Journal of Engineering, Sciences, Assiut University, Vol. 35, No. 6, pp. 1443-1453, November 2007

SCHEDULING HARD REAL-TIME TASKS WITH PRECEDENCE CONSTRAINTS ON MULTIPROCESSOR SYSTEMS

E. M. Saad, H. A. Keshk, M. A. Saleh, and
Faculty of Engineering Helwan University, Cairo

A.A. Hamam
Thebes Academy

(Received March 19, 2007 Accepted November 3, 2007)

In this paper, a scheduling algorithm based on deadline time and precedence constraints was developed to schedule hard real-time tasks on multiprocessor systems. The real-time tasks are characterized by their arrival time, deadline time, computation time and precedence constraints. Scheduling problem for these tasks has been solved to determine the order of scheduling tasks on the processors to minimize the overall computation time, and obtain speeding up. The effectiveness of the developed algorithm is shown through a simulation study.

KEY WORDS – *Task Graph, Optimization, Real-time system, Parallel processing, multiprocessor scheduling*

1. INTRODUCTION

A real-time system is one in which the correctness of the computations not only depends upon the logical correctness of the computations but also upon the time at which the result is produced [1-3]. Typical examples of real-time systems include air traffic control systems, networked multimedia systems, command and control systems etc. [2].

Each real-time system consists of a set of tasks some of these tasks are periodic in nature, and need to be activated regularly at fixed rates (periods). Normally, periodic tasks have constraints, which indicate that instances of them must be executed once per period. Other real-time tasks are aperiodic, and they are activated irregularly at some unknown and possibly unbounded rate. The time constraint is usually a deadline. [2]. Hence, periodic tasks consist of an infinite sequence of identical tasks that are regularly activated at a constant rate. Depending on the consequences of missing a deadline, real-time tasks are categorized as follows [2]:

1. Hard real-time systems, in which the execution of the task should be completed by a given deadline. i.e. Missing a deadline results in the failure of performance degradation of the system, [1-3]. Command and control systems, air traffic control systems are examples of hard real-time system.
2. Soft real-time systems, in which the system will properly perform as long as deadline is met most of the time. Missing a few deadline will not affected the system [1-3]. Telephone switching systems and image processing applications are examples of soft real-time systems.

3. Firm real-time systems, in which the task computations may be finished before their deadlines. [2].

Hard real-time system must execute a set of concurrent real-time tasks in such a way that all time critical tasks meet their specific deadlines.

Real-time scheduling can be categorized into hard and soft real-time scheduling. Hard real-time scheduling algorithms can be used for soft real-time scheduling.

Hard real-time scheduling can be classified into two types: Static and dynamic. In static scheduling, the scheduling decisions are made at compile time. A run-time schedule is generated off-line based on the prior knowledge of task-set parameters (execution time, precedence constraints, and deadline time). So, run-time overhead is small. On the other hand, dynamic scheduling makes its scheduling decisions at run-time, selecting one out of the current set of ready tasks. Dynamic scheduler is flexible and adaptive. Preemptive or non-preemptive scheduling of tasks is possible with static and dynamic scheduling. [2].

Multiprocessors systems give a powerful computation means for real-time applications [4-8]. Real-time systems use sophisticated scheduling algorithms to maximize the number of real-time tasks that can be processed without violating timing constraints.

In this paper, we present a scheduling algorithm to schedule hard real-time tasks with precedence constraints on multiprocessor systems . The proposed algorithm generates a feasible schedule by determining the processor which the task should be assigned , and the order in which tasks should be executed., so that the tasks meet its deadline constraints.

The paper is organized as follows, Section 2 reviews the related works in the area of scheduling hard real-time tasks, Section 3 describes scheduling problem, Section 4 discussed the proposed scheduling algorithm, in section 5 the simulation results are presented, Finally in section 6 our conclusions and summary are presented.

