2010 3rd International Conference on Computer and Electrical Engineering (ICCEE 2010)
IPCSIT vol. 53 (2012) © (2012) IACSIT Press, Singapore
DOI: 10.7763/IPCSIT.2012.V53.No.2.80

On-line Scheduling Algorithm with Precedence Constraint in Embeded Real-time System

ZHAO Ming[+] and SONG Xiao-Yu

School of Information & Control Engineering, Shenyang Jianzhu University

Shenyang, China

Abstract—Function design of embedded real-time system introduced the problem of real-time task scheduling with precedence constraint which needs that scheduling sequence produced by algorithm satisfying not only real-time constraint of every task, but also precedence constraint among tasks. Based on parallel topological sort, this paper presents a new on-line scheduling algorithm OSA-RPC which can set priority of task considering real-time parameter and precedence constraint.

Keywords-real-time;scheduling;precedence;constraint; topological sort

1. Introduction

The problem of real-time scheduling with precedence constraint lied in specific function design which requires in practical applications[1], such as embedded control[2], mixed real-time system of data stream and event driven control[3], and real-time test system[4] etc. These applications are usually designed into tasks which scheduled on computing resource and interacting with each other. Because of these tasks are designed to implement the whole computation, scheduling sequence of these tasks needs to satisfy precedence constraint to ensure the correctness of the general function. This kind of precedence constraint can be described with $\tau_i \prec \tau_j$ or $<\tau_i, \tau_j>$ which represents that τ_i must be scheduled before τ_j. All precedence constraints among tasks can also be described with DAG (Directed Acyclic Graph). Obviously, the scheduling of real-time system with precedence constraint among tasks must meet following requirements: scheduling of every task must meet its real-time parameters; scheduling sequence of all tasks must meet precedence constraints among tasks.

Lawler [5] presented on-line scheduling algorithm satisfying precedence constraint LDF (Last Deadline First) which can deal with non-preemptive concurrent task set on 1969. Lenstra [6] demonstrated that valid scheduling to tasks with precedence constraint and ready at different time can not be found in polynomial time. Blazewicz [7] implemented real-time scheduling with precedence constraint by the way to limit the release of successor task must be after the completion of predecessor task. Basing on Blazewicz's idea, CHETTO [8] presented a classic real-time scheduling algorithm with dynamic priority setting EDF* (Modified EDF) which ensures that the start execution of every successor task can not precede the completion of its predecessor task and its release time through the modifying of their real-time parameter.

But to realize the scheduling described above, EDF* either learns real-time parameters of every task and all precedence constraints among tasks before scheduling and then modifies parameter according to principles, or needs to record precedence constraints among tasks at any moment in order to modify the parameters of task released recently. The former needs correct real-time parameters which may not be obtained before the

[+] Corresponding author.
E-mail address: xxzhaoming@sjzu.edu.cn

release of task and usually dither. The latter needs to record precedence constraints among all tasks which needs more space (it is necessary to preserver at least n2 level storage locations for n tasks). But storage resource is limited in embedded system. So EDF* is limited in practical applications.

Real-time scheduling algorithm with precedence constraint is discussed in this paper. The description of the problem is presented in section 2. Design principal and analysis of algorithm is presented in section 3 and simulation experiment is done in section 4. In section 5, the conclusion is given.

2. Problem Description

Definition 1 Embedded real-time system is a task set consists of N real-time tasks among which there are precedence constraints. This task set can be described with $\Pi = (\Gamma, \prec_\Gamma)$ where Γ is real–time task set in system and $\prec_\Gamma$ is precedence set on Γ. Task set is described with $\Gamma = \{\tau_i \mid i = 1...N\}$ and a task is described with $\tau_i(R_i, C_i, D_i)$ inside which R_i is the release time of τ_i, C_i is the max execution time of τ_i and D_i is the deadline of τ_i。 Precedence set is described with $\prec_\Gamma = \{\tau_i \prec \tau_j \mid i = 1...N, j = 1...N, 且 i \neq j\}$.

An embedded real-time system example is demonstrated by $\Pi_1 = (\Gamma_1, \prec_{\Gamma_1})$, where

$\Gamma_1 = \{\tau_1(0,1,2), \tau_2(0,1,5), \tau_3(0,1,4), \tau_4(1,1,3), \tau_5(1,1,7), \tau_6(2,1,6)\}$

$\prec_{\Gamma_1} = \{\tau_1 \prec \tau_2, \tau_1 \prec \tau_3, \tau_2 \prec \tau_4, \tau_2 \prec \tau_5, \tau_3 \prec \tau_6\}$.

