A Multiprocessor Real-Time Process Scheduling Method

Kuan-Yu Chen, Alan Liu and Chiung-Hui Leon Lee
Dept. of Electrical Engineering, National Chung Cheng University
ionic@aiol.ee.ccu.edu.tw, aliu@ee.ccu.edu.tw

Abstract

Multimedia systems like Video-on-Demand systems require a good scheduling method to improve their services because of their real-time requirements. If such systems consist of multiple processors, then the scheduling problem becomes much important. Scheduling is an important problem for both computer science and operation research. It is proved that the complexity for scheduling problems is NP-complete and sometimes NP-hard depending on the constraints of the problems, implying the difficulties for finding a good scheduling approach. In this paper, we propose a method for multiprocessor real-time scheduling algorithm applicable for both computer science and operation research. Our method is general enough to solve different scheduling problems such as wafer lot dispatching and scheduling for behaviors of a robot soccer player. There are scheduling problems exist in multimedia systems with real-time constraints, although the scheduling problems for multimedia systems have some unique characteristics differ from process scheduling, we believe the generality nature of our method facilitates the possibility of our scheduling method to be helpful for multimedia systems to solve scheduling problems after some minor modifications.

1. Introduction

Scheduling is an important problem for both computer science and operation research. Although computer scientists and operation researchers may focus on different issues such as timeliness for computer scientists and manufacturing cost for operation researchers, they share a common believe that a good scheduling approach can bring advantages such as improving efficiency and altering utilities of CPUs/machines.

It is proved that the complexity for scheduling problems is NP-complete and sometimes NP-hard depending on the constraints of the problems [1], implying the difficulties for finding a good scheduling approach. Putting different requirements and metrics for the needs of different users to scheduling problems makes them more difficult to solve. For instance, for the people sharing a single CPU may love a scheduling method to be as fair and efficient as possible in most of the time but they would not appreciate the scheduler to preempt their jobs to service others for fairness's sake when the jobs are time-critical.

Traditional criteria for a good scheduling method include fairness, efficiency, response time, turnaround time and throughput [2]. These criteria are insufficient when we are dealing with real-time multiprocessor scheduling problem. For a real-time scheduling problem, the timing constraints are of most importance, and real-time metrics such as minimizing the number of missed deadlines and minimizing the total lateness would be preferred rather than traditional metrics such as minimizing schedule length. For the multi-processor problem, we do not only focus on "when to do what" but also "who to do what". In other words, the scheduler must dispatch tasks to CPUs which is similar to the dispatching problem in a factory where jobs must be dispatched to machines. Thus we may take advantages from methods in operation research [3] to solve multi-processor problems and we may also provide a scheduling method suitable for both the computer science domain and the operation research domain.

In this paper, we propose a method for multiprocessor real-time scheduling algorithm applicable for both computer science and operation research. We categorize processes (jobs) into different types and make use of them in our scheduling algorithm to reflect the characteristics of different processes and the needs of users. Although it is believed that there is no silver bullet for scheduling problem, we try to find the common characteristics shared between different scheduling issues and thus facilitate our method to be general enough to solve different scheduling problems such as wafer lot dispatching and scheduling for behaviors of a robot soccer player.

There are scheduling problems existing in multimedia systems with real-time constraints [4]. To provide an acceptable performance for users of real-time multimedia systems, a nice scheduler is needed to provide guarantees to multimedia tasks which consists a set of jobs. Although the scheduling problems for multimedia systems have some unique characteristics that differ from process scheduling, we believe the generality nature of our method facilitates the possibility of our scheduling method to be helpful for multimedia systems to solve scheduling problems after some minor modifications. Assuming that the multimedia system has prior information about a multimedia task's execution including the arrival time and execution time, we are able to apply our scheduling method to multimedia systems. We can use the task segments introduced in our method to model the jobs in a multimedia task. Using the schedulability analysis provided in our scheduling method, a multimedia system is able to keep track of the loading conditions on different CPUs and dispatch multimedia tasks to different CPUs according to the result of the schedulability analysis.

