# Benchmarking Real Time Operating Systems

Guillaume Champagne
Michel Dagenais

May 6, 2019

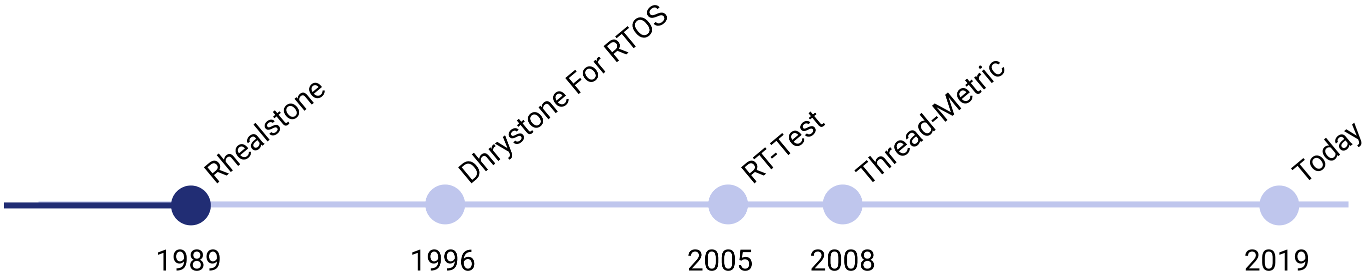
# Benchmarking Real Time Operating Systems

- How to choose the right RTOS?
  - Wide range of products available (Wikipedia lists 160 active projects).
  - Vendors may or may not provide metrics for your target system.

- Why benchmarking?
  - Embedded systems have limited resources.
  - Overhead of the chosen RTOS can impact your design.

- Let’s look at the freely available benchmarks.

# Previous RTOS Benchmarks

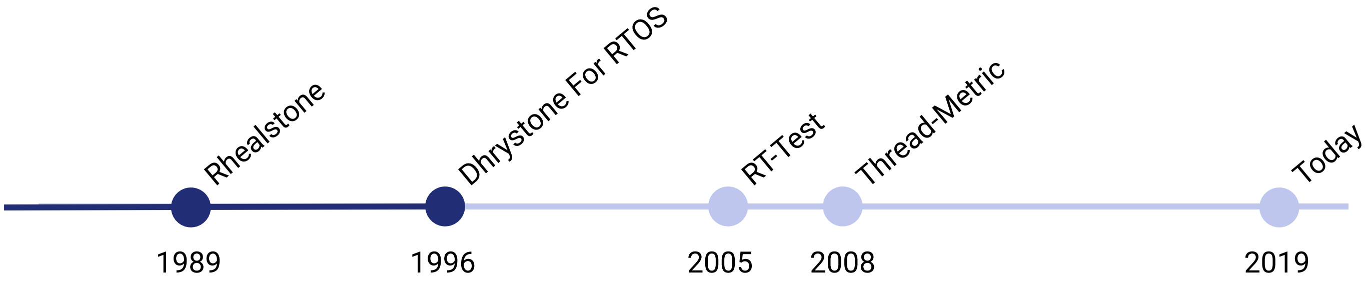


- The Rhealstone benchmark is comprised of 6 test scenarios. [1]

1. Task switching time
2. Task preemption time
3. Interrupt latency time
4. Semaphore shuffling time
5. Deadlock breaking time
6. Intertask message latency

$$RhealStone\ Performance\ Number = \frac{t_1+t_2+t_3+t_4+t_5+t_6}{6}$$

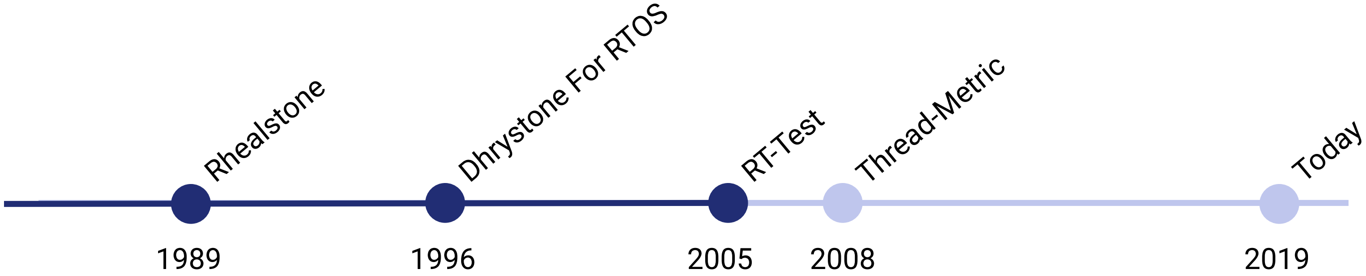
# Previous RTOS Benchmarks



- This benchmark uses a Dhrystone loop as the workload and make uses of RTOS services in 6 scenarios to estimate their overhead. [2]
- Results are expressed in *Dhrystone per seconds.*

1. Round Robin
2. Task Priority Preemption
3. Semaphore
4. Memory alloc/dealloc
5. Interrupt Latency
6. Message passing

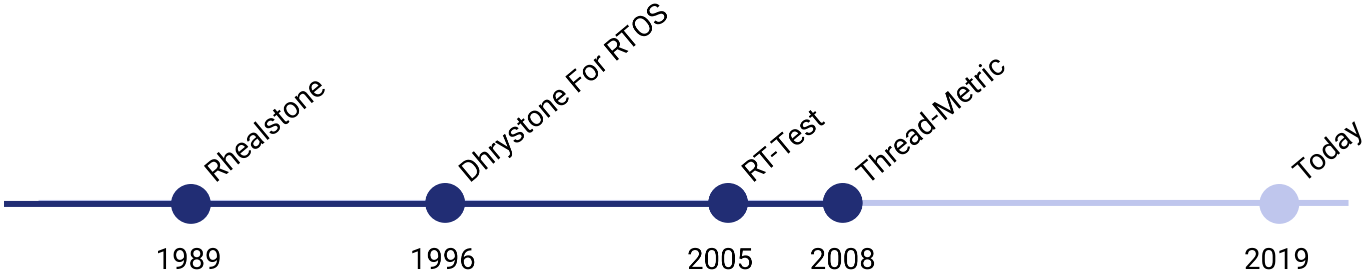
# Previous RTOS Benchmarks



- RT-Test aims measures real time performance of a Linux system. [3]
- *Cyclictest* is its most known test to estimate system latency.
- Results are time measurements.

1. Message Queue latency
2. Semaphore latency
3. Mutex latency
4. Signal Latency
5. Signal round trip
6. Cyclic test

# Previous RTOS Benchmarks

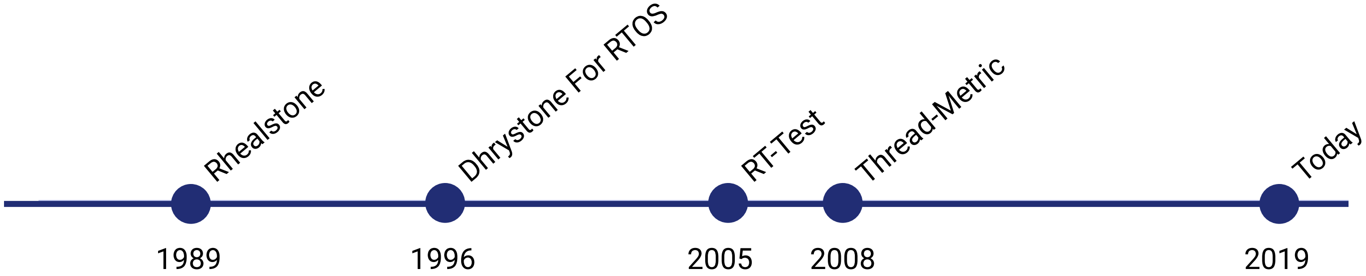


- Published by Express Logic, the developers of ThreadX. [4]
- Express Logic offers an easily portable reference implementation.
- Results are computed using a loop counter.

1. Cooperative context switch
2. Preemptive context switch
3. Interrupt processing

4. Message passing
5. Semaphore processing
6. Memory alloc/dealloc

# Previous RTOS Benchmarks



- The previous benchmarks are either:
  - Tedious to port to a new RTOS.
  - Inaccurate.
- We propose a new benchmark that is more accurate and easy to port.

