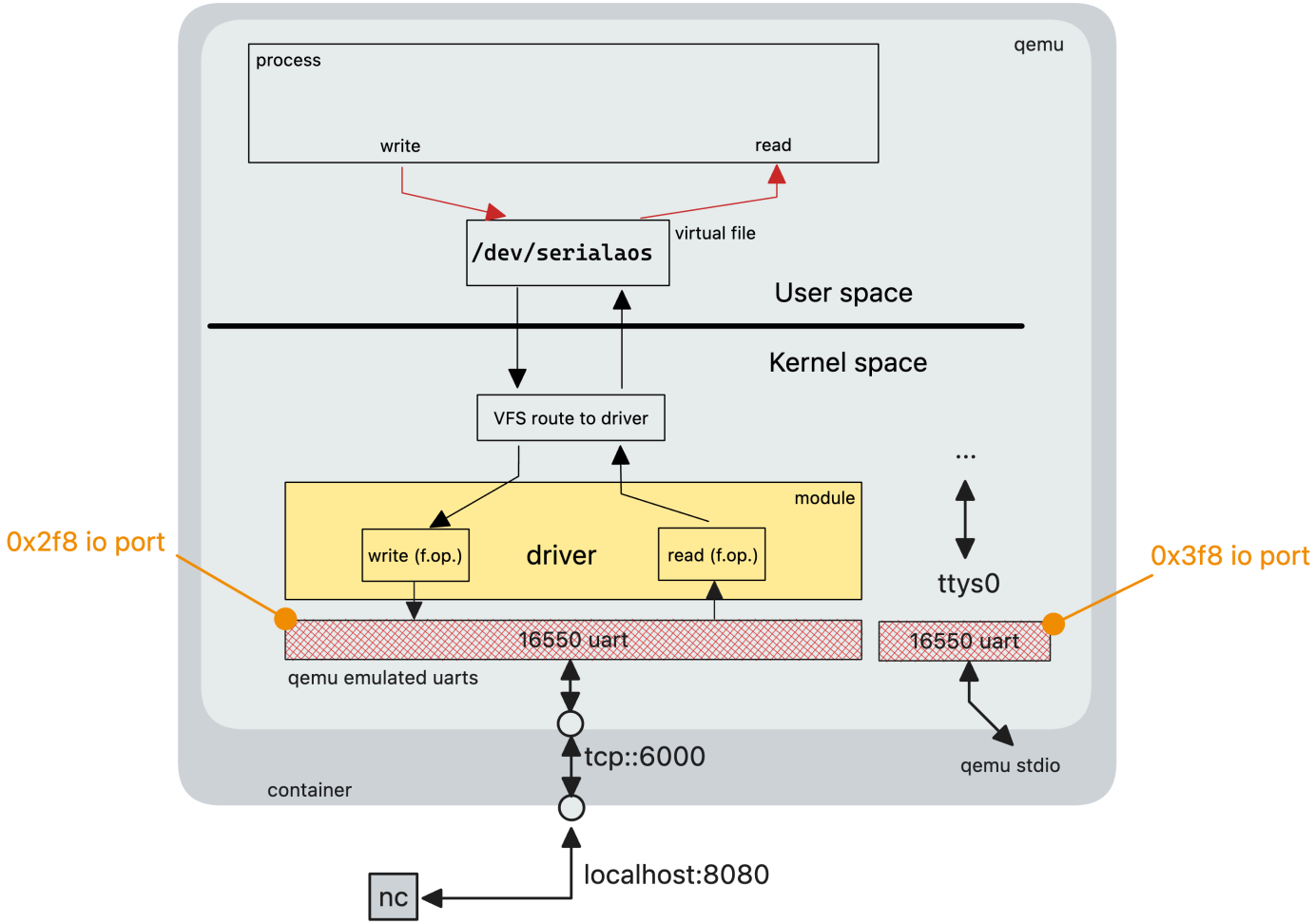
Advanced Operating Systems (labs)

