Research on Resource Access Control Protocol Based on Layered Scheduling Algorithm

Xibo Wang[1,2] , Fenmei Wang[2] , Ge Yu[1]
1.College of Information science and Engineering, Northeastern University, Shenyang, China
2.School of Information Science and Engineering, Shenyang University of Technology, Shenyang , China

wangxb@sut.edu.cn wangfenmei205@126.com yuge@mail.neu.edu.cn

***Abstract*—Complicated real-time application needs operating system providing layered scheduling mechanism to realize two layers scheduling for disjoint tasks subset. For satisfying the running predictability of real-time task under circumstance of layered scheduling, operating system should have corresponding resource access control protocol to avoid infinite priority inversion and deadlock phenomenon while multi-task sharing resource. Aiming at the problems, a new resource access control protocol based on layered scheduling algorithm is proposed. The protocol expands and redefines the priority inheritance protocol and the priority ceiling protocol, distributes resource control lock for each sub-modules, cross-sub-module resource contention is prohibited. Inside a sub-module, priority of task and priority of resource are set respectively, and priority ceiling protocol is realized by dual priority control approach. This protocol controls layered multi-task accessing critical resource effectively and reasonably, meet the demand of schedulable constraint condition for real-time task in layered sub-module situation, improves real-time performance of layered scheduling algorithm. Theoretical analysis and experimental results indicate that layered resource access control protocol has the ability of inhibiting priority inversion and avoiding deadlock phenomenon effectively, enhanced resource access control ability while supporting real-time application.**

***Keywords*—real-time system, critical resource, dual priority, layered protocol**

I. INTRODUCTION

Real-time system not only needs to satisfy function demand of application, but also needs to satisfy real-time requirement put-forwarded by the application. However, real-time requirement of various real-time tasks are different. Besides, some complex correlation and synchronous relations between real-time tasks may exist, such as tasks executing order restriction, mutual exclusion access of sharing resource etc. This has brought great difficulty for assurance real-time performance and real-time task running predictability in real-time system. The optimal scheme to solve the problems is designing suitable scheduling algorithm and corresponding resource access control protocol in real-time operating system to ensure real-time tasks behavioral predictability. The real-time system is not paying attention to average behavior of scheduling algorithm and resource access control protocol, it requires each real-time task to meet the real-time demand in worst cases, i.e., the real-time system pay attention to individual behaves, more accurately, individual's worst situation behaves [1, 2].

While tasks in a real-time system can not satisfy their deadline by RM scheduling algorithm and resource access control protocol, choosing another processor with quicker processing speed is an inferior approach for improve system real-time performance. The optimal method is improving scheduling algorithm and resource access control protocol which has little system expenses and higher CPU handling capacity utilization ratio [3, 4]. Otherwise, some complicated real-time application needs operating system providing layered scheduling mechanism for function disjoint tasks subset. Aiming at the above problem, layered scheduling algorithm based on priority preemptive real kernel μC/OS-II is developed. The fixed round-robin time is set to realize sub-module scheduling, tasks in the sub-module also schedule in preemptive manner. Layered priority inheritance protocol (LPIP) and layered priority ceiling protocol (LPCP) are proposed on the basis of analyzing typical resource access protocol in layered scheduling environment. The proposed protocol can not only guarantee task deadline in sub-module, but also can avoid deadlock phenomenon and inhibit the priority reversion phenomenon. Theoretic analysis and experimental verification show that the layered scheduling algorithm is feasible and the layered resource access control protocol is correct.

II. DEFINITION OF LAYER ALGORITHM AND PROTOCOL

Definition 1: In case of not consider release jitter, tasks set is $T=\{\tau 1,\tau 2, \ldots,\tau n\}$, time attribute of task τi is described by a five-tuple (Mi, Φi, Pi, Di, Ei). Among them, Mi is module ID; Φi show phase; Pi is task cycle, as to aperiodic task, Pi is minimum interval of the aperiodic task; Di shows the deadline of the task; Ei shows the biggest request running time of the task.

