

Received December 26, 2019, accepted January 12, 2020, date of publication January 24, 2020, date of current version January 31, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2969229

Research on Real-Time Embedded Software Scheduling Model Based on EDF

YINGJIE WANG^{1,2,3}, KUANJIU ZHOU^{1,3}, ZUMIN WANG², MINGCHU LI^{1,3},
NAN CHEN², BIN LI^{2,4}, AND HONGXUAN TIAN^{1,2}

¹Software School, Dalian University of Technology, Dalian 116620, China

²College of Information Engineering, Dalian University, Dalian 116622, China

³Liaoning Provincial Key Laboratory of Ubiquitous Network and Service Software, Dalian University of Technology, Dalian 116620, China

⁴School of Software Engineering, University of Science and Technology of China, Suzhou 215000, China

Corresponding author: Kuanjiu Zhou (zhoukj@dlut.edu.cn)

This work was supported in part by the Special Fund for Basic Scientific Research Business Expenses of Central Universities under Grant DUT19ZD104, and in part by the National Nature Science Foundation of China under Grant 71531002 and Grant 71421001.

ABSTRACT Schedulability analysis is a very important part in real-time system research. Because the scenarios faced by real-time systems are very complicated, the functional characteristics must be combined with the predictability of response time. It is necessary to ensure the correctness of the calculation results and meet the real-time requirements. To solve this problem, we propose the IEDF (Improved Earliest Deadline First) algorithm, which is combined with the queuing theory model. The IEDF algorithm is based on the EDF (Earliest Deadline First) algorithm, which is more suitable for the scheduling of real-time embedded system. Scheduling of non-periodic tasks that arrive randomly. There are two types of tasks in the task set, tasks with a high static priority are executed first. In the ready queue of the same priority task, the deadline and execution time are considered. The comparison of simulation experiments shows that: the sum of waiting time in the execution of IEDF with enough deadline is much less than that of ordinary queuing algorithm; the number of errors in the execution of IEDF algorithm with deadline is much less than that of ordinary queuing algorithm. These results demonstrate the feasibility of the IEDF algorithm.

INDEX TERMS Deadline, real-time scheduling, EDF (earliest deadline first) algorithm.

I. INTRODUCTION

The correctness of a real-time system depends not only on the logical results of the system's execution, but also on the response time for the results. When designing a real-time embedded system, we must not only ensure the correctness of the calculation results, but also meet the real-time requirements. The results must be produced within a specified time limit. In order to satisfy the strong timeliness of data processing and control, the system divides the real-time task into several groups and schedules them according to the urgency of the task in the real-time system. In real-time embedded software, the execution of tasks has a deadline, and time constraints limit the execution time of specific tasks, such as the start, end or duration of tasks. When the execution time satisfies the time constraint, the software will exhibit timing characteristics that meet the design expectations [1].

A real-time task is required to be completed before the deadline. Depending on types of systems, missing deadlines

The associate editor coordinating the review of this manuscript and approving it for publication was Vivek Kumar Sehgal.

will cause performance losses or even complete failures of the real-time systems. Another important concept is the release time, which is the time when a real-time job becomes available for execution. The release time and the deadline together can specify a timing constraint of a real-time job.

The timing constraints are generally divided into two types: hard and soft. There are several different definitions of hard and soft timing constraints.

- A real-time constraint is hard, if violating this constraint is considered as a fatal fault and may cause serious consequences.

- A real-time constraint is soft, if meeting this constraint is desirable, but missing this constraint does not seriously damage the system behavior [2].

Taking the access control system as an example, the access control software is a typical embedded software. When a specific event is triggered, the access control is opened or closed according to a certain timing. The state machine model is shown in Figure 1.

A series of time constraints are set in the access control software, including: the access control software will be

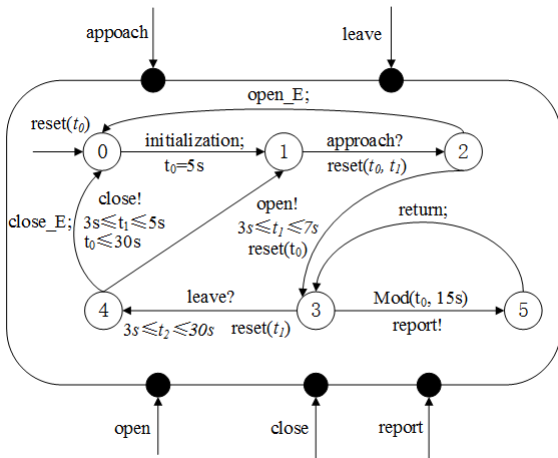


FIGURE 1. Timed automata for access control software with time constraints.

initialized 5s after the system is powered on; The automatic door will be opened within the time interval $[3s, 7s]$ after a personnel approach is detected; the automatic door will be closed within the time interval $[3s, 5s]$, after a personnel leave is detected; When a personnel approach is detected, the person should be detected to leave within the time interval $[3s, 30s]$; After the automatic door is opened, the automatic door should be closed within 30s, that is, the automatic door should not be opened for more than 30s; When the automatic door is opened (state 3), the access control software shall report the opening state of the automatic door to the monitoring center every 15s [3].

Although there are some scheduling methods for real-time system, they lack full consideration of the effects of time constraint or they consider too pessimistic and cause a lot of tasks to be abandoned. In order to ensure the real-time performance of the real-time system, in this paper, we present an IEDF (Improved Earliest Deadline First) scheduling algorithm. In particular, we make the following contributions:

- (1) EDF (Earliest Deadline First) algorithm is a classical dynamic priority scheduling algorithm. We propose the IEDF (Improved Earliest Deadline First) algorithm. The IEDF algorithm can be applied to non-periodic and randomness tasks, it is more universal than the universality of EDF algorithm.
- (2) We present the IEDF algorithm, which is combined with the queuing theory model. We can clearly analyze the relationship among the arrival time, execution time, waiting time, waiting queue length and execution end time of tasks through the queuing model.
- (3) We compare the execution results of three algorithms, which are the ordinary queuing algorithm, EDF algorithm and IEDF algorithm. It provides a basis for the selection of scheduling algorithms for real-time embedded systems in the future.

The rest of the paper is organized as follows. We introduce related works in Section II. The related description of the IEDF algorithm model and the modeling method

are presented in Section III. Section IV presents the model implementation. Section V presents an experimental comparison among the execution results of the ordinary queuing algorithm, EDF algorithm and IEDF algorithm. We conclude this paper in Section VI.

II. RELATED WORK

Real-time scheduling of embedded software is a hot issue in the current research, Chang *et al.* [4] explored the joint considerations of memory management and real-time task scheduling over island-based multi-core architecture, it minimized the number of needed islands to successfully schedule real-time tasks. But its flexibility wasn't good, and its scheduling capacity needs to be improved. The Xian-Fu and Xiao-Yan [5] considered execution time, communication time among nodes and task scheduling costs, a parallel task scheduling algorithm with multi-objective constraints was presented. Experimental results demonstrated that the task scheduling algorithm with multi-objective constraints had better performances than the traditional methods. Qingbing *et al.* [6] proposed a real-time automatic online evaluation method for CPS reliability, in order to effectively analyze and quantify the reliability of CPS system. The method uses machine learning to construct an evaluation framework, and designs an online queuing algorithm to realize real-time online analysis and evaluation of CPS reliability. Using a logic based approach to schedulability analysis in the design of hard real-time systems eases the synthesis of correct-by-construction procedures for both static and dynamic verification processes. Pedro *et al.* [7] proposed a novel approach to schedulability analysis based on a timed temporal logic with time durations. The approach subsumes classical methods for uniprocessor scheduling analysis over compositional resource models by providing the developer with counter-examples, and by ruling out schedules that cause unsafe violations on the system.

The use of hardware-based data structures for accelerating real-time and embedded system applications is limited by the scarceness of hardware resources. Kumar *et al.* [8] present a hybrid priority queue architecture and a scalable task scheduler for real-time systems that reduces scheduler processing overhead and improves timing determinism of the scheduler. Fajardo and Drekić [9] proposed the concept of cumulative priority, the main purpose is to describe the latency distribution of each queue. The system is preemptive, so as to serve customers with higher cumulative priority. However, the data-missing will be caused by the overtime duration of processing because there is no consideration of the deadline issue for this method. Muliukha *et al.* [10] described a preemptive dual-stream queuing system method with random rollout probability, and proposed an efficient algorithm for controlling the data flow of the system. And put that probability is the main parameter of the queuing system.

Scheduling in mixed criticality system is a hot topic. With Vestal's research on scheduling in mixed criticality system, many researchers have subsequently proposed various

scheduling algorithms, such as EDF-VD algorithm, which sets virtual deadlines for tasks with a higher criticality. These algorithms have become typical algorithms on mixed criticality system. In this system, tasks have their criticality levels and worst-case execution time (WCET). WCET represents the longest time it takes for a task to run on a specific platform. When a high-critical task is actually executed, if it has not finished when running time reaches a certain critical point, usually the low-critical tasks are immediately discarded, ensuring that the high-critical tasks can be successfully finished executing before the deadline.

For the 'ordinary' non-MC (mixed criticality) scheduling, the fixed priority policy is sufficient and the EDF (earliest-deadline first) priority assignment algorithm is optimal for any schedulable instance [11], but it does not apply to random aperiodic tasks. Park and Kim [12] introduced a slack-based mixed criticality scheme for EDF scheduled jobs which they called CB-EDF (Criticality Based EDF). In essence they use a combination of off- and on-line analysis to run HI-criticality jobs as late as possible, and LO-criticality jobs in the generated slack. Alternative analysis for EDF scheduled MCS is presented by Mahdiani and Masrur [13] and Santinelli *et al.* [14]. The former derives a separate demand bound function for transition to HI mode. Working around carry-over-jobs, reducing pessimism and relaxing schedulability condition HI mode under MC EDF, resulting in simpler schedulability test and tighter bound on execution demand. The latter make use of multiple demand bound curves to allow sensitivity analysis to be derived that can be applied to the trade-off between resource usage and schedulability [15].

III. MODEL DESCRIPTION

The IEDF algorithm model stores the incoming tasks sequence in the message queue and uses the message queue to drive the state machine to execute. The task set includes two types of tasks: low-priority tasks and high-priority tasks. Tasks with high static priority will be executed first, so high-priority tasks will preempt low-priority tasks. In the ready queue of the same priority tasks, the deadlines are prioritized. When the waiting time of the task is close to the deadline of the task, the task will be executed first, and will be discarded if the waiting time of the task exceeds the deadline. EDF (Earliest Deadline First) algorithm is a classic dynamic priority scheduling algorithm [16]. The priority of tasks in this algorithm is determined according to their Deadline, and the tasks with the latest Deadline are assigned the highest priority, so that they can be scheduled first. In this paper, the improved EDF algorithm is combined with the queuing theory to establish the model, which is called IEDF (Improved EDF) algorithm. The model is described as follows:

In general, tasks τ_i in an embedded soft real-time system are described formally as a 6 tuple:

$$\tau_i = (S_i, W_i, T_i, D_i, L_i, P_i)$$

S_i is the arrival time of task τ_i , and assume that the arrival time of the task is its ready time; W_i is the wait time of the

task; T_i is the Actual Execution Time of the task, and due to being in an uncertain environment, this time length is usually time-varying; D_i is the relative deadline for the task; L_i is the waiting queue length of the task; P_i is the priority of the task.

