# 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(2/3)

計畫類別：個別型計畫
計畫編號：NSC93-2213-E-002-031-
執行期間：93 年 08 月 01 日至 94 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：楊佳玲，逄愛君，陳建佳，許恆瑞，莊凱翔，黃志源

報告類型：精簡報告
報告附件：出席國際會議研究心得報告及發表論文
處理方式：本計畫可公開查詢

中　華　民　國 94 年 5 月 27 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（2/3）**

**計畫編號：NSC93-2213-E-002-031**

**執行期限：計畫自民國 93 年 08 月 01 日起至民國 94 年 07 月 31 日止**

**主持人：郭大維**

**計畫參與人員：楊佳玲, 逄愛君, 陳建佳, 許恆瑞, 莊凱翔, 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。在本研究中，我們考慮了在一即時系統中, 多處理器的省電排程以及 Universal Serial Bus (USB) 1.1/2.0 子系統排程。近幾年來，省電排程已經成為系統設計中十分重要的課題，相異於之前多數的研究，我們探討不同耗電特性的即時工作的排程，使得所有工作都可以在時限之前完成，而且最小化系統耗電。假設所有的工作都有一個共同的時限且在時間 0 可以開始執行，當系統允許工作在多顆處理器上執行，本研究提出一個演算法可以得到最佳的省電排程，當每個工作必須在一個處理器上完成，本研究提出一個具有 1.412 近似比(Approximation ratio)的演算法。對於 USB 2.0/1.1 子系統，本研究提出對於週期性(如同時性)與非週期性(如非同時性)即時工作的資源分配。對於週期性工作，本研究所提出的方法可以保證可排程，對於非週期性的工作可以有機率上有效能保證。而像控制及 **bulk** 傳輸等偶發性的要求,則需給予機率效能的保證此研究成果強調在 Linux 系統標準下之效能評估, 在此我們已有令人振奮的結果。

***Abstract*：**

Research allocation is an important issue for real-time systems. In this research, we explore energy-efficient scheduling in a multiprocessor environment and efficient real-time scheduling for Universal Serial Bus (USB) 1.1/2.0 subsystems. In the past decades, a number of research results have been reported for energy-efficient scheduling over uniprocessor and multiprocessor environments. Different from many of the past results on the assumption for task power characteristics, we consider real-time scheduling of tasks with different power characteristics. The objective is to minimize the energy consumption of task executions under the given deadline constraint. When tasks have a common deadline and are ready at time 0, we propose an optimal real-time task scheduling algorithm for multiprocessor environments with the allowance of task migration. When no task migration is allowed, a 1.412-approximation algorithm for task scheduling is proposed for different settings of power characteristics.

This research also aims at the proposing of an USB-compliant system architecture and real-time scheduling algorithms for the resource allocation of USB 2.0 and 1.1 device requests jointly in a Quality-of-Service (QoS) fashion. Periodic requests, such as isochronous interrupt transfers, are guaranteed with preservation of bus bandwidth and schedulability tests. Sporadic requests, such as control and bulk transfer, are serviced with probabilistic performance guarantees. The capability of this work is demonstrated with performance evaluations over a Linux system prototype, for which we have encouraging results.

**關鍵詞：**

QoS, energy-efficient, multiprocessor

scheduling, resource reservation, real-time scheduling, USB

## 二、前言

With the advanced technology in VLSI circuit designs, many modern processors could now operate at various supply voltages, where different supply voltages lead to different processing speeds. Many computer systems, especially embedded systems, adopt not only voltage-scaling processors but also various energy-efficient strategies in managing their subsystems intelligently. Beside the energy-efficiency designs for battery-powered systems, how to reduce energy consumption for multi-processor systems, such as server farms, also receives a lot of attention in the past decade. As pointed out in [1], multiprocessor implementations of real-time systems could be more energy-efficient than uniprocessor implementations, due to the convex power consumption functions.

Energy-efficient scheduling is to derive a schedule for real-time tasks with minimization on the energy consumption such that the timing constraints can be met. The considerations of timing constraints in task scheduling significantly complicate the problems in energy-efficient scheduling. Uniprocessor energy-efficient scheduling problems have been widely explored, e.g., [2,3,4,5,6]. The energy-efficient real-time task scheduling problems over multiprocessors are often NP-hard. When all of the power consumption functions of tasks are the same, Chen, et al. [7,8] proposed approximation algorithms to schedule frame-based tasks over multiprocessors with and without independent voltage scalings, where all the tasks share a common deadline and arrive at the same time. In [9,10,11], energy-efficient scheduling algorithms based on list heuristics were proposed to schedule real-time tasks with precedence constraints. Mishra, et al. [12] explored energy-efficient scheduling issues with the considerations of task communication delay.

