

OSH Lab1

实验配置环境

本次实验配置的操作环境

- Ubuntu version 17.10.1 x64
- Linux version 3.18.6
- Linux version 4.1.51
- gdb version 7.11.1
- VMWare Fusion version 10.0.1
- QEMU version 2.12.0

linux内核源码的获取及分支管理

在kernel的官网上有三种版本，mainline，stable，longterm

- mainline是主线版本，最新的
- stable是稳定版
- longterm是长期支持版

kernel的官方网站(<https://www.kernel.org/>)获得，在这里采取了直接在kernel官网下载然后拖入vmware虚拟机中进行解压的方法。

解压完成后输入一下代码

```
1 xz -d linux-3.18.6.tar.xz
2 tar -xvf linux-3.18.6.tar
3 cd linux-3.18.6
4 make clean
5 make x86_64_defconfig
6 #此时发现这边有一个小问题，就是没有打开compile the kernel with debug info 选项
7 #所以重新再make menuconfig下进行设置
8 #先输入sudo apt-get install libncurses5-dev libncursesw5-dev
9
10 make menuconfig
11 #再在kernel hacking 下勾上compile the kernel with debug info
12 #此时再次使用make进行重新编译
13 make -j 4 #将vmware可使用的cpu
14
```

注意上一步中其实make内核的时候容易产生较大的问题

因为gcc版本所以会出现bug

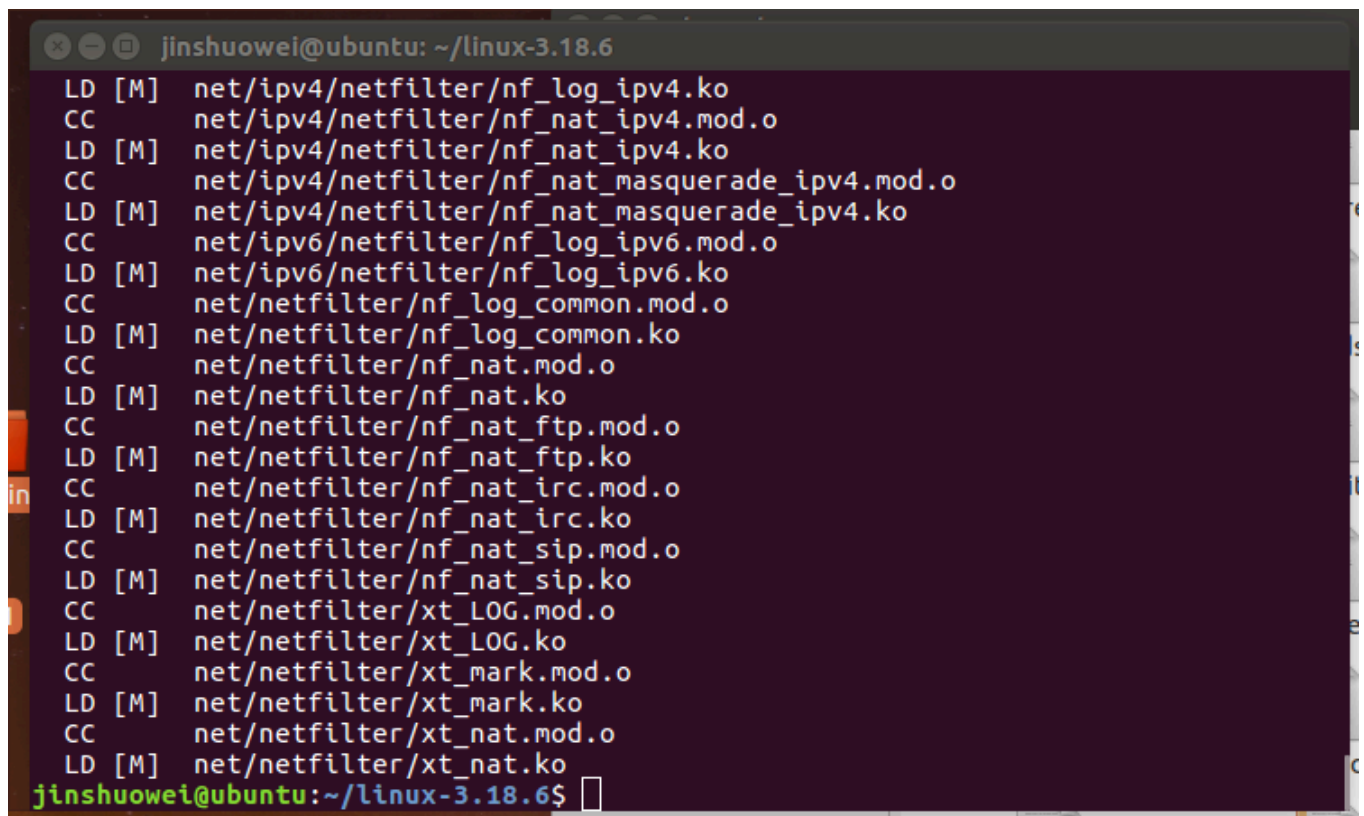
此时就需要将gcc版本默认降级就可以成功make了

具体降级的办法如下

```

1 sudo apt-get install gcc-4.8
2 sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-4.8 100
3 sudo update-alternatives --config gcc

```



```

jinshuwei@ubuntu: ~/linux-3.18.6
LD [M] net/ipv4/netfilter/nf_log_ipv4.ko
CC net/ipv4/netfilter/nf_nat_ipv4.mod.o
LD [M] net/ipv4/netfilter/nf_nat_ipv4.ko
CC net/ipv4/netfilter/nf_nat_masquerade_ipv4.mod.o
LD [M] net/ipv4/netfilter/nf_nat_masquerade_ipv4.ko
CC net/ipv6/netfilter/nf_log_ipv6.mod.o
LD [M] net/ipv6/netfilter/nf_log_ipv6.ko
CC net/netfilter/nf_log_common.mod.o
LD [M] net/netfilter/nf_log_common.ko
CC net/netfilter/nf_nat.mod.o
LD [M] net/netfilter/nf_nat.ko
CC net/netfilter/nf_nat_ftp.mod.o
LD [M] net/netfilter/nf_nat_ftp.ko
CC net/netfilter/nf_nat_irc.mod.o
LD [M] net/netfilter/nf_nat_irc.ko
CC net/netfilter/nf_nat_sip.mod.o
LD [M] net/netfilter/nf_nat_sip.ko
CC net/netfilter/xt_LOG.mod.o
LD [M] net/netfilter/xt_LOG.ko
CC net/netfilter/xt_mark.mod.o
LD [M] net/netfilter/xt_mark.ko
CC net/netfilter/xt_nat.mod.o
LD [M] net/netfilter/xt_nat.ko
jinshuwei@ubuntu: ~/linux-3.18.6$

