Work Already Published: Priority-Based Concurrency and Shared Resource Access Mechanisms for Nested Intercomponent Requests in CAmkES

Marion Sudvarg*, Zhuoran Sun†, Ao Li‡, Chris Gill§, Ning Zhang¶,
Department of Computer Science & Engineering
Washington University in St. Louis
St. Louis, Missouri
Email: *msudvarg@wustl.edu, †zhuoran.sun@wustl.edu, ‡ao@wustl.edu, §cdgill@wustl.edu, ¶zhang.ning@wustl.edu,

***Abstract*—Component-based design encapsulates and isolates state and the operations on it, but timing semantics cross-cut these boundaries when a real-time task's control flow spans multiple components. Under priority-based scheduling, intercomponent control flow should be coupled with priority information, so that task execution can be prioritized appropriately end-to-end. However, the CAmkES component architecture for the seL4 microkernel does not adequately support priority propagation across intercomponent requests: component interfaces are bound to threads that execute at fixed priorities provided at compile-time in the component specification. In our work-already-published [1],[1] we present a new library for CAmkES with a thread model that supports multiple concurrent requests to the same component endpoint. Propagation and enforcement of priority metadata ensures those requests are appropriately prioritized. Our library provides implementations of Non-Preemptive Critical Sections, the Immediate Priority Ceiling Protocol, and the Priority Inheritance Protocol for components encapsulating critical sections of exclusive access to a shared resource, and extends these mechanisms to support nested lock acquisition. We measure overheads and blocking times of our implementation and discuss schedulability analysis. The analysis uses a new hyperbolic bound for rate-monotonic scheduling of tasks with blocking times that allows tasks to be assigned non-unique priorities. Evaluations on both Intel x86 and ARM platforms demonstrate that our library allows CAmkES to provide suitable end-to-end timing for real-time systems.**

I. Overview

As the complexity of software systems has increased, component-based software engineering has emerged as a key approach for providing structure, modularity, and reusability in system design [2]. Components encapsulate state, computation, and communication, allowing for (1) separation of functional concerns and (2) isolation of resource utilization within components to ensure timing and other para-functional properties, while allowing (3) sophisticated behaviors to be realized, and (4) desired properties to be enforced locally and end-to-end, through composition and coordination of multiple components. CAmkES, which targets the seL4 microkernel [3], provides a description language for the functional requirements of a component-based embedded system, and for static assignment of para-functional attributes such as priorities to component threads. Such static assignment, however, may be problematic in systems where real-time task execution crosses component boundaries. Under priority-driven scheduling, tasks are assigned priorities to ensure their deadlines are met. Tasks and components may be orthogonal: a task may be decomposed into execution across multiple components, and a single component may execute on behalf of multiple tasks. Thus, by assigning priorities to *components* rather than to *tasks*, CAmkES does not fully support priority-driven scheduling of multi-component tasks.

To address this limitation, in [4] we presented a new library to enable priority-aware inter-component requests in CAmkES running atop seL4. The library provides a concurrency framework that allows multiple concurrent tasks to execute across shared components, while retaining end-to-end task prioritization, in only 659 lines of code. It supports (1) multiple concurrent requests to the same component procedural interface (CPI) endpoint; (2) priority propagation, which couples requests with priority metadata and ensures that each component thread is prioritized according to the task for which it executes; and (3) implementations of Non-Preemptive Critical Sections (NPCS), Immediate Priority Ceiling Protocol (IPCP), and Priority Inheritance Protocol (PIP), for components encapsulating exclusive access to a shared resource.

Our work-already-published [1][1] extends our prior work in [4], introducing mechanisms to support nested lock acquisition via intercomponent requests, including an implementation of nested PIP. It provides an overview of how to do schedulability analysis for a component-based task system specified with our extensions to CAmkES, taking into account blocking times induced by both library overhead and shared resource access under our supported protocols. To this end, in [1, Appendix A], we present a new formulation of the hyperbolic bound for rate-monotonic scheduling of tasks with

[1]Available as a preprint from https://www.sudvarg.com/publications/RTS23_Nested_Locking_CAmkES.pdf

blocking times, which allows tasks to be assigned non-unique priorities. The mechanisms our library provides are designed to be both fast and predictable in execution time; we present measurements demonstrating that overheads are appropriately bounded. We also demonstrate, through empirical timing measurements of task sets running on both Intel x86 and ARM hardware platforms, that our implementation is successful in meeting end-to-end deadlines for cross-component task execution in real-time systems.

II. Summary of Results

In [1], we evaluate our library using the CAmkES 3.10.0 framework, targeting version 12.1.0 of the seL4 kernel running on an Intel x86-64 platform with two Xeon Gold 6130 Skylake processors running at 2.1 GHz, and on a Raspberry Pi 3 Model B+ running at 700 MHz.

