行政院國家科學委員會專題研究計畫　期中進度報告

即時資源配置：理論與即時作業系統實作(1/3)

計畫類別：個別型計畫
計畫編號：NSC92-2213-E-002-065-
執行期間：92 年 08 月 01 日至 93 年 07 月 31 日
執行單位：國立臺灣大學資訊工程學系暨研究所

計畫主持人：郭大維

計畫參與人員：吳晉賢，陳雅淑，謝仁偉，楊川岳，楊東偉，蔡易霖，張立平，黃志源

報告類型：精簡報告

處理方式：本計畫可公開查詢

中　華　民　國 93 年 5 月 24 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：即時資源配置-理論與即時作業系統實作（1/3）

計畫編號：NSC92-2213-E-002-065

執行期限：計畫自民國 92 年 08 月 01 日起至民國 93 年 07 月 31 日止

主持人：郭大維

計畫參與人員：吳晉賢, 陳雅淑, 謝仁偉, 楊川岳, 楊東偉, 蔡易霖, 張立平, 黃志源

執行單位 ： 國立台灣大學資訊工程所

一、摘要

中文摘要：

資源配置向來是即時系統中一個重要的課題。除了以往所考慮的處理器排程，在先進的計算環境中，輸出輸入子系統目前也提供一些基本的排程機制。應用程式是否能夠獲得預期的效能保證，取決於處理器以及輸出輸入系統各方面是否能夠提供適當的支援。本計畫針對即時作業環境 RTAI-Linux 與 Universal Serial Bus 子系統作為一個整合性的研究目標。Universal Serial Bus (USB)是相當普及的個人電腦之週邊設備溝通協定標準。雖然 USB 規格中定義提供了等時傳輸模式以及大量傳輸模式的支援，然而目前卻鮮少研究討論關於保證每個設備在特定時間單位內保證最低可使用之頻寬。我們提出一個基於 cyclic-executive 的頻寬保留和排程方法，對等時性的工作提供 QoS 的保證。此外, 在處理器排程方面，為了利用 RTAI-Linux 達到可容忍些許誤差的 QoS 保證, 本研究中我們將 budget-based 資源保留的方法實作。綜合上述處理器以及 USB 方面的資源配置方法，其整體的有效性已於實作視訊監視系統中獲得驗證。

***Abstract*：**

Resource partition is an important issue in the design of real-time systems. In current computing system, many I/O devices now support scheduling policies. The QoS guarantee of applications must be provided for services over CPU and I/O devices. We realize the objective in QoS guarantees over RTAI-Linux and the USB subsystem. Universal Serial Bus (USB) is a popular standard for PC peripheral devices because of its versatile peripheral interconnection specifications. USB not only provides simplified hardware connectors but also supports for various bus traffics, such as isochronous and bulk transfer activities. Although the USB specifications provide a way for users to specify the upper bound on the number of bytes for each data transfer in a 1ms time frame, little work is done to provide QoS guarantees for devices (e.g., the lower bound on the bytes for each device type in a 1ms time frame) with resource reservation or resource partition techniques and a mechanism in enforcing the guarantees. In this research, we propose a cyclic-executive-based bandwidth reservation and scheduling method to support QoS guarantees over USB, especially for those isochronous bus activities. The proposed bandwidth reservation/partition, scheduling method and admission control method could reserve USB bandwidth for devices in an on-demand fashion. Besides, for soft QoS guarantee over RTAI-Linux, we also design and evaluate our budget-based resource reservation method over RTAI- Linux on x86 architectures.

The capability of the proposed scheme was shown by the implementation and demonstration of the proposed resource reservation methods for CPU and USB.
關鍵詞：

QoS, resource reservation, real-time scheduling, RTAI, USB

二、前言

With the advance of hardware and software technology, computer systems are now very modularized. Various subsystems are interconnected with proper interface definitions. Consider I/O subsystems as an example. Various I/O devices are manufactured and connected to few common interfaces, such as SCSI, USB, and IEEE-1394. The needs for resource allocation no longer remains at the so-called kernel part. Instead, I/O subsystems (and of course many other subsystems) now become one of the major players in providing a quality-of-service (QoS) guarantee for applications. The playing of a video stream from a disk serves as a good example in exploring the major parties involved in the resource allocation of system resources.

