RTOS Lab WS 2022

Christoph Scholl

University Freiburg

Hard- and Software

- Development Kit with ESP32 Microprocessor
  - Low cost option (< 10 Euro)
  - Compatible with Arduino-IDE or ESP-IF IDE
  - We will use the ESP-IF IDE

- Real-Time Operating System: FreeRTOS
  - Open source RTOS
  - Written in C
  - Simple fixed priority scheduling
  - https://www.freertos.org/

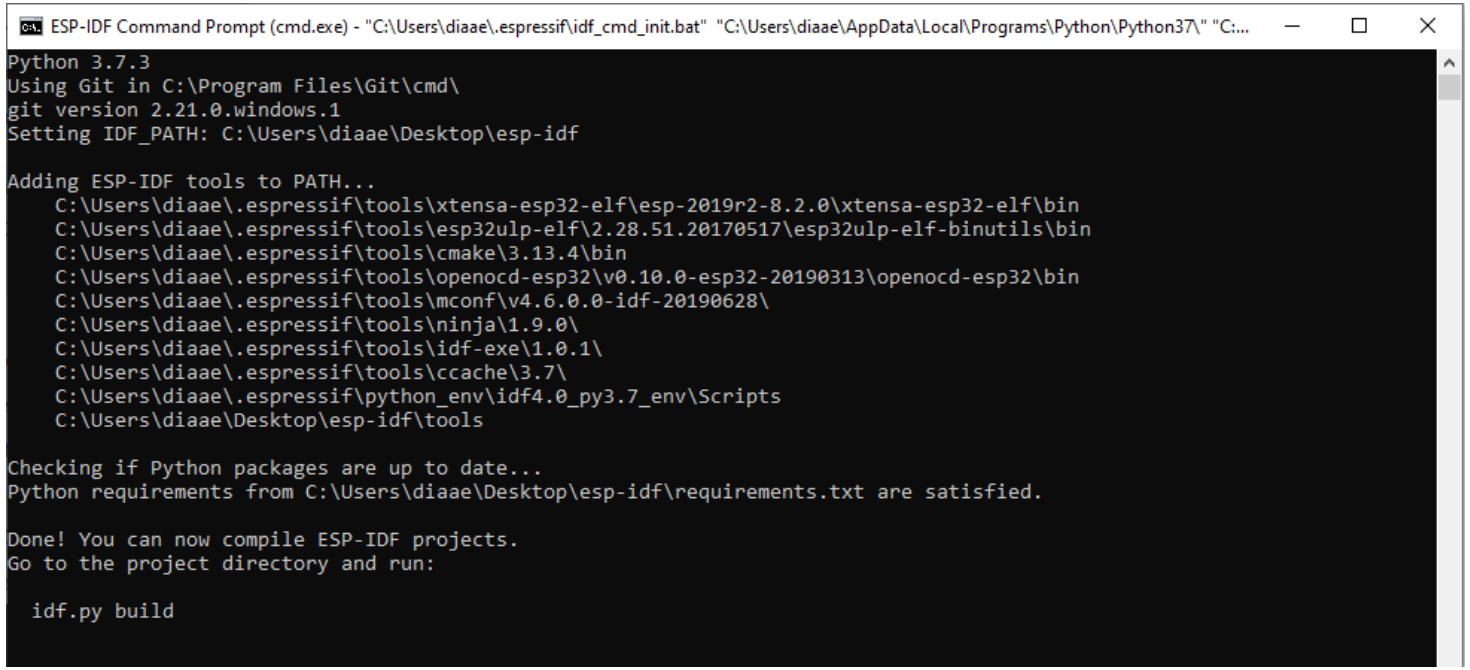
Task 1 – Getting started

- Get freeRTOS running on development board
  - First install the ESP32 Framework:
    - Follow the setup instructions on the following webpage according to your operating system: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html
    - A folder named esp-idf will be in the directory of your choice. The following icon will be created on your Desktop

Task 1 – Getting started

- …

- After having successfully installed the ESP32 Framework, running the ESP-IDF Command Prompt (cmd.exe) should result in the following window.



Task 1 – Getting started

- ESP-IDF comes with a FreeRTOS version called ESP-IDF FreeRTOS which is based on Vanilla FreeRTOS v10.4.3
- As a first application, start a few a-periodic tasks and use the internal fixed priority scheduler of freeRTOS to run them
- For doing so, build on the sample installation of freeRTOS in ESP-IDF:
  - Start the ESP-IDF cmd
  - Navigate to directory esp-idf\examples\system\freertos\
  - For building your own application you may first make a backup copy of the existing folder “real_time_stats”
  - Change into directory real_time_stats →main and do modifications (e.g. modify main.c, add new files etc.)
- Run your freeRTOS application:
  - Navigate to directory esp-idf\examples\system\freertos\
  - idf.py build
  - idf.py -p COM4 flash (COM4 is the serial port to which your ESP32 is connected)
  - idf.py -p COM4 monitor > **outputfile**
  - Check the output by looking at file **outputfile** in a text editor

