Software
Requirement Specification
for
Device driver with non-blocking read/write

1. Introduction

1.1 Abstract

A device driver is a computer program that operates or controls a particular type of device that is attached to a computer. A driver provides a software interface to hardware devices, enabling operating systems and other computer programs to access hardware functions without needing to know precise details about the hardware being used.

Key terms:

- **Device file:** This is a special file that provides an interface for the driver. It is present in the file system as an ordinary file. The application can perform all supported operation on it, just like for an ordinary file. It can *move, copy, delete, rename, read* and *write* these device files.
- **Device driver:** This is the software interface for the device and resides in the kernel space.
- **Device:** This can be the actual device present at the hardware level, or a pseudo device.

1.2 Project Scope

We are writing a driver `/dev/iitpipe` with non-blocking read/write. Input of `iitpipe0` is connected to output of `iitpipe1` and vice versa. So that the read write between the two pipelines goes into a buffer space .
On `write()` the data is written into the pipe after a delay. We use a kernel timer or workqueue. The write delay should be controllable get and set via `ioctl()`. We use dynamic memory allocation.

1.3 Goals of our project:

Measuring the throughput of our driver with read/write buffer size set to 1 B, 1 KB, 1 MB and with delay set to 0, 1 ms, 1 sec.

**Assuming the total data transferred be at least 100 MB in the 0 delay case and lower in other cases.

1.4. Overview of Document

The next chapter, the Overall Description section, of this document gives an overview of the functionality of the product. It describes the informal requirements and is used to establish a context for the technical requirements specification in the next chapter.

In Section 3 there are use case diagrams ,System Sequence Diagram and Interaction Diagrams.

2 System Description

Linux has a monolithic kernel. For this reason, writing a device driver for Linux requires performing a combined compilation with the kernel. Another way around is to implement your driver as a kernel module, in which case you won’t need to recompile the kernel to add another driver.

So in our project we are thinking to go with the second option.

Language details:

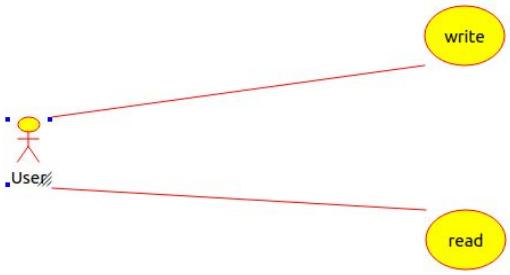
The Linux kernel was developed using the C programming language and Assembler. C implements the main part of the kernel, and Assembler implements parts that depend on the architecture.

- We are going to use C language in our project.