# A Modern Benchmark Proposal

- Covers the most common RTOS services
  - Semaphores
  - Mutex
  - Event Flags
  - Message Queues
  - Cooperative Scheduling
  - Preemptive Scheduling
- Offers a reference implementation written with portability in mind.
  - Executing on a new RTOS only requires writing a thin porting layer.
- Produces accurate timing results.
- Results are either:
  - Printed on the standard output
  - Computed in Trace Compass and synchronized with an OS trace.

# How Scenarios Are Built

- Examples: Cooperative scheduling and Semaphore scenarios.
  - Get precise measurements rather than averages.

Figure 1. Cooperative scheduling scenario

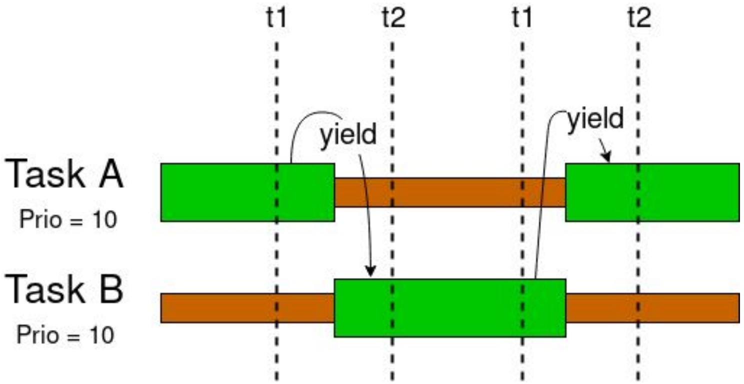
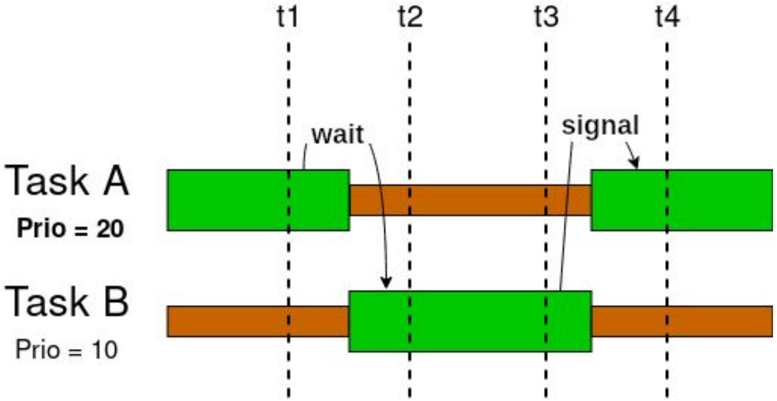


Figure 2. Semaphore wait-block and signal-wakeup scenario



# Benchmark Setup

- Benchmark executed on a 32 bit RPI 2B+ (MPCore Cortex A-7 900MHz).
  - FreeRTOS v10.1
  - uCOS-III
  - RTEMS v4.11
  - Linux 4.14.98-v7+
- L1 & L2 cache enabled, 1 to 1 virtual to physical address mapping (except Linux).
- 1kHz tick rate (1 ms period).
- Measurements are obtained through the Cycle Count Register.

# Semaphore

- Measure the overhead of this service in different scenarios.
  - Signal with empty wait queue. (Signal)
  - Wait on available semaphore. (Wait)
  - Signal causing context switch. (Signal-unblock)
  - Wait on semaphore causing context switch. (Wait-block)

