Scheduling and Thread Management with RTEMS

Joel Sherrill, Ph.D.

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

Gedare Bloom, Ph.D.

gedare@gwu.edu
George Washington University
Washington DC USA

August 2013

RTEMS Applications

Images Credit: NASA and ESA

Outline

- Real-Time Embedded Systems Introduction
- RTEMS Overview
- RTEMS Thread Set Management
- RTEMS Thread Scheduling
- RTEMS Scheduling Simulator
- SMP Application Challenges
- Open Areas

Real-Time Embedded Systems Background

Real-Time?

- In computing, the term real-time means …
  - *the correctness of the system depends not only on the logical result of the computations but also on the time at which the results are produced*
- Examples include:
  - Flight control systems, robotics, networking and telecom equipment, traffic signals

Predictable is better than fast and variable

Embedded System?

- An embedded system is …
  - *any computer system which is built into a larger system consisting of multiple technologies such as digital and analog electronics, mechanical devices, and sensors*
- Examples include:
  - Consumer Electronics, Control Systems, Routers

Anything that contains a computer
where it isn't apparent to the user there is a computer

Operating System?

- An operating system is …
  - *a collection of software services and abstraction layers that enable applications to be concerned with business logic rather than the specifics of device control and protocol stacks.*
- Enables application portability across hardware platforms

An operating system is a Master Control Program (MCP)

Real-Time Operating System?

- A real-time operating system is …
  - an operating system designed to provide applications services which assist in meeting application requirements which including timing.
- Timing requirements are typically related to interaction with external devices, other computers, or humans.

OS is predictable and
aids in the construction of predictable applications

Real-Time Embedded Systems

- Include timing requirements
- Size, Weight, and Power (SWaP) Concerns
- Hardware is often custom
- Safety and/or reliability are concerns
- Debugging is often difficult
- User interface not present or non-standard
- Interact with environment

We interact with real-time embedded systems on a daily basis

Real-Time/Embedded Systems

- Automotive control
  - engine control
  - active suspension
  - anti-lock brakes
- Cell phone
- Industrial control
- Telecom switch
- GPS

- Consumer electronics
  - smart televisions
  - set top boxes
  - DVD/Blu-ray
  - MP3 player
- Air traffic management
- Traffic signals
- Medical devices

