

# OSH Lab 1

## 实验环境配置

本次实验所采用的操作环境

- Ubuntu 17.10 x64 (Linux version 4.13.0-32-generic (buildd@lgw01-amd64-016) (gcc version 7.2.0 (Ubuntu 7.2.0-8ubuntu3)))
- gdb 8.0.1
- gcc 7.2.0
- qemu 2.12.0-rc0
- busybox 1.28.1

通过qemu模拟器+gdb追踪分析linux-4.1.50内核的启动过程。

## 环境搭建

### 内核编译

从[kernel.org](http://kernel.org)获取4.1.50版的Linux内核源码，并在本地编译。

```
1 sudo apt-get install libncurses5-dev libncursesw5-dev
2 make clean
3 make menuconfig
4 make x86_64_defconfig
5 make
```

在make menuconfig一步，进入内核配置菜单，选择Kernel hacking->Compile-time checks and compiler options->勾选Compile the kernel with debug info

### qemu编译

之后从[qemu.org](http://qemu.org)获取qemu 2.12.0-rc0版，并编译anzhuna

```
1 sudo apt-get install git libgl2.0-dev libfdt-dev libpixman-1-dev zlib1g-dev
2 make clean
3 ./configure
4 make
5 make install
```

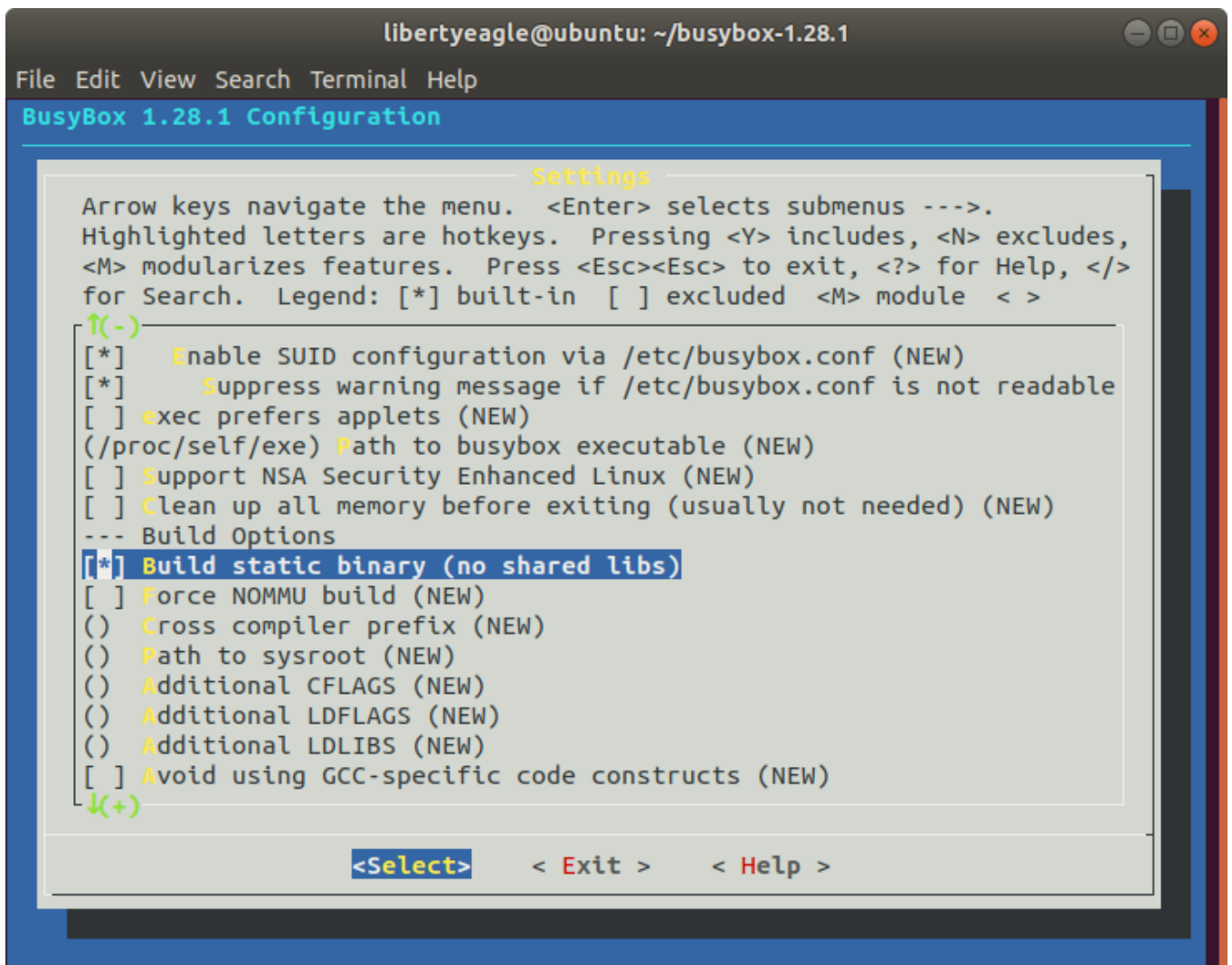
### 用busybox制作根文件系统 (RootFS)

获取busybox 1.28.1版本

wget http://busybox.net/downloads/busybox-1.28.1.tar.bz2

```
1 make menuconfig
2 make
3 make install
```

其中menuconfig中勾选Settings->Build Options->Build static binary (no shared libs)



更新文件系统

```
1 cd _install
2 mkdir proc sys dev etc etc/init.d
3 touch _install/etc/init.d/rcS
```

然后编辑rcS文件，修改为以下内容

```
1 #!/bin/sh
2 mount -t proc none /proc
3 mount -t sysfs none /sys
4 /sbin/mdev -s
```

