OS —— Lab3实验报告

22373407 王飞阳

一、思考题

Thinking 3.1

在`UVPT`以上到`ULIM`之间，有一个4MB的空间，正好可以满足对整个4GB进程空间的页目录自映射。`e->env_pgdir[PDX(UVPT)] = PADDR(e->env_pgdir) | PTE_V`中代码，左边表示页目录的`PDX(UVPT)`项，而右边则是页目录自身的物理地址，并将其标记为有效。这样做的结果是`UVPT`这个虚拟地址现在指向页目录自己，实现了自映射。这意味着进程可以通过访问`UVPT`这个虚拟地址来直接访问和修改其页目录和页表，从而不需要切换到内核模式。

Thinking 3.2

我们在`load_icode`的函数实现中找到以下代码段：

```
 //kern/env.c
 ELF_FOREACH_PHDR_OFF (ph_off, ehdr) {
        Elf32_Phdr *ph = (Elf32_Phdr *)(binary + ph_off);
        if (ph->p_type == PT_LOAD) {
            // 'elf_load_seg' is defined in lib/elfloader.c
            // 'load_icode_mapper' defines the way in which a page in this
segment
            // should be mapped.
            panic_on(elf_load_seg(ph, binary + ph->p_offset, load_icode_mapper,
e));
        }
    }
```

可以发现，`load_icode`调用`elf_load_seg`，并且 将函数中一个参数是`struct Env *e`（`e`指向当前进程控制块）传递给`elf_load_seg`的`data`参数。

作用：

在这种设计中，`data`参数通常用于在回调函数中携带额外的信息，比如当前的进程或环境的上下文（比如进程控制块指针），或者任何特定于调用上下文的数据。这允许`map_page`回调函数在不同的上下文中重用，具有更高的灵活性和通用性。如果没有`data`参数：`map_page`函数将无法接收到外部传递的上下文或状态信息，这限制了其在不同环境下的可用性和适应性。特定的操作，如访问当前进程的内存布局或特定资源，也将更加复杂或不可能实现，因为`map_page`函数将完全独立于其它执行上下文。

Thinking 3.3

- **权限设置：** 根据段的属性，函数设置页面权限。如果`ph->p_flags`包含`PF_W`（写权限标志），则页面权限将包括`PTE_D`（允许写操作）。所有页面默认包括`PTE_V`（表示页面有效）。
- **页面偏移处理：** 函数开始时，会计算虚拟地址`va`相对于页面大小`PAGE_SIZE`的偏移量`offset`。如果`offset`不为零，意味着段的起始地址没有对齐到页面边界。此时，函数需要处理这种非对齐的情况，将第一个部分的数据加载到一个可能已经部分填充的页面中。

- **正常的页面加载：** 一旦处理完初始的偏移问题，函数进入一个循环，加载段的剩余部分。在每次循环中，函数会映射一个完整的页面大小（或者如果是最后一个页面且数据不足一个完整页面，则只映射所需的部分）。这部分确保整个文件段的 `bin_size` 被逐页加载到内存中。
- **扩展页面映射以满足 `sgsize`：** 如果段的内存大小 `sgsize` 大于文件大小 `bin_size`，说明需要在内存中为该段额外分配空间，以达到 `sgsize` 指定的大小。这通常用于数据初始化和运行时需要的额外内存，例如 BSS 段。在 `bin_size` 加载完毕后，函数将继续映射额外的页面直到达到 `sgsize` 的要求，这些额外的页面将被初始化为零（NULL指向的数据表示无需复制数据，只需分配并清零内存）。

Thinking 3.4

虚拟地址

Thinking 3.5

- `handle_int：kern/genex.S`
- `handle_mod，handle_tlb：lib/genex.S`，抽象成了函数 `handle_\exception`

