北京大学
PEKING UNIVERSITY

操作系统内核赛分享

2024年5月

目录 Contents

一、环境准备

几个关键内容请注意：

Piazza 有更详细的

安装docker

- 有很多参考的资料，有很多的方法学习了解
- 动手多试

选择正确的镜像

- docker pull docker.educg.net/cg/os-contest:2024p6
- 有些文档推荐的都是往年的镜像，环境有各种小问题

启动镜像参数

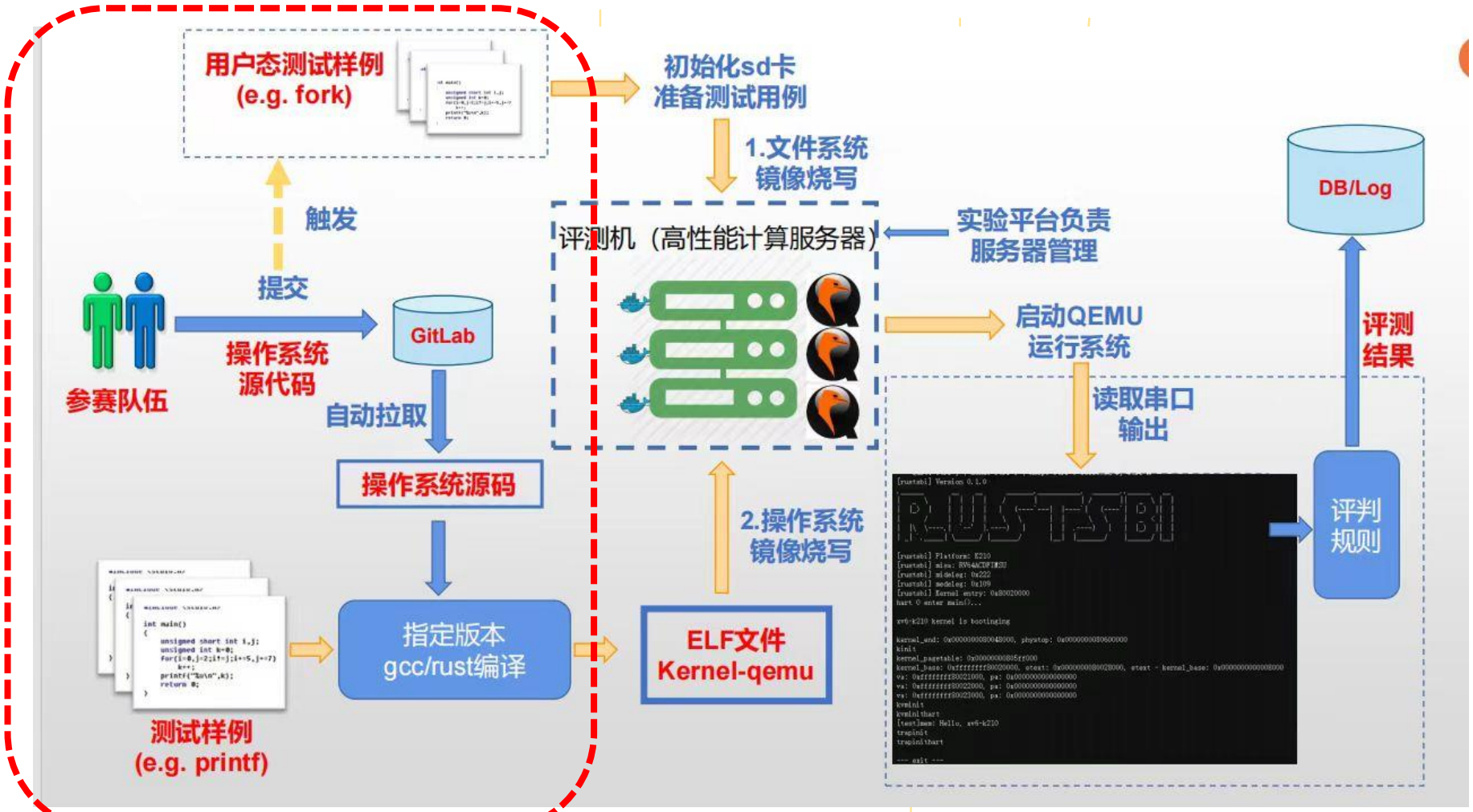
- run -it --privileged=true 镜像名/镜像ID
- 带上参数：--privileged=true ,特权模式运行容器，具有访问主机系统的权限。