(1) An instance of each task in the algorithm has different runtimes, and it's not a fixed value but a random value randomly assigned within a range.

(2) Static P_i priority is divided into two categories: Class I priority and Class II priority, and Class I priority is higher than Class II priority.

(3) Class I priority tasks and Class II priority tasks enter the queuing system with "Poisson" form, obeying the Poisson process of λ_1 and λ_2 respectively. Class I priority tasks and Class II priority tasks receive services with a negative exponential distribution of u_1 and u_2 , respectively.

(4) Class I priority tasks have preemptive priority for Class II priority tasks. In a priority queuing system, when a Class I priority task arrives at the system, if there is no execution task in the queue, the system will immediately respond to the task; If the system is processing a Class II priority task service, the Class I priority task will preempts receiving the service, and the Class II priority task returns to the queue to continue to wait for receiving the service.

(5) In this algorithm, the final priority of task scheduling is not only determined by static priority (Class I and II), but also determined by scheduling priority (like deadline and execution time). The following provisions are made in this algorithm:

- Tasks with a high static priority are executed first.
- In the ready queue of the same priority task, the deadline and execution time are considered.

(6) Set two queues, each queue is set according to the flag bit, and the high priority preempts the low priority task. The IEDF algorithm is used to deal the same level tasks (the same flag). When the task waiting time W_i is close to the task relative deadline D_i , the task τ_i is preferentially executed, and if the task waiting time W_i exceeds the relative deadline D_i , the task will be discarded.

The difference between IEDF algorithm and EDF algorithm:

(1) The typical EDF algorithm assumes that all tasks are ready at the same time, without considering the randomness of the tasks arrival time. The IEDF algorithm assumes that the tasks enters the queuing system with Poisson distribution, obeying the Poisson process of λ_1 and λ_2 respectively, which is more consistent with the actual situation when the task arrives.

(2) The typical EDF algorithm assumes that all tasks are periodic tasks. All tasks in IEDF algorithm are not periodic tasks, and each task has different running time, which is more consistent with the task execution process of embedded system.

(3) In a typical EDF algorithm tasks' priorities change according to their deadlines. In IEDF algorithm, the final priority of task scheduling is not only determined by deadline, but determined by static priority, deadline and execution time together.

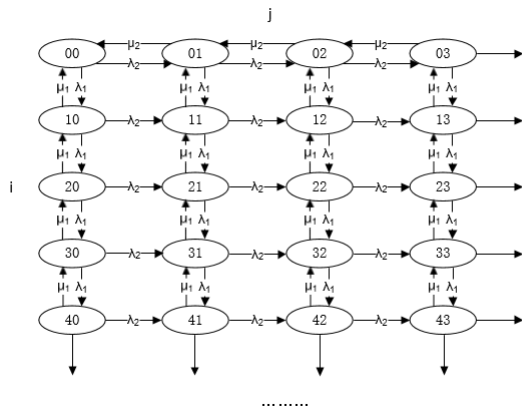


FIGURE 2. State space and transferring process.

Suppose $N(t) = \{N_1(t), N_2(t)\}$ are two types of priority tasks in the system. $N_1(t)$ has a higher priority than $N_2(t)$, so $N(t) = \{N_1(t), N_2(t)\}$ is a two-dimensional Markov process. When the system reaches a steady state, set

$$P_{ij} = \lim_{t \rightarrow \infty} P\{N_1(t) = i, N_2(t) = j\} \quad (1)$$

$$P_i = (P_{i,0}, P_{i,1}, P_{i,2}, \dots, P_{i,N}) \quad i \geq 0 \quad (2)$$

will get state space and transferring process, as shown in Figure 2.

When the service system runs long enough, the system goes into a stable state. At this time, for each state, the number of tasks entering this state is equal to the number of tasks leaving the state for a fixed period of time. Obtain the stationary equation in each state as follows:

$$(\lambda_1 + \lambda_2)P(0, 0) = u_2P(0, 1) + u_1P(1, 0) \quad (3)$$

$$(\lambda_1 + \lambda_2 + u_2)P(0, j) = u_2P(0, j + 1) + u_1P(1, j) + \lambda_2P(0, j - 1), \quad j \geq 1 \quad (4)$$

$$(\lambda_1 + \lambda_2 + \mu_1)P(i, 0) = \lambda_1P(i - 1, 0) + u_1P(i + 1, 0), \quad i \geq 1 \quad (5)$$

$$(\lambda_1 + \lambda_2 + \mu_1)P(i, j) = \lambda_1P(i - 1, j) + \mu_1P(i + 1, j) + \lambda_2P(i, j - 1), \quad i \geq 1, j \geq 1 \quad (6)$$

