OSH_LAB_01

赵瑞 PB16120853

环境：Ubuntu 16.04，QEMU 2.12.0-rc1，kernel Linux-4.15.14，busybox-1.28.1，GDB 8.1

调试跟踪工具安装

1. 下载kernel、busybox、QEMU

   都是下载源码编译的。

   > 这三个的版本的匹配真的很重要，刚开始我装QEMU是直接`apt install qemu`，装的QEMU版本比较旧，于是后面在GDB调试的时候就出现了问题

2. 安装QEMU，先解压，然后如下：

   ```
   $ cd qemu-2.12.0-rc1/
   $ sudo make clean
   $ ./configure
   $ sudo make
   $ sudo make install
   ```

   为了后面方便，把QEMU加入path

   ```
   ln -s /usr/bin/qemu-system-x86_64 /usr/bin/qemu
   ```

3. 安装busybox，也是先解压，然后如下安装

   > 注意！：在第二行命令时，要修改一些东西
   >
   > 因为Linux运行环境当中是不带动态库的，所以必须以静态方式来编译Busybox。修改
   >
   > **Busybox Settings ---> Build Options ---> [*] Build Busybox as a static binary(no shared libs)**

   ```
   $ sudo make defconfig
   $ sudo make menuconfig
   $ sudo make
   $ sudo make install
   ```

4. 编译内核（注意：在`make menuconfig`时候要把debug的东西选上。

```
~$ cd linux/
~/linux$ ls
build-initrd.sh  busybox-1.28.2.tar.bz2  linux-4.15.14.tar.gz
busybox-1.28.2   linux-4.15.14
$ cd linux-4.15.14/
$ sudo su
# make clean
# make menuconfig
# make -j10
```

5. 准备根文件系统
   1. 我在 `/home/username/` 下 建立了 `myfile` 文件夹，在这个 文件夹下：

      ```
      dd if=/dev/zero of=busyboxinitrd4M.img bs=4096 count=1024
      mkfs.ext3 busyboxinitrd4M.img
      mkdir rootfs
      sudo mount -o loop busyboxinitrd4M.img rootfs/
      ```

   2. 回到Busybox目录：输入如下命令：

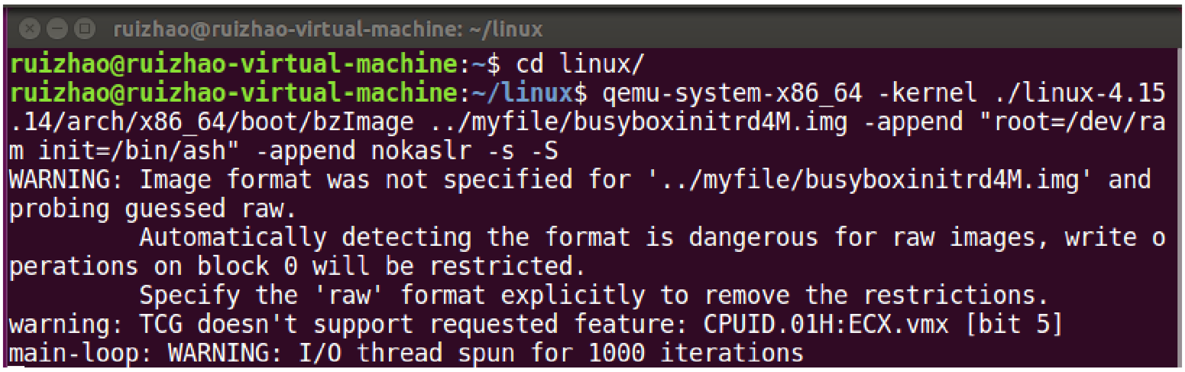
      ```
      make CONFIG_PREFIX= /home/ruizhao/myfile/rootfs/ install
      ```

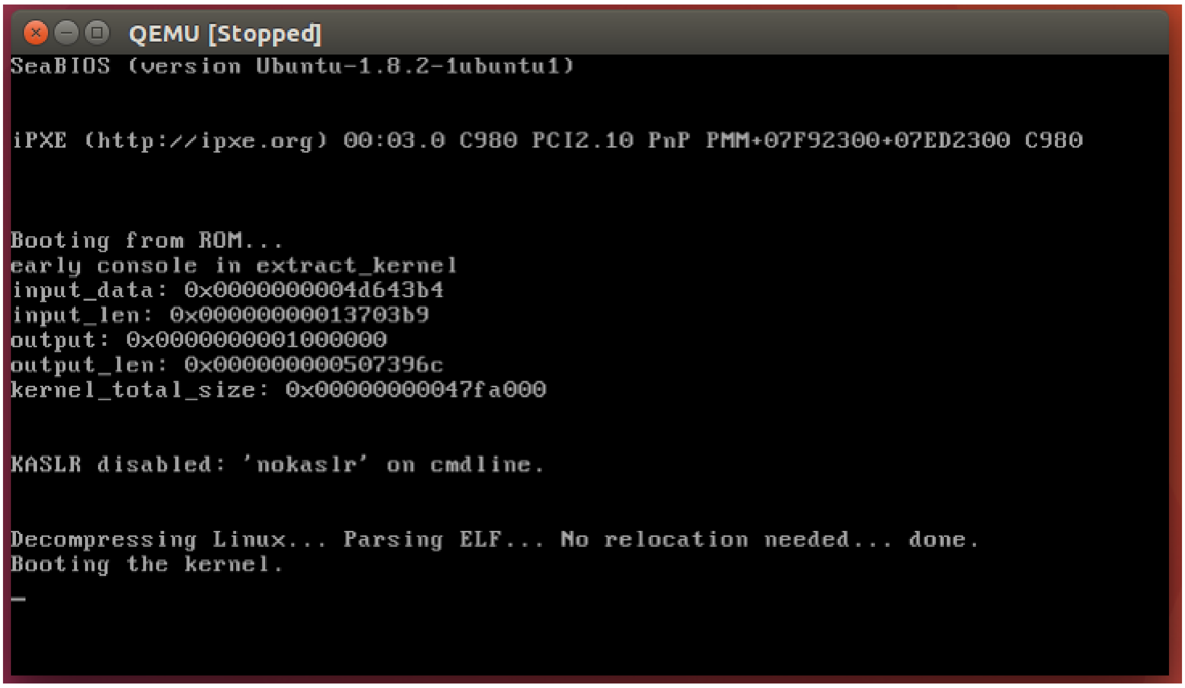
   3. 再次回到 `myfile` 目录下：

      ```
      umount rootfs
      ```

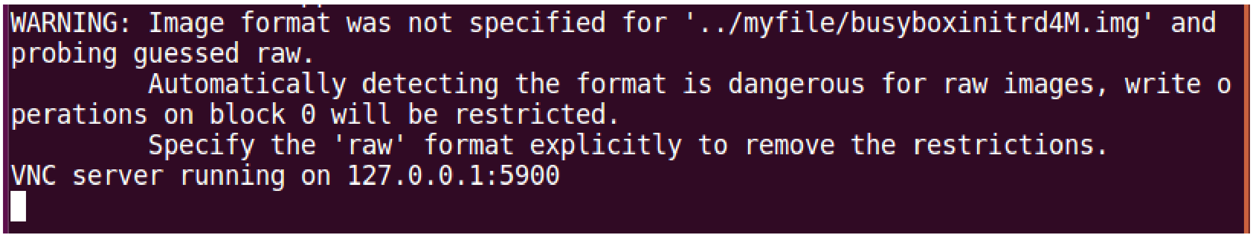
6. 开始调试

```
~/linux$ qemu-system-x86_64 -kernel ./linux-4.15.14/arch/x86_64/boot/bzImage
../myfile/busyboxinitrd4M.img -append "root=/dev/ram init=/bin/ash" -append nokaslr -s -S
```

> 下面是错误的历史：
>
> 第一次安装的QEMU版本旧的时候，就是这里出了问题，出现的界面如下
>
> 
>
>
>
> ```
> ruizhao@ruizhao-virtual-machine:~$ cd linux/
> ruizhao@ruizhao-virtual-machine:~/linux$ qemu-system-x86_64 -kernel ./linux-4.15
> .14/arch/x86_64/boot/bzImage ../myfile/busyboxinitrd4M.img -append "root=/dev/ra
> m init=/bin/ash" -append nokaslr -s -S
> WARNING: Image format was not specified for '../myfile/busyboxinitrd4M.img' and
> probing guessed raw.
>          Automatically detecting the format is dangerous for raw images, write o
> perations on block 0 will be restricted.
>          Specify the 'raw' format explicitly to remove the restrictions.
> warning: TCG doesn't support requested feature: CPUID.01H:ECX.vmx [bit 5]
> main-loop: WARNING: I/O thread spun for 1000 iterations
> ```
>
> 还出现了QEMU的界面，不过没有内容，此时一切正常，我再开一个terminal来GDB，

> 过程如下：
>
> ```
> ~$ cd linux/
> ~/linux$ gdb
> (gdb) file linux-4.15.14/vmlinux
> (gdb) target remote:1234
> (gdb) break start_kernel
> (gdb) c
> ```
>
> 此时QEMU就显示了新内容，停在了start_kernel的部分。
>
> 
>
>
>
> ```
> SeaBIOS (version Ubuntu-1.8.2-1ubuntu1)
>
>
> iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+07F92300+07ED2300 C980
>
>
>
> Booting from ROM...
> early console in extract_kernel
> input_data: 0x00000000004d643b4
> input_len: 0x000000000013703b9
> output: 0x0000000001000000
> output_len: 0x000000000507396c
> kernel_total_size: 0x00000000047fa000
>
>
> KASLR disabled: 'nokaslr' on cmdline.
>
>
> Decompressing Linux... Parsing ELF... No relocation needed... done.
> Booting the kernel.
> _
> ```
>
> 但是终端不可以用list来显示附近的代码，而且再次continue的话会显示该线程正在被使用，Google后有人说是GDB版本太久的bug，于是我更新了GDB，结果问题依然存在，后来同学也有这种问题，助教后来说是可以换Ubuntu17.10或者更新QEMU试试，我装了 Ubuntu17.10后又在前面的步骤遇到了新的问题，就回到Ubuntu16.04更新QEMU了，结果调试的问题小时了！看来真的是QUEM版本太旧的锅

结果：

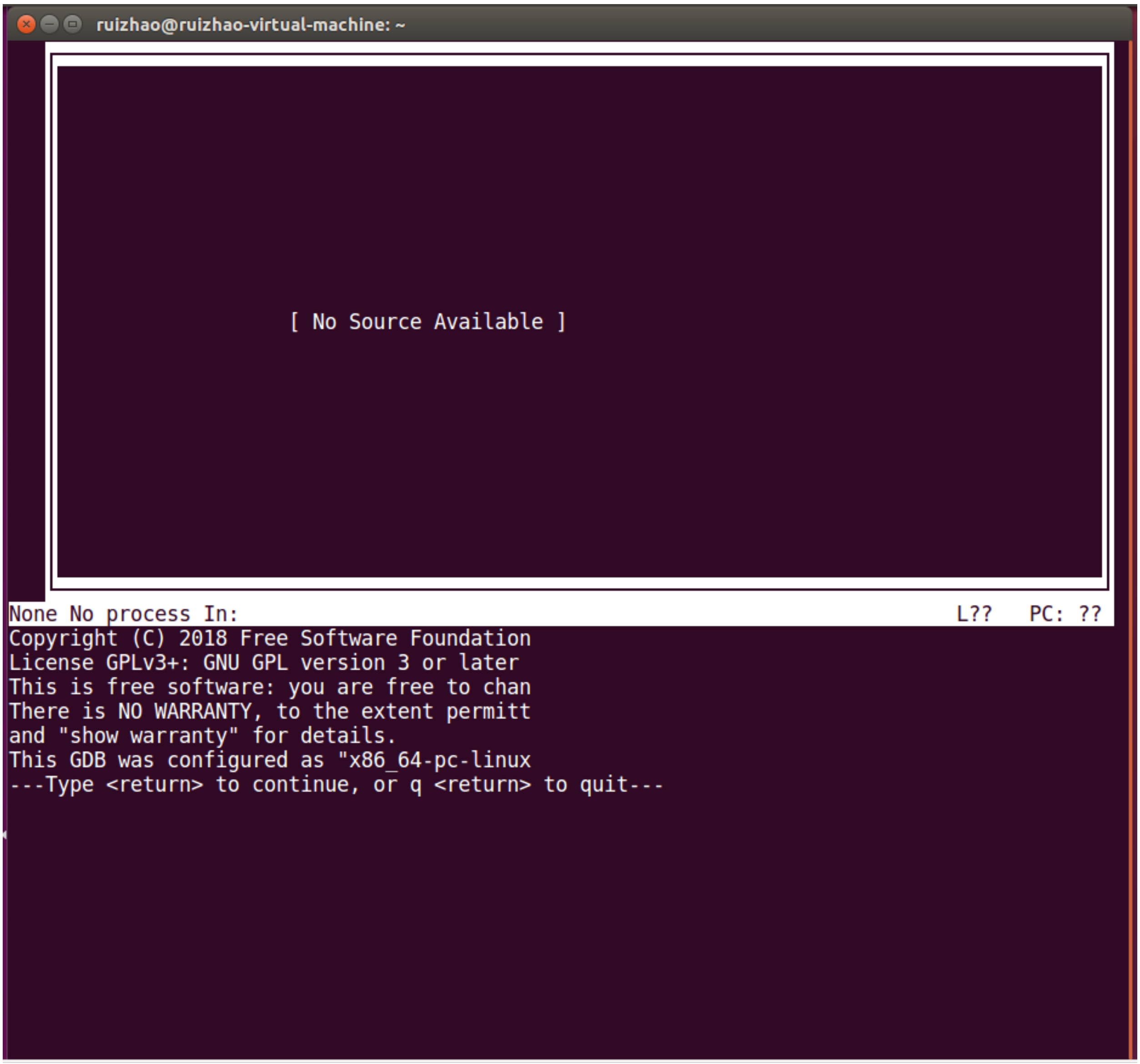


```
WARNING: Image format was not specified for '../myfile/busyboxinitrd4M.img' and
probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write o
perations on block 0 will be restricted.
         Specify the 'raw' format explicitly to remove the restrictions.
VNC server running on 127.0.0.1:5900
```

