Thread Migration for Mixed-Criticality Systems

Alexander Zuepke
alexander.zuepke@hs-rm.de

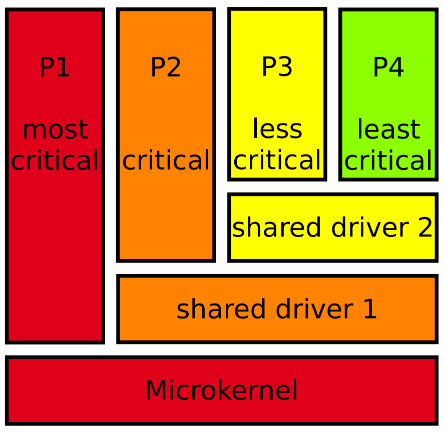
Resource Sharing in Mixed-Criticality Systems

Mutexes and semaphores do not fulfill the *freedom of interference* requirements of ISO 26262 (functional safety in automotive) and IEC 61508 (general functional safety) when sharing resources among functional units of different criticality.

Instead, we propose the following approach to resource sharing:
- isolate access to shared resources into *dedicated driver components*
- analyze drivers *independently* from their users
- use *criticality-aware locking protocols* to access these drivers
- use *thread migration* for fast context switching between components

We aim at lowering complexity and improve the analysis of mixed-criticality systems.

System Architecture



Shared drivers need to have *same* or *higher* criticality levels than their clients. Here, driver 1 is shared by {P2 .. P4} and has to fulfill P2's level.

Thread Migration

Definition in literature:
- a client lends its thread to the server
- the server is a passive entity

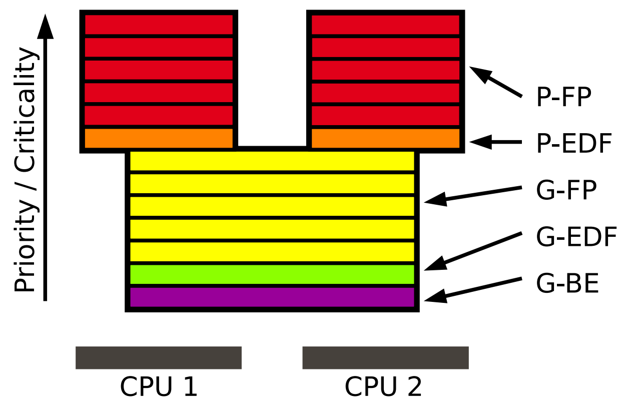
When put into the world of mixed-criticality systems, this concept provides a fast mechanism for context switching between isolated components.

Our proposed *body & soul* concept allows to exploit thread migration at the API level:
- “body” keeps a thread's user space attributes like associated execution context and stack
- “soul” abstracts a thread's scheduling properties

Mixed-Criticality Scheduler

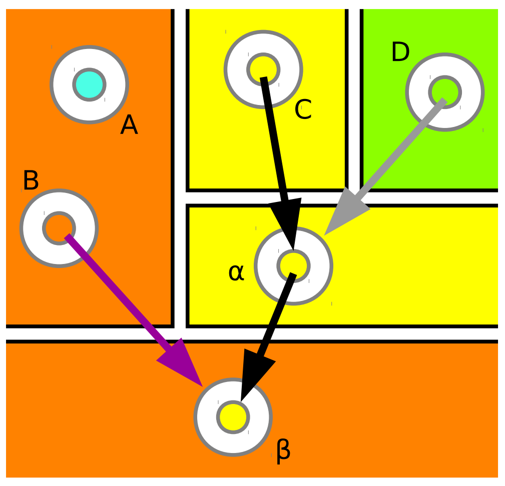
Higher criticality levels require large time budgets due to their more conservative WCET analyses. The emerging gap between theoretical and actual processor utilization is filled by lower critical workload. Additionally, higher critical workloads bind their execution to specific CPU cores, while lower critical and best-effort applications can utilize the remaining processing time on any CPU.

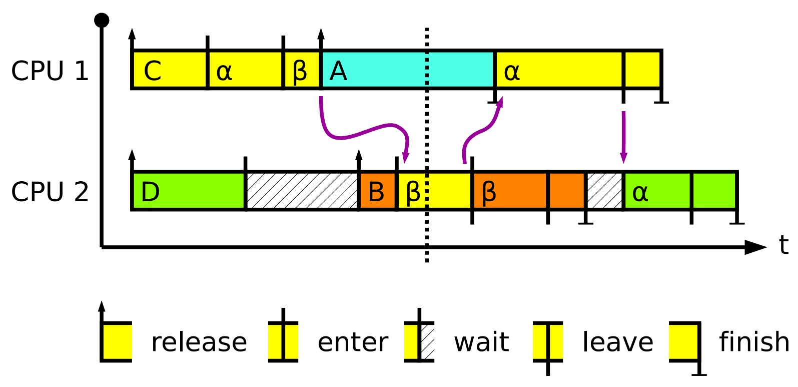
The microkernel scheduler takes care of these different scheduling requirements by mapping the scheduling policies into a single priority space:
- partitioned vs global scheduling Is distinguished by a threshold priority
- inside a priority level, different queuing policies like FIFO or EDF are supported

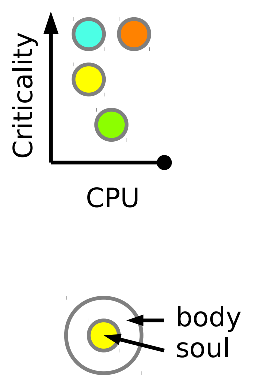


Criticality Inheritance Protocol

Example Scenario:
- C migrates into shared driver α
- D can not enter α and waits
- C migrates to β
- A preempts C
- B can not enter β
- the inheritance protocol pulls C onto B's CPU
- C leaves β, B can enter β
- C continues on its original CPU after A finishes
- C leaves α, D can finally enter α







Wingert OS

Wingert is a German word for vineyard, originating from ancient *wîngarte*, literally *wine garden*.

Wingert OS, or Wiesbaden Next Generation Experimental Real-Time OS, is our operating system.

We target 32 and 64-bit CPUs with MMU.
Userspace:
- Bionic Libc
- OpenMP
- paravirtualized Linux

Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim