µ111MP—Porting the µ111 RTOS on Dual-core Cortex-A7 Processor

M. I. Ben Salah, J.-M. Koller, E. Franzi

Multicore processors are common in high-end embedded system/single board computers, often using Linux as their OS. On the other hand, Multicore Real-Time OSes (RTOS) are uncommon. The goal of this exploratory project is to procure an initial insight about the required effort and technical options to adapt our in-house RTOS, µ111, to a multicore processor. The applications of this new kind of RTOS are manifold: Embedded machine learning inference and high reliability processing are typical examples.

The aim of the project is to develop a multicore version of µ111 named µ111MP running on the STM32MP157 processor. The MP157 embeds a dual core Cortex-A7 for high-end processing and a Cortex-M4 for real-time operation. In this project, all the peripherals are controlled by the Cortex-A7 in a symmetric multiprocessor setup (SMP) with instruction cache enabled.

µ111 is a CSEM RTOS with dynamic priority processes and a preemptive scheduler. The main operating principles of µ111 are the following (example in Figure 1):

- All the processes move between linked lists depending on their state (not initialized, ready, running, suspended, waiting on a resource).
- When needed the scheduler scans the processes that are suspended and determine if they reached a timeout condition and have a higher priority than the current process. If it is the case, a pre-emption signal is sent to the scheduler.
- Each process has a configurable timeout slot after which the scheduler interrupts the process.
- The scheduler is notified of kernel events (process waiting a specific time, waiting on a specific resource such as. mailboxes, semaphores, etc.) through kernel messages which gives the scheduler information about his call.

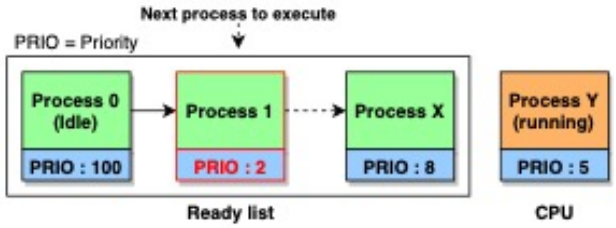

Figure 1: µ111 scheduler selecting next process to run on CPU (lower priority value means higher priority).

The scheduler is split across multiples functions (Figure 2): *sche_chgCxt* handles the scan, moving the processes between the linked-list and the selection of the next process that will be executed. *sche_callbackTrap* analyzes the kernel message to select the linked list on which the scheduler will operate. *temp_testEOTime* checks if any of the suspended processes has reached a timeout. *sche_callbackSlow* wraps arguments to force a preemption when called.

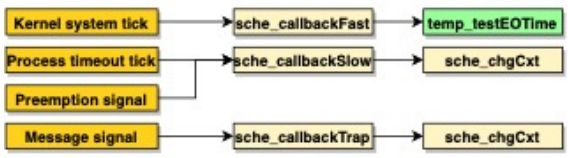

Figure 2: Scheduler call hierarchy.

To provide multiprocessor support the kernel needs to provide inter-CPU locking of shared data structures and consistent shared memory among the CPUs with memory barriers.

Multicore Scheduling: To support a multicore execution the kernel has undergone the following changes (example in Figure 3):

- Processes have a parameter defining on which processor they can run (CPU0, CPU1 or both) called affinity.
- Each CPU has a pre-emption signal to reschedule the highest priority process and a specific timeout timer.
- The kernel system tick interrupt can be handled by any CPU.
- The scheduler can run on any CPU and can interrupt both CPU to reschedule new processes.
- When an interrupt that targets both CPUs is raised, only the first CPU to respond will handle it. The second will identify it as a spurious interrupt and resume the interrupted execution.

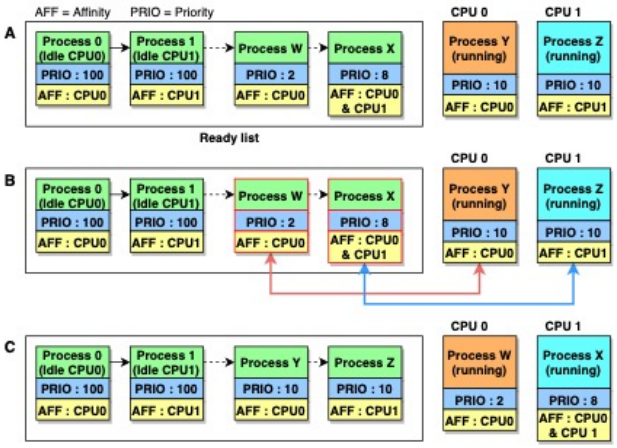

*Figure 3: Example of execution of the µ111MP scheduler. **A:** Scan of the process list that are ready to be run. **B:** Process W and X are with highest priority and the scheduler will send a pre-emption signal on both CPUs. **C**: Both CPUs runs the selected processes.*

Supported kernel primitives: Besides the kernel, a multicore version of µ111 semaphores/mutex has also been implemented and tested. Each kernel primitive will lock the shared global process list when needed, to avoid data races between the CPUs.

Conclusion & next steps: The current version of µ111MP is a proof-of-concept which is tightly linked to the SoC architecture, e.g. by using Hardware Semaphore (HSEM) to lock the shared data structures, which introduces high delays due to wait states as it implemented on another internal bus. The next steps to improve the current implementation are: to enable the data caches and the MMU to handle the cache coherency protocol, to remove HSEM locking and use the ARM specific atomic locking instruction and last but not least to implement a multicore version of the other data structures/services available in µ111 such as mailboxes, signals, memory pool, etc. Another direction of improvement to explore is the development of another kind of schedulers i.e. being cache-aware and using a specific process list per CPU instead of a global process list for both CPUs.