行政院國家科學委員會補助專題研究計畫成果報告

※※※※※※※※※※※※※※※※※※※※※※※※※※
※　　　　　　　　　　　　　　　　　　　　　　　　※
※　　　即時 Linux -資源管理與實作 (I)　　　※
※　　　　　　　　　　　　　　　　　　　　　　　　※
※※※※※※※※※※※※※※※※※※※※※※※※※※

計畫類別：☑個別型計畫　　□整合型計畫

計畫編號：NSC91-2213-E-002-104

執行期間：91 年 08 月 01 日至 92 年 07 月 31 日

計畫主持人：郭大維教授

共同主持人：

計畫參與人員： 張立平、羅習五、吳卓俊、陳建佳、姜柏任、陳建穎、莊凱翔、許恆瑞

本成果報告包括以下應繳交之附件：

- □ 赴國外出差或研習心得報告一份
- □ 赴大陸地區出差或研習心得報告一份
- □ 出席國際學術會議心得報告及發表之論文各一份
- □ 國際合作研究計畫國外研究報告書一份

執行單位：國立台灣大學資訊工程所

中　華　民　國　92　年　8　月　20　日

行政院國家科學委員會專題研究計畫成果報告

計畫名稱：即時 Linux-資源管理與實作(I)

計畫編號：NSC91-2213-E-002-104
執行期限：計畫自民國 91 年 08 月 01 日起至民國 92 年 07 月 31 日止

主持人：郭大維教授
計畫參與人員：張立平、羅習五、吳卓俊、陳建佳、
姜柏任、陳建穎、莊凱翔、許恆瑞
國立台灣大學資訊工程所

一、摘要

中文摘要：

本研究的主要目的在於設計一個以「執行額度」爲基礎（Budget-Based）的 RTAI 實作，其功能在 x86 的 Linux 機器上輔助即時的 LXRT 工作。RTAI 提供一個輕量（Light-weight）且高效能（High-Performance）的介面，藉由此介面，使用者可以在 Linux 的平台下，能夠實作硬即時（Hard Real-Time）及軟即時（Soft Real-Time）的應用程式。本研究延續 RTAI 既有的功能，允許使用者爲他們的每個即時工作設定一個執行額度，而且相容於之前的 RTAI 版本。不同於其他的研究，我們主要實作執行額度在 LXRT 上的工作，這比單純的 RTAI 更爲複雜。本研究修改了中斷處理、RTAI 排程器、以及函式 rt_task_wait_period()，並沒有更改任何 Linux 的原始碼。本研究也提供了一個監視器的驅動程式，還有該監視器的使用者介面；Linux 核心 2.4.0-test10、RTAI 24.1.2，在 PII 和 PIII 的平台上，是我們實驗的平台。

Abstract :

The purpose is to propose a budget-based RTAI (Real-Time Application Interface) implementation for real-time LXRT tasks over Linux on x86 architectures, where RTAI provides a light-weight and high-performance interface for hard and soft real-time tasks over Linux. RTAI API's are extended for programmers to specify a computation budget for each task, and backward compatibility is maintained to the original RTAI design. Different from the past work, we focus on the implementation of budget-based resource reservation for real-time LXRT tasks, that is complicated by the relationship between RTAI and Linux. Modifications on RTAI are limited to few procedures without any change to the Linux source code, such as the timer interrupt handler, the RTAI scheduler, and `rt_task_wait_period()`. We also propose a monitor driver and a monitor user interface to dynamically adjust the budgets for tasks. The feasibility of the proposed implementation is demonstrated by a system over Linux 2.4.0-test10 and RTAI 24.1.2 on PII and PIII platforms.

關鍵詞：

RTAI, Linux, LXRT Tasks, QoS, Budget Reservation, Real-Time Linux

二、前言

Various levels of real-time support are now provided in most commercial operating systems, such as Windows XP, Windows CE .NET, and Solaris. However, most of them only focus on non-aging real-time priority levels, interrupt latency, and priority inversion mechanisms (merely at very preliminary stages). Although real-time priority scheduling

is powerful, it is a pretty low-level mechanism. Application engineers might have to embed mechanisms at different levels inside their codes, such as those for frequent and intelligent adjustment of priority levels, or
provide additional (indirect management) utilities to fit the quality-of-services (QoS) requirements of each individual task. In the past decade, researchers have started exploring scheduling mechanisms that are more intuitive and better applicable to applications, such as budget-based reservation [1, 2, 3] and rate-based scheduling [4, 5, 6].

