**CS492 Assignment 3**

Kernel-Based FIFO (queue)

# Task Description

The objective of this assignment is to implement a kernel-based first-in-first-out (FIFO) queue by extending a barebone character device driver. This FIFO can be used by multiple producers/consumers exchanging data. With this assignment you will familiarize yourselves with concurrency primitives, like mutexes, semaphores, and spinlocks. Specifically, you will:

* Extend a character device driver to implement an FIFO that supports multiple producers/consumers
* Test the code in the VM we distributed for the class, using the default kernel

# Project Steps

1. Fork the repository: <https://classroom.github.com/a/W70cSWup> (you will have a private repo containing the driver after you click the link and accept the assignment)
2. Relative information: LDD3 chapter 3, Slides on Concurrency
3. Extend the character device driver (under driver/). The driver should implement **one** FIFO queue as an **array** of **N** elements of size **ELEMSZ**. N and ELEMSZ can be configured during module load through module parameters. Your tasks:
   1. Correctly allocate the FIFO when the device is opened; Correctly free the FIFO when the device is closed. Information on memory allocation/free in the kernel can be found here <https://www.kernel.org/doc/html/latest/core-api/memory-allocation.html>
   2. Implement the read (consume) FIFO operation in the device driver (a placeholder is inserted in driver/scull.c):

ssize\_t scull\_read(struct file \*filp, char \_\_user \*buf, size\_t count, loff\_t \*f\_pos)

This operation consumes one element, copying the bytes of the element in “buf” and returning the number of bytes copied. The maximum number of bytes that can be copied is (by definition) ELEMSZ. If count is smaller than the size of the next element in the FIFO, then only count bytes of the element will be copied to buf. f\_pos is not used. This operation blocks if there are no elements to consume.

* 1. Implement the write (produce) FIFO operation:

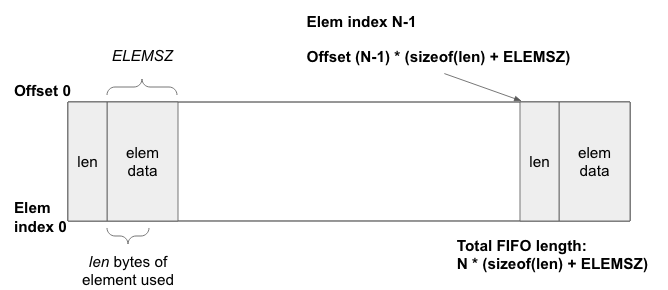
ssize\_t scull\_write(struct file \*filp, const char \_\_user \*buf, size\_t count, loff\_t \*f\_pos)

This interface produces one element, copying count bytes from buf into the new FIFO element. The maximum number of bytes that can be copied into the new element is (by definition) ELEMSZ. If count is larger than ELEMSZ, then only ELEMSZ bytes are copied; otherwise count bytes should be copied. This operation returns the actual number of bytes stored in the FIFO. f\_pos is not used. This operation blocks if there is no space to store a new element.

* 1. Implement synchronization primitives to ensure safe concurrent access of the FIFO. Please use sleeping/waking instead of busy-waiting.
  2. Describe the synchronization primitives you have used to ensure safe concurrent access of the FIFO in REAMDE.md. If you adopt the approach in the slides, you can cite the slides and keep this simple. If you pick a different set of primitives (or combination of them), you need to justify how this approach works and why it satisfies the requirements (both correctness and efficiency). Include your details, including a link to the github classroom repo for the assignment.

NOTES:

* Please do not assume the user space program knows the size of the FIFO array and the size of each element. The driver already implements an IOCTL interface (SCULL\_IOCGETELEMSZ) to return the size of each element (ELEMSZ) to userspace processes.
* You need to design the array to correctly implement scull\_read and scull\_write (the operations must ensure FIFO).
* If you are using semaphores and mutexes to block consumer/producer processes use their interruptible variations, so if you decide to terminate the processes using a signal (e.g., through Ctrl-C), they can be interrupted. Remember to check and return an appropriate, when that happens.
* You need to decide how to allocate/free the FIFO so different N and ELEMSZ can be supported. The only limitation should be memory availability, not your implementation. Hints:
  + You will need to check if your memory allocation succeeds; if not, you should return an error when the user space program tries to open the device.
  + Because the size of the FIFO storage elements (ELEMSZ) is not known at compile time, you will need to do pointer arithmetic to get a pointer to each element. Also, because processes can store less than ELEMSZ in the FIFO, you will need to also store the number of bytes actually stored in each storage element. See picture below.



* The assignment does not require that you submit userspace programs that test your FIFO (we will use ours for testing), however, you are *highly encouraged* to develop your own tests. You can modify src/consumer.c and src/producer.c if you wish and include them in your submission (bonus will be given).

1. **Bonus task**: extend the driver to support a new IOCTL operation (SCULL\_IOCSETSIZE) that (re-)configures the FIFO at run time. Specifically, this new IOCTL can be used to change the size of the FIFO (i.e., the number of elements that can be stored in the FIFO). The IOCTL accepts one argument (int), which specifies the new FIFO size.
   1. If the new size is larger than the current size, the FIFO should be expanded, **while retaining existing data.**
   2. If the new size is smaller than the current size, the FIFO should be shrunk, retaining existing data, unless that would cause dropping FIFO elements. In the latter case, the request should return an error.

NOTES:

* Concurrency should be taken into account while doing this. Resize requests should complete correctly, when other requests, including other resize requests are happening concurrently.
* Update README.md explaining what modifications you had to make to ensure correct operation despite concurrent requests.

1. Create a tar archive of your repo including only .c, .h, and Makefile files. Do not include the .git directories. Only the following files should be included:

fifo

fifo/driver

fifo/driver/debug.h

fifo/driver/scull.init

fifo/driver/Makefile

fifo/driver/scull.h

fifo/driver/access\_ok\_version.h

fifo/driver/scull.c

fifo/README.md

fifo/src

fifo/src/producer.c

fifo/src/Makefile

fifo/src/consumer.c

1. Submit the archive using gradescope: <https://www.gradescope.com/courses/241080/assignments/1162050>

# Assignment Points

c) Main tasks 100 points

| **Task** | **Points** |
| --- | --- |
| Correctly allocating/freeing FIFO | 20 pts |
| Read works correctly (ignoring concurrency) | 15 pts |
| Write works correctly (ignoring concurrency) | 15 pts |
| Correct read/write support for concurrent access | 30 pts |
| Check and handle possible errors correctly (e.g., failures of memory allocations) | 10 pts |
| Explain synchronization primitives used in README.md | 10 pts |

d) Bonus tasks 15 pts

| **Task** | **Points** |
| --- | --- |
| User space programs (for testing) | 5 pts |
| Resize to larger FIFO (w/concurrency) | 5 pts |
| Resize to smaller FIFO (w/concurrency) | 5 pts |

# Grading Policy

* This is an individual assignment. Individual assignments, as the words indicate, are to be done INDIVIDUALLY. Any form of plagiarism (from each other or the Internet) will result in a 0 and you will be reported to the Honor Board.
* Any late submission without pre-approval will result in a 0.
* The assignment will be graded based on correctness and style (well formed code with comments). Code not compiling will result in a 0 in the corresponding assignment task.