接下来修改rcS属性为可执行

```
chmod +x _install/etc/init.d/rcS
```

创建镜像

```
1 cd _install
2 find . | cpio -o --format=newc > ../rootfs.img
```

调试

我们使用下述命令启动qemu模拟器

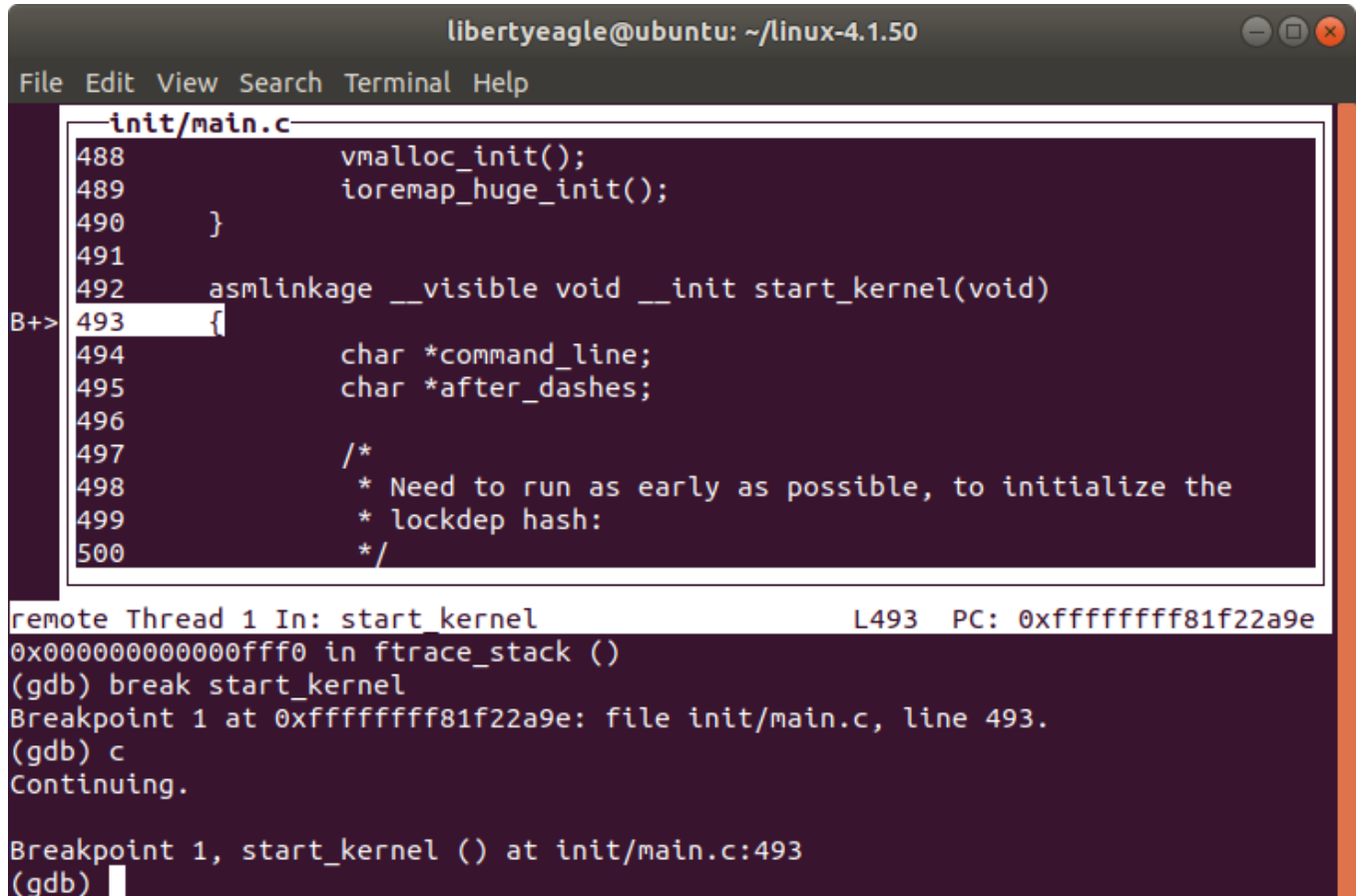
```
qemu-system-x86_64 -kernel ./arch/x86_64/boot/bzImage -initrd rootfs.img -append
"console=tty1 root=/dev/ram rdinit=/sbin/init" -S -s
```

然后使用gdb调试，为了方便，我们使用tui模式操作。将第一个断点设置在start\_kernel()函数处

```
1 gdb -tui
```

```
2 (gdb) file vmlinux # 加载符号表
3 (gdb) target remote:1234 # 连接远程目标
4 (gdb) break start_point
5 (gdb) c # continue
```

之后我们可以看到，qemu停在了内核引导的位置



The screenshot shows a GDB terminal window titled "libertyeagle@ubuntu: ~/linux-4.1.50". The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The main content area displays the source code of "init/main.c" with line numbers 488 to 500. The code includes calls to "vmalloc\_init()", "ioremap\_huge\_init()", and the start of the "start\_kernel" function. A breakpoint is set at line 493, and the program has stopped at that location. The GDB prompt shows "remote Thread 1 In: start\_kernel" and "PC: 0xffffffff81f22a9e". The user has entered "break start\_kernel", "c", and "c" to continue the execution.

