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.1.10

Scheduling Algorithm for Data Flow Model in Real-time Control System

Chenggang Qin[1+], Dong Yu[1], Wenjiang Wu[2], and Xiaozhang Lu[2]

[1]Shenyang Institute of Computing Technology, Graduate University of Chinese Academy of Science, Beijing, China

[2]Shenyang Golding NC Technology CO., Ltd Chinese Academy of Science Shenyang, Liaoning Province, China

Abstract. In many real-time control systems, data flow is the base structure of software model. The QoC (Quality of Control) of some systems not only relate to whether the temporal constraints of real-time tasks can be met, but also relate to whether the data flow is non-suspending. While the data flow is suspending, both the control precision and running speed would degrade. For preventing the data flow suspending, we introduce a real-time scheduling algorithm that considers the risk of data flow suspending. We also present an implementation framework for real-time scheduling algorithm based on RTAI. It can simplify the implementation and verification of real-time scheduling algorithm. The results of verification manifest the DFO-RSA algorithm can prevent the data flow suspending effectively while guarantee the temporal constraints of critical tasks.

Keyword*:* Real time control system, real-time scheduling algorithm, data-flow, real-time operating system

1. Introduction

In a real-time control system, data flow is a wildly used software model. Control commands are processed layer by layer along the data flow, and control signal is outputted at the end of data flow. The tasks in the data flow can be divided into 2 categories: control tasks and service tasks. The control tasks are responsible for executing control algorithm. Other tasks provide data to control tasks, and they can be called service tasks. These tasks in the data flow exchange data through share buffers, so the data flow will suspend while some buffers are underflow.

Normally, the QoC of real-time control system would not be influenced as long as the control task is not in data hungry status. But, in some control systems, service tasks' data hungry will degrade the QoC of system. Like the control precision, the running speed is also an important part of QoC in some control systems. For example, the high-speed high-precision CNC (Computer Numerical Control) system needs the machining process as quickly as possible. Service tasks in the CNC system will get the optimal performance only if the data flow is not suspended. For guaranteeing machining precision, the movement of motors will be stopped while some service tasks are data hungry. So, the machining speed will be reduced. Furthermore, to start and stop the motors frequently will increase the control error also. In conclusion, data flow suspending will degrade the QoC of real-time control system directly. For these real-time control systems, the objective of real-time scheduling should be to guarantee the temporal constraints of control tasks and to keep the data flow non-suspending.

The off-the-shelf real-time operating system schedules the tasks in data flow only according to their allocated attributes, such as period or deadline. In a static real-time control system, the scheduling attributes

[+] Corresponding author.
E-mail address: qinchenggang@sict.ac.cn.

which are elaborately assigned can guarantee the data flow is non-suspended. But, the designer cannot predict accurately the behavior of a complex dynamic control system. In these systems, the constrained conditions may be violated at the run time. For example, the task's overtime (execution time exceeds WCET), the execution of sporadic tasks or error-tolerate tasks may lead to the overloading of system. While the system is overloading, the temporal constrains of some tasks will no longer be guaranteed. If these tasks cannot be executed for several periods, there is a dangerous of data flow suspending. Even if the workload is normal, the abrupt increasing of the data consume rate of some tasks may lead to the data flow suspending too. The preventing of data flow suspending can increase the robustness of real-time control system, and make up for the fragile of static analysis.

A new real-time scheduling algorithm DFO-RSA is introduced in this paper. The algorithm extends the traditional static priority real-time scheduling algorithm with the evaluation for the risk of data flow suspending. It can prevent the data flow suspending while guarantee the control tasks' temporal constraints. An implementation framework of real-time scheduling algorithm is introduced in this paper too. The framework simplifies the implementation and verification process of real-time scheduling algorithm in the off-the-shelf real-time operating system.

The rest of the paper is organized as follows. Section 2 simply introduces the related work of our topic. In section 3, we define the model of data flow. Section 3 presents our real-time scheduling policy. In Section 4, the framework of scheduling which support our scheduling policy is described. Section 6 introduces an implementation framework of real-time scheduling algorithm. In section 7, we evaluate the performance of our scheduling algorithm. The paper concludes with a summary in Section 8.