Thinking 3.6

时钟中断关闭：

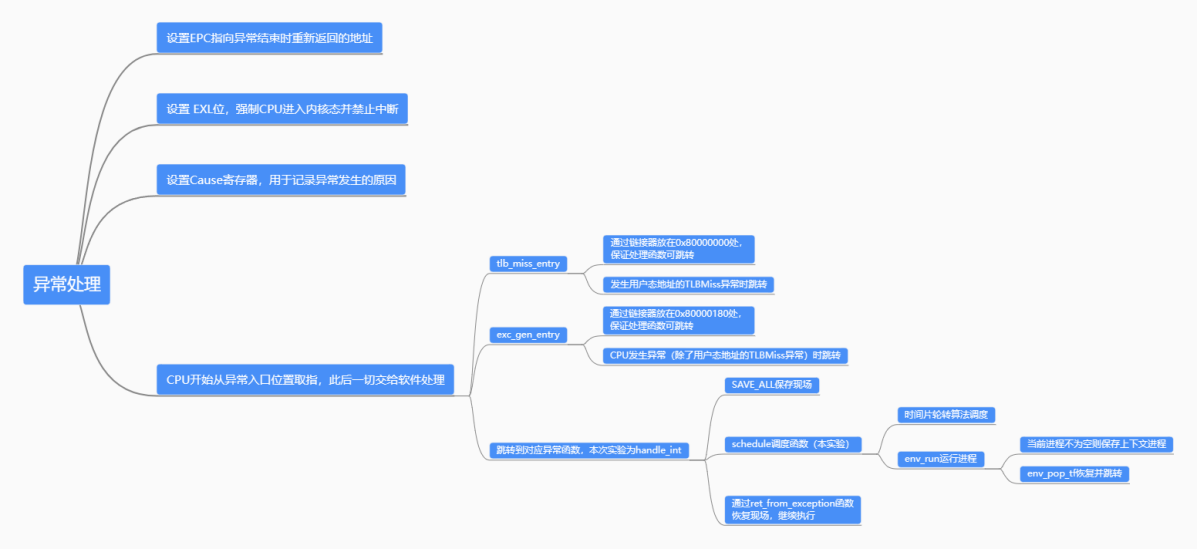
在 `entry.S` 文件中，有一个处理异常入口点 `exc_gen_entry`，该部分的代码在处理异常时（包括时钟中断）会先保存当前状态，然后修改 CP0_STATUS 寄存器的设置。具体地，它通过清除 `STATUS_UM`（用户模式位）、`STATUS_EXL`（异常级别位）和 `STATUS_IE`（全局中断使能位）来关闭中断。这样做可以防止在处理当前异常时发生新的中断，从而实现异常处理的可重入性。

时钟中断开启：

- 在 `genex.S` 文件中，`ret_from_exception` 宏通过 `RESTORE_ALL` 恢复所有寄存器的状态，并通过执行 `eret` 指令返回从中断或异常中恢复正常执行。在恢复所有寄存器的状态时，如果之前的 `STATUS_IE` 位被设置为使能，那么中断会在此时重新开启。因为 `RESTORE_ALL` 包括了状态寄存器 CP0_STATUS 的恢复，这意味着在进入异常处理前中断是开启的，恢复后仍然是开启的。
- 同样在 `env_asm.S` 文件中的 `env_pop_tf` 过程，处理结束时通过 `j ret_from_exception` 跳转到 `ret_from_exception`，这也会通过 `RESTORE_ALL` 和 `eret` 恢复中断设置。

时钟中断在进入异常处理时被关闭，以防止新的中断干扰当前的异常处理过程；而在从异常处理程序返回时，根据之前的中断状态恢复，如果之前中断是开启的，则在返回时重新开启中断。

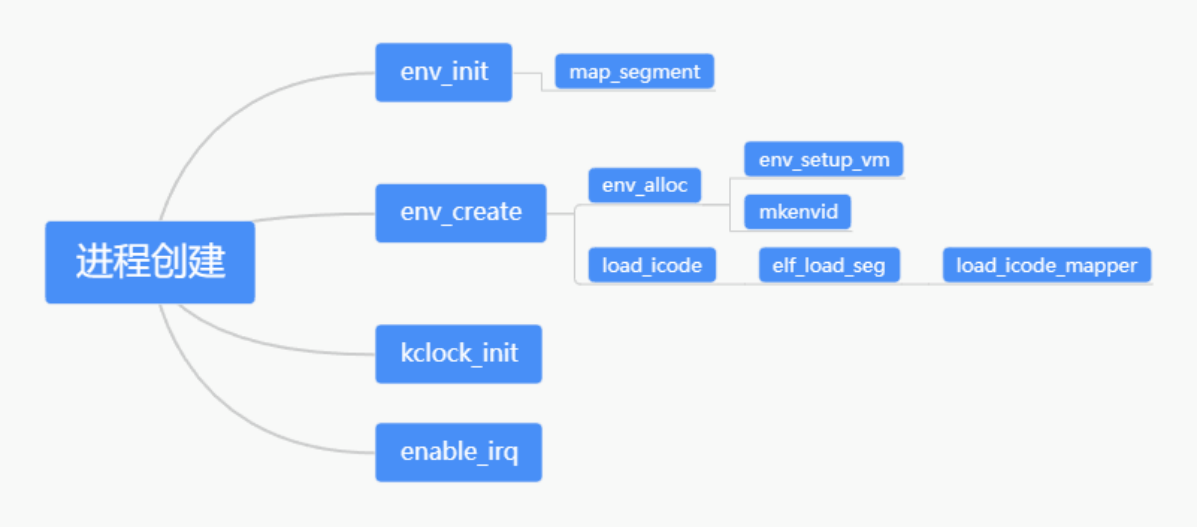
Thinking 3.7



二、难点分析

进程创建与进程加载

流程图



进程创建相关函数

`env_init`

位置：`lib/env.c`

说明：初始化进程创建相关条件

`env_create`

位置：`lib/env.c`

说明：创建进程

`env_alloc`

位置：`lib/env.c`

说明：

- 调用`env_setup_vm`函数，为进程分配页目录并完成部分映射
- 初始化对应PCB，主要包括`env_id, env_parent_id, env_status, env_runs, env_tf.cp0_status, env_tf.regs[29]`
- 将进程从空闲链表中移出，加入调度队列(这和物理内存管理的页链表管理体系很类似)