While various I/O devices are manufactured and connected to few common interfaces, such as SCSI, USB, and IEEE1394[22], I/O subsystems now become one of the major players in providing QoS guarantees for applications. The scheduling of CPU usage alone could not resolve the QoS issues. Instead, resource allocation problems on disks, I/O bus, or even networks, could no longer be negligible.

How to provide QoS guarantees for task executions has been an active research and implementation topic in the past decades Several good system implementations were done over RTLinux [21], that is a real-time extension of Linux, such as those for motor control [20], Ethernet-based real-time communication [17], and a lightweight TCP/IP stack for embedded systems [16]. A real-time emulation program was proposed to build soft real-time scheduling on the top of UNIX [14], and the Linux source code was modified by adding a new scheduling class called SCHQH QOS to let applications to specify the amount of CPU time per period [15]. QoS scheduling is provided for CPU and IDE disks.

## 三、省電排程

Each task $\tau_i$ is characterized by its worst-case execution CPU cycles $c_i$ and power

$$P_i(s) = C_i V_{dd}^2 s, \qquad (1)$$

where $s = \beta((V_{dd}-V_t)^2)/V_{dd}$, and $s$, $C_i$, $V_t$, $V_{dd}$, and $\beta$ denote the processor speed, the effective switch capacitance, the threshold voltage (the minimum voltage that can be supplied to the processor for correct functionality), the supply

voltage, and a hardware-design-specific constant, respectively ($V_{dd} \geq V_t \geq 0$, $\beta > 0$, and $C_i > 0$) [13]. The value of the effective switch capacitance is highly related to the software implementations and the execution path of each task (which could be usually derived by profiling). Each power consumption function $P_i(s)$ can be phrased as $hs^{\alpha}$, where $\alpha$ is a hardware-dependent factor, and $h_i$ is a parameter related to the corresponding task execution.

Suppose that each processor could operate at a speed in [0, ∞], and the speed of each processor could be adjusted independently from each another. We assume that the number of CPU cycles executed in a time interval is linearly proportional to the processor speed, and that the energy consumed for a processor in the execution of a task at the processor speed *s* for *t* time units is the multiplication of its corresponding power consumption at the speed *s* and *t*. Let the amount of CPU cycles completed for a task running at a speed s for *t* time units be the multiplication of *s* and *t*. Assume that the time and energy overheads required on speed/voltage switching be negligible. The energy consumption function $E_i()$ of $\tau_i$ is defined as a function of the execution time $t_i$ of $\tau_i$: $E_i(t_i) = P_i(c_i/t_i)\ t_i = h_i\ c_i^{\alpha}/t_i^{\alpha-1}$. Note that the energy consumption function of the execution time of $\tau_i$ is a strictly convex and *decreasing* function.

- ***Problem Definition***

**Multiprocessor Energy-Efficient Scheduling with Task Migration (MEESM):** Consider a set **T** of independent tasks over *M* identical processors, where all tasks in **T** are ready at time 0 and share a common deadline *D*. Each task $\tau_i$ in **T** is associated with a computation requirement equal to $c_i$ CPU-cycles and a power consumption function $P_i()$ of a given processor speed. The objective is to derive a schedule for **T** such that all of the tasks in **T** complete before *D*, the total energy consumption is minimized, where task migration among processors is allowed.

A variation of the MEESM problem without task migration could be defined similarly as follows: **Multiprocessor Energy-Efficient Scheduling without Task Migration (MEES)**: The input, output, and objective of this problem are as the same as their counterparts of the MEESM problem, where no task migration among processors is allowed.

- ***Optimal Schedule for the MEESM Problem***

*Lemma 1: Given a feasible assignment V of execution times of tasks in* ***T****, a feasible schedule $S_V$ can be derived in O(|**T**|) such that the energy consumption of $S_V$ is equal to that of* V.

*Lemma 2: When |**T**| > M, there exists an optimal schedule which executes some task at any time instant between time 0 and time D on each of the M processors.*

Taking both Lemmas 1 and 2 into considerations at the same time, the MEESM problem can be formulated as a convex programming problem, as follows:

$$\begin{array}{ll} \text{minimize} & \sum_{\tau_i \in \mathbf{T}} E_i(t_i) \\ \text{subject to} & \sum_{\tau_i \in \mathbf{T}} t_i = M \cdot D \text{ and} \\ & 0 < t_i \leq D \quad \forall \tau_i \in \mathbf{T}. \end{array} \qquad (2)$$