The definition of task is expended from four-tuple [5] to five-tuple so as to realize layered management. Module ID is added to indicate which module the task belongs to and provide a sign point to sub-module index.

Definition 2: As to task τi, response time of task instance is the time interval from reaching to finishing of the task instance, the maximum value Ri is called the biggest response time of task τi, if $Ri \leq Di$, claim task τi can be scheduled.

Definition 3: If $Ri \leq Di$, claims task τi can be scheduled. If at a certain time point, all tasks in a certain real-time system can be scheduled, claims tasks set of the system can be scheduled at this moment.

Definition 4: If system can ensure each sub-module task finished within deadline, claim tasks set of the system can be layered scheduled at this moment.

Definition 5: Task section from initial critical resource locked to eventual matching unblocked is called critical section of the task. In addition, release of resources carry out according to the order of last in first out. So, overlapped critical section is strictly nested.

Definition 6: When execution time in the critical section and nested way are all related with discussion, use the square bracket [*R*, *n*, *e*] to express critical section. *R* is resource accessed by task in critical section; *n* is number of the resource; *e* is biggest execution time of the task in the critical section. If number of resource *R* is only one, the value *n* is omitted, uses simple symbol [*R*, *e*] for replacement.

Definition 7: Ω is priority ceiling. Its initial value is the lowest task priority of the system. If task apply resource, priority of resource become present priority ceiling $\Pi(t)$.

III. DESIGNATION OF LAYED MODEL

A. Schedulable analysis for layered scheduling algorithm

Assume: There are *n* independent and preemptive tasks in system, and task relative deadline is equal to its cycle. If the total utilization ratio *U* of the system satisfies (1), then, the task set and can be scheduled on the single processor according to RM scheduling algorithm [6].

$$U_{RM}(n) = n\left(2^{1/n} - 1\right) \leq \ln 2 \tag{1}$$

Deadline of each task must meet the conditions shown in (2).

$$w_i(t) \leq e_i + \sum_{T_k \in T_H(i)} \left\lceil \frac{t}{p_k} \right\rceil e_k \tag{2}$$

(Among them $T_H(i)$ Show the task set in which task's priority is higher than task *Ti*)

While scheduling adopts layered model, sub-module is scheduled according to fixed time slice round-robin scheme, task in the sub-module scheduled according to RM algorithm, the effective supply time *supply (t)* of the sub-module task must meet (3).

$$\text{supply(t)} \leq \left\lfloor \frac{t}{RL} \right\rfloor \tau + \max\left(0, \tau - RL + t - \left\lfloor \frac{t}{RL} \right\rfloor RL\right) \tag{3}$$

Among them RL is round-robin cycle of sub-module, $\tau = \frac{RL}{n}$, *n* is number of sub-module. Deadline threshold values of task *i* in sub-module *j* must satisfy (4).

$$w_{i,j}(t) = e_i + bj + \sum_{T_k \in T_H(i)} \left\lceil \frac{t}{p_k} \right\rceil e_k \tag{4}$$

Where, *bj* is switching time of sub-module. Only expiration is small or equal to threshold value, system could realize correctly scheduling function.

B. Designation of layered priority inheritance protocol

In order to solve the problem of priority inversion in layered model, layer priority inheriting protocol is extended and used in layered scheduling algorithm. Assuming the number of each kind of resource is only one. The protocol rule is defined as follows:

1) Scheduling rules:

- Resources allocated for one sub-module can't be shared with other sub-module, i.e., the resources that a sub-module occupied can't be applied by tasks in different sub-module before releasing, so it is necessary to try hard to dispose suitable resources for each sub-module.
- On the processor, the ready task in sub-module can be scheduled in priority drive preemptive way according to its present priority. The present priority of each task *J* at the time of releasing is equal to its distributional priority. The tasks will always keep this priority unchanging except in case of rule *3)*.

