> Q：引人深思的博客
>
> A：OS助教反馈这个问题之后会进行优化

> 往届：
>
> - 21
>   - lab1-1
>   - exam：输出段表内容
>   - extra：大小端转换
>   - lab1-2
>   - exam：结构体输出
>   - extra：C语言调用汇编函数
> - 22
>   - lab1-1
>   - exam：输出段表内容
>   - extra：判断段表是否重叠（`ROUNDDOWN`方法计算页号）
>   - lab1-2
>   - exam：输出结构体
>   - extra：C语言调用汇编函数
> - 23
>   - exam：输出 range 格式 `(1, 2)`
>   - extra：实现 `sprintf()` 输出到字符指针地址
> - 24
>   - exam：定义新运算 `(x, y, |x ^ 2 - y ^ 2|)`
>   - extra：按步骤实现 `vscanfmt`

> 可以打开 dev 哟~~~

> **实验代码中各个目录的组织以及其中的重要文件**

- 根目录下还存在 `kernel.lds` 这个 linker script 文件，我们会在下面的小节中详细讲解。
- `init` 目录中主要有两个代码文件 `start.S` 和 `init.c`，其作用是初始化内核。`start.S` 文件中的 `_start` 函数是 CPU 控制权被转交给内核后执行的第一个函数，主要工作是初始化 CPU 和栈指针，为之后的内核初始化做准备，最后跳转到 `init.c` 文件中定义的 `mips_init` 函数。在本章中 `mips_init` 函数只是简单的打印输出，而在之后的实验中会逐步添加新的内核功能，内核中各模块的初始化函数都会在这里被调用。
- `include` 目录中存放系统头文件。在本章中需要用到的头文件是 `mmu.h` 文件，这个文件中有一张内存布局图，我们在填写 linker script 的时候需要根据这个图来设置相应节的加载地址。
- `lib` 目录存放一些常用库函数，本章中主要存放用于格式化输出的函数。
- `kern` 目录中存放内核的主体代码，本章中主要存放的是终端输出相关的函数。
- `tests` 目录中存放公开的测试用例，我们在进行本地测试时会用到它。

ELF文件

1. ELF 头，包括程序的基本信息，比如体系结构和操作系统，同时也包含了节头表和段头表相对文件的偏移量（offset）。
2. 段头表（或程序头表，program header table），主要包含程序中各个段（segment）的信息，段的信息需要在运行时刻使用。
3. 节头表（section header table），主要包含程序中各个节（section）的信息，节的信息需要在程序编译和链接的时候使用。
4. 段头表中的每一个表项，记录了该段数据载入内存时的目标位置等，记录了用于指导应用程序加载的各类信息。
5. 节头表中的每一个表项，记录了该节程序的代码段、数据段等各个段的内容，主要是链接器在链接的过程中需要使用。

Section Mapping

```
Section to Segment mapping:
 段节...
  00     .note.ABI-tag .note.gnu.build-id .rela.plt .init .plt .text
  __libc_freeres_fn __libc_thread_freeres_fn .fini .rodata __libc_subfreeres
  __libc_atexit __libc_thread_subfreeres .eh_frame .gcc_except_table
  01     .tdata .init_array .fini_array .jcr .data.rel.ro .got .got.plt .data
  .bss __libc_freeres_ptrs
  02     .note.ABI-tag .note.gnu.build-id
  03     .tdata .tbss
  04
```