```

如图是编译好内核后的结果

接下来开始制作根文件系统

```

1 cd ..
2 mkdir rootfs
3 git clone https://github.com/mengning/menu.git
4 cd menu
5 gcc -o init linktable.c menu.c test.c -m32 -static -lpthread #全部使用静态的方式
6 cd ../rootfs
7 cp ../menu/init ./
8 find . | cpio -o -Hnewc |gzip -9 > ../rootfs.img #把rootfs打包好
9

```

注意在这里第四步的时候，需要在前面执行一句

`sudo apt-get install g++-7-multilib`

7:用版本号代替

否则无法运行

启动MenuOS系统

```

1 cd ..
2 qemu-system-x86_64 -kernel arch/x86_64/boot/bzImage -initrd rootfs.img -S -s
3
4

```

```
QEMU
[ 5.216531] Key type dns_resolver registered
[ 5.216729] mce: Unable to init device /dev/mcelog (rc: -5)
[ 5.218178] Using IPI No-Shortcut mode
[ 5.228278] registered taskstats version 1
[ 5.233671] Magic number: 10:27:988
[ 5.234389] console [netcon0] enabled
[ 5.234467] netconsole: network logging started
[ 5.236891] ALSA device list:
[ 5.236969] No soundcards found.
[ 5.300638] Freeing unused kernel memory: 572K (c1a29000 - c1ab8000)
[ 5.301532] Write protecting the kernel text: 7464k
[ 5.301852] Write protecting the kernel read-only data: 2448k

  *      *                               ****      *
***  ***      **      **      *      *      *      *      **
* * * *      * *      * *      *      *      *      *      *
* * * *      *      *      *      *      *      *      *      *
* * * *      *      *      *      *      *      *      *      *
*      *      *      *      *      *      *      *      *
*      *      *      *      *      *      *      *      *
*      *      *      *      *      *      *      *      *

MenuOS>>[ 5.643450] input: ImExPS/2 Generic Explorer Mouse as /devices/platfo
rm/i8042/serio1/input/input3
```

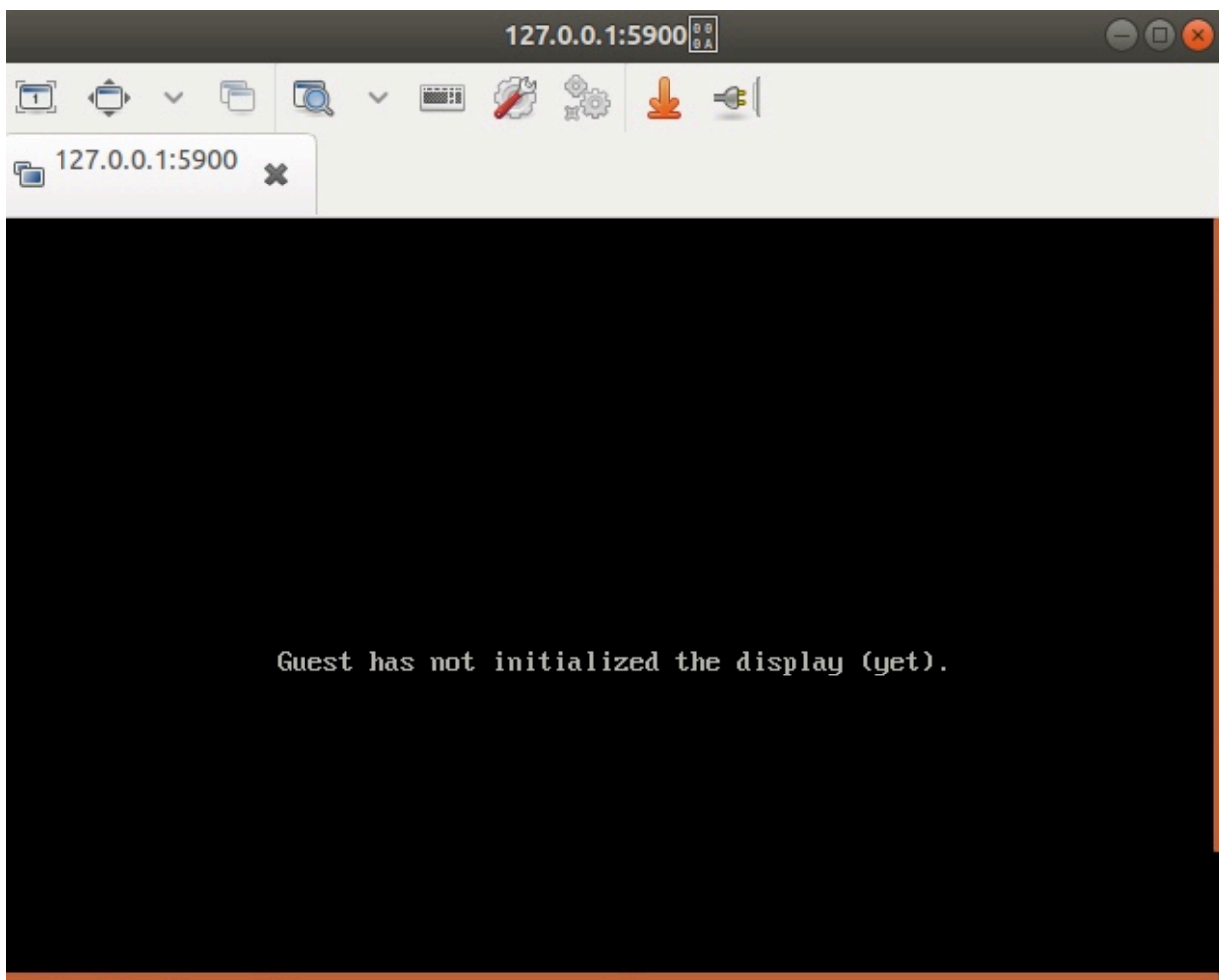
如图是成功启动menuos的界面

使用gdb跟踪调试内核

```
1 cd ..
2 qemu-system-x86_64 -kernel linux-3.18.6/arch/x86/boot/bzImage -initrd rootfs.img -s -S
```