A performance bottleneck is found in the wafer lot dispatching problem due to the unbalanced loading conditions among different semiconductor fabrication tools. Experimental result shows that our scheduling method can improve the performance of tools by balancing the loads on tools. A multimedia system with multiprocessor may face the same performance bottleneck in the wafer lot dispatching problem, we believe that our method provides a solution to eliminate the performance bottleneck.

We review some scheduling methods including branch-and-bound algorithm (B&B), earliest deadline first (EDF) and robust earliest deadline (RED) and discuss advantages and shortcomings of these scheduling methods in Section 2. After the review, we introduce our scheduling method in Section 3 including our scheduling model and method for schedulability according to the model. An analysis of the complexity of our scheduling method and a comparison with other methods are given. In Section 4, we demonstrate our scheduling method with two very different applications, wafer lot dispatching and scheduling for behaviors of a robot soccer player, to show the generality of our method. Finally, a conclusion and future work is given in Section 5.

2. Related Work

In this section, we focus on the introduction of two scheduling methods: earliest deadline first (EDF) and branch-and-bound algorithm (B&B) [4]. We will also discuss the advantages and shortcomings of these two scheduling methods and provide some guidelines to construct our scheduling method. Before the introduction of the two scheduling methods, we will first discuss some important scheduling issues to facilitate the discovery of some improvements of the existing scheduling methods.

Schedulability. A scheduling method should have the ability for schedulability analysis/test to provide some degree of guarantee for tasks with timing constraints. Off-line scheduling which analyze the performance of the scheduler can use schedulability test to examine whether the computation capability is enough or not for the tasks to meet their timing constraints [5]. For on-line scheduling, fault tolerance mechanisms rely on the guarantee provided by the scheduler to determine their behavior. In [2], there states a simple schedulability test for ***m*** periodic tasks with a formula $\sum_{i=1}^{m} \frac{\boldsymbol{Ci}}{\boldsymbol{Pi}} \leq 1$, where task ***i*** requires computation time ***Ci*** and having the period of ***Pi***. The formula above deals only with periodic tasks and is inapplicable for aperiodic tasks which exist in most systems. In the scheduling method we propose, we provide the ability for schedulability test for both periodic and aperiodic tasks.

Overloading. From the observation of the results in [6], we can find that overloading is the major performance bottleneck of the scheduler whether using earliest deadline first algorithm or rate monotonic algorithm. Unfortunately, overloading may occur even after an off-line scheduling/analyzing due to the dynamic characteristic of the scheduling environment. This again states the importance of the on-line schedulability test mentioned before. In this paper, we provide a method to calculate the workload of each CPU on-line to detect overloading and use the calculation results to lead the dispatching of tasks and thus avoid overloading.

Task properties. There are some task properties such as value, task type and content switch time should be integrated into the consideration of scheduling. Most of us may agree with the viewpoint that the more we know about something, the better we can deal with it. For example, we can use the value of tasks to determine which task to reject when overloading occurs. The task type is another important property but being rarely discussed relatively. Different users may run different types of tasks, such as periodic and aperiodic, and different types of tasks should be scheduled in different manners. In our scheduling method, we take these task properties concerning timing constraints into scheduling considerations along with the context switch time which is important but often ignored.

2.1 Earliest Deadline First (EDF)

The scheduling policy for an EDF scheduler is quite simple. It always chooses the job (process) with the earliest deadline. There are many real-time systems using EDF scheduling algorithm not only because the simplicity nature of the algorithm but also because EDF has been proved to be optimal under many conditions [1].

The major shortcoming for EDF is the performance during overloading. Due to lacking the ability for schedulability analysis/test, EDF cannot detect the occurrence of overloading and thus cannot prevent or handle overloading as well.

RED. Buttazzo and Stankovic [7][8] proposed robust earliest deadline (RED) to improve EDF algorithm. RED calculates residual time and workload of jobs as schedulability analysis. By using the schedulability analysis, RED algorithm is able to provide guarantees to jobs and executes its task rejection policy. The performance of RED under overloading is shown to be much better then EDF. Although RED provides a great idea for adding a schedulability analysis to EDF, RED has the limitation on the assumption of all jobs which are aperiodic and implicitly deals with different types of jobs. In our scheduling method, we follow the spirit of RED to provide a schedulability analysis and, furthermore, deal with both aperiodic and periodic jobs and provide a method to represent jobs of different types.

