**Lab3 实验报告**

19373135 田旗舰

**一、实验思考题**

**Thinking 3.1**

**为什么我们在构造空闲进程链表时必须使用特定的插入的顺序？(顺序或者逆序)**

为了使链表中的Env块的顺序与envs数组中的Env块顺序一致，方便操作。

**Thinking 3.2**

**思考env.c/mkenvid 函数和envid2env 函数:**

**• 请你谈谈对mkenvid 函数中生成id 的运算的理解，为什么这么做？**

**• 为什么envid2env 中需要判断e->env\_id != envid 的情况？如果没有这步判断会发生什么情况？**

低10位为Env块在envs数组中的下标，其余高位为调用mkenvid函数的次数，这样的id既包含了Env块在envs中的顺序信息，又使得每个Env块的id互不相同。

防止在没有分配该envid或envid错误时调用该函数，此时后10位为0，错误地返回envs[0]。

**Thinking 3.3**

**结合include/mmu.h 中的地址空间布局，思考env\_setup\_vm 函数：**

**• 我们在初始化新进程的地址空间时为什么不把整个地址空间的pgdir 都清零，而是复制内核的boot\_pgdir作为一部分模板？(提示:mips 虚拟空间布局)**

**• UTOP 和ULIM 的含义分别是什么，在UTOP 到ULIM 的区域与其他用户区相比有什么最大的区别？**

**• 在env\_setup\_vm 函数的最后，我们为什么要让pgdir[PDX(UVPT)]=env\_cr3?(提示: 结合系统自映射机制)**

**• 谈谈自己对进程中物理地址和虚拟地址的理解**

因为实验中的操作系统是2G/2G内存布局，没有真正的内核进程，用户进程可能陷入内核态访问内核，因此需要boot\_pgdir访问内核空间。

UTOP是用户可以使用的空间的最高地址，ULIM是用户空间的最高地址。UTOP到ULIM的区域用户没有权限访问。

env\_cr3储存着页目录的物理地址，通过该语句完成页目录的自映射。

物理地址是真实的地址，所有进程共享；虚拟地址是虚拟的，每个进程独占的。

**Thinking 3.4 思考user\_data 这个参数的作用。没有这个参数可不可以？为什么？（如果你能说明哪些应用场景中可能会应用这种设计就更好了。可以举一个实际的库中的例子）**

不可以，这个参数是为了向内层函数传递参数。

qsort函数，需要传递compare函数。

**Thinking 3.5 结合load\_icode\_mapper 的参数以及二进制镜像的大小，考虑该函数可能会面临哪几种复制的情况？你是否都考虑到了？ （提示：1、页面大小是多少；2、回顾lab1中的ELF文件解析，什么时候需要自动填充.bss段）**

bin\_size<BY2PG-offset

bin\_size>=BY2PG-offset &&(bin\_size-BY2PG+offset)%BY2PG!=0

bin\_size>=BY2PG-offset &&(bin\_size-BY2PG+offset)%BY2PG=0

我都考虑到了:)

sg\_size>bin\_size时需要自动填充.bss段，即为其分配页面。

**这里的e->env\_tf.pc是什么呢？就是在我们计组中反复强调的甚为重要的PC。它指示着进程当前指令所处的位置。你应该知道，冯诺依曼体系结构的一大特点就是：程序预存储，计算机自动执行。我们要运行的进程的代码段预先被载入到了entry\_ point为起点的内存中，当我们运行进程时，CPU 将自动从pc 所指的位置开始执行二进制码。**

