Assignment 4: Writing a Simple Device Driver

Programming Embedded Systems

Exercise

(a) Character Device Driver (15 points) : Write a linux character device driver module that implements the open(), close(), read() and write() system calls for a character device file. This character device file will dynamically manage buffer space to store the last string written to the character device and output this string upon reads. Writes to the device file will allocate buffer space to match the size of the user's write and fill that buffer space with the user's written data. Reads of the device file will return the last string written to the buffer and deallocate the buffer's memory space. The device driver must handle string writes of arbitrary length up to 100 bytes. Writes larger than 100 bytes should be truncated to 100 bytes. The driver must manage the kernel memory space such that the internal buffer is dynamically sized to the minimum required space for the currently stored string. No space should be consumed when the buffer is empty. Reads from an empty buffer should return an -EINVAL error code. Writes to a full buffer should return an -ENOSPC error code. Initially, this buffer will be empty until the first write. Use a semaphore to ensure that race conditions caused by simultaneous writes or reads are handled appropriately. The driver must do the following for full credit:

- Implement methods for each of these four file operations: open(), close(), read() and write().
- Dynamically manage buffer space.
- Handle errors in any kernel function call.
- Handle improperly sized user input.
- Error on full writes and empty reads.
- Use semaphores to lock appropriate data structures and avoid race conditions.
- Clean up properly upon removal.

(b) Test Application (5 points) : Write a test application that writes randomly generated ASCII strings of random length up to 200 bytes to the device file and then reads the data back to confirm the correct operation of the program. The C standard library random() function generates random numbers, read its man pages for details on how to use it. Be sure that the random numbers you generate fit in the range of legally printable ASCII encodings. The test program must iterate this write-read test 10 times. It should print the strings

(in hex) prior to sending them to the device file and print the output from the device file as well as a pass or fail the outcome. Note: You need to use a linux computer on which you have root access to test your device driver code.

Submission Instruction

You need to submit all relevant source and makefiles of the driver and the test application together with a brief report compressed in a single zip file. The report should include all relevant steps to check your submission, like creating the device, compiling and installing your driver, running the test application and a sample output.