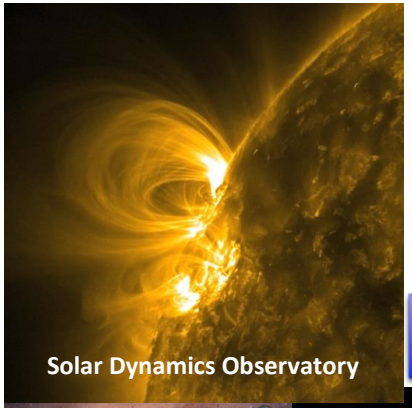
RTEMS Scheduling and SMP

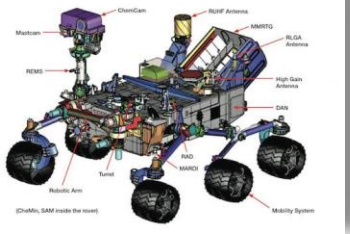
Joel Sherrill, Ph.D.

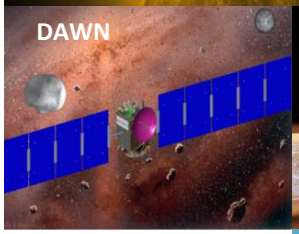
Joel.Sherrill@oarcorp.com
OAR Corporation
Huntsville Alabama USA

