Real-Time Operating Systems for MPSoCs

Hiroyuki Tomiyama
Graduate School of Information Science
Nagoya University
http://member.acm.org/~hiroyuki

Contributors

- Hiroaki Takada
  - Director and Professor
  - Center for Embedded Computing Systems
    Graduate School of Information Science
    Nagoya University

- Shinya Honda
  - Assistant Professor
  - Center for Embedded Computing Systems
    Graduate School of Information Science
    Nagoya University

Outline

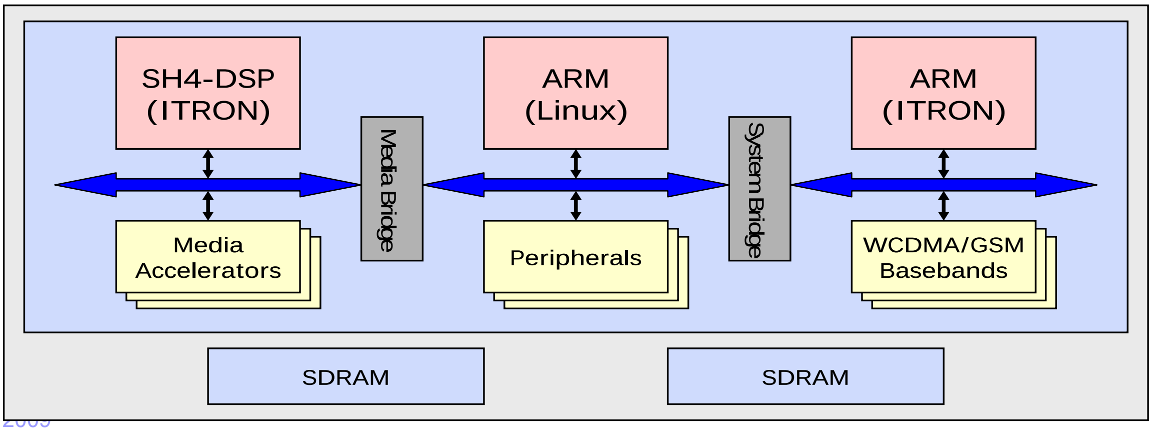
- Why Multicore Embedded Systems?
- Types of RTOS and Real-Time Issues
- RTOSes from TOPPERS Project
- Concluding Remarks

Why Need Multiprocessors?

- Mainly two reasons; one negative, one positive
- To achieve both high performance and low power simultaneously
  - To be honest, software programmers DO NOT want to use multiprocessors
    - They want a single processor with high-performance and low-power
  - But, high-performance processors are inevitably less power-efficient (e.g., lower MIPS/Watt) than low-performance ones
- To achieve different requirements (functionality, performance, real-time response, reliability, etc.) simultaneously
  - Contemporary embedded systems are complex; consisting of a set of sub-systems with their own requirements
    - Ex: Cell phones, automotive electronic systems, NC machines

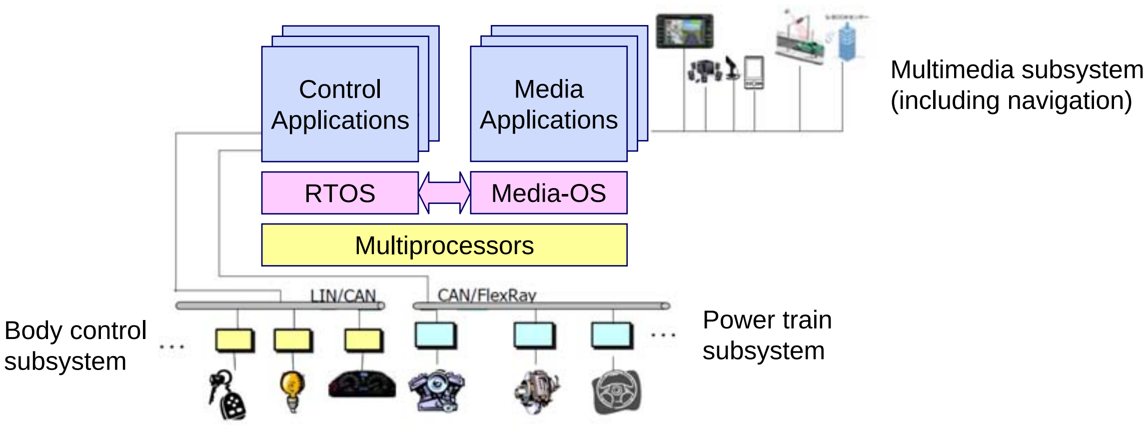
Different Requirements for Cell Phone SoCs