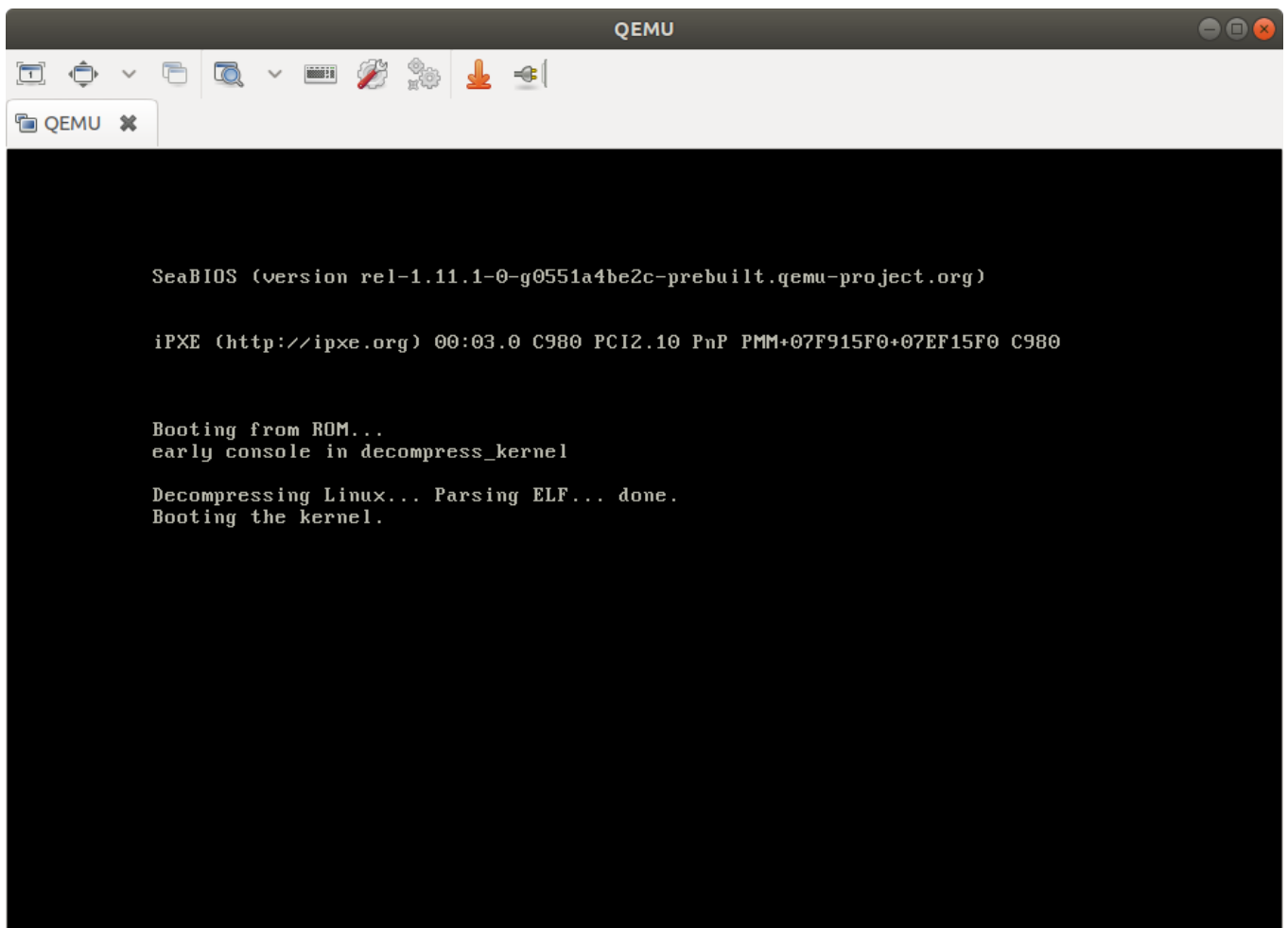
```
libertyeagle@ubuntu: ~/linux-4.1.50
File Edit View Search Terminal Help

init/main.c
488         vmalloc_init();
489         ioremap_huge_init();
490     }
491
492     asmlinkage __visible void __init start_kernel(void)
B+> 493 {
494         char *command_line;
495         char *after_dashes;
496
497         /*
498          * Need to run as early as possible, to initialize the
499          * lockdep hash:
500          */

remote Thread 1 In: start_kernel                L493  PC: 0xffffffff81f22a9e
0x0000000000000000 in ftrace_stack ()
(gdb) break start_kernel
Breakpoint 1 at 0xffffffff81f22a9e: file init/main.c, line 493.
(gdb) c
Continuing.

Breakpoint 1, start_kernel () at init/main.c:493
(gdb) 
```

使用VNC连接远程桌面，可以看到



## start\_kernel - Linux内核的main函数

### 总揽

在完成内核初始化前的一系列预备工作后，`x86_64_start_reservations()`函数便会调用`start_kernel`。`start_kernel()`中调用了一系列初始化函数，以完成核心数据结构的初始化。它的主要目的是完成内核的初始化过程并且启动第一个init进程。

`start_kernel()`会调用一系列初始化函数来设置中断，执行进一步的内存配置，并加载初始 RAM 磁盘

**start\_kernel**的主要流程

- 初始化lock validator (`lockdep_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
- 设定不确定的stack canary，阻止buffer overflow攻击 (`boot_init_stack_canary`)
- 初始化control groups (`cgroup_init_early`)
- 关闭中断操作 (`local_irq_disable`)
- 完成页地址的初始化 (`page_address_init`)
- 打印linux\_banner (`pr_notice`)
- 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`)
- 为SMP系统里引导CPU进行准备工作 (`smp_prepare_boot_cpu`)
- 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)
- 中断管理系统的早期初始化 (early\_irq\_init)
- 初始化软件中断 (softirq\_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)
- 控制台初始化 (console\_init)
- 完成剩余部分 (rest\_init)

在刚开始我们可以看到两个字符串变量 `command_line` 和 `after_dashes`，我们可以用 `gdb` 看到他们的值

```

libertyeagle@ubuntu: ~/linux-4.1.50
File Edit View Search Terminal Help

init/main.c
491
492  asmlinkage __visible void __init start_kernel(void)
B+ 493  {
494      char *command_line;
495      char *after_dashes;
496
497      /*
498       * Need to run as early as possible, to initialize the
499       * lockdep hash:
500       */
501      lockdep_init();
B+> 502      set task stack end magic(&init_task);
503      smp_setup_processor_id();

remote Thread 1 In: start_kernel                L502  PC: 0xffffffff81f22a9f

Breakpoint 2, start_kernel () at init/main.c:502
(gdb) display command_line
1: command_line = 0xffffffff81f85c8f <memblock_reserve+76> "[A\\]\\303UH\211\345A
WAVAUATSH\211\363H\367\323H\203\354\020H9\323", <incomplete sequence \307>
(gdb) display after_dashes
2: after_dashes = <optimized out>
(gdb)

```

可以看到该部分的一些汇编代码

```
libertyeagle@ubuntu: ~/linux-4.1.50
File Edit View Search Terminal Help

init/main.c
489         ioremap_huge_init();
490     }
491
492     asmlinkage __visible void __init start_kernel(void)
B+> 493 {
494         char *command_line;
495         char *after_dashes;
496
497         /*
498          * Need to run as early as possible, to initialize the
499          * lockdep hash:
500          */
501         lockdep_init();

remote Thread 1 In: start kernel L493 PC: 0xffffffff81f22a9e
Dump of assembler code for function start_kernel:
=> 0xffffffff81f22a9e <+0>:      push    %rbp
0xffffffff81f22a9f <+1>:      mov     $0xffffffff81e10480,%rdi
0xffffffff81f22aa6 <+8>:      mov     %rsp,%rbp
0xffffffff81f22aa9 <+11>:     push    %r14
0xffffffff81f22aab <+13>:     push    %r13
0xffffffff81f22aad <+15>:     push    %r12
---Type <return> to continue, or q <return> to quit---
```