Real-time scheduling algorithms [5] were often adopted by researchers in enforcing resource reservation. In the past decades, researchers and the industry have developed techniques in providing the QoS guarantees over traditional operating systems. RTLinux [7] is a real-time extension of Linux to deliver hard real-time capabilities and regular Linux services to applications at the same time. Most commercial operating systems now claim the support for real-time applications. VxWorks [11] 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 [12] 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. [13] at DIAPM shares a very similar system architecture with RTLinux proposed by Yodaiken, et al. [14]. Modifications to the Linux operating system were also made [2, 6] to provide real-time and QoS scheduling for various resources. In addition to the QoS reservation for CPU time, researchers started exploring the resource allocation problems and their QoS issues over various buses. In particular, the control area network (CAN), which is a serial communication protocol for distributed real-time system, was studied by Hon and Kim [3] in proposing a bandwidth reservation algorithm. Kettler, et al. [4] developed formal scheduling models for several types of commonly used buses, e.g., PCI and MCA, in PC’s and workstations.

三、研究目的

The objective of this research is to explore the QoS issues of USB subsystem over budget-based RTAI. USB is designed to support many types of devices, such as human interface devices (e.g., keyboard and mouse), block devices (e.g., disks), communication transceivers, stereo speakers, video cameras, etc. The data transfer modes on USB could be classified into four categories: isochronous transfer, interrupt transfer, control transfer, and bulk transfer. Isochronous transfer and interrupt transfer are periodic data transfers, while

control transfer and bulk transfer introduce aperiodic bus activities. Different types of devices require different bandwidths and ways to interact with USB host controllers, where a host controller manages the bandwidth of a USB bus. For example, a human interface device (HID) demands a periodic but light bus workload. A storage device requires a best effort service from the corresponding host controller. On the other hand, an operating system must manage a number of devices through USB simultaneously, and devices would compete for a limited bus bandwidth. How to properly manage USB bandwidth is of paramount importance if reasonable QoS requirements are, indeed, considered.

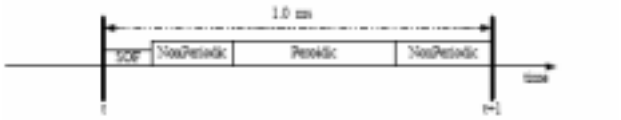

Figure 1. The bandwidth reservation of the USB 1ms time frame.

Although the original design of USB does have an interface definition for QoS (in terms of the percentage of the bandwidth for each type of devices and the upper bound on the number of bytes per data transfer for a device type in a 1ms time frame), there is little work being done for the QoS guarantee of each device (e.g., the lower bound on the number of bytes per device in a 1ms time frame). According to the USB specifications, isochronous and interrupt transfers could be allocated up to 90% of the total bandwidth of a USB bus. Control transfers could have up to 10% of the total USB bandwidth. Bulk transfers are serviced in a best-effort fashion to utilize the remaining USB bandwidth. The USB bandwidth reservation scheme in each 1ms time frame is described in Figure 1, where SOF stands for the starting of the frame, and time frames of USB 1.1 are allnin 1ms. Each USB host controller exercises a series of data structures which are constructed by the operating system to manage devices, as shown in Figure 2. Those data structures are called Transfer Descriptors. The technical issue is how to set up proper in-core data structures so that a host controller could behave and communicate with devices as users request and meet their QoS requirements. The research objective of this paper is to propose a proper bandwidth reservation and scheduling method to support the specified QoS requirements. We shall design a real-time scheduling layer inside the USB driver to let the entire system complying with the USB specification.

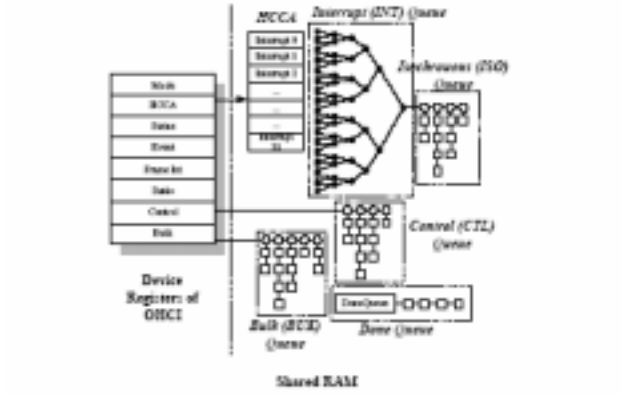

Figure 2. The USB communication channel.

We also conducted a series of experiments to evaluate the overheads, budget precision and QoS guarantee of the proposed reservation method over Linux.

