LINUX 开机过程可视化

制作人：薛雨萌

同组人：无

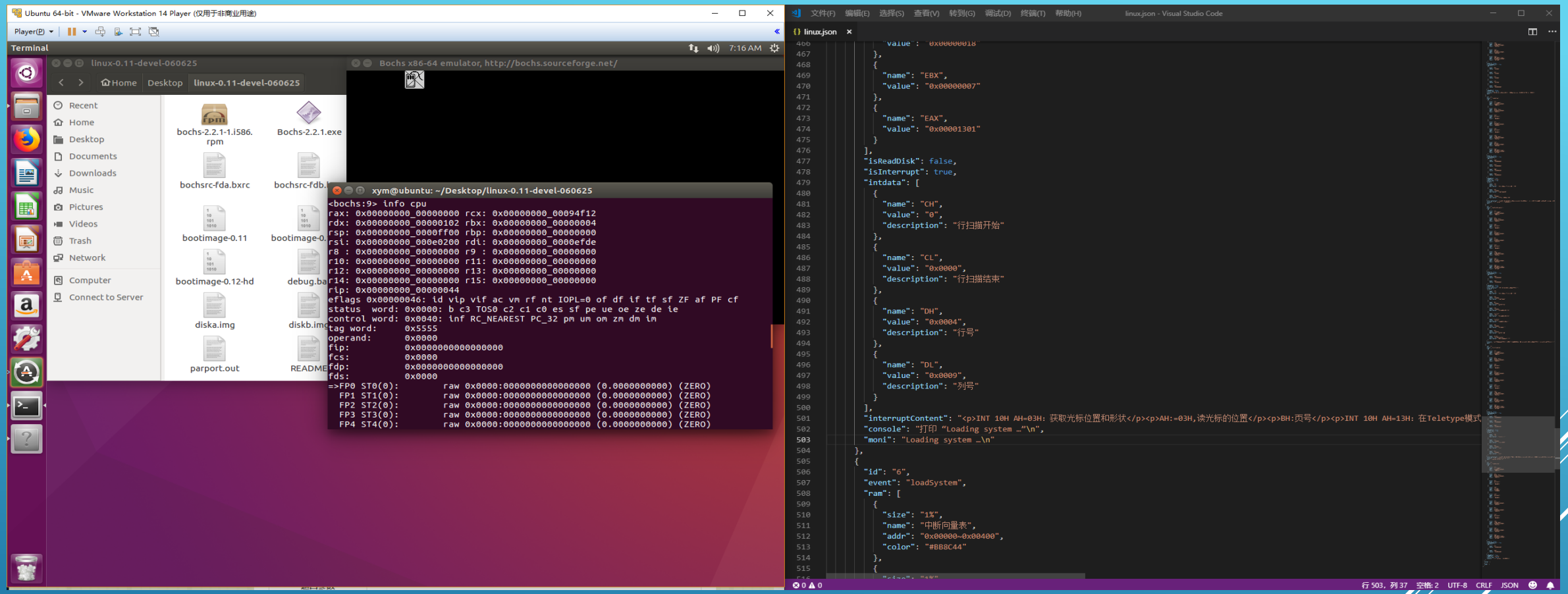
指导老师：杨兴强

阅读代码

- 阅读了第三章，第四章，第五章，第十章等重点章节，确立了要进行开机部分的可视化
- 反复仔细阅读bootsect.S setup.S head.S以及main.c，确定要提取的数据，对比去年杨浩然的数据提取，发现有很多做的不够好的地方，在这些地方进行改进。
- 进入main.c之后主要涉及到各种函数调用，宋振华陈宇翔组的可视化框架可以做这一部份工作，因此主要可视化BIOS执行阶段和另外三个汇编程序执行阶段。
- 三个文件看似不多，但其中的行为并不具备重复性，也就是说这是一个固定的过程，导致关键帧非常多，每个关键帧都有不小的差别，这导致可视化变得很繁琐

