思考题部分：

**思考题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)**

答：操作系统采用LRU算法，因此，最近使用过的空闲进程是放在靠近表头位置的，在调度的时候，将会最先被调度，优先级最高，使用LIST\_FIRSTF第一个被分配的是 envs[0]，所以我们从后往前插入。

**思考题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？

答：（1）首先获得该进程在env数组中的下标，作为id的低位，id的高位设置为

一个增加1的变量。高位代表当前存在的进程数量，低位代表进程在envs数组中的下标。为了达到ASID的20位。

（2）如果envid对应的进程不存在，就不应该得到对应的进程块。tlb的ASID区域只有20位。

**思考题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 step4 中我们为什么要让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**

答：（1）./include/mmu.h里面的内核布局，我们其实就是2G-2G模式，用户态占用2G，内核态占用2G。对于不同的进程而言，其虚拟地址ULIM以上的地方，映射关系都是一样的！这是因为这2G虚拟地址与物理地址的对应，不是由进程管理的，是由内核管理的。而我们将内核所能管理的那部分页表boot\_pgdir拷贝过去就足够了。

（2）UTOP =0x7f40 0000代表用户能够自定义数据部分的虚拟空间,ULIM=0x8000 0000代表用户可用虚拟空间的最大值。这一部分有页表、进程映射的虚拟地址部分，具有一定的数据结构和映射关系，且可以通过该部分的页表项访问到内核地址的一些部分。

（3）env\_cr3的值是pgdir的入口地址，即页目录入口地址，将pgdir[PDX(UVPT)]=env\_cr3就是使页目录中的某一个偏移项的值为页目录入口地址，完成自映射。

（4）进程中的虚拟地址需要先经过一次段式转换，将虚拟地址转换为线性地址，再通过访问页表才能得到相应的物理地址。

**思考题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）**

答：需要有这个参数， load\_icode\_mapper函数中将传进来的user\_data值赋给了env指针，后面page\_insert函数用到了env指针。

**思考题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？**

答：该段在文件中的内容的大小达不到ELF中该段在内存中所应有的大小(sgsize)，用0填充；第一次复制从bin指针所指的位置开始，需要判断offset与bin\_size大小关系；之后的每次复制都从bin+i-offset开始，需要判断BY2PG与bin\_size-i大小关系.

**思考题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？**

答：查阅kerelf.h文件知识虚拟地址；只要是同一elf文件下的就是一样的入口；如果不是同一个elf文件下就不是一样的。

**思考题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？**

答：env\_tf.pc设置成curenv->env\_tf.cp0\_epc异常pc的值，以便下次运行时从epc返回。

**思考题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**

答：

（1）实际上TIMSTACK就是0x82000000，TIMESTACK应该是时钟中断后的存储区。

（2） /include/stackframe.h文件里有：

li sp, 0x82000000

j 2f

nop

1:

bltz sp, 2f

Nop

lw sp, KERNEL\_SP

（3）KERNEL\_SP应当是系统调用后的存储区。其实KERNEL\_SP在第一个进程执行完之后就没更新过，因为我们第一个进程启动后，就再也没有给过内核进程控制权。

**思考题9.阅读 kclock\_asm.S 文件并说出每行汇编代码的作用**

答：开头是一个宏定义setup\_c0\_status，然后保存所有设置，读取当前CP0\_STATUS寄存器，相或，和异或，最后再写回寄存器，做恢复保存的设置。set\_timer部分前两步是向 0xb5000100 写入数值 1，偏移量为0x100 表示来设置实时钟中断的频率。存储当前栈指针，设置cp\_status寄存器，返回。

**思考题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。**

答：set\_timer函数中我们设置了映射时钟的位置，并且时钟频率为1.只要时钟周期一到，那么中断就会触发，实时钟对于 R3000 来说绑定到了 4 号中断上，故这段代码其实主要用来触发了 4 号中断；而 MIPS 将 PC 指向0x80000080，从而跳转到.text.exc\_vec3代码段执行；通过执行后会调用在 genex.S 中的handle\_int 函数，判断CP0\_CAUSE寄存器是不是对应的 4 号中断位引发的中断，如果是，则执行中断服务函数 timer\_ irq。在 timer\_ irq 里直接跳转到 sched\_ yield 中进行切换进程。

实验难点图示：

1. Lab3前两个内容和之前lab2相似，比较容易地完成。
2. env\_setup\_vm函数不是特别能理解其中一些步骤，step4和step4Pgdir的设置，对应着Mmu.h文件中内核地址分布图的理解。
3. Env\_alloc函数是进程申请和初始化的函数，也是进程创建运行的主要函数，对寄存器和新进程的赋值是一个难理解的点。
4. 对于加载elf镜像文件不懂是什么意思，不知道如何下手，这一点和进程的联系也不清楚。为新进程的程序分配空间来容纳程序代码，难道这个就是加载二进制代码吗？加载的作用是将 ELF 二进制文件读入内存。
5. Env\_run函数，这个函数是运行进程（其实是进程上下文保存和恢复）的函数，这里sp是TIMSTACK区，env\_pop\_tf 负责放东西到sp(这里是TIMESTACK)中去；而这开头的一段负责从sp里取出东西来。还有一个小坑的地方在于如果要env\_pop\_tf的时候，要先lcontext切换了页目录，否则是会出错的。env\_pop\_tf 是把env里的tf 存到寄存器里去。其中，lcontext实现在lib/env\_asm.S中，其作用是把传入的参数给存入mcontext中而mcontext定义在boot/start.S中，然后再使用env\_pop\_tf来把他之前保存的进程环境给还原，其实也就是恢复寄存器值。GET\_ENV\_ASID函数原码是#define GET\_ENV\_ASID(envid) (((envid)>> 11)<<6) 把envid中的高位（也就是是mkenvid中那个不断增加的id）取出，并左移6位。

6.进程中断，主要是优先级调度函数的编写，异常中断大部分是通过调用宏函数和汇编代码实现，进程切换是通过两个就绪状态链表实现插入和删除。

STEP

05

中断处理，进程

调度和切换

STEP

04

Env\_run函数

STEP

03

Load\_icode函数

STEP

02

Env\_alloc函数

STEP

01

env\_setup\_vm

函数÷

体会与感想：

本次lab3作业难度特别高、内容特别多，存在很多与理论课上没有提到的东西。实验几乎是在黑暗中慢慢摸着过来的，仍存在许多疑点。不过这个进程东西多，但是还是比较具体的，就相当于有东西在跑，在运行一样，你需要对它运行的轨迹进行操作，什么时候该1号，什么时候换2号。这也得益于lab2中内存地址映射的前提，虽然之前是比较抽象的，一直是映射关系，然后lab3真正意义上讲映射、内存、进程联系起来，还是有很大收获的。