2.2 Branch-and-Bound Algorithm (B&B)

The branch-and-bound algorithm (B&B), introduced from the field of artificial intelligence, searches the problem space for optimal or near-optimal schedules [4]. A B&B scheduler takes following steps to search for solution:

Step 1. Selection: The scheduler selects a vertex from a candidate set of vertices as a new search point.

Step 2. Generation: The scheduler generates the children vertices of the selected vertices as new candidates to search.

Step 3. Evaluation and pruning: Each child vertex generated in Step 2 is evaluated by the scheduler. Once the scheduler finds that a child vertex cannot reach a feasible solution, the child vertex is eliminated from the search space. After evaluation, the qualified vertices are put into a candidate set for the next selection step.

A B&B scheduler iterates the above steps until a feasible solution is found or no more candidate vertex to explore. The B&B algorithm has the advantages of flexibility. The scheduling strategy can be changed for optimization of different metrics by applying different parameters such as vertex selection rule, vertex branching rule and vertex elimination rule [9]. The major shortcoming of the B&B algorithm comes from the exponential complexity of the space search. To apply the B&B algorithm for real-time scheduling where efficiency of the scheduler is a very important issue, the designer of a B&B scheduler must take extremely care for designing the pruning mechanism to improve the efficiency of the B&B scheduler. Our scheduling method can be viewed as a limited version of the B&B algorithm which searches only a very limited space to ensure the efficiency of the scheduler.

3. Method for Multiprocessor Real-Time Scheduling

In this section, we introduce our scheduling method which is applicable for both uni-processor and multiprocessor real-time scheduling. The scheduling model including assumptions of our scheduling method, CPU timetable model and task model is provided in Section 3.1. In Section 3.2, we describe the schedulability analysis in our scheduling method and provide some guidelines for applying our method under different situations. Since efficiency of the scheduling algorithm is an important issue for real-time scheduling, we give a complexity analysis of our scheduling method in Section 3.3. A comparison with other scheduling methods is given in Section 3.4.

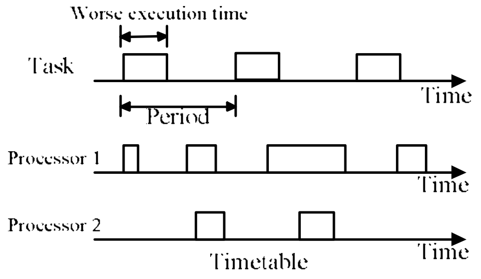

Figure 1. Timetable for processors and a task to be scheduled

3.1 Scheduling Model

To facilitate our method to be general enough for both the computer science and the operation research domains, we establish our scheduling model with the concern for both domains. We assume the scheduling environment contains M processors and the value of M may vary with time. In other words, new processor may be dynamically added into the environment and some existing processor may also be removed from the environment. The scenario of adding or removing processors rarely occurs in operation systems on computers but it happens occasionally in factories to add or remove machines (tools) for maintenance or repair. For a uniform notation, we use the term "tasks" for processes in the computer science domain and jobs in operation research domain. Tasks can be preemptive or non-preemptive and we assume there are no precedence constraints among tasks.

In our scheduling method, we establish a timetable for each processor to facilitate the calculation of schedulability. The timetable consists of a set $L = \{l_t : 0 \le t \le T\}$ of workload information and T is the length of the time table. The workload information l_t consists of the load of the processor at the specific time t and the tasks contributing the load. Figure 1 shows the timetable of two processors and a task to be scheduled. Using the timetable model, the scheduling problem is transformed into a problem of finding a suitable timetable and a suitable position on the timetable to insert the task which is to be scheduled.