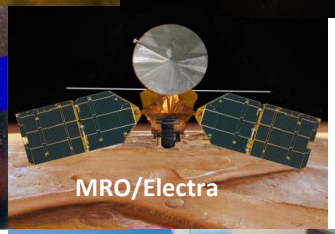
December 2013

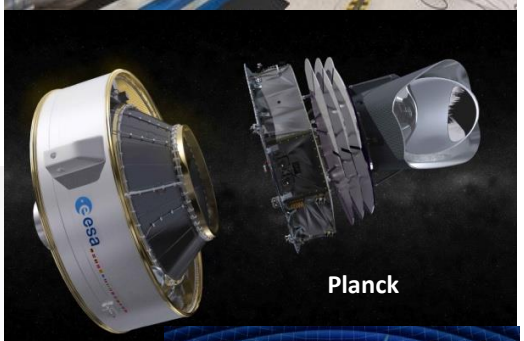
RTEMS Applications

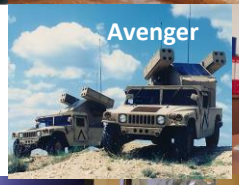


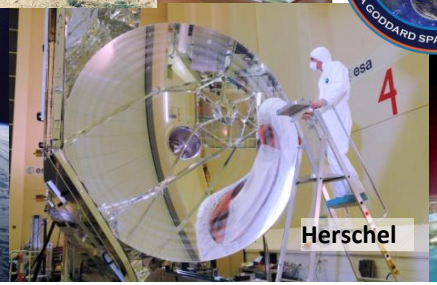


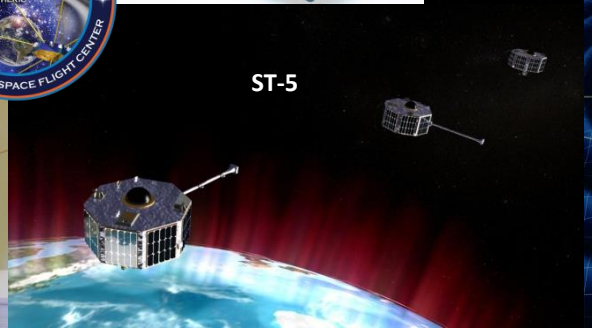


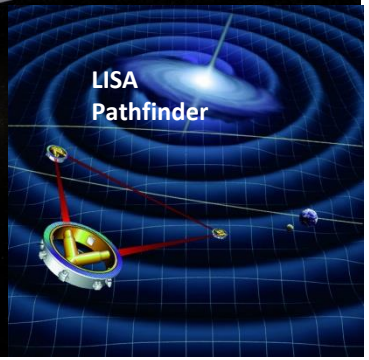












Images Credit: NASA and ESA

Outline

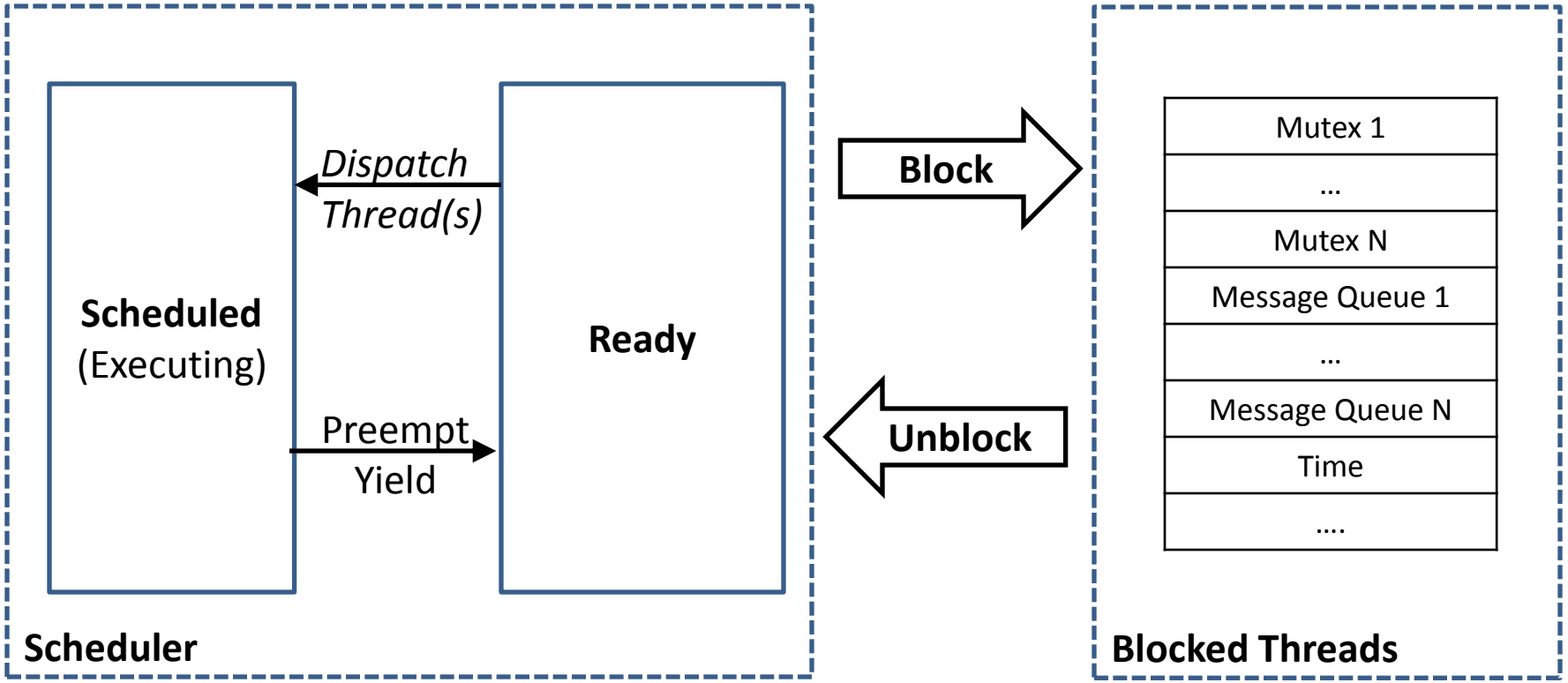
- RTEMS Thread Set Philosophy
- RTEMS Scheduling Algorithms and Framework
- SMP Application Challenges
- Open Research Areas

What does RTEMS Schedule?

