Appears in: Proceedings of the 30th Euromicro Conference on Real-Time Systems Work-in-Progress Session (ECRTS '18 WiP)
Barcelona, Spain, 3–6 July 2018

Towards System-Wide Timing Analysis of Real-Time–Capable Operating Systems

Simon Schuster, Peter Wägemann, Peter Ulbrich, Wolfgang Schröder-Preikschat
Department of Computer Science, Distributed Systems and Operating Systems
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)

***Abstract*—In the context of static timing analysis of real-time operating systems, the usage of generically implemented algorithms necessitates annotation languages to express application-specific knowledge. That is, developers have to provide precise loop bounds or exclude program paths if automatic timing analysis fails or yields too pessimistic results. Current annotation approaches are not able to express and propagate information across all layers of complete real-time systems (i.e., application and system-call layer, operating system, machine-code level).**

To solve this problem, we present our work in progress within the SWAN project to enable System-wide WCET Analyses. Specifically, we provide details on PLATINA, a high-level annotation language, which is processed by an optimization-aware compiler and timing-analysis infrastructure. PLATINA enables expressing parametric program-flow facts based on the real-time system's context-sensitive state (e.g., number of currently active tasks), which are propagated through and usable on all layers of the real-time system. Eventually, PLATINA allows determining if a generically implemented system is real-time–capable and whether timing bounds can be guaranteed during execution.

I. INTRODUCTION & MOTIVATION

The worst-case response time (WCRT) is a vital temporal property of tasks with hard deadlines. Its evaluation demands a sound worst-case execution time (WCET) analysis of all implementation artifacts. Accordingly, static timing analysis of real-time applications is a well-established field with commercially available tools (e.g., aiT [1]) that yield precise bounds on the WCET in practice. They, nonetheless, rely on a largely static program structure, which is inherent to safety-critical applications, to accurately infer data and control flows, to, for instance, bound loops and resolve function pointers.

However, tasks are typically embedded and executed in a broader system's context using a real-time operating system (RTOS). Consequently, its implementation has to be subject to the same temporal analysis. The induced overhead is typically treated as constant and pessimistically added in a deferred analysis step to each task's WCRT [2], [3]. The reason is that system calls and preemptive scheduling intermit data and control flows and thus necessitate a dedicated analysis of kernel paths. The use of such a compositional approach is consequently only feasible when given a static setting, which means that operating system (OS) implementations are tailored to a specific set of statically decidable application parameters.

A relatively new development in safety-critical real-time systems is the employment of dynamic runtime environments, such as Real-Time Linux [4], [5] that correspond more to a general-purpose OS and promote code reuse by generically implemented algorithms and interfaces. A reliable indicator of this development is the future Adaptive AUTOSAR standard [6], which addresses the increasing complexity of driver-assistance and autopilot functions in vehicles. Here, the aggregation of the individual (i.e., application and kernel) WCETs is fraught with inevitable overestimations that rise tremendously with system complexity [7]. The cause is in the loss of a tailored and static OS implementation and the decoupling of control flows within the application and the OS kernel resulting from the system's dynamic system-call layer and runtime reconfigurability. Consequently, system facilities are designed to serve all possible application scenarios and thus exhibit dynamic data structures and execution paths, which in turn jeopardize conventional analyses [3]. However, flexible deployment of dynamic RTOS in safety-critical settings still requires realistic bounds for the induced overheads.

Our Contribution: We present our work in progress on system-wide WCET analysis that makes available context-specific knowledge to the kernel analysis by PLATINA, an expressive annotation language. PLATINA features parametric flow facts that can be linked to system state (e.g., number of currently active tasks) in an application- and context-sensitive manner. In conjunction with an optimization-aware compiler and timing-analysis infrastructure, we can infer tighter yet sound bounds for generically implemented real-time systems.

II. RELATED WORK & PROBLEM STATEMENT

Colin and Puaut [8] were among the first to pinpoint the fundamental problems of OS WCET analysis: meaningful construction of global control-flow graphs over kernel boundaries. They identified indirect function calls, immanent in the system calls interface, as well as dependencies between application properties and kernel loop bounds as crucial issues. Their approach was to modify the system's source code, restoring a statically analyzable implementation. Despite these tedious and non-generalizable measures, a high degree of overestimation (avg. 86 %) remained. Later Sandell et al. [9] reported a large number of "uninteresting" kernel paths (e.g., error handling) in their analysis. Furthermore, they observed the mediocre performance of the data-flow analysis within the OS kernel. They used an annotation language with constant expressions at assembly level to address the problems, which, however, involved a high degree of recurring effort with still unsatisfactory reduction of pessimism.

