## LAB3

这个实验开始设计到用户态的东西了。

注意，这个实验中的所有“环境”（env）相关的术语，就是和UNIX中的进程（proc）一一对应的，其实两者本质就是一个东西，但是因为提供的接口不同，所以在实验中给予区分。

这个实验接下来的很多实现，和UNIX的很多实验都有不少区别，虽然可以找到某种对应关系。

### Part A

首先要阅读新增的头文件，inc/env.h，弄清楚相关的数据结构的含义。

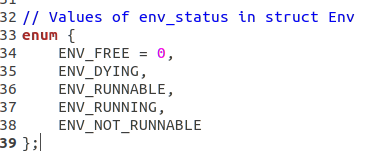
首先是结构体，env使用数组存放，数组的大小已经固定，所以整个系统中最多有NENV个进程运行（由宏可知为1024个）



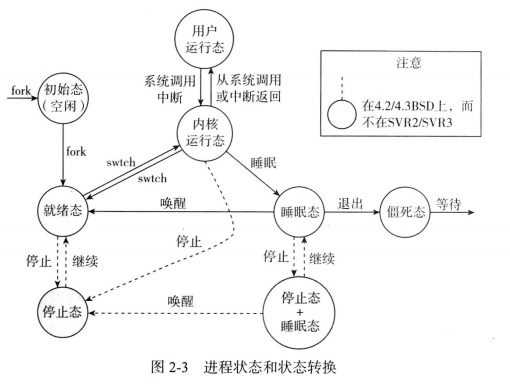
数组中有些环境结构没有被分配出去，它们都处于env\_free\_list上。

Curenv指针指向当前运行中的环境结构体变量。

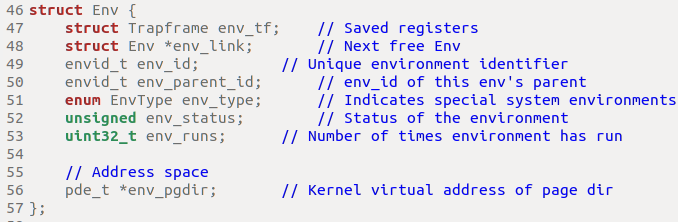
这个env数组，就对应于UNIX系统中的进程表，即proc结构数组（见《深入理解UNIX》2.3.4）



这几个枚举值表示了进程的状态，可以和书本上进行对照。



其中僵死态dying，就是等待父进程的wait回收返回值和其他信息，回收完后进程就彻底消失了。



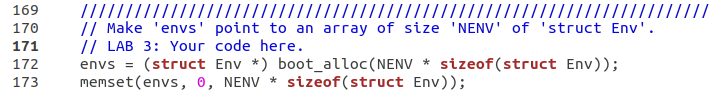
这里env\_tf里存放了寄存器等硬件信息，相当于描述线程。由于这里只有一个结构体变量，所以实现的系统只支持一个进程对应单个线程。而env\_pgdir变量就对应虚拟地址空间的概念。

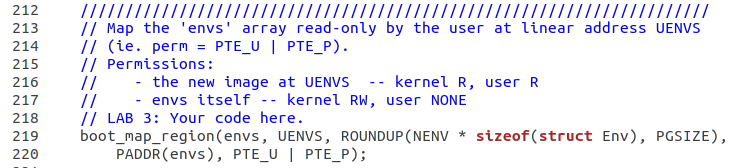
在UNIX中每个进程都有对应的内核栈（kernel stack），但是在JOS中只有一个全局的kernel stack（注意区别于内核区的栈，这个栈是用于处理用户请求的）。因为JOS中同一时刻只会有一个进程处于执行状态，因此不需要考虑内核的可重入问题。

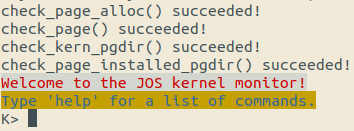
#### Exercise 1

Modify mem\_init() in kern/pmap.c to allocate and map the envs array. This array consists of exactly NENV instances of the Env structure allocated much like how you allocated the pages array. Also like the pages array, the memory backing envs should also be mapped user read-only at UENVS (defined in inc/memlayout.h) so user processes can read from this array.

You should run your code and make sure check\_kern\_pgdir() succeed







通过了测试！

注意这里不需要初始化空闲进程链表，而是在env.c中进行初始化。