In the following, we first obtain an optimal solution by ignoring the condition $t_i > 0$. After that, we show that $t > 0$ is satisfied for every task $\tau_i$ in **T** for the solution. To apply the Karush-Kuhn-Tucker optimality condition for concave programming, Equation (2) is reformulated as a concave programming

problem, as follows:

$$\begin{array}{ll} \text{maximize} & \sum_{\tau_i \in \mathbf{T}} \bar{E}_i(t_i) \\ \text{subject to} & \sum_{\tau_i \in \mathbf{T}} t_i = M \cdot D \text{ and} \\ & t_i \le D \quad \forall \tau_i \in \mathbf{T}, \end{array} \tag{3}$$

where $\bar{E}_i(t_i)$ is defined as $-E_i(t_i)$. The Karush-Kuhn-Tucker optimality condition for Equation (3) is to find a vector $(\lambda_1, \lambda_2, \ldots, \lambda_{|\mathbf{T}|})$, a vector $(t_1^*, t_2^*, \ldots, t_{|\mathbf{T}|}^*)$, and a constant $\lambda$ such that

$$\begin{array}{l} \bar{E}_i'(t_i^*) - \lambda_i = \lambda, \quad t_i^* \le D, \ \forall \tau_i \in \mathbf{T}, \text{ and} \\ (t_i^* - D)\lambda_i = 0, \quad \lambda_i \ge 0, \\ \sum_{\tau_i \in \mathbf{T}} t_i^* = MD, \end{array} \tag{4}$$

where $\bar{E}'_i()$ is the derivative of $E_i()$. We will show that such a vector $(t_1^*, t_2^*, \ldots, t_{|\mathbf{T}|}^*)$, could be determined by the Lagrange multiplier technique. Let n be an index, where $0 \le n < M$. If the execution time of $\tau_i$ is set as $D$ for $i = 1, 2, \ldots, n$, the concave programming in Equation~(4) could be rephrased as

$$\begin{array}{ll} \text{maximize} & \sum_{i=n+1}^{|\mathbf{T}|} \bar{E}_i(t_i) \\ \text{subject to} & \sum_{i=n+1}^{|\mathbf{T}|} t_i = (M-n) \cdot D, \end{array} \tag{5}$$

by further ignoring the inequalities $t_i \le D$ for $i = n+1, \ldots, |\mathbf{T}|$.

Equation (5) could be solved by applying the Lagrange multiplier technique. Since $\bar{E}'_i(t_i) = (\alpha\text{-}1)c_i^{\alpha}/t_i^{\alpha}$, given an index $n$, the conditions $\bar{E}'_i(t_i) = \bar{E}'_j(t_j)$ for all $n < i,j \le |\mathbf{T}|$ for the Lagrange multiplier technique lead to

$$\begin{array}{l} \frac{t_i^{\alpha}}{t_j^{\alpha}} = \frac{(\alpha-1)h_i c_i^{\alpha}}{(\alpha-1)h_j c_j^{\alpha}}, \quad \forall\, n < i,j \le |\mathbf{T}| \\ \sum_{j=n+1}^{|\mathbf{T}|} t_j = (M-n) \cdot D. \end{array}$$

Therefore, the optimal solution for Equation (5) is to assign $(t_{n+1}, t_{n+2}, \ldots, t_{|\mathbf{T}|})$ as $(t_{n+}{}^*{}_1, t_{n+2}{}^*, \ldots, t_{|\mathbf{T}|}{}^*)$,where

$$\sum_{j=n+1}^{|\mathbf{T}|} t_{n+1}^* \frac{c_j}{c_{n+1}} \left(\frac{h_j}{h_{n+1}}\right)^{1/\alpha} = (M-n)D \tag{6a}$$

$$t_j^* = t_{n+1}^* \frac{c_j}{c_{n+1}} \left(\frac{h_j}{h_{n+1}}\right)^{1/\alpha}, \forall n+1 < j \le |\mathbf{T}|, \tag{6b}$$

and the Lagrange multiplier $\lambda^*$ is $\bar{E}'_{n+1}(t_{n+1})$. As a result, the time complexity to derive the optimal solution of Equation (5) for an index $n$ is O(|**T**|-n).

It is clear that every $t_j^*$ in $(t_{n+}{}^*{}_1, t_{n+2}{}^*, \ldots, t_{|\mathbf{T}|}{}^*)$ derived from Equation (6) is greater than 0. Therefore, if each $t_j^*$ in $t_{n+}{}^*{}_1, t_{n+2}{}^*, \ldots, t_{|\mathbf{T}|}{}^*)$ is no greater than $D$ when $n=0$, then assigning $t_i$ as $t_i^*$ for $\tau_i$ in **T** is an assignment of execution times with the minimum energy consumption. Therefore, we only have to consider the other case. For the rest, let **T** be sorted by a *non-decreasing* order of $E_i'(D)$. The following lemma helps to construct an assignment of execution times of tasks in **T** with the minimum energy consumption.