在start\_kernel(), 我们看到其调用的第一个函数是lockdep\_init(), 这个函数会初始化内核死锁检测机制的哈希表。接下来, 会调用set\_task\_stack\_end\_magic(&init\_task)函数

## set\_task\_stack\_end\_magic()

用gdb设置断点, 我们可以看到这部分的代码

```
1 set_task_stack_end_magic(struct task_struct *tsk)
2 {
3     unsigned long *stackend;
4     stackend = end_of_stack(tsk);
5     *stackend = STACK_END_MAGIC; /* for overflow detection */
6 }
```

set\_task\_stack\_end\_magic()函数将栈底地址设置为STACK\_END\_MAGIC, 作为溢出的标记 (canary) init\_task代表下面的初始任务结构

```
struct task_struct init_task = INIT_TASK(init_task);
```

其中, task\_struct存储了关于一个进程的所有信息。这个结构在include/linux/init\_task.h中定义。init\_task被INIT\_TASK宏初始化, 以完成第一个进程的配置

- 将进程的状态置为runnable
- 将进程的标记置为PF\_KTHREAD (kernel thread)
- 配置一个runnable的task list
- 配置进程的地址空间

之后会初始化进程的内存堆栈, 由thread\_union结构表示

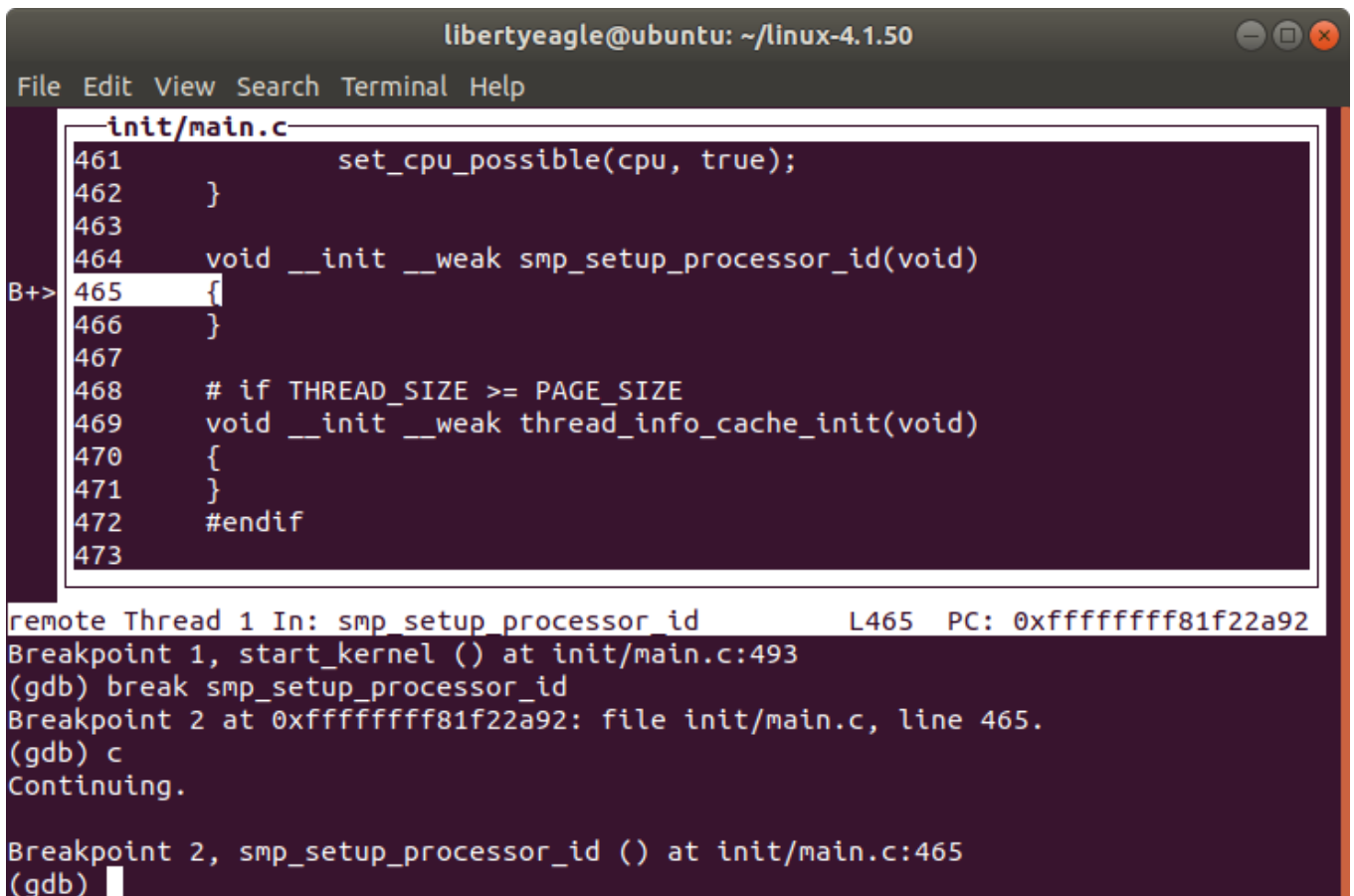
```
1 union thread_union {
2     #ifndef CONFIG_THREAD_INFO_IN_TASK
3         struct thread_info thread_info;
4     #endif
5     unsigned long stack[THREAD_SIZE/sizeof(long)];
6 };
```

在include/linux/sched.h中可以找到

其中thread\_info的定义在arch/x86/arch/x86/include/asm/thread\_info.h中

```
1 struct thread_info {
2     struct task_struct *task;      /* main task structure */
3     struct exec_domain *exec_domain; /* execution domain */
4     __u32 flags; /* low level flags */
5     __u32 status; /* thread synchronous flags */
6     __u32 cpu; /* current CPU */
7     int saved_preempt_count;
8     mm_segment_t addr_limit;
9     struct restart_block restart_block;
10    void __user *sysenter_return;
11    unsigned int sig_on_uaccess_error:1;
12    unsigned int uaccess_err:1; /* uaccess failed */
13 };
```