RTEMS is designed for deeply embedded systems

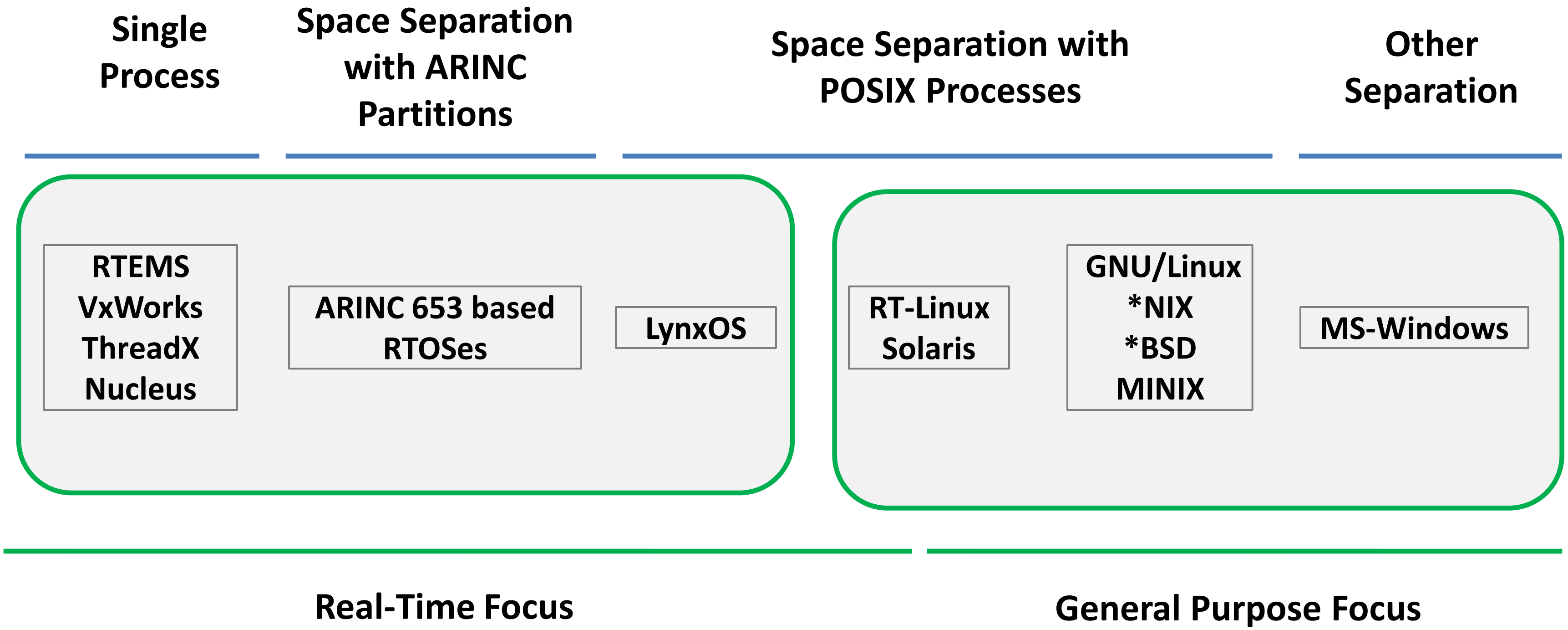
RTEMS

Real-Time Executive for Multiprocessor Systems

RTEMS in a Nutshell

- RTEMS is an embedded real-time O/S
  - deterministic performance and resource usage
- RTEMS is free software
  - no restrictions or obligations placed on fielded applications
- Supports open standards like POSIX
- Available for 15+ CPU families and ~175 BSPs
- Community driven development process
- Training and support services available

RTEMS in OS Spectrum



Single Process

Space Separation with ARINC Partitions

Space Separation with POSIX Processes

Other Separation

RTEMS
VxWorks
ThreadX
Nucleus

ARINC 653 based RTOSes

LynxOS

RT-Linux
Solaris

GNU/Linux
*NIX
*BSD
MINIX

MS-Windows

Real-Time Focus

General Purpose Focus

Does not address hypervisors or embedded OSes which are not focused on real-time

RTEMS Features at a Glance

- RTEMS is a Commercial Grade Real-Time Operating System
- Truly free in price, liberty, and end user requirements
  - All source code for OS, support components, tests, documentation, and development environment
  - Test coverage is openly reported
- High performance with deterministic behavior
- Low overhead with predictable resource consumption
  - Full executables currently as small as 16K
- Highly configurable with unused features left out by linker

More RTEMS Features

- POSIX/Single Unix Specification Support
  - Single process, multi-threaded
- Classic API with hard real-time features
- Multiple file systems (e.g. FAT, NFS, RFS, IMFS)
- FreeBSD TCP/IP w/httpd, ftpd and telnetd
- FreeBSD USB w/removable mass storage
- Customizable shell with ~100 commands

Advanced RTEMS Features

- Symmetric Multiprocessing (SMP)
  - SPARC, ARM, x86, and PowerPC
- Asymmetric and Distributed Multiprocessing
- Priority Inversion Avoidance Protocols
  - Priority Inheritance Protocol
  - Priority Ceiling Protocol
- Statistics on stack usage, CPU usage, and periods

RTEMS Community Driven Focus

- Without users, project has no reason to exist!
- Users drive requirements
  - Please let us know what you need
- Users provide or fund many improvements
  - Again those reflect your requirements
- Most bug reports are from users

