The Resource Reservation Paradigm for Real-Time Systems

Raj Rajkumar
Professor, ECE, CS & CyLab
Director, Real-time and Multimedia Systems Laboratory
Co-Director, GM-CMU Collaborative Research Laboratory
Carnegie Mellon University
raj@ece.cmu.edu
http://www.cs.cmu.edu/~rajkumar

Outline

• Real-Time System Development Issues
• The Resource Kernel (RK) approach
  – CPU reservations
  – Disk Bandwidth reservations
  – Network Bandwidth reservations
• Power-Aware Resource Kernels
• RK in Wireless Sensor Networks
• Summary and Future Challenges
Real-Time Systems Are ……..

Carnegie Mellon University

- **Network elements & Appliances**
  - Network switches, DVD players, set-top boxes, smart phones, printers, ...
- **Interactive & on-demand applications**
  - Gaming consoles, video-conferencing, ...
- **Transportation systems & vehicular control**
  - Avionics, air traffic control, space shuttle/space station, shipboard control, automobiles, satellites, ...
  - Motion control in general
- **Process control & manufacturing**
  - Power generation, petroleum refinement, chemical processing, semiconductor fabs, automated assembly, bottling, packaging, ...
- **Aerospace and Defense systems**
  - Navigation systems, global positioning, distributed command and control, weapon systems, ...

Building Real-Time Systems

Carnegie Mellon University

**Real-time Community Perspective:**
- “*Time is part of the system’s correctness – therefore, need scheduling policy and resource management at the core*."

**Result:**
- RMS, DMS, EDF, ..., Resource Sharing
- Maximizing resource utilization

**Systems Developer Perspective: Balance multiple needs**
- What coding practices do we need?
  - What must each developer know? Does (s)he need to have a PhD?
- How do we test, verify and validate?
- How do we integrate?
  - How do we ensure that subsystems developed by geographically distributed teams work together?
  - How do we ensure that a fault in one subsystem does not affect other systems (and indeed the entire system)?
- What is available commercially?
  - Network switches, system buses and backplanes do not support dynamic scheduling policies like EDF
- What do industry standards support?
Bringing the Two Sides Together

• The real-time community takes “Scheduling policies” too literally
  – they are policies for resource management, but mechanisms for system construction
• Scheduling priorities are not application-level requirements, but OS-internal mechanisms for arbitration
  – Application-level importance is a user-level attribute that the OS cannot decide.
• Address space partitioning with MMU support has proven itself
  – Each process is ‘guaranteed’/given its own dedicated address space.
  – Logical failures in any one process cannot affect another process.
  – Very few programmers need to understand how the MMU/Virtual memory system actually works.
    • Sharing mechanisms are also available?
• Can we have a notion similar to address space for capturing time?
  – Each ‘application’ is guaranteed its own ‘temporal’ space
  – The timing behavior of a process cannot affect that of another process
  – Test each application locally – not all possible combinations with other and future applications

Resource Kernel (RK)

• A Kernel that provides to applications Timely, Guaranteed, and Enforced access to System Resources
• Allows Applications to specify only their Resource/Timing demands
  – leaving the Kernel to satisfy those demands using (hidden) resource management schemes

Reservation Parameters

“T”: Period (1/f)
“C”: Execution time within period
“D”: Deadline within period

Apps
- Real-Time and Multimedia Applications
- Publisher/Subscriber Services
- QoS Mgr
- Real-Time Java
- RT Filesystem

Middleware Services

Resource Kernel

Physical resources
- Disk BW
- CPU
- Memory

Linux/RK Architecture

User-Level

Kernel
- Resource Kernel
- Loadable Kernel Module

Hardware
Linux/RK Abstractions

Linux/RK supports several abstractions and primitives for real-time scheduling of processes with real-time and QoS requirements:

- **Resource reservations with latency guarantees**
  - CPU cycles
  - Network bandwidth
  - Disk bandwidth
  - Physical memory (spatial reservation)
- Support for **periodic tasks**.
- Support for 256 real-time fixed-priority levels.
- **High-resolution timers and clocks**.
- **Bounding of priority inversion** during synchronization operations.
- Wiring down of memory pages.

Reservation Types

**Hard Reservations** (guarantees with **No** extras even if resource is idle)

**Firm Reservations** (guarantees with Extras only if no non-real-time)

**Soft Reservations** (guarantees with extras)
Hierarchical Reservations

- Child reservations can be created inside parent reservations (recursively)
  - A "default reserve" is used to obtain a parent’s resources unused by child reserves
  - An "idle reserve" "accumulates" unused resources for accounting.
  - Arbitrary depths supported
- Hierarchical Temporal Isolation
  - Local Schedulability test in each level of hierarchy
- ‘Excess resource’ sharing control
- Timing guarantee with co-existing different scheduling policies.