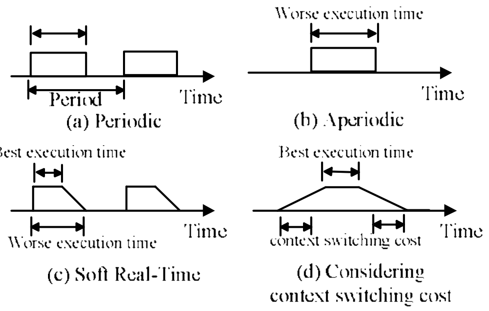

Figure 2. Different task models for different types of tasks

The task model in our scheduling method is similar to the timetables for the processors. Each task is modeled as a set of workload on the time axis. To represent different types of tasks, we provide different task models as shown in Figure 2. We can model both periodic and aperiodic tasks using our task model. As illustrated in Figure 2(c), We use a deceased load value $L = \{l_i \ge l_j : i \le j; i, j \in T\}$ in the time interval $T = \{\mathbf{b} \le t \le w\}$, where b is the best execution time and w is the worst execution time, to represent a soft real-time task and allow two tasks to share one processor during underloading time interval $T = \{t : l_t^i + l_t^j \le 1\}$ where l_t^j denotes the load value contributed by task j at time t. We assume the maximum workload equals to 1 for a processor in any time instance. With the strategy to represent a load less then 1, we are able to represent the time interval which has a possibility for the processor to be either occupied or idle. This strategy may also be helpful for representing the context switching cost by adding a small amount of load beside the workload of the task as shown in Figure 2(d). The advantage of this strategy is to prevent the scheduler from being too pessimistic and, in the same time, prevent it from being too optimistic. A pessimistic scheduler always considers the worst execution time of tasks and is suitable for applications where timing constraints are critical and violations to the time constraints are absolutely not allowed. For some soft real-time applications, the utilization of the processor may be a more important issue so the pessimistic scheduler may not be preferred. From the other aspect, an optimistic scheduler always considers the best execution time of tasks and ignores the context switching time may not be allowed for hard real-time applications. Our scheduling method provides different task models for the user to choose according to his/her needs.

3.2 Schedulability Analysis

We use the timetables of processors and task models introduced in Section 3.1 to facilitate our schedulability analysis. We transform the scheduling problem into the problem of fitting the task into the timetables of processors. Our schedulability analysis tries to insert the task into a timetable and gathers information from insertion for future use. Before further introduction to our schedulability analysis, we introduce the concept of task segments first. To ease the calculation of schedulability analysis, we separate tasks, especially periodic tasks, into task segments. A task segment s consists of a continuous set $L = \{l_t > 0 : \forall t, t' < t < t''\}$ of workload greater then zero. After the segmentation, a task becomes a set $S = \{s_i : 1 \le i \le m\}$ of task segments and each task segment may have its own deadline. A periodic task is therefore transformed into a set of aperiodic task segments. Task segments also help our schedulability analysis to focus only on the time interval where the load value of the task is not equals to zero.

Our algorithm for schedulability analysis for one task segment is described in Figure 3. In the algorithm, we check the timetable for space to insert the task segment. A task segment is able to insert into a timetable at time t if and only if the condition $l_{t+\Delta}^{TB} + l_{\Delta}^{s} \le 1$ holds for all $\Delta = \{t : t \in T^s\}$, where $l_{t+\Delta}^{TB}$ denotes the load value of the timetable at the time $t + \Delta$,

```
Algorithm Schedulability Analysis (Task Segment S, Timetable TB, Shift TS)
    Pre: S is a task segment to be scheduled to a processor with timetable TB.
         L(t) is the load value on TB at time t.
         l is the length of S and T is the length of TB.
         D is the deadline of S and D is less then T.
    Return: The residual time, R, of T, the collision value, C, of T in TB and the time
            shifted t
    Begin:
      t:= start time of S + shift from previous segment TS;
      C:= 0;
      while( t < T )
          if( T can be inserted into TB at time t )
            for( t':=0; t' < l; t':=t'+1)
                C:= C + L(t+t');
            R = D – t – l;
            Return R, C, (t+TS);
        else
              C := C + L(t);
              t:= t+1;
      End while;
      R = D – T;
      Return R, C, (t+TS);
```