## smp\_setup\_processor\_id()



The screenshot shows a GDB terminal window with the title 'libertyeagle@ubuntu: ~/linux-4.1.50'. The window contains a list of menu items: File, Edit, View, Search, Terminal, Help. Below the menu is a code editor showing the file 'init/main.c'. The code is as follows:

```
461     set_cpu_possible(cpu, true);
462 }
463
464 void __init __weak smp_setup_processor_id(void)
465 {
466 }
467
468 # if THREAD_SIZE >= PAGE_SIZE
469 void __init __weak thread_info_cache_init(void)
470 {
471 }
472 #endif
473
```

Below the code editor, the GDB output shows the execution of the function:

```
remote Thread 1 In: smp_setup_processor_id          L465  PC: 0xffffffff81f22a92
Breakpoint 1, start_kernel () at init/main.c:493
(gdb) break smp_setup_processor_id
Breakpoint 2 at 0xffffffff81f22a92: file init/main.c, line 465.
(gdb) c
Continuing.

Breakpoint 2, smp_setup_processor_id () at init/main.c:465
(gdb)
```

smp\_setup\_processor\_id()中smp指的是symmetric multi-processor，与之对应的是NUMA和MPP。用以设置SMP模型的CPU ID

```
1 # if THREAD_SIZE >= PAGE_SIZE
2 void __init __weak thread_info_cache_init(void)
3 {
4 }
5 #endif
```

## cgroup\_init\_early()

```
libertyeagle@ubuntu: ~/linux-4.1.50
File Edit View Search Terminal Help

kernel/cgroup.c
5023  */
5024  int __init cgroup_init_early(void)
> 5025  {
5026      static struct cgroup_sb_opts __initdata opts;
5027      struct cgroup_subsys *ss;
5028      int i;
5029
5030      init_cgroup_root(&cgrp_dfl_root, &opts);
5031      cgrp_dfl_root.cgrp.self.flags |= CSS_NO_REF;
5032
5033      RCU_INIT_POINTER(init_task.cgroups, &init_css_set);
5034
5035      for_each_subsys(ss, i) {

remote Thread 1 In: cgroup init early L5025 PC: 0xffffffff81f3f826
Breakpoint 3 at 0xffffffff81f22abb: file init/main.c, line 504.
(gdb) c
Continuing.

Breakpoint 3, start_kernel () at init/main.c:511
(gdb) stepi
cgroup_init_early () at kernel/cgroup.c:5025
(gdb) 
```

该函数定义在kernel/cgroup.c中，在系统启动时初始化control groups，并且初始化任何需要early init的子系统  
其中cgroup\_subsys结构的定义可以在include/linux/cgroup-defs.h中找到

```
1 struct cgroup_subsys {
2     struct cgroup_subsys_state *(*css_alloc)(struct cgroup_subsys_state *parent_css);
3     int (*css_online)(struct cgroup_subsys_state *css);
4     void (*css_offline)(struct cgroup_subsys_state *css);
5     void (*css_released)(struct cgroup_subsys_state *css);
6     void (*css_free)(struct cgroup_subsys_state *css);
7     void (*css_reset)(struct cgroup_subsys_state *css);
8     void (*css_e_css_changed)(struct cgroup_subsys_state *css);
9
10    int (*can_attach)(struct cgroup_subsys_state *css,
11                    struct cgroup_taskset *tset);
12    void (*cancel_attach)(struct cgroup_subsys_state *css,
13                        struct cgroup_taskset *tset);
14    void (*attach)(struct cgroup_subsys_state *css,
15                 struct cgroup_taskset *tset);
16    void (*fork)(struct task_struct *task);
17    void (*exit)(struct cgroup_subsys_state *css,
18               struct cgroup_subsys_state *old_css,
19               struct task_struct *task);
20    void (*bind)(struct cgroup_subsys_state *root_css);
21
22    int disabled;
23    int early_init;
24
25    /*
26     * If %false, this subsystem is properly hierarchical -
27     * configuration, resource accounting and restriction on a parent
28     * cgroup cover those of its children. If %true, hierarchy support
29     * is broken in some ways - some subsystems ignore hierarchy
```



```

30     * completely while others are only implemented half-way.
31     *
32     * It's now disallowed to create nested cgroups if the subsystem is
33     * broken and cgroup core will emit a warning message on such
34     * cases. Eventually, all subsystems will be made properly
35     * hierarchical and this will go away.
36     */
37     bool broken_hierarchy;
38     bool warned_broken_hierarchy;
39
40     /* the following two fields are initialized automatically during boot */
41     int id;
42     const char *name;
43
44     /* link to parent, protected by cgroup_lock() */
45     struct cgroup_root *root;
46
47     /* idr for css->id */
48     struct idr css_idr;
49
50     /*
51     * List of cftypes. Each entry is the first entry of an array
52     * terminated by zero length name.
53     */
54     struct list_head cfts;
55
56     /*
57     * Base cftypes which are automatically registered. The two can
58     * point to the same array.
59     */
60     struct cftype *dfl_cftypes; /* for the default hierarchy */
61     struct cftype *legacy_cftypes; /* for the legacy hierarchies */
62
63     /*
64     * A subsystem may depend on other subsystems. When such subsystem
65     * is enabled on a cgroup, the depended-upon subsystems are enabled
66     * together if available. Subsystems enabled due to dependency are
67     * not visible to userland until explicitly enabled. The following
68     * specifies the mask of subsystems that this one depends on.
69     */
70     unsigned int depends_on;
71 };

```

cgroup\_sb\_opts则在同一文件(cgroup.c)中

```

1 struct cgroup_sb_opts {
2     unsigned int subsys_mask;
3     unsigned int flags;
4     char *release_agent;
5     bool cpuset_clone_children;
6     char *name;
7     /* User explicitly requested empty subsystem */
8     bool none;
9 };

