RTEMS-SMP Improvement for LEON multi-core

- Contract No: 4000116175/15/NL/FE/as
- Contractor: embedded brains GmbH (Germany)
- TRP (95k Euro)
- Duration: 12 months (KO: Feb 2016, FR: May 2017)
- TO: M. Verhoef / T. Tsiodras

RTEMS SMP - Ready for Launch

Sebastian Huber

embedded brains GmbH

May 8, 2017

Overview

Topics of this Presentation

- What is RTEMS?
- Overall RTEMS features
- Some RTEMS SMP details

What is RTEMS?

Real-Time Operating System for Multiprocessor Systems (RTEMS)

- Operating system
- Multi-threaded
- Single address-space
- No kernel-space/user-space separation
- Real-time
- Permissive open source license (GPLv2 with linking exception, no obligations for application code)

RTEMS History

1988 RTEMS development started by On-Line Applications Research Corporation (OAR)

- Classic real-time operating system
- $O(1)$ priority scheduler
- Non-transitive priority inheritance
- Priority ceiling

2008 EDISOFT tailors RTEMS 4.8.0 now used in over 20 missions, qualified to DAL-B

2009 Astrium uses of tailored RTEMS 4.6.1 for space applications

2014 Start of Symmetric Multiprocessing (SMP) support development

- Sponsored by ESA with two parallel projects Gaisler/Airbus/OAR and SpaceBel/EB/UoP
- Other RTEMS users

2017 State-the-Art SMP support available as a result of this project (RTEMS 4.12)

- System initialization via constructors
- Scalable timer/timer support
- Giant lock removal
- OMIP implementation

RTEMS Features - SMP Platforms

SMP Platforms

- SPARC
  - GR712RC
  - GR740
- PowerPC
  - QorIQ (e.g. P1020, P2020, T2080, T4240, etc.)
- ARMv7-A
  - Altera Cyclone V
  - Xilinx Zynq
  - Raspberry Pi 2
- Other (ARMv8, RISC-V, x86) - just ask for support

RTEMS Features - APIs

APIs

- Classic
- POSIX (pthreads)
- C11 threads
- C++11 threads
- Newlib and GCC internal
- Futex (synchronization via user-space atomic operations combined with futex system calls)

A broad range of standard software runs on RTEMS

RTEMS Features - Programming Languages/Compiler

Programming Languages

- C/C++/OpenMP (RTEMS Source Builder, RSB)
- Ada
- Google Go
- Fortran (RSB)
- Erlang
- Python and MicroPython

Available Compiler

- GCC (default, best supported and recommended)
- LLVM/clang (works, but currently not available via RSB)
- Other (not out of the box)

RTEMS Features - Devices

Devices

- Termios (serial interfaces)
- I2C (Linux user-space API compatible)
- SPI (Linux user-space API compatible)
- Network stacks (legacy, libbsd, lwIP)
- USB stack (libbsd)
- SD/MMC card stack (libbsd)

libbsd

- Port of FreeBSD user-space and kernel-space components to RTEMS
- Easy access to FreeBSD software for RTEMS
- Support to stay in synchronization with FreeBSD

RTEMS Features - Basic Infrastructure

Basic Infrastructure

- C11/C++11 thread-local storage
- Lock-free timestamps (FreeBSD timecounters)
- Scalable timer and timeout support
- Link-time configuration (RTEMS is a library)
- System initialization via constructors (linker sets, similar to global C++ constructors)

RTEMS Features - Schedulers and Locking Protocols

Clustered Scheduling

- Independent scheduler instances for processor subsets (cache topology)
- Flexible link-time configuration
- Fixed-priority scheduler
- Job-level fixed-priority scheduler (EDF)

Locking Protocols for Mutual Exclusion

- Transitive priority inheritance
- $O(m)$ Independence-Preserving Protocol (OMIP)
- Priority ceiling
- Multiprocessor Resource-Sharing Protocol (MrsP)

What is new?

Symmetric Multiprocessing (SMP) Support for RTEMS

SMP machines consist of a set of processors (players) attached to a common memory (field).

The operating system provides means to ensure fair play.

Why use SMP?