Acknowledgments: This work is supported by the German Research Foundation (DFG), in part by Research Grant no. SCHR 603/9-2, the SFB/Transregio 89 "Invasive Computing" (Project C1), and the Bavarian Ministry of State for Economics under grant no. 0704/883 25.

Among others, Schneider [7] determined the dynamic reconfigurability and system-call interface, the tracing of call graphs, as well as the internal feedback between the RTOS and the applications as further challenges towards realistic WCET estimations. He proposed an integrated WCET and scheduling analysis as a potential solution. In 2009 Lv et al. compiled a survey [3] on RTOS-analysis attempts, from which they derived a set of challenges. Like Schneider, they question the usefulness of single, global WCET estimates for individual operations but instead suggest a parametric analysis that captures specific WCETs for each invocation.

Since then, research focused on circumventing the problem by tailoring the OS to be deterministic again. Undecidable artifacts (e.g., loop iterations, indirect calls) are eliminated by application-specific source-code modifications: For example, in the seL4 kernel [2] preemption points in loops are added to limit the length of consecutive kernel execution. In our work on SysWCET [10], we leveraged the scheduling semantics to eliminate globally infeasible system execution paths by using a tailored operating system. However, for larger RTOSs (e.g., Real-Time Linux) these code-tailoring approaches are not feasible, due to the high complexity and recurring effort.

Our challenge: Instead of tailoring, we opt for a generic annotation of the OS implementation and subsequent context-aware WCET analysis that proves real-time capabilities specifically for a given application setting. We identified three practical symptoms on the fundamental problem of context-sensitive control flows: (1) Control-flow reconstruction issues, (2) paths that are either unanalyzable or become infeasible in certain contexts, and (3) overly pessimistic bounds as context knowledge cannot be automatically inferred. The resulting challenge is therefore to provide a unified way of formulating, passing, and evaluating state- and context-sensitive flow facts in a system-wide WCET analysis, especially when analysing OS operations. As assembly-level annotations [2], [9] are not an option due to their poor reusability and manageability, this requires parametric annotations at the source-code level.

III. Approach

We tackle our challenge by SWAN, an approach to enable System-wide WCET Analyses. Figure 1 illustrates its fundamental concept: the core (middle) is PLATINA, a parametric annotation language expressing context-sensitive information on the source-code level. The annotations are associated with system facts (top) that hold context-dependent information. Finally, to achieve a context-aware WCET analysis (bottom), SWAN provides a semantic-preserving transformation from code to assembly level. We detail those steps in the following.

Our goal with PLATINA is to bridge the immanent semantic gap between application and kernel analysis by propagating context-dependent information (*system facts*) to the low-level WCET analysis in an automated fashion. Hence, we provide parametric (i.e., context-dependent) annotations to address said three challenging symptoms that so far prevent tighter bounds on generic OS operations: (1) Annotation of indirect function calls to aid the control-flow reconstruction. This is a key