再打开一个terminal：使用GDB来调试：

```
$ gdb -tui
```

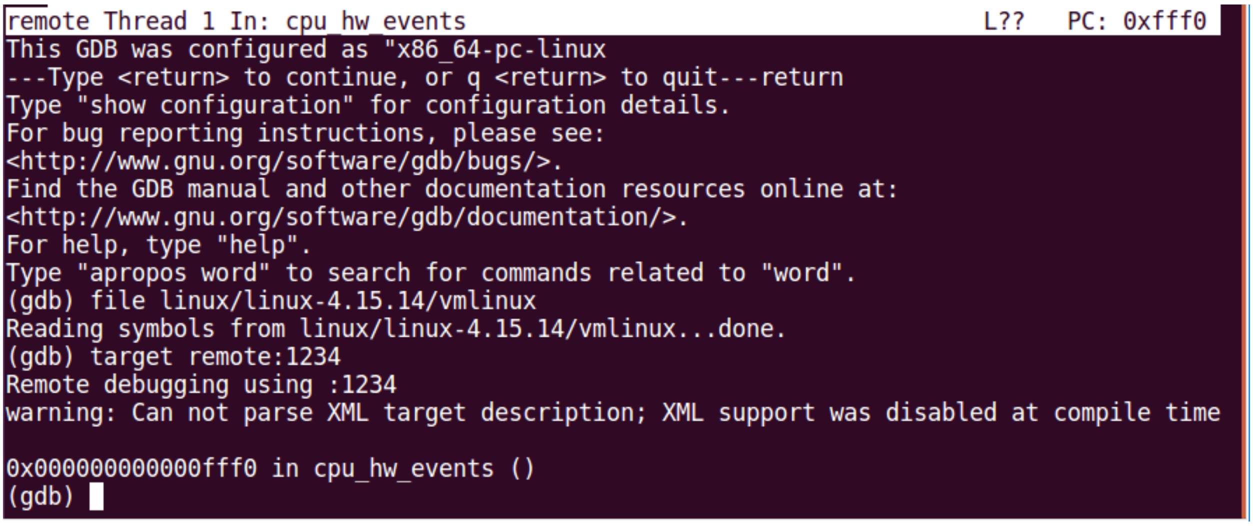
得到界面如图：



```
[ No Source Available ]

None No process In:                                                       L??   PC: ??
Copyright (C) 2018 Free Software Foundation
License GPLv3+: GNU GPL version 3 or later
This is free software: you are free to chan
There is NO WARRANTY, to the extent permitt
and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux
---Type <return> to continue, or q <return> to quit---
```

输入

```
return
(gdb) file linux/linux-4.15.14/vmlinux
(gdb) target remote:1234
```

