Implementation of Real-Time Scheduling Algorithm on Multi-Core Platform

Xuemei Zhang

School of Computer Science and Engineering
Xi'an Technological University
Xi'an, 710021, China
E-mail: 875781076@qq.com

Shujuan Huang

School of Computer Science and Engineering
Xi'an Technological University
Xi'an, 710021, China
E-mail: 349242386@qq.com

Jinghui Li

School of Computer Science and Engineering
Xi'an Technological University
Xi'an, 710021, China
E-mail: 715319152@qq.com

***Abstract*—With the increasing demand for computing power in embedded systems, multi-core processor architectures have become increasingly common[1]. This paper studies and analyzes various real-time scheduling algorithms and test platform LITMUSRT under the embedded multi-core platform, and proposes a multi-core scheduling method for real-time tasks with dependencies. The static real-time scheduling algorithm RM (Rate-Monotonic), the algorithm and the proposed multi-core scheduling method for real-time tasks with dependencies are implemented in LITMUSRT. Test cases show that the proposed multi-core scheduling method with dependencies can be implemented in the actual Linux environment and can meet the real-time requirements of tasks in real-time scheduling.**

Keywords-Component; LITMUSRT; Multi-Core Systems; Rate Monotic Algorithm; Real-Time Scheduling Algorithm

I. INTRODUCTION

In 1973, Liu and Layland proposed a static priority scheduling algorithm-Rate Monotonie (RM) scheduling algorithm [2], but the algorithm was based on an ideal scheduling model based on a series of ideal assumptions, and In application, consider the influence of various factors. In addition, the real-time scheduling algorithm that has been implemented on the LITMUS^{RT} platform does not consider the dependencies between tasks, nor does it consider the operation of safety-critical tasks [3].

This paper implements the RM scheduling algorithm and the scheduling algorithm of real-time periodic tasks with dependencies by modifying the code of the LITMUS^{RT} platform, and implements the operation of real-time periodic tasks with safety-critical factors on the platform, which solves the research field of multi-core real-time scheduling Some practical problems.

II. LITMUSRT PLATFORM AND SCHEDULING ALGORITHM

A. LITMUSRT platform

LITMUS^{RT} platform was researched and developed by Professor James H. Derson and his multi-verification time scheduling team in the United States. The original design of LITMUS^{RT} platform is to compare the performance, constraint and application scope of different types of multi-core scheduling algorithms in a unified real-time system. This platform can perform scheduling test for tasks with the same load for different scheduling algorithms, and then compare and analyze some index data. It is based on Linux kernel. LITMUS^{RT} has modified Linux kernel, supported random task model, modular scheduling algorithm plug-in, and provided a new synchronization algorithm. The current version is version 2011.1, supporting X86, ARM and other architectures.

The platform mainly consists of five parts: the underlying data structure, the scheduling plug-in, the synchronous processing mechanism, the scheduling trace, and the system call.

The underlying data structure and scheduling tracking are part of LITMUS^{RT} support. The underlying data structure is responsible for the implementation of the data structure of the scheduling plug-in and synchronous processing mechanism, and the scheduling tracking part uses Linux counters to record the scheduling information.

The scheduling plug-in and synchronous processing mechanism are part of LITMUS^{RT} processing. The scheduling plug-in includes the implementation of various scheduling algorithms and is integrated into the plug-in form

978-1-7281-7083-1/20/$31.00 ©2020 IEEE
DOI 10.1109/ICCNEA50255.2020.00023

in LITMUS^{RT}; the synchronization processing mechanism partially implements the synchronization mechanism.

The system call interface provides 14 system calls for users to use to write real-time task programs. The architecture of LITMUS^{RT} is shown in Figure 1:

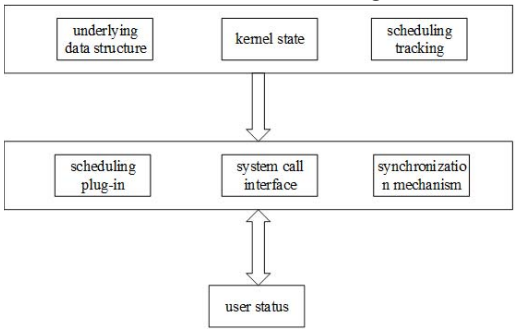

Figure 1. LITMUSRT architecture diagram

B. Introduction to RM (Rate-Monotonic) scheduling algorithm

The idea of the static priority scheduling algorithm is: as soon as the task is generated, its priority is determined and will not change until the task dies, and the task and the job have the same priority. The RM scheduling algorithm is a representative of this type of scheduling algorithm [4]. The task priority of the scheduling algorithm is allocated according to the task period T. The scheduling priority is determined according to the length of the task's execution period. Those tasks with small execution periods have The higher the priority, the lower the priority of long-period tasks.