确立关键帧

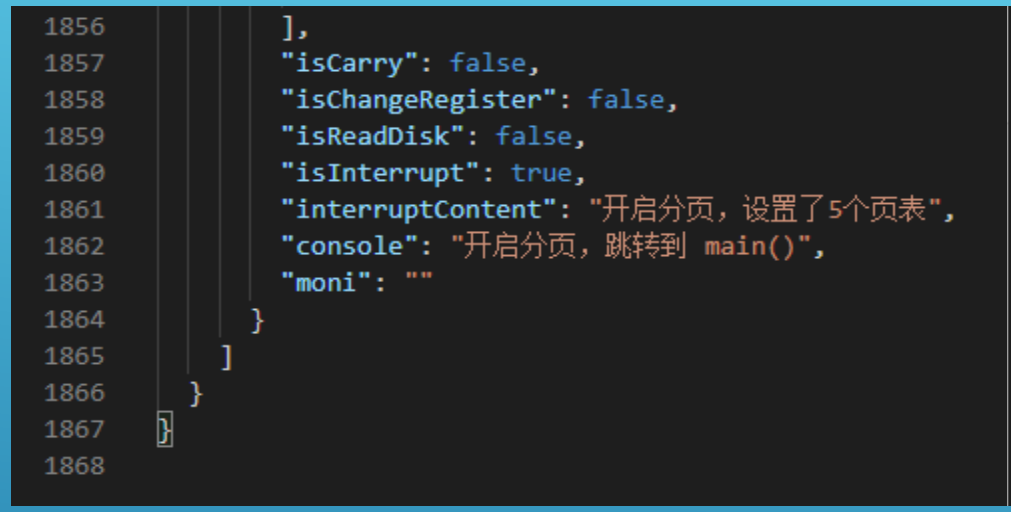
- 因为关键帧多达几十个，而且基本很多相同点差不少，因此我分析了关键帧中哪些没有变化，哪些变化少，哪些变化太大以至于必须做出专门的组件才能可视化
- 总结了三份分析报告，帮助确定关键帧


Ubuntu 64-bit - VMware Workstation 14 Player (仅用于非商业用途)
Player(P)
Terminal
7:16 AM
linux-0.11-devel-060625
Home
Desktop
linux-0.11-devel-060625
Recent
Home
Desktop
Documents
Downloads
Music
Pictures
Videos
Trash
Network
Computer
Connect to Server
bochs-2.2.1-1.i586.rpm
Bochs-2.2.1.exe
bochsrc-fda.bxrc
bootimage-0.11
bootimage-0.12-hd
diska.img
parport.out
Bochs x86-64 emulator, http://bochs.sourceforge.net/
xym@ubuntu: ~/Desktop/linux-0.11-devel-060625
<bochs:9> info cpu
rax: 0x00000000_00000000 rcx: 0x00000000_00094f12
rdx: 0x00000000_00000102 rbx: 0x00000000_00000004
rsp: 0x00000000_0000ff00 rbp: 0x00000000_00000000
rsi: 0x00000000_000e0200 rdi: 0x00000000_0000efde
r8 : 0x00000000_00000000 r9 : 0x00000000_00000000
r10: 0x00000000_00000000 r11: 0x00000000_00000000
r12: 0x00000000_00000000 r13: 0x00000000_00000000
r14: 0x00000000_00000000 r15: 0x00000000_00000000
rip: 0x00000000_00000044
eflags 0x00000046: id vip vif ac vm rf nt IOPL=0 of df if tf sf ZF af PF cf
status word: 0x0000: b c3 TOS0 c2 c1 c0 es sf pe ue oe ze de ie
control word: 0x0040: inf RC_NEAREST PC_32 pm um om zm dm im
tag word: 0x5555
operand: 0x0000
fip: 0x0000000000000000
fcs: 0x0000
fdp: 0x0000000000000000
fds: 0x0000
=>FP0 ST0(0):   raw 0x0000:0000000000000000 (0.0000000000) (ZERO)
  FP1 ST1(0):   raw 0x0000:0000000000000000 (0.0000000000) (ZERO)
  FP2 ST2(0):   raw 0x0000:0000000000000000 (0.0000000000) (ZERO)
  FP3 ST3(0):   raw 0x0000:0000000000000000 (0.0000000000) (ZERO)
  FP4 ST4(0):   raw 0x0000:0000000000000000 (0.0000000000) (ZERO)
文件(F) 编辑(E) 选择(S) 查看(V) 转到(G) 调试(D) 终端(T) 帮助(H)
linux.json - Visual Studio Code
linux.json
"value": "0x00000018"
"name": "EBX",
"value": "0x00000007"
"name": "EAX",
"value": "0x00001301"
"isReadDisk": false,
"isInterrupt": true,
"intdata": [
"name": "CH",
"value": "0",
"description": "行扫描开始"
"name": "CL",
"value": "0x0000",
"description": "行扫描结束"
"name": "DH",
"value": "0x0004",
"description": "行号"
"name": "DL",
"value": "0x0009",
"description": "列号"
"interruptContent": " INT 10H AH=03H: 获取光标位置和形状 AH:=03H,读光标的位置 BH:页号 INT 10H AH=13H: 在Teletype模式
"console": "打印 “Loading system …”\n",
"moni": "Loading system …\n"
"id": "6",
"event": "loadSystem",
"ram": [
"size": "1%",
"name": "中断向量表",
"addr": "0x00000~0x00400",
"color": "#BB8C44"
行 503，列 37 空格: 2 UTF-8 CRLF JSON