- Applications in cellular phones
  - Telephone, video phone, e-mail, web browser, digital still camera, video camera, TV, music player, game machine, e-money, etc.
- Each application has its own requirements such as throughput, real-time responsiveness, reliability, etc.
- Renesas SH-Mobile G1
  - Main control, user interface, java, etc: rich functionality, loose real-time response
  - Baseband communication: hard real-time response
  - Media processing: high computational power, soft real-time response



Car Navigation System

- Multicore ECUs (electronic control units)
- Two different OSes in cooperation with each other
  - RTOS for body control and power-train subsystems
  - Rich OS for multimedia subsystem



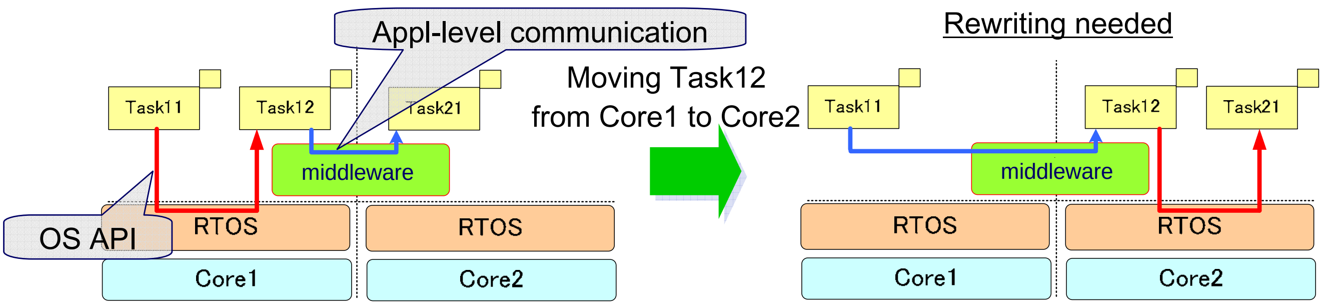
Types of RTOS and Real-Time Issues

Broad Classification

- UP-RTOS (Uni-Processor RTOS)
  - Designed for single-processor real-time systems, still can be used for multiprocessor systems
- MP-RTOS (Multi-Processor RTOS)
  - Designed for multiprocessor real-time systems
  - SMP-RTOS (Symmetric Multi-Processor RTOS)
  - AMP-RTOS (Asymmetric Multi-Processor RTOS)

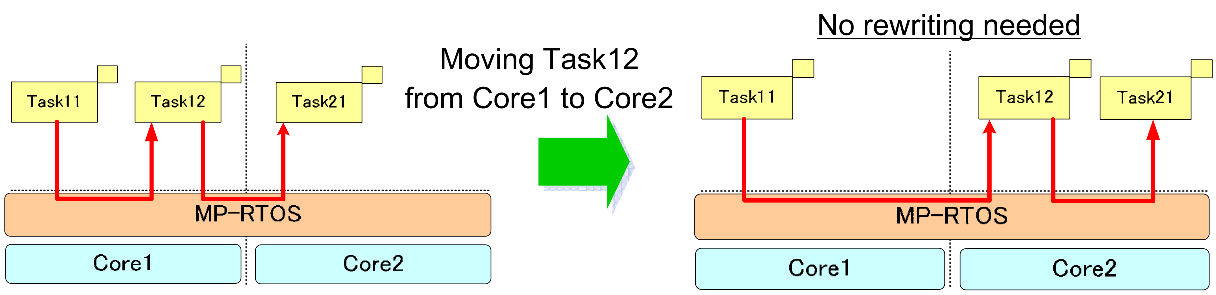
Use of UP-RTOS in Multiprocessors

- UP-RTOS runs on each processor
- Intra-processor communication is realized by RTOS API (e.g., mail boxes, data queues, etc.) while inter-processor communication is realized at an application-software level (using middleware).
- Problems
  - Dependency between task development and task allocation
  - Run-time task migration is difficult (should be realized at application level)