Most commercial operating systems now claim the support for real-time applications. VxWorks [7] provides efficient preemptive scheduling, deterministic context switching time, and swift interrupt handling, and is used in many mission-critical applications ranged from anti-lock braking to inter-planetary exploration. pSOSystem 3 [8] is designed for quick development of embedded devices. Its pSOS+ 3 multitasking kernel is small, fast, reliable, and deterministic. RTAI (Real-time Application Interface) proposed by Mantegazza, et al. [9] at DIAPM shares a very similar system architecture with RTLinux proposed by Yodaiken, et al. [10]. All interrupts are intercepted by a tiny kernel under Linux such that hard real-time tasks are supported by the tiny kernel, where Linux is considered as a ``process" for the tiny kernel. BlueCat RT from LynuxWorks adopts RTLinux inside its RT kernel. Embedix Realtime, i.e., Lineo's real-time Linux distribution, is derived based on RTAI. MontaVista provides a fully preemptible Linux kernel, where kernel-mode tasks could be interrupted, and rescheduling is possible. REDICE-Linux [11] delivered by RedSonic represents a hybrid real-time Linux solution, which revises Linux and integrates RTAI inside its kernel. One integrated kernel is used to schedule all real-time and non-real-time tasks. Solaris 8 supports high resolution timers and user-level priority inheritance, where priorities from 100 to 159 are belonging to real-time tasks, priorities from 160 to 169 are for interrupt servicing threads, and the rest are for ordinary user tasks. Windows CE .NET provides 256 non-aging real-time priority levels and bounds the interrupt service routine (ISR) latency within 1ms. A very preliminary one-level priority inheritance is provided to allow a task to inherit the priority of a higher-priority task blocked by the former task. RTX from VenturCom is widely adopted by many Windows-family operating systems and vendors to provide high-performance real-time support. A preemptive scheduling policy with priority promotion to prevent priority inversion is provided.

The concept of budget-based reservation, that is considered as an important approach for applications' QoS support, was first proposed by Mercer, et al. [3]. A microkernel-based mechanism was implemented to let users reserve CPU cycles for threads. Windows NT middlewares [1, 2] were proposed to provide budget reservations and soft QoS guarantees for applications over Windows NT. REDICE-Linux implemented the idea of hierarchical budget groups to allow tasks in a group to share a specified amount of budget. There were also many other research and

implementation results on the QoS support for real-time applications. In particular, Adelberg, et al. [12] presented a real-time emulation program to build soft real-time scheduling on the top of UNIX. Childs and Ingram [13] chose to modify the Linux source code by adding a new scheduling class called SCHED_QOS which let applications specify the amount of CPU time per period. Abeni, et al. presented an experimental study of the latency behavior of Linux [14]. Several sources of latency was quantified with a series of micro-benchmarks. It was shown that latency was mainly resulted from timers and non-preemptable sections. Swaminathan, et al. explored energy consumption issues in real-time task scheduling over RTLinux [15].

三、研究目的

The purpose of this paper is to explore budget-based resource reservation for real-time LXRT tasks which run over Linux and propose its implementation. We first extend RTAI API's to provide hard budget guarantees to hard real-time tasks under RTAI. We then build up an implementation for soft budget guarantees for LXRT tasks over that for hard real-time tasks under RTAI. Backward compatibility is maintained for the original RTAI (and LXRT) design. We present our software framework and patch for the proposed implementation. We try to minimize the modifications on RTAI without any change to the Linux source code. Modifications on RTAI are limited to few procedures, such as the timer interrupt handler, the RTAI scheduler, and `rt\_task\_wait\_period()`. A monitor driver and a monitor user interface are designed and implemented for users to dynamically assign/modify the budgets of tasks. The feasibility of the proposed implementation is demonstrated over Linux 2.4.0-test10 and RTAI 24.1.2 on PIII and PII platforms. We conducted a series of experiments to evaluate the overheads and budget precision of the proposed implementation.