The matrix is used to represent the service process, then the generator matrix of this service process is Q

$$Q = \begin{pmatrix} S_0 & E & 0 & 0 & 0 & 0 & \dots \\ F & S_1 & E & 0 & 0 & 0 & \dots \\ 0 & S_2 & S_1 & E & 0 & 0 & \dots \\ 0 & 0 & S_2 & S_1 & E & 0 & \dots \\ 0 & 0 & 0 & S_2 & S_1 & E & \dots \\ 0 & 0 & 0 & 0 & S_2 & S_1 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \quad (7)$$

$$S_0 = \begin{pmatrix} -\lambda & \lambda_2 & 0 & 0 & \dots \\ \mu_2 & -(\lambda + \mu_2) & \lambda_2 & 0 & \dots \\ 0 & \mu_2 & -(\lambda + \mu_2) & \lambda_2 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \quad (8)$$

$$S_1 = \begin{pmatrix} -(\lambda + \mu_1) & \lambda_2 & 0 & 0 & \dots \\ 0 & -(\lambda + \mu_1) & \lambda_2 & 0 & \dots \\ 0 & 0 & -(\lambda + \mu_1) & \lambda_2 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \quad (9)$$

$$E = \begin{pmatrix} \lambda_1 & & & & \\ & \lambda_1 & & & \\ & & \lambda_1 & & \\ & & & \ddots & \\ & & & & \lambda_1 \end{pmatrix} \quad (10)$$

$$F = S_2 = \mu_1 I = \begin{pmatrix} \mu_1 & & & & \\ & \mu_1 & & & \\ & & \mu_1 & & \\ & & & \ddots & \\ & & & & \mu_1 \end{pmatrix} \quad (11)$$

IV. MODEL IMPLEMENTATION

Assuming a higher-priority task request arrives, the system can consider as that no other level of task request exists. At this time, the system can be regarded as the Poisson distribution whose task arrival rate obeys the parameter λ_{I_1} , and the service time of the task obeys the M/M/1 queuing system with the negative exponential distribution of the parameter μ_{I_1} [17,18]. W_{q_i} and W_{s_i} are the average queue waiting time and average stay time of the task of the level i priority in the system. It is easy to get the average queue waiting time $W_{q_{I_2}}$ and average dwell time $W_{s_{I_2}}$ of level I_1 task in the system,

$$W_{q_{I_1}} = \frac{L_{q_{I_1}}}{\lambda_{I_1}} = \frac{\lambda_{I_1}}{\mu_{I_1}(\mu_{I_1} - \lambda_{I_1})} \quad (12)$$

$$W_{s_{I_1}} = \frac{L_{s_{I_1}}}{\lambda_{I_1}} = \frac{1}{\mu_{I_1} - \lambda_{I_1}} \quad (13)$$

$L_{q_{I_1}}$ is the average wait length of the system under M/M/1 model, and $L_{s_{I_1}}$ is the average wait length of the system.

When a lower priority task request arrives, the system can be considered to have I_1 and I_2 priority task requests due to preemption priority. $W_{s_{I_1 \sim I_2}}$ is the average dwell time of I_1 and I_2 priority task requests in the system, that:

$$(\lambda_{I_1} + \lambda_{I_2}) W_{s_{I_1 \sim I_2}} = \lambda_{I_1} W_{s_{I_1}} + \lambda_{I_2} W_{s_{I_2}} \quad (14)$$

that is,

$$W_{s_{I_2}} = \left(1 + \frac{\lambda_{I_1}}{\lambda_{I_2}}\right) W_{s_{I_1 \sim I_2}} - \frac{\lambda_{I_1}}{\lambda_{I_2}} W_{s_{I_1}} \quad (15)$$

For $W_{s_{I_1 \sim I_2}}$ in the formula, when the higher-priority task request comes, the service of the task that is being served at the lower level can only be interrupted and rejoin the queue. Its service is still negative exponential distribution and the parameters are the same as the previous. $W_{s_{I_1 \sim I_2}}$ can be considered as the M/M/1 queuing system with the arrival rate $\lambda = \lambda_{I_1} + \lambda_{I_2}$, and the service time is

$$W_{s_{I_1 \sim I_2}} = \frac{1}{\mu - (\lambda_{I_1} + \lambda_{I_2})} \quad (16)$$



FIGURE 3. Ordinary queuing algorithm execution results with enough deadlines.

Substitute equation (13) and (16) into equation (15), then

$$W_{s_{l_2}} = \left(1 + \frac{\lambda_{l_1}}{\lambda_{l_2}}\right) \left[\frac{1}{\mu - (\lambda_{l_1} + \lambda_{l_2})} \right] - \frac{\lambda_{l_1}}{\lambda_{l_2}} \left(\frac{1}{\mu I_1 - \lambda_{l_1}} \right) \tag{17}$$

$$W_{q_{l_2}} = W_{s_{l_2}} - \frac{1}{\mu} \tag{18}$$

The pseudo-code of the IEDF algorithm is shown as Algorithm 1.

Space Complexity of the algorithm: $O(2n)$. Execute queue pool n , and it is additionally required to set two high/low priority pools of length n . Therefore, the space complexity is $O(2n)$.

Algorithm time complexity: All processes need to be cycled n times. Each cycle needs to sort the execution job pool, and the task approaching the deadline is executed first. The time complexity of the sorting is n , because the data in the execution job pool is almost ordered, and the total time complexity $T(n) = O(n^2)$.

