1. 创建 envs 数组，以及将其映射到虚存，方法和创建 pages 一样，换下参数就行了。
2. env_init():

   主要是链表操作和初始化 env_id 和 env_status 两个域

   env_setup_vm():

   根据提示 pp_ref 要加一，而其他的 UTOP 以上的页都不需要维护 pp_ref，所以直接将 kern_pgdir 中 UTOP 以上的页表项复制到 env_pgdir 中。

   region_alloc():

   根据提示把 va ROUNDDOWN，va + len ROUNDUP，然后可以算出要分配多少页，调用 page_alloc(0)分配，用 page_insert 插入页表项。

   load_icode():

   主要逻辑和 boot/main.c 中的 bootmain 差不多:读 elf 头，读程序元信息，分配页，复制程序内容到虚存。需要注意的是这里的虚存是 env 的虚存，所以在解虚存地址引用前要将页表切换至 env_pgdir，函数出口处再切换回 kern_pgdir。最后分配栈及将 trapframe 的 eip 指向程序入口。

   env_create():

   调用 env_alloc()和 load_icode()，设置 type

   env_run():

   把提示翻译成代码就行了
4. 关于错误码：https://wiki.osdev.org/Exceptions

   根据有无错误码使用不同的宏进行 handler 的声明

   _alltrap：

   1. 构造 trapframe，查看 trapframe 的结构，CPU 已经将一部分压入栈中，还需要压入%ds,%es 和通用寄存器，使用 pushl 压入%ds 和%es，使用 pushal 压入通用寄存器。
   2. 将 GD_KD 加载入%ds 和%es，movw $GD_KD, %ds 并不可行，需先 movw $GD_KD, %ax,再 movw %ax, %ds。
   3. Pushl %esp。
   4. Call trap

   trap_init():

   先声明函数 void NAME_handler();

   再使用 SETGATE 宏定义 idt 项

   SETGATE(idt[T_NAME], 0, GD_KT, NAME_handler, 0)
5. 在 trap_dispatch 中判断 tf->tf_trapno，如果等于 T_PGFLT, 则调用 page_fault_handler 并返回。
6. 将 T_BRKPT 对应的 idt 项的 DPL 设为 3，在 trap_dispatch 中判断 tf->tf_trapno 并调用 monitor。
7. trapentry.S 中加入一条 TRAPHANDLER_NOEC(syscall_handler, T_SYSCALL)

   trap_init()中加入对应的 idt 项，istrap 为 1，DPL 为 3

   trap_dispatch 中调用 syscall，参数在 tf->tf_regs 中，注意将 syscall 的返回值赋给 tf->tf_regs.reg_eax

   syscall.c 中根据 syscallno 分别调用函数即可，注意参数类型
8. 若采用 sysenter，那么用户的系统调用的调用链如下：

   lib/syscall.c:syscall-> 指令 sysenter->trapentry.S 中 wrmsr 将 sysenter 绑定到的 sysenter_handler 函数->kern/trap.c 中的包装函数->kern/syscall.c:syscall

需要修改的文件如下：

lib/syscall.c,kern/trapentry.S,inc/x86.h ,kern/trap.c

首先是 kern/trapentry.S 中增加一个 sysenter_handler 函数

该函数构造 trapframe 并调用 kern/trap.c 中的包装函数，此函数设置 curenv->env_tf 并调用 syscall。

Sysenter_handler 函数最后应调用 sysexit，调用前应设置 edx 和 ecx，如下：

RDX — The canonical address in this register is loaded into RIP (thus, this value references the first instruction to be executed in the user code). If the return is not to 64-bit mode, only bits 31:0 are loaded.

ECX — The canonical address in this register is loaded into RSP (thus, this value contains the stack pointer for the privilege level 3 stack). If the return is not to 64-bit mode, only bits 31:0 are loaded.

由于 sysenter 的调用约定 esi - return pc

ebp - return esp

所以代码为

```
movl %ebp, %ecx
movl %esi, %edx
sysexit
```

然后在 inc/x86.h 中增加 wrmsr 的定义，并在 kern/trap.c 中使用绑定 sysenter 指令，具体如下：

IA32_SYSENTER_CS (MSR address 174H) — The lower 16 bits of this MSR are the segment selector for the privilege level 0 code segment. This value is also used to determine the segment selector of the privilege level 0 stack segment (see the Operation section). This value cannot indicate a null selector.

IA32_SYSENTER_EIP (MSR address 176H) — The value of this MSR is loaded into RIP (thus, this value references the first instruction of the selected operating procedure or routine). In protected mode, only bits 31:0 are loaded.

IA32_SYSENTER_ESP (MSR address 175H) — The value of this MSR is loaded into RSP (thus, this value contains the stack pointer for the privilege level 0 stack). This value cannot represent a non-canonical address. In protected mode, only bits 31:0 are loaded.

所以代码是：

```
wrmsr(0x174, GD_KD, 0);
wrmsr(0x175, ts.ts_esp0, 0);
wrmsr(0x176, (uint32_t)&sysenter_handler, 0);
```

最后是修改 lib/syscall.c 使其调用 sysenter 指令

9. thisenv = envs + ENVX(sys_getenvid());
10. struct Env 中加一个 env_break 字段，根据定义 brk 是堆结束的地方，应该在加载镜像时初始化为 bss 段（data 段）的末尾。所以应在 load_icode 中初始化，data 段可以用 ph->p_flags 来判断。Syscall 中用 region_alloc 分配页即可。
11. !(tf->tf_cs & 3)为真则在内核模式。

    User_mem_check，用 pgdir_walk 获得 pte 然后判断权限。

    Sys_cputs 需要加入参数检查

13. 代码大致都给了，稍微修改一下，让 call_fun_ptr 调用一个全局函数指针，最后出来有 bug，发现是 lret 的时候出了 general protection exception，检查了栈发现 eip 和 cs 都是正确的，gdt 的 entry 也恢复了，最后也没搞懂怎么回事。