*Lemma 3: Suppose that every $t_j^*$ in $(t_{n+}{}^*{}_1, t_{n+2}{}^*, \ldots, t_{|\mathbf{T}|}{}^*)$ derived from Equation (6) is less than D for an index $n^*$, and $\bar{E}'_{n^*}(D)$ is no less than $\bar{E}'_{n^*+1}(t_{n^*+1}{}^*)$ where $1 \le n^* < M$. The assignment of $t_i$ as D, for i = 1, 2, ...,n*, and $t_j$ as $t_j^*$, for $j = n^*+1, n^*+2, \ldots, |\mathbf{T}|$, would derive an assignment of task execution times with the minimum energy consumption.*

By Lemma 3, an assignment of execution times with the minimum energy consumption for the MEESM problem can be derived in $O(M|\mathbf{T}| + |\mathbf{T}| \log|\mathbf{T}|)$ by setting n from *0* to *M-1* sequentially. Moreover, the following lemma helps the reducing of the time complexity to $O(|\mathbf{T}| \log|\mathbf{T}|)$ by a binary search on the setting of $n$.

*Lemma 4: Suppose that every $t_j$* in ($t_{n+}$*$_1$, $t_{n+2}$*,..., $t_{|T|}$*) derived from Equation (6) is than D for an index n*, and $\bar{E}'_{n*}(D)$ is no less than $\bar{E}'_{n*+1}(t_{n*+1}*)$, where $1 \le n* < M$. If n^* < ň <M, the Lagrange multiplier for Equation (5) by setting n as ň is strictly greater than $\bar{E}'_{ň}(D)$. If ň≤ n*, the Lagrange multiplier for Equation (5) by setting n as ň is no greater than $\bar{E}'_{ň}(D)$..*

**Algorithm 1** : BIN

**Input:** $(\mathbf{T}, D, M)$;
**Output:** An optimal schedule for the MEESM problem;
1: **if** $|\mathbf{T}| \le M$ **then**
2: return the schedule by executing each task $\tau_i$ in $\mathbf{T}$ at the speed $\frac{c_i}{D}$ on the $i$-th processor from time 0 to $D$;
3: sort $\mathbf{T}$ in a non-decreasing order of $E'_i(D)$;
4: $left \leftarrow 0$ and $right \leftarrow M$;
5: **while** $left < right - 1$ **do**
6: $\hat{n} \leftarrow \lfloor (left + right)/2 \rfloor$;
7: apply Equation (6) by setting $n$ as $\hat{n}$;
8: **if** $\lambda > \hat{E}'_{\hat{n}}(D)$ **then**
9: $right \leftarrow \lfloor (left + right)/2 \rfloor$;
10: **else**
11: $left \leftarrow \lfloor (left + right)/2 \rfloor$;
12: $n^* \leftarrow left$;
13: $V \leftarrow (t_1, t_2, \ldots, t_{|\mathbf{T}|})$ by setting $t_i$ as $D$ for $i = 1, 2, \ldots, n^*$ and $t_j$ as $t'_j$ derived from Equation (6) for $j = n^* + 1, n^* + 2, \ldots, |\mathbf{T}|$;
14: return the schedule by applying Lemma 1 on $V$;

Our proposed algorithm denoted as Algorithm BIN (shown in Algorithm1) adopts the binary search strategy. After all, we have the following theorem.

***Theorem 1: Algorithm BIN can derive an optimal schedule for the MEESM problem in O(|T|log|T|).***

- ***Approximation Algorithm for the MEES Problem***

For the rest, let $t_i$* denote the *estimated execution time* of task $\tau_i$ in **T**, which is defined as the execution time of $\tau_I$ in the optimal solution derived from Algorithm BIN when task migration is allowed. The estimated execution times of tasks in **T** are then used to assign tasks onto these *M* processors. Let $e_i$* be the *estimated energy consumption* of task $\tau_i$ when the execution time of $\tau_i$ is the estimated execution time of $\tau_i$ i.e., $e_i^* = E_i(t_i*)$. Let $p_m$ denote the *load* on the *m*-th processor. The load of a processor is defined as the total amount of estimated execution time of the tasks assigned onto this processor. For notational brevity, let $T_m$ denote the set of the tasks assigned onto the *m*-th processor. Our proposed algorithm shown in Algorithm 2 (denoted as Algorithm LEET) adopts the Largest-Estimated-Execution -Time-First strategy. That is, tasks are considered in a non-increasing order of their estimated execution time.

For notational brevity, for the rest, let **T** be a sorted set in a *non-increasing* order of the estimated execution time. Algorithm LEET considers the tasks in the sorted order from $\tau_I$ to $\tau_{|T|}$. Once task $\tau_i$ is considered, $\tau_i$ is assigned onto the m-th processor whose current load is the smallest. After the assignment of the tasks onto the *M* processors is done, we have to assign the execution times of these tasks to meet the timing constraint. For every task $\tau_i$ assigned onto the *m*-th processor, the execution time of $\tau_i$ is set as $t_i*D/p_m$. After all, it is clear that the total execution time of the tasks assigned onto each processor is exactly equal to D. The time complexity of Algorithm LEET is $O(|\mathbf{T}| \log |\mathbf{T}|)$, which is dominated by applying Algorithm BIN, the sorting of the tasks, and the procedure to find the minimum $p_m$.