2. RELATED WORK

The problem of scheduling hard real-time tasks has been studied extensively and a number of algorithms have been proposed [4-10]. J. Jousson [4], proposed an improved version of the slicing technique and extends it to heterogeneous distributed hard real-time system. This technique can be applied to computational resources such as processors, shard data structures. He does not take into consideration general resources requirements. D. Wespetal, et. a.l., [5], proposed an algorithm for parallel task scheduling using long path. This algorithm could not be applied to other graph problems. K. Li [6], introduced an algorithm for scheduling parallel tasks on multiprocessor system. This algorithm was not used in scheduling real parallel computation on real parallel machines. V. Salmani, et. al., [7], proposed a modified version of the maximum uregency first scheduling algorithm which combines the advantages of fixed and dynamic scheduling to provide the dynamically changing systems with flexible scheduling. This algorithm, however, has a major shortcoming due to its scheduling mechanism that may cause a critical task to fail. A.Burchard, et. al., [8], developed an efficient heuristic algorithm for scheduling a set of periodic tasks

on a multiprocessor systems. This algorithm has better performance than rate-monotonic scheduling algorithm (RM). RM is optimal for uniprocessor systems with fixed priority assignments. A.S. Wu, et. al., [9], developed a genetic algorithm (GA) approach to the problem of task scheduling for multiprocessor systems. Key features of this approach include a flexible, adaptive problem representation and an incremental fitness function. The advantages of this algorithm are that it is simple to use, require minimal problem specific information and is able to effectively adapted in dynamically changing environment. The primary disadvantage of this algorithm is that it has a long execution time. J. Goossens, et. al., [10], proposed a new priority-driven scheduling algorithm for scheduling periodic task system (PriD). They prove that the PriD algorithm is superior to the earliest deadline first algorithm (EDF) in the sense that schedules all periodic task systems that EDF can schedule, and in addition schedules some periodic task systems for which EDF may miss some deadlines.

3. SCHEDULING PROBLEM

The input of scheduling hard real-time tasks system with precedence constraints is a set of real-time tasks and precedence constraints.

Each task is defined as follows T= {j,a,c,d}

Where
j is the serial number of the task
c is the computation time (also called worst-case computation time or execution time or processing time) of the task
a is the arrival time of the task; the time at which a task arrive to system (sometimes called ready time or the earliest time at which task can start its processing)
d is the deadline of the task; the time by which the task complete its execution.

The problem is scheduling these tasks so that all of the tasks should meet their timing constraints. Sometimes the ready time of a job may be later than that its successors, or its deadline may be earlier than that specified for its predecessors. This situation makes no sense. So, we apply the following method in order to achieve an effective ready time.

- If a job has no predecessors, its effective ready time is its ready time.
- If it has predecessors, its effective ready time is the maximum of its ready time and the effective ready times of its predecessors [2].

An effective deadline is calculated as:

- If a job has no successors, its effective deadline is its deadline.
- If it has successors, its effective deadline is the minimum of its deadline and the effective deadlines of its successors [2].

3.1 Task Model

We assume that the real-time system consists of n tasks and m (m>1) identical processors, which are connected through fully connected network without communication cost.. A task may be assigned to any one of the processors. Each task is aperiodic and Non-preemptive and is describe by:

at arrival time
ct computation time
dt deadline time
st start time
ft finish time

A task T_i meets its deadline if $at_i \leq st_i \leq dt_i - ct_i$ and $at_i + ct_i \leq ft_i \leq dt_i$. The start time of the task T_i is $st_i = \max(at_i, ptime(p_i))$ where $ptime(p_i)$ is the time at which the processor, p_j becomes available. The deadline time $dt_i = at_i + ct_i +$ variable value between [60,70] time units, and the finish time $ft_i = st_i + ct_i$.

4. PROPOSED SCHEDULING ALGORITHM

The proposed algorithm uses scheduling method based on deadline time and task dependency. First, all tasks are put in the queue, the scheduler picks the first task and tries to assign it to the available processor. At time zero all processors are available.
The pseudo code for our algorithm is as follows:

```
Begin
    While not empty queue Do
        Begin
            Get a task at the front of the queue
            Select a suitable processor
            Calculate starting time of a task
            Assign task to the selected processor
            Calculate finish time of a task
            Check a task for meeting its deadline
                If a task meets its deadline add it to schedule list
                Other wise add it to missing list
        End
    Calculate critical path
    Calculate speed up parameter
End
```

Where :
Critical path is the longest path from the entry node to the exit node.
Speed up parameter is the total processing of all tasks in task graph over completion of all tasks on multiprocessor according to developed algorithm.

