Priority-Based Concurrency and Shared Resource Access Mechanisms for Nested Intercomponent Requests in CAmkES

Marion Sudvarg (msudvarg@wustl.edu)
Zhuoran Sun, Ao Li, Chris Gill, Ning Zhang

Accepted for publication in the Springer Real-Time Systems Journal
Preprint available at www.sudvarg.com/priority-aware-camkes

Overview

The seL4 microkernel is a formally verified, capability-based, lightweight microkernel that supports real-time and mixed-criticality systems. CAmkES provides a component description language and middleware layer atop seL4. Task execution across components should be prioritized appropriately end-to-end. However, CAmkES component interfaces are bound to threads that execute at fixed priorities provided at compile-time in the component specification. We present a new library for CAmkES with a thread model that supports multiple concurrent requests to the same component endpoint. It provides propagation and enforcement of priority metadata, such that those requests are appropriately prioritized. It allows enables priority-based locking (including nested locking) via:

- **Non-Preemptive Critical Sections (NPCS)**
- **The Immediate Priority Ceiling Protocol (IPCP)**
- **The Priority Inheritance Protocol (PIP)**

Component-Based Design

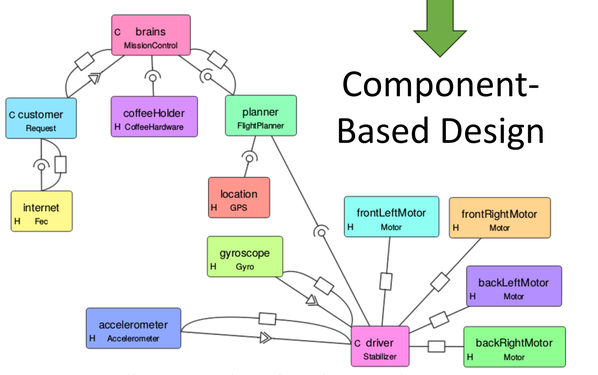


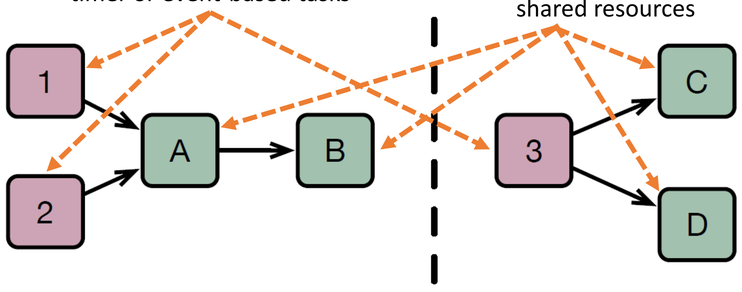
Diagram courtesy https://docs.sel4.systems/projects/camkes/visual-camkes/VisualCAmkES_Layout_Example.png

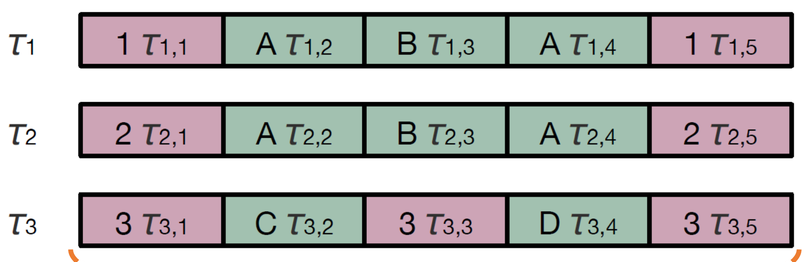
- Components encapsulate and isolate state and functionality
- Provides structure, modularity, reusability
- Components define explicit interfaces for communication and control flow