RTEMS Evolves to Meet User Needs

RTEMS Project Participation In Student Programs

- Google Summer of Code (2008-2013)
  - Over 40 students over the six years
- Google Code-In (2011-2013)
  - High school students did ~200 tasks for RTEMS
  - Included only ten FOSS projects in 2012-13
- ESA Summer of Code In Space (2011-2013)
  - Small program with only twenty FOSS projects involved

rtems

Real-Time Executive for Multiprocessor Systems

Architectures Supported by RTEMS

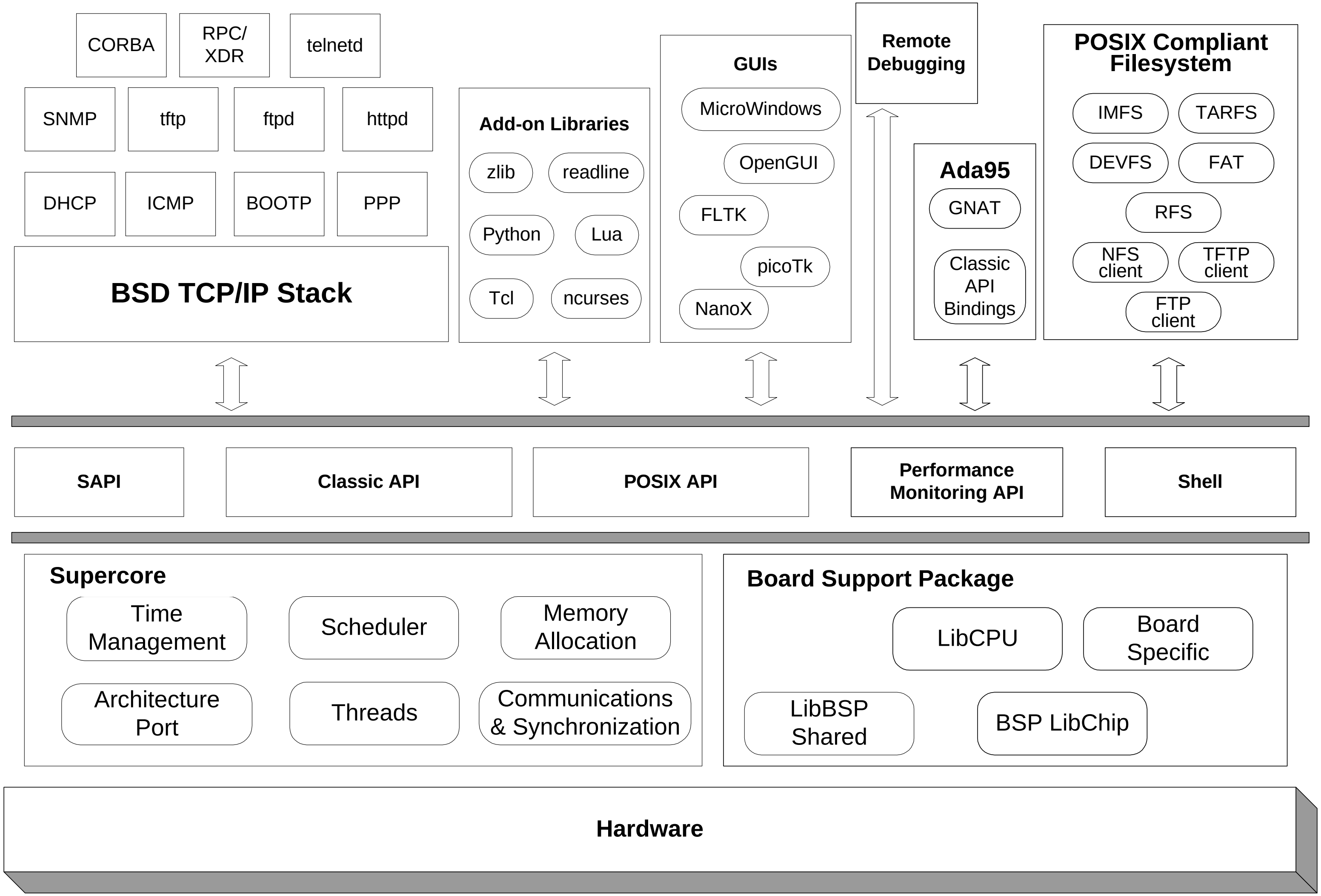
Architecture	4.6	4.7	4.8	4.9	4.10	Dev
Altera NIOS II	No	No	No[1]	No[1]	No[1]	Yes
ADI Blackfin	No	No	Yes	Yes	Yes	Yes
ARM with many CPU models	Yes	Yes	Yes	Yes	Yes	Yes
ARM/Thumb	No	No	No	Yes	Yes	Yes
Atmel AVR	No	Partial	Partial	Partial	Partial	Partial
AMD A29K	Yes	No	No	No	No	No
HP PA-RISC	Yes	No	No	No	No	No
Intel/AMD x86 (i386 and above)	Yes	Yes	Yes	Yes	Yes	Yes
Intel i960	Yes	No	No	No	No	No
Lattice Semiconductor Micro32	No	No	No	No	Yes	Yes
Microblaze	No	No	No	No	No	Partial
MIPS including multiple ISA levels	Yes	Yes	Yes	Yes	Yes	Yes