-S是在程序开始的时候用来冻结CPU的
-s 是使用gdb tcp::1234端口的shorthand
此时输入完命令，发现窗口确实处于被冻结状态
此时使用VNC来连接

```
jinshuowei@ubuntu: ~/Desktop
File Edit View Search Terminal Help
No command 'qemu' found, did you mean:
  Command 'aqemu' from package 'aqemu' (universe)
qemu: command not found
jinshuowei@ubuntu:~/Desktop$ ^C
jinshuowei@ubuntu:~/Desktop$ qemu-system-x86_64 -kernel linux-3.18.6/arch/x86/b
oot/bzImage -initrd rootfs.img -s -S
VNC server running on 127.0.0.1:5900
^Cqemu-system-x86_64: terminating on signal 2
jinshuowei@ubuntu:~/Desktop$ ^C
jinshuowei@ubuntu:~/Desktop$ qemu-system-x86_64 -kernel linux-3.18.6/arch/x86/b
oot/bzImage -initrd rootfs.img -s -S tcp::1234
qemu-system-x86_64: -S: Unknown protocol 'tcp'
jinshuowei@ubuntu:~/Desktop$ qemu-system-x86_64 -kernel linux-3.18.6/arch/x86/b
oot/bzImage -initrd rootfs.img -s -S tcp::1234
qemu-system-x86_64: -S: Unknown protocol 'tcp'
jinshuowei@ubuntu:~/Desktop$ qemu-system-x86_64 -kernel linux-3.18.6/arch/x86/b
oot/bzImage -initrd rootfs.img -s -S^C
jinshuowei@ubuntu:~/Desktop$ ^C
jinshuowei@ubuntu:~/Desktop$ qemu-system-x86_64 -kernel linux-3.18.6/arch/x86/b
oot/bzImage -initrd rootfs.img -s -S
VNC server running on 127.0.0.1:5900
```

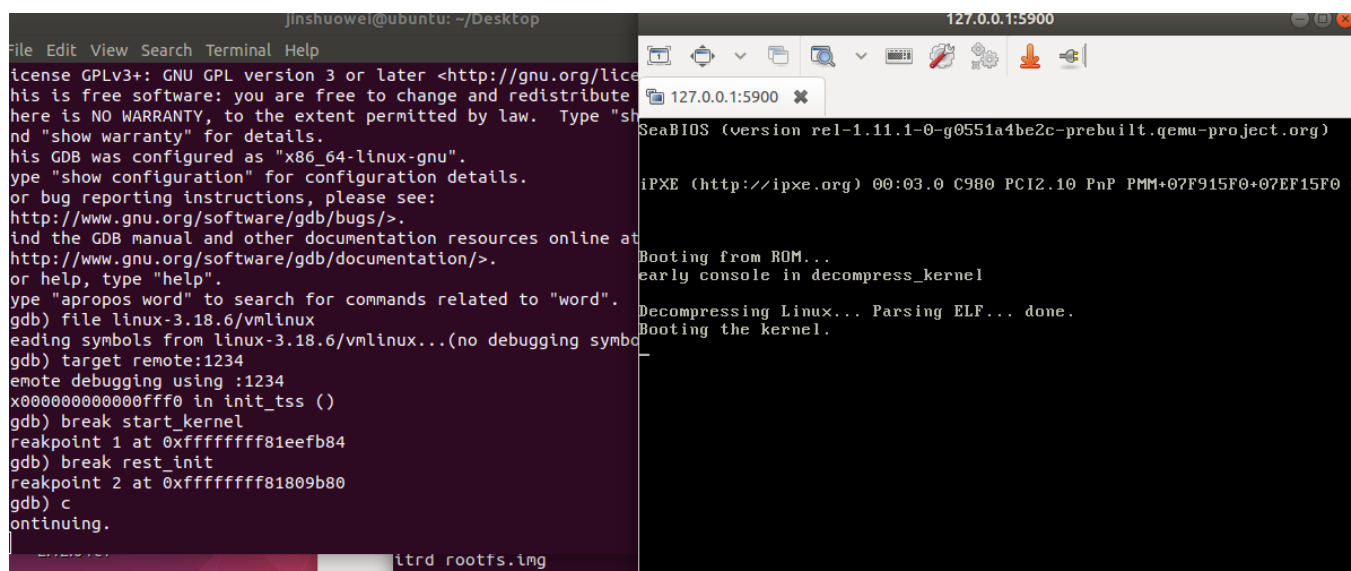


此时再新开一个terminal窗口，用其打开gdb

```
jinshuwei@ubuntu: ~  
jinshuwei@ubuntu:~$ gdb  
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1  
Copyright (C) 2016 Free Software Foundation, Inc.  
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>  
This is free software: you are free to change and redistribute it.  
There is NO WARRANTY, to the extent permitted by law. Type "show copying"  
and "show warranty" for details.  
This GDB was configured as "x86_64-linux-gnu".  
Type "show configuration" for configuration details.  
For bug reporting instructions, please see:  
<http://www.gnu.org/software/gdb/bugs/>.  
Find the GDB manual and other documentation resources online at:  
<http://www.gnu.org/software/gdb/documentation/>.  
For help, type "help".  
Type "apropos word" to search for commands related to "word".  
(gdb) 
```

在gdb窗口输入以下命令

```
1 file linux-3.18.6/vmlinux  
2 target remote:1234  
3 break start_kernel #这样就在内核启动的位置设置了一个断点  
4 c #使得系统开始执行qemu  
5 #用类似的方法可以设置更多的断点  
6 break rest_init
```



在这里发现了qemu程序的一个问题，该程序无法自动启动

所以在这里想要更新一下qemu

于是先到qemu.org上下载了最新版本的qemu

然后直接放在了ubuntu下面进行了解压

然后cd到当那个文件夹下
并输入一下指令

```
1 ./configure
2 make
3 make install
```

在./configure时发现少了一些包，又装了一下这些包
接下来清除编译过程中产生的临时文件和过程中产生的文件

```
1 make clean
2 make distclean
```