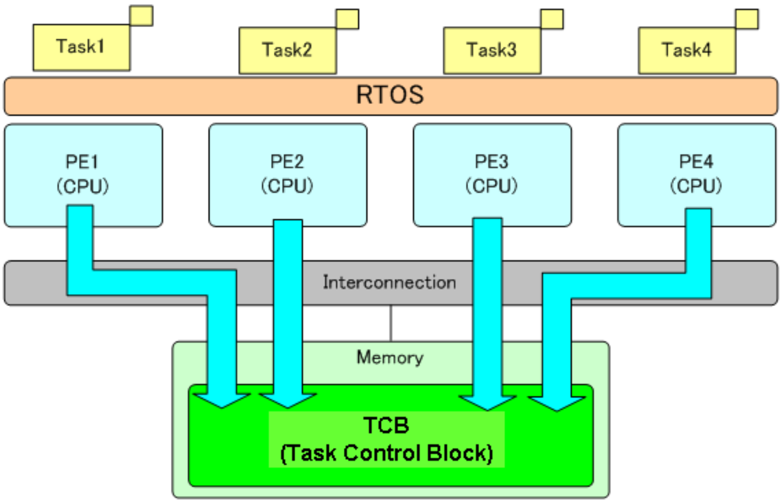
Merits of MP-RTOS

- Provide same API for both inter-processor and intra-processor communication
  - Application programmers do not have to be aware of task allocation
  - Exploration of task allocation is easy



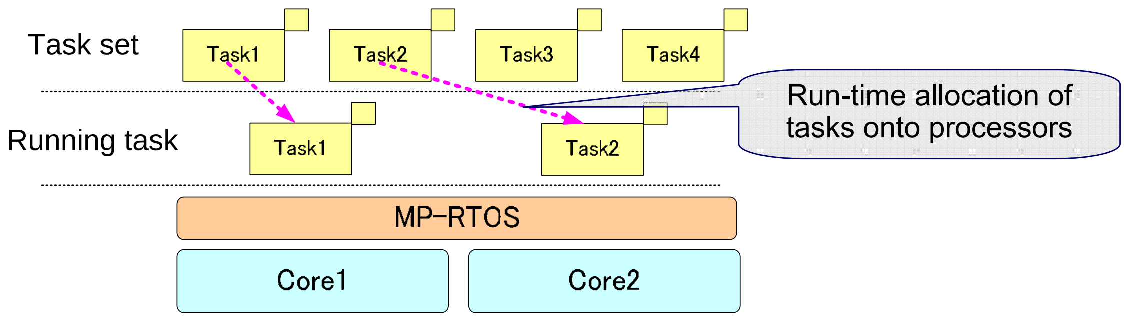
Inter-Processor System Calls

- Two implementation approaches
  - Direct manipulation of TCBs
    - Employed by all SMP-RTOSes and many AMP-RTOSes
  - Remote calls



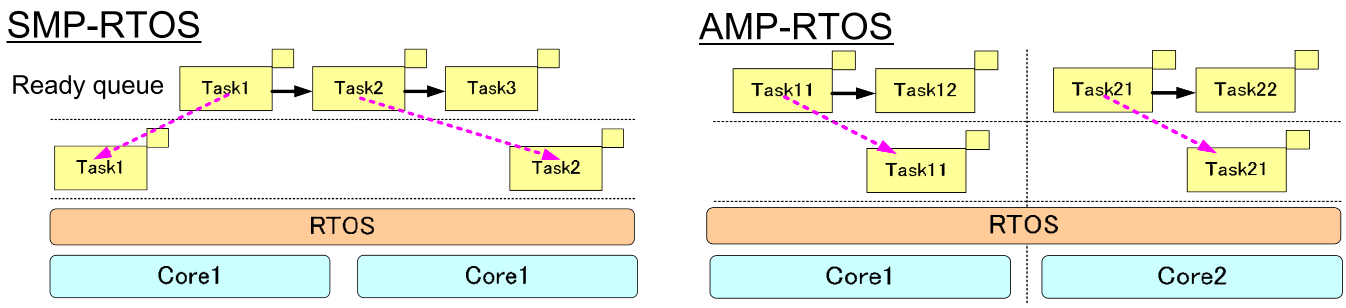
Merits of MP-RTOS

- Run-time task migration (SMP-RTOS only)
  - Allocate tasks onto processors at run-time in order to maximize the throughput depending on load variation.
  - Running tasks can be migrated
- Important: Run-time task migration often degrades real-time responsiveness (worst-case response time of tasks and interrupts).
  - We need to understand how MP-RTOS behaves.