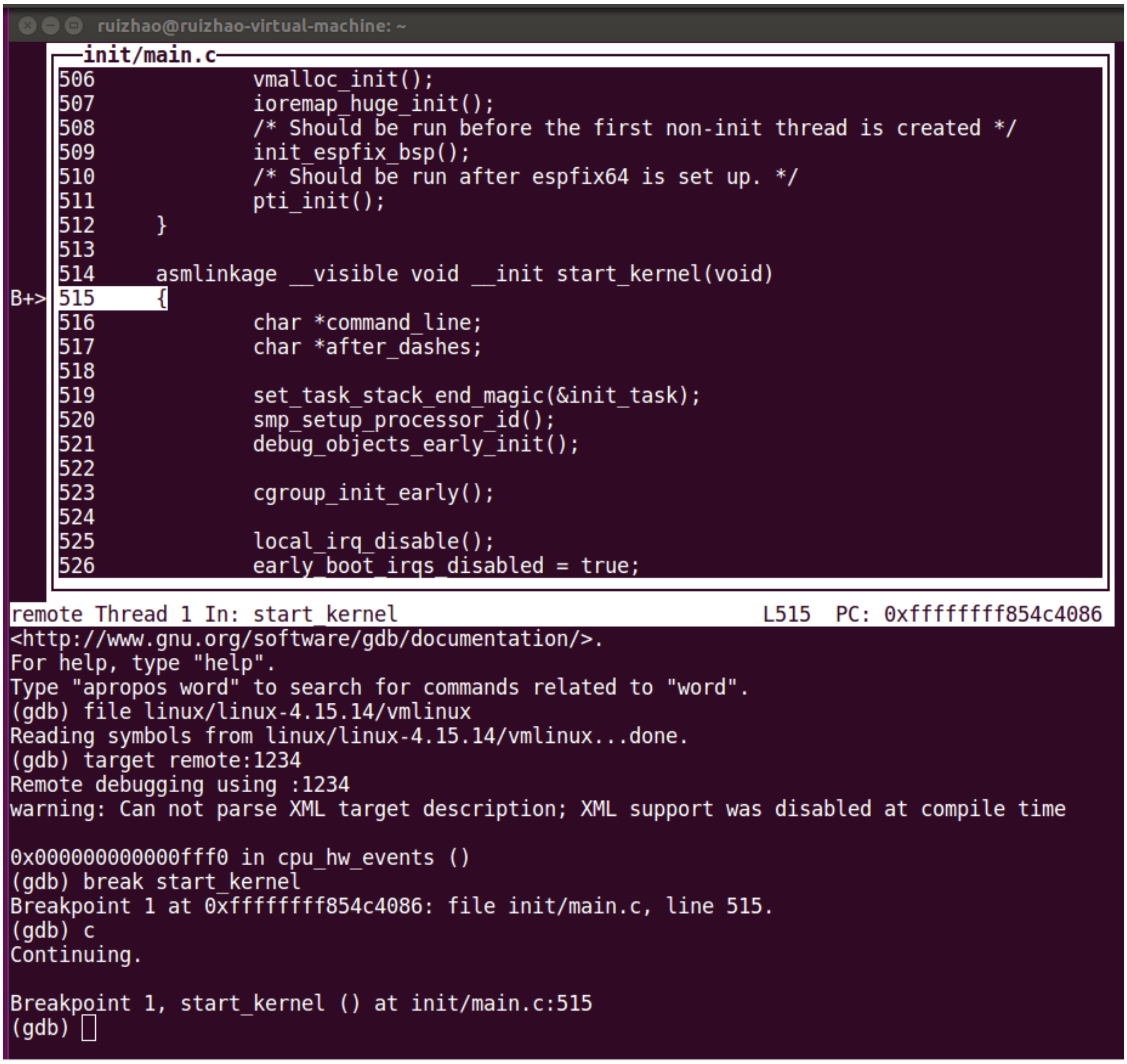


```
remote Thread 1 In: cpu_hw_events                                          L??   PC: 0xfff0
This GDB was configured as "x86_64-pc-linux
---Type <return> to continue, or q <return> to quit---return
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
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) file linux/linux-4.15.14/vmlinux
Reading symbols from linux/linux-4.15.14/vmlinux...done.
(gdb) target remote:1234
Remote debugging using :1234
warning: Can not parse XML target description; XML support was disabled at compile time

0x000000000000fff0 in cpu_hw_events ()
(gdb) 
```

关键事件

1. 设置断点：`start_kernel`，并开始运行

```
(gdb) break start_kernel
(gdb) c
```



```
    ┌─init/main.c──────────────────────────────────────────────────────────────────┐
    │506             vmalloc_init();                                               │
    │507             ioremap_huge_init();                                          │
    │508             /* Should be run before the first non-init thread is created */│
    │509             init_espfix_bsp();                                            │
    │510             /* Should be run after espfix64 is set up. */                 │
    │511             pti_init();                                                   │
    │512     }                                                                     │
    │513                                                                           │
    │514     asmlinkage __visible void __init start_kernel(void)                   │
B+> │515     {                                                                     │
    │516             char *command_line;                                           │
    │517             char *after_dashes;                                           │
    │518                                                                           │
    │519             set_task_stack_end_magic(&init_task);                         │
    │520             smp_setup_processor_id();                                     │
    │521             debug_objects_early_init();                                   │
    │522                                                                           │
    │523             cgroup_init_early();                                          │
    │524                                                                           │
    │525             local_irq_disable();                                          │
    │526             early_boot_irqs_disabled = true;                              │
    └──────────────────────────────────────────────────────────────────────────────┘
remote Thread 1 In: start_kernel                              L515   PC: 0xffffffff854c4086
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) file linux/linux-4.15.14/vmlinux
Reading symbols from linux/linux-4.15.14/vmlinux...done.
(gdb) target remote:1234
Remote debugging using :1234
warning: Can not parse XML target description; XML support was disabled at compile time

0x000000000000fff0 in cpu_hw_events ()
(gdb) break start_kernel
Breakpoint 1 at 0xffffffff854c4086: file init/main.c, line 515.
(gdb) c
Continuing.

Breakpoint 1, start_kernel () at init/main.c:515
(gdb) 
```

2. 在`start_kernel`的初始之初你可以看到这两个变量：

   ```
   char *command_line;
   char *after_dashes;
   ```

   第一个变量表示内核命令行的全局指针，第二个变量将包含`parse_args`函数通过输入字符串中的参数'name=value'，寻找特定的关键字和调用正确的处理程序。

3. 下一个函数是`set_task_stack_end_magic`，参数为`init_task`。`init_task`代表初始化进程(任务)数据结构:

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

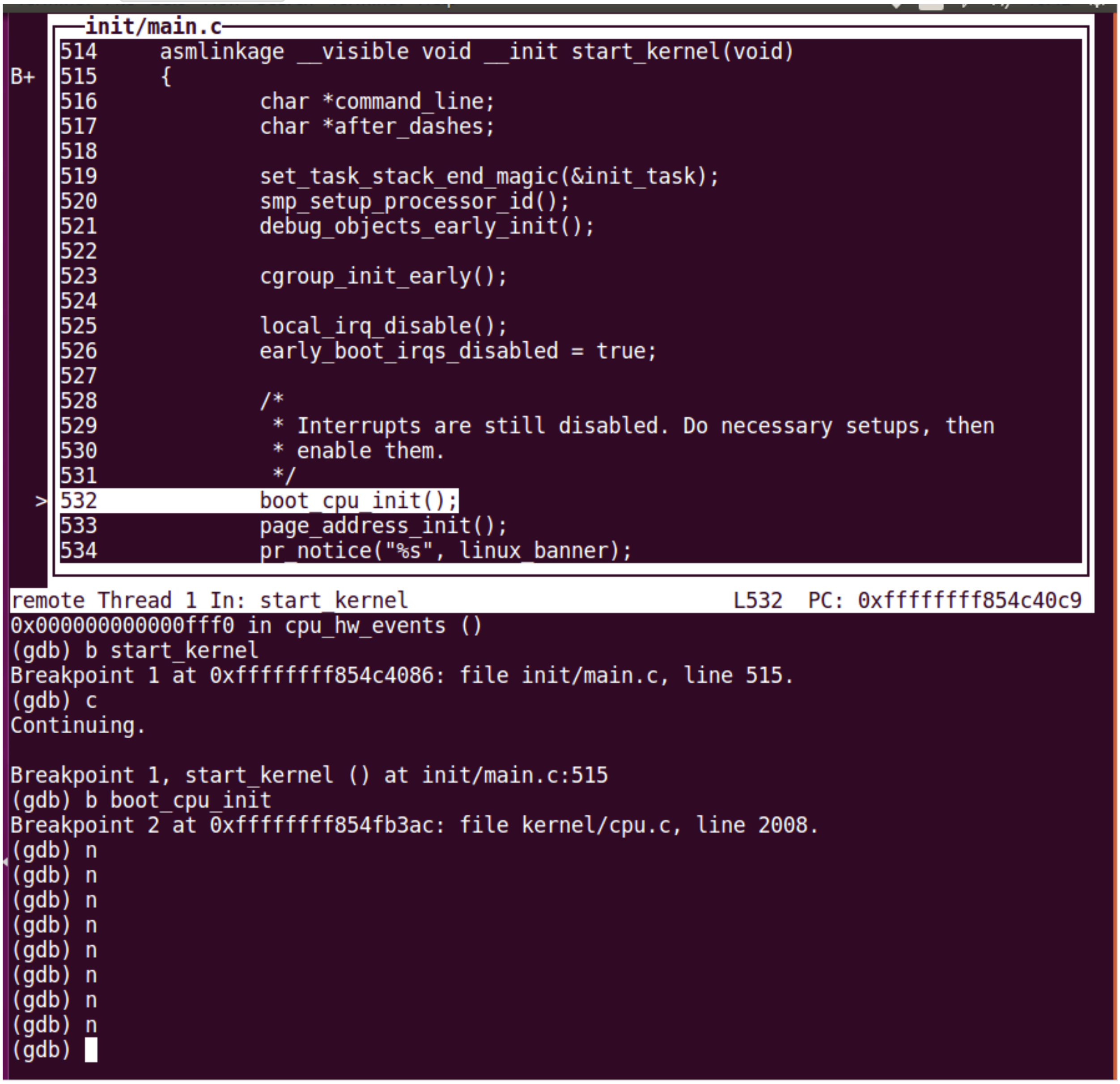
   `task_struct`存储了进程的所有相关信息。

4. `set_task_stack_end_magic`初始化完毕后的下一个函数是`smp_setup_processor_id`.此函数在`x86_64`架构上是空函数：

```
void __init __weak smp_setup_processor_id(void)
{
}
```