Deferrable vs Sporadic Replenishment

- **Replenishment mechanism**
  - When do execution budgets get replenished?
- **Using the Deferrable Server (DS) approach, replenishment is fixed every $T_p$ parent period.**
  - This is how reservations in RK work normally
  - Much easier to implement
  - More complex to analyze
  - Schedulable utilization may be lower
- **Sporadic Server (SS), the resource is replenished $T_p$ from the time it is actually used.**
  - Harder to implement
  - Easier to analyze
  - Schedulable utilization is higher
Hierarchical Admission Control

- Two-step Schedulability test
  - The test of a parent reserve itself
    \[
    \omega^{n+1} = C_i + \sum_{j<i} \omega_j^n T_j \leq C_j
    \]
    \(J_i = 0\) for regular reserve or SS
    \(= T - C_i\) for Deferrable Server, \(\omega^0 = C_i\)
  - The test of its child reserves
    \[
    \omega_i^{n+1} = C_i + \sum_{j<i} \frac{\omega_j^n}{T_j} = C_j + \left[ \frac{\omega_i^n + HB}{T^P} \right] (T^P - C^P)
    \]
    \(\omega^0 = T^P - C^P + C_i\)

- HB is Hierarchical Blocking Term
  - \(HB = 0^P\) for Deferrable Server
  - \(HB = \min(\sum_{j<i} C_j, C^P)\) for Sporadic Server

Only local analysis within a reserve domain needs to be considered.

CPU Utilization of Siblings (regular vs. default)

These 2 reserves are 3 levels deep in the hierarchy.
Performance Overhead

- **Linear with the height of the hierarchy**
  - reservation replenishment and enforcement
- **Constant**
  - admission control (only local schedulability analysis)
  - scheduling (internal priority mapping and disabling/re-enabling of process eligibility to be scheduled.
- **Constraints**
  - reservation period must be greater than twice the parent’s reservation period
- **Hidden overhead**
  - higher degree of interrupts, because of replenishment, enforcement timers going off more frequently.

Degrees of Temporal Isolation

- Different degrees of temporal isolation in the presence of resource-sharing
  - **Strict Isolation**: the timing behavior of an application is not affected by the timing misbehavior of any other application
    - RK applications in the absence of logical sharing of resources
  - **Non-Strict Isolation**: traditional priority-driven systems
  - **Weak Isolation**: timing behavior is not affected by the timing misbehavior of applications with which no logical resources are shared.
Resource-Sharing Protocols in RK

- Analogues to Priority Inheritance and Priority Ceiling Protocols in Resource kernels
- Temporal isolation can only be weak
  - with resource-sharing

- Single-Reserve PCP
  - Assign one reservation to the logical resource execution
  - Very pessimistic allocation is required to maintain PCP semantics

- Multi-Reserve PCP
  - Has the same schedulability analysis as traditional PCP
  - Requires special support in RK
  - Can be applied to client-server models
    - Pass client’s reserve to server along with request

Cooperative Scheduling

Disk (Controlled Resource)
- DiskBW Resource Manager
- Disk driver and scheduler
- CPU (Controlling Resource)
- CSS
- QT video player

(C, T, D)
A Disk Workload w/o Disk Reservations

Carnegie Mellon University

All FS threads miss their deadlines

Disk Workload w/ Reservations

Carnegie Mellon University

FS1, FS3 and FS4 meet all their deadlines

Real-Time and Multimedia Systems Laboratory
Reserved FS1 w/ and w/o CPU-competitive load

Essentially, no effect on any fs-reserved thread

Network Processing Architecture

Resource set Problems
- Non-preemptive System calls
  - Priority Inversion
- Eager Receiver Processing
  - Interrupt-driven
  - Priority(capturing packet) > Priority(protocol proc.) > Priority (application processing)
- Lack of effective load shedding
- Lack of traffic separation
- Inappropriate resource accounting
Threaded Network Processing

- Application Threads
- Resource set
- User Kernel
- Reservation domain extends into the kernel, and its activity is controlled

Network and CPU Service Guarantees

- Reduction in non-preemptibility
- Control of receiver overload (*receive-livelock*)
- Prevention of scheduling disruption
- Separation of individual flows and proper resource accounting
- Packet scheduling for QoS guarantees
Real-Time Java Spec & RK Impact

• Thread Scheduling
  – Fixed Priority as the base scheduling policy
  – Allow different scheduling policies: Periodic, Dynamic Deadline, Priority
  – RK model for enforcement and resource consumption for CPU (optional)

• Synchronization
  – Bound Priority Inversions
  – Two Protocols: Priority Ceiling and Priority Inheritance

• Asynchronous Event Handling
  – Mechanism to implement event-driven application through event handlers.
  – Asynchronously Interrupted Exception is used to enable this new semantic in a
    particular method.