For notational brevity, let **T'** be a subset of **T**, where **T'** consists of those tasks whose estimated execution times are strictly less than *D*. That is, **T'**=$\{\tau_i \mid t_i* < D, \forall\ \tau_i$ in **T**$\}$. In the following lemma, we show that the ratio of the estimated energy consumption to the estimated execution time of task $\tau_i$ is equal to that of task $\tau_j$ if both $\tau_i$ and $\tau_j$ are elements of **T**'.

**Algorithm 2** : LEET

**Input:** $(\mathbf{T}, D, M)$;

**Output:** A feasible schedule for the MEES problem;

1: let $(t_1^*, t_2^*, \cdots, t_{|\mathbf{T}|}^*)$ be the assignment of execution times of $\mathbf{T}$ by applying Algorithm BIN$(\mathbf{T}, D, M)$;
2: sort all tasks in $\mathbf{T}$ in a non-increasing order of their estimated execution times;
3: set $p_1, p_2, \cdots, p_M$ as 0, and $T_1, T_2, \cdots, T_M$ as $\phi$;
4: **for** $i = 1$ to $|\mathbf{T}|$ **do**
5: find the smallest $p_m$; (break ties by choosing the smallest index $m$)
6: $T_m \leftarrow T_m \cup \{\tau_i\}$ and $p_m \leftarrow p_m + t_i^*$;
7: **for** $m = 1$ to $|M|$ **do**
8: **for each** task $\tau_i \in T_m$ **do**
9: $t_i' \leftarrow t_i^* \times \frac{D}{p_m}$;
10: return the schedule $S_{\text{LEET}}$ which executes all of the tasks $\tau_i$ in $T_m$ $(1 \le m \le M)$ at the speed $c_i/t_i'$ on the $m$-th processor one after one;

*Lemma 5: For any two tasks $\tau_i$ and $\tau_j$ in **T**', the ratio of the estimated energy consumption to the estimated execution time of $\tau_i$ is equal to that of $\tau$, i.e., $e_i^*/t_i^* = e_j^*/t_j^*$.*

*Lemma 6:* Suppose that the *m**-th and the *m'*-th processors are assigned with some tasks in **T**' such that $p_{m^*}$ is the maximum and $p_{m'}$ is the minimum after all of the tasks are assigned onto processors in Algorithm LEET, then $p_{m^*}$ is at most twice of $p_{m'}$.

*Theorem 2: The approximation ratio of Algorithm LEET is $((\alpha\text{-}1)^{\alpha-1}(2^\alpha\text{-}1)^\alpha)/((\alpha)^\alpha(2^\alpha\text{-}2)^{\alpha-1})$.*

Because $\alpha$ is at most 3, we have the following corollary.

Corollary 1: The approximation ratio of Algorithm LEET is 1.412.

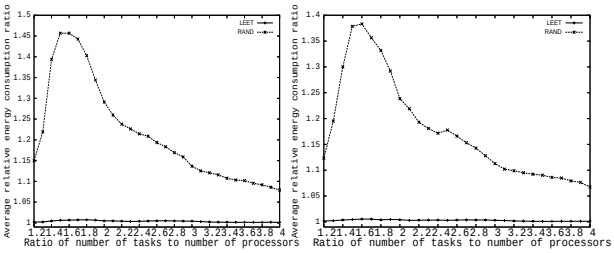
- ***Performance Evaluation***

We provide performance evaluation on the energy consumption of Algorithm LEET. Another algorithm, denoted as Algorithm RAND, which is very similar to Algorithm LEET, was simulated for comparison. The only difference between Algorithm RAND and Algorithm LEET is that tasks are not sorted before the assignment procedure in Algorithm RAND.

The common deadline $D$ of the tasks in a task set was set as 100 units of time in the simulations. For each task $\tau_i$ in **T**, $\tau_i$ was characterized by two different parameters: the number of execution CPU cycles $c_i$ and the coefficient $h_i$ of the power consumption function of $\tau_i$. $c_i$ was generated uniformly in the range $(0, D]$. $h_i$ was uniformly distributed in the range of 2 and 10. The exponent of the power consumption functions of the processor speed $s$ was set as 3, i.e., $P_i(s) = h_i s^3$. We simulated two cases for different numbers of processors with different numbers of tasks. For the first case, we evaluated the algorithms for the effects on the ratio of the number of tasks to the number of processors. For a given ratio $\eta$ of the number of tasks to the number of processors, the number of processors M was an integral random variable between 10 and 30, and the number of tasks was set as the floor of the multiplication of $\eta$ and $M$, i.e., $\lfloor \eta M \rfloor$. For the other case, the number of processors ranged from 2 to 20, and the task-set size ranged from 21 to 60. Experimental results were conducted with 512 independent experiments for each parameter configuration.