选取参考的工程

- K210：git clone https://github.com/HUST-OS/xv6-k210.git
- 往年其它参赛项目。

二、测评流程

因为需要提交到测评平台上，在写代码前先要搞懂测评流程。K210的代码如何提交并通过测评的流程？



这个图来自内核赛网站的技术文档，大概说明了决赛的测评流程。

我们的作业主要是左边红框的步骤。

作业的步骤如下：

第一步：完成操作系统内核代码，将代码提交到gitlab上，需**将项目改为public**。

第二步：在希冀作业平台**提交仓库地址：**https://gitlab.eduxiji.net/pku22000xxxxx/oskernel2024-xxxxxx.git。

第三部：提交后，测评平台会自动下载仓库地址中的代码，执行以下步骤（下页继续）：

测评平台中的步骤：

3.1 测评平台通过`make all`命令生成kernel-qemu内核文件（希冀作业平台中有详细说明）；

3.2 评测平台会调用如下命令自动启动qemu模拟器

```
qemu-system-riscv64 -machine virt -kernel kernel-qemu -m 128M -nographic -smp 2 -bios default -drive file=sdcard.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0 -no-reboot -device virtio-net-device,netdev=net -netdev user,id=net
```

内核启动后，自动加载评测机的sdcard.img镜像**（后面有详细介绍）**

3.3 内核需要依次执行评测机镜像文件根目录下存放的32个测试程序；测试平台根据测试程序的输出结果，评判得分。

3.4 内核调用完测试程序后，主动关机退出。

有的同学提交后，会反馈显示“运行时间过程！”，可能是这个步骤的原因。

我在内核大赛qq群咨询的结果：

提交测试后，显示:运行时间过长！请问可能是什么原因？

====目前是编译+运行一共限制5分钟，可以检查一下是不是编译时下载了包导致时间过长，或者运行结束后没有主动退出qemu？

测评的代码

brk.c
（调用系统函数）

```c
void test_brk(){
    TEST_START(__func__);
    intptr_t cur_pos, alloc_pos,
alloc_pos_1;

    cur_pos = brk(0);
    printf("Before alloc,heap pos: %d\n",
cur_pos);
    brk(cur_pos + 64);
    alloc_pos = brk(0);
    printf("After alloc,heap pos:
%d\n",alloc_pos);
    brk(alloc_pos + 64);
    alloc_pos_1 = brk(0);
    printf("Alloc again,heap pos:
%d\n",alloc_pos_1);
    TEST_END(__func__);
}
```

brk_test.py
（根据调用的输出结果，进行评分）

```python
class brk_test(TestBase):
    def __init__(self):
        super().__init__("brk", 3)

    def test(self, data):
        self.assert_ge(len(data), 3)
        p1 = "Before alloc,heap pos: (.+)"
        p2 = "After alloc,heap pos: (.+)"
        p3 = "Alloc again,heap pos: (.+)"
        line1 = re.findall(p1, data[0])
        line2 = re.findall(p2, data[1])
        line3 = re.findall(p3, data[2])
        if line1 == [] or line2 == [] or
line3 == []:
            return
        a1 = int(line1[0], 10)
        a2 = int(line2[0], 10)
        a3 = int(line3[0], 10)
        self.assert_equal(a1 + 64, a2)
        self.assert_equal(a2 + 64, a3)
```

三、内核实现过程

1.选取合适的基础工程

K210： https://github.com/HUST-OS/xv6-k210.git

其它往年工程： https://github.com/oscomp/2021oscomp-best-kernel-design-impl

本人选取了往年的一个工程，内核结构清晰，并且进行了一些扩展。

另外，也基于K210调通了整个过程，提交到希冀平台上，自带的一些系统函数调用测试用例可正常通过，在不写新的系统函数情况下，可得54分。

由于大多数同学选择的是K210平台，为了便于大家理解，本PPT介绍的流程和实现都是基于K210的。

2.生成sdcard.img（用于本地调试，测评平台上不需要提交上去）

（1）包含哪些文件

参考文档：https://github.com/oscomp/testsuits-for-oskernel/tree/main?tab=readme-ov-file

