INFO 72220 – ASSIGNMENT 5 : (Mock) Linux Device Driver

Release: Monday, November 25, 2019
Due: Sunday, December 8, 2019

Author: Scott Chen

The basic Linux device driver design process demonstrated in class consists of a mock character device and an incomplete driver that merely prints the kernel information message.

In this assignment, you are required to submit a driver, and a client-server program pair that communicates with each other via the driver using the standard C open(), close(), read(), and write() functions. Here is an example interaction model between the client and the server programs via the driver:

- Example Interactions
    - Client.c displaying the menu "when: get time, who: get name, where: get place" before prompting user input
    - User types in "when". Client.c sends the command "Cwhen" using write() into the mock driver, with the leading 'C' character indicating it is a client command.
    - After sending the command, Client.c starts a timer of 5 seconds before attempting to read() from the driver. After every read(), check the first character and see whether it is a response 'R'; if yes, display the response and end the timer; otherwise, reset the 5-second timer and read the response later.
    - Server.c read() from the driver once every second, and check whether there is an incoming command with a leading character 'C'. If not, retry the next second; if yes, start processing the command.
        - Command "when" => output the current time
        - Command "who" => output your name
        - Command "where" => output your school name
        - Other commands can be added based on your likings.
    - Server.c writes() the response to the mock driver with 'R' appended at the start of the response.
    - Client.c should read the response with a leading character 'R' after server responded, display the server response on the client terminal screen.

- You are not restricted to implement a different client-server command and response model, as long as the interaction goes through the buffer in the driver.
- You can choose to keep both the server and the client as a one-shot program, or an ongoing program until an exit command is issued to close the two programs.
- You are not restricted to implement a single buffer in the driver.
- You are not required to implement a command and response queue. The gist of the assignment is the design of the driver, NOT the buffer mutex or client-server coordination. As a result, race conditions are not going to affect your marks unless it completely destroys your driver functionalities.

Tips on Kernel Memory Buffer implementation in the driver

Implement a kernel memory buffer in your mock character device driver requires you to use the kmalloc() family of functions. A recommended example buffer arrangement would be:

- The buffer is allocated in the constructor static function
- The buffer is deallocated in the destructor static function
- The buffer holds a string up to 128 characters
- The write static function should write to the buffer with a special termination character (e.g. a newline character '\n'). If the string is longer than 127 characters, truncate the string to 127 characters + 1 termination character.
- Every write operation to the buffer completely overwrites the previous contents in the buffer for simplicity. (no queue required)
- The read static function should read the contents from the buffer and only pass back the string content up to the termination character.
- The string manipulation process should remain completely transparent to the user processes.

Tips for preparing your Client and Server programs

Modify the original TCP Socket client.c and server.c example code demonstrated in the socket programming class, or any client-server-based codes you've developed in previous assignments, such that:

- The client and server communicate to each other via the buffer in the character driver using your predefined ASCII communication protocol.
- Both client and server must access the character driver using the standard C open(), close(), read() and write() functions as if the character device is a file object.
- The client should show a menu of commands before prompting user inputs, and send the corresponding command to the server via the standard write() function.
- The server should receive the command via read() function, process the command from the client, and write() back to the buffer with its response.
- The client should read() the response from the client after write() its command into the driver.
- You may or may not have to close() the driver after each client-server action.

Your submission must consist of the Device Driver C file, the modified client.c, and the modified server.c.

- The Device Driver C file must be compilable with the kernel library to produce the required kernel module (*.ko) immediately mountable to the Linux kernel. You are recommended to also submit all the driver-related files in a separate zip file.
- The client.c and server.c must both be compilable and run in two separate terminals to imitate two processed on different machines talking across a device using the mock device driver.

- Marking Scheme:
    - Correct implementation of kernel memory buffer allocation in the driver (2 marks)
        - 1 mark for allocation
        - 1 mark for deallocation
    - Correct implementation of command and response string process in the read() and write() functions in the driver (5 marks)
        - 2 mark for read()
            - 0.5 mark for newline character deployment
            - 1.5 mark for proper readout procedure
        - 3 mark for write()
            - 0.5 mark for newline character deployment
            - 0.5 mark for truncating strings longer than 127-byte
            - 0.5 mark for overwriting the previous contents in the buffer properly
            - 1.5 mark for proper write-in procedure
    - Correct implementation of client.c (1.5 mark)
        - 0.5 mark for requesting user input
        - 0.5 mark for proper coordination between command write() and response read().
        - 0.5 mark for displaying proper UI (providing menu instructions when prompting for user input, and display responses from server properly)
    - Correct implementation of server.c (1.5 mark)
        - 0.5 mark for reading, checking and parsing the incoming command
        - 0.5 mark for interpreting the incoming command and produce the correct responses.
        - 0.5 mark for proper implementation of response write().