**Thinking 3.6 思考上面这一段话，并根据自己在lab2 中的理解，回答：**

**• 我们这里出现的” 指令位置” 的概念，你认为该概念是针对虚拟空间，还是物理内存所定义的呢？**

**• 你觉得entry\_point其值对于每个进程是否一样？该如何理解这种统一或不同？**

虚拟空间，因为虚拟空间是连续的。

是不是一样的取决于二进制文件首地址的定义，对于本实验中的user\_a和user\_b是一样的。都是从elf文件中读取得到的。

**Thinking 3.7 思考一下，要保存的进程上下文中的env\_tf.pc的值应该设置为多少？为什么要这样设置？**

env\_tf.cp0\_epc。cp0会保存中断发生时的pc，这样设置在处理完中断后可以回到中断发生前的位置继续执行。

**Thinking 3.8 思考TIMESTACK 的含义，并找出相关语句与证明来回答以下关于TIMESTACK 的问题：**

**• 请给出一个你认为合适的TIMESTACK 的定义**

**• 请为你的定义在实验中找出合适的代码段作为证据(请对代码段进行分析)**

**• 思考TIMESTACK 和第18 行的KERNEL\_SP 的含义有何不同**

TIMESTACK是用来保存上下文信息的栈指针

stackframe.h中

.macro get\_sp

mfc0 k1, CP0\_CAUSE

andi k1, 0x107C

xori k1, 0x1000

bnez k1, 1f

nop

li sp, 0x82000000

j 2f

nop

1:

bltz sp, 2f

nop

lw sp, KERNEL\_SP

nop

2: nop

.endm

get\_sp首先先判断是否发生的是4号中断引起的异常，然后选择相应的栈指针。

TIMESTACK是产生时钟中断异常时用的栈指针，KERNEL\_SP是非时钟中断异常用的栈指针。

**Thinking 3.9 阅读 kclock\_asm.S  文件并说出每行汇编代码的作用**

LEAF(set\_timer)

li t0, 0x01

sb t0, 0xb5000100#将1存入0xb5000100，开启实时钟

sw sp, KERNEL\_SP #将sp设为内核栈指针

setup\_c0\_status STATUS\_CU0|0x1001 0 #设置cp0状态

jr ra #返回上一级

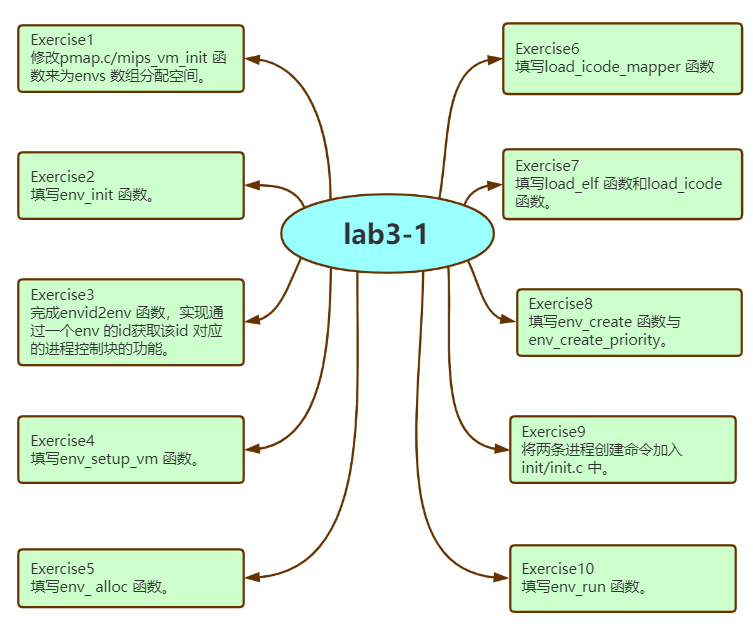
nop

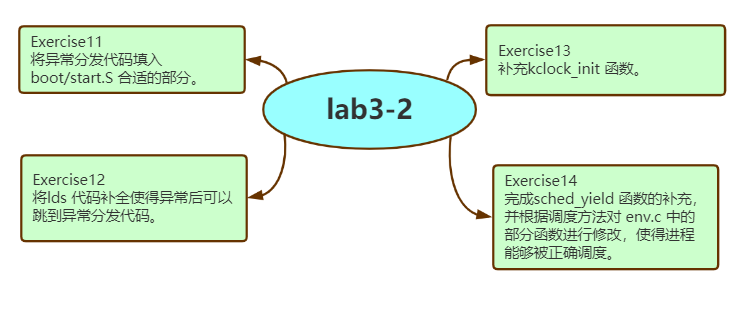
END(set\_timer)

