Lab3 Solution

516030910006 方俊杰

Exercise 1.

修改 `kern/pmap.c` 中的 `mem_init` 函数，用 `boot_alloc` 为定义在 `kern/env.c` 的环境结构数组 `envs` 分配空间，再用 `boot_map_region` 映射到 `UENVS`，类似上个lab中对 pages 的操作。
注：这里似乎有个问题，仅当前代码无法 check_kern_pgdir(), 会 panic 提示非法 kernel address，对 memset 手动修改进行发现，在 `kern_pgdir` 为 0xf01a1000 的时候 `memset(kern_pgdir, 0, 0)` 语句将 kern_pgdir 指针所在内存也置 0 了，导致了 panic 的发生，原因无法理解，可能我有什么地方做错了。但是将 boot_alloc 函数中初始化 `end` 的部分直接加大 PGSIZE， 使得初始时 `kern_pgdir` 即为 0xf01a2000 即能通过测试。

Exercise 2.

编写完善 `kern/env.c`，实现对 env 的管理。

1. `env_init` 函数初始化 envs，设置 env_id 为 0，env_link 指向 `env_free_list`。根据要求，env 在 `env_free_list` 的顺序要和在 envs 数组中相同，故应逆序处理每个 env 结构元素，使得 `env_free_list` 指向 env[0]。
2. `env_setup_vm` 函数分配一个 `page directory` 给 new env 并初始化 kernel 部分，根据 hint 设置环境的 `env_pgdir` 域为 page2kva(p)。使用 `kern_pgdir` 作为模板映射。
3. `region_alloc` 函数分配并映射物理内存，使用 page_alloc 分配内存，用 page_insert 映射到 [ROUNDDOWN(va),ROUNDUP(va+len)]的部分
4. `load_icode` 解析并加载 ELF 二进制image，类似 `boot/main.c` 对 ELF 的处理，但是不需要读取磁盘。根据 hint 把 binary 转换为 Elf 结构，获取到 Proghdr，先用 region_alloc 分配内存，再用 memcpy 拷贝 binary 中相应部分到分配的内存。注意先用 lcr3(e->env_pgdir) 使用对应环境的页表。然后设置环境的入口 entry point。
5. `env_create` 创建环境并加载 image，使用上述 env_alloc 和 load_icode 函数即可
6. `env_run` 在 user mode 运行给定的环境，即调度它，根据 hint 判断 curenv 与 e 的关系，设置两者的运行状态 `env_status`，状态定义在 `inc/env.h`,更新环境，并加载页表，调用 `env_pop_tf`运行对应env。

Exercise 3.

阅读关于 Exception 和 Interrupt 的内容。

Exercise 4.

编辑 `trapentry.S`，利用给出的宏，为每一个 定义在 `inc/trap.h` 中的 trap 用汇编的方式声明一个中断处理函数，即 entry point，注意有是否需要 error code 的区别，可以在这里找到。然后定义一个 `_alltraps` 入口，根据提示编写其行为，最后调用 trap（不返回）。
另外在 `trap_init` 中初始化中断描述符表IDT，为每一个 trap，使用定义在 `inc/mmu.h` 的 SETGATE 设置中断处理函数

Question

1. 因为不同的 exception/interrupt 会存在 error code 是否需要push 的情况，若用同一个 handler 无法做到
2. 在SETGATE 宏中介绍，idt 中的 gd_dpl 设定 DPL(描述符权限级别)，规定了trap发生来源软件是不能直接使用 int 指令产生 gd_dpl 为 0 的 trap，只能由硬件发出。如果想要让他能发出，就把 14 即 T_PGFLT 对应的 SETGATE 的 gd_dpl 设为 3，就可以发生了，但是实际上不应该这样做，这会导致软件能够告诉内核发生了 page fault，破坏内核对内存的管理。

Exercise 5.

在 `trap.c` 中已经提供了 `page_fault_handler`，修改 `trap_dispatch` 分发 PF 到其中。
只要当 `tf->tf_trapno == T_PGFLT` 时调用 上述函数即可。

Exercise 6.

