An Essential Relationship between Real-time and Resource Partitioning

Yoshitake Kobayashi
Advanced Software Technology Group
Corporate Software Engineering Center
TOSHIBA CORPORATION

2013/10/25
Overview

- Background
- Requirement
- Hardware resource partitioning
- Summary
Background

- **Hardware**
  - Multi-core CPU
  - Larger memory
  - Larger storage space
  - Hardware assisted virtualization

- **Software**
  - Operating system
    - Linux
  - Virtual Machine Monitor
Background

**Hardware**
- Multi-core CPU
- Larger memory
- Larger storage space
- Hardware assisted virtualization

**Software**
- Operating system
  - Linux
- Virtual Machine Monitor

**Issues on real-time systems**
- Meet its required deadline
  - ex. Control systems
- Performance requirement
  - Interrupt latency
  - Response time
Requirement (1)

- All Real-time application should meet its real-time constrain
  - Response time (Deadline): 100μs – 100ms
  - Event response time (Interrupt latency): 10μs – 100μs
Requirement (2)

- A system needs to be able to run both real-time (RT) application and general purpose (GP) application at same time
A sample of current implementation

- Prepare two hardware
- Implement RT application on a specific one
- Implement GP application on other one
- Connect each other by a bus or share memory

Real time process

General purpose processes

RTOS

Linux kernel

Devices

CPU

BUS
Requirement (3): Make a simple world

- A system software able to control RT and GP
  - System software: OS, VMM
Hybrid OS vs. Single OS approach
Hybrid OS vs. Single OS approach

**Hybrid OS**
- Two or more operating systems runs on same hardware
  - RT specific tasks run on RTOS (Real-Time Operating System) and the other tasks runs on GPOS (General Purpose Operating System)
  - ex. uITRON for RTOS and Linux for GPOS
- Possible implementations
  - By VMM
  - Run GPOS as a task on RTOS
- RTOS and GPOS have different APIs
  - Xenomai

**Single OS**
- Just use one OS to run both RT and GP applications
- Same API can be used for all applications
- Possible implementations
  - Kernel level RT process
  - RT-Preempt patch
Hybrid architecture (Xenomai)

- **Xenomai**
  - Reference: [http://www.xenomai.org](http://www.xenomai.org)
  - Dual kernel approach based on Adeos/I-Pipe
  - I-Pipe works to dispatch events (ex. Interrupts)
  - Xenomai skins build on top of the Xenomai nucleus to provide RTOS APIs such as VxWoks, uITRON
Hybrid architecture (TOPPERS SafeG)

- **SafeG (Safety Gate)**
  - Dual-OS monitor
  - Execute an RTOS (Real-Time Operating System) and a GPOS (General-Purpose Operating System) on the same hardware platform
  - ARM TrustZone security extensions uses to introduce the concept of Trust and Non-Trust states
  - On the other hand, code running under Non-Trust state, even in privileged mode, cannot access memory space (devices included) that was allocated for Trust state usage, nor can it execute certain instructions that are considered critical.

![Diagram of RTOS (uITRON) and Linux with safe G gate and IRQ/FIQ devices](image)
Hybrid OS vs. Single OS approach

**Hybrid OS**
- More than one OS runs on same hardware
  - RT specific tasks run on RTOS (Real-Time Operating System) and the other tasks runs on GPOS (General Purpose Operating System)
    - ex. Linux and uITRON
- Possible implementations
  - By VMM
  - Run GPOS as a task on RTOS
  - RTOS and GPOS have different APIs
    - Xenomai

**Single OS**
- Just use one OS to run both RT and GP applications
- Same API can be used for all applications
- Possible implementations
  - Kernel level RT process
  - RT-Preempt patch
Actual requirement (3): Linux

- Linux runs both RT and GP applications

Real time process

General purpose processes

Linux

Devices

CPU
Summary of requirements

1. Run RT processes and GP processes on a hardware platform
2. Need to meet required deadlines
   - One of the most important perspective for embedded systems
3. Use single OS approach
   - Linux
Actual requirement (3): Linux

- Linux runs both RT and GP applications

- This is not a good idea if you don’t care anything
Issues to run RT process and GP process

- **Determinism**
  - RT process should have a deterministic behavior
  - GP process doesn’t assume deterministic behaviour
How to improve real time performance?