Architectures Supported by RTEMS

Architecture	4.6	4.7	4.8	4.9	4.10	Dev
Moxie	No	No	No	No	No	Yes[1]
Freescale MC680xx and MC683xx	Yes	Yes	Yes	Yes	Yes	Yes
Freescale Coldfire	Yes	Yes	Yes	Yes	Yes	Yes
OpenCores OR32	Yes	No	No	No	No	No
PowerPC	Yes	Yes	Yes	Yes	Yes	Yes
Renesas H8/300	Yes	Yes	Yes	Yes	Yes	Yes
Renesas M32C	No	No	No	No	Partial	Partial
Renesas M32R	No	No	No	No	Partial	Partial
Renesas SuperH (SH1 through SH4)	Yes	Yes	Yes	Yes	Yes	Yes
SPARC including ERC32 and LEON	Yes	Yes	Yes	Yes	Yes	Yes
SPARC64	No	No	No	No	No	Yes
TI C3x/C4x	Yes	No	No	No	No	No
NEC V850	No	No	No	No	No	Yes

(1) Moxie support is complete but pending merger to main repository

RTEMS Architecture



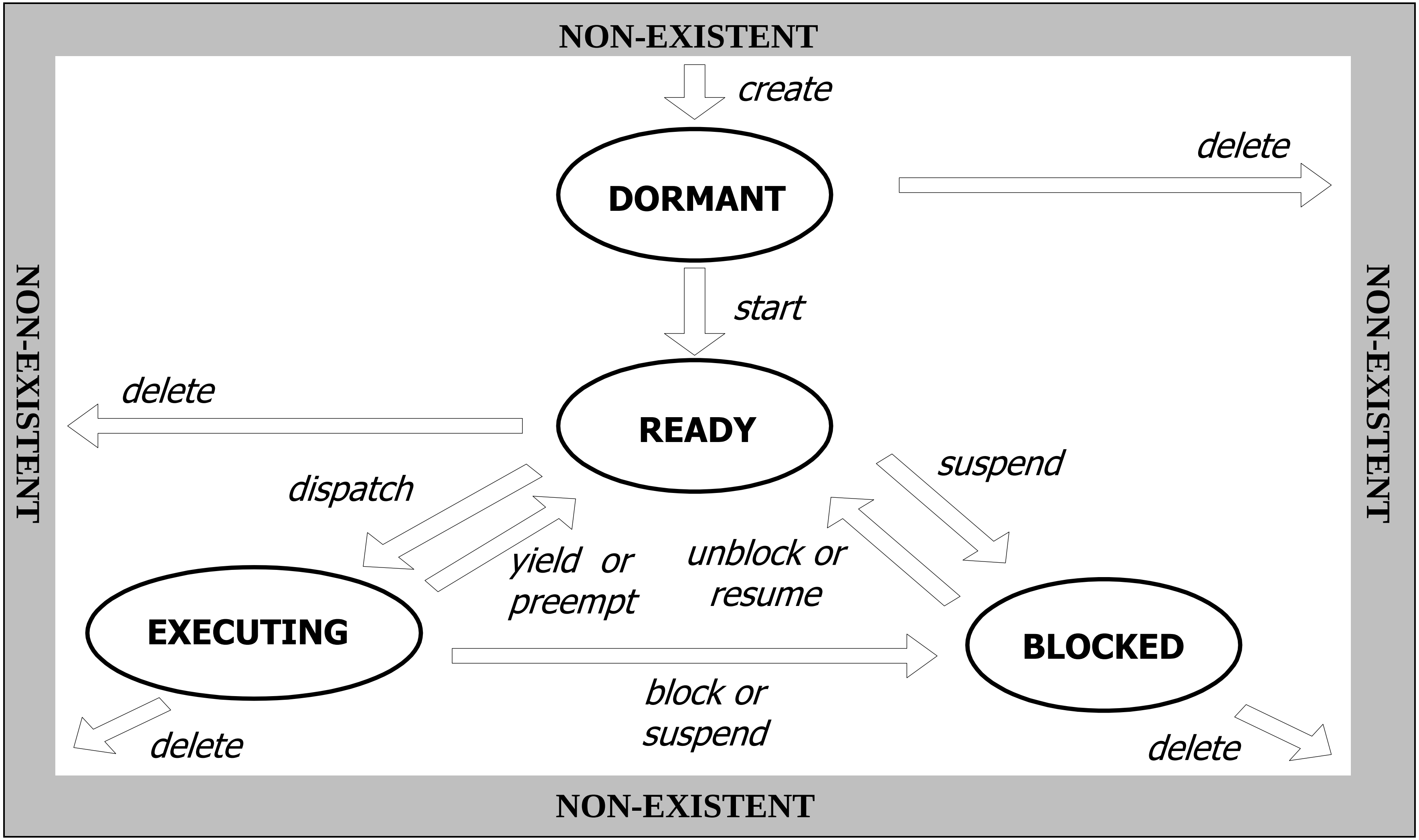
What does RTEMS Schedule?