四、實作方法

A Period Modification Policy and Admission Control

For the purpose of USB bandwidth reservation, we include additional information

regarding deadlines, service frequencies, and USB bandwidth requirements inside URB, where URB is a data structure used for the interchanging of control and data information between the USB Core and device drivers. In Linux, the USB Core is named as the USB driver (USBD). An abstracted real-time scheduling layer is proposed to communicate between the device driver layer and the queues in USBD. It determines how Transfer Descriptors of a request are inserted into the Endpoint Descriptor of the device to meet the QoS requirements of devices.

The Linux USB OHCI driver restricts periodic service requests (for interrupt and isochronous devices) to a set of fixed periods (e.g., 1, 2, 4, .., and 32ms). An OHCI host controller travels through each path from a leaf node to the root node of the corresponding polling binary tree. A node could have a list of Endpoint Descriptors, and an Endpoint Descriptor could have a list of Transfer Descriptors. An OHCI host controller processes pending Transfer Descriptors on its way from a leaf node to the root node. The handling of pending Transfer Descriptors on each path should not be more than 1ms; otherwise, the host controller will abort the path and then continue its work to the next path according to the service sequence. Any aborted path is considered as an overflowed path. As a result, any nodes at the ith level (counted from the root node) are for services at every 2^i ms.

When the Endpoint Descriptor of a request is moved from a node to the root node, extra protocol overheads will occur, and the Endpoint Descriptor should be revised to fit the polling periods of the former node and the root node.

The admission control policy is defined as follows: Because the period modification policy moves all Endpoint Descriptors to the root node, and the root node is serviced for every 1ms, all requests are schedulable if the services of all Endpoint Descriptors could be done within 1ms. Because only 1500 bytes could be transferred within every 1ms under USB1.1, the total number of bytes for the services of all Endpoint Descriptors should be no more than 1500. That is

$$\sum_{i=1}^{n}\left(\left\lceil \frac{b_i}{p_i} \right\rceil + O\right) <= 1500$$

where b_i denotes the number of bytes to be transferred for every p_i ms, and O denotes the protocol overheads of an USB payload. The overheads include packet preambles, packet headers, and some other necessary fields.

Insertions of Endpoint and Transfer Descriptors

The objective of choosing proper nodes in the tree structure for the inserting of proper Endpoint and Transfer Descriptors is to move (and modify) Endpoint and Transfer Descriptors to nodes of larger heights to minimize the protocol overheads and the additional bandwidth requirements due to the period modification policy

Given a collection of admitted requests, we shall try to push Endpoint Descriptors (ED's) and their Transfer Descriptors (TD's) at the root node down to nodes of larger heights as far as possible. The further we push the Endpoint and Transfer Descriptors away from the root node, the lower the protocol overheads would be. There are two basic guidelines for

this ED/TD reinsertion policy: The first guideline is that the destination node of an original ED (and its TD's) should not have a polling period longer than the period p_i of the corresponding request. The second is that the ED/TD reinsertion policy should meet the 1ms time frame for each path of a tree structure. However, the ED/TD reinsertion problem is a combinatorial problem. We proposed a greedy algorithm for ED/TD reinsertion.

The evaluation of soft QoS guarantee over RTAI-Linux

For QoS guarantee, we proposed a budget-based resource reservation method which could be implemented for real-time tasks over RTAI- Linux on x86 architectures. It could guarantee good budget precision on the resumption, suspension, and preemption of real-time RTAI tasks. In this work, we also demonstrate the capability of the proposed method.

Real-time RTAI tasks always met the specified budget-based resource reservation. In order to evaluate the system overheads of the implementation, we measured the actual budget received by a real-time RTAI task, referred to as the *received budget*. The assigned budget of a real-time RTAI task is the amount specified by users for resource reservation, referred to as *assigned budget*. As a result, the received budget of a real-time RTAI task is the assigned budget minus the system overheads (if no compensation mechanism is adopted). Let the number of timer expirations during each servicing of the assigned budget be *N*. The relationship between an assigned budget and the corresponding received budget was as follows, where *N* could change among periods:

*ReceivedBudget=AssignedBudget-N*SystemOverhead*

The experimental results were shown in the following table. The simulation time was 500ms. The average overheads was derived by dividing the total system overheads by the number of timer expirations over the simulation time. The maximum overhead was the maximum observed system overheads for a timer expiration over the simulation time.