Figure 3. The algorithm for schedulability analysis

l^{s}_{Δ} denotes the load value of the task segment s at the time Δ. If the task segment cannot be inserted at time t, we shift the task segment to the right and try to insert the task segment at time $t+1$. After the schedulability analysis, we obtain the information of residual time. Positive residual time guarantees the task segment to meet its timing constraints while negative residual time indicates lateness of the task segment and we may apply a task rejection policy according to the residual time. Besides residual time, we obtain a collision value $C = \sum_{\forall t, t \in T'} l^{TB}_{t}$ where $T' = [t \sim t'']$, t is the time value of the start point on the timetable TB we try to insert the task segment s and t'' equals to the sum of t' and l where t' is the time value of the point we successfully insert s into TB and l is the length of s. The collision value can be used to compare the loading conditions of processors when the residual time from different processors make no differences, such as all residual time are positive or negative.

Scheduling. We can apply schedulability analysis to both uni-processor and multiprocessor scheduling. For uni-processor scheduling, the schedulability analysis can be applied off-line as an analysis tool to discover overload conditions and online to provide guarantees and facilitate the task rejection mechanism. For multiprocessor scheduling, we use the architecture illustrated in Figure 4. We assume that the environment consists of a set $P = \{p_n : 1 \le n \le M\}$ of processors. The scheduler picks up a task with earliest deadline in the task queue for scheduling and uses schedulability analysis to obtain the residual time and collision value of the task on each processor. The scheduler then dispatches the task to the processor with a positive residual time and lowest collision value. A simple task rejection policy can be defined as rejecting the task if the condition $\forall p \in P : R_p < 0$ holds where R_p denotes the residual time on the processor p.

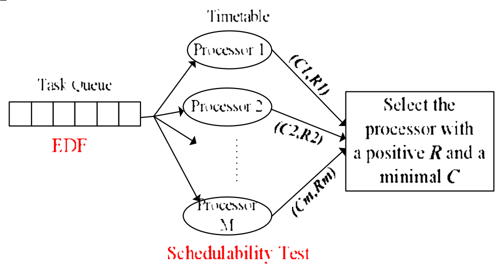

Figure 4. Multiprocessor scheduling

3.3 Complexity Analysis

We assume all task segments having unit length since the length of the task segment is short compared with the length of the time table, T. From Figure 3 we can discover that the complexity of the schedulability analysis for one task segment is $O(T)$. There are two stages in our multiprocessor scheduling method. The first stage is an EDF algorithm to choose a task with the earliest deadline from the task queue and the complexity of EDF algorithm is **O(*n*)** where ***n*** is the number of tasks in the task queue. The second stage is to perform ***M*** schedulability analyses, which have complexity **O(*T*)**, to ***M*** processors and thus the complexity for the second stage is $\mathbf{O}(M \times T)$. (The complexity of the second stage can be reduced to **O(*T*)** if the schedulability analyses can proceed in parallel.) The overall complexity for our multiprocessor scheduling to schedule one task segment is **O(*n*)** if $n > M \times T$ and is $\mathbf{O}(M \times T)$ otherwise. To produce a complete schedule for n tasks, the complexity is $\mathbf{O}(n^2)$ if $n > M \times T$ and is $\mathbf{O}(n \times M \times T)$ otherwise. For applications which require a high time resolution having a small ***n*** and a very large ***T***, it may be disappointing because of the complexity being dominated by ***T***. However, there are some applications, such as wafer lot dispatching, having a very large ***n*** and a relatively small ***T***, the performance of the scheduling algorithm may be satisfactory. We are working on a continuous representation for the timetable to eliminate the effects of ***T***.

3.4 Comparison with Other Scheduling Methods

Our scheduling method incorporates EDF with a schedulability analysis. We also follow the spirit of RED to provide a schedulability analysis mechanism to facilitate the ability to provide guarantees and execute task rejection policy. Furthermore, we extend our method to multiprocessor scheduling and provide different task models to represent different types of tasks.