- **Real-time Preemption Patch**
  - Fully preemptive kernel
  - Improvement for latency

- **CPU affinity**
  - Prohibit process migration from one core to another
  - Protect from GP process behaviour
  - Maybe good for determinism

Effects by workload

```
RT thread
RT thread
RT thread
Scheduler
RT CPU core

GP thread
GP thread
Scheduler
GP CPU core
```
Definition of hardware resource partitioning

- Partition is a set of hardware resource
  - CPU cores, Memory, Devices, ..
- Each partition must be isolated from the others
  - No device sharing
A use case of CPU affinity for RT process

- Run a set of process and thread on specific CPU core

Advantage

- No process migration
  - Process migration is not friendly with real time behaviour
  - Migration timing cannot be expected
- Just RT process runs on specific cores
  - Isolate all GP process into the other cores
Evaluation of interrupt latency with CPU affinity

- **Evaluation environment**
  - Hardware: Pandaboard
  - Period: 300μs
Evaluation of interrupt latency with CPU affinity

W/O CPU Affinity

CPU Affinity

CPU Affinity (in some case)
What’s occurred?

Period

Timer cascade occurs here
Limitation of CPU affinity

- Example for CPU core specific kernel thread
  - **Timer**, High resolution timer
  - Process migration
  - Etc..

CPU core 0

CPU core 1

Kernel thread

CPU core specific kernel thread

Process
Cascade timer list

- Cascade timer
  - Register the next timer list to the end of current one

- Impact of cascade timer to interrupt latency
  - Runs with interrupt disabled context
  - No limits for the number of timers
  - Timer process cost becomes higher when tickless kernel used
Control cascade timers on RT CPU core

Solution

- Keep the timer list empty on RT core to protect from cascade timers

![Diagram of timer control](image)

- RT CPU core
  - Kernel thread
  - Timer list (Empty)
  - Timer

- GP CPU core
  - Timer list
  - Timer
Three issues which cause cascade timers

(I) Registered by GP process before migration
(II) Registered by RT Core specific kernel thread
(III) Registered by RT Core specific kernel thread before RT task runs

Expired timers causes cascade
Solution for the issue (I)

- **Preparation**
  - Log all timer registration by kernel thread

- **Solution**
  (A) Migrate kernel threads or a GP processes to GP core
  (B) Migrate registered times to GP core refer the log
  (Timer migration)
Solution for the issues (Ⅱ) and (Ⅲ)

(Ⅱ) Registered by RT Core specific kernel thread
(Ⅲ) Registered by RT Core specific kernel thread before RT task runs

Case 1: Not enough time to migrate
Wait for expiration

Case 2: Enough time to migrate
Migrate the timer to GP core

Restrict to register new timers to the GP core only
Evaluation

Before

CPU 0

CPU 1

After

CPU 0

CPU 1

Cascade occurs here
Summary

- **Requirement**
  - RT processes and GP processes on a same hardware platform
  - Just use Linux for both processes
  - Meet its required deadline for RT process

- **Hardware resource partitioning**
  - Set of hardware resources which is isolated from the others
  - Define CPU cores as RT core and GP core

- **Issues to implement the resource partitioning**
  - Some kernel thread cannot be migrated
    - Core specific kernel thread
  - Need to care with CPU affinity feature
    - Focused on cascade in timer.c
    - Protect from cascade function on RT core
      - Keep timer list empty

- **Future plan**
  - Fixing issues
  - SCHED_DEADLINE on RT core with fine granularity support
Questions?

The latest slide is available at the following URL: http://elinux.org/ELC_Europe_2013_Presentations