SHORT REPORT

1 Creating the circular queue

A circular queue was implemented on a character array. Functions to initialize, insert and extract characters were created: `circular_queue.h`

2 Creating the device

2.1 Initialization

A device `my_char_device` was registered in the kernel using `alloc_chrdev_region()` giving 0 as major number so that a dynamic allocation will take place. The value returned was saved as major_num.

From the shell, "`mknod /dev/my_char_device <major_num> 0`" was executed to create the device. The device was given a permission of 777.

A cdev structure was allocated using `cdev_alloc()` passing this registered device details. The operations of the allocated structure was set as "fops", which was the set of device access functions I made.

The cdev was added to the kernel using `cdev_add()` function.

2.2 device_open: Opening the device

A mutex lock was applied on mutex variable `mutex_dev`.

2.3 writer: Writing into device

A mutex lock was applied on variable `mutex_prodcons` to avoid synchronization problem between readers and writers. The amount of free space in the queue is compared with the bytes to write, and the smaller value is selected for writing. Using a for loop, that number of bytes are taken from input buffer and inserted into the queue. Finally, the mutex is unlocked.

2.4 reader: Reading from device

The mutex_prodcons mutex lock was applied. The number of bytes stored in the queue and the number of bytes to read/delete from the queue is compared, and the smaller value is selected. Using a for loop, that number of bytes are deleted from the queue and put into the output buffer. Finally, the mutex is unlocked.

2.5 device_close: Closing the device

The mutex lock `mutex_dev` is unlocked.

2.6 Removal

The device was unregistered using `unregister_chrdev_region()` function call.

3 Creating the procfs device

The current state of a queue – number of reads and writes, remaining space, positions of front and rear – were stored in a structure `q_state`. Functions for initializing, updating state on queue write and read, and to return current queue state (procfs) were implemented: `qstate.h`

3.1 Initialization

A procfs file was created using `proc_create()` function, passing `my_char_device` as filename and `proc_fops` as file operations. It is a set of functions for reading from procfs which I implemented.

3.2 Reading Queue State

The open call of the procfs is executing a function that calls `get_QState()` which returns the current queue state. It then prints the state and exits.

3.3 Removal

The procfs device was removed on module unload using `remove_proc_entry()` function call.

4 Output

The first state output after initializing the queue:

```
Reads       : 0
Writes      : 0
Free Space  : 10
Front       : 0
Rear        : -1
```