Digital electronics

J.-M Friedt

FEMTO-ST/time & frequency department

jmfriedt@femto-st.fr

slides at `jmfriedt.free.fr`

February 8, 2025

Plan

7 lessons/lab sessions (4-hour long schedules):

1. Executive environments: principles and introduction, getting started with FreeRTOS
2. FreeRTOS, RTEMS, Nuttx ... multitasking and associated methods to make sure shared data and resources are kept in known states (mutex & semaphore)
3. Using the scheduler, mutex and semaphores to solve the "philosopher problem"
4. Embedded systems with GNU/Linux – POSIX compatible operating system
   Architecture of an operating system, kernel v.s userspace
   Internet connectivity and networking
5. Accessing hardware resources from userspace – memory translation from physical to virtual address space (Memory Management Unit) – `/dev/mem`
6. Accessing hardware resources from a web server – internet connected instrument
7. From userspace to kernel space: character device (*char device*) for communicating between users and the kernel

Background: FreeRTOS

- An executive environment makes you feel you are using an operating system (tasks with priorities and local stack ⇒ semaphores and mutex, queues)
- `github.com/jmfriedt/tp_freertos` provides some basic usage examples
- `common` provides basic functionalities such as hardware access as no driver is supported
- `0no_freertos` provides some basic baremetal C examples to validate that `common` behaves as expected
- `Makefile` assume that FreeRTOS [1] is located at the same level than `tp_freertos`

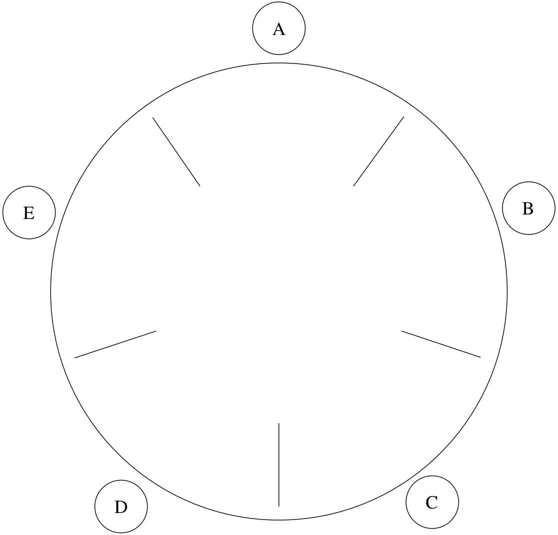
[1] `https://github.com/FreeRTOS/FreeRTOS-LTS`

Dining philosophers problem

Define **constaints** and leave it to the scheduler to find the solution after providing the transition rules between the state machine

- N philosophers are seated around a table,
- each philosopher has a chopstick at its left and another chopstick at its right,
- each philosopher must grab two chopsticks to eat,
- once a philosopher has eaten, it drops the chopsticks it has used.
- Two philosophers cannot use the same chopstick at the same time.

How can this problem be expressed in the FreeRTOS framework for its scheduler to find the solution?



Dining philosophers problem

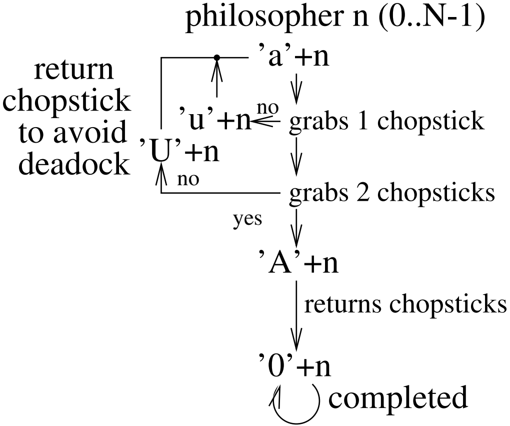
- how is a philosopher represented in the FreeRTOS framework?
- how is a chopstick represented in the FreeRTOS framework?
- how would you exit the case where all philosophers have selected to grab the chopstick at their right?