Diagram 1.  Average Cycles for Semaphore

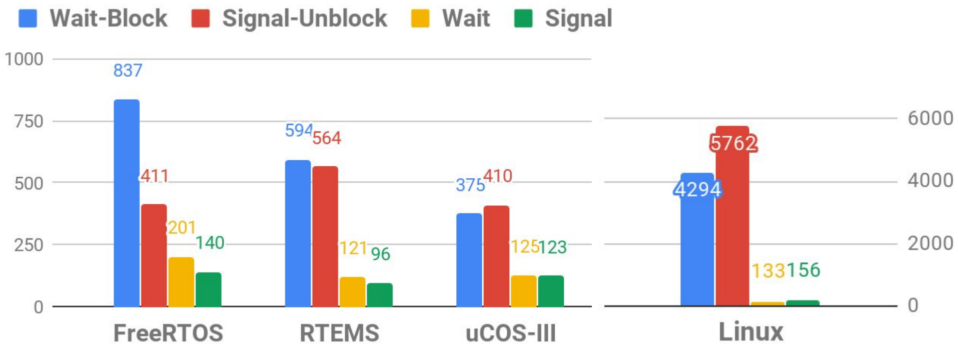
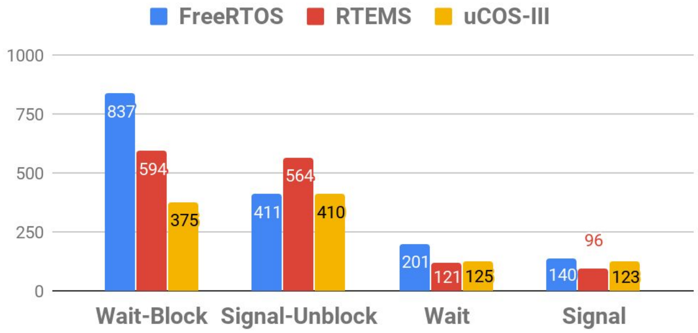


Diagram 2. Average Cycles for Semaphore



# Understanding the Results

- FreeRTOS takes twice as much cycles to take and block on a semaphore than to signal and switch to a new task.
  - Other OS do not exhibit the same behavior.
- Tracing the execution can provide insight.

Diagram 1. Average Cycles for Semaphore

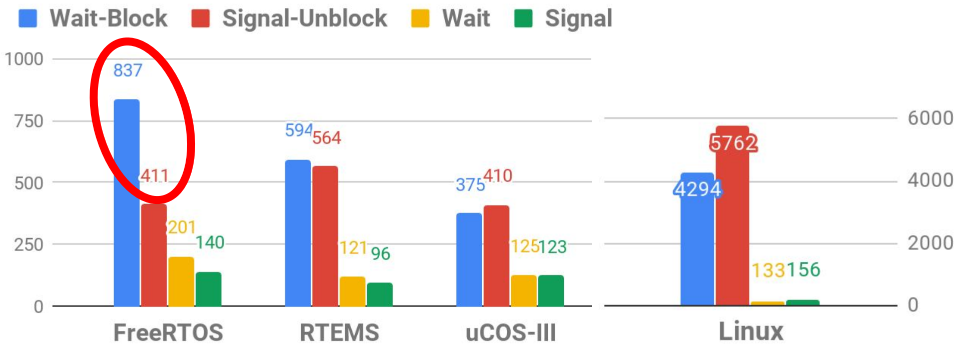
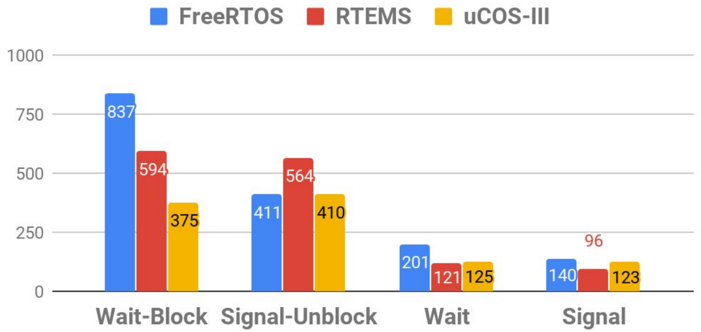