2) Distributional rules:

When task *J* of a sub-module asks a resource *R* at moment *t*:

- If *R* is idle, distributes it to task *J*, resource-lock is added to ensure other sub-module no longer has the ability to apply resource *R* before *J* releases it, but can be applied by other task in the same sub-module.
- If *R* is not idle, check whether it lock by other sub-module or not, if it is locked by other sub-module, task *J* turns into ready queue to wait next time slices, then applies once again. If *R* is used in this module, *J* insert into block queue.

3) Priority inheritance rules:

When higher priority task *J* apply for resources occupied by lower priority task J_l, *j* block and J_l will inherit the present priority $\pi(t)$ of *J*, the task J_l will carry out according to its inherited priority $\pi(t)$ until it releases the resource *R*; After J_l withdraw critical section, priority of task J_l return back to its original priority.

C. Designation of layed priority ceilinge protocol

Because of layered priority inheritance protocol can not solve deadlock problem [7], layered priority ceiling-protocol is proposed for avoiding deadlock phenomenon while multi-task

sharing resource at circumstance of layered scheduling. Layered priority ceiling-protocol is defined as following:

1) Scheduling rules:

- For each task J_{ij} in sub-module, priority $\pi(t)$ is equal to its distribution priority at release moment t. The tasks will keep this priority unchanging except in case of rule *3)*.
- Each ready task J in sub-module is scheduled in the way of priority driven preemption according to its present priority $\pi(t)$.

2) Distributional rules:

If task J of a sub-module asks a resource R in moment t, one of the three cases will occur.

a) R is being occupied by another task in the same sub-modul: the request of J fails, and J turns into blocks state.

b) R is being occupied by another task of different sub-module: the request of J fails, and J enters the ready queue and waits for next time slice and apply once again.

c) R is idle:

- If priority $\pi(t)$ of task J is higher than present priority ceiling $\Pi(t)$, then R is distributed to task J.
- If priority $\pi(t)$ of task J is not higher than present priority ceiling $\Pi(t)$, could distribute R to task J only when J occupy the resource that its priority equal to priority ceiling $\Pi(t)$, otherwise, the request of J will be refused, J will turn into block state.

3) Priority inheritance rules:

When higher priority task J apply for resources occupied by lower priority task j_l, j block and J_l will inherit the present priority $\pi(t)$ of J, the task J_l will carried out according to its inherited priority $\pi(t)$ until it releases the resource R; After J_l withdraw critical section, priority of task J_l return back to its original priority.

IV. ANALYSIS AND IMPLEMENTATION OF LAYERED PROTOCOL MODEL

A. Implementation of layered priority inheritance protocol

Non-preemptive scheduling is the prerequisite of priority inversion. Because critical resources has the property of non-preemption, resource competition among multi-task may lead to priority inversion, even execution of tasks can be preempted, task of higher priority may be blocked by task of lower priority [8]. The priority inversion situation is demonstrated as following.

Observe an example firstly. There are three sub-modules in one system, sub-module schedule based on round-robin time slice. The time of one round RL is 3 time units, length of time slice is 1 unit. Each sub-module can get one time slice in a round. Suppose there are three tasks J_{11}, J_{12}, J_{13} in sub-module 1, feasible section of them are respectively (6, 14], (3, 17], (0, 18]. Every vertical line on the time line marks releasing time and deadline for each task. These tasks are scheduled on the processor based on strategy of fixed priority algorithm RM (The period is equal to deadline). So, the priority of J_{11} is the highest, the priority of $J13$ is the minimum. Among them, task $J11$ and task $J13$ ask for same resource R. Critical section of $J11$ and task $J13$ are respectively $J11$ [R、1] and $J12$ [R、1.5]. The task scheduling diagram shown in Fig. 1, the black rectangle shows the time of task in its critical section. From the scheduling diagram, it can be seen that the priority of $J11$ decrease to the level of $J13$, priority inversion occurs. If $J12$ need long execution time, $J11$ will delay very long time. Most serious, if there are more medium priority task exists, $J11$ can be delay execution unrestricted long, the system scheduling may be collapse.

