Resource Sharing in RTSJ and SCJ Systems

Andy Wellings, Charlie Lin and Alan Burns

Structure

- Motivation and Background
- Single Processor Resource Control
- Multiprocessor Policies
- Summary of Protocols
- Application to RTSJ and SCJ
- Nested Resources and Deadlock
- Adding Flexibility

Motivation and background

- RTSJ Version 1.1 provides more explicit support for multiprocessor systems
- Global, partitioned and cluster scheduling are all allowed
- Resource sharing is still largely unresolved
- Consider
  - current literature in RT resource sharing
  - impact this could have on the specification

Single Processor Resource Control

- Well understood:
  - Priority inheritance
  - Priority ceiling protocols
  - Non preemptive critical sections
  - Stack resource policy
- Usual assumption:
  - No self suspension holding a resource
  - *Not enforced by Java*

Multiprocessor Policies

- Multiprocessor Priority Ceiling Protocol (MPCP)
- Distributed Priority Ceiling Protocol (DPCP)
- Multiprocessor Stack Resource Policy (MSRP)
- Flexible Multiprocessor Locking Protocol (FMLP)
- Parallel Priority Ceiling Protocol (PPCP)
- O(M) Locking Protocol (OMLP)

Summary of Protocols

	Scheduling	Global/Local Resources	Nested Resources	Access Priority	Queuing
MPCP	Partitioned	Yes	No	Ceiling (priority boosting)	Suspends (priority queue)

Summary of Protocols

	Scheduling	Global/Local Resources	Nested Resources	Access Priority	Queuing
MPCP	Partitioned	Yes	No	Ceiling (priority boosting)	Suspends (priority queue)
DPCP	Partitioned	Yes	Yes	Ceiling (priority boosting)	Suspends (priority queue)

Summary of Protocols

	Scheduling	Global/Local Resources	Nested Resources	Access Priority	Queuing
MPCP	Partitioned	Yes	No	Ceiling (priority boosting)	Suspends (priority queue)
DPCP	Partitioned	Yes	Yes	Ceiling (priority boosting)	Suspends (priority queue)
MSRP	Partitioned	Yes	No	Non Preemptive	Spins (FIFO queue)

Summary of Protocols

	Scheduling	Global/Local Resources	Nested Resources	Access Priority	Queuing
MPCP	Partitioned	Yes	No	Ceiling (priority boosting)	Suspends (priority queue)
DPCP	Partitioned	Yes	Yes	Ceiling (priority boosting)	Suspends (priority queue)
MSRP	Partitioned	Yes	No	Non Preemptive	Spins (FIFO queue)
FMLP	Partitioned and Global	No	Group Locks	Short : Non preemptive Long: Inheritance	Short: Spins (FIFO queue) Long: Suspends (priority queue)

Summary of Protocols Cont.

	Scheduling	Global / Local Resources	Nested Resources	Access Priority	Queuing
PPCP	Global	No	No	Inheritance	Suspends (priority queue)

Summary of Protocols Cont.

	Scheduling	Global / Local Resources	Nested Resources	Access Priority	Queuing
PPCP	Global	No	No	Inheritance	Suspends (priority queue)
OMLP	Partitioned and Global	Yes	Group Locks	Global: inheritance Partitioned: preemptive	Suspends (in token contention and priority queue) Suspends (in FIFO and priority queue)

Summary of Protocols Cont.

	Scheduling	Global/Local Resources	Nested Resources	Access Priority	Queuing
PPCP	Global	No	No	Inheritance	Suspends (priority queue)
OMLP	Partitioned and Global	Yes	Group Locks	Global: inheritance Partitioned: preemptive	Suspends (in token contention and priority queue) Suspends (in FIFO and priority queue)
Cluster OMLP	Clustered	No	Group Locks	Priority donation	Suspends (FIFO queue)

Application to RTSJ and SCJ

RTSJ

- Allows self suspension
- Nested resources allowed
- Clusters allowed
- A single approach not possible?

SCJ

- Does not allow self suspension
- Nested resources allow
- Level 1: Partitioned only
  - DPCP
  - but how to do migration?
- Level 2: Clusters allowed
  - Cluster OMLP?
  - But how to deal with nested resources?

Nested Resources and Deadlock

- For deadlock to occur
  - mutual exclusion
  - hold and wait
  - no preemption
  - a circular chain
- Dealing with deadlock
  - deadlock prevention
  - deadlock avoidance (Group locks)
  - deadlock detection and recovery

Ceiling Priorities and Deadlock

- On a single processor (deadlock prevention)
  - Ceiling of nested resource must be greater than ceiling of calling resource
  - breaks the circular wait
- Priority used for
  - execution eligibility
  - preemption control
  - resource ordering

EDF and deadlock with SRP

- Deadline is used for execution eligibility (dynamic priority)
- Preemption levels used for
  - preemption control
  - ordering

Multiprocessors

- Often priority is used to get non-preemption
- Therefore need to separate out order property
- Is their a GlobalPriorityCeilingEmulation protocol?
  - Local resources: usual priority ceiling emulation
  - Global: non preemptive, order attribute ensures no circular chains
  - But introduces transitive block chains

Adding Flexibility

- It seems a single monitor control policy will not fit all multiprocessor applications
- Can obviously add a GlobalPriorityCeilingEmulation policy

```
package java.realtime;
public class GlobalPriorityCeilingEmulation extends
             PriorityCeilingEmulation {
  public int getPartialOrder();
  public static GlobalPriorityCeilingEmulation
               instance(int partialOrder);
}
```

Global Priority Ceiling Emulation

```
package java.realtime;
public class GlobalPriorityCeilingEmulation extends
             PriorityCeilingEmulation {
  public int getPartialOrder();
  public static GlobalPriorityCeilingEmulation
               instance(int partialOrder);

  public interface LockPolicy {};
  public interface QueueOrder {};
  public static LockPolicy Spin;
  public static LockPolicy Suspend;
  public static QueueOrder Fifo;
  public static QueueOrder Priority;
  public void setQueuePolicy(LockPolicy l);
  public void setQueueOrder(QueueOrder o);
  public void setQueueLength(int l);
}
```

User-defined Locking

- Would give greater flexibility; JVM delegates locking to application

```
package javax.realtime;
public abstract class MonitorControl {
  ... // as before
  protected void lock();
  protected void unlock();
  protected void await();
  protected void signal();
  protected void signalAll();
}
```

Conclusions

- RTSJ V1.1 will provide more explicit support for developing multiprocessor systems
- The lack of standardization in the area of resource control protocols has resulted in simple priority inheritance being adopted as the main monitor control policy
- The current draft SCJ standard adopts priority ceiling emulation and assumes the programmer will set appropriate ceilings

Conclusions RTS

- Cannot standardize on a single policy due to the freedom given in Java and RTSJ
- Nested global calls and the ability to suspend inside a monitor whilst holding the monitor lock undermine the state of the art
- Consequently, more flexibility is required within the RTSJ to allow a developer to program their own resource control policies

Conclusions SCJ

- Appropriate to define a conservative model that is well understood: SCJ already supports a restrictive programming model
- A resource control protocol based on the Global Priority Ceiling Emulation could be used
- To avoid deadlocks when accessing nested resources, a partial order must be defined for nested global resource accesses
- However, unlike the single processor PCE protocol, transitive blocking is not prevented