Dining philosophers problem

Solution example (atomic communication):

`eabdcEB4D1A3C02`

1. all philosophers wish to grab a chopstick : eabdc,
2. E grabs two chopsticks
3. B opposite to E grabs two chopsticks
4. 4 (E) finishes eating and returns both chopsticks
5. ... allowing D to grab both chopsticks.
6. 1 (B) has finished eating and drops both chopsticks
7. ... allowing A to grab both chopsticks
8. 3 (D) finishes eating and drops the chopsticks
9. C can now grab both chopsticks
0. 0 (A) finishes eating ...
1. ... and so does 2 (C)



Dining philosophers problem: emulator

```
$ qemu-system-arm -M stm32-p103 -serial stdio -kernel output/main.bin

eabcdEBung
wng
4D1A03C2
```

qemu supporting the STM32F1: `https://beckus.github.io/qemu_stm32/`

See `hw/arm`: `stm32_p103.c` platform

Adding a serial port:

```
DeviceState *uart3 = DEVICE(object_resolve_path("/machine/stm32/uart[3]", NULL));
assert(uart3);
stm32_uart_connect((Stm32Uart *)uart3, serial_hds[0], STM32_USART3_NO_REMAP);
```

Solution depending on scheduler settings

Different scheduling schemes depending on rules (timeout mutex = 500 ms):

- no delay (no one returns a chopstick, no delay between first and second mutex)
  `ebcdazEwBy4Dv1xA3C02`
- 200 ms delay (all philosophers have returned their chopsticks while only a single one is eating)
  `eabcdzvwxyqeDmanboc3zvwxCqemanb2zvwBqema1zvAqe0zE4`
- 500 ms delay (two philospohers at opposite positions on the table eat at the same time)
  `ebcdazwxyvqeDnbocA3zx0wEC42B1`
- 800 ms delay (same as 200 ms)
  `eabcdzvwxyqeDmanboc3zvwxCqemanb2zvwBqema1zvAqe0zE4`

Emulator: using an un-initialized peripheral

Message error from the emulator when an un-initialized is used:

```
USART_InitTypeDef USART_InitStructure;
...
USART_Init(USART2, &USART_InitStructure);
USART_Cmd(USART2, ENABLE);
put_char(USART1,'0');
put_char(USART2,'0');
```

We attempt using serial port 1 which was never initialized ⇒ neither clock source nor configuration

```
$ qemu_stm32/arm-softmmu/qemu-system-arm -M stm32-p103 -serial stdio -serial stdio -serial stdio -kernel main.bin

qemu stm32: hardware warning: Warning: You are attempting to use the UART1 peripheral while its clock is disabled.

R00=40013800 R01=00000030 R02=008e0001 R03=00000030
R04=20004fe8 R05=08000b1c R06=00000000 R07=20004fc0
R08=00000000 R09=00000000 R10=00000000 R11=00000000
R12=0000000f R13=20004fc0 R14=08000519 R15=08000808
PSR=20000173 --C- T svc32
qemu: hardware error: Attempted to write to USART_DR while UART was disabled.
CPU #0:
R00=40013800 R01=00000030 R02=008e0001 R03=00000030
R04=20004fe8 R05=08000b1c R06=00000000 R07=20004fc0
R08=00000000 R09=00000000 R10=00000000 R11=00000000
R12=0000000f R13=20004fc0 R14=08000519 R15=08000808
PSR=20000173 --C- T svc32
FPSCR: 00000000
Aborted
```

Access to shared resources

- Messages were so far atomic (single lettre) to avoid interferences between multiples messages sent by philosophers.
- `qemu-system-arm -M stm32-p103 -serial stdio -serial stdio -serial stdio -kernel output/main.bin`

```
LED Off
4 wants to eat
0 wants to eat
1 wants to eat
2 wants to eat
3 wants to eat
4 has grabbed one chopstick
0 has grabbed one chopstick
0 has grabbed two chopsticks
2 has grabbed one chopstick
3 has grabbed one chopstick
4 drops chopsticks
4 wants to eat
3 has grabbed two chopsticks
1 wants to eat
0 has eaten
2 drops chopsticks
2 wants to eat
3 has eaten
4 has grabbed one chopstick
4 has grabbed two chopsticks
1 has grabbed one chopstick
2 has grabbed one chopstick
2 has grabbed two chopsticks
4 has eaten
1 drops chopsticks
1 wants to eat
2 has eaten
1 has grabbed one chopstick
1 has grabbed two chopsticks
1 has eaten
```

