OS_lab3实验报告

思考题

Thinking 3.1 对物理地址和虚拟地址的理解

`e->env_pgdir[PDX(UVPT)] = PADDR(e->env_pgdir) | PTE_V;`这句代码实现了一级页表项（页目录）的自映射，虚拟地址中，UVPT开始的4MB是进程自己的页表，由于页表和虚拟地址之间的线性映射关系，页目录的第一个页表项应该对应了第一个页表的地址，也就是UVPT。

`PDX(UVPT)`是第一个二级页表地址的对应的页目录偏移量，`e->env_pgdir[PDX(UVPT)]`仍然是页目录这个页表的地址，因为页目录中的页表项对应了二级页表的地址。`PADDR(e->env_pgdir)`就是页目录的虚拟地址对应的物理地址，`PTE_V`代表只读权限。

上述操作实现了将UVPT虚拟地址映射到页表项对应的物理地址上。

Thinking 3.2 data 的作用

```
// lib/elfloader.c
const Elf32_Ehdr *elf_from(const void *binary, size_t size);
int elf_load_seg(Elf32_Phdr *ph, const void *bin, elf_mapper_t map_page, void
*data);
// kern/env.c
static int load_icode_mapper(void *data, u_long va, size_t offset,
u_int perm, const void *src, size_t len);
```

`data`是进程控制块的指针，`elf_load_seg`函数在调用`load_icode_mapper`函数的时候，将`data`传入其中，在进行虚拟地址到物理地址的映射的时候，需要`data`来提供`pgdir`和`asid`等数据。

```
page_insert(env->env_pgdir, env->env_asid, p, va, perm);
```

Thinking 3.3 elf_load_seg 的不同情况

- 首先，需要考虑需加载到的虚拟地址 va、该段占据的内存长度 sg_size 以及需要拷贝的数据长度 bin_size 都可能不是页对齐的。函数需要进行相应的页面偏移。
- 然后，调用回调函数 map_page，完成单页的加载，依次将段内的页映射到物理空间。
- 最后，如果该段在文件中的内容的大小达不到为填入这段内容新分配的页面大小，余下的部分用 0 来填充。

Thinking 3.4 EPC 的含义

虚拟地址。cp0中的epc寄存器存储了发生中断异常的时候cpu的pc寄存器指向的指令，cpu处理的地址都应该是虚拟地址，所以epc中也应该是虚拟地址。

Thinking 3.5 异常处理函数的实现位置

- `handle_int`函数在`kern/genex.S`中。
- handle_mod和handle_tlb都是通过genex.S文件中的宏函数BUILD_HANDLER实现的。

```
.macro BUILD_HANDLER exception handler
```

```
NESTED(handle_\exception, TF_SIZE + 8, zero)
    move    a0, sp
    addiu   sp, sp, -8
    jal     \handler
    addiu   sp, sp, 8
    j       ret_from_exception
END(handle_\exception)
.endm

.text

FEXPORT(ret_from_exception)
    RESTORE_ALL
    eret

NESTED(handle_int, TF_SIZE, zero)
    mfc0    t0, CP0_CAUSE
    mfc0    t2, CP0_STATUS
    and     t0, t2
    andi    t1, t0, STATUS_IM7
    bnez    t1, timer_irq
timer_irq:
    li      a0, 0
    j       schedule
END(handle_int)

BUILD_HANDLER tlb do_tlb_refill

#if !defined(LAB) || LAB >= 4
BUILD_HANDLER mod do_tlb_mod
BUILD_HANDLER sys do_syscall
#endif
```

Thinking 3.6 时钟的设置

- 时钟中断开启：MOS 中，时钟中断的初始化发生在调度执行每一个进程之前。首先，`env_asm.S` 中的 env_pop_tf 调用了宏 RESET_KCLOCK，然后`genex.S`中调用宏 RESTORE_ALL 中恢复了 Status 寄存器，开启了中断。

```
# env_asm.S
.text
LEAF(env_pop_tf)
.set reorder
.set at
    mtc0    a1, CP0_ENTRYHI
    move    sp, a0
    RESET_KCLOCK
    j       ret_from_exception
END(env_pop_tf)

# genex.S
FEXPORT(ret_from_exception)
    RESTORE_ALL
    eret
```