3. Use Case Diagram

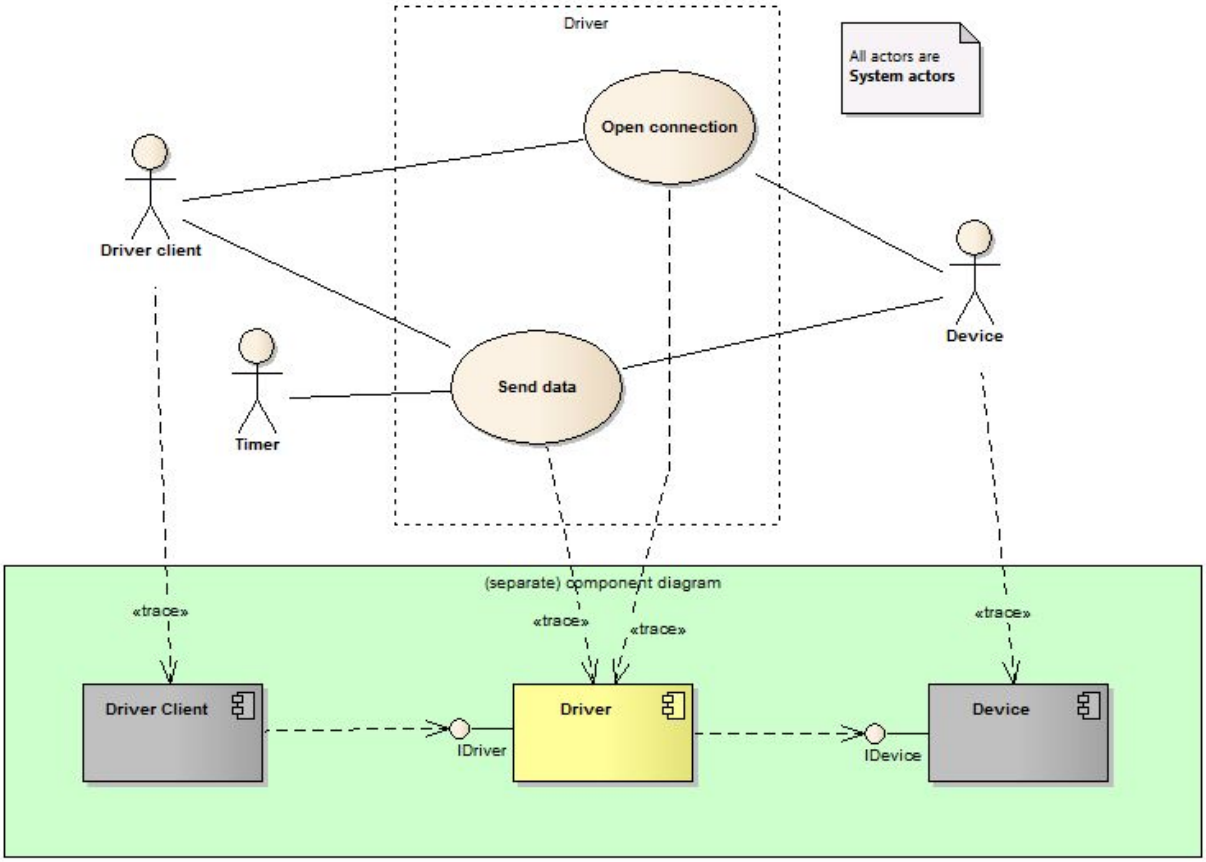
Actor : user of the driver
Use cases:

(1) Write data into Buffer via driver pipeline.
(2) Read data into Buffer via driver pipeline.



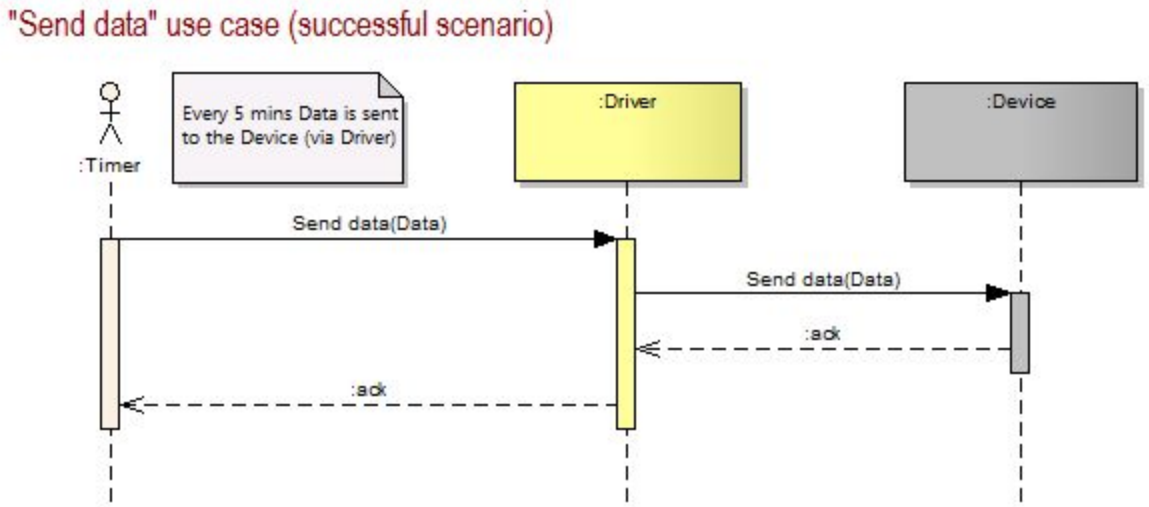
*** Note**:
The 'user' in a use case is not necessarily a person hitting keys or anything. It is an 'actor' external to the system being described who can interact with the system being described.

