uKernel Micro-Kernel

SETR 2021/22

Davide Castro — up201806512
Henrique Ribeiro — up201806529
João Costa — up201806560

Architecture

Section 1

Tasks

Each task is characterized by:

- **Period**;
- **Deadline**;
- Initial **delay**;
- Task's **code** + optional **argument**;
- **Stack**.

The task's code Follows **POSIX thread** structure:

- Only called once;
- (optional) Take a **void pointer** as **argument**;
- (optional) Initial setup section;
- Followed by a loop containing the code to execute.

Task stack

- Each task has its own stack given by the kernel:
  - Determining the stack’s size is the responsibility of the developer.
- (optional) **Stack canaries** are available:
  - Detect problems related to the size of stack;
  - Can ease the process of finding the optimal stack size.
- After every task’s yield process, stack canaries are checked:
  - Upon detecting a problem, the kernel enters a failure state.

Failure State and Asserts

- Only occurs when the **“assert” functionality** is active;
- Performs **health-checks on multiple critical conditions** such as:
  - Failure of memory allocations;
  - Out-of-bound vector accesses;
  - Alignment of stack addresses.
- **Failure state is entered when an assertion fails**:
  - Sending a **message detailing the error** (message, line, function, and file);
  - Making the built-in LEDs continuously spell SOS in Morse code.
- Developer can add their own checks in the code of the tasks.

Scheduler

- Schedules tasks using the Earliest Deadline First (**EDF**) algorithm;
- To avoid overflow on the deadlines once P ticks have passed (in a 16-bit representation, **P=2^{16}=65536**), the group followed the approach described by **Giorgio Buttazzo** et al. in 'Efficient EDF Implementation for Small Embedded Systems':
    - With this, the deadlines' **time are represented cyclically**;
    - **Assumes two deadlines aren't more than P/2 time units apart**.

COMPARING DEADLINES WITH THE ICTOH ALGORITHM (OPTIMIZATION)

Comparison	Expression	Meaning
$t(e_i) > t(e_j)$	$(e_i - e_j) > 0$	e_i is later than e_j
$t(e_i) < t(e_j)$	$(e_i - e_j) < 0$	e_i is earlier than e_j
$t(e_i) = t(e_j)$	$(e_i - e_j) = 0$	e_i and e_j are at the same time

Dispatcher

- Fetch the first element of the (sorted) tasks list;
- If the task is **READY**, and its priority is higher priority, the current task is preempted, and the new task starts executing;
- When a task yields, the dispatcher passes the execution to the next highest priority task, thus maintaining maximum uptime;
- When **no task is READY** to execute, the dispatcher executes the **“idle task”**.

Resource Sharing and Mutexes

- These mutexes use the Priority Inheritance Protocol (**PIP**) to bound the duration of the periods of priority inversion.
- Each task contains a map that maps each **mutex** to the highest priority of the task the **mutex** is currently blocking.
- When locking/unlocking a **mutex**, **preemption at the kernel level is disabled**.

Development Environment

Section 2

Development Environment

- Had to work on both Windows and Linux;
- To achieve this, the group used and modified the system from the ***Arduino-Makefile* project**:
    - Adding new targets for debugging;
    - Support for defined conditional compilation regions;
    - Changing some defaults;
    - Improve code files directory structure.
- **SimAVR** was a vital part during development since it enabled the use of the GNU Project Debugger (GDB);