32个测试用例，每个用例对应1个c程序文件、1个python文件。如： sleep.c、 sleep.py ……

C测试用例：通过系统调用访问内核的系统函数，根据输出，判断是否正确。

Python脚本：判断每个用例的输出，判定得分。

（2）如何生成sdcard.img

https://github.com/oscomp/testsuits-for-oskernel/blob/main/fat32-info.md

```
#以下是makefile中的 sdcard.img 目标
#（1）利用linux下的dd命令生成128M的文件
#（2）然后利用linux下的mkfs.vfat命令制作成fat32格式
#（3）将fsimg目录下的内容都copy到$(MNT_DIR)下，这样sdcard.img下就包含了所有fsimg下的文件
sdcard.img:
	@dd if=/dev/zero of=$@ bs=1M count=128
	@mkfs.vfat -F 32 $@
	@mount -t vfat $@ $(MNT_DIR)
	@cp -r $(FSIMG)/* $(MNT_DIR)/
	@sync $(MNT_DIR) && umount -v $(MNT_DIR)
```

3.如何自动调用评测系统的测试用例

和其他同学一样，初次接触内核赛，这块是花时间和精力最多的地方，主要是要理解测评的流程，如何在内核中调用sdcard.img中的测试用例。

第一步：将原来代码中启动sh交互界面，修改成自动调用测评用例。主要修改 xv6-user/init.c

```
main(void)
{
  ......
  dev(O_RDWR, CONSOLE, 0);
  dup(0);  // stdout
  dup(0);  // stderr

  for(;;){
    printf("init: starting sh\n");
    pid = fork();
    if(pid < 0){
      printf("init: fork failed\n");
      exit(1);
    }
    if(pid == 0){
      exec("sh", argv);
      printf("init: exec sh failed\n");
      exit(1);
    }
  ......
```

```
char *tests[] = {
    "brk",
    "chdir",
...... ......          }
int main(void) {

  for (int i = 0; i < counts; i++) {
    printf("init: starting tests\n");
    pid = fork();
    ......
    if (pid == 0) {
        exec(tests[i], argv);
        printf("init: exec tests failed\n");
        exit(1);
    }
   ......
 __syscall0(SYS_shutdown); //关机！！！！
```

3.如何自动调用评测系统的测试用例

如何主动自动退出qemu？

其实，如果5分钟能等把所有测试用例都跑完、得分，感觉不关机也没有影响。

参考大赛资料：https://github.com/oscomp/testsuits-for-oskernel/blob/main/riscv-syscalls-testing/user/lib/syscall.h。直接照搬过来了，具体逻辑也没细研究，

```
#define __asm_syscall(...)                  \
    __asm__ __volatile__("ecall\n\t"        \
                         : "=r"(a0)         \
                         : __VA_ARGS__      \
                         : "memory");       \
    return a0;
static inline long __syscall0(long n)
{
    register long a7 __asm__("a7") = n;
    register long a0 __asm__("a0");
    __asm_syscall("r"(a7))
}
```

调用方式：

```
__syscall0(SYS_shutdown);
```

注：在kernel/include/sysnum.h 增加调用号

```
#define SYS_shutdown    210
```

testsuits-for-oskernel / riscv-syscalls-testing / user / lib / arch / riscv / syscall_arch.h

elliott10 Init testsuits

Code Blame 80 lines (70 loc) · 2.39 KB

```
1   #define __SYSCALL_LL_E(x) (x)
2   #define __SYSCALL_LL_O(x) (x)
3
4   #define __asm_syscall(...)              \
5       __asm__ __volatile__("ecall\n\t"    \
6                            : "=r"(a0)     \
7                            : __VA_ARGS__  \
8                            : "memory");   \
9       return a0;
10
11  static inline long __syscall0(long n)
12  {
13      register long a7 __asm__("a7") = n;
14      register long a0 __asm__("a0");
15      __asm_syscall("r"(a7))
16  }
17
```

4.如何自动调用评测系统的测试用例

第二步：编译`init.c`生成_init可执行文件（文件位置：xv6-user/_init）后，通过`runtest.sh`脚本将_init可执行文件转成16进制格式的文本（存放位置：kernel/include/initcode.h）