2. Related Works

Early researches on data flow was concerned with the prevention of deadlock [1], the minimization of buffer size [2], and the maximization of throughput of data flow [3]. These studies are mainly in the areas of signal processing and multimedia. In these areas, the data production rate and data consume rate of buffers varies greatly. Internet is a link of the data flow in these applications, transmission speed of Internet is unstable. It is very hard to maintain smooth for data flow. But the methods that have been introduced are not suitable for control system. The topology of control systems is relatively simple. Most control systems only have a single computer. There is no Internet in the data flow of these systems. Even if a control system has multiply nodes, the connection media would be a fieldbus that has a relatively fixed transmission delay. Therefore, the static analysis for the data flow in a control system is relatively easy. The static analysis can guarantee the data flow would not suspend in a normal status. The main problem in a control system is keeping the data flow would not be suspended in the abnormal status, such as the system is overloading.

The dynamic feedback scheduling algorithm adjusts the scheduling parameters according to the system state. It can improve the dependability of the system [4]. [5-8] introduced the methods to use the feedback scheduling algorithms in a control system, and improved the theory of the co-design of control and real-time scheduling. [9-10] apply the modern robust control theory to scheduling algorithm, improved the system's dependability and stability. For the graceful degradation while the system is overloading, many feedback scheduling algorithm will decrease the execution rate of some tasks. But the method is not suitable for the real-time control system. While a buffer will underflow immediately, decreasing the execution rate would promote the suspending of data flow.

3. The Model of Data Flow

3.1 The real-time control system based on a data flow

Following the development of hardware and real-time operating system, the implementation of real-time control system has been changed from specified hardware platform to the off-the-shelf hardware. The method that uses the general purpose hardware platform, such as PC, and special software improve the openness and decrease the cost of the real-time control system. The data flow model shows in Figure 1 is a typical example of the software model of the real-time control system. Task 4 locates at the end of data flow, is the control task. Other tasks are service tasks. The control task computes the control signal according to the input data provided by task3 and the sampling data gathered from sensors. The control signal is sent to plants. In a CNC system, service tasks include interpreter, the interpolation task, the Acc/Dec control task et al. Control task is the position control task that execute the PID algorithm.

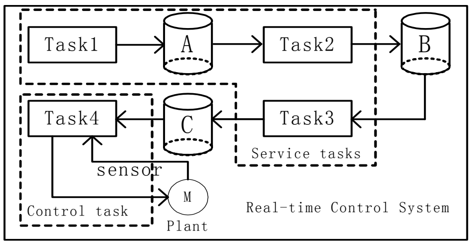

Fig. 1: The data flow structure in a real-time control system.

The data flow model connected by share buffers decreases the coupling between the concurrency tasks. The tasks only enter a critical section while they need access a share buffer. The model simplifies the software design and decreases the difficulty of implementation of control system. The simplicity of data flow model decreases the potential faults in the software, and improve the dependability of whole system. But the data flow model has the dangerous of suspending. Although some programming skill can avoid the potential hazard, but can not avoid the degradation of QoC while the data flow is suspending. For example, the data hungry of interpolation task in the CNC system would break the machining speed planning, the motors must decelerate to zero and accelerate soon. The process extends the time of machining, the motor's acceleration and deceleration lead to the jitter of following error and the jitter increase the machine error. The data hungry of position control task would make the servo cannot get the new position, the machining process must be stopped. If the data hungry of position control task is very frequency, the motor would be joggling. The similar problems exist in many real-time control systems. So, the prevention of data flow suspending is the widespread requirement of many real-time control systems.

3.2 The risk assessment model of buffer underflow

The reason of data flow suspending is the underflow of buffers. For preventing the data flow suspending, we need assess the risk of buffer underflow. While the risk of buffer underflow is high, the producer of this buffer should be scheduled preferentially. The amount of data can be used as parameters for risk assessment of the buffer underflow.