Vittorio Zaccaria | Politecnico di Milano | '24/25

Writing a UART driver for Linux



The structure of the driver we are going to build

The 16550 UART device

0x5 enable RDA and RLS

0x0 disable hw FIFO

Bit No.	REGISTER ADDRESS											
	0 DLAB=0	0 DLAB=0	1 DLAB=0	2		3	4	5	6	7	0 DLAB=1	1 DLAB=1
	Receiver Buffer Register (Read Only)	Transmitter Holding Register (Write Only)	Interrupt Enable Register	Interrupt Ident. Register (Read Only)	FIFO Control Register (Write Only)	Line Control Register	MODEM Control Register	Line Status Register	MODEM Status Register	Scratch Register	Divisor Latch (LS)	Divisor Latch (MS)
	RBR	THR	IER	IIR	FCR	LCR	MCR	LSR	MSR	SCR	DLL	DLM
0	Data Bit 0[1]	Data Bit 0	Enable Received Data Available Interrupt (ERBFI)	"0" if Interrupt Pending	FIFO Enable	Word Length Select Bit 0 (WLS0)	Data Terminal Ready (DTR)	Data Ready (DR)	Delta Clear to Send (DCTS)	Bit 0	Bit 0	Bit 8
1	Data Bit 1	Data Bit 1	Enable Transmitter Holding Register Empty Interrupt (ETBEI)	Interrupt ID Bit (0)	RCVR FIFO Reset	Word Length Select Bit 1 (WLS1)	Request to Send (RTS)	Overrun Error (OE)	Delta Data Set Ready (DDSR)	Bit 1	Bit 1	Bit 9
2	Data Bit 2	Data Bit 2	Enable Receiver Line Status Interrupt (ELSI)	Interrupt ID Bit (1)	XMIT FIFO Reset	Number of Stop Bits (STB)	Out 1	Parity Error (PE)	Trailing Edge Ring Indicator (TERI)	Bit 2	Bit 2	Bit 10
3	Data Bit 3	Data Bit 3	Enable MODEM Status Interrupt (EDSSI)	Interrupt ID Bit (2) [2]	DMA Mode Select	Parity Enable (PEN)	Out 2	Framing Error (FE)	Delta Data Carrier Detect (DDCD)	Bit 3	Bit 3	Bit 11
4	Data Bit 4	Data Bit 4	0	0	Reserved	Even Parity Select (EPS)	Loop	Break Interrupt (BI)	Clear to Send (CTS)	Bit 4	Bit 4	Bit 12
5	Data Bit 5	Data Bit 5	0	0	Reserved	Stick Parity	0	Transmitter Holding Register (THRE)	Data Set Ready (DSR)	Bit 5	Bit 5	Bit 13
6	Data Bit 6	Data Bit 6	0	FIFOs Enabled [2]	RCVR Trigger (LSB)	Set Break	0	Transmitter Empty (TEMT)	Ring Indicator (RI)	Bit 6	Bit 6	Bit 14

0x4 = character received

1 if the THR reg is free.

