Lab3 笔记

2019年4月15日 19:00

Exercise1

- 修改mem_init()函数，申请并映射envs数组，并且envs权限为用户只读
- 用boot_alloc分配空间
- 用boot_map_region映射虚拟地址UENVS

Exercise 2

env_init();

- 初始化envs数组，并把envs添加到env_free_list

env_setup_vm();

- 设置环境页目录e->env_pgdir并且初始化内核部分页目录。
- 代码已经申请一页的空间，将e->env_pgdir指向改地址即可，
- 根据代码注释提示可知所有envs的虚拟地址空间在UTOP以上都是一样的，所以只需将UTOP以上的空间设置与kern_pgdir相同即可。

region_alloc();

- 分配并映射环境物理内存。
- va向下取整为起始页地址，va+len向上取整为末尾地址
- for循环申请页，并用page_insert映射到对应虚拟地址

load_icode()；解析ELF二进制文件，把文件内容装载进新的用户环境，该函数只有在内核初始化时，第一个用户环境前调用

- 用ELF_MAGIC判断是否是一个有效的elf结构类型
- 只需要装载ph->p_type == ELF_PROG_LOAD
- 段虚拟地址ph->p_va
- 内存大小ph->p_memsz
- 文件大小ph->p_filesz
- 从binary+ph->p_offset起的内容需要拷贝到ph->p_va，其余内容清零
- 所有页对用户可读可写
- 为用户栈初始化分配一页内存

env_create();

- 用env_alloc分配一个用户环境
- 然后用load_icode加载elf二进制文件
- 该函数仅在内核初始化的时候调用

env_run();

- curenv状态为RUNNING，置为RUNNABLE
- 将curenv换为e
- 用env_pop_tf恢复寄存器状态

interrupts通常由异步时间引起，比如硬件的通知，exception由同步事件引起，比如除0异常。Interrupt vector值在0~31为exception，大于31部分为软件中断或外部硬件引起。

Exercise4

- 编辑trapentry.s和trap.c，完成IDT的跳转。
- TRAPHANDLER为有错误码的跳转，TRAPHANDLER_NOEC为无错误码的跳转。用这两个宏定义设置好每个handler
- 根据要求完成_alltraps，把GD_KD装载到ds和es寄存器，pushal将相应的寄存器压栈，pushl esp传递一个指向trap frame的指针作为trap()的参数。并调用trap()函数
- 最后在trap_init()内申明每个entry对应的函数名，然后用SETGATE配置idt表。

Exercise5

修改trap_dispatch，实现分发页错误到page_fault_handler。

只需要在trap_dispatch内加一行(判断错误类型如果为T_PGFLT，则调用page_fault_handler)，此时make grade可得50/90分

```
1  if(tf->trapno == T_PGFLT)
2     page_fault_handler(tf)
```

Exercise6

- 修改trap_dispatch()函数，当异常为breakpoint时，调用monitor函数。

Exercise7

- 添加system call相关的handler，与之前的idt入口代码相似，优先级设置为3(用户)
- 首先在trapentry.S和trap_init()内，添加T_SYSCALL的entry和对应的函数。
- 修改trap_dispatch()函数，分发T_SYSCALL异常，调用syscall()函数
- 实现syscall()函数，根据传入参数去调用响应的函数，无效参数则返回-E_INVAL

Exercise8

- 与7相同的功能，只是实现不同，用sysenter和sysexit指令来实现。
- 在trapentry.S内添加sysenter_handler代码，与7不同的是，sysenter_handler的代码直接调用syscall()函数，不用trap_init()来进行分发。
- 把eax edx ecx ebx edi压栈进行syscall()的传参，调用syscall()后，执行sysexit指令
- 在inc/x86.h文件内添加wrmsr的代码

- 在trap_init()函数内用wrmsr来设置sysenter_handler()函数入口
- 在lib/syscall.c中用sysenter指令来实现system call，返回的地址设置到esi中

Exercise9

- 在libmain()中添加代码，让user/hello输出"i am environment 00001000"
- 根据libmain()的注释，把thisenv指向envs[]即可

Exercise10

- 完成sbrk()函数，该函数扩展heap，并返回当前程序需断点。
- 在struct Env结构中添加变量uintptr_t env_heap来记录heap的底部。
- 修改load_icode 代码，因为该函数第一次申请了内存，所以在该函数内初始化env_heap为USTACKTOP-PGSIZE
- 最后实现sbrk，用region_alloc申请内存，然后更新env_heap

Exercise11

- 内存保护，用户模式不可以访问某些内存
- 如果在内核模式下发生page fault，则panic报错，用tf_cs低两位检测当前处于什么模式
- 实现user_mem_check()函数，主要检查地址是否超出ULIM，页权限是否与perm符合
- 补全Sys_cputs，用user_mem_assert检查权限
- 最后在kdebug.c中对usd, stabs, stabstr进行地址检测

Exercise13

- 用ring0特权级来调用fun_ptr指向的函数，然后返回到ring3特权级。
- 用GDT来设置一个调用门(call gate)，call gate可以跨特权调用函数，在该call gate设置一个新的函数
- 在这个新的函数内去调用fun_ptr指向的函数
- lcall调用这个新的函数
- 函数内调用fun_ptr，并lret返回
- 注：该部分代码，好像已经被助教写好了，不需要做什么修改即可跑通测试。