Scheduling algorithm of this paper is to do non-preemptive real-time scheduling on the embedded system described as definition 1 and the detailed goal are demonstrated as follows:

1. To ensure the correctness of system function, when scheduling two tasks with precedence constraint between them, the precedence constraint is mainly considered and the real-time request of task is discarded. That is, if $\tau_i \prec \tau_j$, τ_i must be scheduled before τ_j.
2. To improve real-time performance of task running, when scheduling tasks without precedence constraint between them, takes non-preemptive EDF as scheduling principle. That is, if $D_i < D_j$ and there does not exist $\tau_i \prec \tau_j$, τ_i must be scheduled before τ_j.

3. On-line scheduling algorithm on real-time task set with precedence constraint (OSA-RPC)

3.1. Scheduling principle

As a kind of basic research on graph theory, topological sort mainly discusses how to make a sequence of the vertices in graph. So, topological sort is often used as a basic theory to study non-preemptive scheduling problem. Parallel topological sort [9], considering not only serizlizability between the vertices which connecting with vertex but also parallelism between vertices without vertex connecting, is presented recently. This idea is used in the algorithm presented in this paper. When scheduling real-time tasks with precedence constraints, the parallelism between tasks without precedence constraint is also considered. Furthermore, in order to improve real-time performance, the task with urgent real-time request is selected to schedule first among the parallel tasks without precedence constraint.

According to the analysis above, in design, the algorithm assigns parallel number to every task to show its execution level. If two tasks are serial, algorithm assigns them different number to demonstrate that they are in different execution level. If two tasks are parallel, algorithm assigns them same number to demonstrate that they are in same execution level. Algorithm always chooses the task with smallest parallel number to run first and if there are several tasks, the task with latest deadline is selected.

3.2. Algorithm design

The algorithm assumes that any task releases after its predecessors which can be implemented by tasks synchronization. Every task records its predecessors, and once it is released, if all of its predecessors are released before, after assigning parallel number to it, scheduler pushes it in ready queue and modifies predecessor's status of its successors in waiting queue, else pushes it in waiting queue.

The algorithm deals with precedence constraints with its predecessors when the task is released, and reflects the precedence constraint between predecessor and successor as different parallel number, according to the principals as follows:

Assuming τ_k is the task released recently is to deal with by the algorithm, following assignments can be made:

$\text{pall}(\tau_k) = \text{pall}(\tau_i) + 1$

$where \tau_i \prec \tau_k, pall(\tau_i) \geq pall(\tau_j), \forall j \neq i and \tau_j \prec \tau_k$ $pall(\tau_k)$ is the parallel number of τ_k。

At the same time, according to real-time parameters, adjusts the parallel number of τ_m (parallel task of τ_k in τ_i 's level) to reflect that relation levels of tasks are changed due to the release of τ_k. That is,

if $D_m > D_k$, $pall(\tau_m) = pall(\tau_m) + 1$.

The processing procedure of example system discussed in section 2 is demonstrated as follows:

When τ_1 is released, because there is no precedence constraint, the algorithm assigns pall (τ_1) =1;

When τ_2 is released, due to the precedence constraint < τ_1 , τ_2 >, the algorithm assigns pall (τ_2) = pall (τ_1) +1=2.

When τ_3 is released, due to the precedence constraint < τ_1 , τ_3 >, the algorithm assigns pall (τ_3) = pall (τ_1) +1=2.

When τ_4 is released, due to the precedence constraint < τ_2 , τ_4 >, the algorithm assigns pall (τ_4) = pall (τ_2) +1=3. At the same time, in order to improve real-time performance, because τ_4 and τ_3 are parallel tasks, the algorithm adjusts pall(τ_3)=3 basing on the fact that D3>D4.

When τ_5 is released, due to precedence constraint < τ_2 , τ_5 >, the algorithm assigns pall(τ_5)= pall(τ_2)+1=3.

When τ_6 is released, due to precedence constraint < τ_3 , τ_6 >, the algorithm assigns pall(τ_6)= pall(τ_3)+1=4. At the same time, due to the fact that D6<D5 and τ_5 is the parallel task of τ_6, the algorithm adjusts that pall(τ_5)=4. Because D6>D4, the parallel number of τ_4 don't need to adjust.