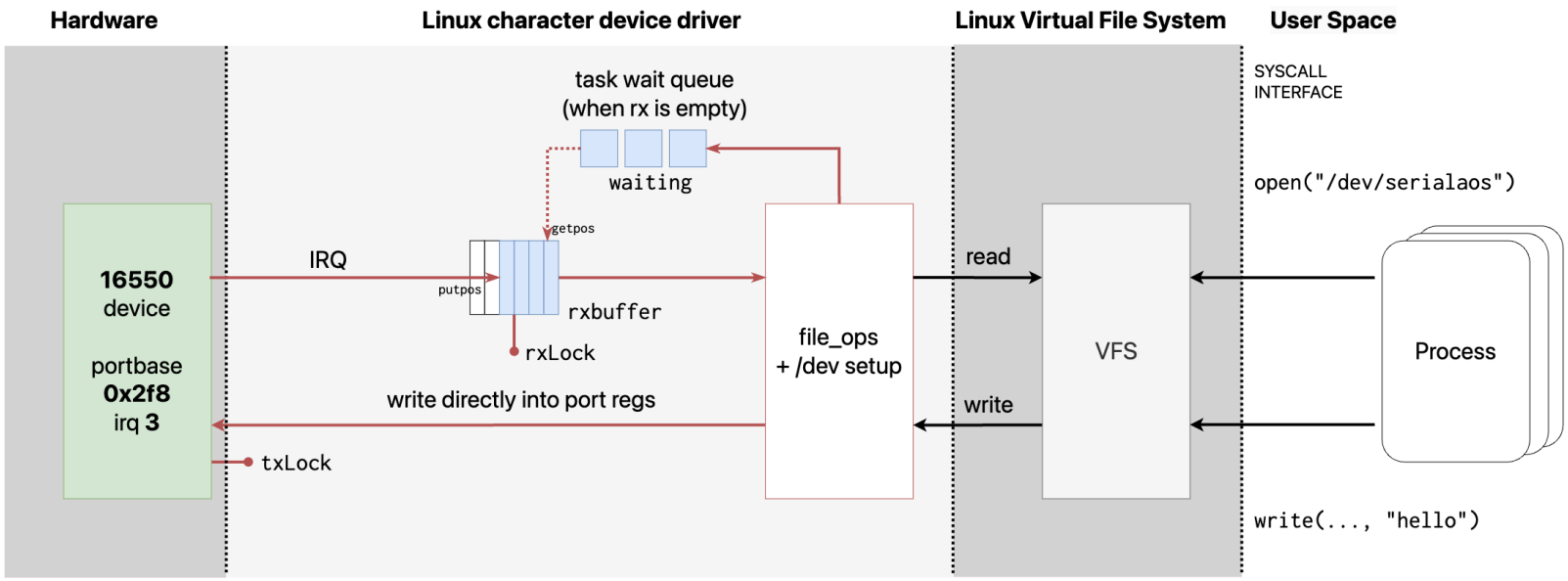
Registers used to interact with the UART

The 16550 UART device

```
#define PORT_BASE 0x2F8
#define PORT_SIZE 8
#define PORT_IRQ  3

//16550 registers, see datasheet
#define RBR PORT_BASE+0
#define THR PORT_BASE+0
#define IER PORT_BASE+1
#define IIR PORT_BASE+2
#define FCR PORT_BASE+2
#define LSR PORT_BASE+5
```

Driver's architecture



Serialaos driver as implemented during the lab, using only low-level character device primitives

Running the demo

```
make enter-container # this exports container's inner port 6000 over 8080
# In the container
/repo/stage/start-qemu-serial.sh
```

```
# In another shell
nc localhost 8080
```

Running the demo

Once booted into Linux:

```
insmod modules/lab-5-serialaos.ko
cat /proc/devices # We learn the major number for serialaos is 248
mknod -m 600 /dev/serialaos c 248 1
ls -l /dev
echo "testing" > /dev/serialaos # Should see the data coming out from the socket on the other shell
cat /dev/serialaos # Whet you type on the other shell should be printed
```

Perf graphs

We now have a way to get and put files from the vm. Let us get some perf data and try to plot it.

```
perf record -F 99 -ag -- sleep 2
perf script > perf.txt
cat perf.txt > /dev/serialaos


# On the host
nc localhost 8080 > perf.txt # wait a bit
```

Either load `perf.txt` it into speedscope or create an svg locally (with the scripts taken from Brendan Gregg's Flame Graphs):

```
# on the host
./scripts/sc.pl perf.txt > perf.folded
./scripts/fg.pl perf.folded > perf.svg
```

Either load it into

./stackcollapse-perf.pl out.perf > out.folded

Linkography

- https://linux-kernel-labs.github.io/refs/heads/master/labs/interrupts.html
- https://www.kernel.org/doc/html/v4.16/driver-api/basics.html
- https://www.kernel.org/doc/html/latest/locking/locktypes.html
- https://alexrhodes.io/blog/post/22/ (good tutorial, but code has some bugs)
- https://en.wikipedia.org/wiki/16550_UART
- https://pccomponents.com/datasheets/NATI-PC16550.pdf
- https://github.com/fedetft/miosix-kernel/blob/master/miosix/arch/common/drivers/serial_lpc2000.cpp
- PCI peripheral driver: https://github.com/cirosantilli/linux-kernel-module-cheat/blob/master/kernel_modules/qemu_edu.c
- Using DMA: https://gist.github.com/proywm/15b38a293770b0aaa99eaf1f03a2c9f7