Technical Disclosure Commons

Defensive Publications Series

March 20, 2019

Virtual device for driver testing

Rob Springer

Kyle Lucke

Andy Koch

Clinton Smullen

Virtual device for driver testing

ABSTRACT

A device driver is a component of an operating system that enables the OS to recognize a certain device and communicate with it. Testing a device driver normally requires either the physical device or a software-emulated version of the device. Physical and simulated devices are intended to work correctly, e.g., they do not generally behave in an unexpected manner. This means that during testing, it is difficult or impossible to verify if the driver is able to handle unexpected behavior of a device, e.g., device behavior that is not supposed to arise in common usage but nevertheless does.

Per the techniques of this disclosure, the driver-under-test is run within a virtual machine (VM). The driver communicates with a mock device controlled by programs running outside the VM. The mock device includes no actual operational logic; rather, it is scripted to simply react to driver actions, e.g., by triggering interrupts, reading/writing registers, etc. Outlier test cases of a driver, e.g., triggered by erroneous device behavior, can be tested by causing the mock device to behave in a scripted manner.

KEYWORDS

Device driver; kernel driver; test and measurement; virtual machine; device emulator; device simulator; mock device; virtual device

BACKGROUND

A device driver, also known as kernel driver or simply driver, is a component of an operating system that enables the OS to recognize a certain device and communicate with it. Testing a device driver normally requires either the physical device or a software-emulated version of the device. Physical and simulated devices are intended to work correctly, e.g., they do not generally behave in an unexpected manner. This means that during testing it is difficult or impossible to verify if the driver is able to handle unexpected behavior of the device, e.g., device behavior that is not supposed to arise in common usage but nevertheless does. This in turn implies that a released device driver may cause errors when unexpected device behaviors occur in practice.

DESCRIPTION

Per the techniques of this disclosure, the driver-under-test is run within a virtual machine (VM). The driver communicates with a mock device controlled by programs running outside the VM. The mock device includes no actual operational logic; rather it is scripted to simply react to driver actions, e.g., by triggering interrupts, reading/writing registers, etc. Outlier test cases of a driver, e.g., triggered by erroneous device behavior, can be tested by causing the mock device to behave in a scripted manner.

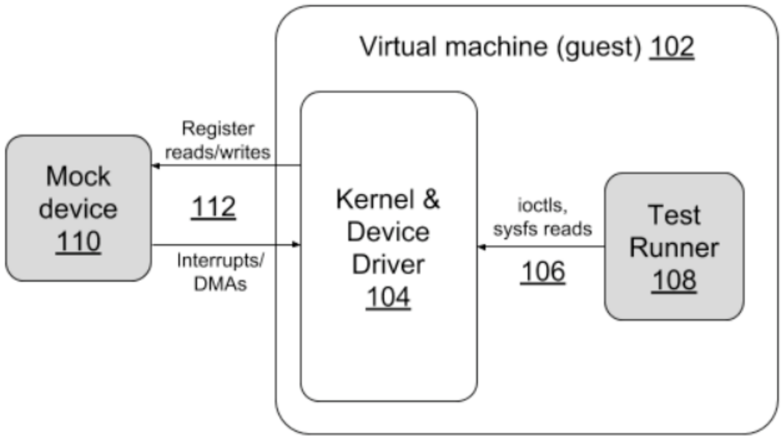

Fig. 1: Mock device and driver-under-test in communication with each other

Fig. 1 illustrates a device driver under test and a mock device in communication with each other. The device driver and OS kernel (104) run inside a virtual machine (102), which is itself a guest executing on a physical host machine. The device driver is controlled by a test runner (108) that communicates with the device driver using, e.g., ioctl calls, sysfs reads, (106) etc.

The test runner is provided by a test developer to drive device/driver behavior in some particular way. A mock (virtual) device (110) simulates an actual device for which the device driver is designed. The mock device is managed by the driver, but its behavior is controlled by a program running outside the VM. The mock device includes no real (operating) device logic; it only models the input-output responses of the actual device. The mock device is scripted by the test writer/developer such that it offers simple, targeted reactions to driver actions.

