Requirement specification and model-checking of a real-time scheduler implementation

Khaoula BOUKIR, Jean-Luc BÉCHENNEC, Anne-Marie DÉPLANCHE

Laboratoire des Sciences du Numérique de Nantes

June 2020

Context

If RTOS are to use the new scheduling policies:

- the challenge: scheduling programming is a complex task since multiple implementation constraints are abstracted in literature and must be considered.
- the need: the implementation of scheduling policies have to be rigorously verified

$\Longrightarrow$ **Our intention:** study the suitability of model-checking for conducting such a verification.

Context

If RTOS are to use the new scheduling policies:

- the challenge: scheduling programming is a complex task since multiple implementation constraints are abstracted in literature and must be considered.
- the need: the implementation of scheduling policies have to be rigorously verified

$\Longrightarrow$ **Our intention:** study the suitability of model-checking for conducting such a verification.

Trampoline RTOS

- Real-time operating system based on OSEK/VDX and AUTOSAR standard [Béc2006]
- Partitioned fixed task priority scheduling

Does the implemented scheduler **behaves correctly**?

Our approach

- We propose a verification approach based on model-checking to formally verify an implementation of a global scheduler in Trampoline

Our approach

- We propose a verification approach based on model-checking to formally verify an implementation of a global scheduler in Trampoline

Steps of our verification approach

1. Elaborate the model of the scheduling components inside the OS model
2. Define requirements describing the expected behavior of the scheduling components
3. Build scenario generators of scheduling events in relation with requirements
4. Conduct a modular verification of the requirements with respect to the scenarios

Our approach

previous study

- Implementation of G-EDF in Trampoline
- Modeling the implementation
- First try of verification of some requirements over a single application

Partial verification !

Our approach

previous study

- Implementation of G-EDF in Trampoline
- Modeling the implementation
- First try of verification of some requirements over a single application

Partial verification !

This work

Conduct the verification in depth by:
- rigorously developing the specification requirements
- carrying the verification by generating all possible scenarios of **interleaved** scheduling events

Outline

1. **BACKGROUND**
   - G-EDF implementation architecture
   - **Step 1:** G-EDF implementation modeling
2. **APPLYING THE VERIFICATION APPROACH**
   - **Step 2:** Requirement specification and formalization
   - **Step 3:** Verification scenarios
   - **Step 4:** Verification process

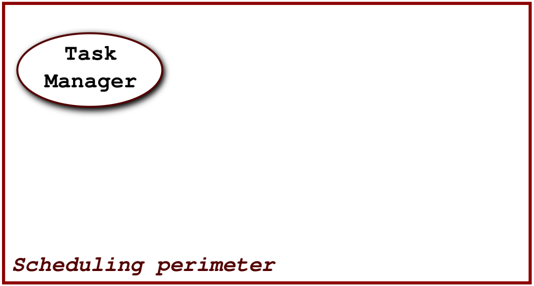
Scheduling perimeter

- the implementation of Trampoline's scheduler is ***kernel-based*** ==> several OS components are contributing to the scheduling decision: **"The scheduling perimeter"**

Scheduling perimeter

- the implementation of Trampoline's scheduler is ***kernel-based*** ==> several OS components are contributing to the scheduling decision: **"The scheduling perimeter"**

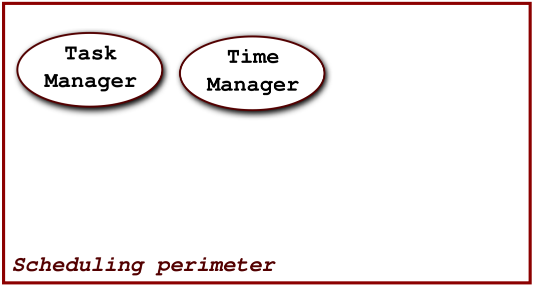
Responsible for operations related to the activation and termination of tasks



Scheduling perimeter

- the implementation of Trampoline's scheduler is ***kernel-based*** ==> several OS components are contributing to the scheduling decision : **"The scheduling perimeter"**

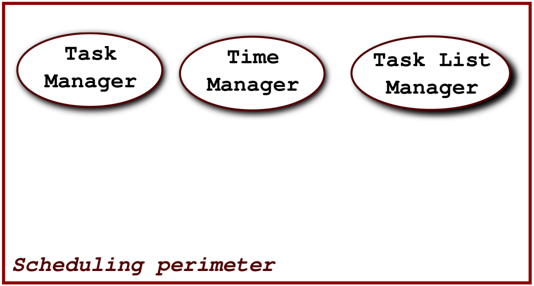
Responsible for the calculation and comparison of task deadlines