- How many mutex are needed to synchronize access to the serial port?
- What are the consequences of removing the mutex?

Conclusion

1. an emulator allows for completing the job even without hardware platform
2. an emulator provides some hint at the internal state of the processor and prevents the user from making mistakes ...
3. ... assuming the peripheral is properly emulated.
   ADC[2], DAC, timer, GPIO, USART fonctionnal for STM32 **but** dependency on Python2 to compile which is obsolete.
   Eclipse includes an emulator for the STM32F4:
   `https://github.com/xpack-dev-tools/qemu-arm-xpack/` [3] (STM32F4-Discovery)
4. qemu can act as gdb server for probing embedded software execution state:

```
qemu-system-arm -M stm32-p103 -s -S -serial stdio -kernel main.bin  # -s: wait connection on port 1234
```

   and

```
gdb-multiarch main.elf
target remote localhost:1234
continue
```

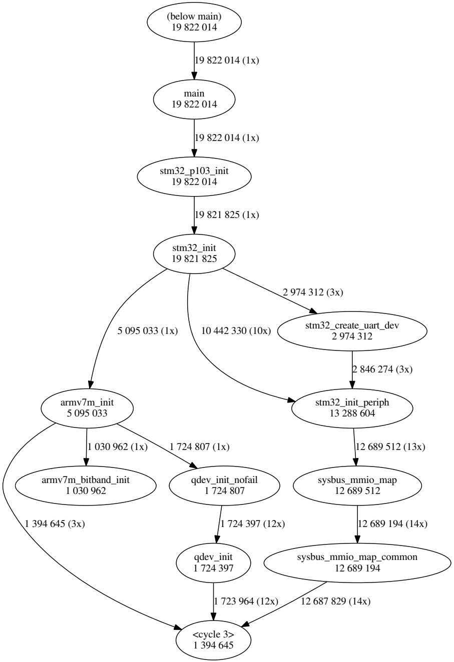
---

[2] error in ADC flag handing detected when using `libopencm3` which tests `ADC_CR2_SWSTART` bit of `ADC_CR2` to check that conversion has started: this bit should be set to 0 upon conversion start: `https://github.com/beckus/qemu_stm32/issues/24`, see `adc_start_conversion_regular()` in `libopencm3/lib/stm32/common/adc_common_v1.c` where `while (ADC_CR2(adc) & ADC_CR2_SWSTART);`