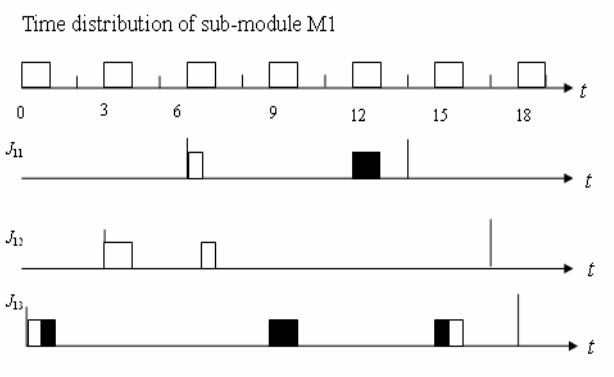

Figure 1. Priority inversion in layered scheduling algorithm

The tasks of this sub-module are scheduled again by employ the layered priority inheritance protocol. Scheduling diagram shown in Fig. 2. It can be seen that while task J_{11} ask for critical resource occupies by task J_{13}, task J_{13} inherit the priority of J_{11}, task J_{13} will run at high priority level, use the critical resource as soon as possible, and then, withdraw its critical section. In this way, task J_{12} has no chance to preempt task J_{13}, infinite priority inversion is inhibited. After J_{13} withdraw its critical section, its priority return back to its original priority. While resource distribute to task in certain sub-module, it lock in the sub-module, tasks in other sub-module can not apply the locked resource before the sub-module release it. Resource cross-sub-module competition is not allowed. Resource control power could only transfer to other sub-module after current sub-module release it.

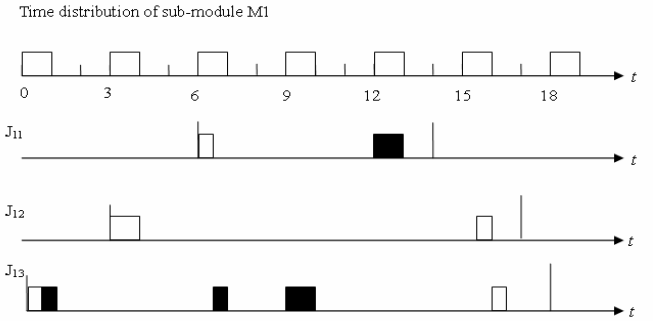

Figure 2. Priority inversion being inhibited by layered priority inheritance protocol

It can be seen from the example that layered resource access control protocol can dispel priority infinite inversion behavior caused by critical resource competition. But the

protocol can not avoid deadlock while two kinds of resources are applied concurrently by multi-tasks. Deadlock is a serious calamity in real-time application, suitable measures must be consider for solve deadlock problem. On basis of analysis typical protocol, layered priority ceiling protocol is proposed to resolve deadlock problem in layered scheduling case.

B. Implementation of layered priority ceiling protocol

Take tasks in sub-module 2 as an example to describe the implementation approach of layered priority ceiling protocol, tasks parameters described in Tab.1.

TABLE I. PARAMETERS OF TASKS IN SUB-MUDLE M2

Task	*Ri*	*Ei*	*π*	*Critical secton*
J_{21}	12	1	1	[Shaded;0.5]
J_{22}	8	1	2	[Black;0.5]
J_{23}	5	1	3	
J_{24}	2	2	4	[Shaded;1[Black;0.5]]
J_{25}	0	3	5	[Black;2]

Scheduling method based on layered priority ceiling protocol is described as follows:

- At time point 1, task J_{25} gets time slice and run at distributed priority 5. Especially, priority ceiling of the sub-module is Ω at 1.5 time point. When J_{25} asks for Black resource, resource is allocated according to first case of part of *c)* in layered priority ceiling protocol rule *2)*. After distribut Black resource, the priority ceiling of the system is prompted to 2, i.e., priority ceiling of Black resource.
- At time point 4, task J_{24} has already been released, it preempt task J_{25} and execute. At time point 4.5, task J_{24} asks for Shaded resource. Shaded is idle at this moment, but because of the priority ceiling of the system $\Pi(4.5)$ is equal to 2, it is higher than the priority of task J_{24}, the request of task J_{24} is refused according to second part of *c)* in layered priority ceiling protocol rule *2)*. J_{24} is blocked, thus J_{25} inherits the priority of J_{24} and carries out with priority 4.
- At time point 7 , task J_{23} preempte J_{25}, at moment 8, task J_{23} finished, in the moment 10, J_{22} preempte J_{25}, after execute 0.5 time slice, J_{22} asks for Black resource and is blocked directly by task J_{25}, task J_{25} inherits the priority of J_{22} and execute with priority of J_{22} until task J_{21} is ready and preempte it. During this period, the priority ceiling of the system keeps in 2.
- When task J_{21} applies for Shaded resource, its priority higher than ceiling of system. So, the request of task J_{21} is accepted according to first part of *c)* in layered priority ceiling protocol rule *2)*. J_{21} can enter its critical section and finish at moment 14, at moment 16, J_{25} resumes execution.
- Task J_{25} releases Black resource at moment 6.5, the priority resumes 5 and priority ceiling of system dropped to Ω, task J_{22} is not in blocked state, assign Black resource to it according to first part of *c)* in layered priority ceiling protocol rule *2)*, accept request of J_{22} and finishes at time point 17.
- At time point 19, running task J_{24} applys Shaded resource, because priority of J_{24} is higher than priority ceiling of the system at this moment, therefore the Shaded resource can assigned to it and it begin to access critical resource. The priority ceiling of the system rises to 1. J_{24} Release the resource Shaded after 0.5 time slice, then applys for Black resources. J_{24} is the only task of applying for resource at present, resources is distributed to it.
- At moment 22, task J_{24} and J_{25} finish execution.

According to the priority ceiling protocol algorithm describes above, scheduling diagram is shown in Fig. 3, where, black rectangle shows the critical section of task that occupy Black resource, shade rectangle shows critical section of task that occupy Shaded resource.

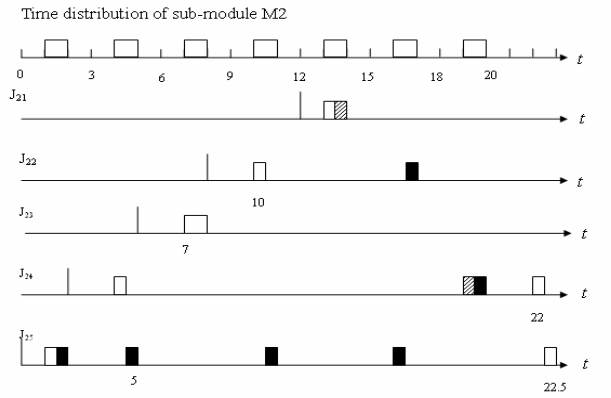

Figure 3. Scheduling diagram by using priority-ceiling protocol in module 2

As describe previously, priority ceilings of resource Black and Shaded are 2 and 1 respectively. In module M2, the priority ceiling $\Pi(t)$ changes with tasks access critical resource. Priority ceiling diagram of the above example is shown in Fig. 4. It can be seen from Fig. 3 and Fig. 4 that task J_{24} is blocked at 4.5 time point based on rule of second part of *c)* in layered priority ceiling protocol rule *2)*. As a result, task J_{24} with lower priority is sacrificed, so as to make task J_{21}, J_{22}, J_{23} with higher priority finish earlier. This is just the effect the protocol provided.