V. EXPERIMENTS

This scheduling algorithm is based on the ideal environment as follows: (1) tasks are independent, that is, each task’s request does not depend on other tasks’ start or completion requests; (2) the resources of the task execution environment are sufficient, that is, except CPU resources there are no resources competing; (3) some time costs are ignored such as task switching and performance adaptive control calculation; (4) when a task exceeds the deadline, the task will abandon the execution and give up the CPU resources. (5) the task is not allowed to be preempted when it is executed in the critical region.

In order to test the IEDF scheduling algorithm proposed in this paper, a simulation comparison experiment is conducted between this algorithm and the ordinary queuing model [19]. There are 20 test tasks in the task set in Table 1, and they arrive randomly. Setting flag bits for each task represents its

Algorithm 1 Pseudo-Code for IEDF Algorithm

```

Input:
Task information
Output:
Output tasks in sequence
1: Input All tasks enter the wait queue
2: Procedure EDF(pool)
3: Initialization highTaskList High priority list, lowTaskList Low priority list
4: time = 0;
5: while pool.length > 0 && highTaskList.length>0 && lowTaskList.length>0 do
6:   if pool.queueFront().arriveTime == time then // The arrival time of the first task in the waiting queue is equal to the current time
7:     work = pool.pop();
8:     if High priority task then
9:       highTaskList.push(work);
10:    else
11:      lowTaskList.push(work);
12:    if highTaskList.length>0 then
13:      Update highTaskList Other tasks waiting time
14:      Update highTaskList.queueFront().restTime-1
15:      if highTaskList.queueFront().restTime == 0 then
16:        work = highTaskList.pop();
17:        output work;
18:      sort by (the task with a small value between the cut-off time and the remaining running time get priority)
19:    if highTaskList.length>0 && highTaskList.queueFront().waitTime+highTaskList.queueFront().restTime > highTaskList.queueFront().offTime then
20:      work = highTaskList.pop();
21:      output work(Forced out of the queue);
22:    else Low priority queue is not empty then
23:      Update lowTaskList Wait times for other tasks
24:      Update lowTaskList.queueFront().restTime-1
25:    if lowTaskList.queueFront().restTime == 0 then
26:      work = lowTaskList.pop();
27:      output work;
28:    sort by (the task with a small value between the cut-off time and the remaining running time get priority)
29:    if lowTaskList.length>0 && lowTaskList.queueFront().waitTime+lowTaskList.queueFront().restTime > lowTaskList.queueFront().offTime then
30:      work = lowTaskList.pop();
31:      output work(Forced out of the queue);
32:    else
33:      Exit
34:    end
35:  end procedure
    
```

TABLE 1. Test data.

The serial number	TASKS	Priority flag bit (0 Low 1 high)	Arrival time	Required Execution Time
1	Task 1	0	1703	2360
2	Task 2	0	1978	2118
3	Task 3	1	3274	516
4	Task 4	1	3386	3277
5	Task 5	1	3900	1692
6	Task 6	1	4400	5147
7	Task 7	0	5120	2120
8	Task 8	1	5660	3515
9	Task 9	1	6339	1075
10	Task 10	0	6486	1862
11	Task 11	0	6504	598
12	Task 12	0	6676	1923
13	Task 13	1	8087	2235
14	Task 14	0	8208	180
15	Task 15	1	8777	385
16	Task 16	0	8983	823
17	Task 17	0	9225	2638
18	Task 18	0	9402	90
19	Task 19	1	9599	595
20	Task 20	1	15439	9782

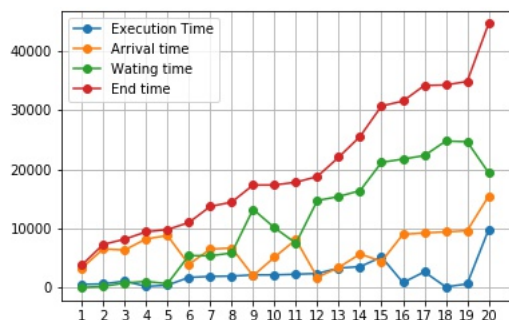


FIGURE 4. EDF algorithm execution results with enough deadlines.



FIGURE 5. IEDF algorithm execution results with enough deadlines.

static priority, 1 for Class I (high) priority and 0 for Class II (low) priority. Each test runs the same task, and the task set execution time T_i is randomly generated during the test time. The relative deadline D_i is 4 times of the task execution time, which is carried out in an uncertain environment. For the purpose of illustration, the time scale has been expanded by 10,000 times and the new time unit is seconds.

As shown in Figure 3 to Figure 8 and Table 2, when tasks have enough deadline, the comparison among the

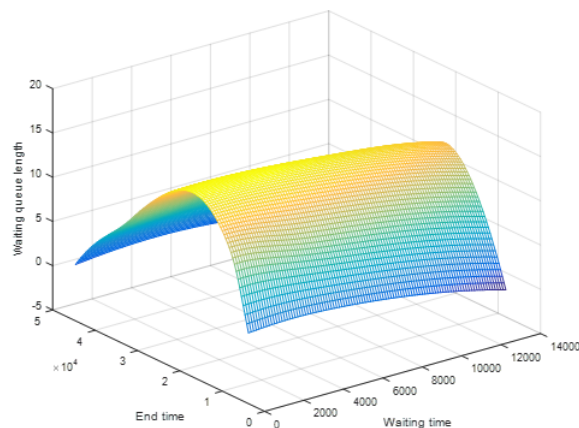


FIGURE 6. Execution results of ordinary queuing algorithm with enough deadlines.

