**Boot**

1. 实验目的

探索最基本的使用方法，学会如何调试

二、实验内容

[Homework: boot xv6 (mit.edu)](https://pdos.csail.mit.edu/6.828/2018/homework/xv6-boot.html)

1. 实验步骤与实验结论

启动xv6

git clone git://github.com/mit-pdos/xv6-public.git

make

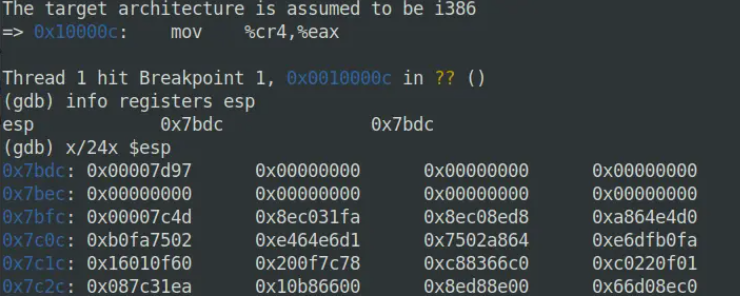
make qemu-gdb make gdb

注意：官方给的代码需要加入 gdb: gdb -n -x .gdbinit

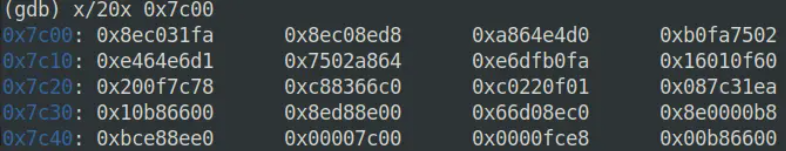
命令：nm kernel | grep \_start



内核的起始地址0x10000c，所以在这里打一个断点，然后看看此时在栈内的内容



start标号的地址在0x7c00



Q. 将断点设置在0x7c00,逐步调试，在那里初始化了栈？

Q. 逐步执行call bootmain，现在栈里面有什么？

Q. bootmain.asm当中第一条对栈做了什么操作？仔细查看bootmain.asm

Q. call 0x10000c做了对栈做了什么呢?

A. call bootmain上一句代码初始化了栈，在calll bootmain后，在栈内压入了0x7c4d,调用函数的时候先要执行push %ebp，所以bootmain.asm当中第一局指令压入了ebp。在call 0x10000c的时候，又压入了0x7d97。

Q. 当程序断在上面的断点(0x0010000c)时,使用x/24x $esp查看寄存器和栈中的内容,并简短描述栈中每个非0值的含义

A. 第一行第一个地址是bootmain中call entry()的下一条指令地址,即函数entry()的返回地址,第三行第一个是bootasm.S中call bootmain()后的下一条指令地址,即函数bootmain()的返回地址,由第一步调试可知栈从0x7c00开始向下增长,查看bootblock.asm可知0x7c00往上的地址中(第三行第二个地址往后)存储的都是指令。

具体地址情况：

0x7c00:8ec031fa是执行完movl $start, %esp后的结果

0x7bfc:0x00007c4d存的是call bootmain指令的下一条指令的地址

0x7bf8:0x0存的是bootmain的调用者程序的栈帧基地址ebp:0x0

0x7bf4、0x7bf0、0x7bec:0x0分别是保存edi、esi、ebx的值，都为0

0x7be8~0x7be0:0x0是执行了sub $0xc, %esp后的结果

0x7bdc:0x00007db存的是call\*0x10018(内核入口)指令的下一条指令的地址

四、反思总结

知道了内部调用流程,学会了调试方法