提取数据

提取数据

- 由于进入main.c之前其它的一切方法都不可用，只能使用bochs手动提数据，基本相当于自己一行代码一行代码的看，一行代码一行代码的调试，非常的复杂繁琐，下一级同学参考我的数据就足够了，基本这部分数据都让我提取出来了。
- 和上一级杨浩然的相比，我使用的都是真实的数据，而不是只是为了展现效果的描述，这部分看我的数据就知道了。



```
1856            ],
1857            "isCarry": false,
1858            "isChangeRegister": false,
1859            "isReadDisk": false,
1860            "isInterrupt": true,
1861            "interruptContent": "开启分页，设置了5个页表",
1862            "console": "开启分页，跳转到 main()",
1863            "moni": ""
1864        }
1865    ]
1866  }
1867 }
1868
```

```
285 }
286
287 {
288     "module":"HEAD",
289     "event": "setpage",
290     "provider": "yhr",
291     "time": 22,
292     "data":{
293         "description":"开启A20地址线"
294     }
295 }
296
```

左图为我的数据行数，右图为去年的

左图为我的某个数据的一部分，右图是相同功能的数据的全部内容

改进

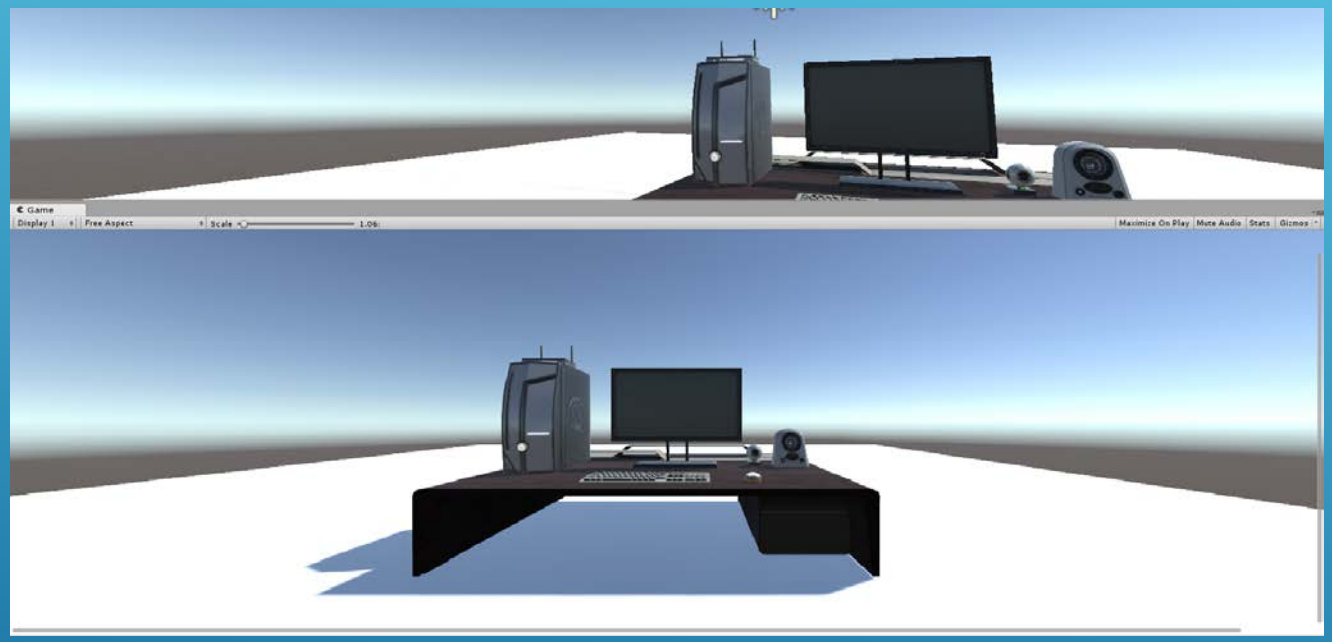
- 相比去年的数据，多了很多新的数据，详细丰富了很多，基本将整个开机阶段的数据完整提取了出来。
- 数据真实，完全是手动调试保存下来的，不仅包括代码里写好的对寄存器操作的值，更是连中断返回的寄存器的值都包括，中断返回值的意义也有说明。

可视化

- 在我看来，可视化不仅是做出能够可视化的东西。
- 比如做动画，虽然效果很好，但是不可复用，过程是死的，数据稍有变化就不能用了。
- 因此封装可视化的组件，适应各种数据的变化才是可视化的目的。
- 我的可视化页面的组件基本都可以复用，比如可视化内存布局的组件，按照我的数据格式写好，不同的数据都可以正常使用，可供下一级使用。

可视化（放弃的方案）

- 最初我打算用unity3d可视化，进bootsect之前它的效果还是不错的，但发现UGUI各种不便，于是换成了web



- 下面开始展示我的最终方案

可视化（最终方案）