4.1 Selecting A Suitable Processor

We keep track of all processors in the system by using the array ptime of p(j), j=1,2, …. , m (number of processors) which represent the time at which the processor p(j) becomes available. Ex. Suppose we have four processors; at time zero all processors are available i.e. ptime [p[j]] =0 for all j=1,2, … 4.. At time unit 16 a new task arrives to the system and ptime [p[j]] was as follows:
Ptime [p[j]]

13	23	15	30

In this situation we select the processor number one because the processor number one has minimum time out of the different available time of all processors.

The following is the pseudo code for the procedure of selecting a suitable processor according to the criteria of selecting a processor with minimum available time.

Input: arrival time of the task(at) , the available time of all processors
Output: a suitable processor, starting time of the task (st)
Begin

If a task T_i has no predecessor tasks

$$Minval = \min_{J=1}^{m} (ptime[p[j]])$$

Suitable processor is a processor with minval
Starting time st $[T_i]$ =Max $[at[T_i]$, minval]

Enid if
If a task T_i has one predecessor task T_k

$$Minval = \min_{J=1}^{m} (ptime[p[j]])$$

Suitable processor is a processor with minval
Maxval=Max $[at[T_i]$, $ft[T_k]]$
Where ft is the finish time of the predecessor task T_k
Starting time st $[T_i]$ =Max [maxval, minval]

End if
If a task T_i has more than one predecessor tasks $T_{i-1,\ ,\ ,\ldots\ldots,\ ,}\ T_k$

$$Minval = \min_{J=1}^{m} (ptime[p[j]])$$

Suitable processor is a processor with minval
Maxval=Max $[at[T_i]$, $ft[T_{i-1}]$, $ft[T_{i-2}]$, ……., $ft[T_k]$]
Starting time st $[T_i]$ =Max [maxval, minval]

End if

End

To implement the proposed algorithm efficiently we maintain four lists and one task dependency matrix described as follows:

- at List of arrival time for all tasks in the system
- ct List of computation time for all tasks in the system.
- dt List of deadline time for all tasks in the system.
- sl List of scheduled tasks.
- pt matrix represent the task dependency.

Illustrative example for Pt: we consider a ten task T1,T2, ….., T10. as an example

Pt

T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
0	0	0	0	1	2	0	0	7	0
0	0	0	0	4	3	0	0	8	0

From the above figure we notice that pt [0][6]!= 0 && pt [1][6] !=0. This means that the task T6 depend on T2 and T3, so task T6 must start its computation after tasks T2 and T3 finishes their computations.

The flowchart used to describe our algorithm is as follows:

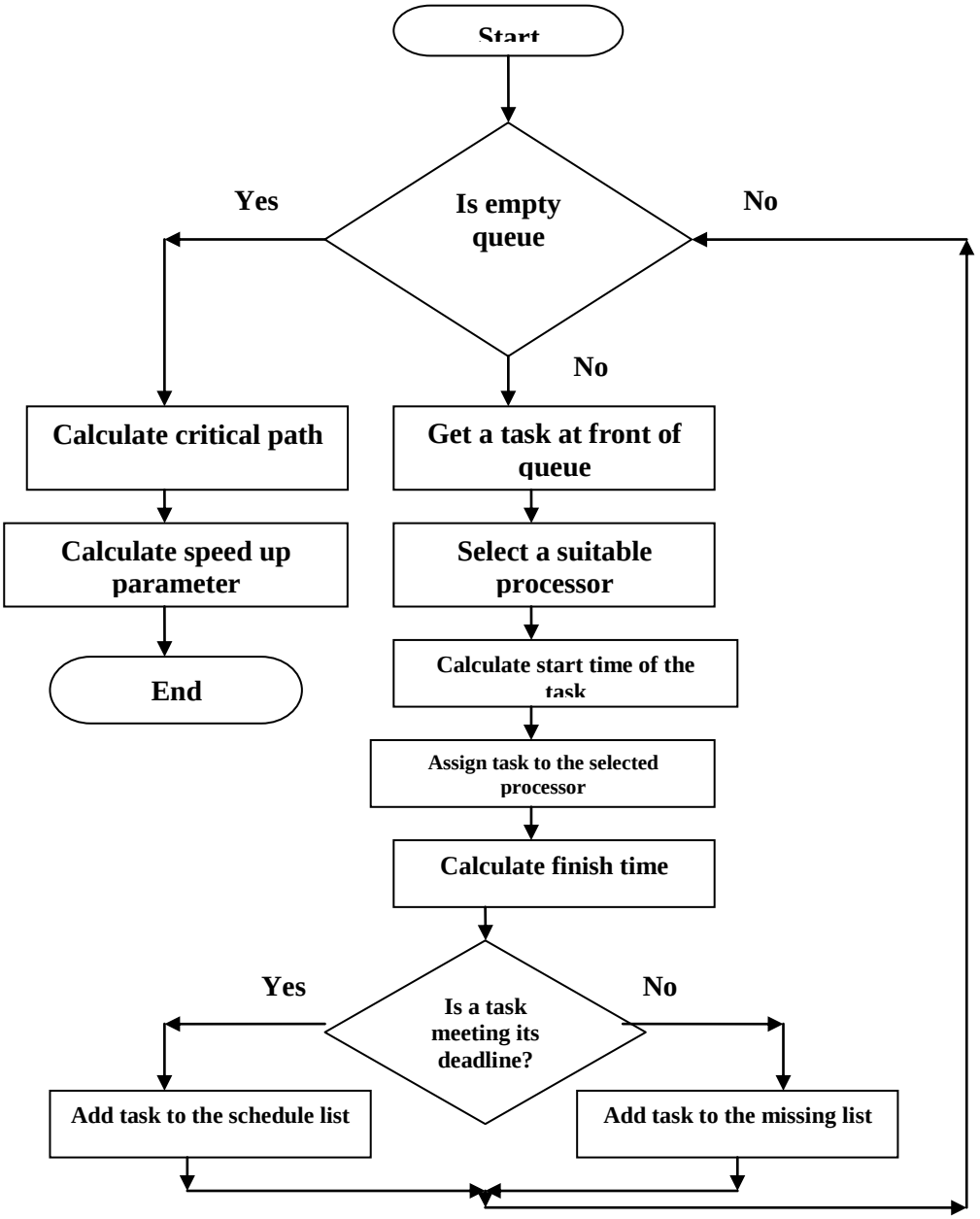


Fig. (1): A flowchart for our algorithm

The following is an illustrative example for scheduling ten tasks on three processors.

Task No.	At	Ct	Predecessor tasks	St	ft	Suitable processor
T1	5	2		6	8	P1
T2	6	3	1	9	12	P1
T3	6	3		6	9	P2
T4	6	8		6	14	P3
T5	6	6		10	16	P2
T6	12	2		13	15	P1
T7	12	2		15	17	P3
T8	12	5		16	21	P1
T9	12	7	1, 3	17	24	P2
T10	12	1	9	25	26	P2

First, we schedule the task number one on any of the three processors, then we select processor number one; on which task one starts its processing at time unit 6 and finishes at time unit 8. We notice that task number two depends on task one, Therefore task two start its processing at time unit 9 although it arrives at time unit 6 and finishes at time unit 12. Similarly, tasks numbers nine and ten.

5. SIMULATION RESULTS

We examine our algorithm on two problems. First problem 50 (50-4) tasks with minimum number of predecessor tasks equal to 0, maximum number of predecessor tasks equal to 4 and average number of predecessor tasks equal to 1. Second problem 50 (50-12) tasks with minimum number of predecessor tasks equal to 0, maximum number of predecessor tasks equal to 12 and average number of predecessor task equal to 4.66 which should be schedule onto 6 processors. Number of tasks is generated randomly as computation times for every task and the precedence constraints are taken from [11]. In this case, the tasks arrive in the system 5 tasks every 6-time units. For this problem the critical path length which means that the longest path from the entry task to the exist task is 32, 112 time units respectively, and it has been also taken from [11]. Applying the developed algorithm, the critical path becomes 79,124 time units.

Second, we test the problem with 100 (100-4) tasks with minimum number of predecessor tasks equal to 0, maximum number of predecessor tasks equal to 4 and average number of predecessor tasks equal to 1. Second problem 100 (100-12) tasks with minimum number of predecessor tasks equal to 0, maximum number of predecessor tasks equal to 12 and average number of predecessor task equal to 3 which should be schedule onto 15 processors. In this case, the tasks arrive in the system 10 tasks every 6-time units. The critical path length is 40, 74 time units respectively and taken from [11]. Applying the developed algorithm, the critical path become 79, 87 time units.

