kay.heider [☺] tu-dortmund.de

Exercise for lecture Real-Time Operating Systems Design and Implementation
Winter semester 2025/2026

Project: Build Your Own Debug Tool

Thursday, November 13, 2025

To successfully complete this project, you need to achive at least 25 points out of the total 50 points. Upload your completed project here until 17.12.2025. You have to work on this project in groups of 2–4 people. When uploading the project, provide the names and matriculation numbers of all group members. Your projects also has to include a README file that describes how to use it and which modifications are made to FreeRTOS. You can choose to upload a zip that contains a working copy of your debug tool or just the files along with a description how to integrate them into the FreeRTOS source code.
Additionally, your group must present the debugging tool in the exercise session on 18.12.2025. All group members must take part in the presentation. If your group cannot attend the exercise session, please inform us in advance with a reason. Your group will need to record a video presentation of your debugging tool and upload it until 18.12.2025. Questions regarding your tool will be asked in a later session.

Project Introduction

In the last exercise, you have gotten familiar with the basics of FreeRTOS trace macros. For this project, you will build your own debugging tool based on these trace macros. Your debugging tool should collect various runtime information about the queue operations, housekeeping operations, and job executions. Additionally, you should implement a way to extract and display this information after the program has finished or a certain time span has elapsed. Finally, you should design a way to visualize the collected trace information, e.g., by plotting the job executions as schedule diagrams (like the EDF schedule on slide 3 of the lecture slides for "Periodicity and Timer").

1 Queue Tracing (10 points)

Your debug tool should trace important queue operations during runtime that have an effect on the behavior of the job executions. For this, FreeRTOS lets you trace queue operations with the `traceQUEUE_X` macros.

1.1 Implementation of macros (4 points)

Implement the following trace macros to log queue operations:

- `traceQUEUE_RECEIVE`
- `traceQUEUE_RECEIVE_FAILED`
- `traceQUEUE_RECEIVE_FROM_ISR`
- `traceQUEUE_RECEIVE_FROM_ISR_FAILED`
- `traceQUEUE_SEND`
- `traceQUEUE_SEND_FAILED`
- `traceQUEUE_SEND_FROM_ISR`
- `traceQUEUE_SEND_FROM_ISR_FAILED`

1.2 Log Entries (6 points)

Each log entry should include the current tick count, a time stamp (more fine-grained than ticks), the queue handle, the number of ticks that the job will wait for, and an identifier of the task that calls the function. Implement a way to store the log messages in a buffer during runtime and print them on the serial console once the program has finished or a certain time span has elapsed.

2 Housekeeping Operations Tracing (10 points)

An RTOS needs to perform several housekeeping operations to manage tasks and scheduling. Your debug tool has to trace these operations. This includes the increment of the system tick, the creation and deletion of tasks, and the task delay functions. All log entries have to include the current tick count, a time stamp (more fine-grained than ticks), and an identifier of what is being logged (e.g., `traceTASK_INCREMENT_TICK`). Again, store the log messages in a buffer during runtime and print them on the serial console once the program has finished or a certain time span has elapsed.

2.1 Tick Increment Macro (2 points)

Implement the `traceTASK_INCREMENT_TICK` macro. The log entry must also include the new tick count after the increment.

2.2 Task Creation and Deletion Macros (4 points)

Implement the following trace macros:

- `traceTASK_CREATE`
- `traceTASK_CREATE_FAILED`
- `traceTASK_DELETE`

The log entries for tracing task creation and deletion must include an identifier of the created/deleted task.

2.3 Task Delay Macros (4 points)

Implement the following trace macros:

- `traceTASK_DELAY`
- `traceTASK_DELAY_UNTIL`

The log entries for tracing task delays must include an identifier of the task and the number of ticks that the task will delay for.

3 Job Execution Tracing (10 points)

In order to analyze the timing behavior of your program, your debug tool has to trace the execution of every job. That means, whenever a job is selected for execution, and whenever a job finishes its execution, a log entry must be created. The resulting trace should then contain enough information to reconstruct the schedule of all jobs.
To trace the job executions, you need to implement the following trace macros:

- `traceTASK_SWITCHED_IN`
- `traceTASK_SWITCHED_OUT`

Each log entry must include the current tick count, a time stamp (more fine-grained than ticks), an identifier of the task that is being switched in or out and which event is being traced. Again, store the log messages in a buffer during runtime and print them on the serial console once the program has finished or a certain time span has elapsed.

4 Trace Information Extraction (10 points)

With all the trace macros implemented, your debug tool should now be able to collect a comprehensive set of runtime information that includes tracing of queue events, housekeeping operations, and the schedule. At the moment all log entries are printed on the serial console, however to post-process your trace data, you need to extract it in a structured way.

4.1 Extracting the Trace Data (6 points)

Your tool has to include a way to automatically extract the trace data from the serial console without manual intervention. This could be done, for example, by reading the serial console output with a Python script line-by-line and processing the log entries based on the type of event being traced. In this first step, your tool has to provide some statistics about the collected trace data, e.g., the total number of log entries, the number of log entries per event type, and the time span covered by the trace data.

4.2 Export in a Structured Format (4 points)

After you have extracted the trace data from the serial console, your tool should export the data in a CSV format. Each row in the CSV file should represent a log entry, and each column should contain the specific attributes of the log entry (e.g., tick count, time stamp, event type, task identifier, etc.). Make sure that all entries are exported and properly formatted.

5 Visualization (10 points)

Based on the previously extracted trace data, your debugging tool should provide a way to visualize the collected information.

5.1 Basic Schedule Diagram Visualization (5 points)

In the moodle course, a Python script will be provided that takes the CSV file generated in the previous step as input and generates a basic schedule diagram from the log entries created by the `traceTASK_SWITCHED_IN` and `traceTASK_SWITCHED_OUT` macros. Therefore, the rows in the CSV file that will be used for this script will distinguish the following columns: event type (`traceTASK_SWITCHED_IN` or `traceTASK_SWITCHED_OUT`), tick count, time stamp, and task identifier. Make sure you format the CSV file properly, especially the logged event type has to match `traceTASK_SWITCHED_IN` or `traceTASK_SWITCHED_OUT`.

5.2 Visualizing other trace points (5 points)

Extend the provided Python script to visualize other trace points from your collected trace data. For example, you could add annotations at the corresponding time points that visualize task creation and deletion events or system tick increments. You can also visualize queue operations by showing when jobs are waiting for queue operations to complete and thereby are switched out.