Recent Results the ITRON Subproject

Hiroaki Takada

Dept. of Information and Computer Sciences
Toyohashi Univ. of Technology
1–1 Hibarigaoka, Tempaku-cho
Toyohashi, Aichi 441–8580, Japan

Kiichiro Tamaru

System ULSI Engineering Lab.
TOSHIBA Corporation
580–1 Horikawa-cho, Saiwai-ku
Kawasaki, Kanagawa 210–8520, Japan

Abstract

The ITRON Subproject started several new standardization activities in 1996 and in 1997, with which the second stage of the subproject has been launched. In this paper, we briefly survey these activities and report their recent results. The most important result of them is the proposed specification of μITRON4.0, the next generation μITRON real-time kernel specification. In this paper, the overview of the proposed specification is presented the future plan to complete the specification is described.

1 Introduction

Since the beginning of the TRON Project in 1984, the ITRON Subproject has published a series of ITRON real-time kernel specifications. The reason for putting the emphasis on the kernel specifications is that many embedded systems (especially, small-scale ones) use only kernel functions.

As embedded systems grow larger and more complex, however, the need has increased for standardization efforts that take into account software components, development tools, and other specifications related to embedded system software. Among them, we have determined to put an emphasis on software component-related standardization at first in the ITRON Subproject, and started several standardization activities described below. With these activities, the second stage of the ITRON Subproject has started.

The standardization activities we started in 1996 and 1997 are as follows.

ITRON Hard Real-Time Support Study Group

The ITRON Hard Real-Time Support Study Group was started in November, 1996 to satisfy the preconditions for promoting the development and circulation of software components. Specifically, the study group is focused primarily on the following two themes.

The first is to resolve the issue that porting of software components up to now has been difficult due to the large difference in specifications among ITRON-specification kernel implementations. This requires that the level of standardization of the kernel specifications be raised while retaining the benefits of loose standardization.

The second theme is to support software components with hard real-time characteristics. Many software components demand real-time response. Examples include software modem, voice compression/decompression, and MPEG encoding/decoding. What is needed is a framework that allows coexistence of those software components with applications while satisfying their real-time constraints, and enabling use of multiple software components each with their own real-time needs.

In April, 1997, two working groups, the kernel-specification WG and the application design guideline WG, were formed and have been working on these two themes.

Embedded TCP/IP Technical Committee

In order to promote wide use of software components, the application program interface (API) of software components should be standardized. The ITRON Technical Committee called the establishment of an activity for standardizing the API of the TCP/IP protocol stack, which has taken increasing significance recently. As the result of the call, the Embedded TCP/IP Technical Committee was established and started in April, 1997.

Though the widely used API for the TCP/IP protocol stack today is the socket interface, some inadequacies of the interface for embedded systems have been pointed out. The major goal of the technical commit-

tee is to design a new TCP/IP API which matches the requirements of embedded systems.

RTOS Automotive Application Technical Committee

With current practice, real-time kernels are difficult to apply to vehicle control systems, mainly because vehicle control applications generally require very short response with very limited hardware resources, and because the overhead of real-time kernels is not permissible. In the recently developed systems, however, the control systems grow larger and require more sophisticated run-time software.

The RTOS Automotive Application Technical Committee was started in June, 1997 to bring together the requirements on real-time kernels used in vehicle control systems and to propose a real-time kernel specification suitable for them.

Java Technology on ITRON-specification OS Technical Committee

The ITRON Technical Committee established the Java Technology on ITRON-specification OS Technical Committee in November, 1997 to define the interface specifications between ITRON-specification real-time kernels and Java Application Environments for embedded systems.

Java technologies are becoming increasingly important in embedded systems arena. Although several Java ports have already been created on ITRON-specification kernels, there is no standard reference. The purpose of this committee is to standardize the communication interface between Java applications (or applets) and ITRON tasks, and the implementation approach of Java threads with ITRON tasks. If some extensions to the ITRON specification are found to be necessary, the result will be reflected to the next version of the ITRON specification.

