int kernel_sp = 0x81000000;
```

代码 1.4: arch/mips32/arch.h

```c
#ifndef _ARCH_H
#define _ARCH_H

//  machine params
#define MACHINE_MMSIZE 128 * 1024 * 1024     // 128MB
#define MACHINE_SDSIZE 16 * 1024 * 1024 * 2  // 32M Sectors
#define CHAR_VRAM_SIZE 128 * 32 * 4          // 128*32*4
#define PAGE_TABLE_SIZE 256 * 1024           // 4MB
```

```
#define GRAPHIC_VRAM_SIZE 1024 * 512 * 4     // 1024*512*4 b-g-r

//  Virtual Memory
#define BIOS_ENTRY 0xbfc00000
#define KERNEL_STACK_BOTTOM 0x81000000
#define KERNEL_CODE_ENTRY 0x80001000
#define KERNEL_ENTRY 0x80000000
#define USER_ENTRY 0x00000000

extern unsigned int* const CHAR_VRAM;
extern unsigned int* const GRAPHIC_VRAM;
extern unsigned int* const GPIO_SWITCH;      // switch read-only
extern unsigned int* const GPIO_BUTTON;      // button read-only
extern unsigned int* const GPIO_SEG;         // Seg R/W
extern unsigned int* const GPIO_LED;         // LED R/W
extern unsigned int* const GPIO_PS2_DATA;    // PS/2 data register, R/W
extern unsigned int* const GPIO_PS2_CTRL;    // PS/2 control register, R/W
extern unsigned int* const GPIO_UART_DATA;   // UART data register, R/W
extern unsigned int* const GPIO_UART_CTRL;   // UART control register, R/W
extern unsigned int* const GPIO_CURSOR;      // Cursor 8-bit frequency 8-bit row
    8-bit col
extern unsigned int* const VGA_MODE;         // enable graphic mode

// kernel sp
extern volatile unsigned int kernel_sp;

// PS/2 control register:
//  [5:0]: RX buffer load(R)
//  [13:0]:TX buffer load(R)
//  [18:16]: Error code(R)
//  [31]:  Interrupt enable(RW)

//  UART control register:
//  [7:0]: RX buffer load(R)
//  [15:8]:TX buffer load(R)
//  [18:16]: baud rate(RW)
//  [31]:  Interrupt enable(RW)

#endif
```

代码 1.5: arch/mips32/Makefile

```
# 分别对应arch.c和start.s生成的中间文件
OBJS := arch.o start.o
DIRS :=

include $(SUB_MAKE_INCLUDE)
```

代码 1.6: arch/Makefile

```
OBJS :=
# $(ARCH)是在根Makefile中指定的，默认为mips32
DIRS := $(ARCH)

include $(SUB_MAKE_INCLUDE)
```

代码 1.7: include/init_place_holder.h

```
// 这些是没有具体实现的初始化流程
// 在后面的实验中会不断实现这些初始化流程与相关模块
#ifndef _INIT_PLACE_HOLDER_H
#define _INIT_PLACE_HOLDER_H

void init_interrupts() {}
void init_exception() {}
void init_syscall() {}
void init_pgtable() {}
void init_vga() {}
void init_ps2() {}
void init_mem() {}
void init_fs() {}
void init_ps() {}
void init_pc() {}

#endif // ! _INIT_PLACE_HOLDER_H
```

代码 1.8: kernel/init.c

```
#include <arch.h>
#include <init_place_holder.h>

void init_kernel() {
    // Exception
    init_exception();
    // System call
    init_syscall();
    // Page table
    init_pgtable();
    // Drivers
    init_vga();
    init_ps2();
    // Memory management
    init_mem();
    // File system
    init_fs();
    // Process control
    init_pc();
```

```
    // Interrupts
    init_interrupts();
    // Init finished, write seg
    *GPIO_SEG = 0x11223344;
    // Halt
    while(1);
}
```

代码 1.9: kernel/Makefile

```
OBJS := init.o
DIRS :=

include $(SUB_MAKE_INCLUDE)
```

- 在根目录下执行 make
- 将编译得到的 kernel.bin 复制到 sd 卡中，插入 SWORD 板卡
- 按下 CPU RESET 重启板卡

1.3 实验预期结果

根据 init.c 的代码，在初始化流程结束后，板卡的七段数码管会显示 11223344。即，如果看到 11223344 显示，说明系统启动与初始化已经完成并且成功。