- Multiple threads within a single process
- Thread communication and synchronization
  - Semaphores, mutexes, message queues, events, conditional variables, periods, barriers, etc.
- Threads change state and …
  - Multiple algorithms available for scheduling them

The purpose of an RTEMS scheduling algorithm is to select the set of threads which will execute

RTEMS Thread Set View



UP: 1 set of 1
SMP: 1 set of N

UP: 1 set of N
SMP: 1 set of N

UP: M sets of N
SMP: M sets of N

RTEMS Scheduling Framework

- A scheduler is a set of methods which are invoked indirectly at specific points in the system timeline
  - initialization, thread creation, yield, clock tick, blocking, etc.
- One of the many configuration parameters at application compilation/link time is the desired scheduler
- Scheduling algorithm may be selected such that is meets application desired
  - scheduling algorithm behavior
  - time and memory requirements
- Supports user providing unique scheduler WITHOUT modifying RTEMS source

Priority Based Schedulers

The goal of a priority based scheduler is to guarantee that the highest priority set of ready tasks executes

- Allows for the immediate response to external events
- Determines which set of ready tasks allocated to the processors
- Algorithm may additionally take into account preemption and timeslicing for each thread

Deterministic Priority Scheduler (DPS)

- Uniprocessor priority based scheduler
- Array of linked list of TCBs with one list per priority
- Head of lowest array entry with TCBs on list is the highest priority task
- Two level bitmap of priorities which have one or more ready tasks

Optimized for deterministic (e.g. fixed time) execution

Simple Priority Scheduler (SPS)

- Uniprocessor priority based scheduler
- Single linked list of task control blocks (TCBs) ordered by priority
- First task on the ready chain is the highest priority task
- Lower memory footprint than DPS but not deterministic

Text book view of priority scheduling

Earliest Deadline First (EDF) Scheduler

- Optional Uniprocessor Scheduling Algorithm
- Rate Monotonic periods are treated as deadlines
  - deadline == start of next period
- Logically two bands of tasks
  - those with deadlines (e.g. they are periodic)
  - those without deadlines
- Tasks without deadlines are background tasks
- Ready task with the next deadline is selected to run

Constant Bandwidth Server (CBS) Scheduler

- Optional Uniprocessor Scheduling Algorithm
- Extension to EDF Scheduler
  - adds capability to mix sporadic and periodic tasks
- Per period CPU budget associated with each CBS task
  - When budget exceeded, callback invoked
- "QoS" library provided to interact with scheduler
- Fundamental rules
  - Task cannot exceed its registered per period CPU budget
  - Task cannot be unblocked when the time between remaining budget and remaining deadline is higher than declared bandwidth

RTEMS Simple SMP Scheduler

- This is an implementation of a Global Job-Level Fixed Priority Scheduler (G-JLFP)

- Extension of uniprocessor Simple Priority Scheduler (SPS) to multiple cores
- First scheduler implemented as simple

RTEMS Deterministic Priority SMP Scheduler

- This is an implementation of a Global Fixed Priority Scheduler (G-FP)

- Extension of uniprocessor Deterministic Priority Scheduler (DPS) to multiple cores
- Implemented in a manner that shares much code with the Simple SMP Scheduler
  - different data structure for the ready set

Partitioned/Clustered Scheduling

- Set of processors of a system can be partitioned into pairwise disjoint subsets
- Each subset is managed by a scheduler instance
- Necessary to better support mixed OS configurations
- Requires improvements to internal locking

- Reference: Björn B. Brandenburg, Scheduling and Locking in Multiprocessor Real-Time Operating Systems, 2011

Planned for implementation in 2014

Thread Affinity

- Allows the binding and unbinding of a thread to a subset of the cores in a system.
  - The thread will execute only on the designated core(s) rather than on any available core
- Can be used to statically load balance
- Examples
  - Locking I/O threads to a core subset
  - Locking computational threads to a core subset

Currently being implemented

SMP Application Challenges

- Multiple cores creates new opportunities for…
  - true concurrency
  - cost-effective performance improvements
  - threading behavior assumptions to be violated
  - critical section assumptions to be violated

- Multiple threads WILL be running at the same time
- Increases the complexity of analyzing an application
- Uniprocessor embedded applications only had to concern themselves with ISRs and task switches

Race Conditions

SMP means race conditions which never or rarely happened in uniprocessor systems are likely to occur.