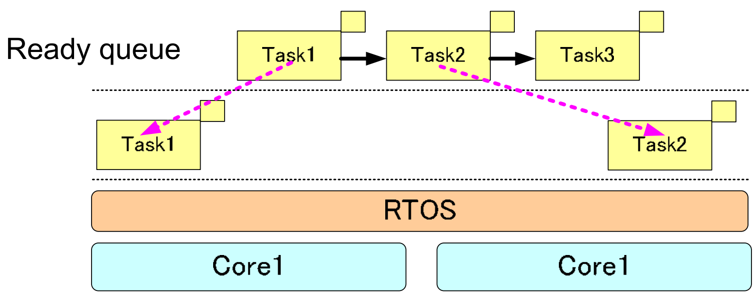
SMP-RTOS and AMP-RTOS

- SMP-RTOS: Symmetric Multi-Processor RTOS
  - Dynamic task allocation
  - Well suited to SMP architectures with homogeneous processors and coherent cache
  - Most commercial MP-RTOSes
- AMP-RTOS: Asymmetric Multi-Processor RTOS
  - Static task allocation
  - Suited to both SMP and AMP architectures



SMP-RTOS: Task Scheduling

- Typically, a single ready queue shared by all processors
- The first *N* tasks in the ready queue are dispatched to *N* processors
- Many SMP-RTOSes provide ability to statically allocate tasks to specific processors
  - Task scheduling becomes a little complicated

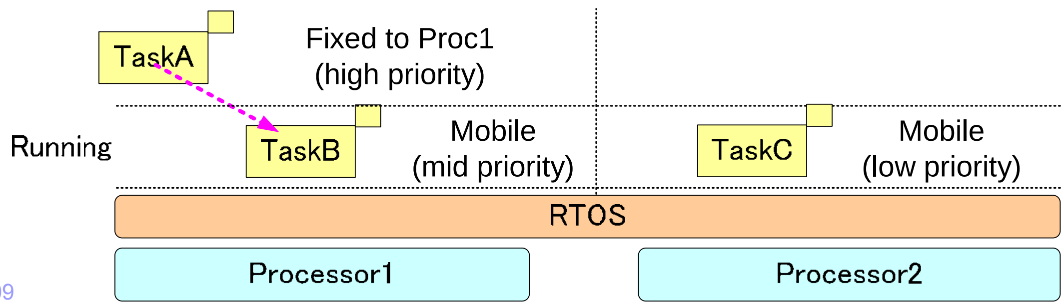


Some Important Issues in SMT-RTOS

- Dynamic task allocation / migration
  - Needs performance overhead
  - Moreover, often degrades the cache performance
    - especially, in case currently-running tasks are migrated
  - Tasks should remain on the same processors as long as possible
  - Tradeoff between average- and worst-case performance
- Priority inversion
  - Complicates analysis and guarantee of schedulability
  - Occurs very easily in SMP systems
  - Tradeoff with task migration
- Resource conflicts inside SMT-RTOS
  - A number of data structures inside SMT-RTOS are shared by tasks
  - Accesses to the shared data structures may cause conflicts

Priority Inversion with Static Tasks

- Many SMP-RTOSes have ability to fixedly allocate tasks to specific processors.
- What should RTOS do in the following scenario?
  - Assume Tasks B and C are running
    - Both tasks are mobile over processors
  - Now, Task A which is fixed to Processor 1 arrives
- Large overhead necessary if we strictly follow the task priorities
  - Save the context of Task C in memory, and put the task back in ready queue
  - Migrate Task B from Processor 1 to Processor 2
  - Dispatch Task A to Processor 1



Some Important Issues in SMT-RTOS

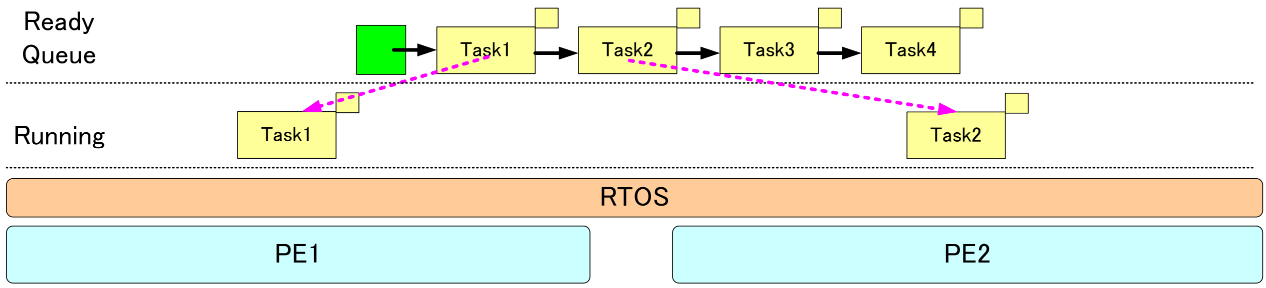
- Dynamic task allocation / migration
  - Needs performance overhead
  - Moreover, often degrades the cache performance
    - especially, in case currently-running tasks are migrated
  - Tasks should remain on the same processors as long as possible
- Priority inversion
  - Complicates analysis and guarantee of schedulability
  - Occurs very easily in SMT systems
  - Tradeoff with task migration
- Resource conflicts inside SMT-RTOS
  - A number of data structures inside SMT-RTOS are shared by tasks
  - Accesses to the shared data structures may cause conflicts