[3] formerly `http://gnuarmeclipse.github.io/qemu/`

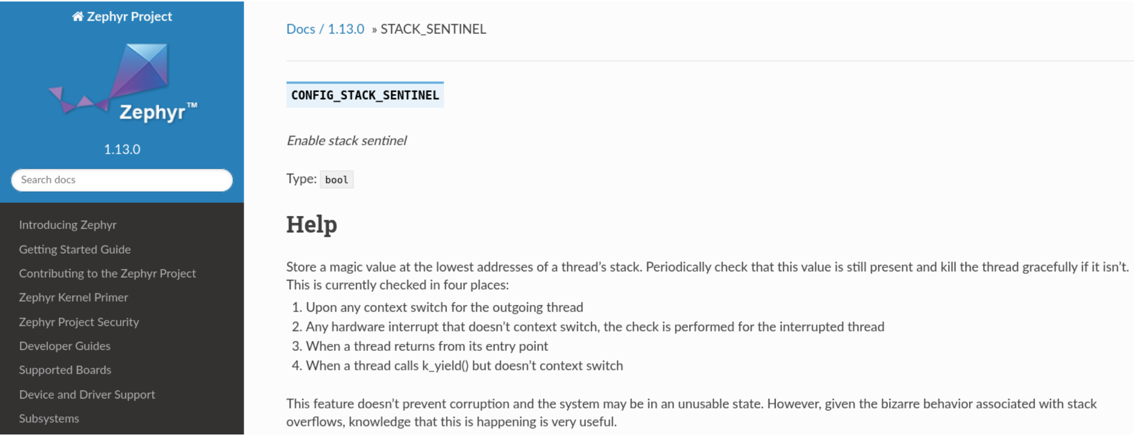
Analyzing the execution of an emulator

- qemu based on callback functions called when an event occurs requesting the emulation of a peripheral
- ⇒ sequence of executed functions hard to follow as they are not sequentially explicitly called
-
```
valgrind --tool=callgrind -v \
--dump-every-bb=10000000  \
../qemu-system-arm \
-M stm32-p103 \
-serial stdio  \
-kernel temperature/main.bin
```
- `kcachegrind callgrind.out.8964` displays a chart of the called functions and the associated resources.



Beyond FreeRTOS ...

Many more executive environments to explore and discover, with more created (and dying) every day.



↑ Example of stack debugging functionality provided by Zephyr Project.

Introduction to NuttX [in French]: G. Goavec-Merou, J.-M. Friedt, *Un environnement exécutif visant la compatibilité POSIX : NuttX pour contrôler un analyseur de réseau à base de STM32*, GNU/Linux Magazine France (Dec. 2017) at `http://jmfriedt.free.fr/lm_nuttx.pdf`

Installing Zephyr-OS

```
sudo apt install gcc-arm-none-eabi python3-pyelftools device-tree-compiler
pip install west --break-system-packages
west init
cd zephyr/
west update
source zephyr-env.sh
# export ZEPHYR_TOOLCHAIN_VARIANT=cross-compile
# export CROSS_COMPILE=/usr/bin/arm-none-eabi-
cd samples/basic/blinky/
mkdir build
cd build
cmake  -DBOARD=olimexino_stm32 ../
```

outputs `zephyr/zephyr.bin` to be flashed and executed on the microcontroller (STM32F103 [4]) ...
or on qemu:

```
~/zephyr/samples/basic/minimal/build
cmake -DBOARD=qemu_cortex_m3 ../
make -j12
make run
```

[4] On the STM32F100: if unable to reprogram the microcontroller after running Zephyr, connect Boot pin to 3V3.

Hardware description in Zephyr-OS: devicetree

Configuration files for the STM32VL-Discovery STM32F100 board:

- List of supported drivers, hardware configuration (RAM) and toolchain: `https://github.com/zephyrproject-rtos/zephyr/blob/main/boards/st/stm32vl_disco/stm32vl_disco.yaml`
- Compilation options: `https://github.com/zephyrproject-rtos/zephyr/blob/main/boards/st/stm32vl_disco/stm32vl_disco_defconfig`
- Hardware description through the devicetree: `https://github.com/zephyrproject-rtos/zephyr/blob/main/boards/st/stm32vl_disco/stm32vl_disco.dts`
- Driver configuration through the devicetree nodes:

```
/dts-v1/;
#include <st/f1/stm32f100Xb.dtsi>
#include <st/f1/stm32f100r(8-b)tx-pinctrl.dtsi>
#include <zephyr/dt-bindings/input/input-event-codes.h>
/ {model = "STMicroelectronics STM32VLDISCOVERY board";
   compatible = "st,stm32vldiscovery";
...
   leds {
      compatible = "gpio-leds";                            /* DRIVER NAME       */
      green_led:ld3 {gpios = <&gpioc 9 GPIO_ACTIVE_HIGH>; /* DRIVER PARAMETERS */
                     label = "User LD3";
                     };
      blue_led: ld4 {gpios = <&gpioc 8 GPIO_ACTIVE_HIGH>;
                     label = "User LD4";
                    };
      };
...
```