Evaluation of Results

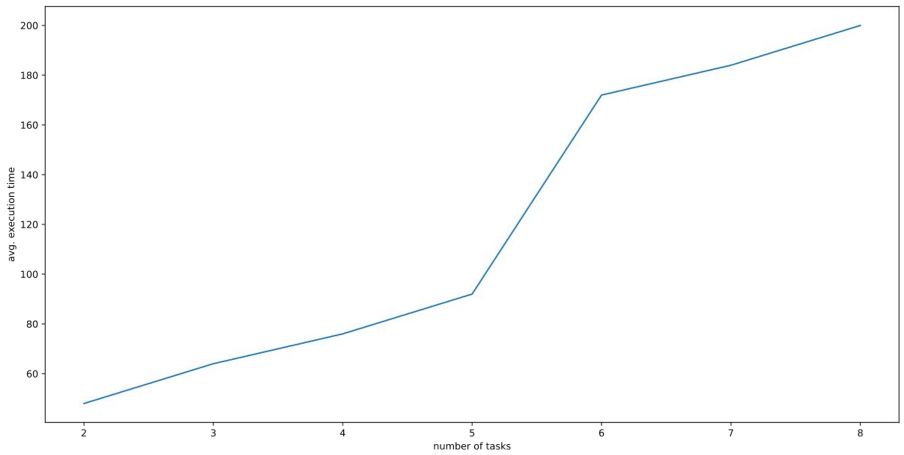
Section 3

Memory Footprint

- Task Control Block (**TCB**):
  - Static memory related to the task = **28 bytes**;
  - Task's stack = **defined by the developer**;
  - Memory associated with the mutexes = each locked mutex uses **4 bytes**.
- Kernel size:
  - Static memory = **13 bytes**;
  - Tasks on the task set = **2 bytes for each task**;
  - Idle task = **see TCB above**.
- Kernel code size:
  - **21.7%** of the Arduino's available **program space**;
  - 13.1% of the Arduino's available **data space**.

Performance – ISR execution time

- By default, executes **250 ticks per second**;
- Fig. shows the evolution of the execution time depending on the number of tasks;



Performance – Task activation time

- Time taken between the activation of a task, and it’s execution;
- **Average = 4015 microseconds**;
- Affected by many factors:
  - Existence of higher priority tasks;
  - Tasks yielding/sleeping;
  - Time until the next ISR tick.
- 4000 microseconds per tick:
  - Coherent with the results found.