Scheduling perimeter

- the implementation of Trampoline's scheduler is ***kernel-based*** ==> several OS components are contributing to the scheduling decision : **"The scheduling perimeter"**

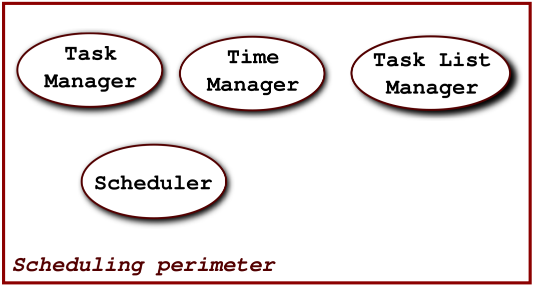
Responsible for the insertions and extractions on ready job lists



Scheduling perimeter

- the implementation of Trampoline's scheduler is ***kernel-based*** ==> several OS components are contributing to the scheduling decision : **"The scheduling perimeter"**

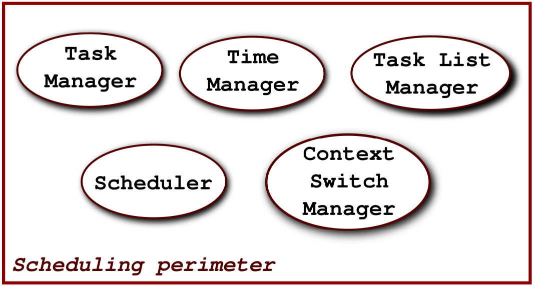
Responsible for determining the scheduling decision according to G-EDF



Scheduling perimeter

- the implementation of Trampoline's scheduler is ***kernel-based*** ==> several OS components are contributing to the scheduling decision : **"The scheduling perimeter"**

Responsible for applying the scheduling decisions

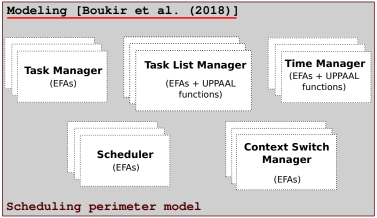


Step 1: G-EDF implementation modeling

General modeling techniques

Extended Finite Automata + UPPAAL Functions

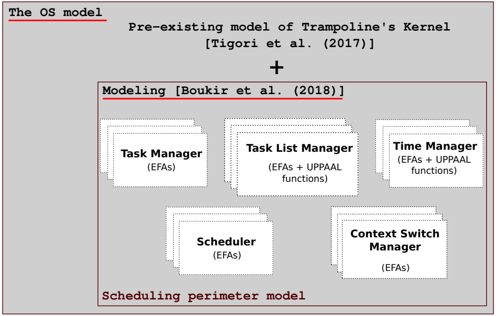
- a function is modeled either by an automaton or an UPPAAL function
- variables used in the model are variables of the OS.
- actions and conditions attached to each transition are the same ones of the system.



General modeling techniques

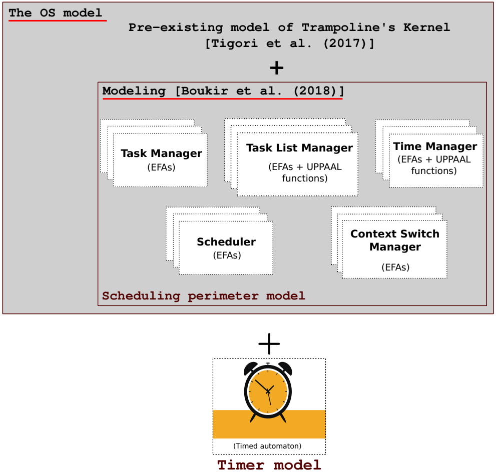
Extended Finite Automata + UPPAAL Functions

- a function is modeled either by an automaton or an UPPAAL function
- variables used in the model are variables of the OS.
- actions and conditions attached to each transition are the same ones of the source code of the system.



General modeling techniques

- the OS kernel is **untimed** ==> its execution is performed in zero time



G-EDF scheduler is based on jobs absolute deadline
$\Longrightarrow$ add an automaton that models the time progression and allows retrieving the current time.