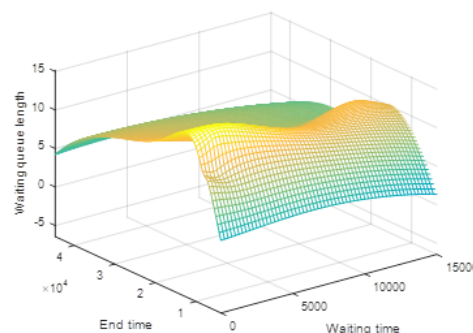


FIGURE 7. Execution results of EDF algorithm with enough deadlines.

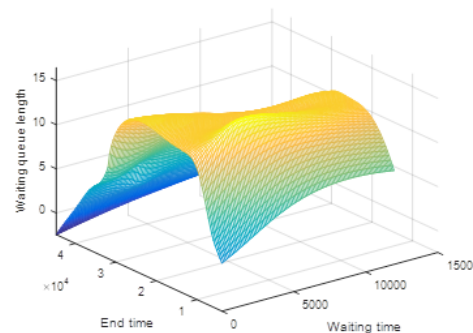


FIGURE 8. Execution results of IEDF algorithm with enough deadlines.

execution results of the ordinary queuing, EDF and IEDF algorithm can lead to conclusions: the total waiting time of IEDF algorithm with enough deadline is much less than that of ordinary queuing algorithm and EDF algorithm; in the IEDF algorithm, tasks with high static priority will be executed first, so high-priority tasks will preempt low-priority tasks. In the ready queue of the same priority task, consider the deadline, when a task waiting time is close to the task deadline, this task will be executed first. So, in low-priority tasks, the shorter task will be executed first.

TABLE 2. Comparison of execution results of three algorithms with enough deadlines.

Algorithm	Total waiting time	End time	Number of unfinished tasks
Ordinary queuing algorithm	134269	44634	0
EDF algorithm	230454	44634	0
IEDF algorithm	101902	44634	0

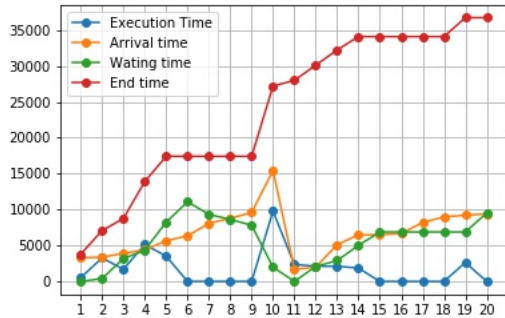


FIGURE 9. Ordinary queuing algorithm execution results with deadlines (deadline is 4 times of the task execution time).

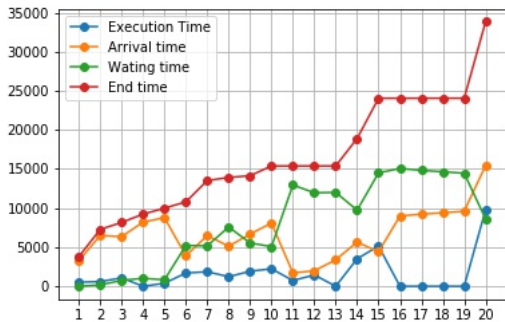


FIGURE 10. EDF algorithm execution results with deadlines (deadline is 4 times of the task execution time).

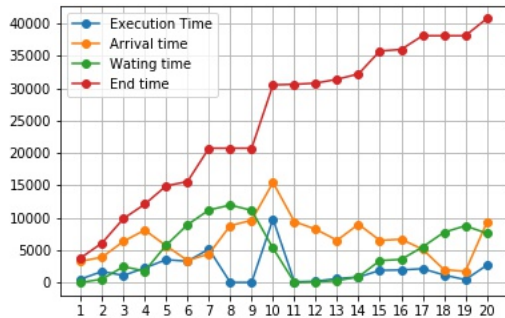


FIGURE 11. IEDF algorithm execution results with deadlines (deadline is 4 times of the task execution time).

As shown in Figure 9 to Figure 15 and Table 3, when tasks have deadlines, the comparison among the execution results of the ordinary queuing algorithm, EDF algorithm and IEDF

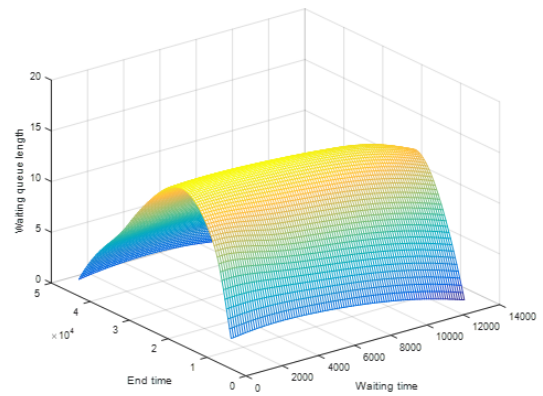


FIGURE 12. Execution result of ordinary queuing algorithm with deadlines (deadline is 4 times of the task execution time).

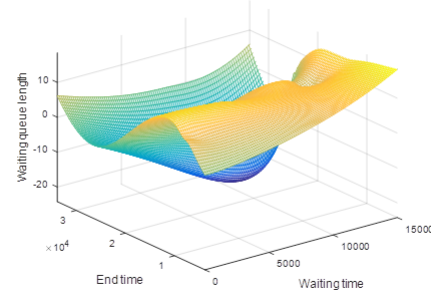


FIGURE 13. Execution result of EDF algorithm with deadlines (deadline is 4 times of the task execution time).