The result assigned by the algorithm is given in TABLE 1.

TABLE I. PARALLEL NUMBER ASSIGNMENT OF EXAMPLE

	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6
Parallel number	**1**	**2**	**3**	**3**	**4**	**4**
Deadline	**2**	**5**	**4**	**3**	**7**	**6**

According to table 2 and the scheduling principal of the algorithm, scheduling sequence ($\tau_1, \tau_2, \tau_4, \tau_3, \tau_6, \tau_5$) satisfying precedence constraints and real-time constraints can be reached.

3.3. Example analysis and contrast

The scheduling results of example system in section 2 by non-preemptive EDF, non-preemptive Balzewicz and OSA-RPC are given in Figure 1.

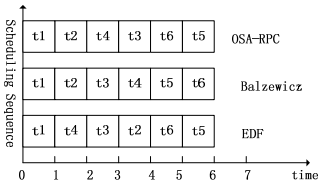

Fig.1. Scheduling results under every algorithm

Conclusions as follows can be gotten from figure 2: when scheduling by non-preemptive EDF, real-time constraint of every task is satisfied, but precedence constraints among tasks are not satisfied because the relations among tasks are not considered during the schedule of EDF; when scheduling by non-Bazewicz, precedence constraints among tasks are satisfied, but some task missed deadline; only after the scheduling of OSA-RPC, real-time constraint of every task and precedence constraints among tasks are both satisfied.

4. Simulation Experiment

Simulation experiment is done to compare OSA-RPC with non-preemptive EDF and non-preemptive Balzewicz using Matlab scheduling toolbox as experiment platform.

The data used in experiment is set as follows: twenty tasks is produced every time; the deadline of every task is the random value between 20 and 60; the execution time of every task is the random value between 1 and 3; the release time of every task is the result of its number minus 1; precedence constraints among tasks are produced according to following principle.

The first task is produced without predecessor.

Tasks of next level are produced according to precedence constraint ratio (the ration between upper level and lower level) and precedence constraints of every task with the tasks in upper level are produced randomly.

According to the principle above, five task sets are produced using precedence constraint ratios $R_p = 1.5,2,3,4,20$ and the precedence constraint levels number are 6,5,4,3,2. Details of task parameters and precedence constraints are given in TABLE II.

TABLE II. INPUT DATA FOR EXPERIMENT

Rp	*Deadline*	*Execution time*	*Precedence constraints among tasks*	*level number*
1.5	44,41,43,47 37,54,42,20 24,49,42,47 23,42,44,23 43,44,47,45	2,3,1,2,2, 2,1,2,3,3, 3,2,3,3,3, 1,2,3,2,3	<1,2><1,3><2,4,><3,5,> <3,6,><6,7><4,8> <6,9><5,10><4,11> <8,12> <7,13> <8,14> <8,15><9,16> <7,17> <7,18><11,19><17,20>	6
2	36,33,31,20 34,28,27,38 35,41,31,40 21,31,36,36 50,36,42,44	1,3,2,1,2, 3,1,3,1,2, 1,2,1,3,2, 3,1,3,1,2	<1,2><1,3><2,4> <3,5> <3,6><3,7><4,8><4,9> <4,10> <6,11><6,12> <7,13><6,14><5,15> <10,16><9,17> <15,18> <12,19> <12,20>	5
3	58,59,54,44 40,46,47,55 48,39,40,41 41,46,37,36 29,38,46,35	1,3,1,3,2, 2,1,1,3,3, 2,2,3,1,2, 1,3,2,1,2	<1,2><1,3><1,4> <2,5> <2,6><3,7> <2,8> <4,9> <4,10><2,11> <2,12><4,13> <6,14> <11,15> 9,16><13,17> <12,18> <10,19> <8,20>	4
4	44,38,36,44 46,32,32,34 33,36,40,34 31,25,34,20 28,23,33,22	1,1,1,1,3, 2,2,2,2,2, 3,3,3,2,2, 3,1,1,2,1	<1,2><1,3><1,4,> <1,5,> <3,6,><3,7><5,8> <4,9><3,10><2,11> <4,12> <4,13> <4,14> <2,15><4,16> <3,17> <2,18> <2,19><5,20>	3
20	21,24,51,33 25,28,27,31 20,31,30,49 51,21,29,46 47,20,35,47	1,3,1,2,2, 1,2,3,3,3, 2,2,2,3,1, 1,2,1,2,1	<1,2><1,3><1,4><1,5> <1,6><1,7><1,8> <1,9><1,10><1,11> <1,12><1,13><1,14> <1,15><1,16><1,17> <1,18><1,19><1,20>	2

After inputting the task sets demonstrated in table 2 into experiment platform to schedule, average response time of every task set is calculated which given in TABLE III and FIGURE II.