Liu and Layland proved the optimality of the RM scheduling algorithm, which is the basic theory of real-time scheduling. Even if the system is overloaded instantaneously, it is entirely possible to predict which tasks have lost time limits. The disadvantage is that the processor utilization rate is low. In the worst case, when n→∞, it does not exceed ln2 (≈ 70%) [5].

C. Task scheduling algorithm with dependencies

There may be data or control constraints between real-time tasks, so that a certain task must wait for the completion of other tasks before it can run. If task 2 must run after task 1 is completed, it is called task 2 dependent on task 1. The data and control dependencies between tasks restrict the order in which tasks are executed. Let T={T1, T2, T3,..., Tn} be the task set. If the relationship Tj depends on Ti, then call Ti the precursor of Tj (Tj is the successor of Ti), that is, Tj can only be executed after Ti is executed. start execution.

The real-time system has been widely used in various critical safety systems. Its main function is to use an appropriate scheduling mechanism to allow more tasks to complete the calculation before the dead limit is reached, because it generally contains multiple tasks However, the processing capacity of the processor is limited, so at a certain moment there will be an instant overload of the task, and some tasks will miss their dead limit. However, the current real-time scheduling mechanism does not consider the criticality of each task, nor does it have an overload handling mechanism. Therefore, this paper realizes that by adding key factors to the real-time task to determine the importance of the task, it can ensure that it can run better. The larger the value of the key factor, the more critical the task. When there is no dependency between the two tasks, if they are released at the same time, then specify the task with the larger key factor to execute first, or the task with the larger key factor can preempt the smaller key factor Tasks, run first.

The scheduling process is shown in Figure 2:

III. REALIZATION OF REAL-TIME SCHEDULING ALGORITHM IN LITMUSRT

A. Implementation of RM algorithm in LITMUSRT

The steps to add RM algorithm to LITMUS^{RT} are as follows:

1) In the litmus2010/include/litmus directory, copy a copy of edf_common.h, rename it to rm_common. h, and replace all edf in it with rm.

2) In the litmus2010/litmus directory, copy a copy of edf_common.c, rename it to s_rm_common.c, and replace all edf in it with rm. The principle of the RM scheduling algorithm is to determine the scheduling priority according to the length of the task execution cycle. The smaller the execution cycle, the higher the task priority, and the longer the execution cycle, the lower the task priority, which is a static priority. Therefore, you only need to add the judgment about real-time task cycle comparison in the rm_higher_prio function in the s_rm_common.c file. The code added in the added rm_higher_prio function is as follows:

```
If (first_task->rt_param.task_params.period < second_
task->rt_param.ta
return 1;
}
else if (first_task->rt_param.task_params.period > sec
ond_task->rt_param.task_params.period) {
return 0;
}
```

As can be seen from the above code, if the period of the first task is less than the period of the second task, it returns 1, that is, the priority of the first task is high; otherwise if the period of the first task is greater than the second The period of the task returns 0, that is, the priority of the second task is high. If the priorities of the two tasks are equal, the RM scheduling algorithm fails, assuming that the algorithm in the original edf_common. c is used to judge.

The flow chart is shown in Figure 3:

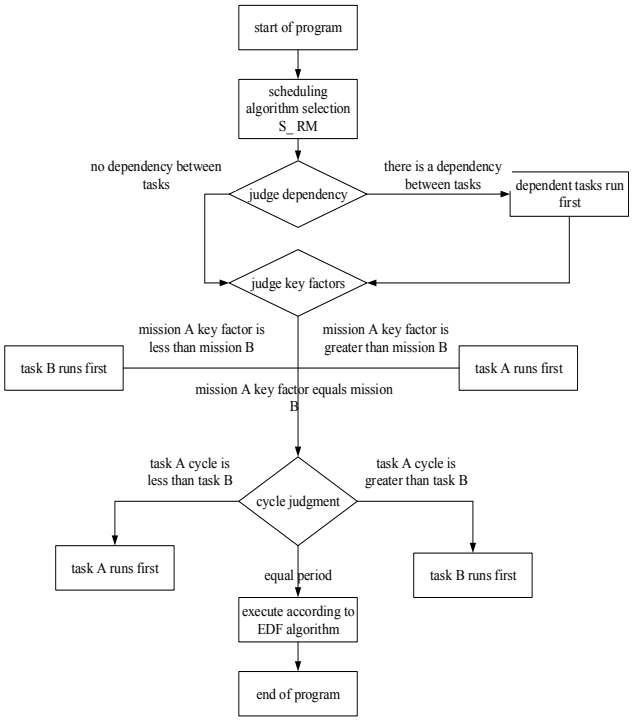

Figure 2. Task scheduling flowchart with dependencies

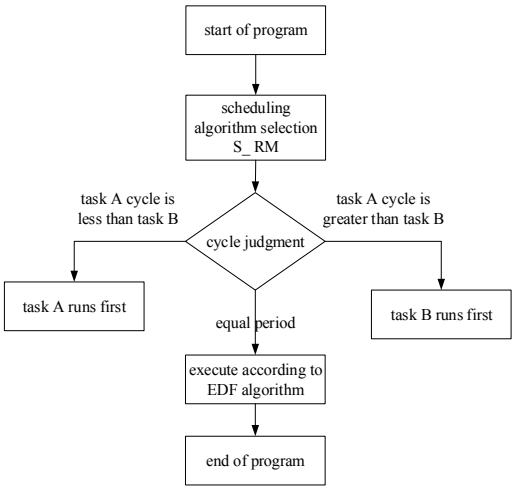

Figure 3. Flow chart of RM scheduling algorithm

3) In the litmus2010/litmus directory, copy a copy of sched_gsn_edf.c, rename it to sched_s_rm.c, and replace all edf in it with rm. In the sched_s_rm.c file, there will be a registration function to display the options of the S_RM scheduling algorithm in the interface when running ./setsched.

4) Add in litmus2010/litmus/Kconfig:

```
config PLUGIN_S_RM
        bool "S_RM"
    depends on X86 && SYSFS
        default y
        help
            Include the s_rm plugin in the kernel.
```

When installing LITMUS^{RT}, the selection scheduling algorithm that appears when running make menuconfig is added in this Kconfig file. At this time, make menuconfig selects the scheduling algorithm, and you can see that S_RM has been selected by default.

5) In the Makefile in litmus2010/litmus, add s_rm_common.o in obj –y, then add obj -$(CONFIG_PLUGIN_S_RM) += sched_s_rm.o below.

6) Compile in the litmus2010 directory, the specific order is:

```
make bzImage
make modules
make modules_install
make install
```

In order to make full use of the quad-core CPU, you can specify options as follows:

```
make –j8 bzImage
make –j8 modules
make –j8 modules_install
make –j8 install
```

Among them, -j is an option of the make command, and you can view the usage of make through man make.

7) After compiling, restart. Enter the liblitmus directory and run./setsched, you can see the option of adding S_RM scheduling plugin, as shown in Figure 4:

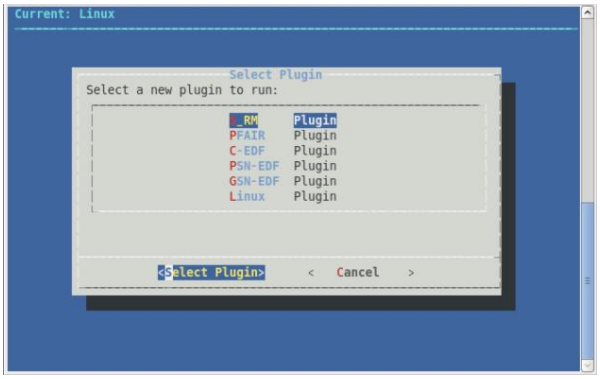

Figure 4. Scheduling algorithm selection interface

B. Implementation of real-time task scheduling model with dependencies in LITMUSRT

The steps to add a real-time task scheduling algorithm with dependencies to LITMUS^{RT} are as follows:

1) Add struct task_struct request to the rt_param structure; this variable represents the dependency of the task.*

2) Modify in the bheap.c file in the litmus2010/litmus directory:

First, add some necessary header file, In addition, two global variables are defined and one variable is extern.

```
raw_spinlock_t queue_lock;
struct bheap_node *zzhmax;
rt_domain_t s_rm;
```

Three functions have also been added:

*a) void swap_node (struct bheap_node *first,struct bheap_node *second)*

*b) void trav (bheap_prio_t higher_prio, struct bheap_node *node)*

*c) void traversal(bheap_prio_t higher_prio, struct bheap_node *node)*

These three functions are mainly used to traverse the binomial heap. Among them, the swap_node function is used to exchange two nodes and is called in the traversal function. The trav function traverses the binomial heap recursively and is called in the traversal function. The traversal function is the calling function and is called in the __bheap_min function.