Resource Conflicts inside SMP-RTOS

- Minimization of interferences among tasks is important in order to guarantee real-time responsiveness of the tasks
- Inter-task interferences include
  - communications/synchronizations
  - preemptions
  - resource conflicts
- There exist a number of shared resources not only in hardware (such as memories and buses) but also inside RTOS.
- Example: Ready queue
  - one of the most important, frequently-accessed resources in RTOS

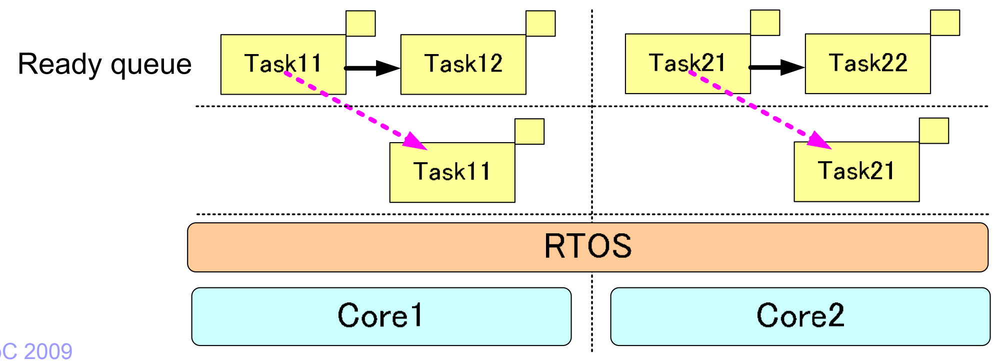
Ready Queue in SMP-RTOS

- Ready queue
  - Manipulated by many system calls
  - Accesses to ready queue must be mutually excluded
- Typical SMP-RTOSes have a single ready queue
- Scenario
  - Task 1 completes its execution
  - At the same time, Task 2 goes into waiting state for synchronization with an external device
  - Both Tasks 1 and 2 need to be removed from the ready queue. Conflict!
- Note that this conflict happens even if Tasks 1 and 2 are independent of each other
  - This does not happen in AMP-RTOS with a ready queue for each processor



AMP-RTOS: Task Scheduling

- Multiple ready queues; one for each processor
- On each processor, the highest-priority task in ready queue is executed
  - In the same way as UP-RTOS. Independent among processors
- Uniprocessor-based schedulability analysis can be applied if no inter-processor communication exists
  - This is not the case for SMP-RTOS even if all tasks are statically allocated to specific processors



SMP-RTOS: Pros and Cons

- High throughput (average performance) via load balancing
- Easy software development, high reusability of software
- Expensive hardware required
  - Coherent cache, fast interconnection network, etc.
- Difficult schedulability analysis
- Degraded worst-case responsiveness
  - More shared resources, more possibilities of resource conflicts

AMP-RTOS: Pros and Cons

- Lower throughput (average performance)
  - Dynamic task allocation needs to be implemented at an application level
- More work at design time
  - Task allocation
- Higher performance expected via application-specific customization
- Less expensive hardware
- Easier schedulability analysis
- Better worst-case responsiveness