MIPS内存布局

1. kuseg 0x00000000-0x7FFFFFFF(2 GB)：这一段是用户态下唯一可用的地址空间（内核态下也可使用这段地址空间），大小为 2 GB，也就是 MIPS 约定的用户内存空间。需要通过 MMU（Memory Management Unit）中的 TLB 进行虚拟地址到物理地址的变换。对这段地址的存取都会通过 cache。
2. kseg0 0x80000000-0x9FFFFFFF(512 MB)：这一段是内核态下可用的地址，MMU 将地址的最高位清零 (& 0x7fffffff) 就得到物理地址用于访存。也就是说，这段的虚拟地址被连续地映射到物理地址的低 512 MB 空间。对这段地址的存取都会通过 cache。
3. kseg1 0xA0000000-0xBFFFFFFF(512 MB)：与 kseg0 类似，这段地址也是内核态下可用的地址，MMU 将虚拟地址的高三位清零 (& 0x1fffffff) 就得到物理地址用于访存。这段虚拟地址也被连续地映射到物理地址的低 512 MB 空间。但是对这段地址的存取不通过 cache，往往在这段地址上使用 MMIO（Memory-Mapped I/O）技术来访问外设。
4. kseg2 0xC0000000-0xFFFFFFFF(1 GB)：这段地址只能在内核态下使用并且需要 MMU 中 TLB 将虚拟地址转换为物理地址。对这段地址的存取都会通过 cache。

指针问题

- 此处应该联想 `int a[]; a[4] == a + 4`，当指针有了具体类型指向后，指针的数值偏移就是直接按类型大小进行
  1. **`binary` 的指针运算按字节计算**
     当执行 **`binary + ehdr->e_shoff`** 时，加法操作直接按字节偏移计算。例如，若 **`e_shoff = 100`**，则结果地址是 **`binary`** 的地址加上100字节。
  2. **`ehdr` 的指针运算按结构体大小计算**
     若将代码改为 **`ehdr + ehdr->e_shoff`**，由于 **`ehdr`** 是 **`Elf32_Ehdr*`** 类型，指针加法会根据 **`sizeof(Elf32_Ehdr)`** 调整偏移量。假设 **`Elf32_Ehdr`** 大小为52字节（实际值可能因平台而异），则 **`ehdr + 100`** 实际偏移量为 **`100 * 52 = 5200`** 字节。这会导致计算出的地址远超合法内存范围，触发段错误。
- `const char *fmt` 的定义表示 `fmt` 是一个指向常量字符的指针。这里的 `const` 修饰的是指针所指向的内容，而不是指针本身

MIPS

- 涉及汇编代码编译

```
CC := /OSLAB/compiler/usr/bin/mips_4KC-gcc
INCLUDES := -I../include

all:
    $(CC) -c my_driver.S $(INCLUDES)
    $(CC) -c my_cal.c $(INCLUDES)
```

- C语言调用MIPS函数

  使用正确的寄存器（如 `$a0`-`$a3` 用于参数传递，`$v0`-`$v1` 用于返回值）

```
#include <asm/regdef.h>
```

```
#include <asm/cp0regdef.h>
#include <asm/asm.h>

.text

LEAF(_my_getchar) /*LEAF is defined in asm.h and LEAF
functions don't call other functions*/

    loop_begin:
        li s1, 0xb0000000
        lb t1, 0x00(s1)
        beq t1, zero, loop_begin
        nop

    sb t1, 0x00(s1)
    move v0, t1
    jr ra
    sb zero, 0x10(s1)

END(_my_getchar)            /*the function defined in asm.h*/

LEAF(_my_putchar)

    li s1, 0xb0000000
    sb a0, 0x00(s1)
    jr ra
    sb zero, 0x10(s1)

END(_my_putchar)

LEAF(_my_exit)

    li s1, 0xb0000000
    sb zero, 0x10(s1)

END(_my_exit)
```

- 寄存器是否需要加 `$`，建议还是看已有代码怎么写的，参照着描述

  > 某些汇编器（如 GNU 获 MARS）支持省略，但建议始终使用 `$`

- 如果要使用随编译工具链提供的工具 `readelf`，用法：

  `readelf -S <FILE>`

  `readelf -h` 查看 ELF 文件头

- `.text` 包含了可执行文件中的代码，`.data` 包含了需要 被初始化的全局变量和静态变量，而 `.bss` 包含了未初始化的全局变量和静态变量
- ~~(供快速查看)~~guidebook中，GCC 编译时指定链接方式