- 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

Embedded With RTEMS™ www.rtems.org

Thread State Diagram



NON-EXISTENT

create

DORMANT

delete

start

NON-EXISTENT

NON-EXISTENT

delete

READY

suspend

dispatch

yield or preempt

unblock or resume

EXECUTING

BLOCKED

block or suspend

delete

delete

NON-EXISTENT

Non-existent State

- State before a task is created
- State after a task is deleted

- Internally, the TCBs are all preallocated at initialization time but they are not associated with a user task.
- During the creation process, TCBs are associated with stacks and user application code

Dormant State

- State after a task is created
- State before a task is started
- Cannot compete for resources

- In the Classic API, creating a task transitions that task from non-existent to dormant
- In the POSIX API, a thread is created and started as part of its initialization process and thus is never in the dormant state from a user’s perspective

Ready State

- State after a task is started
- State after a blocked task is readied
- State after the processor is yielded
- Task may be allocated to the processor
- Scheduler considers the set of ready tasks the candidates for execution

Blocked State

- State after a blocking action is taken
- Task may not be allocated to the processor
- The task is blocked waiting for a readying action such as
  - Resume
  - Timeout
  - Release of resource
  - Delay ended

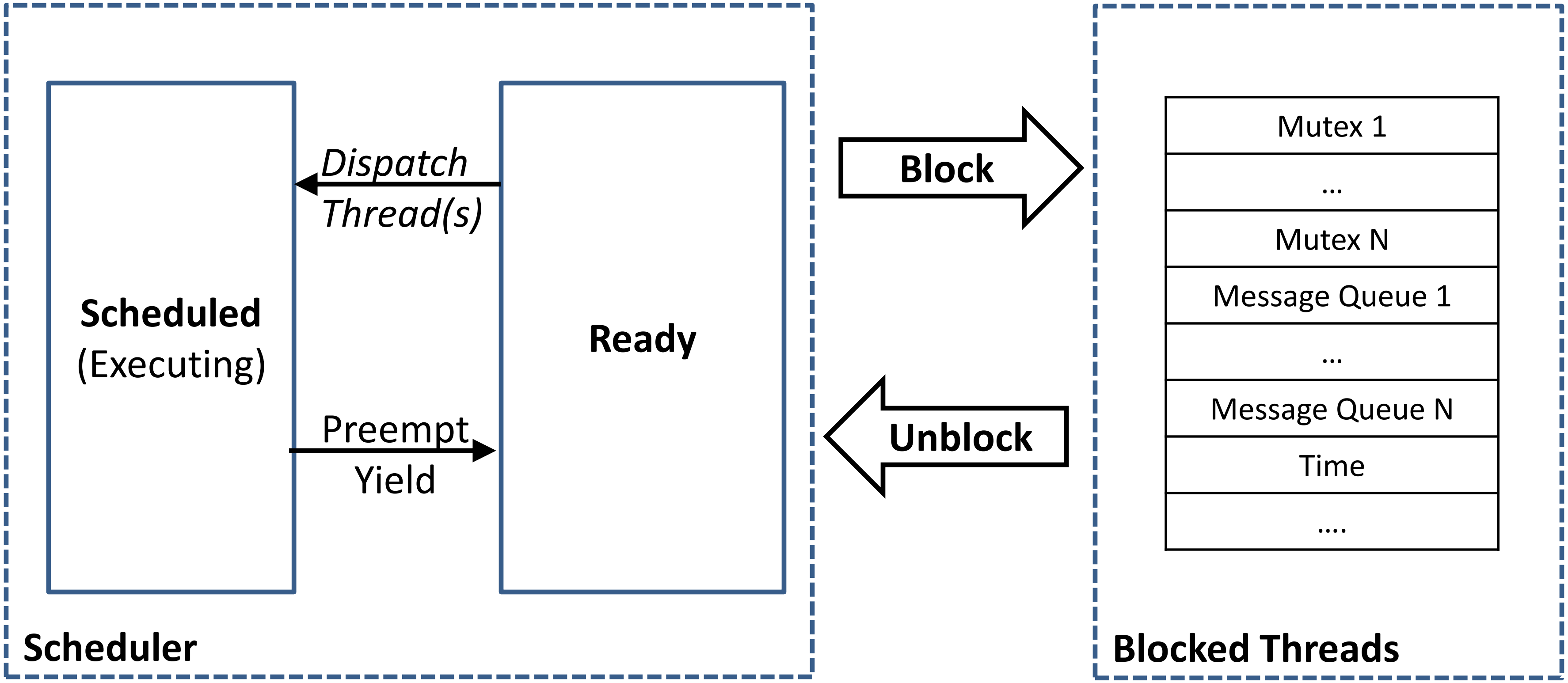
Executing State

- State of task when allocated the processor
- Selected from set of ready tasks based upon current scheduling criteria
- RTEMS internal variable
  - ***_Thread_Executing*** points to the TCB of the currently executing thread
- After 4.10:
  - ***_Thread_Executing*** is now a macro
  - SMP support means one executing thread per core

RTEMS Thread Scheduling

- Logically RTEMS just manages sets of threads
  - one or more is executing
  - one or more is ready
  - one or more is waiting for a resource or time
- Application can set various parameters at run-time on a per thread basis
  - priority
  - preemption
  - …

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

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

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 ready task(s) are allocated to the processor
- 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
- Bit map of priorities which have one or more ready tasks

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

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}

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 Job-Level Fixed Priority Scheduler (G-JLFP)
- 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

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

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);

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

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.

- With there are multiple cores and it should be assumed that a thread is executing on each of those threads
- Race conditions in existing code will be exposed

Highest Priority Task Assumption

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

- With multiple cores, again it should be assumed that a thread is executing on each of those threads
- Each thread represents an opportunity for the implicit critical section to be violated

Disable Preemption Assumption

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

- With multiple cores, again it should be assumed that a thread is executing on each of those threads
- Each thread represents an opportunity for the implicit critical section to be violated

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 memory caching

- 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

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

Conclusion

- Introduction to Real-Time Embedded Systems
- Overview of RTEMS
- RTEMS Thread Scheduling
- RTEMS Thread Set Management
- Challenges and Opportunities

Contact Information

Joel Sherrill, Ph.D.

OAR Corporation
Huntsville Alabama USA

Joel.Sherrill@oarcorp.com

Gedare Bloom, Ph.D.

George Washington University
Washington DC USA

gedare@gwu.edu