Solve same problem faster - Amdahl's law

$$Speedup(n) = \frac{1}{(1 - p) + \frac{p}{n}}$$

Solve larger problem in the same time - Gustafsons's law

$$Speedup(n) = 1 - p + np$$

Special case: Space and Time Partitioning (TSP)

No reason for SMP

Simplify application development – you use SMP since you must

RTEMS SMP Details

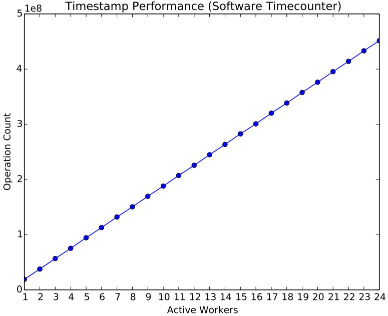
Topics

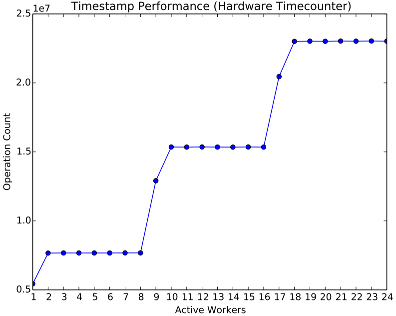
- Timestamps
- Timer/Timeout Support
- System Initialization
- Clustered Scheduling
- Locking Protocols

Plot Data: Testsuite Results

All plots are generated (Python Matplotlib) from data obtained by standard RTEMS testsuite resuls (XML).

Lock-Free Timestamps






```
void worker(void)
{
  while (true) {
    timestamp();
  }
}
```

- Timestamps for uptime and wall clock time
- Port of FreeBSD Timecounters
- Time synchronization via NTP and PPS possible
- Timestamp performance obtained by *SPTIMECOUNTER 2* test program
- Example platform QorIQ T4240 running at 1.5*GHz*
- With software timecounter approximately 79 processor cycles per timestamp

Timer/Timeout Support

Timer

Perform an action at a certain time in the future. Timer usually expire.

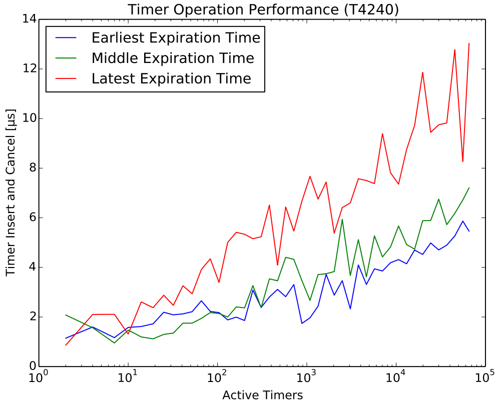
Timeouts

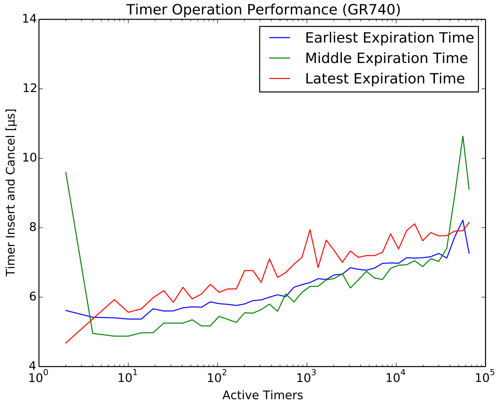
Set time limits to actions. Timeouts hopefully expire rarely.

Timer/Timeout Implementations

- Priority queues (expiration time as key), e.g. red-black tree
  - $O(log(n))$ insert and cancel operations (n active timer count)
  - $O(m \cdot log(n))$ expire operation (m count of timer to expire)
  - Used by RTEMS
- Timer wheel (hash table)
  - $O(1)$ insert and cancel operations
  - Unpredictable expiration operation runtime
  - Used by network stack

Timer Support - Scalable with Active Timer Count





- Timer implementation based on red-black trees
- Timer performance obtained by *TMTIMER 1* test program
- Example platform QorIQ T4240 running at 1.5*GHz* (left)
- Example platform GR740 running at 250*MHz* (right)

Timer Support - Scalable with Processor Count

Per-Processor Timer Maintenance

- Each processor has its own data set to maintain timers
- Thread operation timeouts use current processor
- Timer use dedicated processor set during timer creation

System Initialization via Constructors (1)

Standard System Initialization without Constructors

```
void system_init(void)
{
  init_subsystem_a();
  init_subsystem_b();
  init_subsystem_c();
  init_subsystem_d();
  init_subsystem_e();
}
```

Disadvantage

In case a subsystem is not required by the application, it is still initialized

System Initialization via Constructors (2)

System Initialization via Constructors

```
void system_init(void)
{
  constructor *c = constructor_begin;

  while (c != constructor_end) {
    (*c->init)();
    ++c;
  }
}
```

Subsystem X

```
void subsystem_x_init(void)
{
  /* Some init stuff */
}

REGISTER_CONSTRUCTOR(subsystem_x_init, ORDER_X);
```

Advantage

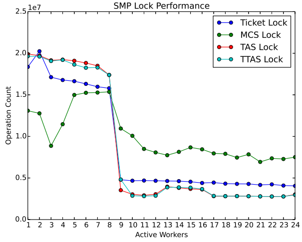
Only subsystems used by the application are initialized and present in the executable

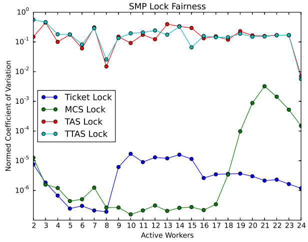
Disadvantage

Requires linker and object file format support

Used by major software systems, e.g. C++, Linux, FreeBSD, etc.