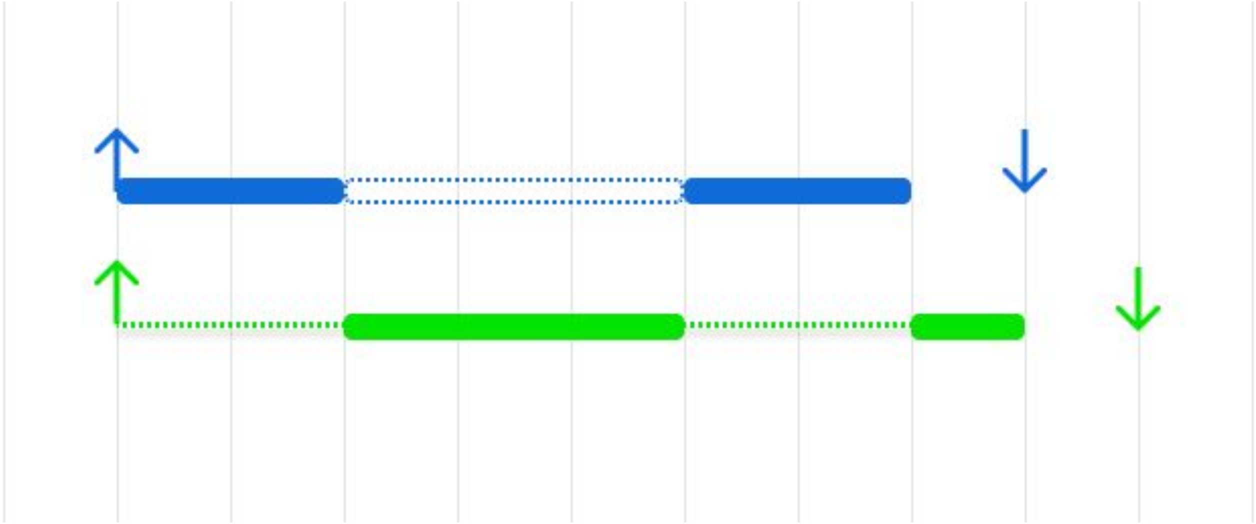
Performance – Context switching time

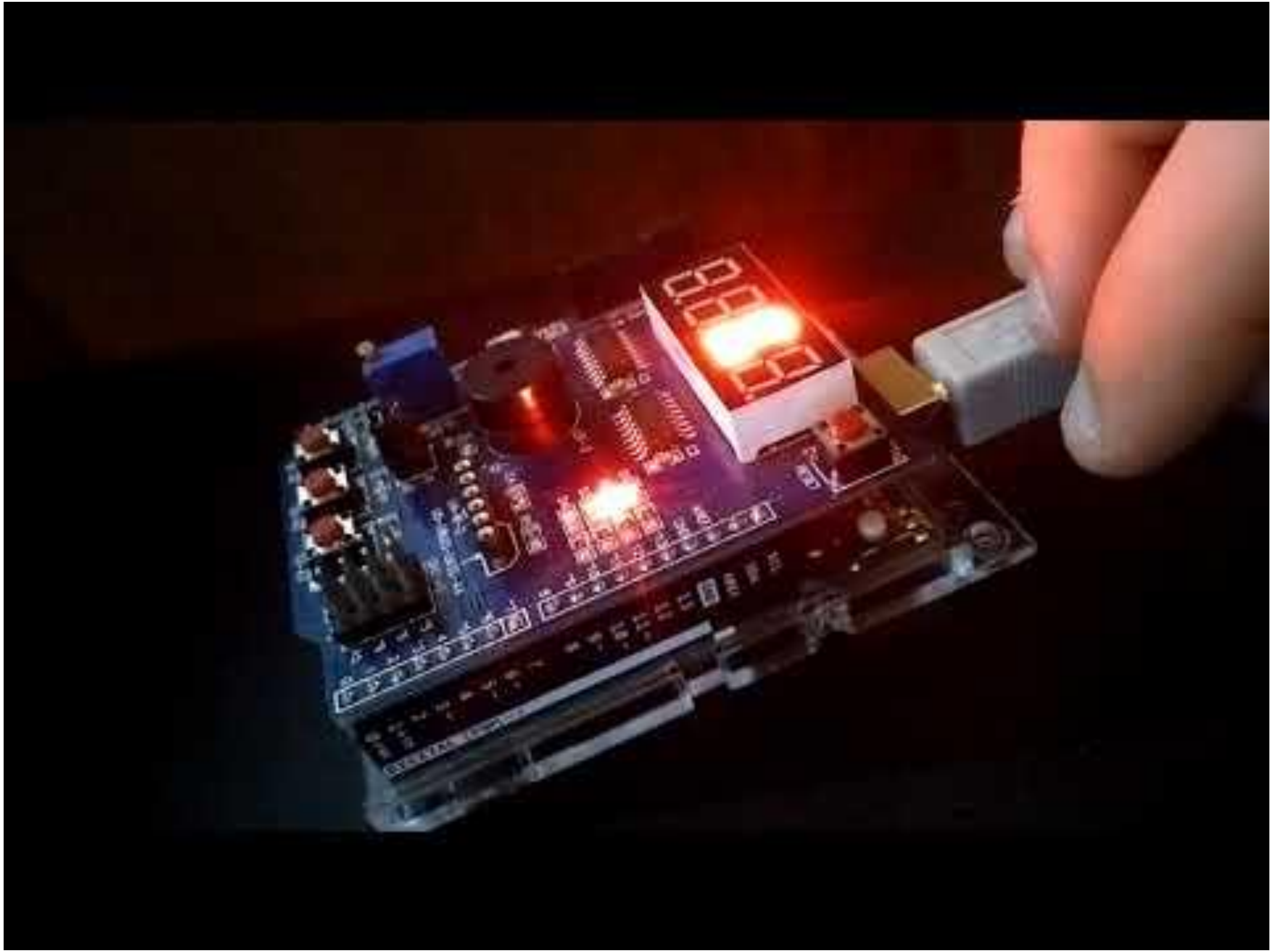
- The time the kernel takes to store a task's context and load the context of another;
- This happens during preemption;
- Measured using 2 tasks: 1 long and 1 short and frequent;
- Measuring the yielding of one task to another makes it easier to discard the effect of *Scheduler* (during **ISR**);
- Average: **165 microseconds**;

Demo

Section 4

Demo 1 – Preemption and sleep





Demo 2 – Mutexes with Priority Inheritance