  `gcc -o test test.c -T test.lds -nostdlib -m32`

  ```
  SECTIONS
  {
  . = 0x10000;
  .text : { *(.text) }
  . = 0x8000000;
  .data : { *(.data) }
  .bss : { *(.bss) }
  }
  ```

- `EXPORT(function)` 表示声名一个函数

Makefile

- `=` 递归展开赋值（变量值被使用时展开）
- `:=` 立即展开赋值（变量值在赋值时立即展开），若被展变量尚未定义，则赋为 `0`
- Makefile其他文件夹，更多采用 `$(MAKE) --directory=$targetDir clean` 格式

  `$(MAKE)` 表示当前使用的 Make 工具

  不是直接 `make`，可以确保递归调用时使用相同的 Makefile 工具和选项
- 如何使用交叉编译的 `gcc` 和 `ld` 以及 `objdump`

  ```
  /* Makefile */
  CROSS_COMPILE := /OSLAB/compiler/usr/bin/mips_4KC-
  CC            := $(CROSS_COMPILE)gcc
  LD            := $(CROSS_COMPILE)ld
  OBJDUMP       := $(CROSS_COMPILE)objdump
  TARGET    := hello.c
  FUNC      := myhello.c

   all:
          $(CC) -c $(FUNC) -o myhello_c.o
          $(CC) -E $(TARGET) > hello_E.txt
          $(CC) -c $(TARGET) -o hello_c.o
          $(OBJDUMP) -DS hello_c.o > dump_c.txt
          $(LD) -o hello_exe -N hello_c.o myhello_c.o
          $(OBJDUMP) -DS hello_exe > dump_exe.txt
  ```

- Makefile 特殊变量

```
$@ # 当前规则的 target
$< # 当前规则的第一个依赖文件
$^ # 当前规则的所有依赖文件，空格分隔
```

ELF

- ELF文件示意图

```
|--------------------
|elf_header{
|       section_header_table_offset
|       program_header_table_offset
|       ...
|}
|--------------------
|       ...
|--------------------
|       section header table:
|               an array of section header{}
|       section header-1 {
|               section addr
|               ...
|       }
|
|       section header-2 {
|               section addr
|       }
|
|       ...
|--------------------
```

C语言

- 关于过程输出的测试，可以在 `init/init/c` 进行测试
- **小端 ⇔ 大端**

```
#define REVERSE_32(n_n) \
    ((((n_n)&0xff) << 24) | (((n_n)&0xff00) << 8) | (((n_n)
>> 8) & 0xff00) | (((n_n) >> 24) & 0xff))

#define REVERSE_16(n_n) \
    ((((n_n)&0xff) << 8) | (((n_n) >> 8) & 0xff))
```

- 在大小端输出的过程中，一定要注意哪些数据需要**转换一下**才能拿出来用

  比如在 ELF 输出每个**段**或**节**的数据时，总共需要 **4** 次转换

  ```
  // 以 0x12345678 为例
  // 大端序
  0x12 34 56 78
  // 小端序
  0x78 56 34 12
  ```

- 对于单独输出字符（不需要考虑格式时），可以用以下两种

  ```
  out(data, "(", 1);
  print_char(out, data, '(', 0, 0);
  ```