3) Other code changes are omitted due to space limitations.

4) Compile in the litmus2010 directory, the specific order is:

```
make bzImage
make modules
make modules_install
make install
```

In order to make full use of the quad-core CPU, you can specify options as follows:

```
make –j8 bzImage
make –j8 modules
make –j8 modules_install
make –j8 install
```

Among them, -j is an option of the make command, and you can view the usage of make through man make.

5) Restart after compilation.

C. Realization of scheduling method of real-time tasks with key factors

Based on the existing RM scheduling algorithm and dependencies, the key factors of real-time tasks are added to make a comprehensive scheduling algorithm, that is: first determine the dependencies, and determine the key factors under the premise that the dependencies are satisfied, the key factors are the same According to the RM scheduling algorithm, if the cycle of the two real-time tasks is the same again, it is judged according to the EDF algorithm.

On the basis of chapter 3.2, the specific steps for adding key factors are as follows:

1) In the rt_param.h file in the include/litmus directory, add a variable int priority to the rt_param structure;

2) In the litmus.c file under the litmus directory, add the system call function.

3) In the unistd_32.h file in the litmus2010/include/litmus directory:

Add #define __NR_change_request __LSC(13), then change the following #define __NR_null_call __LSC(13) to #define __NR_null_call __LSC(14), and change #define __NR_litmus_syscalls 14 to #define __NR_litmus_syscalls 15.

4) Add the system call to the syscall_table_32.S file in the litmus2010/arch/x86/kernel directory, and add .long sys_change_request above the .long sys_null_call.

5) Changes in other places are omitted due to space limitations.

6) Compile in the litmus2010 directory, the specific order is:

```
make bzImage
make modules
make modules_install
make install
```

In order to make full use of the quad-core CPU, you can specify options as follows:

```
make –j8 bzImage
make –j8 modules
make –j8 modules_install
make –j8 install
```

Among them, -j is an option of the make command, and you can view the usage of make through man make.

7) Restart after compilation.

IV. TEST CASE

A. The correctness verification of RM scheduling algorithm

In order to test the correctness of the added RM scheduling algorithm, the test cases shown in Table 1 are designed for verification. The test cases are placed in the text of args.txt.

TABLE I. TEST DATA OF RM SCHEDULING ALGORITHM

Task	Execution time	Execution cycle
T0	20	35
T1	110	140
T2	8	10
T3	11	20
T4	40	70

Enter the liblitmus directory, run the ./setsched script, select the added S_RM scheduling algorithm, run

the ./showsched script to verify that the scheduling algorithm is selected correctly, run st_trace in the ft_tools directory for data statistics, and finally run base_mt_task to get four bin files, use unit -trace –c –v *.bin can draw graphics, some graphics screenshots are shown in Figure 5:

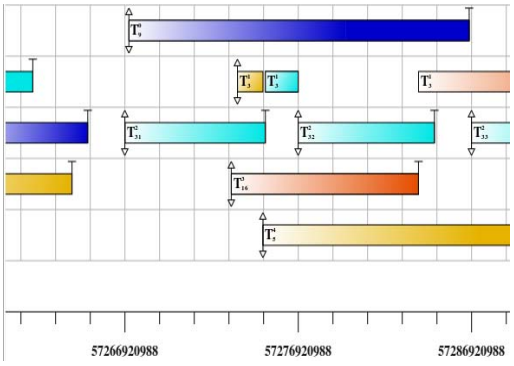

Figure 5. RM scheduling algorithm graph

It can be seen from the figure that the third run of task 1 runs on CPU2, and then the fifth run of task 4 preempts task 1, because the cycle of task 4 is short, the RM algorithm is to determine the priority according to the length of the cycle, the longer the cycle, The lower the priority, the shorter the period and the higher the priority, so task 4 can preempt task 1. After task 31 runs on CPU0 for the 31st time, CPU0 is idle, and task 1 runs on CPU0. Since task 2 has a shorter cycle than any one, it is preempted by task 2 for the 32nd run. This proves the correctness of adding RM algorithm to LITMUS^{RT}

B. *Graphical display verification of real-time task dependencies and key factors*

In order to test the accuracy of the added real-time task dependencies and key factors, the test cases shown in Table 2 are designed to verify, and the test cases are placed in the text of args.txt.