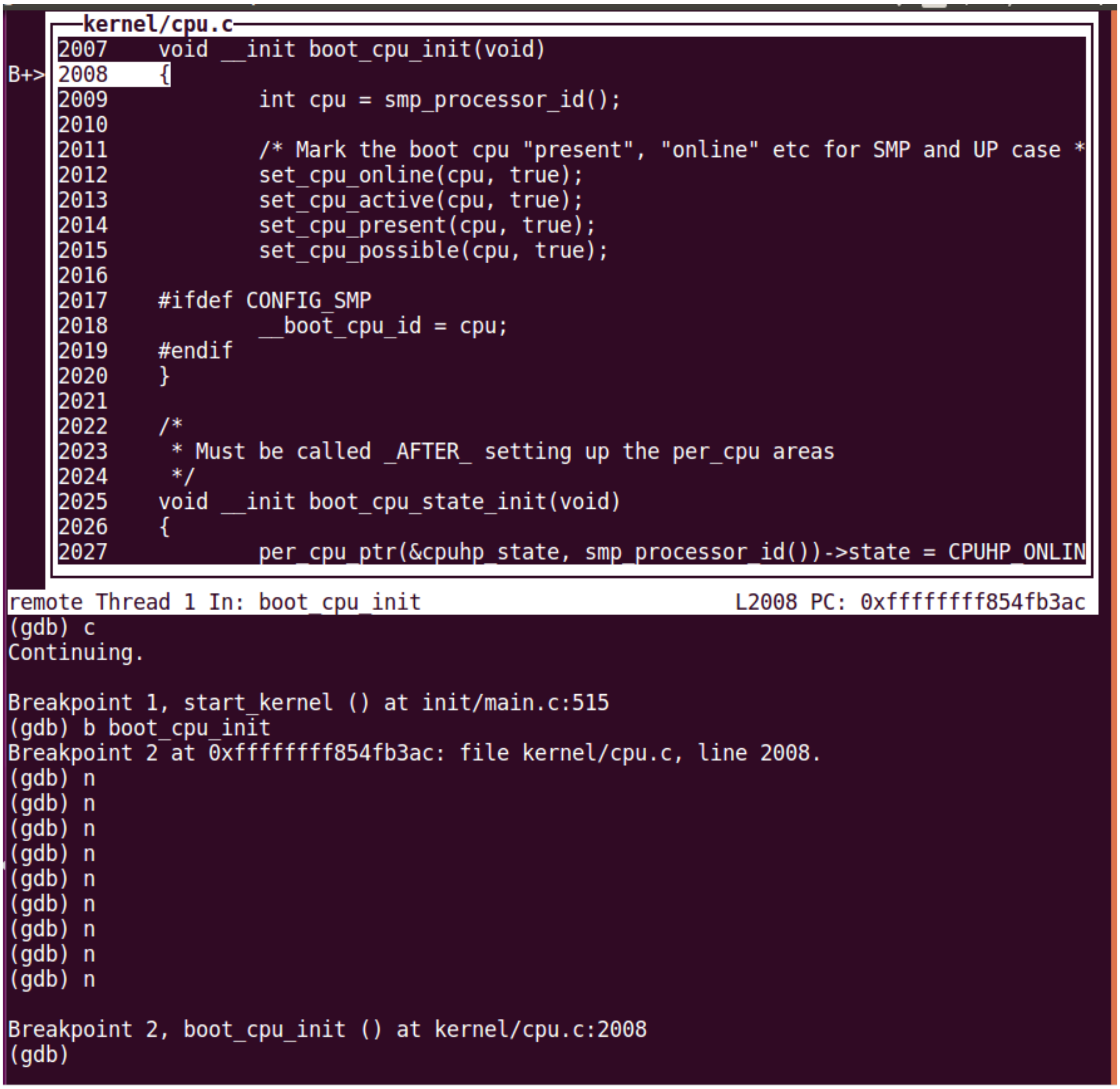
在此架构上没有实现此函数

5. 然后运行到`boot_cpu_init`，这里是激活第一个CPU事件



```
 init/main.c
   514     asmlinkage __visible void __init start_kernel(void)
B+ 515     {
   516             char *command_line;
   517             char *after_dashes;
   518
   519             set_task_stack_end_magic(&init_task);
   520             smp_setup_processor_id();
   521             debug_objects_early_init();
   522
   523             cgroup_init_early();
   524
   525             local_irq_disable();
   526             early_boot_irqs_disabled = true;
   527
   528             /*
   529              * Interrupts are still disabled. Do necessary setups, then
   530              * enable them.
   531              */
  >532             boot_cpu_init();
   533             page_address_init();
   534             pr_notice("%s", linux_banner);

remote Thread 1 In: start_kernel                      L532   PC: 0xffffffff854c40c9
0x000000000000fff0 in cpu_hw_events ()
(gdb) b start_kernel
Breakpoint 1 at 0xffffffff854c4086: file init/main.c, line 515.
(gdb) c
Continuing.

Breakpoint 1, start_kernel () at init/main.c:515
(gdb) b boot_cpu_init
Breakpoint 2 at 0xffffffff854fb3ac: file kernel/cpu.c, line 2008.
(gdb) n
(gdb) n
(gdb) n
(gdb) n
(gdb) n
(gdb) n
(gdb) n
(gdb) n
(gdb)
```

进入这个函数观察：



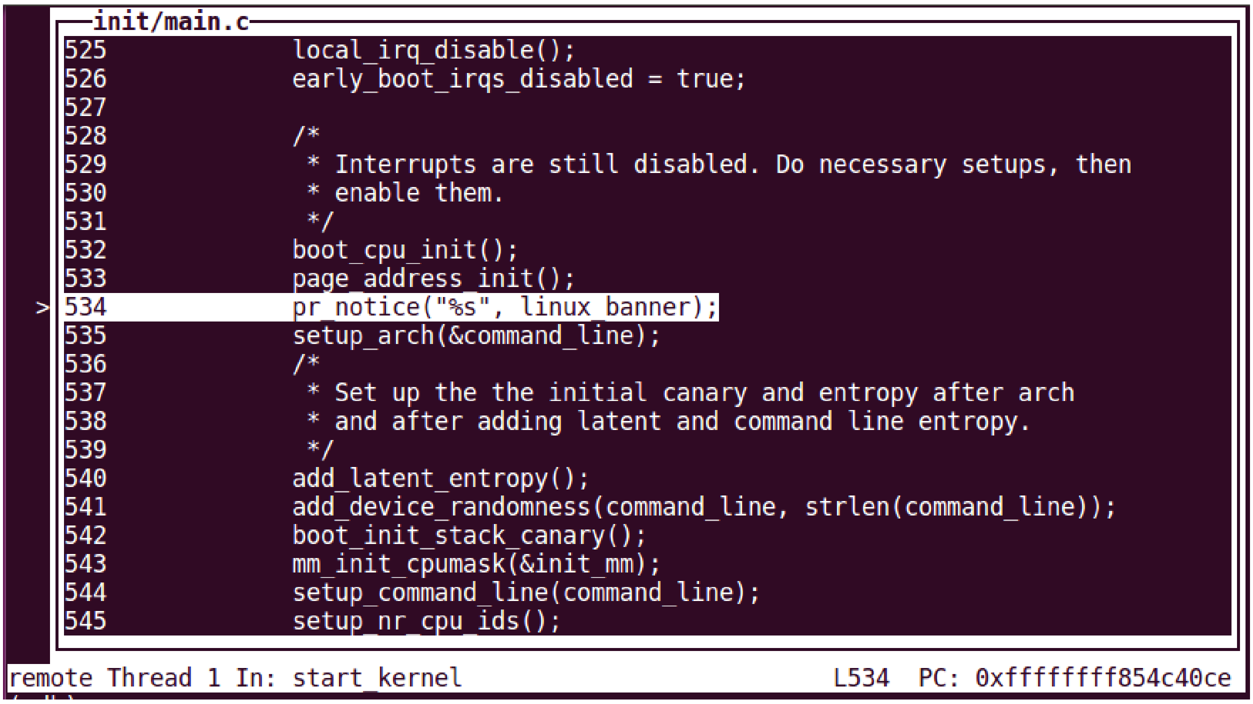
```
   ┌─kernel/cpu.c─────────────────────────────────────────────────────────────────┐
   │2007    void __init boot_cpu_init(void)                                        │
B+>│2008    {                                                                      │
   │2009            int cpu = smp_processor_id();                                  │
   │2010                                                                           │
   │2011            /* Mark the boot cpu "present", "online" etc for SMP and UP case *
   │2012            set_cpu_online(cpu, true);                                     │
   │2013            set_cpu_active(cpu, true);                                     │
   │2014            set_cpu_present(cpu, true);                                    │
   │2015            set_cpu_possible(cpu, true);                                   │
   │2016                                                                           │
   │2017    #ifdef CONFIG_SMP                                                      │
   │2018            __boot_cpu_id = cpu;                                           │
   │2019    #endif                                                                 │
   │2020    }                                                                      │
   │2021                                                                           │
   │2022    /*                                                                     │
   │2023     * Must be called _AFTER_ setting up the per_cpu areas                 │
   │2024     */                                                                    │
   │2025    void __init boot_cpu_state_init(void)                                  │
   │2026    {                                                                      │
   │2027            per_cpu_ptr(&cpuhp_state, smp_processor_id())->state = CPUHP_ONLIN
   └───────────────────────────────────────────────────────────────────────────────┘
remote Thread 1 In: boot_cpu_init                          L2008 PC: 0xffffffff854fb3ac
(gdb) c
Continuing.

Breakpoint 1, start_kernel () at init/main.c:515
(gdb) b boot_cpu_init
Breakpoint 2 at 0xffffffff854fb3ac: file kernel/cpu.c, line 2008.
(gdb) n
(gdb) n
(gdb) n
(gdb) n
(gdb) n
(gdb) n
(gdb) n
(gdb) n
(gdb) n

Breakpoint 2, boot_cpu_init () at kernel/cpu.c:2008
(gdb)
```