We individually measure the overheads for both sending and replying to requests over an endpoint, separately profiling our PIP implementation for requests to a CPI with an already-acquired lock versus those with an available lock. We additionally measure the overheads of nested requests from a CPI implementing PIP to CPIs implementing both PIP and priority propagation, respectively, reporting both the call and reply times, as well as the time to send a nested priority inheritance update. All measurements are listed in [1, Table 2]. We compare these overheads to that of a request over the CAmkES built-in seL4RPCCall connector; while our protocols do induce additional overhead, the maximum values we measured (a nested call and reply to a CPI with priority inheritance induced up to about 13,200 cycles of overhead on Intel and about 9,800 cycles on ARM) equates to less than 6.3 μs on Intel and 14.1 μs on ARM, which is suitably low for task sets running with periods as small as 10 ms.

Furthermore, benchmarked performance numbers for the seL4 kernel without the CAmkES framework report an average overhead of 383 and 389 cycles respectively for IPC call and reply between threads on an Intel x86_64 Skylake platform; and 404 and 409 cycles respectively on the ARMv8 platform in 64-bit mode [5]. Even with nested priority inheritance, our mean overheads exceed this by only about $14.4\times$ on the Xeon Gold 6130 and $7.84\times$ on the Raspberry Pi 3B+. Benchmarked performance numbers from the related Patina framework [6] are only available from an ARM Cortex-A9 processor running on the Zynq-70000 XC7Z020, and as Patina's seL4 implementation is not open-sourced, we cannot perform a direct comparison. However, their maximum reported overheads for requests to the mutex service were 11,165 cycles (unlocked) and 13,918 cycles (locked); more than the maximum observed overhead on ARM of our mechanisms, even for nested locking (which Patina does not support).

To facilitate checking the schedulability of actual task sets running in CAmkES atop seL4 on our selected hardware platforms, we generate synthetic task sets over a representative topology of interacting components (illustrated in Figure 1), all running on a single core. In each task set, components originating tasks **t1** and **t2** both request a CPI provided by component **A**, while those originating **t3** and **t4** both request a CPI provided by component **B**. Both **A** and **B** encapsulate exclusive execution using PIP. We evaluate 4 combinations of assignments of IPCP, priority propagation, and PIP to components **C**, **D**, and **E** as outlined in [1, Table 3]. For each configuration, we generate 10 task sets for each utilization from 0.1 to 1.0, for a total of 400 task sets. Task utilizations are assigned according to the UUniSort algorithm [7], with periods selected from a set of harmonic values from 10 ms to 1 second. Each task set is run for 10 hyperperiods, with each task releasing up to 2000 jobs. Given the predictable and well-bounded nature of the overheads exhibited by our library on both hardware platforms, no deadlines were missed for any of our tested task sets.

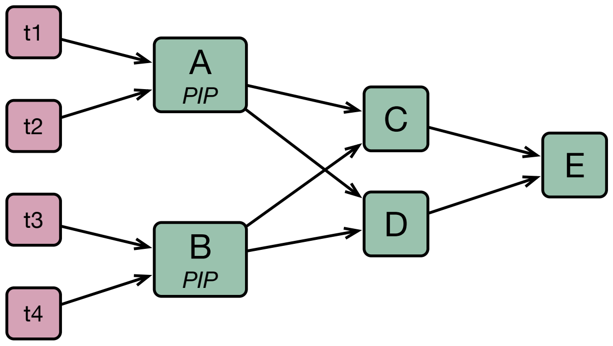

Fig. 1. Task and Component Test System

In summary, our work-already-published [1] extends our earlier concurrency framework presented in [4] to support nested lock acquisition, including nested priority inheritance. The results of our evaluations demonstrate that our extensions to the CAmkES component framework can prioritize cross-component control flows effectively. Overheads remain low, and our userspace framework remains easy to integrate into existing CAmkES-based systems.

***Index Terms*—real-time systems, component middleware, priority protocols**

References

[1] M. Sudvarg, Z. Sun, A. Li, C. Gill, and N. Zhang, "Priority-based concurrency and shared resource access mechanisms for nested intercomponent requests in camkes," *Real-Time Systems*, 2023. [Online]. Available: https://www.sudvarg.com/publications/RTS23_Nested_Locking_CAmkES.pdf

[2] M. D. McIlroy, "Mass-produced software components," *Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct 1968*, pp. 79–85, Jan 1969.

[3] "The sel4 microkernel," https://docs.sel4.systems/projects/sel4/, seL4 Foundation, accessed: 23 Jan, 2022.

[4] M. Sudvarg and C. Gill, "A concurrency framework for priority-aware intercomponent requests in camkes on sel4," in *2022 IEEE 28th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2022, pp. 1–10.

[5] "sel4 benchmarks," https://sel4.systems/About/Performance/, seL4 Foundation, accessed: 02 June, 2023.

[6] S. Jero, J. Furgala, R. Pan *et al.*, "Practical principle of least privilege for secure embedded systems," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2021, pp. 1–13.

[7] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Syst.*, vol. 30, no. 1–2, p. 129–154, May 2005.