We use the integer *R* in the interval [0, M] represents the risk of a buffer's underflow. While the data level is high, there is no danger, i.e. *R=0*. While the amount of data below a certain level, the value of *R* should be increased. Following the gradual reduction of the amount of data, the acceleration of risk increased become larger and larger. If the amount of data equals zero, the risk will reach the maximum, *M*. We can use following quadratic function to describe how the risk of underflow changes.

$$R(x) = \left\lceil \frac{1}{2}kx^2 - (\frac{1}{4}kB + \frac{2M}{B})x + M \right\rceil \qquad k \le \frac{8M}{\mathrm{B}^2} \tag{1}$$

x is the data amount of a buffer, *B* is the size of buffer, *M* is the upper limit of risk. The acceleration of risk changes can be tuned by constant *k*. The k is greater, the acceleration is greater. As the Figure 2 shows, R equals 0 while the data amount equals a half of buffer size. While the data amount equals 0, R equals M.

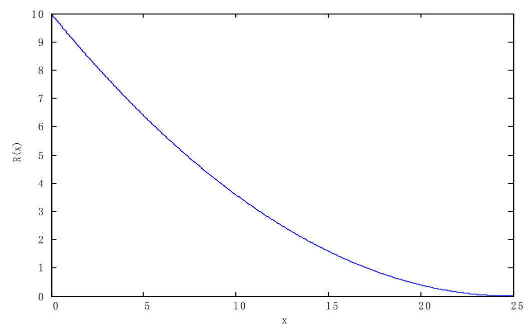

Fig. 2：The risk with data amount changes. B=50, M=10, k=0.32

4. The Real-Time Scheduling Policy Based on the Risk Assessment of Buffer Underflow

As mentioned above, the tasks in a real-time control system can be divided into control tasks and service tasks. Control task is hard real-time task; the QoC (Quality of Control) will degrade if the temporal constraints of the task are lost. Service tasks are implemented as hard real-time tasks usually, for guaranteeing the buffer does not underflow under the period and priority which is given in the static analysis. However, the QoC would not be degradation even if the temporal constraints of service tasks are violated, while the data level of buffer is high. So, service tasks are hard real-time tasks only at special situations. This phenomenon shows the priority of service tasks can be changed while the risk of data flow suspending is high.

4.1 The priority allocation of scheduling algorithm

The temporal constraints of control tasks are intrinsic requirements of control algorithm, they must be guaranteed strictly. So, we should avoid that the control tasks are affected by other tasks. For the reasons, the priority in our scheduling framework is divided into 2 classes: the priority of control tasks and the priority of other tasks. The maximum priority of service tasks is lower than minimum priority of control tasks. Because the producers of buffers in the data flow are service tasks, the scheduling policy is introduced in this paper only oriented to the service tasks. The priority of control tasks will not be changed.

The priority interval of service tasks is $[P^{\mathrm{Min}}, P^{Max}]$. The interval can be further divided into static priority and dynamic priority. The static priority is assigned at the design phase according to the importance of tasks. Along the data flow, the importance of every task is larger and larger. The priority of the task that locates at the top end of data flow is minimal; and the priority of the task that locates at the low end of data flow is maximal. The dynamic priority maps the risk of buffer underflow into scheduling policy. The interval of dynamic priority is $(0, M]$. The interval of static priority is $[P^{Min}, P^{Max} - M]$. The priority that will be used for selecting the running task is composed of static priority P^s and dynamic priority P^d together. The priorities that greater than P^{Max} are the priorities for control tasks, and the priorities that lower than P^{Min} are the priorities for the non-real-time tasks.

4.2 The allocation of dynamic priority

The dynamic priority introduced the risk of data flow suspending to the real-time scheduling policy. The result of formula 1 can be used as dynamic priority directly. While the buffer is underflow already, the importance of producer task of the buffer is more and more importance following time. If the producer task cannot be executed for a long time, the data flow is suspending always. So, the priority of the producer should be increased while the buffer is underflow. Summary, the allocation of dynamic priority should abide the following rules:

(1) While the buffer is not underflow, the dynamic priority of producer task equals the risk of buffer underflow, *R(x)*. *x* is the data amount of buffer.

(2) If the buffer is underflow, and the dynamic priority of the producer task is lower than *M*, the priority of producer will accumulate each consumer task's period.