Real-Time Component-Based Systems

Some components originate timer or event-based tasks

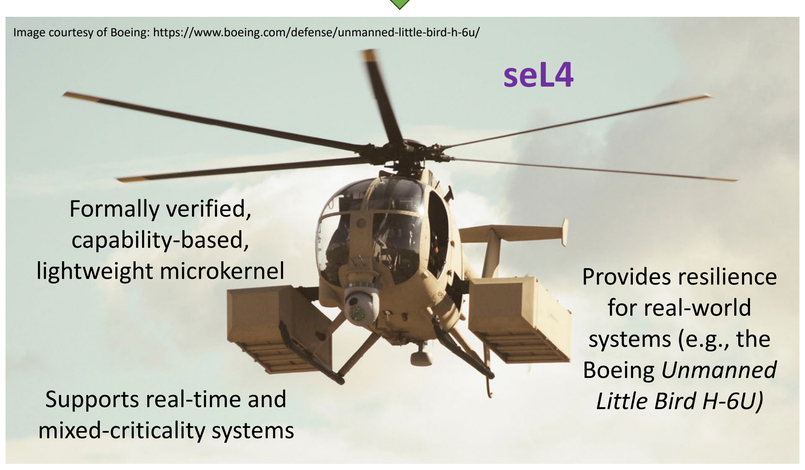
Some components provide common functionality or shared resources





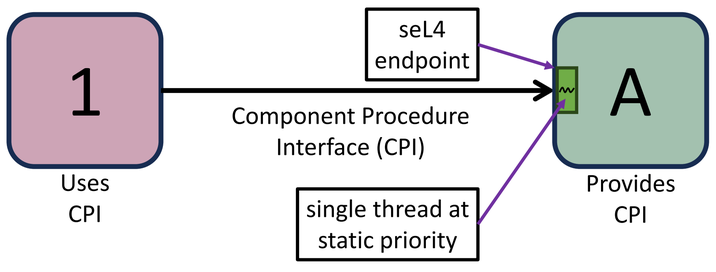
Tasks execute end-to-end across components via synchronous RPC invocations

Image courtesy of Boeing: https://www.boeing.com/defense/unmanned-little-bird-h-6u/



CAmkES

Component **A**rchitecture for **m**icro**k**ernel-based **E**mbedded **S**ystems
Component description language and middleware framework for seL4



Problem: Tasks of *different* priorities may concurrently send requests to the same CPI!

Components Encapsulate

Shared Resources

Components may encapsulate critical sections of exclusive (**locked**) access to a shared resource

1 NPCS

Critical section executes nonpreemptively

2 IPCP

Critical section cannot be preempted by task with lower priority than any task that can request it

Both implemented by binding a single thread with static priority to component procedure interface's underlying endpoint

- Component thread assigned highest system priority (255)
- (Under RR scheduling, tasks restricted to priorities 0-254)
- Thread executes at priority ceiling of possible requestors
- (Under RR scheduling, tasks restricted to even priorities, IPCP component restricted to odd priorities)

3 PIP

- Critical section executes at highest priority among waiting requestors
- Multiple threads (one per possible requestor) wait on endpoint at priority ceiling to receive new requests
- Priority passed with request message
- If **unlocked**, handling thread sets **locked** then reduces its priority to that of requestor
- If **locked**, handling thread elevates priority of thread with lock then waits in userspace priority queue (**Notification Manager**)
- Once request is complete, notification sent to thread at head of queue

Shared Functionality

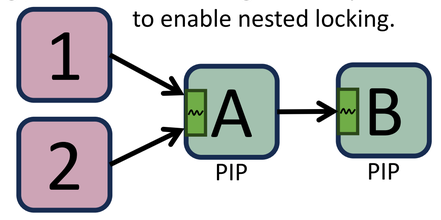
Components may need to support multiple concurrent requests to thread safe, reentrant functionality.

