SF3P: A Framework to Explore and Prototype Hierarchical Compositions of Real-Time Schedulers

Andres Gomez, Lars Schor, Pratyush Kumar, Lothar Thiele

Intl. Symp. on Rapid System Prototyping
New Delhi, October 16-17, 2014
Motivation

- Highly integrated real-time systems are showing:
  - Increasingly complex functionality
  - Need for sophisticated scheduling techniques (mixed-criticality)

- Scheduler designers need to validate at early design stages
  - Prototype schedulers on different HW platforms

- Prototyping platforms should:
  - Offer a high level of abstraction (extendable)
  - Have minimal system requirements
  - Inexpensive to execute (low overhead)
Software Options in Real-Time Systems

- Unix-like OS
  - High HW/SW compatibility
  - Limited scheduling options

- Modified Kernel Space
  - High HW compatibility
  - Customizable scheduling options
  - Limits SW compatibility/portability

- Custom RTOS
  - Finely tuned scheduler
  - Limited HW/SW compatibility

Our proposal:
- Add **flexible** scheduling layer on top of a **standard** kernel
Our Scheduling Model

Task

insert

select

Task

Executes

remove

Task

Finishes

preempt
Scheduling in Unix-like Operating Systems

Unix-like

Kernel Space

User Space

Task 1 Wrapper

Task N Wrapper

Task N Wrapper

Concurrency Mngr

Scheduler

Hw-dep. Sw

Hardware

MyMngr

MySched

AQuoSA
Scheduling Framework for Fast Prototyping (SF3P)

Unix-like

Our Solution

User Space

Task 1 Wrapper
Task N Wrapper

Concurrency Mngr
Scheduler
Hw-dep. Sw
Hardware

Kernel Space

Task 1
... Task N

Sched1
... SchedN

SF3P

Concurrency Mngr
Scheduler
Hw-dep. Sw
Hardware

Config
Our Goals

- We can add a scheduling layer in the User Space

  1. Portable to different platforms with no cost

  2. Extendable to new schedulers with low cost

  3. Low Overhead
Our Goals

- We can add a scheduling layer in the User Space

1. Portable to different platforms with no cost

2. Extendable to new schedulers with low cost

3. Low Overhead
SF3P – Concurrency Manager Interaction
Basic Concept – How does SF3P Schedule?

Priorities

- high
  - sched
  - active
  - inactive

Task

SF3P

Concurrency Mngr

Priority Setter

Timers

Insert

inactive

select

Remove

inactive

Executes

Preempt
Time Triggered Scheduling

- Time Division Multiple Access
Our Goals

- We can add a scheduling layer in the User Space
  
  1. Portable to different platforms with no cost
  
  2. Extendable to new schedulers with low cost
  
  3. Low Overhead
Our Goals

- We can add a scheduling layer in the User Space
  
  1. Portable to different platforms with no cost ✓
  
  2. Extendable to new schedulers with low cost
  
  3. Low Overhead
Adding a New Scheduler

- **Generic Scheduler**
  - insert
  - select
  - remove
  - preempt

- **Decoupled Insertion**
  - insert
  - select
  - remove
  - preempt

- Implemented: FIFO, FP, EDF, RM, TDMA
More Hierarchical Scheduling
Criteria Inheritance

TDMA

Slot 1  Slot 2

FIFO

TSet1

T1  Ddln  Pr

T1  Ddln  Pr

T2  Ddln  Pr

TSet3

Task 1  ...  Task N

EDF  RM  ...  TDMA

SF3P

Concurrency Mng

Scheduler

Hw-dep. Sw

Hardware

Config
Our Goals

- We can add a scheduling layer in the User Space
  1. Portable to different platforms with no cost
  2. Extendable to new schedulers with low cost
  3. Low Overhead
Our Goals

- We can add a scheduling layer in the User Space
  1. Portable to different platforms with no cost
  2. Extendable to new schedulers with low cost
  3. Low Overhead
Evaluation Mechanism

- **Configuration File**
  - Specify schedulers, tasks, criteria

- **Dispatcher Library**
  - Simulate task arrivals

- **Analysis Tools**
  - Calculate metrics
Experimental Evaluation

- Desktop Testing Environment

  - Linux Kernel: 3.2
  - Processor: Intel i7 @ 3.4GHz
  - Memory: 16 GB RAM
  - Linux Runlevel: 1

Diagram:

- Task 1 → ... → Task N
- Sched1 → ... → SchedN
- SF3P
- Concurrency Mngr
- Scheduler
- Hw-dep. Sw
- Hardware
- Config
Experimental Evaluation (II)

- Embedded Testing Environment (Raspberry Pi)
  - Linux Kernel: 2.6
  - Processor: ARM V6 @ 700MHz
  - Memory: 512 MB RAM
  - Linux Runlevel: 1
Schedulability Analysis

- A schedule is feasible if tasks meet all of their deadlines

- In classical algorithms:
  - Utilization test
    \[ U = \sum_{i} \frac{C_i}{T_i} \]
  - If \( U < U_{\text{LUB}} \) then the schedule is feasible

- Generate (random) schedules and verify feasibility
  - \( N_{\text{tasks}} \in [5,50] \quad U \in [20,100]\% \)
  - \( C_{\text{long}} \in [40,50]\text{ms} \quad C_{\text{short}} \in [5,10]\text{ms} \)
Rate Monotonic Schedulability (Desktop)

![Diagram showing the schedulability region with Guaranteed area and No Guarantee area.](image-url)
EDF Schedulability (Desktop)

$U_{LUB} = 99.9\%$
EDF Schedulability (RPI)

\[ U_{\text{LUB}} = \sim 60\% \]
**SF3P Overhead**

**Overhead:** time spent executing *anything* other than tasks
SF3P Overhead

Scheduler Overhead
- Algorithm-dependent

Non-Scheduler Overhead
- Platform-dependent
Increasing the Levels of Hierarchy (L)

$L = \emptyset$

$N = 32$

$L \in [1,5]$

$U \in [50,90] \%$

$C \in [10,40] \text{ ms}$
Overhead vs Levels of Hierarchy

- Scheduler overhead increases linearly.
- Non Scheduler overhead remains constant.

Desktop

RPI

Fixed cost is dominant!
Re-running EDF with long (10x) Tasks on RPI

$U_{\text{LUB}} = 99.1\%$
Our Goals

- We can add a scheduling layer in the User Space
  1. Portable to different platforms with no cost
  2. Extendable to new schedulers with low cost
  3. Low Overhead
Our Goals

- We can add a scheduling layer in the User Space

  1. Portable to different platforms with no cost
  2. Extendable to new schedulers with low cost
  3. Low Overhead
SF3P Summary

- Framework for fast prototyping of real-time schedulers
  - Modular, extendable, composable

- New hierarchical schedulers
  - Suitable for complex scheduling needs

- Low overhead

Available at: http://www.tik.ee.ethz.ch/~euretile/scheduling