Figures (2) and (3) show the relationship between number of processors used and the critical path length respectively. It is clear from the two figures that the critical path length decreased monotonically with increasing the number of processors until the

increasing of the number of processors does not effect on the critical path length (reach constant critical path length). This is the optimal situation.

Another factor is speed up parameter which means that total processing of all tasks in task graph over completion of all task on multiprocessors according to developed algorithm. For first problem are 8.9, 3.0 and 19.5, 10.87 for second problem. Also taken from [11]. Applying the developed algorithm becomes 3.5, 2.7 and 9.9, 9.8 respectively.

Also, another factor is used to evaluate the performance of the developed algorithm, the missing rate, which mean the percentage of tasks that missed the deadline time. Figures (4) and (5) show the relationship between number of processors used and missing rate respectively. It is clear from the two figures that the missing rate decreases monotonically with increasing the number of processors until the increasing of the number of processors does not affect on the missing rate (reach constant missing rate), this is the optimal situation.

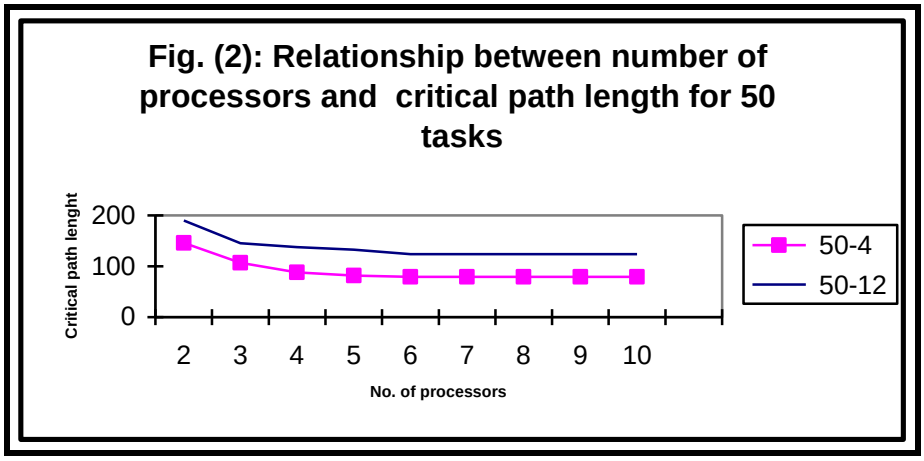


Fig. (2): Relationship between number of processors and critical path length for 50 tasks