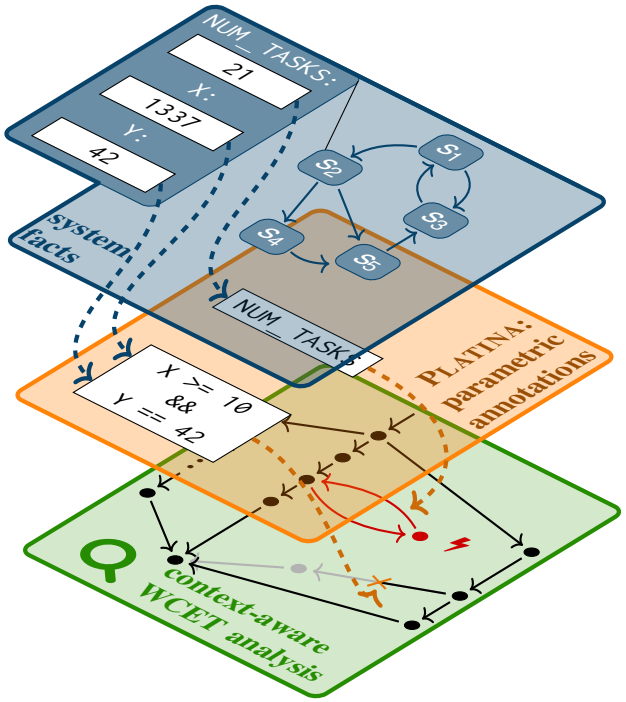

Fig. 1. Information propagation within SWAN: semantic system facts are gathered from a semantic model. Parametric annotations use these pieces of information to express the semantics of context-sensitive execution flow at the source code level. After lowering to the assembly-code level, the static WCET analysis leverages this information to exclude infeasible execution flows.

enabler for the subsequent propagation of context-dependent flow facts. (2) State-dependent annotation of branches in conditional executions to eliminate paths that become infeasible only in certain contexts. (3) Context-sensitive annotations to bound computations by application and configuration knowledge. These building blocks form an expressive annotation language that allows associating the semantics of individual system facts with the actual execution flow. To reflect the complexity of control flows within the OS, PLATINA's annotation language allows the programmer to combine multiple system facts and even reuse expressions by defining custom facts or functions. Furthermore, to foster long-term maintainability of annotations, we decided to integrate them at the annotated program point within the source code (e.g., C++ program).

The *system-facts layer* parametrizes and instantiates PLATINA annotations. Therefore, the layer provides and stores static and application-specific configuration knowledge as well as context-dependent information, which we call *system facts*: the atomic entities on which the annotation language operates. Higher-level analysis, such as our SysWCET [10] approach to incorporate scheduling semantics, also engage at this level to further derive and enrich the model with further system facts.

Finally, the annotation expressions are evaluated over a given context of system-fact values within our *context-aware WCET analysis*, which is based on and extends the PLATIN toolkit [11]. While boosting (re-)usability, our earlier decision to keep the annotations at source-code level poses the question of how they can be propagated soundly to the assembly-code level of the WCET analysis. Here, we rely on optimization-

aware compilation [12] to transform context-specific flow information into the machine-code–level control-flow graph. Consequently, our lowering preserves the annotation semantics, which ultimately allows us to infer specific *facts* on the execution *flow* from the contexts of system calls. Examples include loop bounds that are specific to the number of runnable tasks or paths that are only infeasible in the given context. That way, we obtain context-sensitive and thus more accurate bounds for OS overheads, which proofs that the OS is real-time capable in the given setting. Our lowering, while based on an optimization-aware compilation, enables parametrization without recompilation: the same binary can be evaluated for arbitrary system facts and in multiple contexts, which retains scalability and usability even for large implementations.

In summary, SWAN provides sound propagation of high-level, semantic, context-sensitive information throughout the compilation and analysis processes down to the level of the static WCET analysis. The parametric annotation mechanism allows obtaining tailored, context-aware timing bounds even for generic OS kernels. Thus, SWAN proves their real-time capability in a specific context with potentially higher analysis accuracy compared to traditional decoupled WCET analysis.

IV. Preliminary Results

To demonstrate the feasibility of SWAN, we conducted a case study on FreeRTOS and relevant parts of the Linux kernel (i.e., scheduler). In both cases, traditional WCET analysis does not obtain realistic bounds; partly it even fails in the reconstruction of the control-flow graph. This observation, first of all, substantiates the need for an integrated, context-sensitive approach to WCET analysis. Our first results with SWAN are promising, and we were already able to annotate and analyze various intricate parts of both settings. As quantitative results depend on the given application scenario, we graphed the impact of our approach on the example of FreeRTOS's `vSuspendTask` system call in Figure 2: Whenever a task is suspended, a reschedule is triggered, entailing the selection of the next runnable task. Here, FreeRTOS relies on a set of queues (one per priority level), which are probed with decreasing priority. For our experiments, we configured FreeRTOS for the ARM Cortex-M4 platform and a total of 42 priority levels. Without our extensions, PLATIN computed (supported by constant annotations) a static upper bound of 6485 cycles for the worst case, which assumes no runnable task in any of the queues and therefore requires 42 probing steps. However, as there is a linear relationship between the number of probing steps and the system call's WCET, this bound is too pessimistic in most cases: we observed divergence of over 178 %. With SWAN, we were able to accurately express this context dependency by combining the number of priority levels and the priority of the next task. The former is a system-configuration property; the latter is a context-dependent property (e.g., available from our state-transition graph [10]). Consequently, our bounds correctly reflect the maximum number of runnable tasks for any given state and context; thus ultimately relieving OS overheads from unnecessary pessimism.