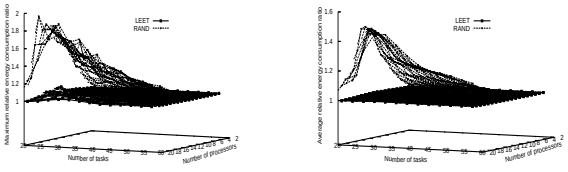
The *relative energy consumption ratio* was adopted as the performance metric in our experiments. The relative energy consumption ratio for an input instance was defined as the energy consumption of the schedule derived by the algorithm to that of an optimal schedule with the allowance of task migration. The energy consumption of an optimal schedule with the allowance of task migration can be derived in an efficient manner. Since the problem is NP-hard, the performance metric *relative energy consumption ratio* aimed

at the providing of an approximated index. When the results were for the average relative energy consumption ratio, their results were averaged. When they were for the maximum relative energy consumption ratio, the maximum value was returned.




(a) (b)

**Figure 1**: (a) and (b): maximum and average relative energy consumption ratios, respectively




(a) (b)

**Figure 2**: (a) and (b): maximum and average relative energy consumption ratios, respectively

For the evaluation of the effects on the ratio of the number of tasks to the number of processors, Figures 1(a) and 1(b) present the maximum and average relative energy consumption ratios for the simulated algorithms. The performance of Algorithm LEET was very close to that of the optimal solutions. The maximum and average relative energy consumption ratios for Algorithm LEET were less than 1.11 and 1.01, respectively. Furthermore, the maximum and average relative energy consumption ratios for Algorithm RAND were less than 1.82 and 1.46, respectively. When the ratio of the number of tasks to the number of processors was small, both of Algorithm LEET and Algorithm RAND might assign a task along with improper tasks on a processor. Such an assignment might result in a significant increase on the energy consumption of these tasks when the energy consumption for the other tasks were almost as the same as that in the optimal schedule. Such an observation explained why the maximum relative energy consumption ratios in Figure 1(a) decreased when the ratio of the number of tasks to the number of processors increased. However, when the ratio of the number of tasks to the number of processors was small, in most cases, most processors were assigned with only one task, and the assignment was almost as the same as that of an optimal schedule. Therefore, the average energy consumption ratio was relatively small when the ratio of the number of tasks to the number of processors was less than 1.5.

When the number of processors ranged from 2 to 20, and the task set size ranged from 21 to 60, Figure 2(a) (/Figure 2(b)) presents the maximum(/average) relative energy consumption ratios for the simulated algorithms. The performance of Algorithm LEET was again very close to the optimal solution. The maximum and average relative energy consumption ratios for Algorithm LEET were less than 1.084 and 1.01 respectively, where the maximum and average relative energy consumption ratios for Algorithm RAND were less than 1.941 and 1.485, respectively. The trends of the simulation results were similar to those in Figures 2(a) and 2(b). The results indicated that Algorithm LEET could derive effective schedules for the MEES problem.

## 四、即時 USB 子系統排程

- ***Problem Definition***

The objective of this research is to explore the QoS-based system design for USB 2.0 and 1.1

device management. We first propose an integrated real-time driver architecture over existing USB implementations for compatibility considerations. We then propose a real-time scheduling algorithm that mixes requests in the time frame for USB 1.1 (referred to as a m-frame) and those for USB 2.0, where one m-frame is equal to eight USB 2.0 time frames. A fixed-rate service approach and the corresponding schedulability test are presented to provide guaranteed QoS services to isochronous and interrupt requests for USB 2.0 and 1.1 devices. We then propose methodology in request insertions for the data structure dedicated for the USB host controllers. For best-effort requests, such as that for bulk transfers, we revise the existing methodology and provide a probabilistic performance guarantee. The capability of the proposed scheduling algorithms and methodology is demonstrated by the evaluation of the system overheads and performance over a system prototype, for which we have very encouraging results.

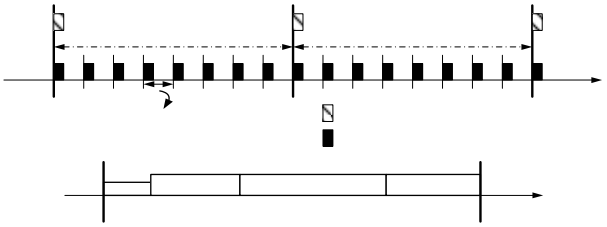


Figure 3 (a)The USB m-frame/u-frame Time Frame. (b)The Bandwidth Reservation of the USB m-frame/ u-frame Time Frame.