```

**boot\_cpu\_init()**

```
libertyeagle@ubuntu: ~/linux-4.1.50
File Edit View Search Terminal Help

init/main.c
451      *      Activate the first processor.
452      */
453
454      static void __init boot_cpu_init(void)
455      {
B+> 456          int cpu = smp_processor_id();
457          /* Mark the boot cpu "present", "online" etc for SMP and UP
458             set_cpu_online(cpu, true);
459             set_cpu_active(cpu, true);
460             set_cpu_present(cpu, true);
461             set_cpu_possible(cpu, true);
462      }
463

remote Thread 1 In: start_kernel                                L456 PC: 0xffffffff81f22ac1
Breakpoint 2 at 0xffffffff81f22ac1: file init/main.c, line 456.
(gdb) c
Continuing.

Breakpoint 2, start_kernel () at init/main.c:520
(gdb) step
boot_cpu_init () at init/main.c:456
(gdb) 
```

该函数激活第一个CPU，先取得CPU的ID，然后将该CPU标记为online, active, present, possible

```
1 static void __init boot_cpu_init(void)
2 {
3     int cpu = smp_processor_id();
4     /* Mark the boot cpu "present", "online" etc for SMP and UP case */
5     set_cpu_online(cpu, true);
6     set_cpu_active(cpu, true);
7     set_cpu_present(cpu, true);
8     set_cpu_possible(cpu, true);
9 }
```

`pr_notice("%s", linux_banner)`

```
libertyeagle@ubuntu: ~/linux-4.1.50
File Edit View Search Terminal Help

kernel/printk/printk.c
1850  asmlinkage __visible int printk(const char *fmt, ...)
1851  {
1852      printk_func_t vprintk_func;
1853      va_list args;
1854      int r;
1855
> 1856      va_start(args, fmt);
1857
1858      /*
1859       * If a caller overrides the per_cpu printk_func, then it n
1860       * to disable preemption when calling printk(). Otherwise
1861       * the printk_func should be set to the default. No need to
1862       * disable preemption here.

remote Thread 1 In: printk                                L1856 PC: 0xffffffff8189897a
Type "help" followed by command name for full documentation.
Type "apropos word" to search for commands related to "word".
Command name abbreviations are allowed if unambiguous.
(gdb) display linux_banner
2: linux_banner = 0xffffffff81a00080 <linux_banner> "Linux version 4.1.50 (liber
tyeagle@ubuntu) (gcc version 7.2.0 (Ubuntu 7.2.0-8ubuntu3.2) ) #1 SMP Mon Mar 26
08:03:29 PDT 2018\n"
(gdb) █
```

这一步会调用printk()打印linux\_banner

printk函数在kernel/printk/printk.c中

我们用gdb可以看出其内容是

```
Linux version 4.1.50 (libertyeagle@ubuntu) (gcc version 7.2.0 (Ubuntu 7.2.0-8ubuntu3.2) )
#1 SMP Mon Mar 26 08:03:29 PDT 2018\n
```

**setup\_arch(&command\_line)**

```
libertyeagle@ubuntu: ~/linux-4.1.50
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)
B+> 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      }

remote Thread 1 In: setup_arch                                L862  PC: 0xffffffff81f257e6
Breakpoint 7 at 0xffffffff81f25580: file arch/x86/kernel/time.c, line 95.
(gdb) c
Continuing.

Breakpoint 3, setup_arch (
  cmdline_p=0xffffffff81e03f78 <init_thread_union+16248>)
  at arch/x86/kernel/setup.c:862
(gdb) █
```

这一部分为体系结构的初始化函数，定义在arch/x86/kernel/setup.c中，接受command\_line为参数

## trap\_init

```
libertyeagle@ubuntu: ~/linux-4.1.50
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)
B+> 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;
962      }

remote Thread 1 In: trap_init                                L954  PC: 0xffffffff81f2502e
Breakpoint 2 at 0xffffffff81f2502e: file arch/x86/kernel/traps.c, line 954.
(gdb) break mm_init
Breakpoint 3 at 0xffffffff8104f450: mm_init. (2 locations)
(gdb) c
Continuing.

Breakpoint 2, trap_init () at arch/x86/kernel/traps.c:954
(gdb) █
```

这一部分定义在arch/x86/kernel.traps.c中，用来构建中断描述符号表  
可以观察出一些关键部分

```

1  set_intr_gate(X86_TRAP_DE, divide_error);
2  set_intr_gate(X86_TRAP_NMI, &nmi, NMI_STACK);
3  /* int4 can be called from all */
4  set_system_intr_gate(X86_TRAP_OF, &overflow);
5  set_intr_gate(X86_TRAP_BR, bounds);
6  set_intr_gate(X86_TRAP_UD, invalid_op);
7  set_intr_gate(X86_TRAP_NM, device_not_available);
8  set_intr_gate(X86_TRAP_DF, &double_fault, DOUBLEFAULT_STACK);
9  set_intr_gate(X86_TRAP_OLD_MF, coprocessor_segment_overrun);
10 set_intr_gate(X86_TRAP_TS, invalid_TSS);
11 set_intr_gate(X86_TRAP_NP, segment_not_present);
12 set_intr_gate(X86_TRAP_SS, stack_segment);
13 set_intr_gate(X86_TRAP_GP, general_protection);
14 set_intr_gate(X86_TRAP_SPURIOUS, spurious_interrupt_bug);
15 set_intr_gate(X86_TRAP_MF, coprocessor_error);
16 set_intr_gate(X86_TRAP_AC, alignment_check);

```

可以看出来这部分代码定义了各种exception，例如

- #DE -> 被0除
- #NMI -> 不可屏蔽中断 (Non-maskable Interrupt)
- #OF -> 溢出
- #BR -> 边界检查错误
- #UD -> 无效指令 (Invalid Opcode)
- #NM -> 设备不可用

## mm\_init()

```

libertyeagle@ubuntu: ~/linux-4.1.50
File Edit View Search Terminal Help
init/main.c
479      /*
480      * page_ext requires contiguous pages,
481      * bigger than MAX_ORDER unless SPARSEMEM.
482      */
483      page_ext init_flatmem();
B+> 484      mem_init();
485      kmem_cache_init();
486      percpu_init_late();
487      pgtable_init();
488      vmalloc_init();
489      ioremap_huge_init();
490  }
491
remote Thread 1 In: start_kernel                                L484  PC: 0xfffffffff81f22c83
Continuing.

Breakpoint 3, start_kernel () at init/main.c:554
(gdb) break sched_init
Breakpoint 4 at 0xfffffffff81f3c20e: file kernel/sched/core.c, line 7145.
(gdb) step
mm_init () at init/main.c:484
(gdb)

