Exercise 1

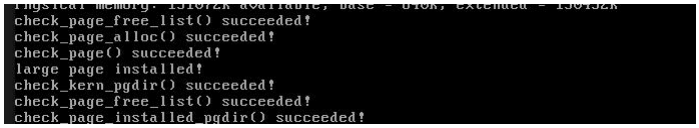
与在 lab2 中的 meminit 类似

先使用 envs = (struct Env *) boot_alloc(NENV * sizeof (struct Env));对 envs 进行空间分配

然后使用 boot_map_region(kern_pgdir,UENVS , PTSIZE , PADDR(envs) , PTE_U);对于 envs 的地址空间进行映射

其中需要注意在 boot_alloc 中可能会指向自己（因为 round up 的缘故），然后 memset 将自己清空，所以在初始化 nextfree 的时候将 nextfree 再加一个 pagesize

运行 make qemu，得到如图所示



Exercise 2

在 kern/env.c 中实现各个函数

env_init() 初始化 envs 并且把它们加入 env_free_list，每次都加在 list 的头部，并调用已经写好的 env_init_percpu()，其中 env_run 因为会调用 envs[0]所以需要倒着连接按照 free->0->1->2 这样进行连接，否则会报错异常页目录

env_setup_vm() 为 user_environment 申请一个页目录并使用内核的 pgdir 的内容来初始化其页目录

Region_alloc() 为 user_environment 把对应的物理地址分配好并进行映射，先按照提示对于 start 和 end 分别进行 roundup 和 roundend 然后对于在该范围内的每一个虚拟页分配并映射物理内存（先使用 pagealloc 然后使用 pageinsert）

Load_icode() 读取 ELF binary image 并写入 user address space，实现思路在注释中已经给出了，同时发现这个与 load/main.c 的思路差不多，仿照 load/main.c 先得到 ph 和 eph，然后设置 cr3 为当前进程的页表，从而可以使 memmove 和 memset 可以修改当前进程的虚拟地址，再循环判断 p_type 是否为 ELF_PROG_LOAD,如果是，则调用 region_alloc 并按照提示来进行读取，将 cr3 修改回来，最后设置对应的程序入口最后分配 USTACKTOP-PGSIZE 为 userstack。

Env_create() 调用 env_alloc 以及 load_icode 来导入 ELF 文件，先初始化一个 construct Env* e，然后使用&e 和 0 分别作为 store 和 pid 作为参数，调用 env_alloc 初始化 e，然后设置 e->env_type = type，最后调用 load_icode()加载 ELF 程序。

Env_run() 将一个程序在用户态跑起来,根据提示，分为两个 step，在第一个 step 中，判断是否是运行一个新的进程，如果是的话，就按照提示做一系列的操作，修改当前进程，修改进程状态，修改 cr3 等操作，在第二个 step 中，调用 env_pop_tf 来恢复当时进程保存的寄存器状态。

Exercise 3

读对应的章节就完事了

Exercise 4

在 trapentry.S 中使用给定的宏 TRAPHANDLER 以及 TRAPHANDLER_NOEC 来实现 handlerX 中具有每个特性的东西，也即 push 每个对应的错误号

然后在_alltraps 中实现它们共有的东西，也即把保存信息放入栈中，然后调用 trrap

然后实现 trap_init，先声明所有在 trapentry.S 中使用宏声明的 handlerX 函数，然后调用宏

SETGATE 来配置 IDT 表来实现绑定，其中 INTO， INT 3，BOUND 是允许软件中断的，所以对应的 dpl 需要设置为 3.然后即可通过测试，顺便为了后面的 exercise 考虑，在 dispatch 函数，对于 exception 进行了一个 switch 的分类

Exercise 5

当 Exception number 为 T_PGFLT 时，使之调用 page_fault_handler 即可

Exercise 6

当 Exception number 为 T_BRKPT 时，使之调用 monitor 即可

Exercise 7

与之前的方法大同小异，分别在 trapentry.S 以及 trap_init()中加入 SYSTEM_CALL 这一特殊的 Exception 类型，并使用宏 SETGATE 来更新 IDT 表，然后在 dispatch 中调用 syscall 并存入 eax 中作为结果,最后实现 syscall，根据 syscallno 调用对应的函数并返回即可

Exercise 8

晚点再做

Exercise 9

在/lib/libmain 中完成对应的代码，使得 thisenv 指向当前 env，实现相当简单，根据提示，调用 sys_getenvid 并调用 ENVX 宏得到在 envs 中的索引并从 envs 中取出来即可

Exercise 10

首先修改 inc/env.h 中的 env 结构，增加一个记录当前堆栈底部的指针 env_ds_bottom，在 env.c 的 load_icode 的最后初始化该值为 USTACKTOP-PGSIZE，然后只要申请多个页并把它们插入到页表的正确位置，也即调用 region_alloc(将其改为全局并引入头文件）即可，然后更新 env_ds_bottom 就完成了该 exercise

Exercise 11、12

实现 pagefault，并且当内核 page fault 的时候直接 panic，根据文档给出的机制，我们可以通过如下方法，判断 pagefault 是由内核引起的还是程序引起的：通过 tf_cs 的低位检测当前处于什么模式。首先通过上述机制，修改 kern/trap.c 中的 pagefault handler，使之在内核态陷入的话，就直接 panic，然后实现 user_mem_check 来判断某个内存 user 是否可以访问，通过循环加上 pgdir_walk 来得到对应的内存区域的 pte，并通过 pte 的 perm 得到是否与 perm 相符以及是否超过 ULIM 来判定是否可以访问，然后给 syscall 以及 kdebug 中根据提示加上对应的内存访问检查即可。