Otto-Friedrich-Universität Bamberg
Fakultät WIAI

Lehrstuhl für Praktische Informatik, insbes.
Systemnahe Programmierung – Prof. Dr. Michael Engel
https://www.uni-bamberg.de/sysnap
michael.engel@uni-bamberg.de

Operating Systems Engineering

Exercises
Summer 2022

Exercise 5

Synchronization

Discussion on Monday, 25.7.2022

Solving this exercise will require quite a bit of reading of the specs and experimenting with the interrupt handler. You are not expected to solve all detail problems here (but you should of course try...). The following lecture will discuss details of the PLIC.

5.1 UART interrupt handling

Write an interrupt handler for the UART:

- Initialize the UART (baudrate, format, enable **receive interrupts**) in `setup.c`
- Initialize the PLIC interrupt controller to enable external interrupts from the UART (interrupt number: 10) in `setup.c`
- Enable M-level external interrupts in `setup.c`
- Extend the exception handler in `exception.c` to detect and handle external interrupts
- Handle interrupts for incoming (received) characters by printing the character to the console in the interrupt handler
- *Hint:* Reading the `RBR` register clears the receive interrupt

Hint: Sending via the UART will remain in polled mode so far, so *do not* enable the transmit interrupt (or any other except for the receive interrupt)!

5.2 Unix-like UART device driver

Now write the other half of the device driver for the UART. Enable access to the UART only when a UART device is opened.

- Implement two syscalls `uart_open` and `uart_close`. A process that wants to read characters from the UART now has to open it before and close it afterwards. Only a single process may access the UART at the same time, so trying to call `open_uart` when another process is using the UART will result in an error
- Accordingly, make the existing syscall `getachar` block the calling process (only valid if a previous `open_uart` succeeded, otherwise return an error) for the next character to arrive. This implies that you should implement *process states* now. So far, we implicitly have one process in state `RUNNING` and all others in `READY`. With a new `BLOCKING` state, a process that is blocked will not be considered by the scheduler
- Communicate between the interrupt handler and the system call using a shared variable and return the character to the blocked process after the exception handler has noticed the character's arrival

5.3 Ring buffer and mutexes

Now implement a *ring buffer* to communicate between the UART interrupt handler and the system call handler. Protect access to the ring buffer using a *mutex*.