Lab4 笔记

2019年5月4日 11:17

partA

Exercise1

- 实现mmio_map_region，该函数用来进行mmio映射的
- 对地址进行PGSIZE对齐，从base开始映射
- 不能超过MMIOLIM，映射成功后，修改base值

Exercise2

- 修改pge_init()函数
- 保留MPENTRY_PADDR起的一页地址，用来复制和运行AP bootstrap代码。

Exercise3

- 修改mem_init_mp()函数
- 为每个cpu分配独立的栈
- 栈大小为KSTKSIZE，每个栈之间还有大小为KSTKGAP的保护页

Exercise4

- 初始化每个cpu的TSS和IDT
- 因为每个cpu的栈不同，所以需要根据每个cpu的id初始化ts.ts_esp0
- 根据每个cpu的id初始化idt对应的项

Exercise5

- 多核同时运行，会有竞争状态，最简单的解决办法是一个大的全局锁
- 环境进入内核态时加锁，退出时放锁
- 根据提示需要用lock_kernel()和unlock_kernel()在4个地方加锁/放锁

Exercise6

- 实现rr循环调度，并修改syscall()分发sys_yield()
- 修改init.c运行三个user/yield.c环境
- 从当前环境后一个环境开始调度，寻找一个状态为runnable的环境，如果当前环境为空，从0开始寻找
- 如果没有找到合适env，并且当前环境仍然在当前cpu运行，则继续运行当前env

Exercise7

Sys_exofork()

- Sys_exofork()函数，创建一个新的environment，用env_alloc()创建

- 父进程为当前environment
- 寄存器状态和当前environment相同，同env_tf复制实现
- 分配出错，返回对应的错误码

Sys_env_set_status()

- 为环境设定状态，ENV_RUNNABLE和ENV_NOT_RUNNABLE
- 成功返回0，错误返回错误码

Sys_page_alloc()

- 分配一页内存，并映射到va，设定权限
- 页初始化为0

Sys_page_map()

- 复制映射关系，新老地址都映射到相同的物理地址
- 成功返回0，失败返回小于零的错误码
- 检查perm是否有效，检查读写权限，srcenv_id和dstenv_id对应的env是否存在，检查srcva是否在对应env中存在

Sys_page_unmap()

- 取消映射关系

注：由于执行dumpfork时，该函数**通过trap来调用syscall**，所以我**注释了以前lab3 由sysentry和sysexit**实现的进入syscall方式。还需要**修改lib/syscall.c代码，直接调用syscall**，不走sysentry。trap没有返回值，**dumpfork用eax寄存器存返回值**，所以trapdispatch函数内，将eax赋值为syscall返回值。并且通过trap调用syscall直接返回，如果break;继续执行会调用destroy_env，导致下一次进入找不到env。做完以上

partB

Exercise8

- 实现sys_env_set_pgfault_upcall()函数
- 调用成功返回0，否则返回错误码
- 把对应env的env_pgfault_upcall设置为func，这样用户级别的page fault就会调用该函数

Exercise9

- 用户发生页错误
- 检查页错误来源：两种来源，一种是用户，一种是page_fault_handler
- 检查写权限
- 设置栈上的UTrapframe结构
- 运行env_pgfault_upcall

Exercise10

- 完成lib/pfentry.s汇编代码，此时C代码的trap_handler已经执行完毕并返回，现在汇编代码需要恢复trap_time时的状态
- 把trap-time时的eip 放到trap_time栈上
- Pop出所有trap_time时的寄存器状态，包括esp，此时esp第一个值即为trap_time时的eip
- 执行eip恢复状态

Exercise11

- 完成set_pgfault_handler函数
- 如果第一次调用该函数，为用户exception栈分配一页空间，并调用`sys_env_set_pgfault_upcall`设置page fault handeler
- 否则直接将handler赋值给_pgfault_handler

Exercise12

- 实现fork duppage和pgfault函数
- 根据文档提示fork基本流程：
  - 设置子进程page_fault_handler
  - 调用sys_exofork创建子环境，小于0报错，等于0则当前为子进程，修正thisenv值即可返回0
  - 用duppage复制
  - 父进程设置子进程page_fault
  - 标记子进程runnable
- Duppage
  - 判断是否为copy on wirte
  - 是的话，复制映射并设置权限位为copy on write，并且重新复制自己的映射
  - 否则权限位为PTE_U | PTE_P
- Pgfault
  - 判断是否权限位是否copy on wirte
  - 为PFTEMP分配一页地址
  - 复制内容到PFTEMP
  - 把PFTEMP映射关系复制到addr，最后取消PFTEMP映射关系

partC

Exercise13

- 修改trapentry.s和trap.c，初始化IRQs 0~15的idt表，即与之前的idt实现相似
- 修改env.c中的env_alloc确保tf_eflags被设置了FL_IF
- 在init.c中运行spin.c，测试发生hardware interrupt