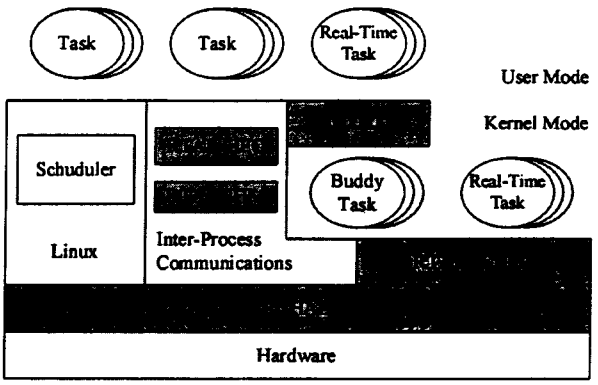

Figure 1: The RTAI architecture

四、實作方法

Hard Budget-Based Resource Reservation

Each real-time RTAI task could be assigned a hard execution budget per period. The process control block of each real-time RTAI task must include additional members. We must include additional members in the process control block of real-time RTAI tasks (`rt_task_struct`) and revise minor parts of the RTAI scheduler (`rt_schedule()`) and the timer interrupt handler (`rt_timer_handler()`) to guarantee hard budget-based resource reservation. The implementation of budget-based resource reservation in RTAI must consider two important issues: 1) The correct programming of the timer chip. Because the resumption and the suspension of a real-time RTAI task are

both driven by the timer, the budget consumption of each task must be considered in the programming of the timer chip. That is, when a task runs out of its budget before it calls `rt_task_wait_period()`, the system must suspend the execution of the task until the next period. Note that the design should not cause any interference with the real-time scheduler in RTAI. This implementation should only focus on the budget overrunning of real-time RTAI tasks. 2) The behavior of `rt_task_wait_period()`. When a task runs out of its budget before it calls `rt_task_wait_period()`, the execution might be suspended (before the invocation of `rt_task_wait_period()`). The remaining execution code of the former period might be delayed to execute in the next period. As a result, it is possible to have more than one invocations of `rt_task_wait_period()` in a period becuare one invocation that should happen in the former period occurs in the next period. The invocation that should happen in the former period will be simply ignored in the next period. The rationale behind this semantics is to let the task gradually catch up the delay, due to overrunning in some period.

Soft Budget-Based Resource Reservation

The budget-based resource reservation implementation for real-time LXRT tasks is on the top of the corresponding implementation for real-time RTAI tasks. A real-time LXRT task is initialized by invoking `rt_task_init()`, and a buddy RTAI task is created by RTAU to execute the RTAI services requested by the real-time LXRT task.

There are two major challenges in the implementation of soft budget-based resource reservation for real-time LXRT tasks, beside the challenges on correct timer-chip programming and `rt_task_wait_period()` modifications: 1) How to interrupt a real-time LXRT task when its budget is exhausted and to transfer the CPU execution right to a proper task. 2) When a higher-priority real-time LXRT task arrives, how to interrupt a lower-priority real-time LXRT task and dispatch the higher-priority real-time LXRT task.

We propose to use *signals* to resolve the first challenging item. That is how to interrupt a running real-time LXRT task and to trigger rescheduling when its budget is exhausted. Note that we wish to restrict modifications on RTAI without any change to the Linux source code.

五、結果與討論

The purpose of this paper is to explore issues on budget-based resource reservation for real-time LXRT tasks and to propose its implementation. Note that although LXRT seems an important implementation for real-time Linux implementations, little work is done on the resource reservation issues. We extend RTAI API's to provide hard budget guarantees to hard real-time tasks under RTAI. We then build up an implementation for soft budget guarantees for LXRT tasks over that for real-time RTAI tasks. Backward compatibility is maintained for the original RTAI (and LXRT) design. We present our software framework and patch for the proposed implementation. We try to minimize the modifications on RTAI without any change to the Linux source code. Modifications on RTAI are limited to few

procedures, such as the timer interrupt handler, the RTAI scheduler, and `rt\_task\_wait\_period()`. A monitor driver and a monitor user interface are designed and implemented for users to dynamically assign/modify the budgets of tasks. The feasibility of the proposed implementation is demonstrated over Linux 2.4.0-test10 and RTAI 24.1.2 on PIII and PII platforms. We conducted a series of experiments to evaluate the overheads and budget precision of the proposed implementation. It was shown that the system overheads were very limited for the resource reservations of real-time RTAI tasks and real-time LXRT tasks. For example, even on PII 266, the maximum overhead of real-time RTAI tasks on resource reservation was only 10μs, and the average overheads were less than 7μs. On PII 266, the maximum overhead of real-time LXRT tasks on resource reservation was only 96μs, and the average overheads were less than 34.69μs. The precision of budget-based resource reservation was also very good. For computation-intensive tasks, a very few number of violations was observed. Even for a complicated workload, such as a MPEG player, the number of violations was only 10% of the total simulation periods.