首先我们需要获取当前处理器的ID通过下面函数：

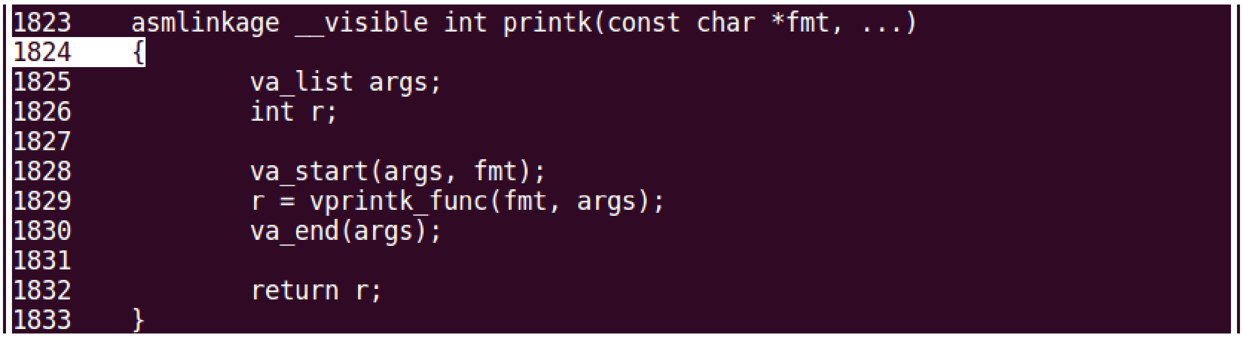
```
int cpu = smp_processor_id();
```

现在是0.

6. Linux 内核的第一条打印信息：



```
init/main.c
        local_irq_disable();
        early_boot_irqs_disabled = true;

        /*
         * Interrupts are still disabled. Do necessary setups, then
         * enable them.
         */
        boot_cpu_init();
        page_address_init();
        pr_notice("%s", linux_banner);
        setup_arch(&command_line);
        /*
         * Set up the the initial canary and entropy after arch
         * and after adding latent and command line entropy.
         */
        add_latent_entropy();
        add_device_randomness(command_line, strlen(command_line));
        boot_init_stack_canary();
        mm_init_cpumask(&init_mm);
        setup_command_line(command_line);
        setup_nr_cpu_ids();

remote Thread 1 In: start_kernel                     L534   PC: 0xffffffff854c40ce
```



```
asmlinkage __visible int printk(const char *fmt, ...)
{
        va_list args;
        int r;

        va_start(args, fmt);
        r = vprintk_func(fmt, args);
        va_end(args);

        return r;
}
```

调用了pr_notice函数。

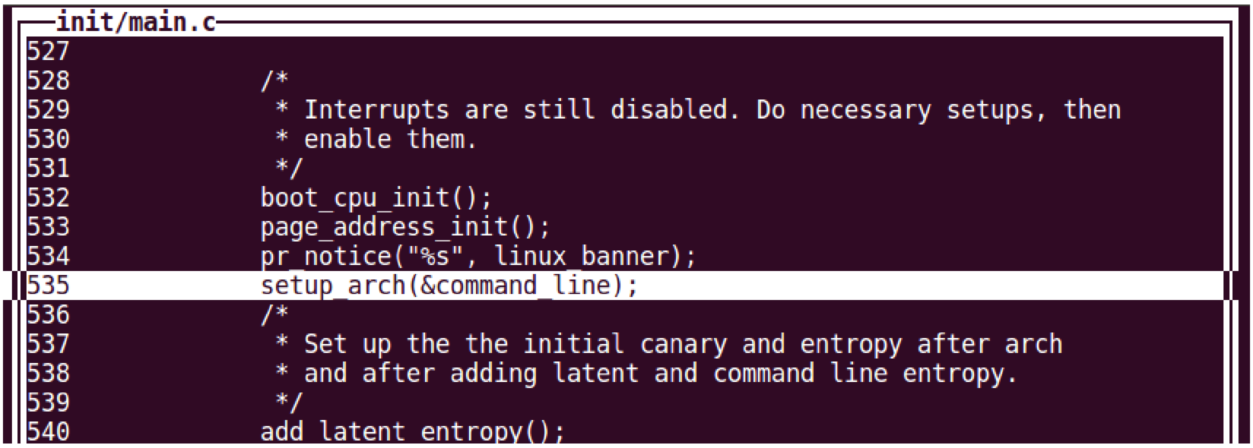
```
#define pr_notice(fmt, ...) \
    printk(KERN_NOTICE pr_fmt(fmt), ##__VA_ARGS__)
```

pr_notice其实是printk的扩展，这里我们使用它打印了Linux 的banner。

```
pr_notice("%s", linux_banner);
```