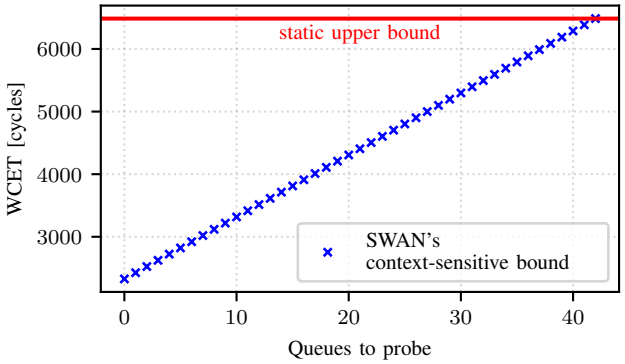

Fig. 2. Runtime of the `vTaskSuspend` system call in relation to the number of task queues (total set of 42) that are probed until a runnable task is found.

V. Conclusion and Future Work

In this paper, we presented SWAN, our ongoing effort on context-sensitive WCET analysis of RTOSs. Its key element is PLATINA, a parametric annotation language to express dependencies between a system's control-flow and application states generically. Combined with system facts and an optimization-aware compiler and timing analysis, we can both prove real-time capabilities in a given context as well as eliminate overly pessimistic constant bounds on RTOS overheads.

We currently finish our prototype and case study of FreeRTOS and Linux. Beyond this proof of concept, we consider the sheer size of, for example, Real-Time Linux as another fundamental challenge. Thus, our next steps are devoted to improving traceability of system facts and annotations as well as the overall scalability and usability. In the future, we see great promise for the integration of high-level application and RTOS semantics to further reduce analysis pessimism.

References

[1] AbsInt., "aiT worst-case execution time analyzers," absint.com/ait.

[2] B. Blackham, Y. Shi, S. Chattopadhyay, A. Roychoudhury, and G. Heiser, "Timing analysis of a protected operating system kernel," in *Proc. of RTSS '11*, 2011.

[3] M. Lv, N. Guan, Y. Zhang, Q. Deng, G. Yu, and J. Zhang, "A survey of WCET analysis of real-time operating systems," in *Proc. of ICESS '09*, 2009.

[4] J. M. Calandrino, H. Leontyev, A. Block, U. C. Devi, and J. H. Anderson, "LITMUS^{RT}: A testbed for empirically comparing real-time multiprocessor schedulers," in *Proc. of RTSS '06*, 2006.

[5] B. B. Brandenburg, "Scheduling and locking in multiprocessor real-time operating systems," Ph.D. dissertation, UNC, 2011.

[6] AUTOSAR, "AUTOSAR the next generation – the adaptive platform," autosar.org/fileadmin/files/presentations/EUROFORUM_Elektronik-Systeme_im_Automobile_2016_-_FUERST_Simon.pdf, 2016.

[7] J. Schneider, "Why you can't analyze RTOSs without considering applications and vice versa," in *Proc. of WCET '02*, 2002.

[8] A. Colin and I. Puaut, "Worst-case execution time analysis of the RTEMS real-time operating system," in *Proc. of ECRTS '01*, 2001.

[9] D. Sandell, A. Ermedahl, J. Gustafsson, and B. Lisper, "Static timing analysis of real-time operating system code," in *Leveraging Applications of Formal Methods*, ser. Lecture Notes in Comp. Science, 2004.

[10] C. Dietrich, P. Wägemann, P. Ulbrich, and D. Lohmann, "SysWCET: Whole-System Response-Time Analysis for Fixed-Priority Real-Time Systems," in *Proc. of RTAS'17*, 2017.

[11] P. Puschner, D. Prokesch, B. Huber, J. Knoop, S. Hepp, and G. Gebhard, "The T-CREST approach of compiler and WCET-analysis integration," in *Proc. of SEUS '13*, 2013.

[12] B. Huber, D. Prokesch, and P. Puschner, "Combined WCET Analysis of Bitcode and Machine Code Using Control-flow Relation Graphs," in *Proc. of LCTES '13*, 2013.