- **记得 `swtich` 的 `case` 后边要加 `break`**

往届代码参考

2022代码

`print.c`：输出结构体

```
  ms = va_arg(ap, struct my_struct *);

            length = PrintChar(buf, '{', 1, 0);
            OUTPUT(arg, buf ,length);

            int size = ms->size;
            length = PrintNum(buf, size, 10, 0, width, ladjust,
padc, 0);
            OUTPUT(arg, buf ,length);
            length = PrintChar(buf, ',', 1, 0);
            OUTPUT(arg, buf ,length);

            char c = ms->c;
            if (ladjust == 0){
                for(i = 0; i < width - 1; i++){
                    length = PrintChar(buf, ' ', 1, 0);
                    OUTPUT(arg, buf ,length);
                }
            }
            length = PrintChar(buf, c, 1, 0);
            OUTPUT(arg, buf ,length);
            if (ladjust == 1){
```

```
                for(i = 0; i < width - 1; i++){
                    length = PrintChar(buf, ' ', 1, 0);
                    OUTPUT(arg, buf ,length);
                }
            }

            if (size == 0){
                length = PrintChar(buf, '}', 1, 0);
                OUTPUT(arg, buf ,length);
            } else {
                length = PrintChar(buf, ',', 1, 0);
                OUTPUT(arg, buf ,length);
            }
            int *arr = ms->array;
            for (i = 0; i < size; i++){
                int temp = arr[i];
                negFlag = 0;
                if(temp < 0){
                    temp = -temp;
                    negFlag = 1;
                }
                length = PrintNum(buf, temp, 10, negFlag, width,
ladjust, padc, 0);
                OUTPUT(arg, buf ,length);
                if (i != size - 1) {
                    length = PrintChar(buf, ',', 1, 0);
                    OUTPUT(arg, buf ,length);
                } else {
                    length = PrintChar(buf, '}', 1, 0);
                    OUTPUT(arg, buf ,length);
                }
            }
            break;
```

`my_driver.S`：MIPS汇编定义函数

- `kseg1` 位置在 `0xA0000000`
- 设备映射地址为 `0x10000000`
- 所以设备的实际物理地址为 `0xB0000000`

```
#include <asm/regdef.h>
#include <asm/cp0regdef.h>
#include <asm/asm.h>
```

```
.text

LEAF(_my_getchar) /*LEAF is defined in asm.h and LEAF functions
don't call other functions*/

    loop_begin:
        li s1, 0xb0000000
        lb t1, 0x00(s1)
        beq t1, zero, loop_begin
        nop

    sb t1, 0x00(s1)
    move v0, t1
    jr ra
    sb zero, 0x10(s1)

END(_my_getchar)            /*the function defined in asm.h*/

LEAF(_my_putchar)

    li s1, 0xb0000000
    sb a0, 0x00(s1)
    jr ra
    sb zero, 0x10(s1)

END(_my_putchar)

LEAF(_my_exit)

    li s1, 0xb0000000
    sb zero, 0x10(s1)

END(_my_exit)
```

`my_cal.c`：调用汇编函数

```
#include <my_cal.S>

char a[1086], b[1086];
int ans, i, n = 1, j;
char x;

int main()
{
    while((x = _my_getchar())!='\n')
```

```
    {
        a[i++] = x;
    }
    for (j = i - 1; j >= 0; j--)
    {
        ans += n * a[j];
        n *= 10;
    }
    for(i = 0; ans; i++)
    {
        b[i] = ans % 2 - '0';
        ans /= 2;
    }
    for (j = i - 1; j >= 0; j--)
    _my_putchar(b[j]);
}
```

`Makefile`：编译汇编代码和C语言代码

```
CC := /OSLAB/compiler/usr/bin/mips_4KC-gcc
INCLUDES := -I../include

all:
    $(CC) -c my_driver.S $(INCLUDES)
    $(CC) -c my_cal.c $(INCLUDES)
```