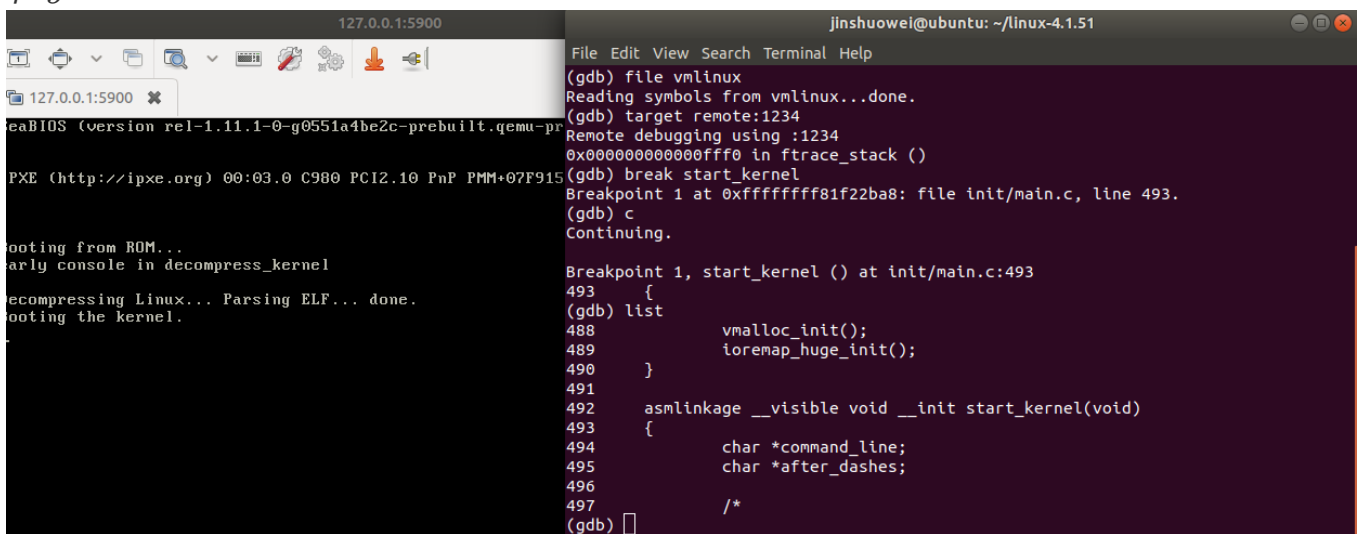
此时需要用VNC连接远程桌面来访问qemu

此时需要

跟踪内核的启动过程

知道了如何跟踪内核后，需要有目的地跟踪

这时候发现由于linux内核问题，在换到ubuntu17.10.1的时候booting的时候无法启动，这里开始测试内核linux 4.1.51



```
127.0.0.1:5900
SeaBIOS (version rel-1.11.1-0-g0551a4be2c-prebuilt.qemu-pr
PXE (http://ipxe.org) 00:03:0 C980 PCI2.10 PnP PMM+07F915
booting from ROM...
early console in decompress_kernel
decompressing Linux... Parsing ELF... done.
booting the kernel.

(gdb) file vmlinux
Reading symbols from vmlinux...done.
(gdb) target remote:1234
Remote debugging using :1234
0x0000000000000000 in ftrace_stack ()
(gdb) break start_kernel
Breakpoint 1 at 0xffffffff81f22ba8: file init/main.c, line 493.
(gdb) c
Continuing.

Breakpoint 1, start_kernel () at init/main.c:493
493 {
(gdb) list
488         vmalloc_init();
489         ioremap_huge_init();
490     }
491
492     asmlinkage __visible void __init start_kernel(void)
493     {
494         char *command_line;
495         char *after_dashes;
496
497         /*
(gdb) █
```

start_kernel

现在来研究一下start_kernel函数的源代码

```
1 asmlinkage __visible void __init start_kernel(void)
2 {
3     char *command_line;
4     char *after_dashes;
5
6     /*
7      * Need to run as early as possible, to initialize the
8      * lockdep hash:
```

```

9      */
10     lockdep_init();
11     set_task_stack_end_magic(&init_task);
12     smp_setup_processor_id();
13     debug_objects_early_init();
14
15     /*
16      * Set up the the initial canary ASAP:
17      */
18     boot_init_stack_canary();
19
20     cgroup_init_early();
21
22     local_irq_disable();
23     early_boot_irqs_disabled = true;
24
25     /*
26      * Interrupts are still disabled. Do necessary setups, then
27      * enable them
28      */
29     boot_cpu_init();
30     page_address_init();
31     pr_notice("%s", linux_banner);
32     setup_arch(&command_line);
33     mm_init_cpumask(&init_mm);
34     setup_command_line(command_line);
35     setup_nr_cpu_ids();
36     setup_per_cpu_areas();
37     smp_prepare_boot_cpu(); /* arch-specific boot-cpu hooks */
38
39     build_all_zonelists(NULL, NULL);
40     page_alloc_init();
41
42     pr_notice("Kernel command line: %s\n", boot_command_line);
43     parse_early_param();
44     after_dashes = parse_args("Booting kernel",
45                               static_command_line, __start__param,
46                               __stop__param - __start__param,
47                               -1, -1, &unknown_bootoption);
48     if (!IS_ERR_OR_NULL(after_dashes))
49         parse_args("Setting init args", after_dashes, NULL, 0, -1, -1,
50                   set_init_arg);
51
52     jump_label_init();
53
54     /*
55      * These use large bootmem allocations and must precede
56      * kmem_cache_init()
57      */
58     setup_log_buf(0);
59     pidhash_init();
60     vfs_caches_init_early();
61     sort_main_extable();
62     trap_init();
63     mm_init();
64

```



```

65  /*
66   * Set up the scheduler prior starting any interrupts (such as the
67   * timer interrupt). Full topology setup happens at smp_init()
68   * time - but meanwhile we still have a functioning scheduler.
69   */
70  sched_init();
71  /*
72   * Disable preemption - early bootup scheduling is extremely
73   * fragile until we cpu_idle() for the first time.
74   */
75  preempt_disable();
76  if (WARN(!irqs_disabled(),
77          "Interrupts were enabled *very* early, fixing it\n"))
78      local_irq_disable();
79  idr_init_cache();
80  rcu_init();
81
82  /* trace_printk() and trace points may be used after this */
83  trace_init();
84
85  context_tracking_init();
86  radix_tree_init();
87  /* init some links before init_ISA_irqs() */
88  early_irq_init();
89  init_IRQ();
90  tick_init();
91  rcu_init_nohz();
92  init_timers();
93  hrtimers_init();
94  softirq_init();
95  timekeeping_init();
96  time_init();
97  sched_clock_postinit();
98  perf_event_init();
99  profile_init();
100 call_function_init();
101 WARN(!irqs_disabled(), "Interrupts were enabled early\n");
102 early_boot_irqs_disabled = false;
103 local_irq_enable();
104
105 kmem_cache_init_late();
106
107 /*
108  * HACK ALERT! This is early. We're enabling the console before
109  * we've done PCI setups etc, and console_init() must be aware of
110  * this. But we do want output early, in case something goes wrong.
111  */
112 console_init();
113 if (panic_later)
114     panic("Too many boot %s vars at `%s'", panic_later,
115         panic_param);
116
117 lockdep_info();
118
119 /*
120  * Need to run this when irqs are enabled, because it wants

```



```

121     * to self-test [hard/soft]-irqs on/off lock inversion bugs
122     * too:
123     */
124     locking_selftest();
125
126 #ifdef CONFIG_BLK_DEV_INITRD
127     if (initrd_start && !initrd_below_start_ok &&
128         page_to_pfn(virt_to_page((void *)initrd_start)) < min_low_pfn) {
129         pr_crit("initrd overwritten (0x%08lx < 0x%08lx) - disabling it.\n",
130             page_to_pfn(virt_to_page((void *)initrd_start)),
131             min_low_pfn);
132         initrd_start = 0;
133     }
134 #endif
135     page_ext_init();
136     debug_objects_mem_init();
137     kmemleak_init();
138     setup_per_cpu_pageset();
139     numa_policy_init();
140     if (late_time_init)
141         late_time_init();
142     sched_clock_init();
143     calibrate_delay();
144     pidmap_init();
145     anon_vma_init();
146     acpi_early_init();
147 #ifdef CONFIG_X86
148     if (efi_enabled(EFI_RUNTIME_SERVICES))
149         efi_enter_virtual_mode();
150 #endif
151 #ifdef CONFIG_X86_ESPFIX64
152     /* Should be run before the first non-init thread is created */
153     init_espfix_bsp();
154 #endif
155     thread_info_cache_init();
156     cred_init();
157     fork_init();
158     proc_caches_init();
159     buffer_init();
160     key_init();
161     security_init();
162     dbg_late_init();
163     vfs_caches_init(totalram_pages);
164     signals_init();
165     /* rootfs populating might need page-writeback */
166     page_writeback_init();
167     proc_root_init();
168     nsfs_init();
169     cpuset_init();
170     cgroup_init();
171     taskstats_init_early();
172     delayacct_init();
173
174     check_bugs();
175
176     acpi_subsystem_init();