Refer to the separate paper for the current states of this committee [1].

In this paper, we will describe the recent results of these activities.

2 Next Generation μITRON Kernel Specification

The most important result of the recent activities is the requirements on real-time kernels, which will be incorporated to μITRON4.0, the next generation μITRON real-time kernel specification.

Note that the specification described in the following sections is the current snapshot of the discussions. It may be changed until the μITRON4.0 specification is fixed.

2.1 Motivations

The two major motivations for designing new real-time kernel specification are to raise the portability of software components (and application software) developed on the μITRON-specification kernels and to incorporate new kernel functionalities, including the functions supporting hard real-time systems.

2.2 Standard Profile

Concept

The standard profile is a set of real-time kernel functions, defined for raising the portability of software components. The software components (or application software) which are required to be portable among different μITRON kernels are recommended to use only the functions included in the standard profile, and the real-time kernels to which the software components are requested to be portable are recommended to implement all the functions included in the standard profile.

Extensions and subsettings of the standard profile are still permitted in order to retain the advantages of the loose standardization policy of the TRON project.

Function Overview

The standard profile of the μITRON4.0 specification includes almost all level S functions of the μITRON3.0 specification. It also includes some extended functions. Some important extended functions are described in the following sections. Below, we will describe some other major modifications from the level S of μITRON3.0.

At first, fixed-sized memorypool and cyclic handler functions, which are level E functions in the μITRON3.0 specification, are incorporated to the standard profile. The detailed specification of the cyclic handlers is revised. The system calls with timeout (`tslp_tsk`, `twai_sem`, etc.), which are also level E functions in μITRON3.0, are now included in the standard profile.

The API related to interrupt handlers is clarified. At first, the system calls started with "i" (for example, `iwup_tsk`) must be used within interrupt handlers. The newly added `ient_int` should be invoked at the beginning of each interrupt handler described in the C language, and `iret_int` (renamed from `ret_int`) should be invoked at its end. `isig_tim`, which is to inform the kernel of time ticks, is also introduced.

Some system call names are modified for consistency. Specifically, `preq_sem` is renamed to `pol_sem`.

snd_msg and rcv_msg are renamed to snd_mbx and rcv_mbx, respectively.

Each level X option will be determined to be or not to be included in the standard profile. For example, the TA_WMUL option of eventflags, with which multiple tasks can wait for an eventflag, is not included in the standard profile. Other options are still under discussion, currently.

Static Definition of Kernel Objects

The system calls to create and delete kernel objects (cre_tsk, cre_sem, etc.), which are level E functions in μITRON3.0, are not included in the standard profile of μITRON4.0.

In most μITRON3.0 implementations without those system calls, what kernel objects should be created is described in the kernel configuration file, instead. The syntax of the kernel configuration file is not implementation-dependent and is not portable.

In the μITRON4.0 specification, in order to ease the software porting, descriptions in kernel configuration files are standardized. For example, the directive for creating a task is CRE_TSK (not that it is described in capital letters) and the parameters to it are basically the same with the cre_tsk system call. With this approach, application programmers are requested to study one API only.

Exception Handlings

Exception handling functions are totally implementation-dependent in the μITRON3.0 specification. In μITRON4.0, two functions for exception handling, CPU exception handlers and task exception routines, are defined in the standard profile.

The CPU exception handlers are to handle CPU exceptions, such as zero-division or bus error. In the standard profile, though the API to define a CPU exception handler is defined, how to write a CPU exception handler is not standardized. This is because the CPU exception mechanisms of processors have great variety, and because it is difficult to standardize it with low overhead. At least, a task exception routine must be able to be invoked with a CPU exception handler.

The task exception routines are to handle exceptional events in task contexts. One task exception routine can be defined for each task. An exceptional event are raised on a task with ras_tex (or iras_tex) system call. The kind of events is passed in a parameter to ras_tex and is notified to the task exception routine with its parameter.

