Lab 0: Compiling and Debugging the kernel

Ankit Goyal
ankit@cs.utexas.edu
CS380L

October 5, 2014

1 Setup

1.1 Hardware

Host Processor: 64 bit 8 core Intel(R) Xeon(R) CPU E3-1270 V2 @ 3.50GHz
Host Memory: 16GB
VM guests run with single 64-bit virtual CPU and 114.54MB of memory (obtained using /proc/meminfo)

1.2 Software

Host Operating System: Ubuntu with 3.13.0-34-generic 64 bit kernel.
Guest Operating System: Ubuntu with 3.16.1 64 bit kernel.
Qemu-kvm is used to run guest operating system.

2 Qemu command

```
qemu-system-x86_64 -drive file=tmpPKS3JS.qcow2 --snapshot \
        -net nic,model=virtio -net user -redir tcp:2222::22 \
        -kernel ~/workspace/kernel/kbuild/arch/x86_64/boot/bzImage \
        -boot c -append "root=/dev/sda1 console=ttyS0,115200n8 console=ttyS0" -s
```

Note: -s is added to launch it in debug mode.

3 dmesg output

A standard PC console screen being detected.

```
[    0.000000] Console: colour VGA+ 80x25
[    0.000000] console [ttyS0] enabled
```

Registering the USB interface.

```
[    0.590510] ACPI: bus type USB registered
[    0.591061] usbcore: registered new interface driver usbfs
[    0.591538] usbcore: registered new interface driver hub
[    0.592148] usbcore: registered new device driver usb
```

Detecting plug and play device interfaces . I don't believe these are actual devices, since I don't have Wacom Penabled attached.

Driver Id	Device Type
PNP0b00	CMOS/real time clock
PNP0303	Keyboard
PNP0f13	Alps Pointing-device
PNP0700	floppy disk controller.
PNP0400	printer
PNP0501	Wacom Penabled HID

```
[    0.698814] pnp: PnP ACPI init
[    0.699139] ACPI: bus type PNP registered
[    0.701140] pnp 00:00: Plug and Play ACPI device, IDs PNP0b00 (active)
[    0.701586] pnp 00:01: Plug and Play ACPI device, IDs PNP0303 (active)
[    0.701930] pnp 00:02: Plug and Play ACPI device, IDs PNP0f13 (active)
[    0.702334] pnp 00:03: [dma 2]
[    0.702549] pnp 00:03: Plug and Play ACPI device, IDs PNP0700 (active)
[    0.703010] pnp 00:04: Plug and Play ACPI device, IDs PNP0400 (active)
[    0.703427] pnp 00:05: Plug and Play ACPI device, IDs PNP0501 (active)
[    0.705823] pnp: PnP ACPI: found 6 devices
```

Keyboard and mouse actually getting detected

```
[    0.904011] uhci_hcd: USB Universal Host Controller Interface driver
[    0.905559] i8042: PNP: PS/2 Controller [PNP0303:KBD,PNP0f13:MOU] \
                      at 0x60,0x64 irq 1,12
```

Detecting ATA devices - Discovered two ATA devices.

```
[    0.885403] ata1: PATA max MWDMA2 cmd 0x1f0 ctl 0x3f6 bmdma 0xc020 irq 14
[    0.885740] ata2: PATA max MWDMA2 cmd 0x170 ctl 0x376 bmdma 0xc028 irq 15
```

ATA devices identified as DVD-ROM and HARDDISK

```
[    1.053424] ata2.00: ATAPI: QEMU DVD-ROM, 2.0.0, max UDMA/100
[    1.056661] ata1.00: ATA-7: QEMU HARDDISK, 2.0.0, max UDMA/100
[    1.058028] ata2.00: configured for MWDMA2
[    1.058844] ata1.00: configured for MWDMA2
```

A SCSI interface to ATA devices is emulated.

```
[    1.065295] scsi 0:0:0:0: Direct-Access  ATA  QEMU HARDDISK 0 PQ: 0 ANSI: 5
[    1.075428] scsi 1:0:0:0: CD-ROM QEMU  QEMU DVD-ROM  2.0. PQ: 0 ANSI: 5
[    1.087248] sr 1:0:0:0: Attached scsi CD-ROM sr0
```

Detecting RAID devices and no devices were found.

```
[    1.098346] md: Autodetecting RAID arrays.
[    1.098541] md: Scanned 0 and added 0 devices.
```

Network Device Detected

```
[    0.897400] virtio-pci 0000:00:03.0: irq 40 for MSI/MSI-X
[    0.897465] virtio-pci 0000:00:03.0: irq 41 for MSI/MSI-X
[    0.897504] virtio-pci 0000:00:03.0: irq 42 for MSI/MSI-X
```

4 Boot Time

Kernel start recording the time only after TSC is read:

```
...
...
[    0.000000] hpet clockevent registered
[    0.000000] tsc: Fast TSC calibration using PIT
[    0.000000] tsc: Detected 3491.968 MHz processor
[    0.012625] Calibrating delay loop (skipped), value calculated using timer frequency...
```

There's a difference of about 2-3 seconds. I recorded the wall time using stopwatch. There could be other delays due to buffering of the output to stdout.