General Driver UML Diagram:



System Sequence Diagram:

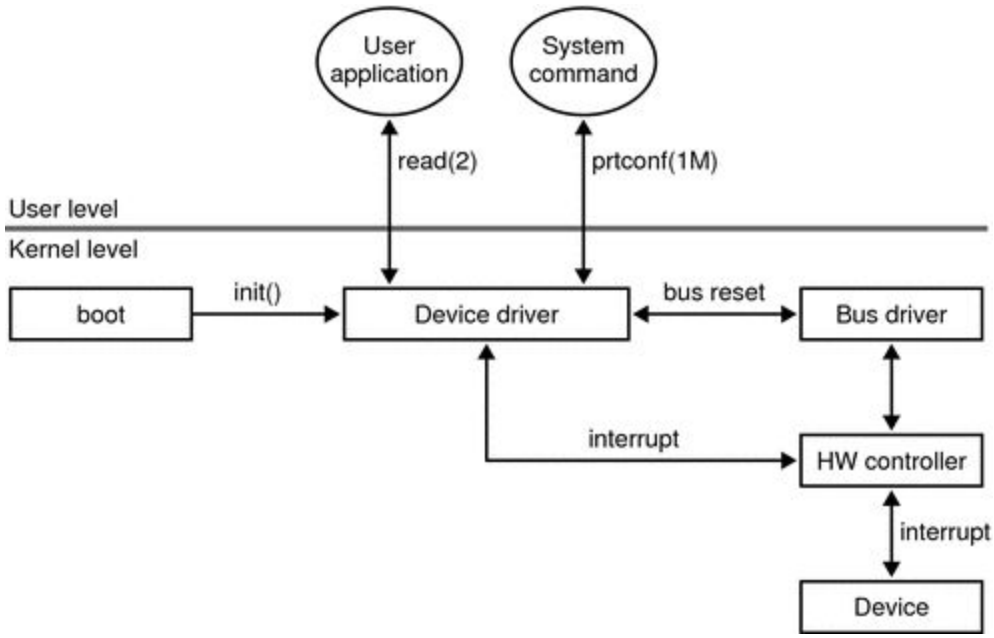
- We know buffer size set to 1 B, 1 KB, 1 MB and with delay set to 0, 1 ms, 1 sec.
- So timer sending with a delay 0, 1 ms, 1 sec.



Interaction Diagram of device driver with elements of operating system

Drivers are accessed in the following situations:

- System initialization. The kernel calls device drivers during system initialization to determine which devices are available and to initialize those devices.
- System calls from user processes. The kernel calls a device driver to perform I/O operations on the device such as open, read, and ioctl.
- User-level requests. The kernel calls device drivers to service requests from commands such as prtconf.
- Device interrupts. The kernel calls a device driver to handle interrupts generated by a device.
- Bus reset. The kernel calls a device driver to re-initialize the driver, the device, or both when the bus is reset. The bus is the path from the CPU to the device.



References

https://www.apriorit.com/dev-blog/195-simple-driver-for-linux-os
https://www.tldp.org/LDP/Linux-Filesystem-Hierarchy/html/dev.html
https://www.tldp.org/LDP/tlk/dd/drivers.html
https://opensourceforu.com/2014/10/an-introduction-to-device-drivers-in-the-linux-kernel/
https://stackoverflow.com/questions/24108298/drawing-use-case-diagram-for-device-driver
https://docs.oracle.com/cd/E19120-01/open.solaris/819-3159/emjjs/index.html