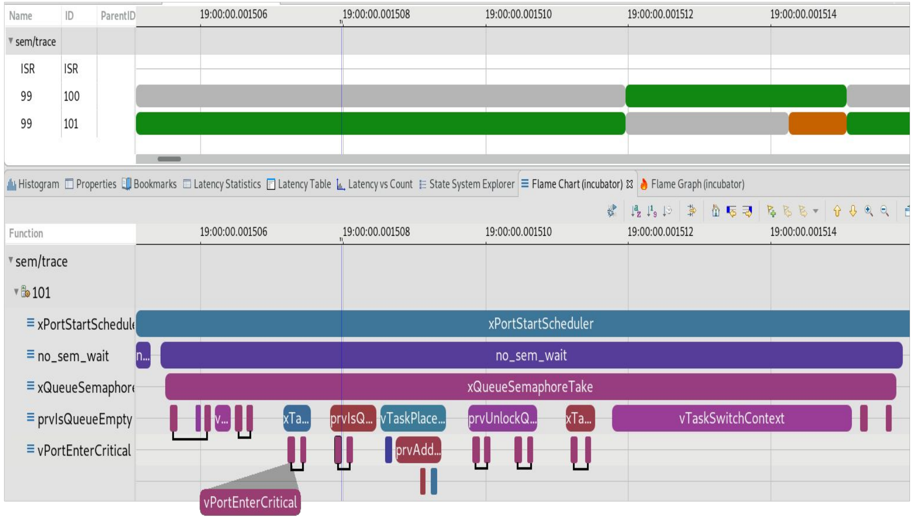
Diagram 2. Average Cycles for Semaphore



# Understanding the Results

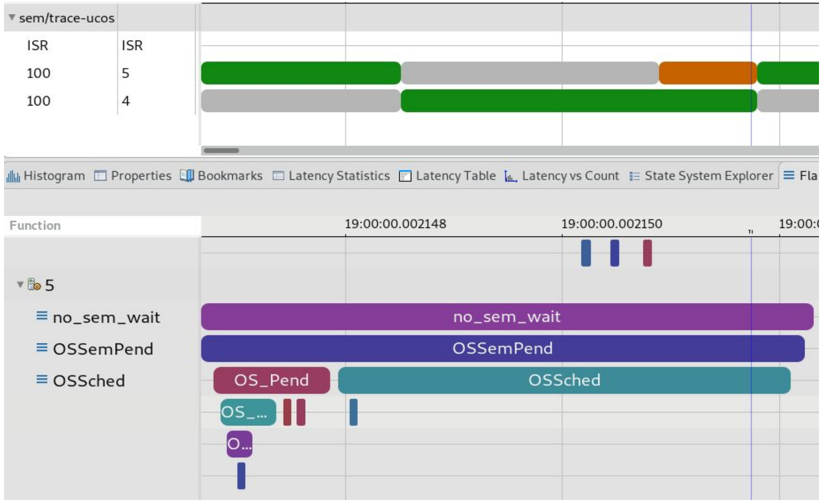
- Comparing the execution of FreeRTOS and uCOS-III.

Figure 3. Trace of FreeRTOS for the semaphore benchmark



- 14 function calls to enter & exit critical sections.

Figure 4. Trace of uCOS-III for the semaphore benchmark

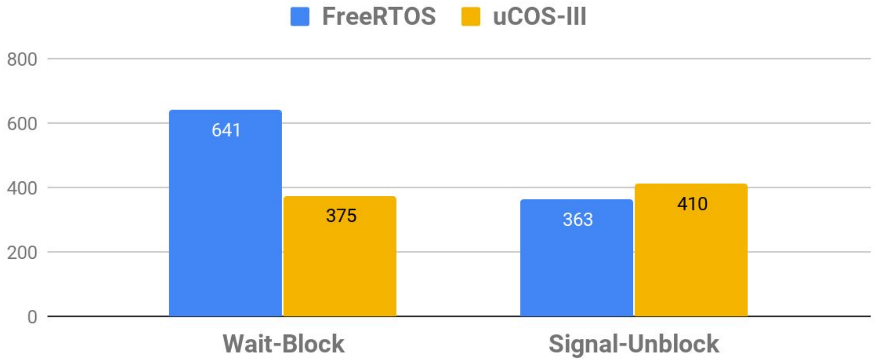


- Entry and exit of critical sections are inlined.
- OS_Pend is the whole critical section.

# Understanding the Results

- Results when inlining entry and exit of critical sections in FreeRTOS.
  - Code size increases by ~2%.
- Trade-off between length of sections with interrupt disabled and execution speed.