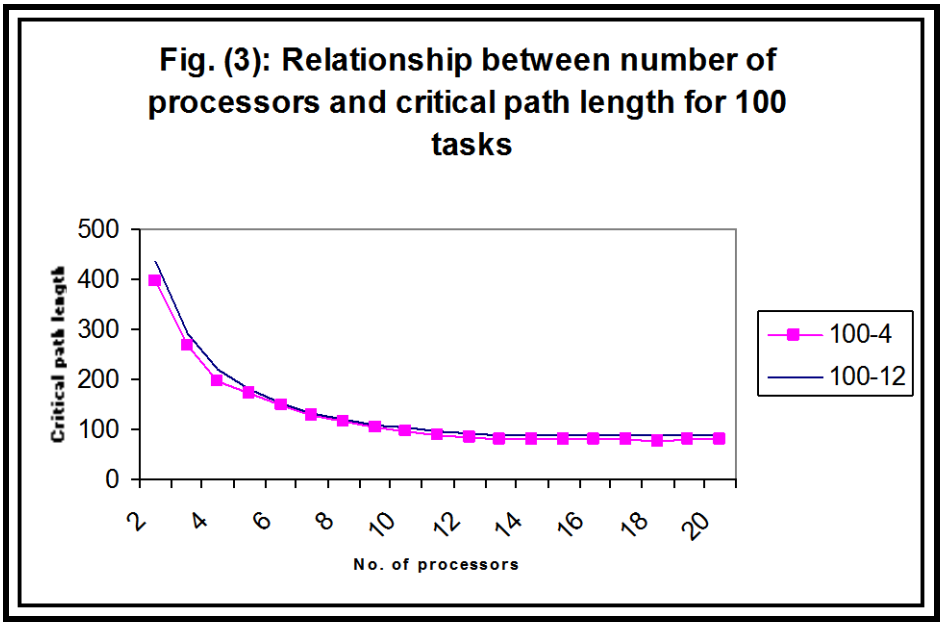
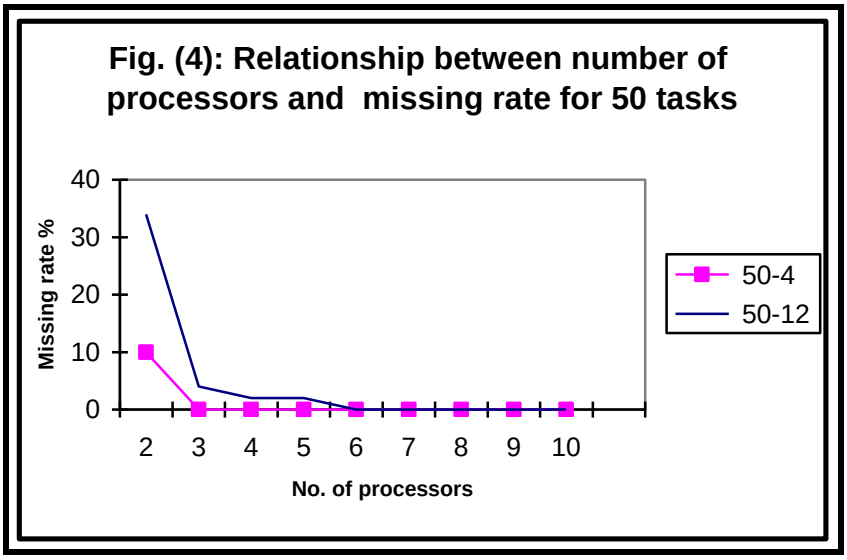
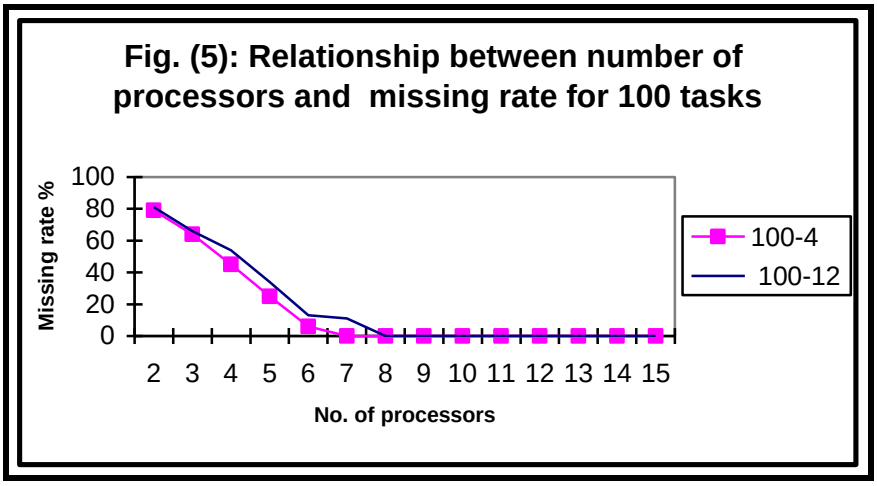


Fig. (3): Relationship between number of processors and critical path length for 100 tasks





The list of tasks scheduled at each time unit on all processors at optimal situation for case 50 tasks with four predecessor tasks are shown in the following table as an example.

P0	P1	P2	P3	P4	P5
T0 (6-8)	T2(6-9)	T3(6-14)	T4(6-12)	T5(12-14)	T6(12-14)
T1(8-11)	T7(12-17)	T11(18-21)	T8(12-19)	T12(30-40)	T13(18-21)
T10(18-27)	T15(24-28)	T16(24-32)	T9(19-20)	T20(30-40)	T18(24-29)
T21(30-31)	T22(30-32)	T17(32-40)	T14(20-22)	T33(42-51)	T23(30-39)
T24(31-39)	T25(36-40)	T32(42-49)	T19(24-32)	T42(54-59)	T29(39-46)
T28(39-42)	T31(42-45)	T38(49-53)	T26(36-41)	T48(60-61)	T36(48-49)
T30(42-48)	T35(48-56)	T43(54-61)	T27(41-51)	T49(61-70)	T37(49-51)
T34(48-58)	T46(60-70)		T41(54-61)		T39(51-55)
T47(60-62)					T40(55-64)
					T44(64-71)
					T45(71-79)

Where T_i(n1,n2): i is task number, n1 is the start time of the task, and n2 is the finish time of the task.