Enter the liblitmus directory, run the ./setsched script, select the added S_RM scheduling algorithm, run the./showsched script to verify that the scheduling algorithm is selected correctly, run st_trace in the ft_tools directory for data statistics, and finally run base_mt_task to get four bin files, use unit -trace –c –v *.bin can draw graphics, some graphics screenshots are shown in Figure 6:

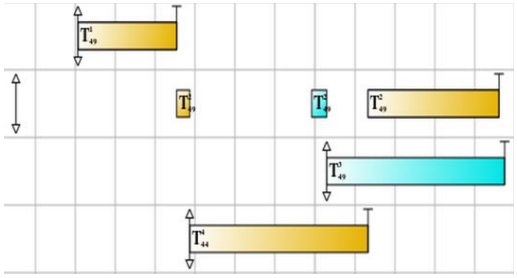

Figure 6. Graphic run by unit-trace

TABLE II. GRAPHICAL DISPLAY OF REAL-TIME TASK DEPENDENCIES AND KEY FACTORS FOR VERIFICATION TEST CASES

Task	Execution Time	Execution Cycle	Dependencies	Key Factors
T0	18	80	0	6
T1	5	80	0	1
T2	8	78	1	3
T3	9	80	1	4
T4	9	89	1	5
T5	40	90	3	1
T6	25	85	2	2
T7	33	87	5	3
T8	83	89	6	4
T9	74	78	7	5

The figure above contains task 1, task 2 and task 3, task 4. From the theoretical analysis in args.txt, we know that the predecessors of task 2, task 3 and task 4 are all tasks 1, task 2, task 3 and task 4. The key factors are 3, 4, and 5, respectively. Because of the existence of the dependency relationship, if the nth job of task 1 is not running, then the nth job of task 2, task 3, and task 4 cannot be run. Due to the existence of key factors, if task 2, task 3, and task 4 do not depend on task 1, tasks with high key factor values can preempt tasks with low key factor values.

Analysis of the experimental results in the figure shows that periodic task 2 is released first, but due to its dependence on periodic task 1, periodic task 2 does not obtain a CPU, then periodic task 1 releases and obtains a

CPU, waiting for periodic task 1 to run After the complete state, at this time, periodic task 2 starts to run on the current CPU, but after running for a period of time, periodic task 4 is released, because the critical factor of periodic task 4 is higher than the critical factor of periodic task 2, there is no other idle at this time CPU (due to screenshots, other data is not shown), so periodic task 4 preempts the CPU of periodic task 2 and runs. After a period of time, periodic task 2 is put on another idle CPU to run. After a period of time, the periodic task is found 3 release, because the critical factor of periodic task 3 is also higher than that of periodic task 2, there is no other idle CPU at this time (other data is not shown due to screenshots), so periodic task 3 also preempts the CPU of periodic task 2 and Run; after a period of time, periodic task 4 becomes complete after running. At this time, periodic task 2 gets idle CPU and continues to complete after running, and then periodic task 3 completes after running on another CPU.

From the above analysis, we can see that the added algorithm is correct.

V. CONCLUSION

The article introduces the test platform LITMUSRT and how to add the RM scheduling algorithm to it, and proposes a real-time task scheduling algorithm with dependencies. On the basis of it, key factors are added. On the premise of achieving dependencies, the key factors High tasks are given priority.

The experimental results verify the correctness of the results added by the RM scheduling algorithm, and test cases of real-time task dependencies and key factors are given, and the correctness of the results is verified graphically.

REFERENCES

[1] Jiang Xiaowen. Research on energy-saving and reliability optimization scheduling of multi-core real-time system [D]. Zhejiang University, 2018.

[2] Rashmi Sharma,Nitin Nitin,Mohammed Abdul Rahman AlShehri,Deepak Dahiya. Priority-based joint EDF–RM scheduling algorithm for individual real-time task on distributed systems[J]. The Journal of Supercomputing,2020(prepublish).

[3] Xiang Hua, Chen Jihong, Zhou Yunfei, Chen Lixin. Research on modeling method of real-time detection system based on PC numerical control[J]. China Mechanical Engineering, 2003(20): 73-75+5-6.

[4] Xu Jiang,Nan Guan,Xiang Long,Yue Tang,Qingqiang He. Real-time scheduling of parallel tasks with tight deadlines[J]. Journal of Systems Architecture,2020,108.

[5] Liu Dong, Meng Qingxin, Pan Zhe. A fault-tolerant scheduling algorithm based on dynamic variable scheduling distance for tasks with dependent contexts in heterogeneous distributed real-time systems [J]. Computer and Network, 2014, 40(Z1): 109 -113.