`runtest.sh`

```shell
riscv64-linux-gnu-objcopy -S -O binary xv6-user/_init oo
od -v -t x1 -An oo | sed -E 's/ (.{2})/0x\1,/g' > kernel/include/initcode.h
```

initcode.h代码片段示例如下：

```c
0x5d,0x71,0xa2,0xe0,0x86,0xe4,0x26,0xfc,0x4a,0xf8,0x4e,0xf4,0x52,0xf0,0x56,0xec,
0x5a,0xe8,0x5e,0xe4,0x62,0xe0,0x80,0x08,0x93,0x05,0x60,0x1b,0x17,0x15,0x00,0x00,
0x13,0x05,0x45,0xdf,0x97,0x10,0x00,0x00,0xe7,0x80,0xe0,0xbb,0x09,0x46,0x97,0x15,
…… ……
```

4.如何自动调用评测系统的测试用例

第三步：在内核启动过程中的最后一步，需要创建第一个用户进程，分配内存物理空间，将上面生成的`initcode.h`的十六进制文本作为进程的指令和数据，加载到用户进程对应的内存中，并运行。

将十六进制的字符串赋值给`initcode[]`数组，然后在`kernel/proc.c`中的`userinit()`函数中，加载并执行上述十六进制代码。

代码如下页：

4.如何自动调用评测系统的测试用例

```
uchar initcode[] = {#include "initcode.h"};
//上面的语法等同于 initcode[] = {0x5d,0x71,0xa2,0xe0,0x86,0xe4,0x26,0xfc …… ……}

// Set up first user process.     //启动第一个用户进程!!!
Void userinit(void)
{
  struct proc *p;
  p = allocproc();
  initproc = p;
  // allocate one user page and copy init's instructions and data into it.
  uvminit(p->pagetable , p->kpagetable, initcode, sizeof(initcode));
  p->sz = PGSIZE;                    //每个内存页的大小
  // prepare for the very first "return" from kernel to user.
  p->trapframe->epc = 0x0;       // user program counter
  p->trapframe->sp = PGSIZE;  // user stack pointer

  safestrcpy(p->name, "init", sizeof(p->name));
  p->state = RUNNABLE;
  p->tmask = 0;
  release(&p->lock);
}
```

4.如何自动调用评测系统的测试用例

注意：initcode[]数组的大小不要超过4k。在K210代码框架中定义了Pagesize 4096 bytes。

kernel/ include/risc.h

```
337  #define PGSIZE 4096 // bytes per page
338  #define PGSHIFT 12  // bits of offset within a page
339
340  #define PGROUNDUP(sz)  (((sz)+PGSIZE-1) & ~(PGSIZE-1))
341  #define PGROUNDDOWN(a) (((a)) & ~(PGSIZE-1))
```

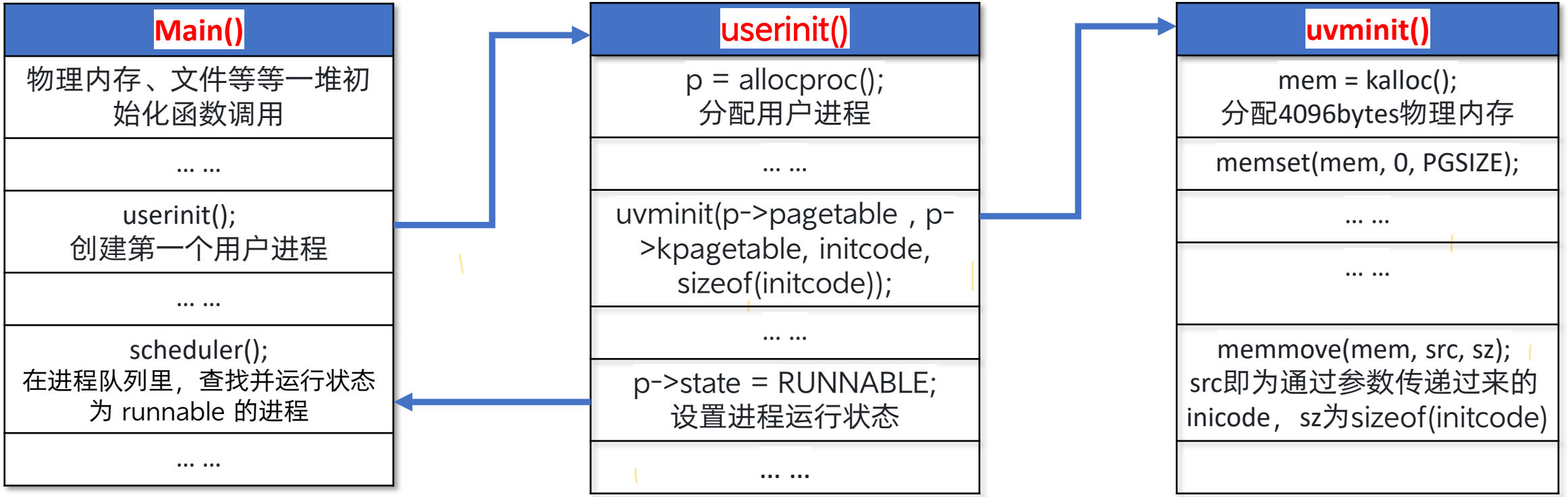
kernel/vm.c

```
261  // Load the user initcode into address 0 of pagetable,
262  // for the very first process.
263  // sz must be less than a page.
264  void
265  uvminit(pagetable_t pagetable, pagetable_t kpagetable, uchar *src, uint sz)
266  {
267    char *mem;
268
269    if(sz >= PGSIZE)
270      panic("inituvm: more than a page");
271    mem = kalloc();
272    // printf("[uvminit]kalloc: %p\n", mem);
273    memset(mem, 0, PGSIZE);
274    mappages(pagetable, 0, PGSIZE, (uint64)mem, PTE_W|PTE_R|PTE_X|PTE_U);
275    mappages(kpagetable, 0, PGSIZE, (uint64)mem, PTE_W|PTE_R|PTE_X);
276    memmove(mem, src, sz);
277    // for (int i = 0; i < sz; i ++) {
278    //   printf("[uvminit]mem: %p, %x\n", mem + i, mem[i]);
279    // }
280  }
```