`readelf.c`：判断段表是否重叠

其中 `ROUNDDOWN(a, n) = (((u_long)(a)) & ~((n) - 1))`

相当于是抽取了页号，页内偏移为 $\log n$ 位

```
#define align (1LL << 12)
int readelf(u_char *binary, int size)
{
    Elf32_Ehdr *ehdr = (Elf32_Ehdr *)binary;

    int Nr;

    Elf32_Phdr *phdr1 = NULL, *phdr2 = NULL, *phdr = NULL;

    u_char *ptr_ph_table = NULL;
    Elf32_Half ph_entry_count;
    Elf32_Half ph_entry_size;
```

```
    // check whether `binary` is a ELF file.
    if (size < 4 || !is_elf_format(binary))
    {
            printf("not a standard elf format\n");
            return 0;
    }

    // get section table addr, section header number and section
header size.
    ptr_ph_table = binary + ehdr->e_phoff;
    ph_entry_count = ehdr->e_phnum;
    ph_entry_size = ehdr->e_phentsize;

    // for each section header, output section number and section
addr.
    // hint: section number starts at 0.

    int flag = 0;
    for (Nr = 1; Nr < ph_entry_count; Nr++)
    {
        phdr1 = ((Elf32_Phdr *)(ptr_ph_table + (Nr - 1) *
ph_entry_size));
        phdr2 = ((Elf32_Phdr *)(ptr_ph_table + Nr *
ph_entry_size));
        long unsigned int r1 = phdr1->p_vaddr + phdr1->p_memsz,
l2 = phdr2->p_vaddr;
        if (ROUNDDOWN(r1, align) == ROUNDDOWN(l2, align))
        {
            printf ("Overlay at page va : 0x%x\n", ROUNDDOWN(l2,
align));
            if(l2 < r1)
                printf ("Conflict at page va : 0x%x\n",
ROUNDDOWN(l2, align));
            return 0;
        }
    }
    for (Nr = 0; Nr < ph_entry_count; Nr++)
    {
        phdr = ((Elf32_Phdr *)(ptr_ph_table + Nr *
ph_entry_size));
        printf("%d:0x%x,0x%x\n", Nr, phdr->p_filesz, phdr-
>p_memsz);
    }
    return 0;
}
```

2023代码

`printk.c`：实现 `sprintf()`

理解参数传递过程以及实现逻辑，可知不需要修改上层代码，只需要修改 `outputk` 即可

注意头文件的引用 `string.h` 以及完善 `string.c`

```
void outputbuf(void *data, const char *buf, size_t len) {
    char *temp = (char *)data + strlen(data);
    for (int i = 0; i < len; i++) {
        *temp = buf[i];
        temp++;
    }
    *temp = '\0;'
}

int sprintf(char *buf, const char *fmt, ...) {
  char *place = buf;
  va_list ap;
  va_start(ap, fmt);
  vprintfmt(outputbuf, buf, fmt, ap);
  va_end(ap);

  return (int)(buf - place - 1);
}
```

or

```
void outputbuf(void **data, const char *buf, size_t len) {
  for (int i = 0; i < len; i++) {
    **(char **)data = buf[i];
    (*data) = (*data) + 1;
  }
    **(char **)data = '\0';
}
// 主要不同体现在没有充当临时变量的char *temp，全程都是void **data取一层值后*data这个char *的自由移动，效果类似，不使用strlen也许效率会高一点？

// 由于outputbuf传参不同，所以这里给vprintfmt传参也需要发生改变，所以说这样的做法不太合适，相当于改变了函数结构
int sprintf(char *buf, const char *fmt, ...) {
  char *place = buf;
  va_list ap;
```

```
  va_start(ap, fmt);
  vprintfmt(outputbuf, &buf, fmt, ap);
  va_end(ap);

  return (int)(buf - place - 1);
}
```

2024代码

`print.c`

```
case 'P':
        case 'p':
            if (long_flag) {
                                x = va_arg(ap, long int);
                        } else {
                                x = va_arg(ap, int);
                        }
             tempX=x;
                        if (x < 0)
                        {
                                if (long_flag) {
                                        x = -x;
                                }
                                else {
                                        x = -x;
                                }
                                neg_flag = 1;

                        }
             temp = "(";
                 out(data,temp,1);
                        print_num(out, data, x, 10, neg_flag,
width, ladjust, padc, 0);
             if (long_flag) {
                                 y = va_arg(ap, long int);
                         } else {
                                 y = va_arg(ap, int);
                         }
              tempY=y;
              neg_flag=0;
                        if (y < 0)
                        {
                                if (long_flag) {
                                        y = -y;
```

```
                                    }
                                    else {
                                            y = -y;
                                    }
                                    neg_flag = 1;

                            }
                            temp = ",";
                            out(data,temp,1);
                            print_num(out, data, y, 10, neg_flag,
width, ladjust, padc, 0);
                int z = (tempX + tempY) * (tempX - tempY);
                z = z < 0 ? -z : z;
                neg_flag=0;
                temp = ",";
                out(data,temp,1);

 print_num(out,data,z,10,neg_flag,width,ladjust,padc,0);
                temp = ")";
                out(data,temp,1);
                            break;
```

