Unit testing Linux kernel drivers

Laurent Carlier
Fosdem 2022
5th February 2022

Who am I?

- Embedded software engineer
- Working in the ML group at Arm
- Love playing with electronics
- Passionate about software engineering and testing

Problem with testing Linux kernel drivers

- Difficult to test assertions from the kernel space
- There is a need to write specific kernel test module + the corresponding user space application
- Error condition and edge cases difficult or impossible to test
- Using gdb to debug kernel module is not easy
- Difficulty to test in isolation

What I want to achieve

- Increase test coverage
- Be able to debug with gdb
- Use TDD when developing my kernel driver
- Do not want to rely on any dependencies
- Create test cases that reflects real world scenario

Advantages of unit testing

- Simple to write
- Small, fast to execute, quick feedback
- Easy to debug
- Allows TDD
- Real world scenario can be tested

Compiling kernel module as user space application

- Linux kernel header file aren't written to be included in user space application
- All the dependencies of your kernel module needs to be compiled and linked

It is very tricky to get something that compile and run

My solution: mock the Linux kernel headers with EasyMock

- EasyMock generates mocks of Linux headers
  - Contain the definition of all the #define macros
  - Contain the declaration of the types that Linux uses (e.g. struct device)
  - Contain a mocked implementation of all the functions the header file declares
- Use the mocked headers instead of the original header file
- Function under tests calls mocks instead of real implementation
  - Mocking breaks the dependencies

The kernel driver code becomes compilable in user space

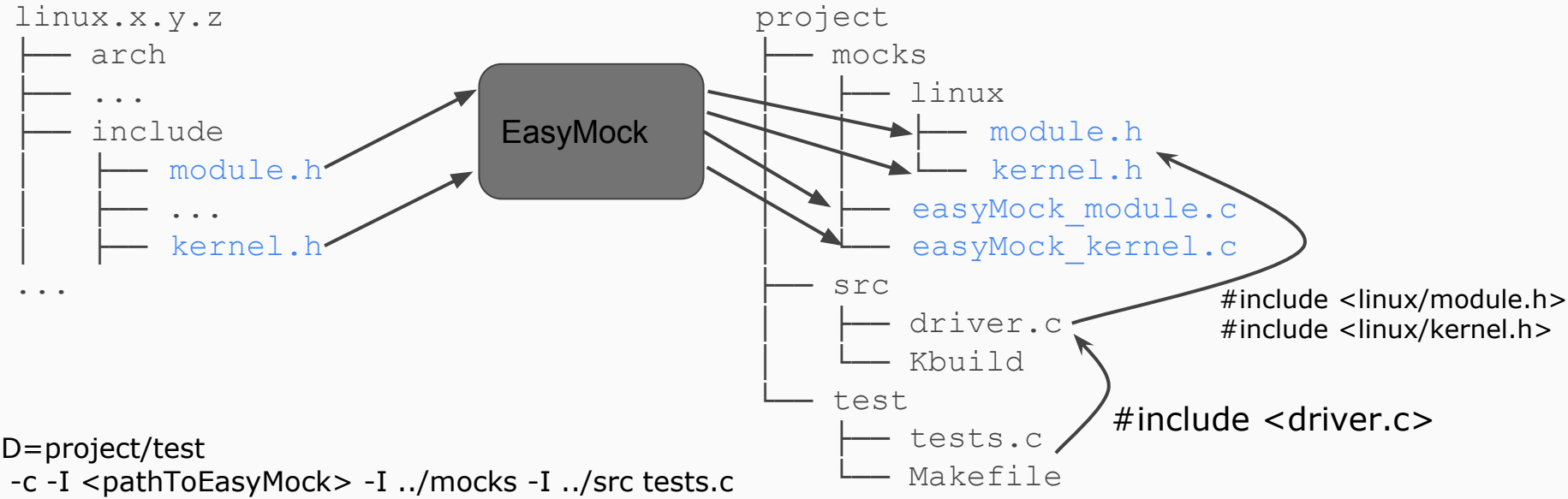
EasyMock helps verifying the function under test

- Each mocked function comes with a function to configure the mock
  E.g. Configuring the mock of `int sum(int a, int b);`
  is done by calling

```
void sum_ExpectAndReturn(
                int expectedA, int expectedB
                int returnValue,
                easyMock_match_list cmpA, easyMock_match_list cmpB);
```

  - By checking that the mock is called with the right parameter, we validate that the function under test is calling the dependencies correctly
  - By configuring the mock's return values, we can check that the function under test is correctly handling those values.
- EasyMock checks that each mocks are call the correct amount of time

Building unit tests



CWD=project/test
gcc -c -I <pathToEasyMock> -I ../mocks -I ../src tests.c
gcc -c -I <pathToEasyMock> ../mocks/easyMock_module.c
gcc -c -I <pathToEasyMock> ../mocks/easyMock_kernel.c

Demo

https://github.com/lcarlier/simpleFifoKernelDriver

Implementation of a FIFO character device

```
# echo "hello world" > /dev/simpleFifo0
# cat /dev/simpleFifo0
"hello world"
```

Conclusion

- Kernel code is compiled in a user space application
- Tests fast to compile
- Tests easy to debug
- Increased test coverage
- Mocks can be used to
  - Simulate user space input
  - Simulate hardware access
  - Verifying MMU mapping
  - ...

Thank you

https://github.com/lcarlier/EasyMock/