4.如何自动调用评测系统的测试用例（小结）

（1）我们需要理解K210的代码启动的整个过程：

kernel/main.c 中的**main()**函数 在各种系统初始化后，调用kernel/proc.c中的函数userinit()来创建第一个用户进程。**userinit()** 首先调用allocproc()来分配一个用户进程p，接着调用kernel/vm.c中的uvminit()函数，为用户进程创建一个物理内存页，将initcode数组作为参数传递给进程p，也就是将initcode放到分配的物理内存中，然后设置进程状态 p->state = RUNNABLE。**uvminit()** 将initcode装载到pagetable 的 0 起始地址，不超过一个pgsize大小。最后，由main()中的`scheduler()`调度运行第一个用户进程。



可详细阅读 K210项目中 \doc\构建调试-进程管理.md，对xv6的进程调度过程进行了详细说明。

4.如何自动调用评测系统的测试用例（小结）

（2）根据上面这个过程，可以有多种调用测试用例的实现方式。**我这里将循环调用32个测试用例放在了init.c中实现，跳过了initcode.S。**

如果参考的是往年的其它参赛项目，这块的实现框架可能不一样，但总体都差别不是太大。

（3）上周同学的分享，是直接修改了initcode.S，再生成二进制文件。如下（**摘自分享PPT**）：

注：这个过程我采用的是修改xv6-k210: xv6-user/initcode.S, 并将其汇编程二进制文件，再通过`od -t xC initcode`
获得其字节序列，最后赋值给 xv6-k210: kernel/proc.c # initcode 来完成的

5.本地gdb调试（参考了往届项目的makefile）

（1）在启动gdb模式前，要保证sdcard.img文件生成。

（2）运行`make gdb`命令启动gdb模式，这个时候gdb模式下的`-S`参数会主动停止CPU运行，进入到挂起的状态，等待另外一个终端下达gdb调试指令。

```
#makefile中增加gdb目标用于调试，和非调试相比，主要多了-S $(QEMUGDB)这个参数，设置TCP监听端口
#gdb模式启动内核后，内核会挂起，等待另外一个窗口下达gdb调试指令
GDBPORT = $(shell expr `id -u` % 5000 + 25000)#因为我们用的是root,id=0,因此端口号就是25000
gdb: kernel-qemu .gdbinit
    $(QEMU) $(QEMUOPTS) -S $(QEMUGDB)
```

（3）在新的终端窗口，运行如下命令进行连接。

```
# 调试kernel-qemu(elf)文件
riscv64-unknown-elf-gdb kernel-qemu
# 连接25000端口
(gdb) target remote :25000
# 如果我们想从入口的main函数调试，可以通过下断点的方式进行逐步调试
(gdb) break main
# 继续执行下一语句
(gdb) continue
```

北京大学
PEKING UNIVERSITY

供参考 谢谢