Experimental results on the relatively hard problems that have been taken from Internet [11] does not consider the deadline time of the task by which the task should be completed. In our algorithm we override this problem, and our result does not look as bad solution because it has only a few time units less than that for the optimal solution [11], especially in case of heavy task graph.

6. CONCLUSION

Meeting deadlines and achieving high processor utilization are two main goals of task scheduling in real-time systems. In this paper, we proposed a scheduling algorithm for scheduling hard real-time tasks based on deadline time by which the task must finish its execution and including precedence constraints on multiprocessor systems which would simulate the practical applications.. Experiment results using task graph taken from [11] show that our algorithm gives near optimum solution specially for heavy task graph, and confirm the effectiveness of the proposed algorithm.

REFERENCES

[1] D. Stewert , "Introduction to real-time scheduling theory", University of Maryland, Department of Electrical & Computer Engineering, springer Verlag, 2000.

[2] A. Mohammadi and S.G. Akl (2005):"Scheduling algorithms for real-time systems", School of Computing, Queen's University, Kingston, Ontario Canada, July 15, 2005.

[3] L. Sha, T. Abd Elzaher, A. Karl-Erik, A. cervin, T. Baker, A. Burns, G. buttazzo, M. Caccamo, J. Lehoczky, A. K. Mok, "Real time scheduling theory: A historical perspective ", real-time system, 28, 101-155, 2004.

[4] J. Jonsson, :"A Robust adaptive metric for deadline assignment in heterogeneous distributed real-time system", Proceedings of the IEEE Int'l Parallel Processing Symposium, San Juan, Puerto Rico, April 12-16, 1999, PP. 678-687.

[5] D. Wespetal, J. Netson and D. Lopez, "Approximating a parallel task schedule using longest path",University of Minnesota-Morris, USA, 2003. http://cda..morris.umn.edu/~lopezdr/publications/longestpath03.doc

[6] K. Li, "Scheduling precedence constrained parallel tasks on multiprocessor using the harmonic system partitioning scheme", Journal of Information Science and Engineering, 21, 309-326, 2005.

[7] V. Salmani, .T. Zargar, and M. Naghibzadeh, "A modified maximum uregency first scheduling algorithm for real-time tasks", Trans. On Engineering Computing and Technology, vol. 9, November 2005, ISSN 1305-5313.

[8] A. Burchard, Y. Oh, J. Liebeherr, S.H. Son, "A linear-time online task assignment scheme for multiprocessor systems", Proceedings, 11th IEEE workshop Real-time operating systems and software, PP. 28-31, may 1994.

[9] A.S. Wu, H. Yu, S. Jin, K.C.Lin, and G. Schiavone, "An incremental genetic algorithm approach to multiprocessor scheduling", IEEE Trans. On Parallel and Distributed systems, Vol. 15, No. 9, Sept. 2004, PP. 824-834.

[10] J. Geoussens, S. Funk, and S. Baruah, :"Priority-driven scheduling of periodic task systems on multiprocessors", Real-time systems, 25, 187-205, 2003.
[11] Advanced computing systems, available from: http://www.kasahara.elec.waseda.ac.jp

ملخص البحث: جدولة المهام للنظم متعددة المعالجات في الزمن الفعلي

في هذا البحث تم تطوير خوارزم جدولة يعتمد علي الوقت الذي لابد ان يتم تنفيذ المهام خلاله و الشروط المسبقة لجدولة المهام للنظم متعددة المعالجات في الزمن الفعلي. المهام في الزمن الفعلي توصف بزمن وصولها الي النظام ، و الزمن الذي لابد ان تنتهي خلاله ، زمن التنفيذ و الشروط المسبقة.

مشكلة الجدولة لهذه المهام لابد ان تحل لتحديد ترتيب تنفيذ المهام علي المعالجات و ذلك لتقليل وقت التنفيذ و زيادة السرعة. فاعلية الخوار زم المقترح يتم اختبارها من خلال دراسة تجريبية.