Different from USB 1.1, USB 2.0 could operates at the so called *high-speed*, at 480 Mbit/sec. Each periodic-frame-list entry is visited in every 1024 u-frame (i.e., 125us time interval).Within each u-frame, USB bandwidth is shared among requests, as shown in Figure3(b). Bandwidth marked as ”Periodic” is for isochronous and interrupt transfers, and bandwidth marked as ”NonPeriodic” is for control and bulk transfers. Periodic transfers could occupy up to 90% of the total bus bandwidth, and control transfers have higher priority than bulk transfers do. Bulk transfers are serviced in a best-effort fashion to utilize the remaining USB bandwidth. Since the time frame for USB 1.1 is in 1ms (referred to as a m-frame), one m-frame is equal to 8 u-frames, as shown in Figure3(a). Each request of a USB 1.1 device is partitioned into 8 sub-requests (the partitioning is referred to as *split transaction* for the rest of this paper) in 8 consecutive u-frames.

The USB specifications do not provide either an absolute or probability-based guarantee for the minimum bandwidth usage for different device requests. This observation underlies the motivation of this research. That is how to propose a proper resource management and scheduling mechanism to guarantee the QoS requirements of different devices and to comply with USB specifications.

- ***QoS Services for Periodic and Sporadic Requests***

We propose to have a layer of real-time request scheduling being built inside the USB driver core. The responsibility of the real-time scheduler is to do admission control and to insert QH’s and TD’s of admitted requests in proper locations of their corresponding data structures to meet their QoS requirements. In order to guarantee the QoS requirements of different devices and to better manage the USB bandwidth, two essential technical issues must be addressed: (1) an admission control policy and QoS guarantees for periodic and sporadic requests (2) the scheduling of QH’s and TD’s with QoS considerations. Under the satisfaction of the QoS requirements for devices, we shall also maximize the utilization of the USB

bandwidth.

- ***A Fixed-Rate Service Policy***

The fixed-rate service policy insert QH's at proper nodes of the polling tree structure. The idea is to insert all USB 2.0 QH's (and their TD's) and all USB 1.1 QH's (and their TD's) at the root and nodes corresponding to the polling rate $2^3$, respectively. An abstracted real-time scheduler layer is proposed in the driver hierarchy. The responsibility of the real-time scheduler is to communicate with the device driver layer and insert requests into the polling tree. The real-time scheduler should also be responsible to the admission control of new QoS requests. Since each USB 2.0 or 1.1 request is assigned a smaller period in the fixed-rate service policy, and the policy would introduce extra overheads because of more packet headers and the ceiling calculation for the number on the bytes to be transferred in a new period. Because 7500 bytes could be transferred within one u-frame, the schedulability test for a given set of n periodic requests B = {(b1, p1), (b2, p2), ..., (bn, pn)} could be derived as follows:

$$\sum_{USB2.0(b_i,p_i)\in B}\left(\left\lceil \frac{b_i}{p_i}\right\rceil + O\right) + \sum_{USB1.1(b_j,p_j)\in B}\left(\left\lceil 8*\frac{b_j}{p_j}\right\rceil /8 + O\right) <= 7500$$

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.

We then propose a workload re-insertion algorithm to reduce protocol overheads in request insertions for the data structure dedicated for the USB host controllers. 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.

- ***Probabilistic QoS Guarantee for Sporadic Transfers***

For sporadic requests, they are serviced in a round-robin fashion with a pre-determined order. For the probabilistic QoS guarantee of bulk transfers, we aim at the proposing of an evaluation method. Given the arrival distributions of bulk transfers, we could derive the average waiting time of a request and thus provide a probabilistic QoS guarantee for bulk transfers. The mean waiting time for a request in each queue list is

$$E(W)=\frac{\delta^2}{2r}+\frac{N[\lambda b^{(2)}+r(1+\lambda b)+\lambda\delta^2]}{2[1-N\lambda(r+b)]}$$

## 五、結果與討論

Research allocation is an important issue for real-time systems. In this research, we explore energy-efficient scheduling in a multiprocessor environment and efficient real-time scheduling for Universal Serial Bus (USB) 1.1/2.0 subsystems. We consider the scheduling of a set of frame-based real-time tasks with different power characteristics. An optimal algorithm is proposed when task migration is permitted with negligible overheads. For systems that do not permit task migration, we proposed a 1.412-approximation algorithm. Experimental results indicate that our proposed algorithm is very close to the optimal solution. We also explore the QoS-based system design for USB 2.0 and USB 1.1 device management. We address scheduling issues related to the mixture

of data transfers of USB 2.0 and USB 1.1 devices. We first propose an integrated real-time driver architecture that complies with the USB 1.1 and 2.0 specifications. A bandwidth reservation algorithm is proposed to partition available USB bandwidth among devices which need different attentions from the system. A fixed-rate service approach and the corresponding schedulability test are presented to provide guaranteed QoS services to periodic requests. We then propose a workload re-insertion algorithm to reduce protocol overheads in request insertions for the data structure dedicated for the USB host controllers. For best effort requests, such as that for bulk transfers, we propose a real-time scheduling algorithm with a probabilistic performance guarantee. The capability of the proposed scheduling algorithms and methodology is demonstrated by the evaluation of the system overheads and system performance over a system prototype, for which we have very encouraging results.