`vscanfmt`

类似快读的方式

```
// init.c
void scanf_1_check() {
    printk("Running scanf_1_check\n");
    char arr[128];
    char arr2[128];
    int dec, hex;
    char ch,ch2,ch3,ch4;
    scanf("%c%c%c%s%s%c",  &ch,&ch2,&ch3,arr,arr2,&ch4);
    printk("%c%c%c%s%s%c",ch,ch2,ch3,arr,arr2,ch4);
}

void mips_init(u_int argc, char **argv, char **penv, u_int
ram_low_size) {
    scanf_1_check();
    halt();
}
// print.c
int vscanfmt(scan_callback_t in, void *data, const char *fmt,
va_list ap) {
```

```
    int *ip;
    char *cp;
    char ch;
    int base, num, neg, ret = 0;
    int reg=0;

    while (*fmt) {
        if (*fmt == '%') {
            ret++;
            fmt++; // 跳过 '%'
            do {
                in(data, &ch, 1);
            } while (ch == ' ' || ch == '\t' || ch == '\n'); //
跳过空白符
            // 注意，此时 ch 为第一个有效输入字符
            switch (*fmt) {
            case 'd': // 十进制
                // Lab 1-Extra: Your code here. (2/5)
                neg = 0;
                if (ch == '-')
                {
                    neg = 1;
                    in(data, &ch, 1);
                }
                base = 10;
                num = 0;
                while(ch>='0' && ch<='9')
                {
                    num = num * base;
                    num = num + (ch -'0');
                    in(data, &ch, 1);
                }
                ip = va_arg(ap, int*);
                if(neg == 1)
                    num = -num;
                *ip = num;
                break;
            case 'x': // 十六进制
                // Lab 1-Extra: Your code here. (3/5)
                neg = 0;
                if (ch == '-' )
                {
                    neg = 1;
                    in(data, &ch, 1);
                }
```

```
                base = 16;
                num=0;
                while (ch>='0'&&ch<='9'||ch>='a'&&ch<='f') {
                    if (ch>='a')
                    {
                        reg = ch-'a';
                        reg+=10;
                    }
                    else {
                        reg = ch-'0';
                    }
                    num=num*base;
                    num=num+reg;
                    in(data,&ch,1);
                }
                if (neg == 1)
                    num=-num;
                ip = va_arg(ap,int*);
                *ip = num;
                break;
            case 'c':
                // Lab 1-Extra: Your code here. (4/5)
                cp = va_arg(ap,char*);
                *cp = ch;
                break;
            case 's':
                // Lab 1-Extra: Your code here. (5/5)
                cp = va_arg(ap,char*);
                num = 0;
                while(ch!='\0'&&ch!='\t'&&ch!='\n'&&ch!=' ') {
                    cp[num]=ch;
                    in(data,&ch,1);
                    num++;
                }
                cp[num]='\0';
                break;
            }
            fmt++;
        }
    }
    return ret;
}
```

`printk.c`：定义了输入函数，由输入函数来调用 `vscanfmt`

并且修改了头文件 `print.h` 以及 `printk.h` 以及 `machine.h`

```
void inputk(void *data, char *buf, size_t len) {
    for (int i = 0; i < len; i++) {
        while ((buf[i] = scancharc()) == '\0') {
        }
        if (buf[i] == '\r') {
            buf[i] = '\n';
        }
    }
}

int scanf(const char *fmt, ...) {
    // Lab 1-Extra: Your code here. (1/5)
    va_list ap;
    va_start(ap, fmt);
    int result = vscanfmt(inputk, NULL, fmt, ap);
    va_end(ap);
    return result;
}
```