```

```

177     sfi_init_late();
178
179     if (efi_enabled(EFI_RUNTIME_SERVICES)) {
180         efi_late_init();
181         efi_free_boot_services();
182     }
183
184     ftrace_init();
185
186     /* Do the rest non-__init'ed, we're now alive */
187     rest_init();
188 }
189
190 /* Call all constructor functions linked into the kernel. */
191 static void __init do_ctors(void)
192 {
193 #ifdef CONFIG_CONSTRUCTORS
194     ctor_fn_t *fn = (ctor_fn_t *) __ctors_start;
195
196     for (; fn < (ctor_fn_t *) __ctors_end; fn++)
197         (*fn)();
198 #endif
199 }
200
201 bool initcall_debug;
202 core_param(initcall_debug, initcall_debug, bool, 0644);
203
204 #ifdef CONFIG_KALLSYMS
205 struct blacklist_entry {
206     struct list_head next;
207     char *buf;
208 };
209
210

```

无论研究什么内核模块，都应该先来研究start_kernel，因为所有内核重要的模块都会在这里初始化

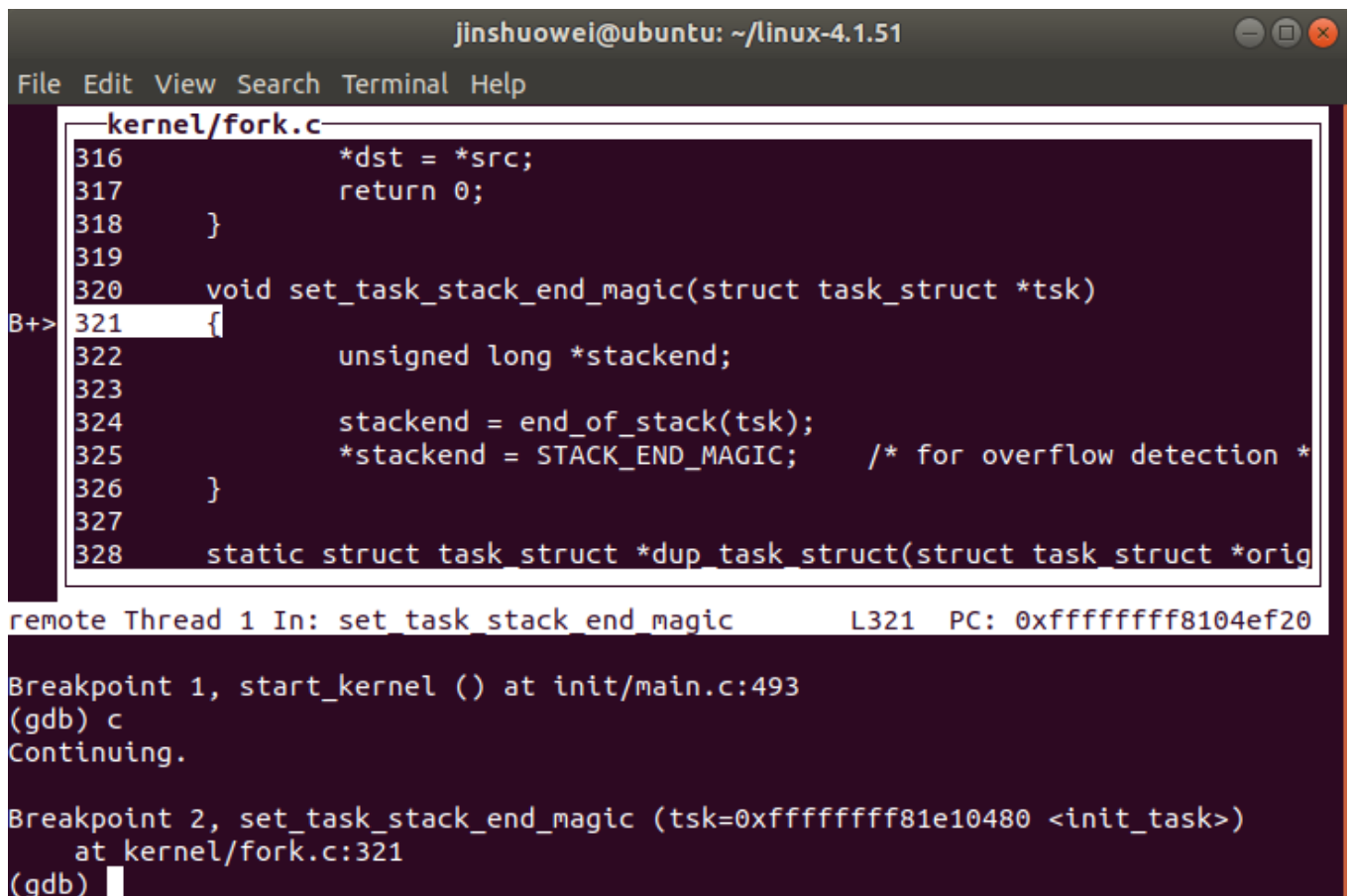
- 初始化lock validator (lockdep_init)
- 初始化高端内存的映射表 (page_address_init())
- 设置操作系统的第一个进程init (set_task_stack_end_magic)
- 设置obj_hash, obj_static_pool两个全局变量
- 内核架构相关初始化函数，包含处理器相关参数的初始化、内核启动参数的获取和前期处理、内存子系统的早期初始化 (setup_arch)
- 初始化文件系统的缓冲区，并计算最大可以使用的文件缓存 (buffer_init())
- 对内核陷阱异常进行初始化，初始化一些中断向量，在ARM系统里是空函数，没有任何的初始化 (trap_init())
- 初始化软件中断 (softirq_init())
- 设置内存页分配通知器 (page_alloc_init)

- 中断描述符号表初始化 (trap_init)
- 初始化内存管理器 (mm_init)
- 对进程调度器的数据结构进行初始化 (sched_init)
- 初始化直接读拷贝更新的锁机制 (rcu_init)
- 初始化内核信号队列 (signals_init())
- 初始化系统时钟，开启一个硬件定时器 (time_init)
- 启用中断操作 (local_irq_enable)
- 检查CPU配置、FPU等是否非法使用不具备的功能，检查CPU BUG，软件规避BUG (check_bugs())
- 控制台初始化 (console_init)
- 完成剩余部分 (rest_init)