- Race conditions in existing code will be exposed more frequently

Highest Priority Task Assumption

When the highest priority task is executing, nothing can externally alter its execution except a hardware interrupt

- Each thread represents an opportunity for the implicit critical section to be violated
- Lower priority threads can run in parallel with higher priority threads

Disable Preemption Assumption

When a thread disabled preemption, it could assume that no other thread would execute until it enabled preemption again

- Each thread represents an opportunity for the implicit critical section to be violated
- Threads continue to execute on other cores despite the disable preemption

Disable Interrupts Assumption

When a thread disabled CPU interrupts, it could assume that no other thread or ISR would execute until it enabled interrupts again

- Altering the interrupt disable mask only impacts the core the thread is executing on
  - interrupts may still occur on other cores
- In addition, those other threads can execute other threads
- Multiple opportunities for the implicit critical section to be violated

Per Task Variables Assumption

A task variable is a set of memory locations that are context switched with the thread

- Usually a pointer to a library's context
- Assumes one memory image and one thread
- With SMP, now there is one memory image and multiple threads
  - the single memory location can't be right for all

Caching Assumptions

Each core has an impact on shared buses. There is interference for both cache and main memory subsystems.

- In uniprocessor systems, only task switches and interrupts disrupted the cache
- In SMP…
  - Cache coherence is critical but can have significant impacts on memory bus bandwidth
  - Execution patterns on other cores can negatively impact each other
- This is a hard problem which will impact system design

Bare Metal SMP Debugging

Debugging SMP systems is much more complicated

- Do the tools meet needs of real users?
  - presentation of per core parallel activities
  - what does it mean to step (e.g. one core, all)
  - SMP aware target resident debug stubs
- Expensive hardware assist debugging devices are likely to be the only option
- Free and open source world has room to improve
- Reference: http://kiwichrisj.blogspot.com.au/

SMP Research/Open Areas

- Practical scheduling algorithms
  - implementable given real world constraints
- Best practices on thread to core assignments
- Debugging aids
- System and application tracing
- Worst case execution analysis
- Synchronization
- Memory Interference

SMP is beginning to be considered for safety critical systems. There are challenges ahead

Conclusion

- RTEMS Thread Set Philosophy
- RTEMS Scheduling Algorithms and Framework
- SMP Application Challenges
- Open Research Areas

Contacts and Acknowledgements

Joel Sherrill, Ph.D.

OAR Corporation
Huntsville Alabama USA
Joel.Sherrill@oarcorp.com

Gedare Bloom, Ph.D.

George Washington University
Washington DC USA
gedare@rtems.org

Chris Johns

Contemporary Software
Sydney Australia
chrisj@rtems.org

Backup Slides

RTEMS Project Participation In Student Programs

- Google Summer of Code (2008-2013)
  - Almost fifty students over the six years
- Google Code-In (2011-2013)
  - High school students did RTEMS ~200 tasks (2012)
  - Included only ten FOSS projects in 2012 and 2013
- ESA Summer of Code In Space (2011-2013)
  - Small program with only twenty FOSS projects involved

Scheduling Mechanisms

- Priority
- Preemptive
- Timeslicing
- Manual Round-Robin
- Rate Monotonic

Priority

- User assigned on a task-by-task basis
- 256 priority levels supported by SuperCore
  - SuperCore 0 is most important, 255 is least
  - Classic API priority levels of 1 – 255 with 1 being highest
  - POSIX API priority levels of 1 – 255 with 1 being lowest
- Round-robin within priority group
- Allows application control over distribution of processor to tasks

IDLE task is SuperCore priority 255 and never yields!
Do NOT use this priority level via ANY API

Preemption

- A task's execution is interrupted by another task
- Higher priority tasks interrupt the execution of a lower priority task
- Task mode - preemption
  - When enabled, task may be preempted by a higher priority ready task
  - When disabled, task must voluntarily give the processor up

- Classic API tasks are preemptible by default
- POSIX API threads are preemptible by default