5 Tracing the kernel

The program used to trace the kernel:

```
#include<unistd.h>
#include<fcntl.h>
int main()
{
         int fd = open("/dev/urandom", O_RDONLY);
        char data[4096];
        read(fd, &data, 4096);
        close(fd);
        fd = open("/dev/null", O_WRONLY);
        write(fd, &data, 4096);
        close(fd);
}
```

Given program is using open system call to open `dev/urandom` file. Internally open system call calls a method name `do_sys_open` as shown here `http://lxr.free-electrons.com/source/fs/open.c#L964`

So to stop the execution while the user program is running, the first breakpoint was set on `do_sys_open`. Process name could be used to verify that the breakpoint was hit during the user program

```
/* Program name */
print ((struct thread_info *)((long long)$rsp & ~16383))->task.comm // a.out(.*)

/* PID */
print ((struct thread_info *)((long long)$rsp & ~16383))->task.pid // 529
```

When the first breakpoint is hit, the second breakpoint is set as

```
break _raw_spin_lock if ((struct thread_info *)((long long)$rsp & ~16383))-> task.pid == 529
```

Now the breakpoint on `do_sys_open` can be disabled. On each breakpoint hit, the process name could be used to verify that the user program was running and is what cause the breakpoint condition.

6 Spinlock calls

6.1 Context 1: Opening or closing files

```
#0  _raw_spin_lock (lock=0xffff88000556d940) at
/home/goyal/workspace/kernel/linux-3.16.1/kernel/locking/spinlock.c:150
#1  0xffffffff811ed271 in spin_lock (lock=<optimized out>)
at /home/goyal/workspace/kernel/linux-3.16.1/include/linux/spinlock.h:303
#2  __alloc_fd (files=0xffff88000556d900, start=0, end=<optimized out>,
 flags=<optimized out>) at /home/goyal/workspace/kernel/linux-3.16.1/fs/file.c:452
#3  0xffffffff811ed3a0 in get_unused_fd_flags (flags=<optimized out>)
at /home/goyal/workspace/kernel/linux-3.16.1/fs/file.c:510
#4  0xffffffff811cf293 in do_sys_open (dfd=89577792, filename=<optimized out>,
flags=0, mode=<optimized out>) at /home/goyal/workspace/kernel/linux-3.16.1/fs/open.c:977
#5  0xffffffff8121f30b in C_SYSC_open (mode=<optimized out>, flags=<optimized out>,
filename=<optimized out>) at /home/goyal/workspace/kernel/linux-3.16.1/fs/compat.c:1093
#6  compat_SyS_open (filename=<optimized out>, flags=<optimized out>,
 mode=<optimized out>) at /home/goyal/workspace/kernel/linux-3.16.1/fs/compat.c:1091
#7  <signal handler called>
#8  0x00000000f771fbc4 in ?? ()
```

Kernel maintains an open file table structure (`files_struct`) for each task, pointed by tsk->files. Whenever a new file is opened, a file structure(`struct file`) is created and kernel allocates a file descriptor to that file.

File descriptor is used as an index to an array of file descriptor associated with the tsk, pointed by `tsk->files->fd` and the fd bit is set in the bitmap pointed to by `current->files->open_fds`. All above changes are done under the write protection of spinlock.

Similar lock is acquired while closing the file, to remove the above file descriptor from an array of open files.

```
#0  _raw_spin_lock (lock=0xffff880005480f40) at
/home/goyal/workspace/kernel/linux-3.16.1/kernel/locking/spinlock.c:150
#1  0xffffffff811ed462 in spin_lock (lock=<optimized out>) at
/home/goyal/workspace/kernel/linux-3.16.1/include/linux/spinlock.h:303
#2  __close_fd (files=0xffff880005480f00, fd=3) at
/home/goyal/workspace/kernel/linux-3.16.1/fs/file.c:578
#3  0xffffffff811cd9c3 in SYSC_close (fd=<optimized out>) at
/home/goyal/workspace/kernel/linux-3.16.1/fs/open.c:1055
#4  SyS_close (fd=<optimized out>) at
/home/goyal/workspace/kernel/linux-3.16.1/fs/open.c:1053
#5  <signal handler called>
```