Low-Level Synchronization - SMP Locks



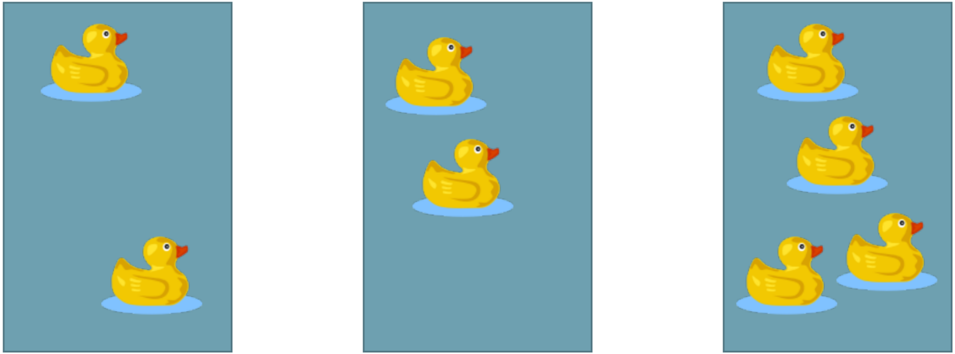


- Several options exist for low-level synchronization in SMP systems
- Test-and-set (TAS)
- Test and test-and-set locks (TTAS)
- Ticket locks
- Mellor-Crummey Scott (MCS) locks
- SMP lock performance obtained by *SMPLOCK 1* test program
- Example platform QorIQ T4240 running at 1.5*GHz*

Basic Requirement: FIFO Fairness

Ticket lock was selected as standard SMP lock for RTEMS SMP

Clustered Scheduling (1)



Clustered Scheduling

Independent scheduler instances for pair-wise disjoint processor subsets

Clustered Scheduling (2)

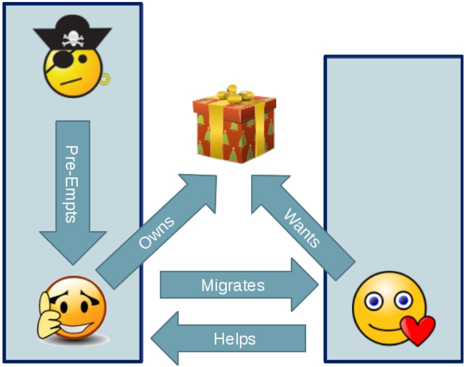
Advantages

- Keep worst-case execution time (WCET) under control: SMP lock FIFO fairness $\Rightarrow$ WCET increases linear with processor count
- Scheduler instances based on cache topology to minimize thread migration overhead (important for priority based schedulers)
- Optimal choice of scheduler algorithms
- Easy implementation compared to schedulers with local run queues and load balancing

Disadvantage

Thread assignment to scheduler instance is a system design decision (bin-packing problem)

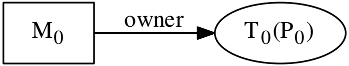
Locking Protocols for Mutual Exclusion (1)



Clustered Scheduling

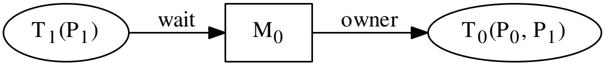
Temporary thread migration is required to minimize latency

Locking Protocols for Mutual Exclusion (2)



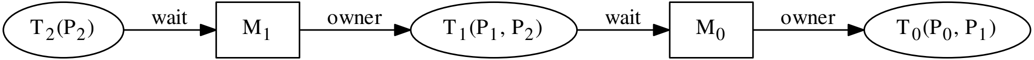
Mutex M_0 with owner thread T_0 (thread priority P_0)

Locking Protocols for Mutual Exclusion (2)



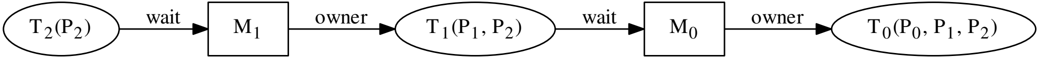
Mutex M_0 with owner thread T_0 and priority inheritance due to waiting thread T_1

Locking Protocols for Mutual Exclusion (2)