(3) While the buffer is not empty, and the dynamic priority of producer task is larger than *M*, the dynamic priority of producer tasks should be reset to the risk of buffer underflow, *R(x)*.

4.3 Scheduling policy

The execution sequence of tasks in the system can be changed by the tasks' dynamic priority. The dynamic priority can be added on static priority as compensation. The priority of a task can be defined as:

$$P_i = P_i^s + \lambda P_i^d \tag{2}$$

P_i is the scheduling priority. The priority should be used by scheduling algorithm to select a task to execute. λ is between [0,1], it can be used as the weight that reflect the degree of the buffer's underflow impact QoC, the buffer is below the task i. If the task that locates at the downstream of task *i* is not sensitive to data hungry, the weight λ of task i should be assigned a lower value. Otherwise, the weight λ of task i should be assigned a higher value. For CNC system, the position control task that locates at the lower end of data flow is most sensitive on the data hungry. The machine process will suspend once this task cannot get data. So the task that locates at the upstream of position control task should be assigned the weight of 1.

5. The Framework of Scheduling

For the algorithm DFO-RSA, the priority of tasks is related to the data amount of the buffer below the task. The scheduler must monitor the data amount of every buffer. We design a scheduling framework for DFO-RSA. As figure 3 shows, the scheduling framework includes monitor, controller, and scheduler. The monitor monitors the data amount of every buffer, and sends the data amount vector to controller. Controller computes the risk and dynamic priority of every task according to the data amount of every buffer. Then the controller computes the scheduling priority of every task using the formula 2. The tasks are sorted in the ready queue by their scheduling priority. If some tasks' priority changed, the controller should adjust the position of tasks in the ready queue. The monitor and controller can be implemented as a single real-time task.

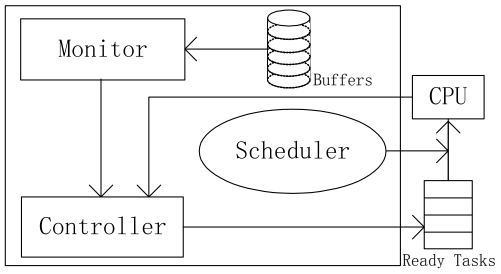

Fig. 3: The scheduling framework of DFO-RSA

If the consumer task is in the data hungry status, it should notify the controller. The controller increases the dynamic priority of the producer task of the relevant buffer according to the rules that is presented in section 4.2.

6. Design and Implementation

The implementation and verification of real-time scheduling algorithm are very difficult. If the real-time scheduling algorithm would be implemented in a real-time operating system, the code of the operating system must be modified. The modification may destroy the compatibility and stability of the operating system. The debugging of the operating system kernel is a trouble work too. The emulational environment, such as TrueTime, RTSIM, et al, can not simulate a real environment. The randomness of scheduling delay, context switching time, interrupt latency, interrupt process latency, execution time of task, and the latency of network communication is very hard to simulate in emulational tools. To simplify the implementation and verification of real-time scheduling algorithm in the true real-time operating system, we design an open verification framework for real-time scheduling algorithm on the RTAI (Real-time Application Interface). The user's scheduling algorithm can be easily implemented using this framework.

RTAI is the real-time extension of Linux, the core functions are implemented using the Linux kernel module. To avoid the modifying of operating system's kernel, we abstract the real-time scheduling policy, and the user can implement his real-time scheduling policy by the kernel module. While the module is inserted into kernel, RTAI will switch to the scheduling policy that defined in the module.

6.1 Structure of the real-time scheduling algorithm framework

The essence of scheduling is switching to the task that has the maximum priority. The essence of scheduling policy is to assign appropriate priority for tasks. RTAI has already supported static scheduling algorithm and EDF (Earliest Deadline First) scheduling algorithm. For the static scheduling algorithm, the priority of every task is assigned by the engineer at the design phase, and the priority is no longer to change at the run time. For the EDF scheduling algorithm, the priority of task equals its deadline. The priority is changing at the run time. The scheduling algorithm of RTAI is implemented in the schedule function rt_schedule() and other similar functions. The extendibility and maintainability of code is very low.

