U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Serial Device Driver Project

Operating Systems - MIEEC

Rodrigo Caldas, up201708987@fe.up.pt

Celso Figueiredo, up201808896@fe.up.pt

State of Development of Device Drivers

The goal of this project was the development of two device drivers for a virtual serial port: one whose operation is a poll-based (SERP) and another which operates based on interrupts (SERI).

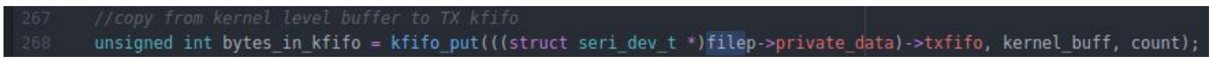
Both the SERP and the SERI device drivers were implemented according to the provided instructions on the 3rd and 4th lab guides.

Furthermore we implemented 5 of the enhancements which were proposed in the 5th lab guide, namely enhancements 3.1, 3.2, 3.3, 3.5 and 3.6.

Enhancements

3.1 - Minimize global variables

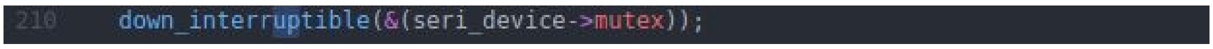
We made an effort to avoid the use of global variables whenever it seemed possible. This was done for all implemented functions except for the init and exit functions, since these 2 functions do not receive arguments nor return values. As a code example, we show how the seri_write function is able to access and place values in the transmission FIFO queue, using the file pointer provided as a function argument instead of a global variable.



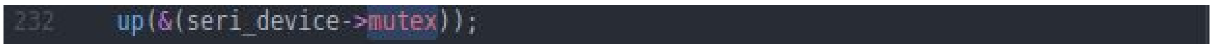
```c
//copy from kernel level buffer to TX kfifo
unsigned int bytes_in_kfifo = kfifo_put(((struct seri_dev_t *)filep->private_data)->txfifo, kernel_buff, count);
```

3.2 - Eliminate race conditions

Race conditions are avoided by the implementation of a kernel semaphore, named mutex and initialized with the value 1. This only allows one thread to run the seri_open function. If a second thread tries to run this function before the first thread increments the semaphore counter, it will be blocked. As code examples, the code lines where the semaphore is respectively decremented and incremented in seri_open and seri_close are shown.



```
    down_interruptible(&(seri_device->mutex));
```



```
    up(&(seri_device->mutex));
```

3.3 - Honor the O_NONBLOCK flag

At the seri_read and seri_write functions, if the O_NONBLOCK flag is active, these functions respectively verify if the reception fifo queue is empty or if the transmission queue is full. If that is the case, it means that no I/O operations are to be done immediately, and that the program would block the operation of the serial port. Therefore, if these conditions are verified, these functions return immediately and print an appropriate alert message. This enhancement can be tested with the provided nonblock.c program, which calls open using the nonblock flag and terminates immediately if no data is present on the driver's reception FIFO queue.

3.5 - Allow for interrupting a process inside a read() syscall

The wait_event_interruptible_timeout function, which is called in the loop that checks the FIFO queue length and waits until it is equal to value provided by the count argument, allows the seri_read function to detect if an interrupting signal occurred, such as the user pressing Ctrl + C. This enhancement can be tested with the provided read.c program, by pressing Ctrl + C while it is running.

3.6 - Add access control to prevent more than one "user" access

We used an integer variable, named sem_status, which is updated along with the kernel semaphore's counter. If seri_open is called and the variable sem_status shows that the device is already being used, seri_open returns immediately and prints an appropriate alert message. It was initially thought to just use the kernel semaphore's counter value itself, instead of using an additional variable. However while reading online documentation about kernel semaphores, we read a warning against accessing any members of the kernel semaphore structures directly. This enhancement can be tested with the provided user.c program, which just calls two consecutive open operations on the device, and causes the access control error.

Testing sequence

Two test programs were created for SERP and SERI in order to test writing (**write.c**) and reading (**read.c**).

- Testing sequence for SERP:
  1. **sudo ./disableserial.sh** in order to disable COM1
  2. make
  3. **sudo ./load.sh serp** to load the module "serp" to the kernel
  4. Compile and run test programs
     a. **./read** to test reading operations

        or

     b. **./write** to test writing operations
  5. **sudo ./remove.sh serp** to remove the module "serp" from the kernel

- Running Process for SERI:
  6. **sudo ./disableserial.sh** in order to disable COM1
  7. make
  8. **sudo ./load.sh seri** to load the module "seri" to the kernel
  9. Compile and run test programs
     a. **./read** to test reading operations and enhancement 3.5 (press Ctrl+C)

        or

     b. **./write** to test writing operations
     c. **./nonblock** to test enhancement 3.3

        or

     d. **./user** to test enhancement 3.6
  10. **sudo ./remove.sh seri** to remove the module "seri" from the kernel