Compared with the B&B algorithm, our scheduling method has a clear bound for complexity while a B&B scheduler may faces exponential complexity for space search if the pruning mechanism is not well designed. Our scheduling method can be viewed as a limited version of the B&B algorithm which searches only a very limited space to ensure the efficiency of the scheduler. We use the EDF algorithm as a vertex selection rule and insertion of the chosen task into timetables of processors as a vertex branching rule. The schedulability analysis acts as the lower-bound cost function and we keep only one vertex which has a positive residual time and the lowest collision value among all children vertices.

Compared with Chen's competitive neural network approach [10], the complexity of our scheduling method is lower and we do not have the problems with initial states of the neural network and the rate of convergence.

4. Applications

In this section, we introduce two applications: wafer lot dispatching and scheduling for behaviors of a robot soccer player, the former is under evaluation at TSMC and the latter application is the result of an NSC project. We believe the generality nature shown in our method facilitates the possibility of our scheduling method to be helpful for other domains with timing constraints, such as multimedia systems, to solve scheduling problems.

4.1 Wafer Lot Dispatching

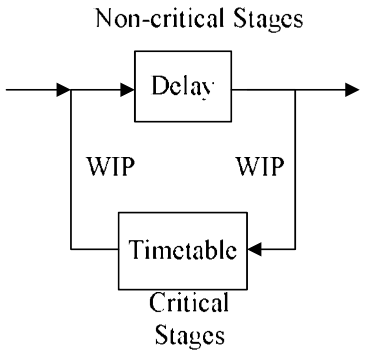

Figure 5: Model of the stage flow

Wafer lot dispatching is a complex yet important problem in semiconductor fabrication. Our goal is aimed to provide dispatching decisions to the supervisor of a specific zone in TSMC. The specific zone consists of a set $P = \{p_n : 1 \le n \le M\}$ of tools to process critical stages. Each lot may have hundreds of stages and tens of critical stages among them in its workflow. As illustrated in Figure 5, lots enter the specific zone to process a critical stage and leave the zone to process non-critical stages. When a lot enter the zone for the first time, the supervisor must make a decision to dispatch the lot to a specific tool. After the lot had been processed on a specific tool once for a critical stage, it has to process all its critical stages on the same tool since processing the critical stages of a lot on different tools will cause an unacceptable yield drop. Due to the characteristics of critical stages, a poor dispatching decision may cause some tools in the zone to be idle while others are overloading and queued with a lot of lots. We use task segments to model critical stages of a lot and use schedulability analysis to discover overloading conditions and thus to determine the best tool for dispatching.

Table 1. The simulation result for wafer lot dispatching

Tool \ Day	0	1	2	3	4	5	6	7	8	9	AVG
T1	0.71	0.71	0.54	0.88	0.88	0.46	0.67	0.79	0.79	0.54	0.697
T2	0	0.42	0.88	0.83	0.25	0.58	0.88	0.71	0.88	0.83	0.626
T3	0.71	0.71	0.79	0.38	0.88	0.71	0.79	0.83	0.92	0.88	0.76
T4	0	0.21	0.63	0.71	0.83	0.83	0.83	0.83	0.79	0.17	0.583
T5	0	0.67	0.79	0.71	0.21	0.63	0.71	0.79	0.79	0.67	0.597
T6	0.88	0.38	0.54	0.63	0.83	0.88	0.79	0.75	0.83	0.71	0.722

Table 1 shows the simulation result for our lot dispatching method. Values in the table denote the average load/day of a tool. From the table we can see that before applying our scheduling method for lot dispatching, the load on tools are extremely unbalanced. At day 0, T2, T4 and T5 are idle for the entire day while T1, T3 and T6 are having heavy loads. After applying our scheduling method, the loads for tools are balanced and the utilization of tools may also be promoted since our scheduling method smoothes the flow of lots.

4.2 Scheduling for Behaviors of a Robot Soccer Player

In our previous work [11], we construct a software system for the robot soccer player. We adopt a behavior-based approach using motor schemas for low-level control. There is no problem for our robot soccer player to run its behaviors on a personal computer (PC) but when we try to port our robot soccer player from a software system to a hardware robot, we must consider the scheduling problem for behaviors since the onboard CPU is much slower than the PC's. We can use our scheduling method to schedule the behaviors onboard. Currently, the physical robot is still under construction but our analysis shows a promising result.