Timeslicing

- Limits task execution time (processor time allocated to the task)
- Fixed time quantum per timeslice which is user configurable
- Task mode - timeslicing
  - Disabled - unlimited execution time
  - Enabled - limited execution time

- Classic API task's timeslice allotment is reset each time it is context switched in
- POSIX API thread may use this algorithm or not have it's allotment reset until it has been consumed

Timeslicing Scheduling Actions

- Upon expiration of the timeslice
  - Another task of the same priority is given to the processor
  - Immediate reallocation of processor when only task of priority group

- Priority and preemption will affect timesliced tasks

Manual Round-Robin

- Voluntary yielding of the processor
- Immediate removal from the processor
- Place at the end of ready chain priority group
- Task will not lose control of the processor when no other tasks at this priority are ready

Rate Monotonic

- Rate monotonic scheduling algorithm ensures that all threads are schedulable if requirements are met
  - based on using periodic threads
- Assumes thread priorities assigned per Rate Monotonic Priority Assignment Rule
  - higher rate threads are more important
- A thread set is said to be *schedulable* if all threads in that set meet their deadlines

RTEMS Scheduling Simulator

- Host based tools using subset of RTEMS source code
- Executes scripts which exercise scheduling algorithm based on predictable events
- Useful for …
  - debugging new algorithms
  - testing with varying core quantities
- Very deterministic and does not require target hardware

DPS Two Level Priority Bitmap

Major Bitmap

Bit Number	Priorities
0	0 - 15
1	16 – 31
2	32 – 47
3	48 – 63
4	64 – 79
5	80– 95
6	96 – 111
7	112 - 127
8	128- 143
9	144 – 159
10	160 – 175
11	176 – 191
12	192 – 207
13	208 – 223
14	224 – 239
15	240 - 255

Minor Bitmap Array

Array Index	Individual Bits for Priorities
0	0 - 15
1	16 – 31
2	32 – 47
3	48 – 63
4	64 – 79
5	80– 95
6	96 – 111
7	112 - 127
8	128- 143
9	144 – 159
10	160 – 175
11	176 – 191
12	192 – 207
13	208 – 223
14	224 – 239
15	240 - 255

DPS Priority Bitmap Example

- Threads at priorities 1 and 255
  - Priority 1 has major 0 and minor 1
  - Priority 255 has major 15 and minor 15

- Major Bitmap:
  - 0x8001
- Minor Bitmap Array:
  - {0x0002, 0x0000 , 0x0000 , 0x0000,
    0x0000, 0x0000 , 0x0000 , 0x0000,
    0x0000, 0x0000 , 0x0000 , 0x0000,
    0x0000, 0x0000 , 0x0000 , 0x8000}

Framework Plugin Points – Initialization Support

- Initialize the scheduling algorithm
  - void (*initialize)(void);
- Scheduler per thread information allocation/deallocation
  - void * (*allocate)(Thread_Control *);
  - void (*free)(Thread_Control *);

Framework Plugin Points – Primary Thread Operations

- Remove thread from scheduling decisions
  - void (*block)(Thread_Control *);
- Add thread to scheduling decisions
  - void (*unblock)(Thread_Control *);
- Extract a thread from the ready set
  - void (*extract)(Thread_Control *);
- Voluntarily yields the processor per the scheduling policy
  - void (*yield)(Thread_Control *thread);

Framework Plugin Points Priority Change Support

- Update scheduler cached information per thread
  - void (*update)(Thread_Control *);
- Perform the scheduling decision logic (policy) when required
  - void (*schedule)(void);
- Enqueue a thread into its priority group
  - void (*enqueue)(Thread_Control *);
  - void (*enqueue_first)(Thread_Control *);

Framework Plugin Points

- Compares two priorities
  - int (*priority_compare)(Priority_Control, Priority_Control);
- Invoked upon release of a new job. Supports deadline based schedulers
  - void (*release_job) (Thread_Control *, uint32_t);
- Perform scheduler update actions required at each clock tick
  - void (*tick)(void);
- Starts the idle thread for a particular processor.
  - void (*start_idle)(Thread_Control *thread, Per_CPU_Control *processor);