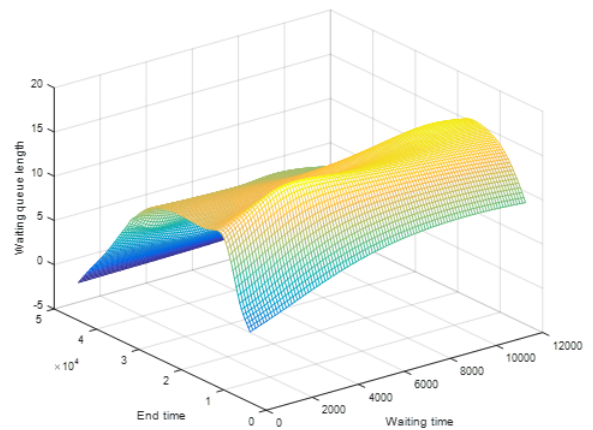
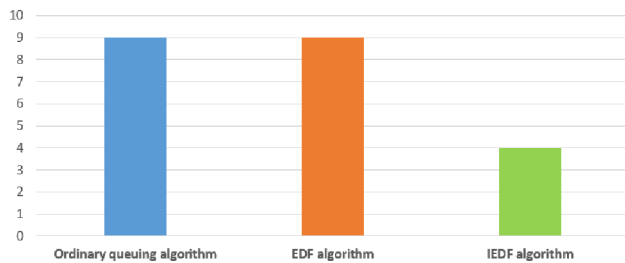


FIGURE 14. Execution result of IEDF algorithm with deadlines (deadline is 4 times of the task execution time).

algorithm can lead to conclusions: the IEDF algorithm makes far fewer errors than the ordinary queuing algorithm and EDF algorithm. When a task goes wrong, it means that the task is not executed, and it happens in this case: tasks with high static priority will be executed first, so high-priority tasks will preempt low-priority tasks. In the ready queue of the same priority tasks, the deadline is taken into consideration. In the ordinary queuing algorithm with deadlines, if the task waiting time of exceeds its deadline, the task will be discarded; in

TABLE 3. Comparison of execution results of three algorithms with deadlines (deadline is 4 times of the task execution time).

Algorithm	Total waiting time	End time	Number of unfinished tasks
Ordinary queuing algorithm	35066	36730	9
EDF algorithm	55616	33841	9
IEDF algorithm	57087	40749	4

**FIGURE 15. Comparison of the error numbers in three algorithms.**

the IEDF algorithm, when the task's waiting time is close to the deadline, the task will be executed first, and if the task's waiting time exceeds the deadline, the task will be discarded.

In the experiments, the IEDF algorithm is compared with the ordinary queuing algorithm and EDF algorithm in two circumstances: deadline is long enough and the deadline is four times of task execution time. The comparison includes end time, total wait time and the error numbers (the total number of tasks which are discarded before they are completed). The conclusions are obtained through comparison of simulation experiments: (1) The total waiting time during the execution of the IEDF algorithm with long enough deadlines is much shorter than the ordinary queuing and EDF algorithms; (2) The number of errors in the IEDF algorithm with deadlines is far less than the ordinary queuing and EDF algorithms. Therefore, the IEDF algorithm proposed in this paper can effectively reduce the waiting time and the number of errors, thereby reducing the number of abandoned tasks, and better ensuring the real-time performance of the system.

VI. CONCLUSION

In this paper, we present the IEDF scheduling algorithm, which is more suitable for the scheduling of real-time embedded system. Then, we provide the model description of IEDF scheduling algorithm. The IEDF scheduling algorithm is based on the EDF (Earliest Deadline First) algorithm. But the IEDF algorithm can be applied to non-periodic and randomness tasks, it is more universal than the universality of EDF algorithm. Finally, in order to analyze the relationship of various time performance indicators during the process of task execution in real-time embedded system, we add

the queuing theory into the IEDF scheduling algorithm. The experiment shows that the IEDF scheduling algorithm can effectively reduce the waiting time and the number of errors and better ensure the real-time performance of the system. These conclusions will have some reference value for tasks scheduling for embedded systems software. However, the IEDF algorithm running on a periodic server remains to be further studied.