Expected behavior of a G-EDF scheduler

- **Priority rule:** at any time *t*, it is the *m* jobs (at most) with the closest absolute deadlines that are running on the *m* processors of the platform.
- **Work-conserving policy:** processor cannot be be free while there is a ready job.

Expected behavior of a G-EDF scheduler

- **Priority rule:** at any time *t*, it is the *m* jobs (at most) with the closest absolute deadlines that are running on the *m* processors of the platform.
- **Work-conserving policy:** processor cannot be be free while there is a ready job.

To be mapped on the implementation level

Implementation

Why taking the implementation specification into account?

- The scheduling decision within Trampoline involves other OS components
- Even if the scheduler operates correctly, the produced scheduling sequence might be wrong

Example: the context switch manager does not apply the scheduling decision after a rescheduling $\Longrightarrow$ the next rescheduling will be based on false results

Modular approach

Describe the expected behaviour of each component in the form of a set of requirements

Modular approach

Describe the expected behaviour of each component in the form of a set of requirements

Task Manager

- *for every job activation or termination, the scheduler shall be called*

Scheduler

- during the execution, the jobs in the RUNNING state have always a lower absolute deadline than any other ready job
- a processor shall never be idle while there is a ready job

Context Switch Manager

- the context switching shall be performed according to the Scheduler decisions

...

Why using observers?

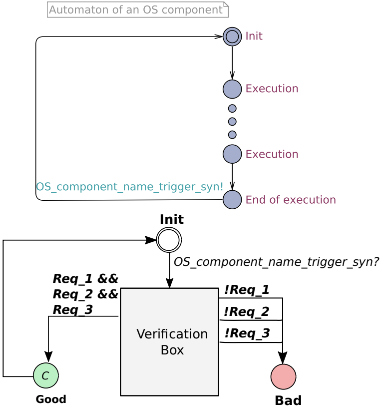
Several requirements are complex and depend on implementation choices ==> the translation to a CTL formulae can become complicated and error-prone

Why using observers?

Several requirements are complex and depend on implementation choices ==> the translation to a CTL formulae can become complicated and error-prone

Structure of an observer

- observer = EFA witch *committed* states running in parallel with the model
- each observer corresponds to an OS component of the scheduling perimeter
- its execution is launched using a triggering broadcast synchronization
- requirement are checked in the verification box using a set of test functions that return true or false depending on the result of meeting the requirements.



Check a requirement

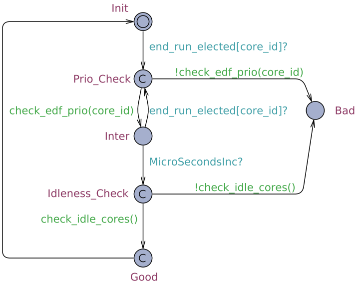
Each requirement is checked using reachability test on the **Good** and **Bad** states of the corresponding observer.

- `A <> Observer.Good`: all paths lead finally to the Good state
- `E <> Observer.Bad`: there exists a path leading to a Bad observer state

Check a requirement

Each requirement is checked using reachability test on the **Good** and **Bad** states of the corresponding observer.

- `A <> Observer.Good`: all paths lead finally to the Good state
- `E <> Observer.Bad`: there exists a path leading to a Bad observer state



- `check_edf_prio()`: *during the execution the m jobs in the RUNNING state have always a lower absolute deadline than any other ready job*
- `check_idle_cores()`: a processor shall never be idle while there is a ready job

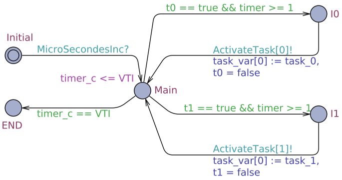
==> `E <> Scheduler_Observer.Bad`

Verification engines

- trigger the scheduler by producing scheduling events
- generate different scenarios of job activation and termination : **activation** and **execution** engines

Verification engines

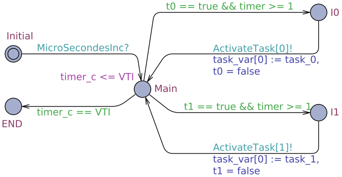
- trigger the scheduler by producing scheduling events
- generate different scenarios of job activation and termination : **activation** and **execution** engines