```

该函数就在init/main.c中，内容如下

```
1 static void __init mm_init(void)
```

```

2 {
3     /*
4     * page_ext requires contiguous pages,
5     * bigger than MAX_ORDER unless SPARSEMEM.
6     */
7     page_ext_init_flatmem();
8     mem_initp();
9     kmem_cache_init();
10    percpu_init_late();
11    pgtable_init();
12    vmalloc_init();
13    ioremap_huge_init();
14 }

```

其中page\_ext\_init\_flatmem与CONFIG\_SPARSEMEM有关，mem\_init释放所有bootmem，kmem\_cahce\_init初始化内核缓存，vmalloc\_init初始化vmalloc

## sched\_init()

```

libertyeagle@ubuntu: ~/linux-4.1.50
File Edit View Search Terminal Help
kernel/sched/core.c
7140     #endif
7141
7142     DECLARE_PER_CPU(cpumask_var_t, load_balance_mask);
7143
7144     void __init sched_init(void)
7145     {
7146         int i, j;
7147         unsigned long alloc_size = 0, ptr;
7148
7149         #ifdef CONFIG_FAIR_GROUP_SCHED
7150             alloc_size += 2 * nr_cpu_ids * sizeof(void **);
7151         #endif
7152         #ifdef CONFIG_RT_GROUP_SCHED

```

```

remote Thread 1 In: sched_init L7145 PC: 0xffffffff81f3c20e
Breakpoint 4 at 0xffffffff81f3c20e: file kernel/sched/core.c, line 7145.
(gdb) step
mm_init () at init/main.c:484
(gdb) c
Continuing.

Breakpoint 4, sched_init () at kernel/sched/core.c:7145
(gdb)

```

定义在kernel/sched/core.c中，主要目的是

- 对相关数据结构分配内存
- 初始化root\_task\_group
- 初始化每个CPU的rq队列(包括其中的cfs队列和实时进程队列)
- 将init\_task进程转变为idle进程

其中我们可以看到

```

1     for_each_possible_cpu(i) {
2         struct rq *rq;
3
4         rq = cpu_rq(i);

```

```

5      raw_spin_lock_init(&rq->lock);
6      rq->nr_running = 0;
7      rq->calc_load_active = 0;
8      rq->calc_load_update = jiffies + LOAD_FREQ;
9      init_cfs_rq(&rq->cfs);
10     init_rt_rq(&rq->rt);
11     init_dl_rq(&rq->dl);

```

这部分代码遍历设置每一个处在possible状态的CPU，为其中的每一个CPU初始化一个runqueue队列

## time\_init()

```

libertyeagle@ubuntu: ~/linux-4.1.50
File Edit View Search Terminal Help
arch/x86/kernel/time.c
90     /*
91      * Initialize TSC and delay the periodic timer init to
92      * late x86_late_time_init() so ioremap works.
93      */
94     void __init time_init(void)
95     {
96         late_time_init = x86_late_time_init;
97     }
98
99
100
101
102
remote Thread 1 In: time_init          L95   PC: 0xfffffffff81f2558c
Continuing.

Breakpoint 2, time_init () at arch/x86/kernel/time.c:95
(gdb) step
(gdb) step
(gdb) print x86_late_time_init
$1 = {void (void)} 0xfffffffff81f25535 <x86_late_time_init>
(gdb)

```

用x86\_late\_time\_init初始化event timer，该代码在arch/x86/kernel/time.c中

## console\_init()



```
libertyeagle@ubuntu: ~/linux-4.1.50
File Edit View Search Terminal Help

drivers/tty/tty_io.c
3566      * we can't necessarily depend on lots of kernel help here.
3567      * Just do some early initializations, and do the complex setup
3568      * later.
3569      */
3570      void __init console_init(void)
B+> 3571      {
3572          initcall_t *call;
3573
3574          /* Setup the default TTY line discipline. */
3575          tty_ldisc_begin();
3576
3577          /*
3578           * set up the console device so that later boot sequences c
remote Thread 1 In: console_init                                L3571 PC: 0xffffffff81f5d619
Continuing.

Breakpoint 7, time_init () at arch/x86/kernel/time.c:95
(gdb) c
Continuing.

Breakpoint 2, console_init () at drivers/tty/tty_io.c:3571
(gdb) 
```

在drivers/tty/tty\_io.c中，用于初始化控制台，在这里仅完成一些early initializations

## rest\_init()

```
libertyeagle@ubuntu: ~/linux-4.1.50
File Edit View Search Terminal Help

init/main.c
382
383      static ninline void __init_refok rest_init(void)
B+> 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
392           * we schedule it before we create kthreadd, will OOPS.
393           */
394          kernel_thread(kernel_init, NULL, CLONE_FS);
remote Thread 1 In: rest_init                                L384 PC: 0xffffffff81896000
do_raw_spin_unlock (lock=<optimized out>) at kernel/locking/spinlock.c:191
(gdb) break rest_init
Breakpoint 6 at 0xffffffff81896000: file init/main.c, line 384.
(gdb) c
Continuing.

Breakpoint 6, rest_init () at init/main.c:384
(gdb) 
```

这是start\_kernel()最后调用的函数，进一步完成内核的初始化

```
1 static ninline void __init_refok rest_init(void)
```



```

2 {
3     int pid;
4
5     rcu_scheduler_starting();
6     smpboot_thread_init();
7     /*
8      * We need to spawn init first so that it obtains pid 1, however
9      * the init task will end up wanting to create kthreads, which, if
10     * we schedule it before we create kthreadd, will OOPS.
11     */
12     kernel_thread(kernel_init, NULL, CLONE_FS);
13     numa_default_policy();
14     pid = kernel_thread(kthreadd, NULL, CLONE_FS | CLONE_FILES);
15     rcu_read_lock();
16     kthreadd_task = find_task_by_pid_ns(pid, &init_pid_ns);
17     rcu_read_unlock();
18     complete(&kthreadd_done);
19
20     /*
21     * The boot idle thread must execute schedule()
22     * at least once to get things moving:
23     */
24     init_idle_bootup_task(current);
25     schedule_preempt_disabled();
26     /* Call into cpu_idle with preempt disabled */
27     cpu_startup_entry(CPUHP_ONLINE);
28 }