- 时钟中断关闭：异常分发的过程中，MOS 清除 Status 寄存器中的 UM、EXL、IE 位，以保持处理器处于内核态（UM==0）、关闭

  中断且允许嵌套异常。

```
# entry.S
mfc0    t0, CP0_STATUS
and     t0, t0, ~(STATUS_UM | STATUS_EXL | STATUS_IE)
mtc0    t0, CP0_STATUS
```

Thinking 3.7 进程的调度

使用时间片轮转算法，即时间片长度被量化为 N × TIMER_INTERVAL。在调度的过程中，用一个**调度链表**存储所有就绪（可运行）的进程，当内核创建新进程时，将其插入调度链表的头部；在其不再就绪（被阻塞）或退出时，将其从调度链表中移除。需要进行进程切换的有以下四种情况：

- 尚未调度过任何进程（curenv 为空指针）；
- 当前进程已经用完了时间片；
- 当前进程不再就绪（如被阻塞或退出）；
- yield 参数指定必须发生切换。

无需进行切换时，我们只需要将剩余时间片长度 count 减去 1，然后调用 env_run 函数，继续运行当前进程 curenv。在发生切换的情况下，我们还需要判断当前进程是否仍然就绪，如果是则将其移动到调度链表的尾部。

难点分析

本次实验主要包括了进程创建和中断异常与进程调度两部分。

进程创建

进程创建给过程大致如下：

- `env_init`函数初始化了`env_free_list`和`env_sched_list`两个列表，并将所有PCB插入`env_free_list`中。随后，创建一个“模板页目录”，设置该页将 pages 和 envs （即所有页控制块和所有进程控制块的内存空间）分别映射到 `UPAGES` 和 `UENVS` 的空间中。
- `ENV_CREATE_PRIORITY`宏调用`env_create`函数创建进程：
  - 调用`env_alloc`函数申请一个新的PCB，调用 `env_setup_vm` 初始化进程控制块的用户地址空间，然后对PCB内容进行初始化。
  - 设置PCB的的优先级以及状态。
  - 调用 `load_icode` 为进程加载 ELF 程序，同时使用 `TAILQ_INSERT_HEAD` 宏将进程控制块加入到调度队列中。

总之，在lab3中，`mips_init`函数会首先调用`env_init`函数完成进程的初始化，然后通过`ENV_CREATE_PRIORITY`宏（实质上是`env_create`函数）来创建进程。

中断异常和进程调度

- 在进程运行过程中，若中断异常产生，CPU会自动跳转到地址 0x80000180 处，也就是异常分发代码 `.text.exc_gen_entry` 的地方。
- `.text.exc_gen_entry` 这段代码会使用 `SAVE_ALL` 保存进程上下文，然后设置cp0_status寄存器，最后根据异常码跳转到相应的异常处理函数。
- lab3只考虑时钟中断，为0号异常，跳转到中断处理函数 `handle_init`，进而跳转到 `timer_irq` 函数处理。
- `timer_irq` 函数会调用调度函数 `schedule`：
  - 取出当前的进程控制块 `curenv`。
  - 当前进程可用的时间片减一。
  - 判断是否存在需要进程切换的以下四种情况：
    - 尚未调度过任何进程（curenv 为空指针）；
    - 当前进程已经用完了时间片；
    - 当前进程不再就绪（如被阻塞或退出）；
    - yield 参数指定必须发生切换。
  - 调用 `env_run` 函数运行当前进程。

以上是lab3中需要处理的进程调度问题，使用了时间片轮转算法，每个进程被分配一个时间段，称作它的时间片，即该进程允许运行的时间。如果在时间片结束时进程还在运行，则该进程将挂起，切换到另一个进程运行。

心得体会

lab3主要是完成了进程管理和中断异常的处理，有了lab2的基础，理解lab3的代码也变得比较容易，再加上lab3中很多实验代码都直接给出了，所以课下的主要难点还是在于从整体上把握进程创建和调度的过程，其中的逻辑是环环相扣的。在lab3中，有一个比较难以理解的地方是页目录的自映射，`env_setup_vm` 函数中用到了相关知识。虽然相关知识点在lab2中已经有提及，但是在本次实验中才在代码中体现出来。理解页目录自映射需要我们对二级内存管理机制有比较熟悉的理解，在这个地方我花费了比较大的功夫。