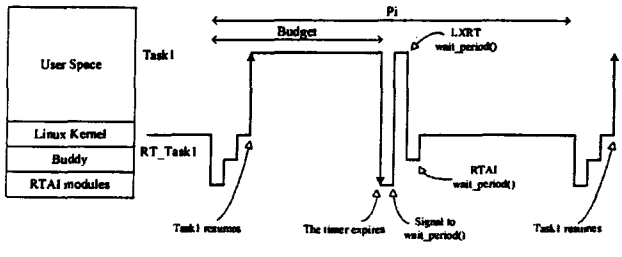

Figure 2: Scheduling flowchart of the revised LXRT

六、參考文獻

[1] M.B. Jones and D. Rosu and M.C. Rosu. CPU Reservation and Time Constraints: Efficient, Predictable Scheduling of Independent Activities. *ACM Symposium on Operating Systems Principles*, 1997.

[2] Tei-Wei Kuo and Guin-Haur Huang and Shie-Kai Ni. A User-Level Computing Power Regulator for Soft Real-Time Applications on commercial Operating Systems. *Journal of the Chinese Institute of Electrical Engineering*, 1999.

[3] Clifford W. Mercer and Ragunathan Rajkumar. An Interactive Interface and RT-Mach Support for Monitoring and Controlling Resource Management. *IEEE Real-Time Technology and Applications Symposium*, 1995.

[4] Z. Deng and J. W.-S. Liu. Scheduling Real-Time Applications in an Open Environment. *IEEE Real-Time Systems Symposium*, 1997.

[5] M. Spuri and G. Buttazzo and Sensini. Scheduling Aperiodic Tasks in Dynamic Scheduling Environment. *IEEE Real-Time Systems Symposium*, 1995.

[6] C.A. Waldspurger and W.E. Weihl. Stride Scheduling Deterministic Proportional Share Resource Management. *Technical Memorandum*, MIT/LCS/TM-528, Laboratory for CS, MIT, 1995.

[7] Henry Neugass and Geoffrey Espin and Hidefume Nunoe and Ralph Thomas and David Wilner. VxWorks: an Interactive Development Environment and Real-Time Kernel for Gmicro. *TRON Project Symposium, 1991.Proceddings.*, Eighth, 1991.

[8] L.M. Thompson. Using pSOS+ for embedded real-time computing. *Compcon Spring '90. 'Intellectual Leverage'. Digest of Papers. Thirty-Fifth IEEE Computer Society International Conference*, 1991.

[9] P. Cloutier and P. Mantegazza and S. Papacharalambous and I. Soanes and S. Hughes and Karim Yaghmour. DIAPM-RTAI POSITION PAPER. *Real Time Operating Systems Workshop*, 2000.

[10] V. Yodaiken. The RTLinux Manifesto. *Proceedings of the 5th Linux Expo*, 1999.

[11] YuChung Wang and Kwei-Jay Lin. Enhancing the Real-Time Capability of the Linux Kernel. *the 5th Real-Time Computing Systems and Applications Symposium*, 1998.

[12] B. Adelberg and H. Garcia-Molina and B.Kao. Emulating Soft Real-Time Scheduling Using Traditional Operating Systems Schedulers. *IEEE 15th Real-Time Systems Symposium*, 1994.

[13] S. Childs and D. Ingram. The Linux-SRT Integrated Multimedia Operating Systems: Bring QoS to the Desktop. *IEEE Real-Time Technology and Applications Symposium*, Taipei, Taiwan, ROC, 2001.

[14] Luca Abeni and Ashvin Goel and Charles Krasic and Jim Snow and Jonathan Walpole. A Measurement-Based Analysis of the Real-Time Performance of Linux. *Eighth IEEE Real-Time and Embedded Technology and Applications Symposium*, 2002.

[15] Vishnu Swaminathan and Charles B. Schweizer and Krishnendu Chakrabarty and Amil A. Patel. Experiences in Implementing an Energy-Driven Task Scheduler in RT-Linux. *Eighth IEEE Real-Time and Embedded Technology and Applications Symposium*, 2002.