5. Conclusion

In this paper, we propose a method for multiprocessor real-time scheduling. In our method, we incorporate EDF and a schedulability analysis. To meet the needs of different users, we provide different task models for different types of tasks such as periodic, aperiodic, hard real-time and soft real-time. The complexity for our scheduling method to produce a complete schedule for $\boldsymbol{n}$ tasks is $\mathbf{O}(\boldsymbol{n}^2)$ if $\boldsymbol{n} > \boldsymbol{M}\times\boldsymbol{T}$ and is $\mathbf{O}(\boldsymbol{n}\times\boldsymbol{M}\times\boldsymbol{T})$ otherwise. The complexity is acceptable for some applications where $\boldsymbol{n}$ is large and $\boldsymbol{T}$ is relative small. We are working on a continuous representation to improve the efficiency of our scheduling algorithm.

Our scheduling method is general enough to be applied to different domains such as wafer lot dispatching and scheduling of behaviors of a robot soccer player. The simulated result shows our scheduling method balances the loads among tools for semiconductor fabrication. We believe the generality nature shown in our method facilitates the possibility of being useful for other domains with timing constraints, such as multimedia systems, to solve scheduling problems.

Acknowledgement

This research was supported in part by the National Science Council under grant NSC 91-2213-E-194-004.

References

[1] J. A. Stankovic, M. Spuri, M. Natale, and G.C. Buttazzo, "Implications of classical scheduling results for real-time systems", *IEEE Computer*, vol.28, no.6, pp.16-25, June 1995.

[2] A. S. Tanenbaum and A. S. Woodhull, "Operaing Systems: Design and Implementation", *Chapter 2, Prentice Hall*, 1997.

[3] F. Wang, Z. Wang, S. Yang and D. Chen, "Solving Nonstandard Job-Shop Scheduling Problem by Loading Balance of Machines Scheduling Algorithm", *in Proc. of the 4th world congress on Intelligent Control and Automation*, June 2002, pp. 2338-2341,

[4] S. Wang, K. J. Lin and S. Peng, "BWE: A Resource Sharing Protocol for Multimedia Systems with Bandwidth Reservation", *in Proc. of the 4th international symposium on Multimedia Software Engineering (MSE)*, December 2002, pp. 158-165

[5] K.Ramamritham and J.A. Stankovic, "Scheduling Algorithms and Operating Systems Support for Real-Time Systems", *in Proc. of the IEEE*, vol. 82, No. 1 , Jan 1994, pp. 55-67.

[6] G. Wainer, "Implementing Real-Time Scheduling in a Time-Sharing Operating System", *ACM Operating Systems Review*, July 1995

[7] G. Buttazzo and J. Stankovic, "Adding Robustness in Dynamic Preemptive Scheduling", *in Responsive Computer Systems: Steps Toward Fault-Tolerant Real-Time Systems*, Edited by D. S. Fussell and M. Malek, Kluwer Academic Publishers, Boston, 1995.

[8] G. Buttazzo and Stankovic, "RED: Robust Earliest Deadline Scheduling", *Technical Report*, Dept. of Computer Science, University of Massachusetts, TR-93-25, 1993

[9] J. Jonsson, K. G. Shin, "A Parametrized Branch-and-Bound Strategy for Scheduling Precedence-Constrained Tasks on a Multiprocessor System", *in Proceedings of the International Conference on Parallel Processing (ICPP),* 1997 , pp. 158-165.

[10] R-M Chen, Y-M Huang, and C-Y Lin, "Competitive Neural Network to Solve Real-Time Scheduling ", *in Proceedings of International Computer Symposium (CD-ROM)*, 2002

[11] K. Chen and A. Liu, "A Design Method for Incorporating Multidisciplinary Requirements for Developing a Robot Soccer Player", *in Proc. of the 4th international symposium on Multimedia Software Engineering (MSE)*, December 2002, pp. 25-32