Diagram 3. Average Cycles for Semaphore Benchmark



# Message Queue

- Scenarios are analogous to the semaphore. Tested with 4 bytes messages.
  - ○ Send with empty wait queue. (send)
  - ○ Receive with 1 message in queue. (Receive)
  - ○ Send causing context switch. (Send-unblock)
  - ○ Receive causing a context switch. (Receive-block)

Diagram 4.  Average Cycles for MQ

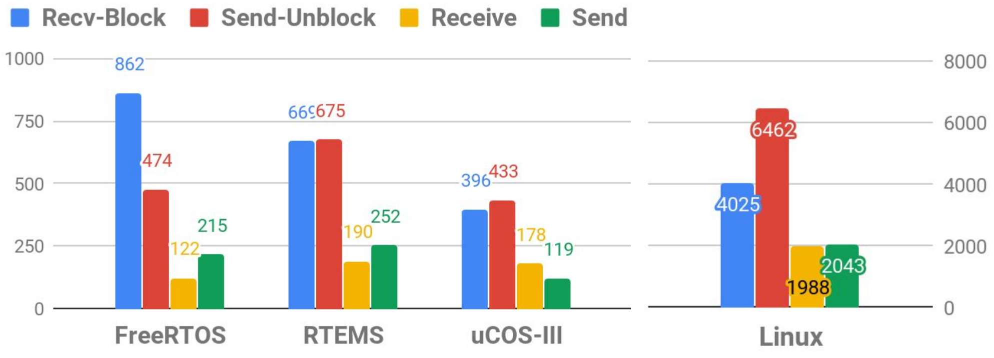
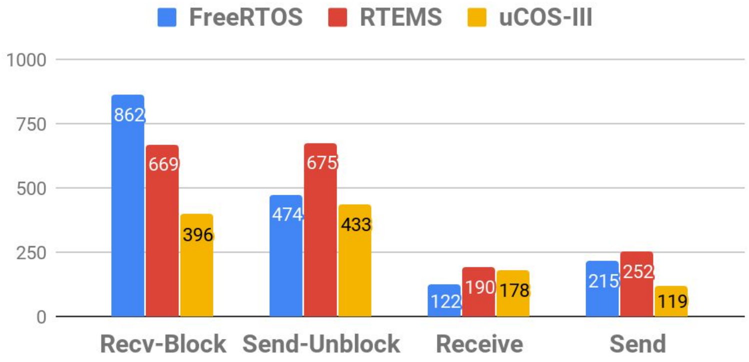
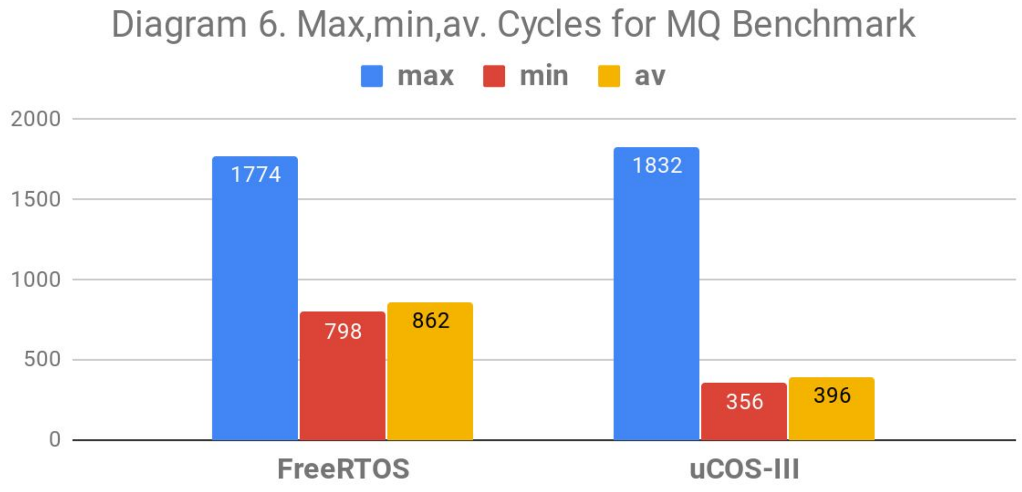


Diagram 5. Average Cycles for MQ Benchmark



# Message Queue

- Results are coherent with the semaphore results.
- Let’s look at maximum, minimum and average times for FreeRTOS and uCOS-III.



# Understanding the Results

- uCOS-III maximum time is 4.5x higher than the average time.
- FreeRTOS maximum time is 2x higher than the average time.
- uCOS-III schedules a Tick Task in the tick interrupt handler.