Platform	PIII 500	PIII Celeon 400	PII 266
Average Overhead	0.00578 ms	0.00688 ms	0.00638 ms
Maximum Overhead	0.007 ms	0.008 ms	0.010 ms

The received budget of a real-time LXRT task is the assigned budget minus the system overheads, including the execution of related code implemented in RTAI, the context switchings of user and kernel space, the cost in signal posting/delivery, etc. The relationship between an assigned budget and the corresponding received budget was as follows:

ReceivedBudget
=AssignedBudget-N*SystemOverhead-Kernel Delays

The system overheads of the implementation for real-time LXRT tasks were shown in the following table.

Platform	PIII 500	PIII Celeon 400	PII 266
Average Overhead	0.01982 ms	0.02708 ms	0.03469 ms
Maximum Overhead	0.035 ms	0.056 ms	0.096 ms

Since real-time LXRT tasks were Linux tasks, the experiments on the budget precision of the implementation were complicated by including more tasks with different characteristics, such as the playing of MPEG streams. This part of experiments consisted of four real-time LXRT tasks, and the last one was a real-time LXRT task that read from the disk for the MPEG streams and outputted the decoded video through the X server. The precision of soft budget-based resource

reservation was measured in terms of the ratio of the received budget and the assigned budget, referred to as the *budget rate*:

$$BudgetRate(\%) = \frac{RceivedBudget}{AssignedBudget} \times 100\%$$

From the experimental results, it shows that the budget precision was 100% for the most of the time. Over the entire experiment interval, no more than 3 violations were observed over 500 periods. We also evaluate the precision of soft budget-based resource reservation for the MPEG player. Many more violations occurred, because the MPEG player made a lot of system calls, and synchronization existed between the MPEG player and the X server. Around 10% of the periods in which the MPEG player received a budget different from the assigned budget. However, the quality of MPEG movie playing still seemed very good.

五、結果與討論

Modern operating systems are often pretty modularized. The delivering of QoS guarantees to applications is the responsibility of not only the kernel but also all of the subsystems involved in the servicing of the applications. In this project, we proposed a cyclic-executive-based bandwidth reservation methodology to guarantee the QoS requirements of USB devices and real-time applications. A bandwidth reservation algorithm is proposed to partition available USB bandwidth among devices which need different attentions from the system. We conducted a series of experiments to evaluate the performance of the proposed admission control policy. It was shown that the acceptance ratio of the proposed admission control policy was close to that of the exhaustive-search-based optimal algorithm when the period range was selected among ranges [1ms, 8ms] and [1ms, 32ms]. When the periods of requests were chosen in the range [16ms, 32ms], the proposed admission control policy would reject more workloads than the exhaustive-search-based optimal algorithm. It was because of the protocol overheads due to the period transformation of the admission control policy. Our work results in a more precise way in guaranteeing the QoS requirements of devices, and better USB bandwidth utilization could be achieved by demonstrated over Linux 2.4.0-test10 and RTAI 24.1.2 which adopted a budget-based method on PIII and PII platforms. For the future research, we shall further explore the USB bandwidth reservation for USB 2.0 which supports transfer rate up to 480 bits per second. The combination of USB 1.1 and 2.0 bandwidth reservation could be a very challenging issue.

六、參考文獻

[1] T.P. Baker and A. Shaw, "The cyclic executive model and Ada," Real-Time Systems Symposium, 1988.

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

[3] S.H. Hong and W.-H Kim, "Bandwidth allocation in CAN protocol," Proceedings of the IEEE Control Theory and Applications, Jan 2000.

[4] K. A. Kettler, J.P. Lehoczky, and J. K. Strosnider, "Modeling bus scheduling policies for real-time systems," IEEE Real-Time Systems Symposium, Dec, 1995.

[5] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard

Real-Time Environment," JACM, Vol. 20, No. 1, January 1973.

[6] Y.-C. Wang and K.J. Lin, "Implementing a General Purpose Real-Time Scheduling Framework in the RED-Linux Real-Time Kernel," IEEE Real-Time Systems Symposium, Arizona, USA, 1999.

[7] V. Yodaiken, "The RT-Linux approach to hard realtime," Department of Computer Science Socorro NM 87801.

[8] Compaq, Intel, Microsoft, and NEC, "Universal Serial Bus Specification," 1998.

[9] Intel, "Universal Host Controller Interface (UHCI) Design Guide, Revision 1.1," 1996.

[10] Compaq, Microsoft, and National Semiconductor, "Open Host Controller Interface Specification for USB," 1996.

[11] 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.

[12] 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.

[13] 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.

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