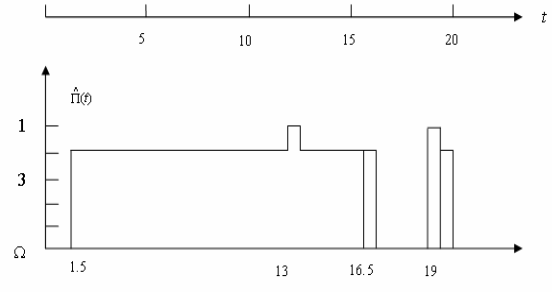

Figure 4. Priority-ceiling diagram of layered protocol

Priority inheritance protocol is greedy, but priority ceiling protocol is not, that is the most fundamental difference between them. Compared with inheritance protocol, the ceiling protocol can prevent occurrence of deadlock, but inheritance protocol is unable to prevent deadlock. According to layered inheritance protocol, scheduling diagram of tasks in Table 1 is shown in Fig. 5. It can be seen from Fig. 5 that task J_{25} ask for Shaded resource after Shaded resource has already distributed to task J_{24} at a certain time point (for instance, at 4.5 moment), then these two tasks will be both blocked, deadlock phenomenal occurs. Such a situation will not appear by using layered priority ceiling protocol.

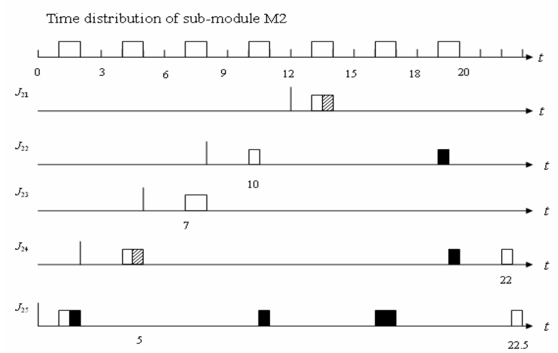

Figure 5. Scheduling diagram of priority inheritance protocol

V. EXPERIMENT EMULATION AND RESULT ANALYSIS

Layered scheduling algorithm needs real-time kernel supporting same priority task scheduling. But, µC/OS-II real-time kernel which being selected for verifying the proposed protocol does not support same priority task scheduling. Therefore, µC/OS-II kernel structure must modify and round-robin time slice scheduling schemes should add in µC/OS-II scheduler to support same priority scheduling before experiment. Above problems has already solved by author's previous research work [9, 10].

In experiment, mutual exclusion signal is employed for imitating application and release of critical resources. Execution diagrams of layered sub-modules scheduling are shown in Fig. 6-8. Fig. 6 shows effect diagram of priority inversion in module M2; Fig. 7 is effect diagram adopting layered inheritance protocol to control priority inversion; Fig. 8 is effect diagram adopting layered ceiling protocol to control priority inversion and avoid deadlock.