Task 1 – Getting started

- For more information on ESP-IDF FreeRTOS see also https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos.html
- For more information on „vanilla“ freeRTOS in general see https://www.freertos.org/
- freeRTOS tutorial: https://www.freertos.org/Documentation/RTOS_book.html

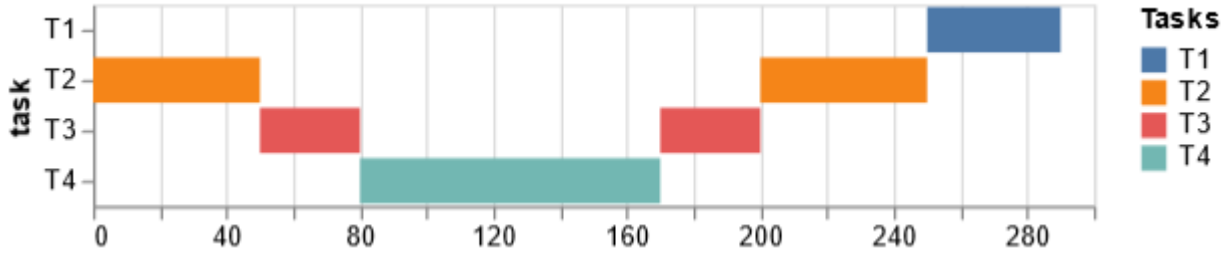
- Note:
  - ESP-IDF FreeRTOS is modified for making use of the dual core on ESP32
  - The SMP (symmetric multiprocessing) option is not needed in this lab
  - You should possibly use the option CONFIG_FREERTOS_UNICORE to run ESP-IDF FreeRTOS only on CPU0.

Task 2 - Visualization

- Visualize the schedule produced by the RTOS
- For logging interesting scheduling events, make use of Trace Hook Macros
  - See https://www.freertos.org/rtos-trace-macros.html
  - Be careful: Code inside trace macros should be as fast as possible to avoid overhead during scheduling actions
  - No printf’s in trace macros!!!
  - Solution:
    - Inside trace macros write information into memory arrays, without any syscalls etc.
    - Define a suitable file format and print recorded information with printf **after** monitored schedule has been finished

Task 2 - Visualization

- Display logged information graphically
- Interpret the file by a suitable visualization software, see e.g. the python library altair
  - https://altair-viz.github.io/getting_started/installation.html
- I.e. write a small python script that interprets the log file and produces an output like this (e.g.)



Task 3 – Task modeling for a-periodic and periodic tasks

- freeRTOS does not contain a build-in mechanism to directly generate periodic tasks
- Function **vTaskDelayUntil** can be used to model periodic tasks
  - See https://www.freertos.org/vtaskdelayuntil.html
  - Basic example:

```
// Perform an action every 10 ticks.
void vTaskFunction( void * pvParameters )
{
TickType_t xLastWakeTime;
const TickType_t xFrequency = 10;

    // Initialise the xLastWakeTime variable with the current time.
    xLastWakeTime = xTaskGetTickCount();

    for( ;; )
    {
        // Wait for the next cycle.
        vTaskDelayUntil( &xLastWakeTime, xFrequency );

        // Perform action here.
    }
}
```

  - Task will be blocked by vTaskDelayUntil and unblocked at time xLastWakeTime + xFrequeny

Task 3 – Task modeling for a-periodic and periodic tasks

- Two options:
  - Option 1 (should be followed first):
    - Use for each periodic task a „generator task“ running at maximal priority, i.e. a wrapper start_periodic_task(tau_i, T, P) with period T, phase P and function pointer tau_i
    - The periodic action of the generator task is to generate a new task (instance) with xTaskCreate which then gets ist own priority
  - Option 2 (less overhead, but does not generate real „task instances“):
    - Model all periodic task instances by **one** task
    - The periodic action is then just a call to function pointer tau_i
    - After the call the task is blocked by vTaskDelayUntil until „arrival time of next task instance“…
- Provide a wrapper to start a-periodic tasks as well
- Add code to check for time overflows …

Task 4 – Integrate EDF into freeRTOS

- Follow a simple user-space approach of integrating EDF into freeRTOS (without modifying the OS code)
- Two options:
  - Option 1:
    - Use trace macros which re-compute task priorities according to deadlines whenever a task (instance) is generated or finished
    - Then just use the integrated priority based scheduler
  - Option 2:
    - Use trace macros to manage own ready, blocked, suspended lists in the user space
    - Only two priorities for periodic tasks – the task instance that should be executed next gets maximal priority
    - This approach is followed in **ESFree** developed by Robin Kase in his Master‘s thesis at KTH Stockholm
      - See http://kth.diva-portal.org/smash/record.jsf?pid=diva2%3A1085303&dswid=436
      - https://github.com/RobinK2/ESFree
- Deep integration of EDF into freeRTOS OS code - not followed in this lab

Possible Extensions

- Integrate Total Bandwidth Server to schedule periodic and a-periodic tasks together
- Look into priority inheritance (implemented in freeRTOS for mutexes, but not for semaphores)
- Integrate priority ceiling protocol

Additional Tasks

- Always document what you did
  - Among others for later usage in exercises of RTOS lectures
- Lab should also result in something like a small „user manual“ for setting up the system