Device driver with non-blocking read/write

Indian
Institute of
Technology
Mandi

Problem Statement:

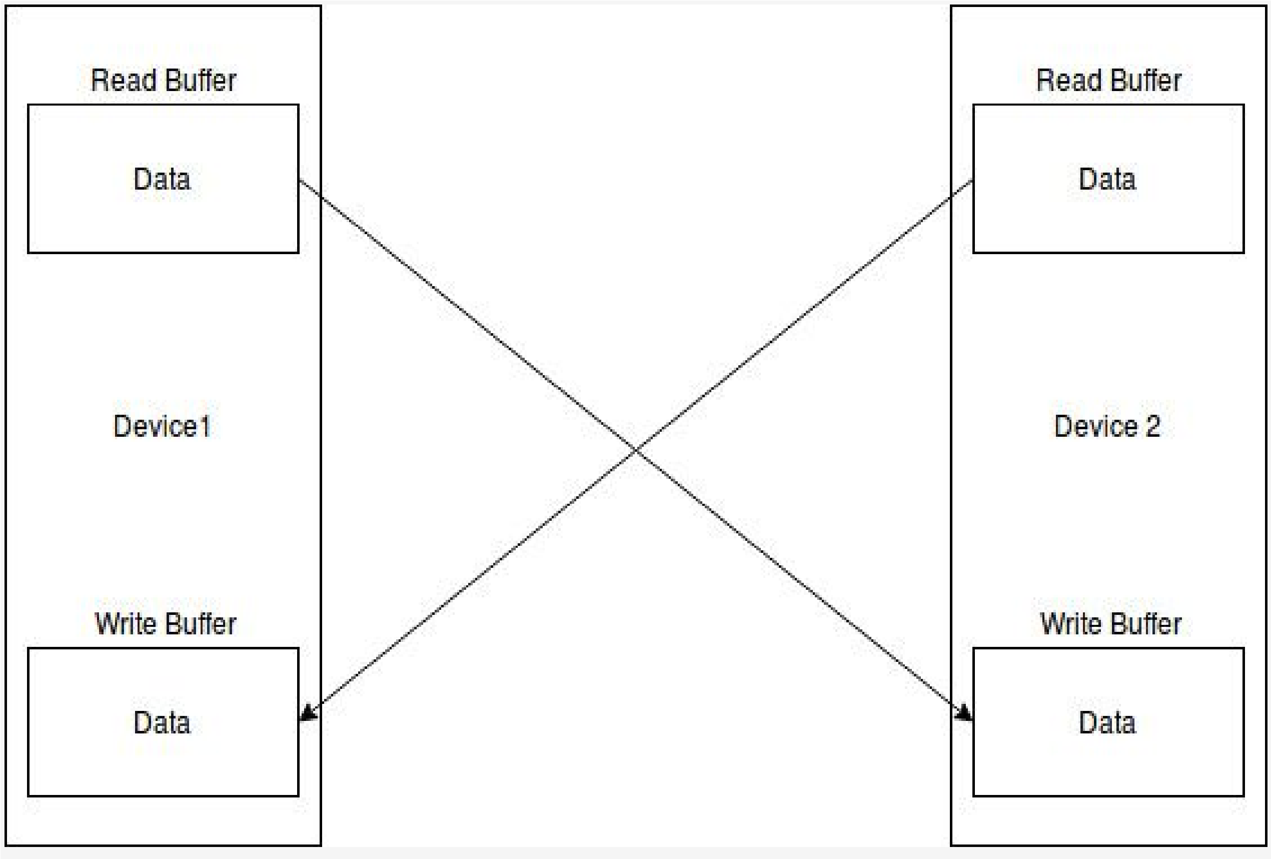
We are writing a driver /dev/iitpipe with non-blocking read/write. Input of iitpipe 0 is connected to output of iitpipe1 and vice versa. So that the read write between the two pipelines goes into a buffer space .
On write() the data is written into the pipe after a delay. We use a kernel timer or workqueue. 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.

System design:



We have two device files with read/write, open/close which act as devices. The Read and Write buffers associated with the devices have the queue data structure.

```
struct circ_queue_node {
        char byte;
        long timestamp;
};
```

Queue API:

circ_queue circ_queue_init(size_t size): Initializes the queue to given size
circ_queue_count(struct circ_queue queue): Counts the length of the queue (i.e nodes)
circ_queue_push(struct circ_queue *queue, struct circ_queue_node node): pushes the node into the queue.
circ_queue_peek(struct circ_queue *queue, struct circ_queue_node *node): outputs the first node of the queue.
circ_queue_pop(struct circ_queue *queue, struct circ_queue_node *node): pops and outputs the node from the queue.
circ_queue_free(struct circ_queue queue): free’s the memory allocated to the queue .

Initialization:

We implemented the device driver as a module. When the module loads the it calls a __init iitpipe_init(void) using module_init(iitpipe_init) in which we first register the major number to the devices and create a class for the devices and then initialize the read/write buffers of the devices using circ_queue circ_queue_init(size_t size) function where size as default size( ).

How the data is read into Read Buffer ?

When we get the stream of data we store it first in a string using inbuilt functions copy_from_user which copies the string which comes as a argument into a local char* memory and then iteratively we push character by character into the read queue of the device.

And we free the local char* memory we created.

Timer-Function:

We have a timer function which calls the update_queue function which checks the time-stamp of the top element of the read buffer of a device and based on delay condition if satisfied it starts writing into the write buffer of the other device.

How IOCTL works?

In our project the ioctl is implemented as my_ioctl using switch which itself is a device file.

Based on the cases it does three tasks:
DELAY_GET : get information of current queues size and their delays parameters.
DELAY_SET : re-initialize the current queues dropping all the data it had initially and setting up the new parameters.
DELAY_CLR : All the memory allocated for the queues is cleared.

Test Cases/Performance results:

Buffer Size: 1 B
Delay 0ms - 110 bytes/s
Delay 1ms - same as above
Delay 1s - oscillates b/w 0 and 1

Buffer Size: 1 KB
Delay 0ms - 230 KB/s and oscillates dips down to 100KB/s some times
Delay 1ms - same as above
Delay 1s - oscillates b/w 1 and 0KB/s

Buffer Size: 1 MB
Delay 0ms - 17.5 MB/s 24MB/s
Delay 1ms - 17.5 MB/s 24MB/s
Delay 1s - Highly erratic and oscillates b/w 1 and 0 MB/s

Conclusion:

Learnt how device drivers work at kernel level. We were targeted to write at 100MB/s and we’ve achieved only 20MB/s.