`env_setup_vm`

位置：`lib/env.c`

说明：

- 为进程的页目录分配物理页
- 以`UTOP`为界，将用户空间分别映射，映射的模板是之前虚拟内存初始化时得到的`boot_pgdir`
  - `0--UTOP`间的用户空间是可读写的，且各个进程在这部分的内容是各不一样的，因此在该函数中我们将其全部清零
  - `UTOP--UTPV`间的用户空间是存储`pages, envs`结构体数组的，这也是在之前虚拟内存初始化时完成映射的，这样在用户空间就可以访问到这两个数组。这部分对于所有进程都是相同的，因此我们以`boot_pgdir`为模板完成映射
  - `UVPT--UTOP`间的用户空间是用来自映射的，这部分4MB空间正好可以映射整个4GB虚拟内存空间，我们用以下代码来初始化这块空间：

    ```
    e->env_pgdir = pgdir;
    e->env_cr3 = PADDR(e->env_pgdir);
    e->env_pgdir[PDX(UVPT)] = e->env_cr3 | PTE_V;
    ```

  - `UTOP`以上的空间不属于用户空间，也是用户态无法访问的内存空间

进程加载相关函数

`load_icode`

位置：`lib/env.c`

说明：

- 分配物理页，并完成从用户栈空间到该物理页的映射
- 通过`load_elf`函数完成文件二进制数据到用户空间的映射
- 设置进程的入口，该入口地址通过上面的`load_elf`函数获得

`elf_load_seg`

位置：`lib/elfloader.c`

说明：分段调用`load_icode_mapper`函数完成映射

`is_elf_format`

位置：`lib/elfloader.c`

说明：判断文件是否符合`ELF`文件格式

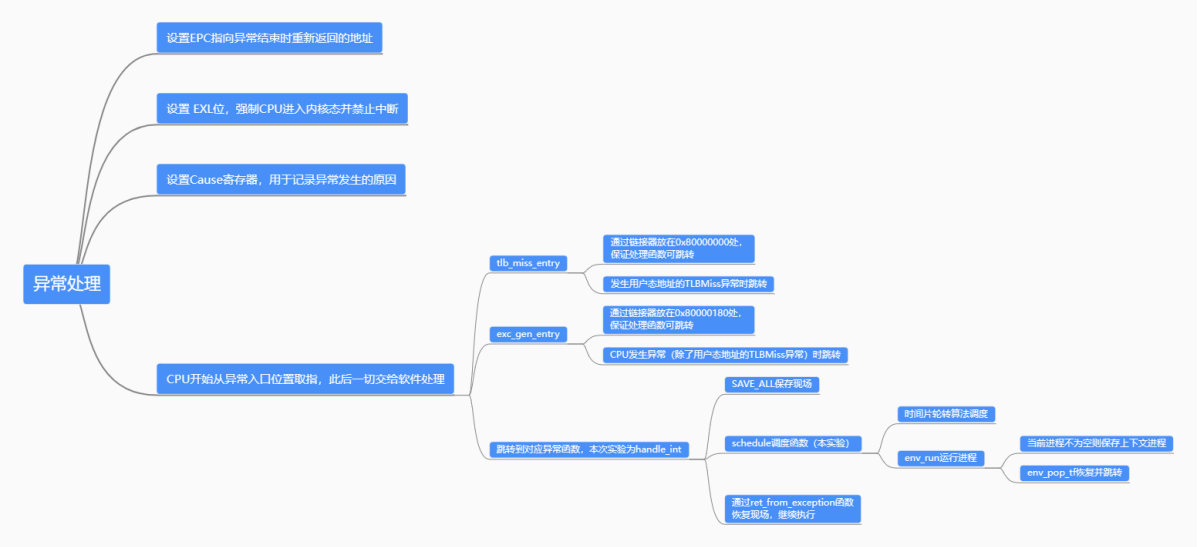
`load_icode_mapper`

位置：`lib/env.c`

说明：将文件二进制数据映射到用户空间

中断与异常

时钟中断处理图



异常处理相关函数

`handle_\exception`

位置：`kern/genex.S`

说明：异常处理函数

`timer_irq`

位置：`kern/genex.S`

说明：时钟中断的处理函数

`schedule`

位置：`lib/sched.c`

说明：调度函数

`env_run`

位置：`lib/env.c`

说明：运行进程

`ret_from_exception`

位置：`lib/genex.S`

说明：恢复现场

异常处理相关宏定义

`SAVE_ALL`

位置：`include/stackframe.h`

说明：保存现场

`RESTORE_SOME/ALL`

位置：`include/stackframe.h`

说明：恢复现场

三、实验体会

本次实验重点就是进程的创建和调度，以及简单异常的处理（时间中断）。

在lab3中，相关函数相比之前增加了很多，导致理清函数的调用关系比较困难。在阅读理解内核实验指导书之后，结合自己的理解，画了两个思维导图，以理清逻辑关系。

本实验中还遇到在`debug`方面的问题，希望下来可以通过查阅资料，增加自己对`debug`工具使用熟练度。