Exercise14

- 修改trap_dispatch()函数，当发生时钟中断时调用sched_yield去寻找一个不同的环境、
- 当tf_trapno == IRQ_OFFSET + IRQ_TIMER时 调用lapic_eoi和sched_yield并返回
- 此处发现了我之前sched_yield的实现错误，循环写法错误，已修正

Exercise15

- 这部分主要是实现进程间的ipc
- 首先在syscall里做send和recieve的分发
- 实现sys_ipc_recv
  - 函数带有一个地址参数，代表收到一个地址映射，如果已经映射该地址，则取消映射
  - 虚拟地址要在UTOP之下，并且PGSIZE对齐，否则返回-E_INVAL
  - 标记env_ipc_recving env_ipc_dstva和env_status
- Sys_ipc_try_send实现
  - 根据注释，实现各个错误判断
  - 设置目标env的各个ipc对应的值
- Ipc_send实现
  - 如果pg为空，传递一个sys_ipc_try_send会出错的值，即UTOP即以上，此处设置为UTOP
  - 一直循环调用sys_ipc_try_send成功为止
  - 返回-E_IPC_NOT_RECV则panic
  - 成功则sys_yield
- Ipc_recv实现
  - pg参数与ipc_send做同样的处理
  - 检查perm_store和from_env_store，如果为空赋值为0，否则用当前thisenv对应的值赋值

Challenge!

我选择了**sfork**作为challenge

- 根据提示，父子进程共享UTEXT到end的内存，用sys_page_map实现即可
- 父子进程的栈需要独立，用copy-on-write实现
- 子进程exception stack需要另外分配一页
- 最后设置子进程page fault handler
- 由于父子进程共享内存，thisenv映射的值一样，所以如果当前运行为子进程，即sys_exofork范围为0，修改thisenv = &envs[ENVX(nums)]
- 最后将修改user/forktree的fork()为sfork()进行测试得到

```
***
*** Use Ctrl-a x to exit qemu
***
qemu-system-i386 -nographic -drive file=obj/kern/kernel.img,index=
0,media=disk,format=raw -serial mon:stdio -gdb tcp::26000 -D
qemu.log -smp 1
pading space in the right to number 22: 22      222.


error! writing through NULL pointer! (%n argument)
show me the sign: +1024, -1024
Physical memory: 131072K available, base = 640K, extended = 130432K
check_page_free_list() succeeded!
check_page_alloc() succeeded!
```

```
check_page_alloc() succeeded!
check_page() succeeded!
check_kern_pgdir() succeeded!
check_page_free_list() succeeded!
check_page_installed_pgdir() succeeded!
SMP: CPU 0 found 1 CPU(s)
enabled interrupts: 1 2
[00000000] new env 00001000
1000: I am ''
[00001000] new env 00001001
[00001000] new env 00001002
[00001000] exiting gracefully
[00001000] free env 00001000
1001: I am '0'
1002: I am '1'
[00001002] new env 00002000
[00001002] new env 00001003
[00001002] exiting gracefully
[00001002] free env 00001002
2000: I am '10'
[00001001] new env 00002002
[00001001] new env 00001004
[00001001] exiting gracefully
[00001001] free env 00001001
[00002000] new env 00002001
[00002000] new env 00001005
[00002000] exiting gracefully
[00002000] free env 00002000
2002: I am '00'
[00002002] new env 00003000
[00002002] new env 00001006
[00002002] exiting gracefully
[00002002] free env 00002002
3000: I am '000'
[00003000] exiting gracefully
[00003000] free env 00003000
1003: I am '11'
[00001003] new env 00004000
[00001003] new env 00003002
[00001003] exiting gracefully
[00001003] free env 00001003
4000: I am '110'
[00004000] exiting gracefully
[00004000] free env 00004000
3002: I am '111'
[00003002] exiting gracefully
[00003002] free env 00003002
2001: I am '100'
[00002001] exiting gracefully
[00002001] free env 00002001
1005: I am '101'
[00001005] exiting gracefully
[00001005] free env 00001005
1004: I am '01'
[00001004] new env 00002005
[00001004] new env 00003001
[00001004] exiting gracefully
[00001004] free env 00001004
3001: I am '011'
2005: I am '010'
[00002005] exiting gracefully
[00002005] free env 00002005
[00003001] exiting gracefully
[00003001] free env 00003001
1006: I am '001'
[00001006] exiting gracefully
[00001006] free env 00001006
No runnable environments in the system!
```