根据要求，在 Braeakpoint exception 发生的时候，使用内核 monitor 作为 debugger，故类似 exercise 5，只要当 `tf->tf_trapno == T_BRKPT` 时调用 monitor(tf) 即可。

Question

3. `int3` 最初也只时产生了 `general protection fault`，而不是 `break point exception`。理由类似，软件不同发出 gd_dpl 设为 0 的中断。修改 `trap_init`，将 T_BRKPT 对应的 gd_dpl 设置为 3 即可。
4. 为什么要有这种机制呢？因为 interrupt 机制是操作系统工作的最重要机制，如果user可以用软件随意引发中断或者异常，可能会对kernel 造成严重后果，比如像 `softint` 软件随意引发 page fault会使得内核无法管理页表，但是 int3 可能不会对环境造成破坏。

Exercise 7.

处理 SYSTEM CALL，类似上述添加中断处理函数的方法，在 `trapentry.S` 添加一条宏使用对应 `T_SYSCALL`；在 `trap_init` 添加对应的 idt 项，注意用户可发出，故 gd_dpl 设为 3；在分发函数 `trap_dispatch` 判断 `tf->tf_trapno == T_SYSCALL` 时调用定义在 `kern/syscall.c` 的 syscall, 注意参数顺序和返回值保存位置。
接下来编写上述 syscall 函数，根据定义在 `inc/syscall.h` 的 syscallno，分发处理不同的 syscall, 暂时不需要实现。

Exercise 8.

要求使用 `sysenter sysexit` 实现 syscall。根据提示,在 `trapentry.S` 增加一个 `sysenter_handler` 以调用 syscall(),注意 push 的参数顺序,另外类似 TRAPHANDLER 需要用汇编声明 global 和 type, 否则其他文件找不到。根据参考还需要在 `kern/init.c` 使用 `wrmsr` 设置 MSR。最后需要修改 `lib/syscall.c` 的 syscall 函数支持上述指令, 需要用 push/pop 保护通用寄存器,并使用给出的指令确定返回地址。完成的功能与 7 中相同。

Exercise 9.

根据要求完成 `lib/libmain.c` 中对 thisenv 变量的赋值, 赋值为 &envs[ENVX(sys_getenvid())] 即可

Exercise 10.

实现 sbrk 函数, 要求能够在 user 要使用大内存的时候通过 syscall 调用 sbrk 分配内存到 heap。先修改 Env 结构，添加一个 `int32_t env_heap` 表示 heap 底部,在 env.c 的 `load_icode` 函数中初始化环境的该变量。
然后实现 `kern/syscall.c` 的 `sys_sbrk` 函数，类似 env.c 中的 region_alloc 函数。调用 page_alloc 分配 e->env_heap - inc 到 e->env_heap 的内存范围,并用 page_insert 插入到 pgdir 即可,注意返回值是当前 env_heap。

Exercise 11.

需要对 user 传递给 kernel 的内存地址进行检查以防止非法访问。
首先,在 kern/trap.c 的 page_fault-handler 中 处理 `kernel page fault`,若 tf->tf_cs 的 低 两位是 0，说明是在 kernel 中发生了 page fault,应该 panic。
然后实现 pmap.c 中的 `user_mem_check` 函数检查 user 传递给 kernel 的指针是否合法,根据注释完成，注意范围，可能要检查 len/PGSIZE +2 数量的 page。若 va 大于 ULIM , pte 不存在或者 perm 不对都应认为不合法。
最后使用 `user_mem_assert` 在 syscall.c 的 sys_cputs 中检查字符串指针，若发现异常访问直接 destory env。使用 user_mem_check 在 kdebug.c 检查 usd、stabs、stabstr 是否合法。
backtrace 时发生 page fault 的原因是此时处于 user mode,但是回溯访问到了调用 libmain 函数的 entry.S 中,是用户不可访问的。

Exercise 12.

上一个练习的实现已经完成了对打印字符串的指针地址检查

Exercise 13.

根据 evilhello2.c 中的 hint, 用 sgdt 保存 GDT 到内存,用 `sys_map_kernel_page` 映射到用户空间,在 GDT 中设置 `CALLCATE`存 wrapper 函数,内联汇编调用 `lcall` 指令, 调用对应函数,然后恢复并用 `lret`返回。添加的 wrapper 函数调用了期望的 evil 函数,然后恢复了原本的 entry 并返回。
代码似乎已经存在,但 wrapper 函数存在问题,内联汇编 `popl %ebp` 不够,应该为 `leave` 或者添加 `movl %ebp, %esp`。
结果是运行时会有一个 `IN RING0!!!` 跟着是一个 page fault,因为在 ring0 的调用会打印,而在 ring3 的会触发 page fault。