4 Priority Propagation

- Multiple threads (one per possible requestor) wait on endpoint at priority ceiling to receive new requests
- Priority passed with request message
- Handling thread reduces its priority to that of requestor
- This allows multiple concurrent requests to be handled at corresponding task's priority

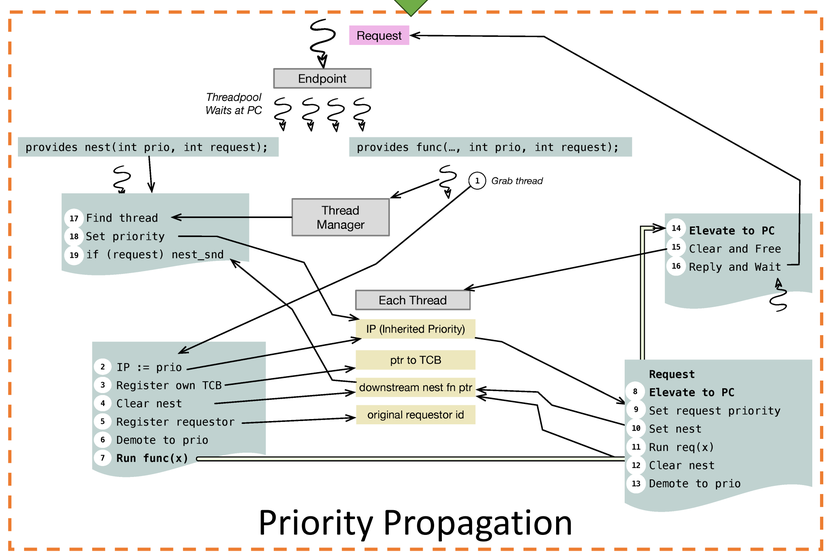
Nested Dependencies

A shared service/resource component may send further downstream requests, e.g., to allow a locked region to request shared functionality or to enable nested locking.

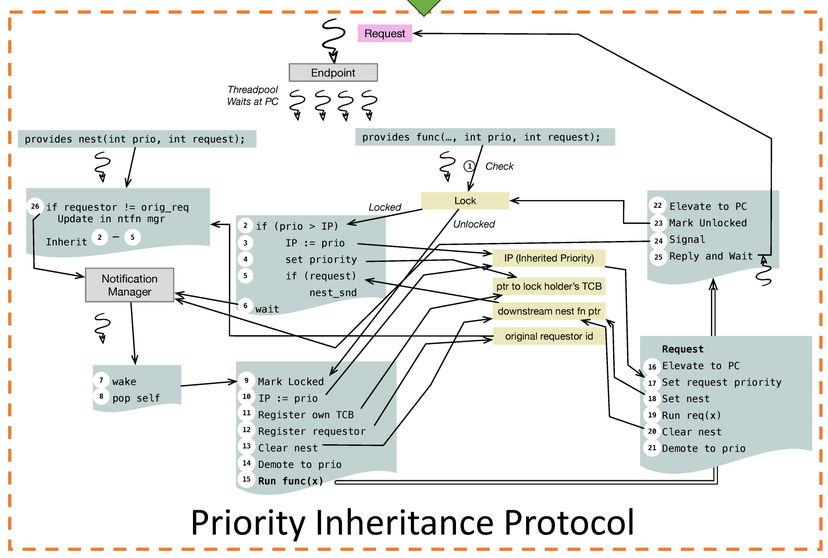


Challenge: Inherited priorities must be propagated to any downstream components handling a request

Solution: PIP and priority propagation CPIs provide an additional function to receive nested priority inheritance updates



Priority Propagation



Priority Inheritance Protocol

Advantages

- Framework now supports nested requests from CPIs implementing PIP
- Nested locking and nested priority propagation are possible
- Even with long request chains, a component needs to only notify the next downstream component
- Uses priority-based locking semantics – no additional atomic operations necessary in userspace framework, bounding overheads

Caveats

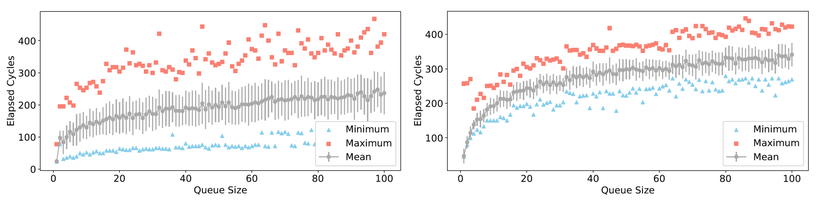
- Additional thread required on CPI endpoint to received nested priority inheritance updates
- Requires minor changes to existing CAmkES component system descriptions to specify which protocol a CPI uses, and to track which CPI is invoking a downstream component's functionality
- Nested PIP does not support round-robin scheduling of threads with equal priorities (requires traditional fixed-priority preemptive scheduling)