**Thinking 3.10 阅读相关代码，思考操作系统是怎么根据时钟周期切换进程的。**

当时钟中断产生时，PC指向异常处理代码段，调用handle\_int函数，读取CPU\_CASUE和CPU\_SATUS到t0和t2中，将t0 and t2存入t0得到具体的中断号，判断如果是4号中断位则执行中断服务函数time\_irq，跳转到sched\_yield，通过时间片轮转算法从而切换进程。

**二、实验难点图示**



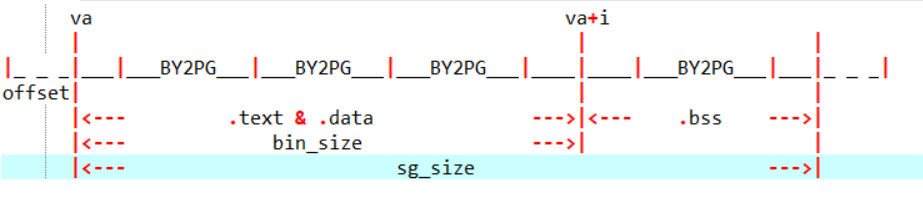


本次实验主要涉及进程的创建及运行、时钟中断的实现以及通过时钟中断实现进程切换。本次实验代码量相比之前的实验明显增大，涉及的函数也较多，而且缺乏便捷有效的本地测试方法，导致出bug的可能较大，而且较难定位。在完成实验的过程中，我仅仅是在load\_icode函数中在初始化新进程的栈时忘记将页面设置为可写，导致在执行env\_run时出现了bug，以致于花费了巨大的精力调试。但这并不是不会或者写错而导致的，仅仅是因为粗心，可见，本次实验的一个难点就是将每一个简单的、单独的函数都写对。

除了需要细心外，我认为本次实验的难点如下：

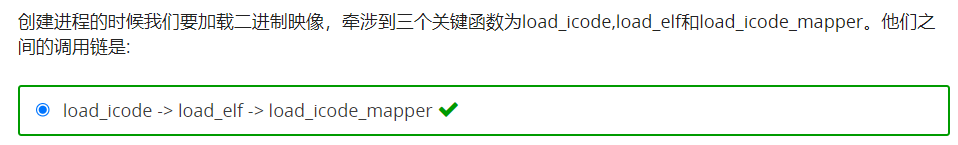
1.load\_icode\_mapper函数

由于va、bin\_size、sg\_size都可能不是BY2PG的整数倍因此要考虑3种不同的对齐情况，如同所示（图片来自讨论区）



因此，在编程的时候不能用一个while就解决，需要考虑offset，在循环外部或内部加入if语句单独判断边界情况。

另外，还需理解好load\_icode、load\_elf、load\_icode\_mapper之间的关系。



2.sched\_yeild函数

函数本身的编写并不是太难，只需要切换进程list以及将进程的priority减1即可，但很容易忽略curenv为NULL的情况，而当运行第一个进程时，curenv即为NULL，因此需要在if中添加条件判断。

**三、体会与感想**

本次实验总体难度较高，其中，将各个函数按照注释填写完整难度并不是太高，但真正理解每一个函数，并且能够在出现bug时定位问题所在并解决难度较大，以至于我在本次实验上花费了大量的时间。完成14个exercise我总共只花费了一个下午加两个晚上的时间，但在本地运行时却出现了bug，后面通过printf等方法debug花费了整整3天，让我第一次在这门课的学习中真正感受到了困难。但在发现bug改正后，回头想想其实也并没有那么难，其实我们的任务还是相对简单的，毕竟大量的代码已经写好，我们要做的只是阅读和理解，再进行补充。但是首先要建立一个整体的进程概念，清楚每一步在做什么，才能尽量在完成实验的过程中不出现bug。

**四、指导书反馈**

在sched\_yeild函数中，需要首先判断curenv为NULL的情况，即执行第一个进程的情况，但指导书上仅提到了当前进程，并没有强调当前进程为空的情况，感觉初次接触调度算法时很容易忽略，且不容易发现，最好能在指导书中加入一定的提示，考虑初始情况。

**五、残留难点**

虽然顺利完成了整个实验，但不看着实验代码还是无法构建出整个进程的布局，也不能详细地描述出进程创建和执行的流程，在整体布局的理解上还需要进一步熟悉。