For enhancing the extendibility and maintainability of real-time scheduling algorithm code, we abstract the scheduling policy to a data structure. The data structure includes data and operations that involved scheduling policy using the thinking of object orient programming. The scheduling policy can be implemented as a scheduling policy object: SCHED_OBJ. This object is an instance of the data structure that is introduced above. The schedule function must be modified for using the scheduling policy object. The structure of scheduler framework with scheduling policy object is shown in figure 4.

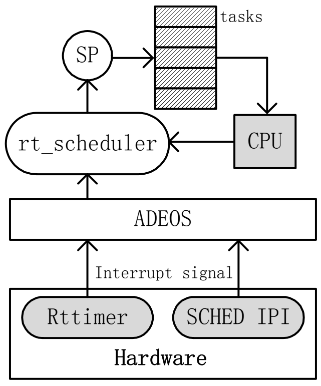

Fig. 4. The structure of scheduling framework

The operating system may contain multiply scheduling policies; every scheduling policy is a scheduling policy object. A global pointer SP points the current scheduling policy.

User can implement his scheduling policy using the Linux kernel module. Once the module is inserted into the kernel, the global pointer SP points to the scheduling policy object automatically. Some common scheduling algorithms should be built in the operating system, such as RM (Rate monotonic) and EDF.

6.2 Scheduling policy object

The implementation of a scheduling policy is the implementation of a SCHED_OBJ. SCHED_OBJ uses the thinking of object oriented programming. In the area of kernel programming, the object oriented programming languages, such as C++, are not popular. We use the mechanism of C to simulate the object oriented programming language. The data structure of C does not support member functions. The function pointer can be used to simulate it. These function pointers should be assigned while define the SCHED_OBJ. The data structure of scheduling policy is :

```
typedef struct schedule_policy{
    unsigned long sched_lock; //The lock to protect SCHED_OBJ;
    unsigned char task_attr_is_extended; //Task's attributions that extended by SCHED_OBJ;
    unsigned char prio_is_online; //Whether update the priority in the scheduler function;
    RTTIME timer_period; //The period of hardware clock;
    int oneshot_timer; //Whether to use oneshot mode;
    … …
    int (*SCHED_OBJ_prio_update)();
    int (*SCHED_OBJ_initialization)(unsigned char prio_is_online, unsigned char task_attr_is_extended,
unsigned char policy_for_same_prio, int oneshot_timer, RTTIME period);    //The initialization function;
    int (*SCHED_OBJ_task_attr_extend)();
    … …
}SCHED_OBJ;
```

The task's attributions may need to expend. For example, the DFO-RSA algorithm needs every task's static priority and dynamic priority. But, the TCB (task control block) of RTAI has no these fields. For the sake of enhancing our framework's extendibility, a mechanism for extending the TCB must be provided. We add a void pointer in the TCB. A data structure RT_TASK_EXTEND that can be used for extending TCB is provided for user. Users can extend their attributions in this data structure. The default initialization function of SCHED_OBJ will allocate memory of this data structure for every task. The start of the memory will be assigned to the task's t_extend. User can use the customizing attribution my_attr in the way: task->t_extend->my_attr.

The most important operation of scheduling policy object is computing of every task's priority. The function SCHED_OBJ_prio_update() of SCHED_OBJ can be used to compute every task's priority. The policy of priority computing can be defined in this function. The function can be called in the schedule function, and user can call this function at custom position. For DFO-RSA, at the interval of two successive execution of schedule function, only one data is inserted to or deleted from one buffer. So, there is no need to call SCHED_OBJ_prio_update() in the schedule function. For Least-Slack-Time-First (LSTF) algorithm, the priority of all tasks should be update while the schedule function is called. The field prio_is_online of SCHED OBJ used to identify whether to update task's priority in the schedule function. If prio_is_online equals 1, the schedule function will call SCHED_OBJ_prio_update() every execution. Otherwise, SCHED_OBJ_prio_update() should be called by user. Such as, user can call this function in a real-time task which has a high priority.