现在我们将断点设置到set_task_stack_end_magic()函数处

因为在start_kernel()中，我们看到调用的第一个函数是lockdep_init(),该函数会初始化内核死锁检测机制的hash table,接下来就是set_task_stack_end_magic()函数。所以先看一下set_task_stack_end_magic()函数

set_task_stack_end_magic()



```

jinsuwei@ubuntu: ~/linux-4.1.51
File Edit View Search Terminal Help
kernel/fork.c
316         *dst = *src;
317         return 0;
318     }
319
320     void set_task_stack_end_magic(struct task_struct *tsk)
321     {
322         unsigned long *stackend;
323
324         stackend = end_of_stack(tsk);
325         *stackend = STACK_END_MAGIC;    /* for overflow detection */
326     }
327
328     static struct task_struct *dup_task_struct(struct task_struct *orig
remote Thread 1 In: set_task_stack_end_magic      L321  PC: 0xffffffff8104ef20
Breakpoint 1, start_kernel () at init/main.c:493
(gdb) c
Continuing.
Breakpoint 2, set_task_stack_end_magic (tsk=0xffffffff81e10480 <init_task>)
at kernel/fork.c:321
(gdb)

```

setup_arch

```
jinshuowei@ubuntu: ~/linux-4.1.51
File Edit View Search Terminal Help

arch/x86/kernel/setup.c
857      *
858      * Note: On x86_64, fixmaps are ready for use even before this is c
859      */
860
861      void __init setup_arch(char **cmdline_p)
862      {
863          memblock_reserve(__pa_symbol(_text),
864                          (unsigned long)__bss_stop - (unsigned long)
865                          _text);
866          early_reserve_initrd();
867
868          /*
869           * At this point everything still needed from the boot load
870           *
871           *
872           *
873           *
874           *
875           *
876           *
877           *
878           *
879           *
880           *
881           *
882           *
883           *
884           *
885           *
886           *
887           *
888           *
889           *
890           *
891           *
892           *
893           *
894           *
895           *
896           *
897           *
898           *
899           *
900           *
901           *
902           *
903           *
904           *
905           *
906           *
907           *
908           *
909           *
910           *
911           *
912           *
913           *
914           *
915           *
916           *
917           *
918           *
919           *
920           *
921           *
922           *
923           *
924           *
925           *
926           *
927           *
928           *
929           *
930           *
931           *
932           *
933           *
934           *
935           *
936           *
937           *
938           *
939           *
940           *
941           *
942           *
943           *
944           *
945           *
946           *
947           *
948           *
949           *
950           *
951           *
952           *
953           *
954           *
955           *
956           *
957           *
958           *
959           *
960           *
961           *
962           *
963           *
964           *
965           *
966           *
967           *
968           *
969           *
970           *
971           *
972           *
973           *
974           *
975           *
976           *
977           *
978           *
979           *
980           *
981           *
982           *
983           *
984           *
985           *
986           *
987           *
988           *
989           *
990           *
991           *
992           *
993           *
994           *
995           *
996           *
997           *
998           *
999           *
1000          */
1001      }

remote Thread 1 In: setup_arch                                L862  PC: 0xffffffff81f25a6f
at kernel/fork.c:321
(gdb) c
Continuing.

Breakpoint 3, setup_arch (
  cmdline_p=0xffffffff81e03f90 <init_thread_union+16272>)
at arch/x86/kernel/setup.c:862
(gdb) 
```

下面找出代码进行分析

```
1 void __init setup_arch(char **cmdline_p)
2 {
3     struct tag *tags = (struct tag *)&init_tags; //定义了一个默认的内核参数列表
4     struct machine_desc *mdesc; //设备描述的结构体
5     char *from = default_command_line;
6     init_tags.mem.start = PHYS_OFFSET;
7     unwind_init();
8     setup_processor(); //汇编的CPU初始化部分
9     mdesc = setup_machine(machine_arch_type);
10    machine_desc = mdesc;
11    machine_name = mdesc->name;
12    //下面一部分通过匹配struct machine_desc结构体数据, 初始化一些全局变量
13    //通过struct machine_desc 中的soft_reboot设置重启类型
14    if (mdesc->soft_reboot)
15        reboot_setup("s");
16
17    //检查bootloader是否传入参数, 如果传入, 则给tags赋值, 若果没有传入则传递默认的启动参数地址
18    if (__atags_pointer)
19        tags = phys_to_virt(__atags_pointer);
20    else if (mdesc->boot_params) {
21#ifdef CONFIG_MMU
22        /*
23         * We still are executing with a minimal MMU mapping created
24         * with the presumption that the machine default for this
25         * is located in the first MB of RAM. Anything else will
26         * fault and silently hang the kernel at this point.
27         */
28        if (mdesc->boot_params < PHYS_OFFSET ||
29            mdesc->boot_params >= PHYS_OFFSET + SZ_1M) {
```

```

30         printk(KERN_WARNING
31                "Default boot params at physical 0x%08lx out of reach\n",
32                mdesc->boot_params);
33     } else
34 #endif
35     {
36         tags = phys_to_virt(mdesc->boot_params);
37     }
38 }
39
40 #if defined(CONFIG_DEPRECATED_PARAM_STRUCT)
41 /*
42  * If we have the old style parameters, convert them to
43  * a tag list.
44  */
45     if (tags->hdr.tag != ATAG_CORE) //内核参数列表第一项为ATAG_CORE
46         convert_to_tag_list(tags);
47     //如果不是，这需要转换成新的内核参数类型，新的内核参数类型用下面的structtag结构表示
48 #endif
49     if (tags->hdr.tag != ATAG_CORE)
50         tags = (struct tag *)&init_tags;
51     //如果没有内核参数，则选用默认的内核参数，在init_tags文件中有定义
52
53     if (mdesc->fixup)
54         mdesc->fixup(mdesc, tags, &from, &meminfo);
55     //用选用的内核参数列表填充meminfo，fixup函数出现在注册machine_desc中，即MACHINE_START、
56     MACHINE_END定义中，这个函数，有些板子有，但在2410中没有定义这个函数。
57
58     if (tags->hdr.tag == ATAG_CORE) {
59         if (meminfo.nr_banks != 0) //如果内存被初始化过
60             squash_mem_tags(tags);
61         //如果是tag_list,那么如果系统已经创建了默认的meminfo.nr_banks,清除tags中关于MEM的参数，以免再
62         次被初始化
63
64         save_atags(tags);
65         parse_tags(tags); //做一些针对各个tags的处理
66     }
67
68     //解析内核参数列表，然后调用内核参数列表的处理函数对这些参数进行处理。比如，如果列表为命令行，则最终会用
69     parse_tag_cmdlin函数进行解析，这个函数用_tagtable编译连接到了内核里
70
71     init_mm.start_code = (unsigned long) _text;
72     init_mm.end_code   = (unsigned long) _etext;
73     init_mm.end_data   = (unsigned long) _edata;
74     init_mm.brk        = (unsigned long) _end;
75     //记录了内核代码的起始，结束虚拟地址
76
77     /* parse_early_param needs a boot_command_line */
78     strcpy(boot_command_line, from, COMMAND_LINE_SIZE);
79
80     /* populate cmd_line too for later use, preserving boot_command_line */
81     strcpy(cmd_line, boot_command_line, COMMAND_LINE_SIZE);
82     *cmdline_p = cmd_line;
83     //将boot_command_line复制到cmd_line中
84
85     parse_early_param(); //解释命令行参数
86