For example, the mock device may simply wait for a register to be written and then immediately sends an interrupt, without executing any of the complex logic that would be performed by an actual device between the events of writing a register and triggering an interrupt. The gray boxes of Fig. 1, e.g., test runner (108) and scripts that define the behavior of the mock device (110) are written by the test developer. The mock device runs on a host (a physical machine) outside the virtual machine that runs the driver-under-test.

For example, the mock device can be inside a program that implements services necessary for the virtual machine to simulate a device. These services (112) provide the device driver access to the mock device over virtualized PCI[e], the ability for the mock device to perform DMA (direct memory access) operations from the device driver, the ability for the mock device to send interrupts to the device driver, etc.

Functionalities provided by the services encapsulating the mock device fall into the following categories:

- write a register over PCI;
- read a register over PCI;

- send an interrupt to the device driver;
- read data from the device driver; etc.

These operations can be combined and scripted by a test writer/developer to create complex behaviors to test.

Example: To verify that the driver writes a value to the device after receiving an interrupt from the mock device, the test script for the mock device is written to do the following:

- wait for the driver to write a register that signals the start of a test script;
- send an interrupt to the driver;
- wait for the driver to write a response value to a register of the mock device; and
- verify that the written value is as expected.

In actual usage, a test starts the mock device on the host, specifying the behavior needed for that test. The test additionally starts the virtual machine, which boots, loads the mock device, and loads the device driver, at which point test-specific behavior drives the rest of execution.

The triggering of device operations off driver-register read/writes enables scripted, e.g., automated, testing of the driver and the driver-device combination. The devices are typically accelerators, coprocessors, etc. For most such devices, the driver constructs a test module; sends test module commands to the device; waits for the device to finish execution of, e.g., register reads/writes, DMA transfers, etc.; and expects an interrupt upon completion.

Scripts can be constructed by chaining together operations that are triggered in a series by register reads and writes, as illustrated by the following example:

Example: Two iterations of a simple command/response queue test are scripted as follows:

Step 1:

- wait for the driver to write the command ready register;

- read the written data from the register;
- perform the command, e.g., execute DMA transfer of data from the driver to the mock device;
- send the appropriate interrupt based on the specific command.

Step 2:

- wait for the driver to write the interrupt acknowledged register;
- reset interrupt state; prepare the device for another command by writing various registers.

Step 3: Repeat step 1.

Step 4: Repeat step 2.

Alternative to the use of mock devices, device drivers can also be tested by writing aggressive unit tests of the components of the driver. In this alternative, the fully integrated driver remains an untested surface. Also, the OS kernel (e.g., Linux kernel) does not lend itself easily unit testing, so the resulting code can be hard to write and maintain.

Another approach is to implement a physical PCI[e] device that can be programmed with arbitrary behavior, e.g., a special- or general-purpose RTL emulator. This approach rests on making available physical hardware (in the form of RTL) to test, which is generally more expensive, requires specialized knowledge to use, and has relatively low accessibility. In the case of an RTL emulator, the work required to specify a script is relatively large, which can however be ameliorated by writing extensive helper libraries.

In this manner, the techniques of this disclosure enable testing of outlier error cases of a kernel driver. Device drivers usually need devices against which to test, and few, if any, devices allow their behavior to be controlled in a way conducive to the types of testing described herein.

Requiring only a virtual device, the techniques enable a developer to run tests on their local system. The techniques apply generically to the development or hardening of devices and their drivers, and perform especially well for complex devices/drivers, e.g., video cards.

CONCLUSION

Per the techniques of this disclosure, a device driver under test is run within a virtual machine. The driver communicates with a mock device controlled by programs running outside the virtual machine. The mock device includes no actual operational logic; rather it is scripted to simply react to driver actions, e.g., by triggering interrupts, reading/writing registers, etc. Outlier test cases of a driver, e.g., triggered by erroneous device behavior, can be tested by causing the mock device to behave in a scripted manner.