SCHED_OBJ_initialization() is a default initialization function for a SCHED_OBJ. The function initializes the parameters of a schedule policy object, according to the information provided by user. If TCB is extended by user, the function will allocate a memory space for the extended fields, and assign the address of the memory to the t_extend of the TCB.

6.3 Implementation of DFO-RSA

The controller in the scheduling framework of DFO-RSA updates the tasks' dynamic priority and schedule priority. So, the function SCHED_OBJ_prio_update() can be called by the controller. Because the TCB in RTAI does not include dynamic priority and static priority, we must extend the TCB, i.e. define a data structure whose name is RT_TASK_EXTEND.

```
struct RT_TASK_EXTEND {
    double dynamic_priority;
    double static_priority;
    double factor; //λ
};
```

The priority field in the original TCB can be used as scheduling priority. For DO_SPO, the SCHED_OBJ of DFO-RSA, prio_is_online should be set to 0, because the SCHED_OBJ_prio_update() need not to be called by the scheduling function at its each execution, but should be called by controller. The controller is implemented as a periodic real-time task that has the highest priority. The task_attr_is_extended field should be set to 1. The function SCHED_OBJ_prio_update() of DO_SPO updates the dynamic priority of every tasks using formula 1 and rule 1, updates the schedule priority of every task using formula 2. In the ready queue, the tasks are ordered according their schedule priority. So, while a task's schedule priority is changed, its position in the ready queue should be arranged again. The time complexity of this operation is high. But the controller's execution rate is low, and the number of tasks in a control system is low, the overhead of this operation is acceptable.

For implementation of DFO-RSA, the buffer model should be defined as :

```
struct Buffer{
    long size;
    long level;
    RT_TASK *producer; // Producer task of the buffer;
};
```

The function SCHED_OBJ_prio_update() is implemented as:
The elements of array B are the buffers in the data flow.

```
SCHED_OBJ_prio_update()
 {
    double old_priority;
    for(i=0;i<N;i++) //N is the number of buffers in the data flow;
     {
        old_priority = B[i].producer->priority;
        if(B[i].size == 0) //Buffer i is underflow;
            B[i].producer->t_extend->dynamic_priority ++;
        else
```

```
                B[i].producer->t_extend->dynamic_priority = R(B[i].level);
            B[i].producer->priority = B[i].producer->t_extend->static_priority + B[i].producer->t_extend->factor * B[i].producer->t_extend->dynamic_priority; //compute the schedule priority of buffer i's producer task;
            if((B[i]. producer->state || TASK_READY) && (B[i].producer->priority != old_priority)) //The producer task of buffer i is ready;
            { //Arrange the producer task's position in the ready queue.
                rm_from_ready_queue(B[i]. producer);
                en_to_ready_queue(B[i]. producer);
            }
        }
    }
```

6.4 The implementation of RM and EDF

RM real-time scheduling algorithm is a static priority algorithm. The priority of task is inversely proportional to the task's period. The task whose execution rate is highest has the highest priority. In the schedule policy object of RM algorithm RM_SPO, the TCB need not to be extended. While the RM_SPO is initialized, the function SCHED_OBJ_prio_update () should be called once, and the reciprocal of task's period is assigned to the task's priority in this function. The field prio_is_online of RM_SPO should be set to 0.

EDF is a dynamic priority algorithm. Task's priority equals its deadline. In the schedule policy object of EDF algorithm EDF_SPO, the TCB need not to be extended too. Every task's priority should be updated with the task's deadline in the function EDF_SPO. SCHED_OBJ_prio_update (). The function EDF_SPO. SCHED_OBJ_prio_update () should be called while the scheduling function executes. The field prio_is_online of EDF_SPO should be set to 1.

7. Performance Evaluation

For verifying the effectiveness and evaluating the performance of DFO-RSA, we implement the DFO-SPO in a software CNC system []. The basic software architecture of CNC system is similar to the data flow that is shown in figure 1. There are total 4 tasks and 3 buffers in the data flow. All of the 4 tasks are periodic tasks. The Buffer A's capacity is 100, Buffer B's is 50, Buffer C's is 10. The attribution of every task is shown in the table 1.

