Kernel Programming + (a bit of luck) skills test

Djob Mvondo
3 weeks (28/07 - 18/08/2021)

July 27, 2021

The test runs for 3 weeks, i.e., from Wednesday 27 July - Wednesday 18 August. You will discuss your achievements in a video call with Djob in an arranged meeting between Thursday 19 August and Sunday the 22 August 2021. The goal is to assess your learning skills and perseverance on things that can be hard to achieve. Try to do your best and have fun. Courage !!!

You will need a Linux distribution (Ubuntu, CentOS, Fedora, Debian, etc.). You can choose the kernel of your choice (https://www.kernel.org/).

This test's main objective is to add a new clock source to your Linux kernel.

Context

A clock source is basically a hardware counter that keeps track of the time for the kernel and user space threads. Depending on which clock source you are relying on, you may lose in precision, accuracy or speed. However, maintaining clock sources is one of the most difficult things to do in a kernel. You have to ensure synchronization, idle times, CPU frequency changes etc. On a Linux distribution, you can view the available clock sources in the file */sys/devices/system/clocksource/clocksource0/available_clocksource* —- the *current_clocksource* file under the same directory gives you the one that's currently being used. OSes are modular enough today, to allow you to switch between different clock sources.

On multi-core systems, it is hard to keep track of the right clock.

Preliminary questions.

1. *Why is it hard to have a correct clock source on multi-core systems ?*
2. *What difference do you make between a MONOTONIC and REAL-TIME clock source ?*

Now, let's write our own clock source that will rely on the TSC (Time Stamp Counter). The idea is really simple (for me at least), each time a thread ask for the time, just get the reading

from the TSC clock and subtract the maximum sleep time of the thread.

Algorithm 1: Prototype of your clock source read time

Result: The time at a time t from your clocksource
task_process p = get_current_task();
tsc_time tsc_time = read_tsc();
return tsc_time - (p->max_sleep_time);

First step: Build your own Linux kernel.

Before hacking the Linux kernel, let's build our custom kernel. The following steps assume that you're running either Ubuntu or Debian.

- Firstly, download a Linux kernel archive (hereafter, I choose Linux 4.4.262).

```
wget https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.4.262.tar.xz
```

- Before building, let's install the necessary dependencies (you may need additional dependencies)

```
sudo apt-get install git fakeroot build-essential ncurses-dev
xz-utils libssl-dev bc flex libelf-dev bison
```

- Unzip the archive

```
sudo tar xfv linux-4.4.262.tar.xz
```

- At this step, you need to define the modules you want for your linux kernel (you can do that by entering in the linux kernel folder and typing `make menuconfig`), but we will just copy the existing config files on your machine

```
cd linux-4.4.262.tar.xz && cp /boot/config-$(uname -r) .config
```

.

- Now, we can build the kernel.

```
make
```

(You can do `make -j$(nproc)` to use as many threads as cores on your machine to speed the building process. But in case of build failure, it may be hard to find the issue).

- If successful, now install the kernel modules.

```
sudo make modules_install
```

- Then, we can install the output kernel (basically, it will generate the initrd and vmlinuz, and copies them to /boot.

```
sudo make install
```

- Update your grub to refresh the list of available kernels

```
sudo update-grub
```

(or `sudo update-grub2` — depends on your grub version.)

Now you can reboot your machine and choose your Linux distribution with your custom kernel version. You should see it in additional kernels for your distro when grub prints the list of available OSes.

... I want the kernel to have my name

.

Find a way to change your custom Linux kernel name (the one that gets printed under grub listings) to have your name instead, e.g., `Linux-djobv1 kernel`.

```
Tip: Inspect the Makefile.
```

... I shall be the time

To implement your clock source, you need to add a new *clock source* interface to the kernel. For that, you must pass the following functions:

- **.name**: the name of your clock source (being printed on available_clocksource)
- **.rating**: the tick frequency of your clock source
- **.read**: the function that reading the value of time
- .mask: the mask of your clock source (used by the kernel to know who is who)
- **.flags**: the flags of your clock source (used by the kernel to know what type of clock source you are)
- **.resume**: function that resumes your clock source counter

They are other arguments, but you won't use everything. For all other functions such as counter increment, initialization, frequency settings etc ..., don't give a shit, just return TSC values.

```
Tip1: Checkout TSC and HPET implementation details here:
HPET: https://elixir.bootlin.com/linux/v4.14.2/source/arch/x86/kernel/hpet.c#L854
TSC: https://elixir.bootlin.com/linux/v4.14.2/source/arch/x86/kernel/tsc.c#L1042
```

3. Build, and test if your new clock source does not crash the kernel.

... I'm better than you iiiiiiiiiiii

Let's evaluate your time sources against the existing ones. To do that, you will implement two simple benchmarks,

- The first one, just reads the value of the current clock source, sleep for 100ms, and repeat the process during x seconds (x should be configurable). Your benchmark should output the different clock source read, and the time needed to finish the benchmark. Plot the result for different clock sources (available on your device and yours) for x=10s (a separate graph from the execution time and for clock sources values).

  4. Analyze the results obtained

- The second one, should create n threads (n being your number of available cores), and each of the threads does the exact work as the first benchmark. Collect every output (different clock sources values and execution time) from each thread, and average difference between each thread values for every clock source and yours (a separate graph from the execution time and for clock sources values). Your plot here should have error bars to indicate the variability of the average.

  5. Analyze your results and comment on your clock synchronization accuracy

.

Useful links

- https://opensource.com/article/17/6/timekeeping-linux-vms
- http://events17.linuxfoundation.org/sites/events/files/slides/Timekeeping%20in%20the%20Linux%20Kernel_0.pdf
- https://www.kernel.org/doc/Documentation/timers/timekeeping.txt