➤ AMP-RTOS is a better choice than SMP-RTOS for hard real-time systems

RTOSes from TOPPERS Project

RTOSes from TOPPERS Project

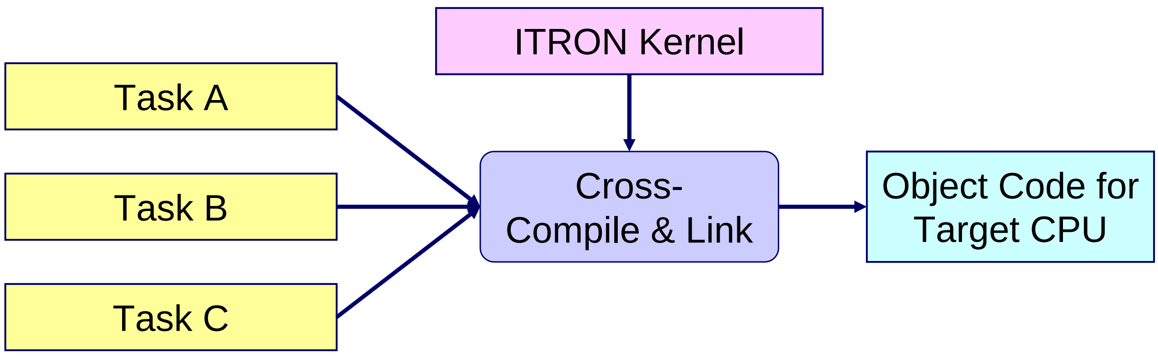
- TOPPERS/JSP Kernel
  - Designed for uniprocessor systems
- TOPPERS/SMP Kernel
  - Designed for symmetric multiprocessor systems
  - Run-time task allocation
- TOPPERS/FDMP Kernel
  - Designed for asymmetric multiprocessors
  - Static task allocation
- TOPPERS/FMP Kernel
  - Designed for asymmetric multiprocessors
  - Static task allocation with limited task migration
- All of them
  - conform to ITRON 4.0 Standard Profile
  - are (or will be) released as open-source software from TOPPERS Project

What's ITRON?

- A standardized specification of RTOS kernel for small- to mid-scale embedded systems.
- Developed and standardized in Japan for >20 years
  - Prof. Takada has been playing the central role
- ITRON is not a software product but a specification.
  - Defines a set of API functions (service calls)
  - There exist a number of ITRON implementations in market
- Most popular RTOS specification in Japan
  - Approximately 50% of embedded systems
  - Especially in consumer electronics.
- Several profiles to cover different application domains
  - Standard Profile, Automotive Profile, etc.

Task and Memory Management in ITRON Standard Profile

- Tasks (and other kernel objects such as semaphores and mail boxes) are statically defined and instantiated at design time
  - No dynamic loading at run time
- Single memory space shared by all of application tasks and RTOS
  - No virtual memory
- Priority-based preemptive scheduling



TOPPERS Project

TOPPERS
Toyohashi OPen Platform for Embedded Real-time Systems

- Not-for-Profit Organization
  - Founded by Professor Takada in September 2003
- http://www.toppers.jp/
- >200 members (universities, companies, and individual volunteers)
- Develop and release open-source software for embedded systems
  - RTOSes, middleware (e.g., FlaxRay/CAN communication middleware), and education materials
- TOPPERS License
  - Can be used for research, education and commercial purposes for free
  - Let us know when used in commercial products: “*Reportware*”
  - You may choose BSD License to be able to link TOPPERS software with GNU software.

TOPPERS Inside

KR-107 (Roland)

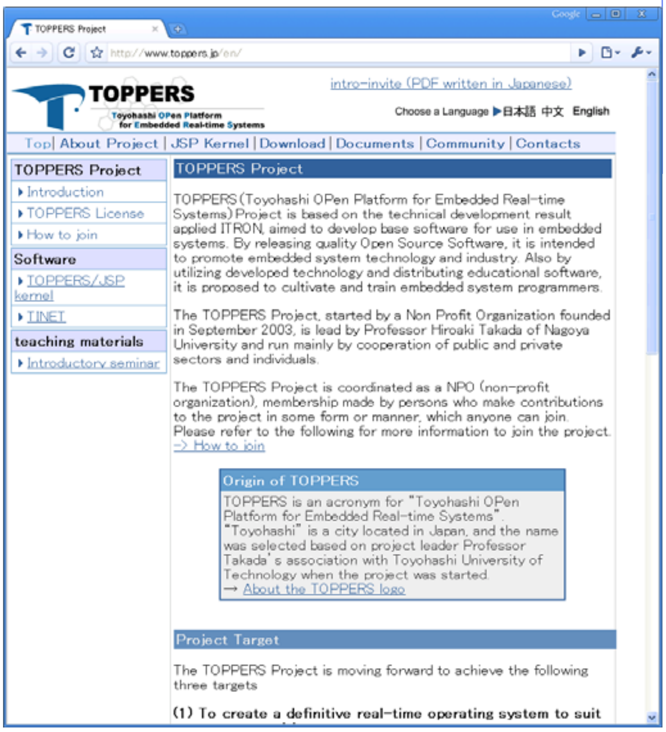
PM-A970 (Epson)