## 六、參考文獻


*[1] J. H. Anderson and S. K. Baruah. "Energy-efficient synthesis of periodic task systems upon identical multiprocessor platforms". In Proceedings of the 24th International Conference on Distributed Computing Systems, pages 428--435, 2004.*

*[2] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez. "Determining optimal processor speeds for periodic real-time tasks with different power characteristics" In Proceedings of the IEEE EuroMicro Conference on Real-Time Systems, page 225, 2001.*

*[3] N. Bansal, T. Kimbrel, and K. Pruhs. "Dynamic speed scaling to manage energy and temperature". In Proceedings of the 2004 Symposium on Foundations of Computer Science, 2004.*

*[4] T. Ishihara and H. Yasuura. "Voltage scheduling problems for dynamically variable voltage processors". In Proceedings of the International Symposium on Low Power Electronics and Design, pages 197--202,.*

*[5] W.-C. Kwon and T. Kim. "Optimal voltage allocation techniques for dynamically variable voltage processors". In Proceedings of the 40th Design Automation Conference, pages 125--130, 2003.*

*[6] F. Yao, A. Demers, and S. Shenker. "A scheduling model for reduced CPU energy". In Proceedings of the 36th Annual Symposium on Foundations of Computer Science, pages 374--382. IEEE, 1995*

*[7] J.-J. Chen, H.-R. Hsu, K.-H. Chuang, C.-L. Yang, A.-C. Pang, and T.-W. Kuo. "Multiprocessor energy-efficient scheduling with task migration considerations". In EuroMicro Conference on Real-Time Systems (ECRTS'04), pages 101--108, 2004.*

*[8] C.-Y. Yang, J.-J. Chen, and T.-W. Kuo. "An approximation algorithm for energy-efficient scheduling on a chip multiprocessor". In Proceedings of the 8th Conference of Design, Automation, and Test in Europe (DATE), pages 468--473, 2005.*

*[9] F. Gruian. "System-level design methods for low-energy architectures containing variable voltage processors". In Power-Aware Computing Systems, pages 1--12, 2000.*

*[10] F. Gruian and K. Kuchcinski. Lenes: "Task scheduling for low energy systems using variable supply voltage processors." In Proceedings of Asia South Pacific Design Automation Conference, pages 449--455, 2001.*

*[11] Y. Zhang, X. Hu, and D. Z. Chen. "Task scheduling and voltage selection for energy minimization." In Annual ACM IEEE Design Automation Conference, pages 183--188, 2002.*

*[12] R. Mishra, N. Rastogi, D. Zhu, D. Moss'e, and R. Melhem. "Energy aware scheduling for distributed real-time systems." In International Parallel and Distributed Processing Symposium, page 21, 2003.*

*[13] A. Chandrakasan, S. Sheng, and R. Broderson. Lower-power CMOS digital design. IEEE Journal of Solid-State Circuit, 27(4):473--484, 1992.*

[14] B. Adelberg, H. Molina, and B.Kao. Emulating soft real-time scheduling using traditional operating systems schedulers. In *IEEE 14th Real-Time Systems Symposium*, Dec 1994.

[15] S. Childs and D. Ingram. The Linux-SRT Integrated Multimedia Operating Systems: Bring QoS to the Desktop. In *IEEE 2001 Real-Time Technology and Applications Symposium*. IEEE Computer Society, 2001.

[16] S. Perez and J. Vila. Building distributed embedded systems with RTLinux-GPL. In *Proceedings of The Emerging Technologies and Factory Automation(ETFA)*, volume 1, pages
161 .168, Step 2003.

[17] H. Sato and T. Yakoh. A real-time communication mechanism for RTLinux. In *Conference of the IEEE on the 26th Annual Industrial Electronics Society, 2000. IECON 2000*, volume 4, 2000.

[18] H. Takagi. *Analysis of Polling System*. MIT Press Series, 1986.

[19] USB. *Open Host Controller Interface Speci_cation for USB*. Compaq, Microsoft, and National Semiconductor, 1996.

[20] T. Hanamoto, H. Ikeda, T. Tsuji, and Y. Tanaka. Sensorless speed control of synchronous reluctance motor using RTLinux. In *Proceedings of the Power Conversion Conference, 2002. PCC-Osaka 2002*, volume 2, pages 699 .703, 2002.

[21] V. Yodaiken. The RT-Linux approach to hard real-time. *Department of Computer Science Socorro NM 87801*, 2000.

[22] IEEE. *IEEE std. 1394-1995, IEEE Std 1394a-2000, IEEE Standard for a High Performance Serial Bus*. IEEE Computer society, 1996.