6.2 Context 2: Updating the global variable jiffies

```
#0  _raw_spin_lock (lock=0xffffffff81c0c184 <jiffies_lock+4>)
at /home/goyal/workspace/kernel/linux-3.16.1/kernel/locking/spinlock.c:150
#1  0xffffffff810df554 in spin_lock (lock=<optimized out>)
 at /home/goyal/workspace/kernel/linux-3.16.1/include/linux/spinlock.h:303
#2  write_seqlock (sl=<optimized out>)
```

```
 at /home/goyal/workspace/kernel/linux-3.16.1/include/linux/seqlock.h:300
#3  tick_do_update_jiffies64 (now=...)
 at /home/goyal/workspace/kernel/linux-3.16.1/kernel/time/tick-sched.c:65
#4  0xffffffff810df625 in tick_sched_do_timer (now=...)
at /home/goyal/workspace/kernel/linux-3.16.1/kernel/time/tick-sched.c:131
#5  0xffffffff810df96a in tick_sched_timer (timer=0xffff880007c0eb00)
at /home/goyal/workspace/kernel/linux-3.16.1/kernel/time/tick-sched.c:1077
#6  0xffffffff81093097 in __run_hrtimer (timer=0xffffffff81c0c184 <jiffies_lock+4>,
now=0xffff880007c0e1b0) at /home/goyal/workspace/kernel/linux-3.16.1/kernel/hrtimer.c:1268
#7  0xffffffff8109347f in hrtimer_interrupt (dev=<optimized out>)
 at /home/goyal/workspace/kernel/linux-3.16.1/kernel/hrtimer.c:1357
#8  0xffffffff81046e47 in local_apic_timer_interrupt ()
at /home/goyal/workspace/kernel/linux-3.16.1/arch/x86/kernel/apic/apic.c:920
#9  0xffffffff81747adf in smp_apic_timer_interrupt (regs=<optimized out>)
 at /home/goyal/workspace/kernel/linux-3.16.1/arch/x86/kernel/apic/apic.c:944
```

Jiffies variable is used to hold the number of ticks that have occurred since the system booted. On every boot it is initialized to 0 and is increment on each timer interrupt. Traditionally jiffies has been a 32 bit counter which is prone to overflow. In the 64 bit kernels, a new variable jiffies64 is defined which is 64 bit. To maintain backward compatibility and for performance reason kernel developers decided to keep 32 bits jiffie variable and used the lower 32 bits of jiffies64 variable. To synchronize access to jiffy variable, spin_lock is used.

6.3 Context 3: Scheduler Tick to update process time

```
#0  _raw_spin_lock (lock=0xffff880007c143c0)
at /home/goyal/workspace/kernel/linux-3.16.1/kernel/locking/spinlock.c:150
#1  0xffffffff8109e532 in scheduler_tick ()
at /home/goyal/workspace/kernel/linux-3.16.1/kernel/sched/core.c:2501
#2  0xffffffff8107b070 in update_process_times (user_tick=0)
at /home/goyal/workspace/kernel/linux-3.16.1/kernel/timer.c:1389
#3  0xffffffff810df905 in tick_sched_handle (regs=0x3302ab0d7d, ts=<optimized out>,
 ts=<optimized out>) at /home/goyal/workspace/kernel/linux-3.16.1/kernel/time/tick-sched.c:151
#4  0xffffffff810df981 in tick_sched_timer (timer=0xffff880007c0eb00)
 at /home/goyal/workspace/kernel/linux-3.16.1/kernel/time/tick-sched.c:1084
```

update_process_times updates the value of process's time. The scheduler_tick() function called from update_process_times decrements the currently running process's timeslice and sets `need_resched` (needs rescheduling) if needed.

7 `/dev/random` vs `/dev/urandom`

The random number generator gathers noise from device drivers and other sources into an entropy pool. `/dev/random` generates randomness using noise from entropy pool and it blocks if the pool is empty. It is usually used when randomness is used to implement security.
`/dev/urandom` is a non-blocking alternative of `/dev/random`. It will reuse the noise in case the entropy pool is empty. The reuse can reduce the randomness of the generated numbers. `/dev/urandom` is good enough for almost all other purposes than security.

Time Spent on the lab 25 hours

8 References:

1. http://www.tldp.org/LDP/lki/lki-3.html
2. http://www.makelinux.net/books/lkd2/ch10lev1sec5
3. http://linux.die.net/man/4/rtc
4. http://linux.die.net/man/4/urandom