Do! KARAOKE (Panasonic)

GT-541 (Brother)

UA-101 (Roland)

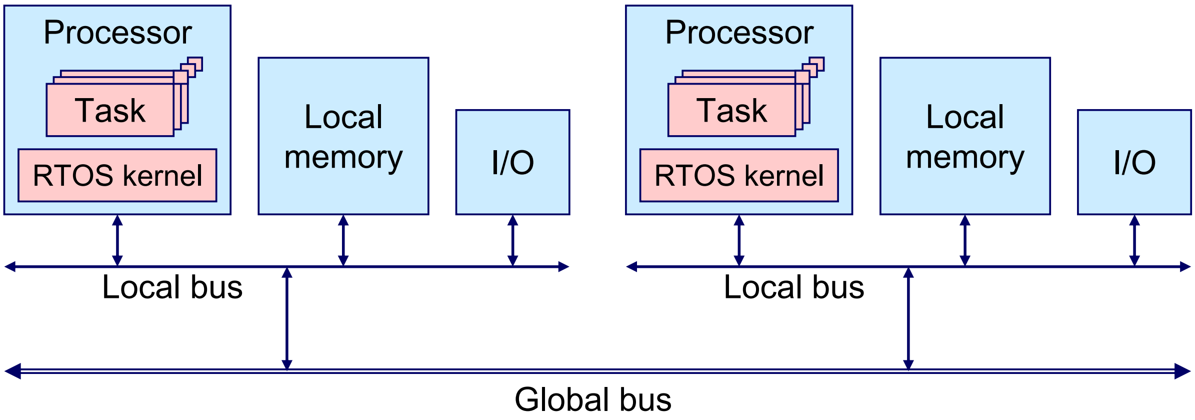
Sorry!
English page has less information



RTOSes from TOPPERS Project

- TOPPERS/JSP Kernel
  - Designed for uniprocessor systems
  - Freely available from the TOPPERS website as an open-source software
- TOPPERS/SMP Kernel
  - Designed for symmetric multiprocessor systems
  - Run-time task allocation
  - Pre-released to limited TOPPERS members
- TOPPERS/FDMP Kernel
  - Designed for asymmetric multiprocessors
  - Static task allocation
  - Freely available from the TOPPERS website as an open-source software
- TOPPERS/FMP Kernel
  - Designed for asymmetric multiprocessors
  - Static task allocation with limited task migration
  - ~~Pre-released to TOPPERS members (will be available to public next year)~~
  - Freely available from the TOPPERS website as an open-source software

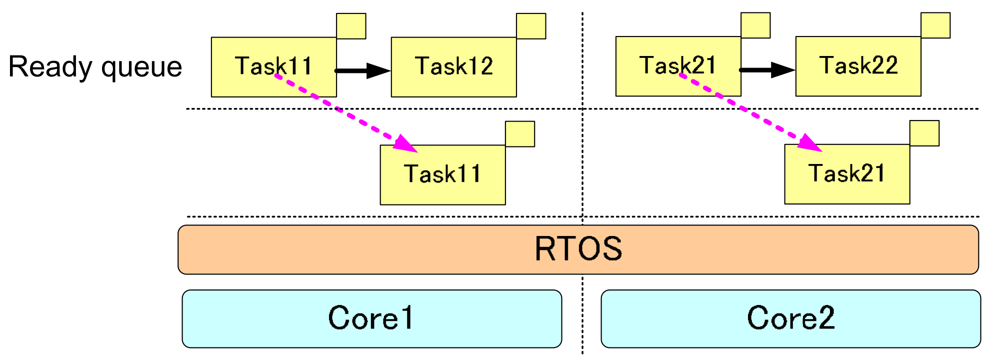
Target Arch. of FDMP/FMP Kernels



- Processors may be homogeneous or heterogeneous
- Local memory can be accessed by other processors
  - Remote memory access latency may be longer
- Tasks are statically allocated to processors
  - FMP Kernel partially allows dynamic task migration