• Asynchronous Transfer of Control
  – New semantic for the interrupt() method, allowing to interrupt the execution of a
    method anywhere in the code.
  – Asynchronously interrupted Exception is used to enable this new semantic in a
    particular method.

• Memory Management
  – RT Garbage Collector
  – Garbage-Collection-Free Memory Regimes

• Physical Memory Access
  – Allows a Memory Area to be mapped to a specific physical address.

• Timers and clocks
  – Provide timely execution of threads and RK reserves
  – Control the priority preemption of the garbage collector
  – Control the priority inversion in thread synchronization
  – Provide the priority preemption of the garbage collector
  – Control the priority inversion in thread synchronization

Chocolate (Real-Time Java)

Starting point for the Real-Time Java Reference Implementation

From Timesys:

• Provide timely execution of threads and RK reserves
• Control the priority inversion in thread synchronization
• Provide the priority preemption of the garbage collector
• Control the priority inversion in thread synchronization

Asynchronous Event Handling (enforce on event arrival)

• Mechanism to implement event-driven application through event handlers.
• Asynchronously interrupted Exception is used to enable this new semantic in a
  particular method.

Asynchronous Transfer of Control

• New semantic for the interrupt() method, allowing to interrupt the execution of a
  method anywhere in the code.
• Asynchronously interrupted Exception is used to enable this new semantic in a
  particular method.

Memory Management

• RT Garbage Collector
• Garbage-Collection-Free Memory Regimes

Physical Memory Access

• Allows a Memory Area to be mapped to a specific physical address.

• Provide timely execution of threads and RK reserves
• Control the priority inversion in thread synchronization
• Provide the priority preemption of the garbage collector
• Control the priority inversion in thread synchronization

Asynchronous Event Handling (enforce on event arrival)

• Mechanism to implement event-driven application through event handlers.
• Asynchronously interrupted Exception is used to enable this new semantic in a
  particular method.

Asynchronous Transfer of Control

• New semantic for the interrupt() method, allowing to interrupt the execution of a
  method anywhere in the code.
• Asynchronously interrupted Exception is used to enable this new semantic in a
  particular method.

Memory Management

• RT Garbage Collector
• Garbage-Collection-Free Memory Regimes

Physical Memory Access

• Allows a Memory Area to be mapped to a specific physical address.
Technology Transition

Carnegie Mellon University

- Integrated products for delivery of end-to-end QoS
  - Flagship product based on major extensions to CMU Linux/RK
  - QoS run-time support in the form of TimeSys Linux/RT™ and JTime™
  - A comprehensive set of development tools
    - TimeWiz®: a high-level modeling, analysis and simulation tool
    - TimeTrace® and TimeCode™: Profiling and visualization tools
    - TimeStorm™: Integrated Development Environment
- Integration with QoS Models (Q-RAM)
- Users:
  - Telecommunications, servers, consumer devices, defense systems

Real-Time and Multimedia Systems Laboratory

TimeSys Linux Features

Carnegie Mellon University

- **Resource kernel and QoS Support**
  - guaranteed, **timely** and enforced access to CPU cycles and network bandwidth
- SMP support with QoS Reservations
- Fully preemptive kernel
- Fixed-priority scheduling (POSIX-compliant)
- High-resolution timer and clock support (microsecond resolution)
- Periodic processes
- Priority inheritance and priority ceiling protocol emulation support to avoid unbounded priority inheritance

Real-Time and Multimedia Systems Laboratory
The power consumption of CMOS circuits is given by:

\[ P \propto C_L \times V_{dd}^2 \times f \]

Running a CPU at lower speed consumes less energy.

Real-Time Tasks are required to complete by their deadlines.

Reducing the CPU speed to complete the task just before its deadline achieves “real-time” performance but saves energy.

The Problem

How to determine the CPU clock frequency (voltage) to satisfy the schedulability of real-time task sets and save energy?