Overheads Associated with Framework

Intercomponent Requests

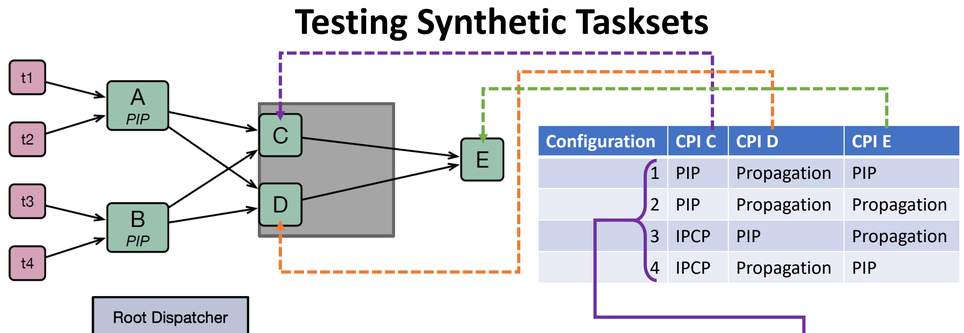
	Intel Xeon Gold 6130				Raspberry Pi 3 Model B+			
	min	max	mean	std	min	max	mean	std
Call, built-in	2478	4568	2528	211	563	3359	619	278
Reply, built-in	2494	2836	2519	34	416	1298	449	86
Call, fixed	3178	4810	3306	189	834	2591	1010	179
Reply, fixed	2590	3342	2659	142	422	954	466	94
Call, propagated	4348	6830	4574	287	2085	5516	2368	351
Reply, propagated	3536	3960	3567	41	1606	2335	1694	76
Call, inherited, unlocked	4344	6834	4625	288	1935	5381	2226	335
Call, inherited, locked	4372	6282	4603	245	1966	5052	2242	314
Reply, inherited, unlocked	3520	4028	3549	50	1483	2095	1531	65
Reply, inherited, locked	3514	3788	3547	28	1470	2319	1531	96
Call, PIP to PIP	5364	6878	5618	240	3079	5696	3355	279
Reply, PIP to PIP	5402	6300	5462	89	2898	4109	3019	121
Nest, PIP to PIP	5144	6720	5346	303	1175	3315	1844	446
Call, PIP to propagated	5292	6306	5539	196	3165	6068	3452	308
Reply, PIP to propagated	4558	5318	4679	130	2563	3552	2668	98
Nest, PIP to propagated	6068	7530	6385	243	1838	5088	2243	351

Insert and Remove from Notification Manager



Intel Xeon Gold 6130 | Pi 3B+ (ARMv8 Cortex-A53)

Testing Synthetic Tasksets



Configuration	CPI C	CPI D	CPI E
1	PIP	Propagation	PIP
2	PIP	Propagation	Propagation
3	IPCP	PIP	Propagation
4	IPCP	Propagation	PIP

- 10 task systems of each configuration at each total utilization 0.1-1.0
- Utilizations split using UUniSort
- Root dispatcher releases system dispatchers sequentially
- Each system dispatcher releases tasks within one task system for 10 hyperperiods
- **No deadlines missed!**

Conclusions

- New framework for CAmkES supports priority propagation, NPCS, IPCP, and PIP
- Extensions to framework support nested PIP
- Priority-based lock semantics do not need additional atomic operations
- These extensions allow CAmkES to provide suitable end-to-end timing guarantees for real-time systems
- We will extend to support asynchronous event notifications
- We intend to modify the CAmkES parser to automatically add priority and task identifier metadata to RPC request messages

Funded by NSF grants CSR-1814739, CNS-17653503, and CNS-2229290, and NASA grant 80NSSC21K1741

Washington University in St. Louis
James McKelvey School of Engineering