```

```

83     arm_memblock_init(&meminfo, mdesc); //将设备实体登记注册到总线空间链表中去
84
85     paging_init(mdesc);
86     request_standard_resources(mdesc);
87
88     #ifdef CONFIG_SMP
89         if (is_smp())
90             smp_init_cpus(); //要配置CONFIG_KEXEC, 否则为空函数, 2410中没有配置
91     #endif
92     reserve_crashkernel();
93
94     cpu_init(); //初始化一个CPU, 并设置一个per-CPU栈
95     tcm_init(); //初始化ARM内部的TCM (紧耦合内存)
96
97     #ifdef CONFIG_MULTI_IRQ_HANDLER
98         handle_arch_irq = mdesc->handle_irq;
99     #endif
100
101     #ifdef CONFIG_VT
102     #if defined(CONFIG_VGA_CONSOLE)
103         conswitchp = &vga_con;
104     #elif defined(CONFIG_DUMMY_CONSOLE)
105         conswitchp = &dummy_con;
106     #endif
107     #endif
108     early_trap_init(); //对中断向量表进行早期的初始化
109     //如果设备描述结构体定义了init_early函数 (应该是早期初始化之意), 则在这里调用
110     if (mdesc->init_early)
111         mdesc->init_early();
112 }

```

trap_init

File Edit View Search Terminal Help

arch/x86/kernel/traps.c

```

949         set_intr_gate(X86_TRAP_PF, page_fault);
950     #endif
951 }
952
953 void __init trap_init(void)
954 {
955     int i;
956
957     #ifdef CONFIG_EISA
958         void __iomem *p = early_ioremap(0x0FFFD9, 4);
959
960         if (readl(p) == 'E' + ('I'<<8) + ('S'<<16) + ('A'<<24))
961             EISA_bus = 1;

```

remote Thread 1 In: trap_init

L954 PC: 0xffffffff81f25122

Breakpoint 3, setup_arch (

cmdline_p=0xffffffff81e03f90 <init_thread_union+16272>)

at arch/x86/kernel/setup.c:862

(gdb) c

Continuing.

Breakpoint 4, trap_init () at arch/x86/kernel/traps.c:954

(gdb)

```

1 void __init trap_init(void)
2 {
3     int i;
4
5     #ifdef CONFIG_EISA
6         void __iomem *p = early_ioremap(0x0FFFD9, 4);
7
8         if (readl(p) == 'E' + ('I'<<8) + ('S'<<16) + ('A'<<24))
9             EISA_bus = 1;
10         early_iounmap(p, 4);
11     #endif
12
13     set_intr_gate(X86_TRAP_DE, divide_error);
14     set_intr_gate_ist(X86_TRAP_NMI, &nmi, NMI_STACK);
15     /* int4 can be called from all */
16     set_system_intr_gate(X86_TRAP_OF, &overflow);
17     set_intr_gate(X86_TRAP_BR, bounds);
18     set_intr_gate(X86_TRAP_UD, invalid_op);
19     set_intr_gate(X86_TRAP_NM, device_not_available);
20     #ifdef CONFIG_X86_32
21         set_task_gate(X86_TRAP_DF, GDT_ENTRY_DOUBLEFAULT_TSS);
22     #else
23         set_intr_gate_ist(X86_TRAP_DF, &double_fault, DOUBLEFAULT_STACK);
24     #endif
25     set_intr_gate(X86_TRAP_OLD_MF, coprocessor_segment_overrun);
26     set_intr_gate(X86_TRAP_TS, invalid_TSS);
27     set_intr_gate(X86_TRAP_NP, segment_not_present);
28     set_intr_gate(X86_TRAP_SS, stack_segment);
29     set_intr_gate(X86_TRAP_GP, general_protection);
30     set_intr_gate(X86_TRAP_SPURIOUS, spurious_interrupt_bug);

```



```

31     set_intr_gate(X86_TRAP_MF, coprocessor_error);
32     set_intr_gate(X86_TRAP_AC, alignment_check);
33 #ifdef CONFIG_X86_MCE
34     set_intr_gate_ist(X86_TRAP_MC, &machine_check, MCE_STACK);
35 #endif
36     set_intr_gate(X86_TRAP_XF, simd_coprocessor_error);
37
38     /* Reserve all the builtin and the syscall vector: */
39     for (i = 0; i < FIRST_EXTERNAL_VECTOR; i++)
40         set_bit(i, used_vectors);
41
42 #ifdef CONFIG_IA32_EMULATION
43     set_system_intr_gate(IA32_SYSCALL_VECTOR, ia32_syscall);
44     set_bit(IA32_SYSCALL_VECTOR, used_vectors);
45 #endif
46
47 #ifdef CONFIG_X86_32
48     set_system_trap_gate(SYSCALL_VECTOR, &system_call);
49     set_bit(SYSCALL_VECTOR, used_vectors);
50 #endif
51
52     /*
53      * Set the IDT descriptor to a fixed read-only location, so that the
54      * "sidt" instruction will not leak the location of the kernel, and
55      * to defend the IDT against arbitrary memory write vulnerabilities.
56      * It will be reloaded in cpu_init() */
57     __set_fixmap(FIX_R0_IDT, __pa_symbol(idt_table), PAGE_KERNEL_RO);
58     idt_descr.address = fix_to_virt(FIX_R0_IDT);
59
60     /*
61      * Should be a barrier for any external CPU state:
62      */
63     cpu_init();
64
65     /*
66      * X86_TRAP_DB and X86_TRAP_BP have been set
67      * in early_trap_init(). However, ITS works only after
68      * cpu_init() loads TSS. See comments in early_trap_init().
69      */
70     set_intr_gate_ist(X86_TRAP_DB, &debug, DEBUG_STACK);
71     /* int3 can be called from all */
72     set_system_intr_gate_ist(X86_TRAP_BP, &int3, DEBUG_STACK);
73
74     x86_init.irqs.trap_init();
75
76 #ifdef CONFIG_X86_64
77     memcpy(&debug_idt_table, &idt_table, IDT_ENTRIES * 16);
78     set_nmi_gate(X86_TRAP_DB, &debug);
79     set_nmi_gate(X86_TRAP_BP, &int3);
80 #endif
81 }
82