Terminology

Some terminologies used in the specification are changed or clarified. For example, RUN state, WAIT state, SUSPEND state, and WAIT-SUSPEND state are renamed to RUNNING state, WAITING state, SUSPENDED state, and WAITING-SUSPENDED state, respectively. Another example is that "delayed dispatching" will not be used in the μITRON4.0 specification. The same concept will be described that "the interrupt handlers have higher priority than the task dispatcher" instead.

2.3 Extended Functions for Hard Real-Time Systems

Two functions will be introduced to μITRON4.0 as the extended functions for hard real-time support: mutual exclusion mechanism with priority ceiling and priority inheritance support and overrun detection mechanism. The detailed specification of the functions is still under discussions.

2.4 Vehicle Control Profile

One of the recommendations from the RTOS Automotive Technical Committee is (basically) a subset definition of μITRON including only necessary functions for many vehicle control applications. In addition, the mailbox functions are modified to be convenient for the applications. A mechanism to share a stack space with multiple tasks is also introduced.

Roughly speaking, the subset definition is a bit smaller than the level S functions of μITRON3.0, thus is quite smaller than the standard profile described above. The subset definition will be incorporated to the μITRON4.0 specification, as another profile than the standard profile.

2.5 Real-Time Kernel without Wait State

The other recommendation from the RTOS Automotive Technical Committee is a real-time kernel specification without wait state. In the previous versions of the ITRON kernel specifications, wait state is mandatory. It is thought to be the prerequisite for a real-time kernel.

In recent studies, however, many application systems, especially small-scale systems, do not necessarily require wait state. Without wait state, all the tasks within a system can share one stack area, and thus removing wait state is very effective for decreasing memory consumption and reducing task dispatching overhead. Though it is still questionable if a real-time kernel without wait state can be called as a "real-time

kernel," it is useful to define such real-time kernel specification as a subset of μITRON an introductory specification.

3 Application Design Guidelines

3.1 Motivations

There are two motivations to define application design guidelines for real-time embedded systems. One of them is to provide a standard approach to design an embedded system using a real-time kernel for embedded application designers. For example, how to divide a system into tasks and how to assign priorities to them should be covered. We think that it is impossible to cover all application fields of embedded systems with one set of guidelines, because of the great varieties of embedded systems. The set of design guidelines being defined is an approach focusing on real-time features of embedded systems.

Another motivation is to support software components with hard real-time characteristics. In order to make software components with hard real-time characteristics coexist with applications while satisfying their real-time constraints, both of the software components and the applications should be designed following a set of rules, or design guidelines.

3.2 Overview of the Guidelines

In order to guarantee the real-time constraints of application systems, the application design guidelines adopt the rate monotonic analysis (RMA) [2] as the basic scheduling theory. With the RMA theory, a higher priority should be assigned to a task with shorter deadline (this policy is called deadline monotonic scheduling).

The guidelines are organized as follows. At first, the application system should be divided into processings which are basic computation units constituting the system and which correspond roughly with functions in C language or subroutines in assembler. The guidelines do not cover this step, but show how to construct processings into a task. To do that, parameters representing the real-time characteristics of the processing, including the deadline, the maximum execution time, the maximum execution frequency, and the significance should be listed up.

Then, the processings having the same (or similar) real-time constraints are built up into a task. The priority of a task is assigned according to the deadline monotonic scheduling policy. After those step, the schedulability of the system is checked using the RMA theory. If the system is found to be unschedulable, some kind of tuning process should be applied.

When a sotfware component having real-time constraints is provided, the provided should present the real-time characteristics of the component. The user of the component can check the schedulability of the system consisting of the software component and their own application programs.

4 TCP API for Embedded Systems

4.1 Motivations

As described in Section 1, the socket interface is the most widely used API for TCP/IP protocol stacks. The socket interface, however, has some problems when it is applied to embedded systems. For example, a TCP/IP protocol stack supporting the socket interface must depend on dynamic memory management facility. When memory space is running short, the protocol stack silently discards packets. This is not suitable for many embedded systems.