Figure 5. Trace of FreeRTOS for the MQ benchmark

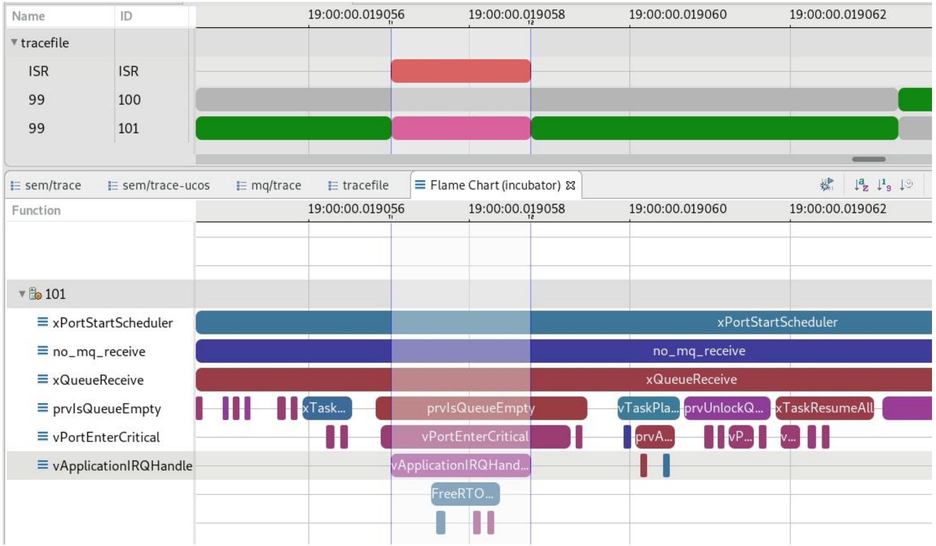
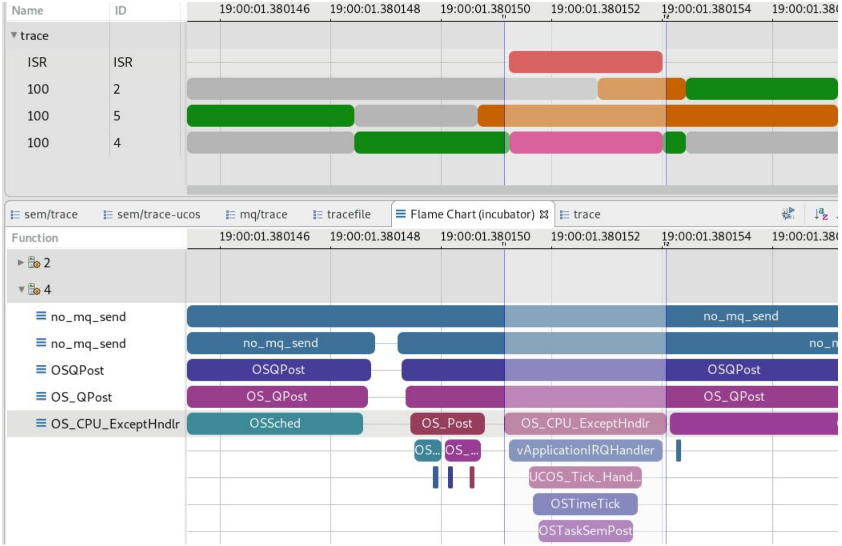


Figure 6. Trace of uCOS-III for the MQ benchmark



# Conclusion

- Benchmarking helps to understand the behavior of your RTOS.
  - Unexpected average execution time.
  - Unexpected worst case time.
- The port for the RTOS might have room for improvement.
  - Inlining entry/exits to critical sections.
- Knowing the available configuration options is the key to getting the best performance possible.
  - Background tasks priority.
- Benchmarking helps you make an informed design choice.

# Thank you!

The benchmark reference implementation will be available shortly on GitHub.

# References

1. Kar, R. P., & Porter, K. (1989). Rhealstone-a real-time benchmarking proposal. *Dr Dobb’s Journal*, *14*(2), 14.
2. McRae, E. (1996). Benchmarking real-time operating systems. *Dr Dobb's Journal*, *21*(5), 48-59.
3. The Linux Foundation. RT-Tests. Tiré de https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/rt-tests .
4. Lamie, W., & Carbone, J. Measure your RTOS’s real-time performance. Tiré de https://www.embedded.com/design/operating-systems/4007081/Measure-your-RTOS-s-real-time-performance