```

rest_init()

```
jinshuowei@ubuntu: ~/linux-4.1.51
File Edit View Search Terminal Help

init/main.c
379      */
380
381      static __initdata DECLARE_COMPLETION(kthreadd_done);
382
383      static noinline void __init_refok rest_init(void)
384      {
385          int pid;
386
387          rcu_scheduler_starting();
388          smpboot_thread_init();
389          /*
390           * We need to spawn init first so that it obtains pid 1, ho
391           * the init task will end up wanting to create kthreads, wh

remote Thread 1 In: rest_init L384 PC: 0xffffffff81893fc0
Continuing.

Breakpoint 4, trap_init () at arch/x86/kernel/traps.c:954
(gdb) c
Continuing.

Breakpoint 5, rest_init () at init/main.c:384
(gdb) 
```

rest_init()函数分析

- (1)rest_init中调用kernel_thread函数启动了2个内核线程，分别是：kernel_init和kthreadd
- (2)调用schedule函数开启了内核的调度系统，从此linux系统开始转起来了。
- (3)rest_init最终调用cpu_idle函数结束了整个内核的启动。也就是说linux内核最终结束了一个函数cpu_idle。这个函数里面肯定是死循环。
- (4)简单来说，linux内核最终的状态是：有事干的时候去执行有意义的工作（执行各个进程任务），实在没活干的时候就死循环（实际上死循环也可以看成是一个任务）。
- (5)之前已经启动了内核调度系统，调度系统会负责考评系统中所有的进程，这些进程里面只有有哪个需要被运行，调度系统就会终止cpu_idle死循环进程（空闲进程）转而去执行有意义的干活的进程。这样操作系统就转起来了。

kernel_init()

```
jinshuowei@ubuntu: ~/linux-4.1.51
File Edit View Search Terminal Help

init/main.c
924     }
925
926     static ninline void __init kernel_init_freeable(void);
927
928     static int __ref kernel_init(void *unused)
B+> 929     {
930         int ret;
931
932         kernel_init_freeable();
933         /* need to finish all async __init code before freeing the
934         async_synchronize_full();
935         free_initmem();
936         mark_rodata_ro();

remote Thread 1 In: kernel_init          L929  PC: 0xffffffff81894040
Continuing.

Breakpoint 2, dup_task_struct (orig=<optimized out>) at kernel/fork.c:361
(gdb) c
Continuing.

Breakpoint 6, kernel_init (unused=0x0 <irq_stack_union>) at init/main.c:929
(gdb) █
```

1. 打开控制台设备

```
1     if (sys_open((const char __user *) "/dev/console", 0_RDWR, 0) < 0)
2         printk(KERN_WARNING "Warning: unable to open an initial console.\n");
3
4     (void) sys_dup(0);
5     (void) sys_dup(0);
```

2. 挂载根文件系统

```
1     if (sys_access((const char __user *) ramdisk_execute_command, 0) != 0) {
2         ramdisk_execute_command = NULL;
3         prepare_namespace();
4     }
```

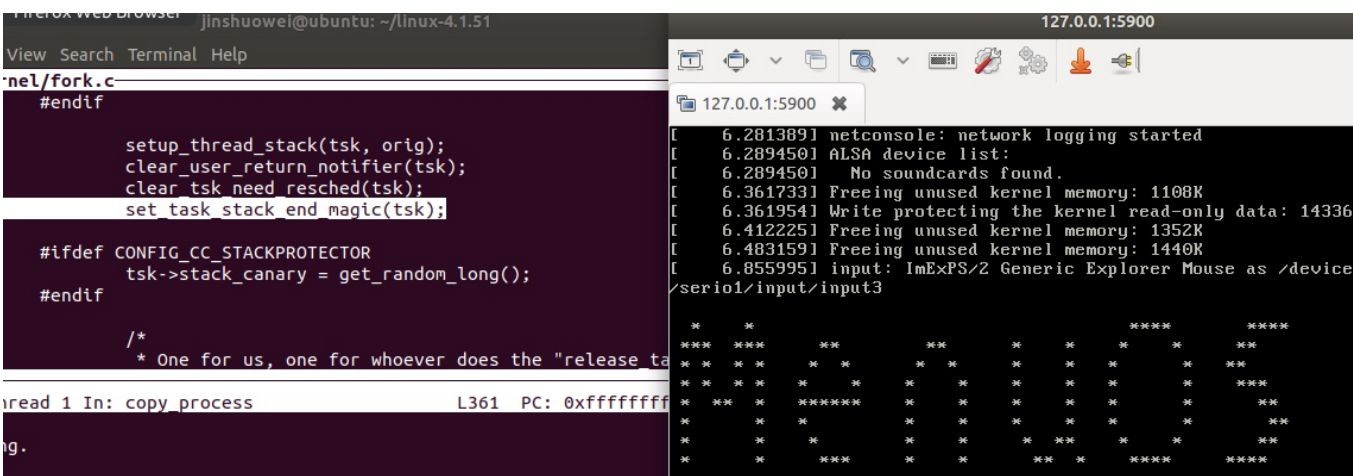
- prepare_namespace这个函数负责挂载根文件系统
- 如果内核挂载根文件系统成功，则会打印出：VFS: Mounted root (xxxx filesystem) on device xxxx.
如果挂载根文件系统失败，则会打印：No filesystem could mount root, tried: xxxx

3. 启动init进程

```
1 if (ramdisk_execute_command) {
2     run_init_process(ramdisk_execute_command);
3     printk(KERN_WARNING "Failed to execute %s\n",
4             ramdisk_execute_command);
5 }
```

```
6
7  /*
8   * We try each of these until one succeeds.
9   *
10  * The Bourne shell can be used instead of init if we are
11  * trying to recover a really broken machine.
12  */
13  if (execute_command) {
14      run_init_process(execute_command);
15      printk(KERN_WARNING "Failed to execute %s. Attempting "
16             "defaults...\n", execute_command);
17  }
18  run_init_process("/sbin/init");
19  run_init_process("/etc/init");
20  run_init_process("/bin/init");
21  run_init_process("/bin/sh");
22
23  panic("No init found. Try passing init= option to kernel. "
24        "See Linux Documentation/init.txt for guidance.");
```

启动完成



实验总结

本次实验麻烦之处其实在于环境的配置，对于linux启动事件的跟踪只要像助教所说那样对于linux源代码进行比较一些研究之后不难分析。

但是在配置环境的时候却遇到了很多麻烦，一开始在ubuntu 16.04下面配置，并选的linux版本号是3.18后来发现qemu会在这种情况下卡死，这个问题不好解决。后来于是换到ubuntu17.10下进行实验，此时却发现gcc版本过高，编译较老的内核的时候又会出错，于是将gcc降级到4.8。后来发现因为内核的原因，在测试的时候qemu会卡在booting界面无法启动，于是将内核版本号升级到4.15最终终于成功。