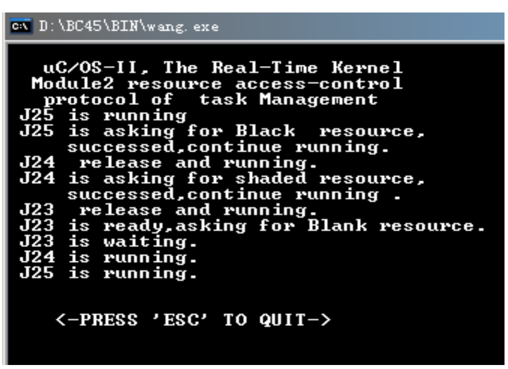

Figure 6. Task23 priority inversion in M2

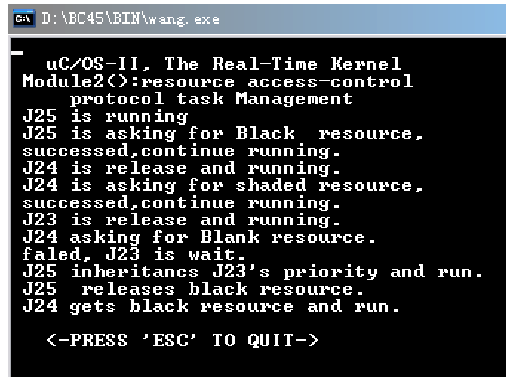

Figure 7. Priority inversion inhibited by priority inheritance protocol

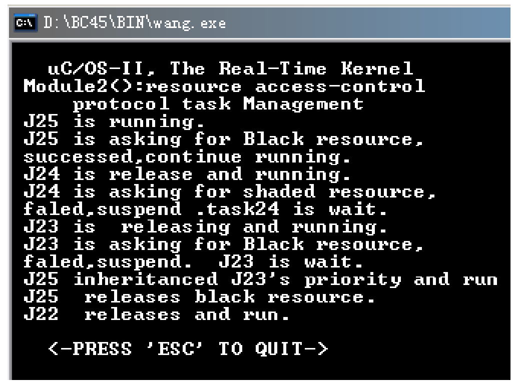

Figure 8. Priority inversion inhibited and deadlock avoided by priority-ceiling protocol

It can be found out from Fig.6 that task J_{23} with higher priority is blocked by task J_{24} with lower priority and appear the phenomenon of priority inversion. Like Fig. 7, task J_{23} is blocked while applying for resources, task J_{25} inherits the priority of task J_{23}, priority inversion is inhibited after using the priority ceiling protocol. Result illustrated in Fig 8 shows that not only priority inversion can be controlled but also prevented deadlock.

VI. CONCLUSION

Through reconstructing µC/OS-II kernel and improving its scheduler, its function of scheduling has been expanded and strengthened, the aim of layered scheduling that supporting complex application can realize. Under circumstance of layered scheduling, resource access control protocol was studied and extended, layered priority inheritance and ceiling protocol rules are proposed for inhibiting priority inversion and avoiding deadlock. The proposed protocol is verified on improved µC/OS-II real-time platform. Theoretical analysis and experiment result show that the proposed protocol is feasible and effective. Next step of research will be multiply processors resource access control protocol.

REFERENCES

[1] G. Xu, F. M. Yang. Review of embed operation system. Application research of Computer):4-9.

[2] J. Sauermann, M. Thelen. Real-time operating system department of computer Engineering. Malardalen University,2003,7-10.

[3] G. J. Huang, Z. G. Hu. A Threshold Based Scheduling Algorithm for Embedded Real-time System. Computer Engineering. 2006,32(3): 68-70.

[4] E. Bini, G. Buttazzo. Rate monotonic analysis: The hyperbolic bound. IEEE Trans. on Computers, 2003,52(7): 933-942.

[5] J. Y-T Leung, J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. Performance Evaluation, 2:237-250, 1982.

[6] Mok, A. K-L., D. Chen. A multiframe model for real-time. Proceedings of IEEE Real-time Systems Symposium,December, 1996.

[7] L. Sha, R. Rajkumar, and C. H. Chang. A real-time locking protocol. IEEE Transactionson Computers, 40(7):793-800, Jul 1991.

[8] J. X. Liu, Y. J. Wang, Y. Wang. Real-Time System Design Based on Logic OR Constrained Optimization . Journal of Software, Vol.17, No.7, July 2006, pp.1641-1649.

[9] C. S. Xie, J. D. Ma, H. Huang. Research on Task Scheduling Policies Based on μC/OS-II. Computer Engineering & Science, 2004, 26(8): 70-73.

[10] X. B. Wang, B. H. Zhou, G. Yu, Q. Li. Homology Priority Task Scheduling in μC/OS-Real-time Kernel.Wuhan University Journal of Natural Sciences,2007,12(1):167-171..

[11] Y. Qiao, H. A. Wang, G. Z. Dai. Developing a New Dynamic Scheduling Algorithm for Real-Time Multiprocessor Systems. Journal of Software, 2002,13(1):51-58.