TABLE III. THE CONTRAST BETWEEN AVERAGE RESPONSE TIME OF TASK

Precedence level	*Blazewiec*	*OSA-RPC*
6	22.37	19.83
5	17.45	15.70
4	12.95	12.95
3	8.4	8.4
2	11.35	11.35

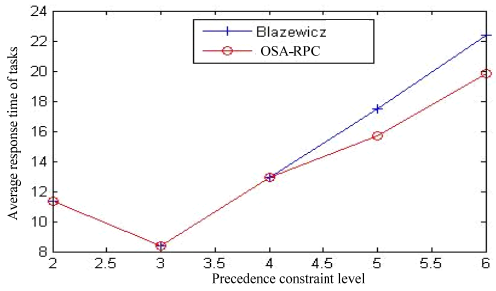

Fig.2. Contrast between response time of task

It is shown that the average response time is bigger because the start time of task is decided by the finish time of its predecessor when precedence constraint level is bigger. Along with the decreasing of precedence level, predecessor number of task becomes smaller, so the finish time of processor does less effect on start time of its successor and the average response time becomes smaller. At the same time, when scheduling same task set, such as precedence level 6 and 5, smaller average response time can be reached using OSA-RPC algorithm than using non-preemptive Blazewiec, because of OSA-RPC improves the release time of successor task and so it may be scheduled earlier. When precedence constraint level is very small, the average response times are same because the start time of task is decided by its release time.

5. Conclusion

Basing on the parallel topological sort, an on-line real-time scheduling algorithm OSA-RPC with precedence constraint is presented in this paper. OSA-RPC can deal with tasks released asynchronously and set the scheduling priority of task considering serizlizability and parallelism among tasks in order to ensure the correctness and real-time performance of scheduling. At the end of the paper, experiment result shows that OSA-RPC is effective and it improves average response time of tasks comparing to same kind algorithm.

6. References

[1] L Mangeruca, M Baleani, A Ferrari, A L Sangiovanni-Vincentelli.Uniprocessor scheduling under precedence constraints[C]. Proceedings of the Twelfth IEEE Real-Time and Embedded Technology and Applications Symposium,2006

[2] LEONARDO MANGERUCA, MASSIMO BALEANI, and ALBERTO FERRARI. Uniprocessor Scheduling Under Precedence Constraints for Embedded Systems Design[J].ACM Transactions on Embedded Computing Systems, 2007,7(01),45-65

[3] Yu Xiao, Wang Jiali. Schedulability test algorithm for periodic tasks with partial order constraint [J]. Journal of Electronic Measurement and Instrument,2009,23(04),65-69

[4] H CHETTO, M SILLY AND T BOUCHENTOUF. Dynamic Scheduling of Real-Time Tasks under Precedence Constraints[J]. The Journal of Real-Time Systems, 1990,12(2), 181-194

[5] E L Lawler. Optimal sequencing of a single machine subject to precedence constraints[J]. Managements Science, 1973,19(05),544-546

[6] J K Lenstra, A H G. Rinnooy Kan. Complexity of Scheduling under Precedence Constraints[J]. OPERATIONS RESEARCH,1978,26(1),22-35

[7] J Blazewicz. Scheduling dependent tasks with different arrival times to meet deadlines[J],. Modelling and Performance Evaluation of Computer Systems,1976,6(5),162-164

[8] H Chetto, M Silly and T Bouchentouf. Dynamic Scheduling of Real-Time Tasks under Precedence Constraints[J]. RTS International Journal of Time Critical Computing Systems,1990,2(3),181-194

[9] Li HongBO, Zai JinGang. A Parallel Algorithm of Topological Sort on Digraph[J]. Yantai Normal University Journal(Natural Science Edition). 2005,21(3),168-171