Software Design Document

for

Device driver with non-blocking read/write

Revision History

Version	Date	Author(s)	Description
v1.0	15/05/2018	Utkrisht,Koti, Sonith	Initial Version

Table of contents:

1. Introduction

We are writing a driver /dev/iitpipe with non-blocking read/write. Input of iitpipe 0 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.

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.

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.[3]
- **Device driver:** This is the software interface for the device and resides in the kernel space.[3]
- **Device:** This can be the actual device present at the hardware level, or a pseudo device.[3]

1.1 Intended Audience

This document is intended for software managers, coders and testers.

2 Detailed Design

2.1Architecture

When we write the data first goes into the buffer space from which the other device reads. Here we are going to implement the varying size buffer with respective delay in I/O. The Buffer size can be set accordingly.

We are maintaining one buffer space for each device. Here the devices are replica of one other.

There is no issue while increasing the size in an ascending manner the issue is only with the decreasing manner.

Issue: When we decrease the buffer size suddenly the will lose the information which is present in the buffer space but not read from the buffer on to device 2 of figure 1.

Solution : When we decrease the size of the buffer wait till the buffer space is empty . Once the buffer space is empty acknowledge the resizing of buffer and setting of its corresponding delays.That is we flush the buffer.

Components:

The components listed in figure 1:

1. Device 1: Convention for the device which writes into buffer.
2. Device 2: Convention for the device which reads from the buffer.
3. Pipe 0 : Pipeline in association with device1.
4. Pipe 1 : Pipeline in association with device 2.
5. Buffer Space: Varying size buffer in which the read/write between pipe 0 and pipe 1 goes into. That is the **<u>storage</u>** in the block diagram in figure 1.

Block Diagram:

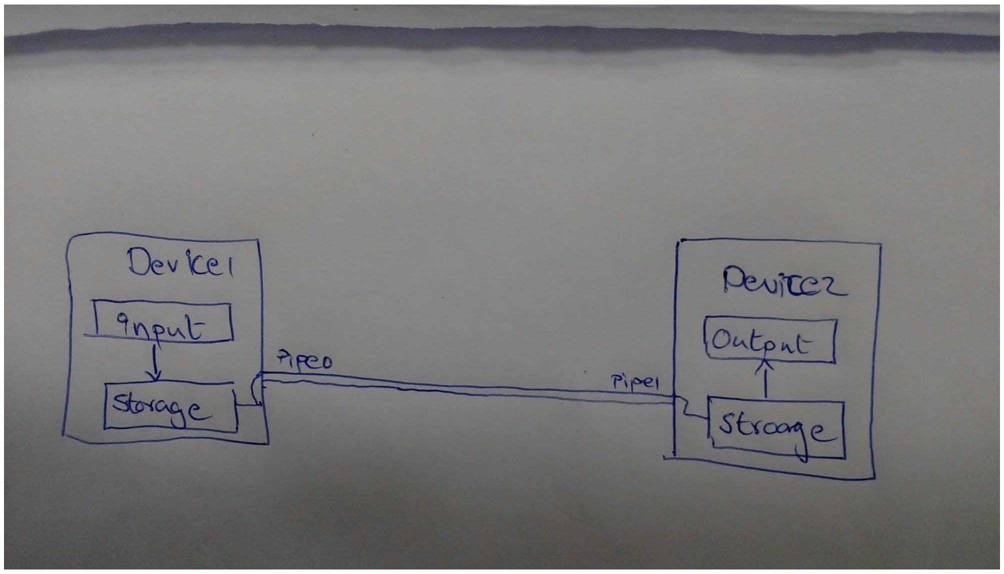


Figure 1 : Block Architecture

Interfaces:

Read, Write, IOCTL, Open, Close.

Ioctl:

Input/Output Control (ioctl, in short) is a common operation, or system call, available in most driver categories. It is a one-bill-fits-all kind of system call. If there is no other system call that meets a particular requirement, then ioctl() is the one to use.

```
int ioctl(int fd, int cmd, ...);
```

Here we use Ioctl with cmds to interact with the loaded module . Bases on the case it sets the Buffer size.