```

例如，rest\_init()首先会完成RCU调度器的启动

```

libertyeagle@ubuntu: ~/linux-4.1.50
File Edit View Search Terminal Help
kernel/rcu/tree.c
3915     * idle tasks are prohibited from containing RCU read-side critical
3916     * sections. This function also enables RCU lockdep checking.
3917     */
3918     void rcu_scheduler_starting(void)
3919     {
> 3920         WARN_ON(num online cpus() != 1);
3921         WARN_ON(nr_context_switches() > 0);
3922         rcu_scheduler_active = 1;
3923     }
3924
3925     /*
3926     * Compute the per-level fanout, either using the exact fanout spec
3927     * or balancing the tree, depending on CONFIG_RCU_FANOUT_EXACT.
remote Thread 1 In: rcu_scheduler_starting L3920 PC: 0xffffffff810a5d30
(gdb) c
Continuing.

Breakpoint 6, rest_init () at init/main.c:384
(gdb) step
(gdb) step
rcu_scheduler_starting () at kernel/rcu/tree.c:3920
(gdb) 

```

```

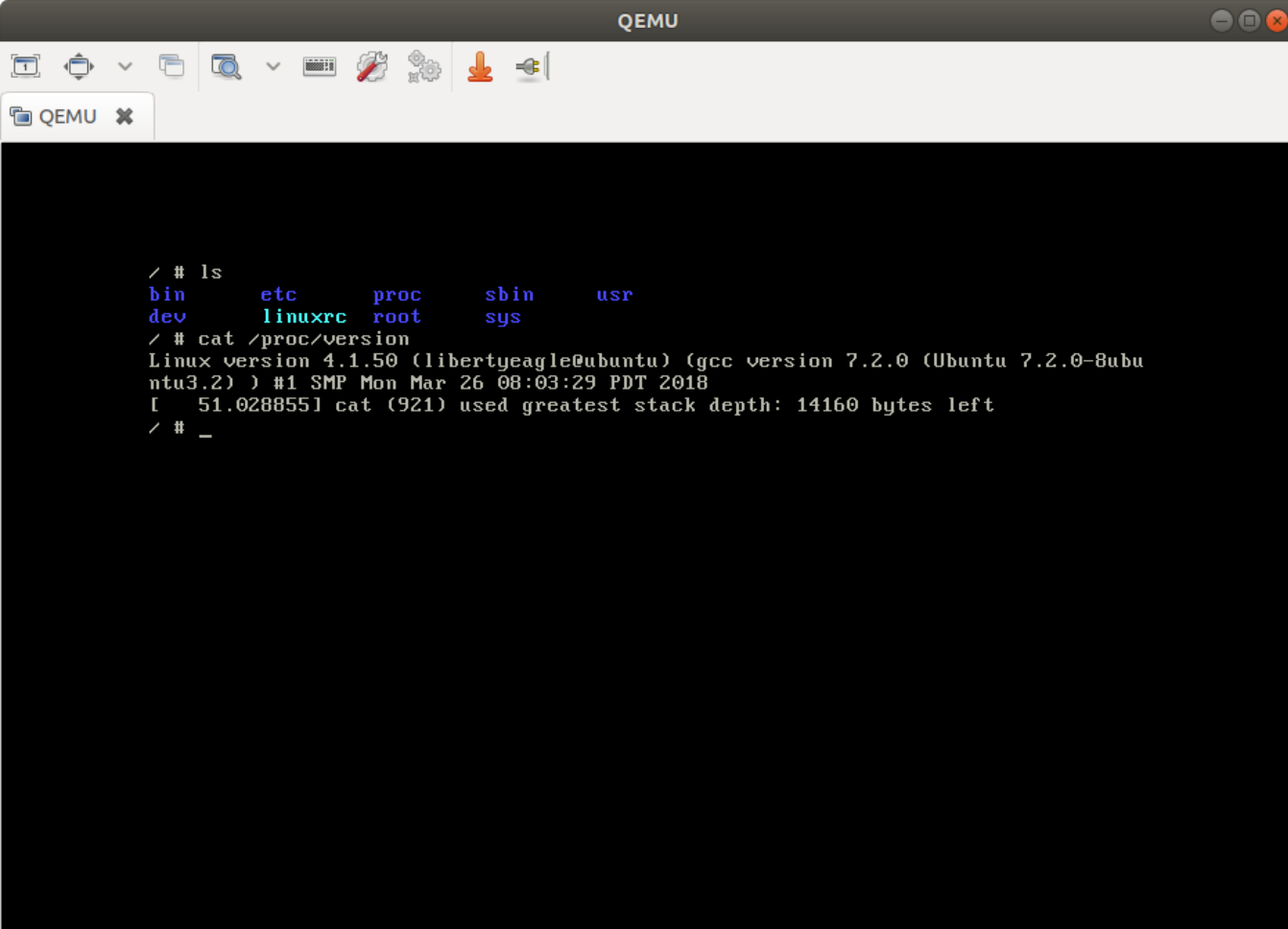
1 void rcu_scheduler_starting(void)
2 {

```

```
3     WARN_ON(num_online_cpus() != 1);
4     WARN_ON(nr_context_switches() > 0);
5     rcu_scheduler_active = 1;
6 }
```

首先会确保当前只有一个CPU在线，且没有上下文切换，之后会将RCU使能

## 启动完成



```
QEMU

/ # ls
bin      etc      proc     sbin     usr
dev      linuxrc  root     sys

/ # cat /proc/version
Linux version 4.1.50 (libertyeagle@ubuntu) (gcc version 7.2.0 (Ubuntu 7.2.0-8ubuntu3.2)) #1 SMP Mon Mar 26 08:03:29 PDT 2018
[ 51.028855] cat (921) used greatest stack depth: 14160 bytes left
/ # _
```