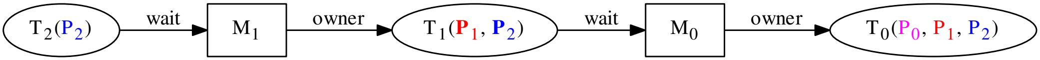
Non-transitive priority inheritance: thread priority P_2 is not propagated to thread T_0

Locking Protocols for Mutual Exclusion (2)



Transitive priority inheritance: thread priority P_2 is propagated to thread T_0 via thread T_1

Locking Protocols for Mutual Exclusion (2)



Transitive priority inheritance and clustered scheduling with three scheduler instances magenta, red and blue

Thread T_0 has access to all three scheduler instances while owning mutex M_0

Implementation Challenge: Fine Grained Locking

Synchonization objects, threads and schedulers have dedicated SMP locks.

Locking Protocols for Mutual Exclusion (3)

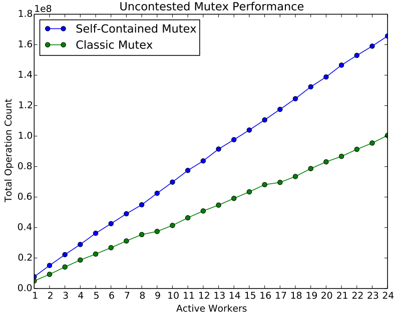
$O(m)$ Independence-Preserving Protocol (OMIP)

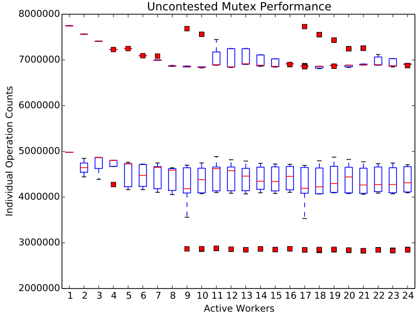
- Generalization of transitive priority inheritance to clustered scheduling
- Suitable for general purpose libraries

Multiprocessor Resource-Sharing Protocol (MrsP)

- Generalization of priority ceiling to clustered scheduling
- User must specify ceiling priorities per scheduler instance
- Protocol design had schedulability analysis in mind

Fine Grained Locking





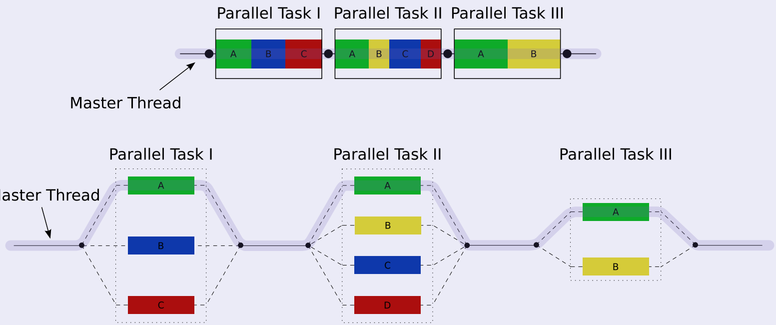

```
void worker(void)
{
  mutex mtx;

  while (true) {
    mtx.acquire();
    mtx.release();
  }
}
```

- Each synchronization object (mutex, message queue, counting semaphore, etc.) has its own SMP lock
- Very important for average case performance
- Mutex performance obtained by *TMFINE 1* test program
- Example platform QorIQ T4240 running at 1.5*GHz*
- Classic API objects are subject to false cache line sharing

OpenMP

OpenMP

- Compiler supported parallelization using a fork-join model



- OpenMP 4.5 support via GCC provided libgomp
- Highly optimized RTEMS configuration of libgomp
- Uses barrier implementation of Linux based on futex system call

Embedded Multicore Building Blocks (EMB^2)/MTAPI

EMB^2

- Set of C/C++ libraries providing:
  - Task management
  - Dataflow
  - Algorithms
  - Containers
- Initially designed for embedded systems
- 2-clause BSD license
- Developed and used by Siemens
- Fully supported by RTEMS

Multicore Task Management API (MTAPI)

- Open source reference implementation contained in the EMB^2
- Custom implementation available from Gaisler

Status and Future Work

Status

- RTEMS SMP is the result of test driven development (RTEMS testsuite contains more than 600 test programs)
- RTEMS 4.12 release is planned for Q2-Q3 2017
- RTEMS SMP is available on the GR712RC and GR740
- Used on Altera Cyclone V, Xilinx Zynq and QorIQ T4240 in production systems

Next Step

Space qualification according to ECSS standards (potential GSTP G617-254SW, maybe available in 2019).

Questions or Lunch?