4.2 Overview of the API

The TCP/IP API proposed by the Embedded TCP/IP Technical Committee is the API for the TCP protocol and the UDP protocol over IPv4 (version 4 of the Internet Protocol). The other APIs necessary for a TCP/IP protocol stack product (for example, APIs for managing the IP routing table and for managing the ARP table) are out of the scope in the current version.

Considering that many existing internet software are based on the socket inferface and that many software engineers are familiar with it, the proposed API is based on the socket interface, and the problems in applying the socket interface to embedded systems are remedied. It is also possible to implement a library implementing the socket interface on top of the proposed API.

Some of the important differences with the socket interface are as follows.

- The API for TCP and that for UDP are separately specified. In other words, the proposed API is protocol-dependent, while the socket interface is protocol-independent. In addition, the current version is focused on IPv4.
- In stead of adopting the socket abstraction, end points for communications are directly handled. Moreover, the end point to wait for TCP connection requests (which corresponds to a socket on which `listen` is called) and the end point for

a TCP connection are managed as different objects, while a socket is an abstraction of both objects. In the proposed specification, the former one is named a TCP reception point (abbreviated as "rep") and the latter one is named a TCP communication end point (abbreviated as "cep").

- A set of TCP APIs with which the number of data copy can be reduced is defined in addition to the usual `read/write` style APIs. Specifically, `tcp_rcv_buf` returns the start address and the length of the buffer in which the received data is stored. After processing the data within the buffer, the application program should call `tcp_rel_buf` to release the buffer space. The APIs for sending data is defined similarly.
- Non-blocking calls and callbacks are supported for asynchronous handling of the protocols (the socket interface of UNIX also supports them). The infamous `select` call is not supported, because it can be emulated with the callbacks and the eventflag functions.
- When an UDP packet is received, the callback function is called instead of storing the packet within a buffer managed by the protocol stack. Within the callback function, the appliation program should allocate a buffer space for the UDP packet and copy the packet to the buffer using `udp_rcv_dat`. Otherwise, the UDP packet is discarded. With this approach, the application can know when memory space is running short.

The first version of the TCP/IP API specification for embedded systems will be finished very soon by the Embedded TCP/IP Technical Committee. After that, the ITRON Technical will approve the specification as an ITRON standard after a review process.

5 Future Plan

In order to complete the μITRON4.0 specification, we will start the activity of the μITRON4.0 Specification Study Group in April, 1998. The study group is an open activity, in that anyone can participate in the activity. According to the current plan, the specification will be finished and published within a half year.

Another important activity we are planning to start within 1998 is the interface standardization between real-time kernels and debugging tools such as software debuggers, in-circuit emulators (ICE), and logic analyzers. With a standard interface between them, making a debugging tool support μITRON-specification kernels becomes easier and we can expect that more software development tools will support μITRON-specification kernels.

6 Summary

In this paper, we have described the recent results of the ITRON Subproject. As the result of the activities since 1996, several outcomes are to be obtained. The resulting specifications and the guidelines will be made open following the basic policy of the TRON Project.

The market environments surrouding the ITRON Subproject are changing very rapidly. We will continue the efforts to catch up the the market requirements and to contribute for the advancement of embedded system technologies.

Acknowledgments

We would like to thank the members of the ITRON Technical Committee and the members of the other ITRON-related study groups and committees described in this paper for their support and efforts for the ITRON Subproject.

References

[1] Y. Nakamoto and H. Takada, "Integration of Java and μITRON," in *Proc. 14th TRON Project Int'l Symposium*, Mar. 1998.

[2] M. H. Klein, T. Ralya, B. Pollak, R. Obenza, and M. G. Harbour, *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems.* Kluwer Academic Publishers, 1993.

The ITRON Technical Committee provides regular information on the ITRON specifications via the Internet, including the latest English-language specifications and the ITRON Newsletter. The URL is "http://tron.um.u-tokyo.ac.jp/TRON/ITRON/".