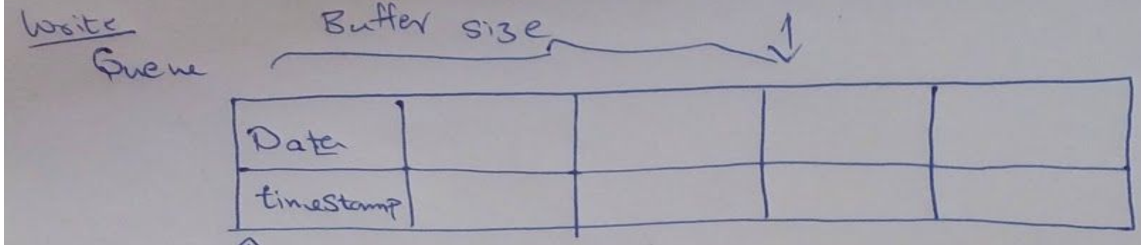
Pseudo code:

```
switch (cmd)
    {
        case set 1 b:
            setbuffersize(writebuffer,1);
            break;
        case set 1 kb:
            setbuffersize(writebuffer,10);
            break;
        Case set 100Mb:
            setbuffersize(writebuffer,10);
            Break;
}
```

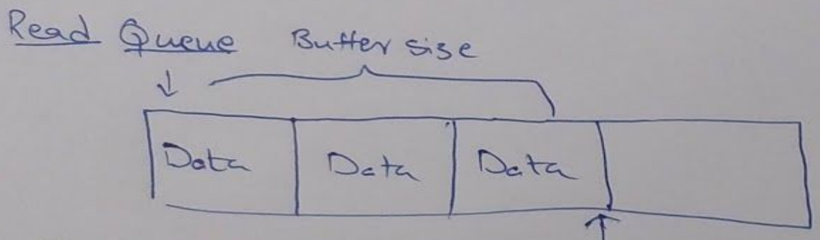
2.2 Algorithms and Data Structures

There are no significant algorithms developed for this product. The only important data structure is a queue for the Buffer.

Queue Data Structure we use



The write Queue has two parts
(or) each node of queue
has (1) Data
(2) The time stamp associated with it.



The read Queue only has data

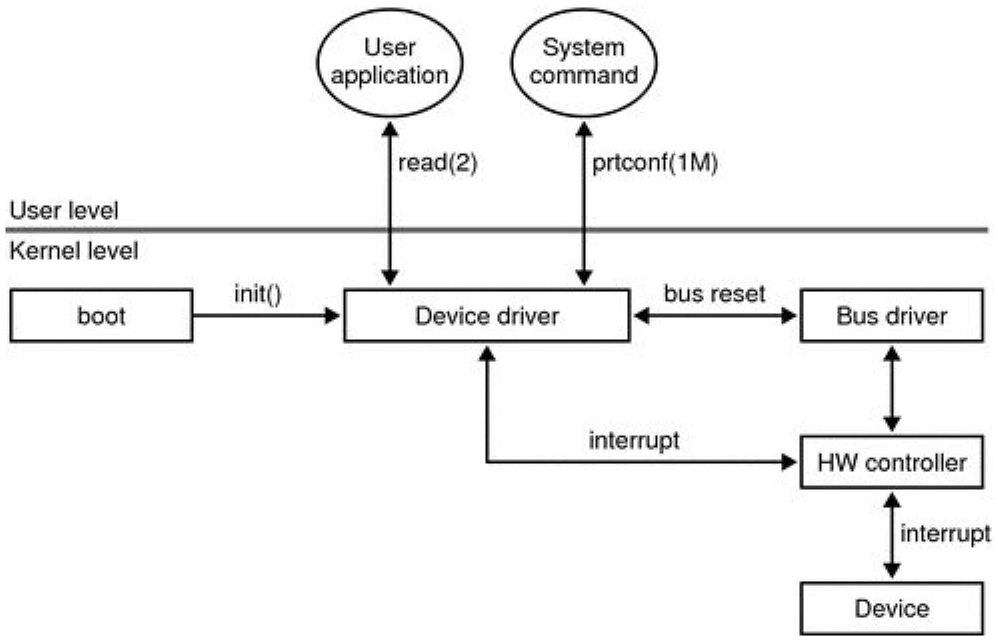
Once the buffer size is exeeeded the data is poped from the queues.

```
Pseudo code
if  Buff size < Size( Write_Queue)
        Reedinto Read Queue
if Buffsize < Size (Read-Queue)
    Read from Read Queue
```

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.



3 References

[1]https://www.apriorit.com/dev-blog/195-simple-driver-for-linux-os
[2]https://www.tldp.org/LDP/Linux-Filesystem-Hierarchy/html/dev.html
[3]https://www.tldp.org/LDP/tlk/dd/drivers.html
[4]https://opensourceforu.com/2014/10/an-introduction-to-device-drivers-in-the-linux-kernel/