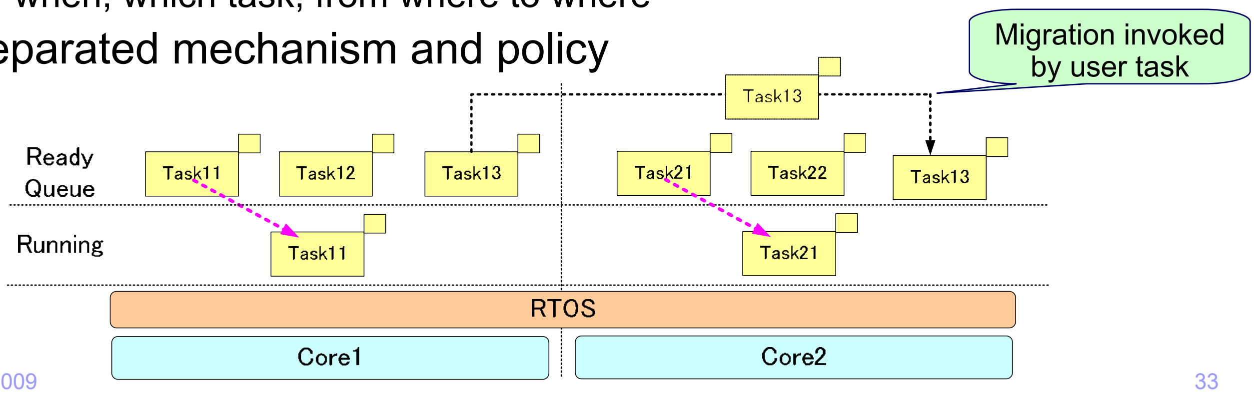
Task Scheduling in FMP/FDMP Kernels

- Priority-based preemptive scheduling on each processor
  - On each processor, the highest-priority task in ready queue is executed
  - Independent among processors
- Uniprocessor-based schedulability analysis can be applied if no inter-processor communication exists
  - This is not the case for SMP-RTOS even if all tasks are fixedly allocated to specific processors



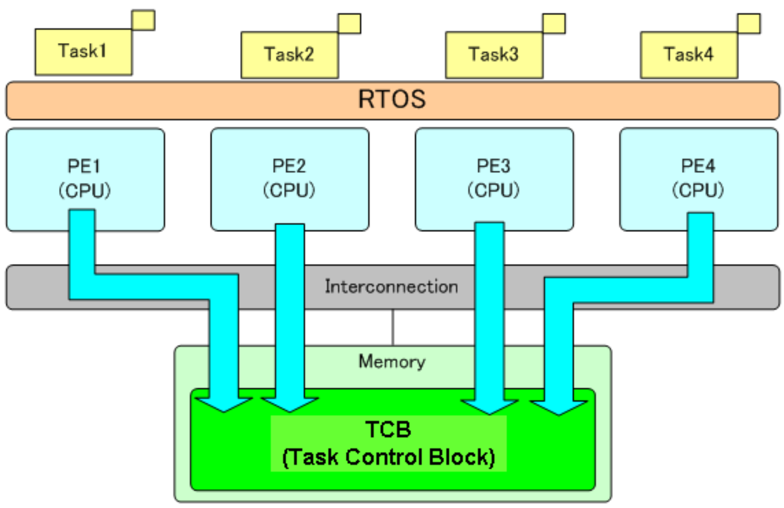
Task Migration in FMP Kernel

- Aim to achieve high throughput with guaranteed real-time responsiveness
- Automatic task migration loses predictability of system behavior
- FMP Kernel only provides API for run-time task migration
  - does not migrate tasks automatically
- It is programmer's responsibility to decide the task migration policy
  - when, which task, from where to where
- Separated mechanism and policy



Inter-Processor System Calls in FDMP/FMP Kernels

- FDMP/FMP Kernels provide the same APIs for intra- and inter-processor system calls
  - This separates task development and task allocation
- The system calls need to manipulate TCBs (task control blocks) of other tasks
- FDMP/FMP Kernels manipulate TCBs of the tasks which are running on different processors



Concluding Remarks

Concluding Remarks

- Bounding and minimizing worst-case response time of tasks and interrupts are critically important, but very difficult in multiprocessor systems.
  - Need to know how RTOS behaves
- TOPPERS/FDMP Kernel and FMP Kernel
  - implement a number of techniques to improve the real-time responsiveness
  - are open-source software available from TOPPERS Project at http://www.toppers.jp/
  - have production-level quality
- Your trial use and feedbacks are highly appreciated