Lab 3 Report

思考题

Thinking 3.1

将进程页表中**虚拟地址** UVPT 对应的**页目录项**内容，设为该进程页表本身页目录的**物理基地址**，并将权限置为只读。

访问虚拟地址UVPT时，其对应页目录项映射到的页表是页目录本身，即**页目录自映射。**经过二级页表映射，访问虚拟地址UVPT的内容是一个页表项。

Thinking 3.2

进程结构体struct Env *e是load_icode函数的参数，在传入elf_load_seg函数时**强制转换**为void *data的参数形式，随后传入load_icode_mapper函数，并在其中转换回进程结构体。

```
struct Env *env = (struct Env *)data;
```

Thinking 3.3

- offset = 0 (页对齐)
- offset != 0
- bin_size < sgsize: 填充0

Thinking 3.4

虚拟地址：Elf32_Addr e_entry;　　// Entry point virtual address

Thinking 3.5

0-3号异常处理函数均在genex.S中，包括一BUILD_HANDLER宏。

- 0: handle_int

```
NESTED(handle_int, TF_SIZE, zero)
    mfc0    t0, CP0_CAUSE
    mfc0    t2, CP0_STATUS
    and     t0, t2
    andi    t1, t0, STATUS_IM4
    bnez    t1, timer_irq
    // TODO: handle other irqs
timer_irq:
    sw      zero, (KSEG1 | DEV_RTC_ADDRESS | DEV_RTC_INTERRUPT_ACK)
    li      a0, 0
    j       schedule        # in sched.c
END(handle_int)
```

- 1: handle_mod
  `BUILD_HANDLER mod do_tlb_mod` 展开宏为：

```
NESTED(handle_mod, TF_SIZE + 8, zero)
    move    a0, sp
    addiu   sp, sp, -8
    jal     do_tlb_mod      # in tlbex.c
    addiu   sp, sp, 8
    j       ret_from_exception
END(handle_mod)
```

- 2-3: handle_tlb
  `BUILD_HANDLER tlb do_tlb_refill` 展开宏为：

```
NESTED(handle_tlb, TF_SIZE + 8, zero)
    move    a0, sp
    addiu   sp, sp, -8
    jal     do_tlb_refill   # in tlbex.c
    addiu   sp, sp, 8
    j       ret_from_exception
END(handle_tlb)
```

Thinking 3.6

- STATUS_CU0: 启动CP0协处理器**用户态**功能
- STATUS_IM4: 可响应4号中断（时钟中断）
- STATUS_IEc: 开启中断

```
LEAF(enable_irq)
    li      t0, (STATUS_CU0 | STATUS_IM4 | STATUS_IEc)
    mtc0    t0, CP0_STATUS              # set Status Register
    jr      ra
END(enable_irq)
```

- KSEG1: 该段地址用于访问**外设**
- DEV_RTC_ADDRESS: RTC is a Real-Time Clock device. Used to retrieve the current time, and to cause periodic interrupts at a specific frequency.

向RTC的DEV_RTC_INTERRUPT_ACK偏移处写入0，用于清除RTC中断状态标志位，相当于确认已经处理了**时钟中断**。

```
timer_irq:
    sw      zero, (KSEG1 | DEV_RTC_ADDRESS | DEV_RTC_INTERRUPT_ACK)
    li      a0, 0                       # schedule(0)
    j       schedule
```

Thinking 3.7

- 时钟中断产生，触发MIPS中断，系统将PC指向0x80000080，跳转到`.text.exc_gen_entry`代码段
- 通过`.text.exc_gen_entry`代码段的分发，最终调用处理**中断异常**的`handle_int`函数
- 在`handle_int`函数中判断Cause寄存器的中断位，4号中断位引发的中断就是**时钟中断**
- 调用时钟中断服务函数timer_irq，最终跳转到调度函数schedule()中
- 若count == 0，代表当前进程的**时间片用尽**，即触发了时钟中断。从调度队列**头部**取出新进程，设置时间片count为该进程的**优先级**
- 调用env_run()**切换进程**

难点分析

3.1 env_init()

Env结构体数组envs: `struct Env envs[NENV] __attribute__((aligned(BY2PG)));`

管理的**进程上限**NENV：1<<10 = 1024

3.2 map_segment()

- page_insert: 将单个物理页面映射到虚拟页面
- map_segment: 将多个连续的物理页面映射到多个连续的虚拟页面(调用 page_insert 函数实现)

在一级页表基地址pgdir对应的两级页表结构中做段地址映射，将**虚拟**地址段[va,va+size)映射到**物理**地址段[pa,pa+size)，同时为相关页表项的权限位设置为perm。

3.3 env_setup_vm()

```
kseg1
kseg0
-----ULIM-----              /|\
Page Table of Process        |
-----UVPT-----             kuseg
Page Table of Kernel         |
-----UTOP-----               |
...                         \|/
```

kuseg是**用户态**下唯一可用空间，因此暴露[UTOP, UVPT)对应内核页表，使用户进程可共享访问。

3.4 env_alloc()

- env_id: mkenvid(struct Env *e)
- env_asid: asid_alloc(u_int *asid)

可使用封装好的try宏来抛出<0的**异常**，其与异常类型均位于error.h。

```
#define try(expr)
    do {
        int r = (expr);
        if (r != 0)
            return r;
    } while (0)
```

3.5 load_icode_mapper()

load_icode_mapper是elf_load_seg中的**回调函数**。

load_icode 函数会从ELF文件中解析出每个segment的段头ph，随后调用elf_load_seg函数将segment加载到进程的地址空间中。

数据在内存中的起始位置: src -- bin -- binary + ph->p_offset

3.7 env_create()

- `load_icode(e, binary, size);`
  binary为可执行文件，如**ELF头**结构体Elf32_Ehdr。
- `TAILQ_INSERT_HEAD(&env_sched_list, e, env_sched_link);`
  将e插入调度队列env_sched_link，其是双端队列，需使用TAILQ系列宏操作。

3.8 env_run()

`env_pop_tf(&curenv->env_tf, curenv->env_asid);` 的实现位于kern/env_asm.S，两个**参数**与汇编函数中的a0,a1对应。

```
LEAF(env_pop_tf)
.set reorder
.set at
    sll     a1, a1, 6
    mtc0    a1, CP0_ENTRYHI     # EntryHi[11,6] = ASID
    move    sp, a0              # set address of env_tf to sp
    j       ret_from_exception
END(env_pop_tf)
```

切换进程的最后准备工作，设置当前进程的ASID，跳转到kern/genex.S/ret_from_exception，通过env_tf的地址，恢复当前进程的**各寄存器**信息。

```
FEXPORT(ret_from_exception)
    RESTORE_SOME                # restore registers
    lw      k0, TF_EPC(sp)
    lw      sp, TF_REG29(sp)    # restore sp
.set noreorder
    jr      k0                  # jump to EPC
    rfe                         # restore SR
.set reorder
```

3.10 kernel.lds

```
. = 0x80000000;
```

“.”可以设置接下来的**节**的起始地址

3.12 schedule()

```
if (e != NULL && e->env_status == ENV_RUNNABLE)
```

对进程结构体e进行操作前，首先要确保e != NULL，否则会出现**访存异常**：address too low。

实验体会

本次实验主要涉及操作系统中的**进程**，我实现了进程的创建与运行，并通过学习操作系统**响应异常**的过程，体会了进程的切换与调度功能的重要意义。

通过本次实验对内核的搭建，我的操作系统具有了更好的**适应性**，能够处理一些异常情况。更重要的是，我拥有了第一个**运行**中的进程，赋予了操作系统生命力。