REFERENCES

- [1] R. I. Davis, S. Altmeyer, L. S. Indrusiak, C. Maiza, V. Nelis, and J. Reineke, "An extensible framework for multicore response time analysis," *Real-Time Syst.*, vol. 54, no. 3, pp. 607–661, Jul. 2018.
- [2] X. Liu, X. Chen, and F. Kong, "Utilization control and optimization of real-time embedded systems," *FNT Electron. Design Autom.*, vol. 9, no. 3, pp. 211–307, 2015.
- [3] W. Bo, B. X. Ying, and C. W. Guang, "Temporal defect detection of embedded software using timed execution trace," *Chin. J. Comput.*, vol. 40, no. 12, pp. 3–25, Dec. 2017.
- [4] C.-W. Chang, J.-J. Chen, T.-W. Kuo, and H. Falk, "Real-time task scheduling on island-based multi-core platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 2, pp. 538–550, Feb. 2015.
- [5] X.-F. Meng and X.-Y. Zhang, "Parallel task scheduling strategy with multi-objective constraints in P2P," *Comput. Integr. Manuf. Syst.*, vol. 14, no. 4, pp. 761–766, 2008.
- [6] Z. Qingbing, L. Yu, and H. Ming, "Method for evaluating reliability of cyber-physical systems online based on machine learning," *Comput. Eng. Appl.*, vol. 50, no. 10, pp. 128–130, 2014.
- [7] A. De Matos Pedro, D. Pereira, L. M. Pinho, and J. S. Pinto, "Logic-based schedulability analysis for compositional hard real-time embedded systems," *SIGBED Rev.*, vol. 12, no. 1, pp. 56–64, Mar. 2015.
- [8] N. C. Kumar, S. Vyas, R. K. Cytron, C. D. Gill, J. Zambreno, and P. H. Jones, "Hardware-software architecture for priority queue management in real-time and embedded systems," *Int. J. Embedded Syst.*, vol. 6, no. 4, p. 319, 2014.
- [9] V. A. Fajardo and S. Dreke, "Waiting time distributions in the preemptive accumulating priority queue," *Methodol. Comput. Appl. Probab.*, vol. 19, no. 1, pp. 255–284, Mar. 2017.
- [10] V. Muliukha, A. Ilyashenko, O. Zayats, and V. Zaborovsky, "Preemptive queueing system with randomized push-out mechanism," *Commun. Nonlinear Sci. Numer. Simul.*, vol. 21, nos. 1–3, pp. 147–158, Apr. 2015.
- [11] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.
- [12] T. Park and S. Kim, "Dynamic scheduling algorithm and its schedulability analysis for certifiable dual-criticality systems," in *Proc. 9th ACM Int. Conf. Embedded Softw. (EMSOFT)*, 2011, pp. 253–262.
- [13] M. Mahdiani and A. Masrur, "On bounding execution demand under mixed-criticality EDF," in *Proc. 26th Int. Conf. Real-Time Netw. Syst. (RTNS)*, 2018, pp. 170–179.
- [14] L. Santinelli, D. Doose, G. Durrieu, F. Boniol, C. Lesire-Cabaniols, and C. Grand, "Schedulability analysis for mixed critical cyber physical systems," in *Proc. IEEE Ind. Cyber-Phys. Syst. (ICPS)*, May 2018, pp. 297–303.
- [15] A. Burns and R. Davis, "Mixed criticality system—A review," 12th ed. Dept. Comput. Sci., Univ. York, U.K., Tech. Rep., Mar. 2019.
- [16] S. Ting-na, C. Zheng-wei, and F. Hong-wei, "Extension EDF fuzzy scheduling for tasks with uncertain characteristics in networked control system," *J. Tianjin Univ.*, vol. 8, pp. 690–694, Aug. 2011.
- [17] D. Strzeżewski and W. M. Zuber, "Modeling and performance analysis of priority queueing systems," in *Proc. Comput. Sci. On-Line Conf.*, 2018, pp. 302–310.
- [18] P. Jayarajan, R. Maheswar, V. Sivasankaran, D. Vigneswaran, and R. Udaiyakumar, "Performance analysis of contention based priority queueing model using N-policy model for cluster based sensor networks," in *Proc. Int. Conf. Commun. Signal Process. (ICCCSP)*, Apr. 2018, pp. 229–233.
- [19] C. Kai, W. Jie, and Z. Kuanjiu, "Reliability of interrupt services for embedded systems," *J. Tsinghua Univ. (Sci. Technol.)*, vol. 56, no. 8, pp. 878–884, 2016.



YINGJIE WANG was born in 1977. She received the M.E. degree from Yanshan University, China, in 2004. She is currently pursuing the Ph.D. degree with the School of Software, Dalian University of Technology. She is currently an Associate Professor with the College of Information Engineering, Dalian University. Her current research interests include software engineering and trustworthy software.



NAN CHEN was born in 1998. He is currently pursuing the bachelor's degree in computer science and technology with the School of Information Engineering, Dalian University.



KUANJIU ZHOU was born in 1966. He received the B.E. and M.E. degrees in computer software and the Ph.D. degree in management engineering from the Harbin Institute of Technology. He is currently a Professor with the Software School, Dalian University of Technology. His current research interests include formal methods, software engineering, and trustworthy software.



ZUMIN WANG was born in 1975. He received the M.E. degree in mechanical manufacturing and automation from the North University of China, in 2004, and the Ph.D. degree in physical electronics from the Chinese Academy of Sciences, in 2007. He is currently a Professor with Dalian University, since 2014. His current research interests include wireless sensor networks, the Internet of Things, and smart city.

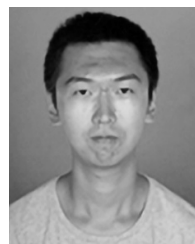


BIN LI was born in 1996. He received the bachelor's degree from Dalian University, China, in 2019. He is currently pursuing the master's degree with the School of Software Engineering, University of Science and Technology.



MINGCHU LI was born in 1963. He received the B.S. degree in mathematics from Jiangxi Normal University, in 1983, the M.S. degree in applied science from the University of Science and Technology, Beijing, in 1989, and the Ph.D. degree in mathematics from the University of Toronto, in 1997. He was an Associate Professor with the University of Science and Technology, Beijing, from 1989 to 1994. He was involved in research and the development of information security with

Longview Solution Inc., and Compuware Inc., from 1997 to 2002. Since 2002, he has been a Full Professor with the School of Software, Tianjin University. He has been with the School of Software Technology, Dalian University of Technology, as a Full Professor, a Ph.D. Supervisor, and the Vice Dean. His main research interests include theoretical computer science and cryptography. His other research interests include graph theory, network security, and game theory.



HONGXUAN TIAN was born in 1998. He is currently pursuing the bachelor's degree in computer science and technology with the School of Information Engineering, Dalian University.

...