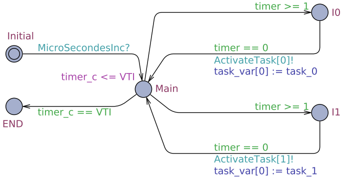


- Activations can occur at anytime and in any order
- Several combinations of task activations are possible: tasks with different deadlines and/or equal deadlines.
- The execution time of a task is indeterministic

Verification engines

- trigger the scheduler by producing scheduling events
- generate different scenarios of job activation and termination : **activation** and **execution** engines

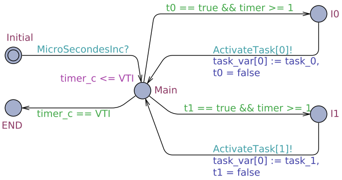


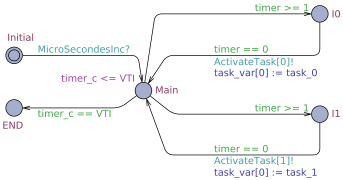


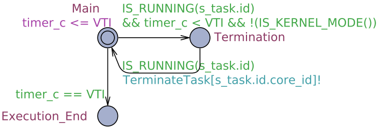
- Activations can occur at anytime and in any order
- Several combinations of task activations are possible: tasks with different deadlines and/or equal deadlines.
- The execution time of a task is indeterministic

Verification engines

- trigger the scheduler by producing scheduling events
- generate different scenarios of job activation and termination : **activation** and **execution** engines







- Activations can occur at anytime and in any order
- Several combinations of task activations are possible: tasks with different deadlines and/or equal deadlines.
- The execution time of a task is indeterministic

Verification engines

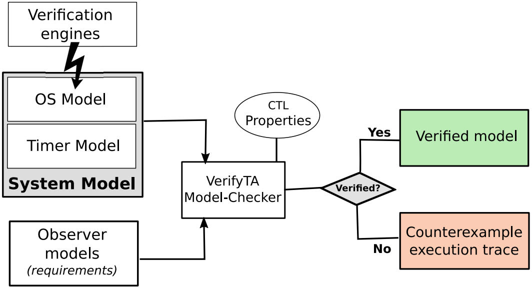
Verification scenarios

- Choose a number of tasks and a verification duration depending on the requirement to be verified

Scenario to verify the context switch manager requirement

- **To be verified:** *"the context switching shall be performed according to the scheduler decisions"*
- **Scenario:** we initiate the activation and termination of one task and observe the reaction of context switch manager regarding the scheduling decisions

Step 4: Verification process



- Observers are combined with the system model (OS + Timer)
- The system model is stimulated using verification engines
- CTL properties are expressed over observer models
- Verification results: good or counterexample scenario

Verification results

- runtime: between 0.9 seconds and 49 hours
- number of states: between 6285 and 1.3×10^9 state

Verification results

- runtime: between 0.9 seconds and 49 hours
- number of states: between 6285 and 1.3×10^9 state

Detected errors:

1. The late update of the ready list regarding the scheduler's decision
2. Not taking into account the scheduling decisions by the context switch manager.
3. Saving the context of a terminating task
4. Trying to load the context of a new activated task

Achieved

- Proposing a modular approach to verify scheduling policies based on model-checking: it allows the checking of specification requirements
- Testing the approach on an implementation of G-EDF within Trampoline
- Detecting implementation errors related to switching the OS from static to dynamic scheduling

Challenges

- The combinatorial explosion of the state space
- Abstracting the system as much as possible

Future works

- Study the integration of model abstraction techniques to limit the explosion of the state space

Thank you for your attention

☺

Bibliography

- Jean-Luc Béchennec, Mikael Briday, Sébastien Faucou, and Yvon Trinquet.
  "Trampoline an open source implementation of the osek/vdx rtos specification"
  In **Emerging Technologies and Factory Automation**, 2006. ETFA'06. IEEE Conference on, pages 62–69. IEEE, 2006.
- Kabland Toussaint Gautier Tigori, Jean-Luc Béchennec, Sébastien Faucou, and Olivier Henri Roux.
  "Formal model-based synthesis of application-specific static rtos"
  ACM Transactions on Embedded Computing Systems (TECS), 16(4) :97, 2017.
- Khaoula BOUKIR, Jean-Luc Béchennec, and Anne-Marie Déplanche.
  "Formal approach for a verified implementation of Global EDF in Trampoline"
  In **Proceeding of the 26th International Conference on Real-Time Networks and Systems**, pages 83-92, 2018