Table 1. The attribution of the tasks in the data flow of CNC

Tasks	Priority	Period	WCET
Task1	3	5ms	0.452ms
Task2	2	1ms	0.161ms
Task3	1	1ms	0.073ms
Task4	0	0.125ms	0.017ms

CNC system is a typical control system, and it is a highly dynamic system. The variation of the tasks' execution time, the execution of sporadic tasks would change the data production or consumption rate of the buffers. Furthermore, because the CNC system is complex, the precise model of software's behavior and system load is very difficult to obtain. In CNC system, the suspending of data flow is frequent. The suspending would degrade the machining quality and machining velocity.

The overflow/underflow times of every buffer before and after enable DFO-RSA are compared. Table 2 shows the overflow/underflow times of three buffers during the total machining process.

Table 2. The overflow/underflow times of every buffer

	DFO-RSA is disabled		DFO-RSA is enabled	
Buffers	underflow	overflow	underflow	overflow
A	16519	154	1276	12
B	8523	107	1359	17
C	7964	22	2713	8

According to the data of table 2, DFO-RSA algorithm restrained the suspending of data flow while the real-time control system is running, and improved the quality of control.

8. Conclusion

In this paper, a real-time scheduling algorithm, DFO-RSA, is introduced. The DFO-RSA algorithm can prevent the data flow suspending while guarantee the temporal constraints of critical tasks. Because the data flow is restrained effectively by the algorithm, the QoC of real-time control system is improved. The DFO-RSA algorithm enhances the robustness of real-time control system too.

An implementation framework for real-time scheduling algorithm is presented in this paper too. This framework use the thinking of object oriented to simplify the implementation and verification of real-time scheduling algorithm. The user can implement their custom algorithm by the Linux kernel model. After the module is inserted into the kernel, the scheduling algorithm is changed to user's custom algorithm automatically. Some examples of the framework are introduced, such as the implementation of DFO-RSA, RM and EDF.

The effectiveness of DFO-RSA algorithm is tested in a CNC system. CNC system is a typical real-time control system; and the data flow is the basic software structure in the system. The results manifest DFO-RSA algorithm can restrain the data flow suspending effectively, and improve the QoC of CNC system.

9. Acknoledgment

This work is supported by the Important National Science & Technology Specific Projects Concerning High-end CNC Machine Tools and Basic Manufacturing Equipment (2009ZX04009-013).

10. References

[1] Jongsoo Park, William J. Dally, "Buffer-space Efficient and Deadlock-free Scheduling of Stream Applications on Multi-core Architectures" Proceedings of the 22nd ACM symposium on Parallelism in algorithms and architectures, Greece, June 2010.

[2] Weichen Liu, et al. An efficient technique for analysis of minimal buffer requirements of synchronous dataflow graphs with model checking. Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis, France, October, 2009.

[3] A.H. Ghamarian, et al. Throughput Analysis of Synchronous Data Flow Graphs. In ACSD' 06 6th International Conference on Application of Concurrency to System Design, Finland, June 2006.

[4] Chenyang Lu, et al. Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms[J]. Real-time Systems. 2002, 23(1-2):85-126.

[5] Anton C, et al. Feedback Feedforward Scheduling of Control Tasks[J]. Real-time Systems. 2002, 23(1-2):85-126.

[6] P. Marti. Analysis and design of real-time control systems with varying control timing constraints [D]. Technical University of Catalonia, 2002.

[7] Cervin A, et al. How does control timing affect performance[J]. IEEE Control Systems Magazine. 2003, 23(3):16–30.

[8] Pingfang Z, et al. Feedback Scheduling for Resource-Constrained Real-time Control Systems[C]. The 5th International Conference on Computer and Information Technology, 2005.

[9] Simon D, et al. Robust control / scheduling co-design: application to robot control Technology[C].//The 11th Real Time and Embedded Technology and Applications Symposium, San Francisco, California, USA. RTAS, 2005: 118-127.

[10] Bing D, et al. FEL-H Robust Control Real-Time Scheduling[J]. Software Engineering & Applications, 2009, 2:60-65.