- Static voltage scaling (SVS) algorithms
  - One clock frequency for the entire task set
  - One clock frequency for each task
- Dynamic voltage scaling (DVS) algorithm
  - Save energy when tasks run for less than their worst-case computation times
- Hardware limitations: non-ideal power-frequency characteristic, finite operating frequencies
Summary of Results

- **Sys-Clock**: optimal system clock frequency assignment with complexity $O(Mn^2)$
- **PM-Clock**: a task clock frequency assignment with the same complexity
  - The clock frequency assigned to a task is greater than or equal to that assigned to a lower priority task
- **Dynamic PM-Clock**: with addition $O(1)$ at every context switch
- **PM-Clock/Dynamic PM-Clock reduce energy by 71% at 50% utilization**
- **Hardware Inefficiencies**:
  - Determine Energy-Inefficient operating frequencies which should be excluded from voltage-scaling algorithm
  - Determine optimal clock frequency grid to minimize energy quantization error with limited # of operating frequencies

**Intel Pentium**
**Compaq iPaq hand-held**
**Intel XScale including MainStone II and Glencoe**
**Transmeta (University of Pittsburgh)**
**PowerPC (ISI and BAE)**

Downloadable from [http://www.cs.cmu.edu/~rtml](http://www.cs.cmu.edu/~rtml)

Wireless Sensor Networks:
The FireFly Project

Rahul Mangharam, Anthony Rowe
Anand Eswaran, Jun Li, Rakesh Reddy,
Sid Singh and Dhiraj Patel
FireFly: Secure Sensor Networks for Physical Infrastructures

- Develop a secure software platform for infrastructure sensor networks to be deployed in physical structures such as factories, buildings, homes, bridges, campuses, vehicles, ships and planes.
- Provide continual monitoring of operational health and safety,
- Report malfunctions (nearly) instantly, and
- Support mobile nodes and track people.

CMU FireFly Hardware Platform

Carnegie Mellon University

- Ultrasonic Positioning
- Time Synchronization
- FireFly Node
- eWatch

Smart Camera
CMU FireFly Sensor Node
Carnegie Mellon University

Hardware Time Sync

Atmega128L Processor
Temperature
Expansion Port
Acceleration

PIR Motion

CC2420 Radio
FLASH
Light
Audio

Real-Time and Multimedia Systems Laboratory

FireFly Synchronization Support
Carnegie Mellon University

Real-Time and Multimedia Systems Laboratory
RT-Link: A Time-Synchronized Link Protocol

Timeline with Scheduled Slot and Contention Slot regions of operation following a synchronization pulse.

Nano-RK Architecture

- Real-Time Sensor OS for FireFly nodes
- Multi-threaded
- Power-Aware
- CPU, Network and Sensor Reservations
- Multi-hop routing
- TDMA scheduling
Energy Reservations

- **{CPU, Network, Peripherals}**
  - Together comprise the total energy usage of the node
  - Reservations can be imposed on each of these resources
- **Static Offline Budget Enforcement**
  - It is possible to calculate a node lifetime given a certain energy budget

<table>
<thead>
<tr>
<th>Node</th>
<th>Reserve [TX, RX]</th>
<th>TX Rate</th>
<th>Lifetime w/ out Reserve</th>
<th>Lifetime w/ Reserve</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>[1,2]</td>
<td>1</td>
<td>8 years</td>
<td>8 years</td>
</tr>
<tr>
<td>b</td>
<td>X</td>
<td>300</td>
<td>3.5 days</td>
<td>3.5 days</td>
</tr>
<tr>
<td>c</td>
<td>[1,2]</td>
<td>1</td>
<td>5 years</td>
<td>5 years</td>
</tr>
<tr>
<td>d</td>
<td>[1,2]</td>
<td>1</td>
<td>3.9 days</td>
<td>5 years</td>
</tr>
<tr>
<td>e</td>
<td>[1,2]</td>
<td>1</td>
<td>4 days</td>
<td>2.9 years</td>
</tr>
</tbody>
</table>

Miner Tracking in Mines

- **Coal Mine in South Hills Test Deployment**
Guard Tracking In Prisons

Carnegie Mellon University

Our FireFly sensor network ‘participated’ in about 10 mock riot exercises by law enforcement officers from around the country.

Real-Time and Multimedia Systems Laboratory

CMU CyLab Deployment

Carnegie Mellon University
Summary

The Resource Reservation paradigm works well under a wide variety of conditions:
- Throughput and timing guarantees can be provided and enforced.
- Works with many resource types including CPU, network, disk, and physical memory.
- Decouples application-level requirements from OS-level mechanisms (of priority, scheduling policy, etc.).
- Flat and hierarchical reservations.
- Minimizes unnecessary testing of combinations.
- Extends very naturally to energy-aware environments.
- Maps to wireless sensor networks (very resource-constrained environment).

What Next?

Extensions for “Statistical Resource Reservations” for highly stochastic workloads.
- Support for large-scale distributed systems.
  - End-to-end resource reservations can be configured to have throughput, timing, security, and fault-tolerance attributes.
- Study applicability to highly mobile dynamic ad-hoc networking:
  - How does the resource reservation paradigm operate?
  - Is it even the right paradigm? Is it where the core problems lie?