打印的是内核的版本号以及编译环境信息。

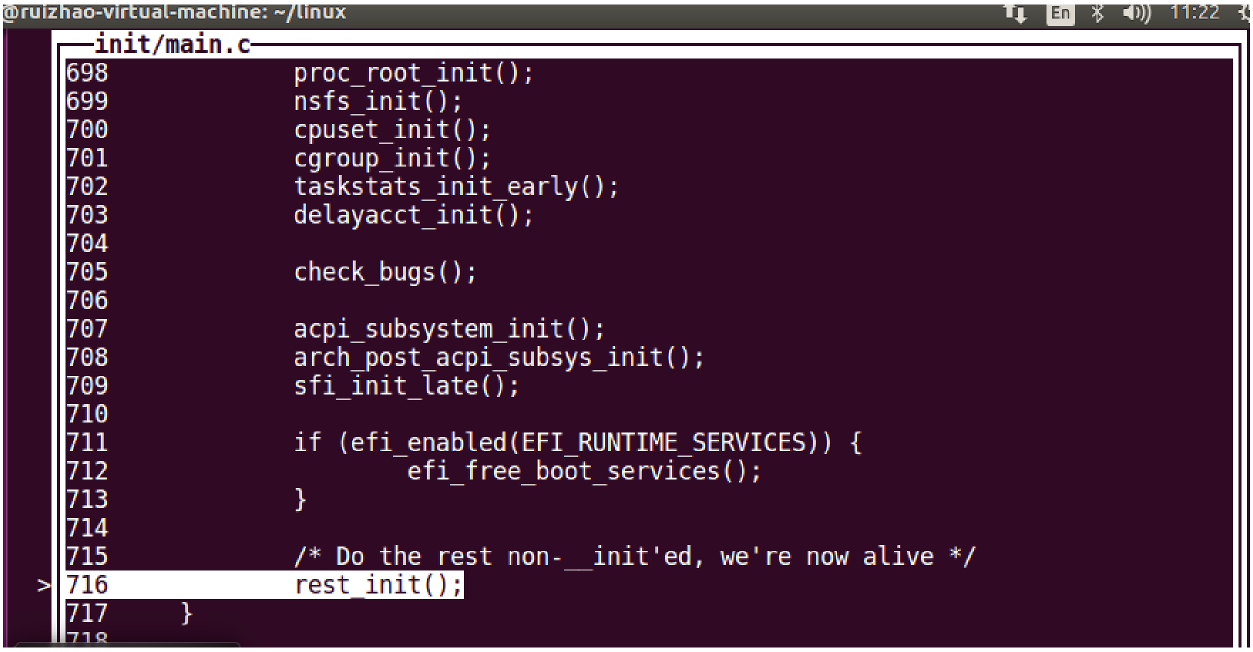
7. 依赖于体系结构的初始化部分



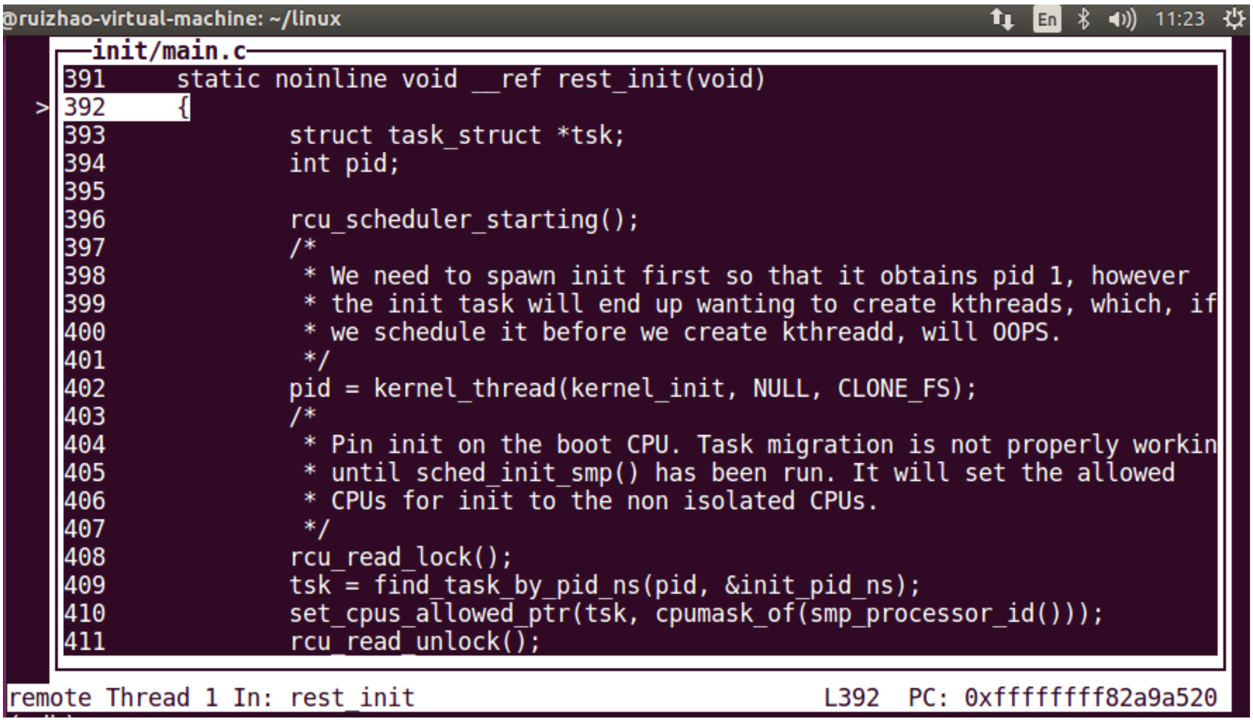
```

        /*
         * Interrupts are still disabled. Do necessary setups, then
         * enable them.
         */
        boot_cpu_init();
        page_address_init();
        pr_notice("%s", linux_banner);
        setup_arch(&command_line);
        /*
         * Set up the the initial canary and entropy after arch
         * and after adding latent and command line entropy.
         */
        add_latent_entropy();
```

8. `rest_init()`

   这是 `start_kernel` 的最后一个函数



```
        proc_root_init();
        nsfs_init();
        cpuset_init();
        cgroup_init();
        taskstats_init_early();
        delayacct_init();

        check_bugs();

        acpi_subsystem_init();
        arch_post_acpi_subsys_init();
        sfi_init_late();

        if (efi_enabled(EFI_RUNTIME_SERVICES)) {
                efi_free_boot_services();
        }

        /* Do the rest non-__init'ed, we're now alive */
        rest_init();
}
```



```
static noinline void __ref rest_init(void)
{
        struct task_struct *tsk;
        int pid;

        rcu_scheduler_starting();
        /*
         * We need to spawn init first so that it obtains pid 1, however
         * the init task will end up wanting to create kthreads, which, if
         * we schedule it before we create kthreadd, will OOPS.
         */
        pid = kernel_thread(kernel_init, NULL, CLONE_FS);
        /*
         * Pin init on the boot CPU. Task migration is not properly workin
         * until sched_init_smp() has been run. It will set the allowed
         * CPUs for init to the non isolated CPUs.
         */
        rcu_read_lock();
        tsk = find_task_by_pid_ns(pid, &init_pid_ns);
        set_cpus_allowed_ptr(tsk, cpumask_of(smp_processor_id()));
        rcu_read_unlock();
remote Thread 1 In: rest_init                         L392   PC: 0xffffffff82a9a520
```

调用 `rest_init()` 函数进行最后的初始化工作,包括创建1号进程 `(init)` ,第一个内核线程等操作。最后，初始化结束。

实验总结

这次实验过程很艰辛，刚开始QEMU的版本问题和编译内核时没有选debug info等导致了前期花费了大量时间来搭调试环境。通过这次实验，对Linux和Linux内核不再陌生了，刚开始做实验的时候都不知道从何看起，熟悉了Linux的一些命令，学习了一些GDB的调试命令，虽然很艰辛，但是感觉很有收获。