Research on Real-time Performance Testing Methods for Robot Operating System

Ming Huang
Software Technology Institute, Dalian Jiao Tong University, Dalian, China

Shujie Guo
Mechanical Engineering Institute, Dalian Jiao Tong University, Dalian, China
Email: shujieguo@126.net

Xu Liang
Software Technology Institute, Dalian Jiao Tong University, Dalian, China
Email: liangxu00@263.net

Xudong Song
Computer Science Department, Database System Research Group Lab
Worcester Polytechnic Institute, MA, USA,

***Abstract*—To test real-time performance of RGMP-ROS, a robot operating system, this paper takes an in-depth study of the basic functions of real-time operating system kernel. By analyzing the main factors that affect the real-time performance of an operating system, we propose a set of real-time performance testing methods based on mixed load. This paper introduces the concept of the calibration procedure program. Takes the average execution time of the program as the standard time unit to normalize the test results of each test indicator, so as to shield the effect of the hardware test environment on the test results to a certain extent. While testing task preemption time and interrupt response time, we use background tasks that can trigger banning preemption or disabling interrupts to simulate the load, which makes the test results more real and reliable.**

***Index Terms*—Operating System, Real-time performance testing, Mixed load**

I. INTRODUCTION

A real-time system is one in which the correctness of the computations not only depends on the logical correctness of the computation but also on the time at which the result is produced. If the timing constraints of the system are not met, system failure is said to have occurred [1]. The operating system is an important part of real-time systems, the real-time performance of operating systems is the key factor that affects the real-time performance of the real-time system. Investigation of Evans Data company shows that, for real-time operating system, developers are most concerned about is the real-time performance[2]. Given the importance of the operating system real-time performance, more and more researchers began to focus on the real-time operating system performance testing, made a lot of testing methods. RP Kar and K. Porter proposed a benchmark test suite for evaluating the performance of operating systems [3]. This suite evaluates the real-time performance of the operating system through measuring the task switching time, task preemption time, interrupt latency time, semaphore shuffling time, deadlock release time and message delay time. Nelson Weiderman proposed an evaluation kit [4] for testing real-time performance of hard real-time system. The kit takes into account the different characteristics of periodic tasks and aperiodic tasks to design targeted testing methods. Express Logic presents a benchmark suite of Thread-Metric [5] for real-time operating system performance assessment. The kit assessed the operating system real-time performance by recording execution times of an operation during a certain period of time. LC Briand put forward a real-time system stress testing method based on genetic algorithm [6]. Lu Jun designed a set of real-time performance testing program based on in-depth analysis of RTLinux system[7]. Liu Yunsheng proposed two methods taking absolute time and applied them to real-time performance test of the operating system [8]. Lijuan presented a real-time performance test method of the airborne embedded real-time system [9]. Shen Baoguo put forward a real-time performance testing program for VxWorks[10]. Chu Wenkui proposed a performance test method by cooperating with hardware and software for the Linux operating system [11]. Zhao Liye put forward a set of indicators to assess performance of real-time systems and analyze the effect of various indicators on real-time performance [12]. Against application requirements for aerospace systems, Dong Jialiang put forward a set of real-time performance evaluation system [13]. Wu Xun analyzed factors affecting the real-time

Project supported by the National High Technology Research and Development Program of China (No. 2012AA041402-4) and the Education Department of Liaoning Province outstanding young scholars growth plan(No. LJQ2013048);

Manuscript received March 6, 2014; revised May 7, 2014; accepted June 5, 2014.

doi:10.4304/jsw.9.10.2685-2692

performance and select the interrupt response latency and scheduling latency of period task as test indicators to complete real-time operating system performance test [14]. Jiang Jianhui put forward an improved embedded operating system real-time performance measurement method based workloads [15]. The introduction of workload made the test results more accurate. Yang Heng Proposed software programming and hardware-assisted collaborative methods for testing real-time performance of poloidal field power supply operation system[16].

RGMP-ROS is a self-developed real-time robot operating system with better certainty, scalability and configurability. It can be applied to a typical X86 processor, embedded processors and multicore processors. To test the performance of RGMP-ROS operating system, we make an in-depth study of the basic function of real-time operating system kernel and analyze the factors affecting the real-time performance of the operating system. On the basis of studying the real-time performance testing techniques described above, a real-time operating system performance testing program based on mixed load is proposed in this paper.

II. Factors That Affect Real-Time Performance Of The Operating System

A real-time operating system is generally consists of a kernel, network system and file system, etc. And kernel is the core component of the real-time operating system. It provides time management, task management, interrupt management, synchronization and communication mechanisms, memory management and other functions. The efficiency of these functions is key factor that determining operating system real-time performance.

A. Time Management and Its Impact on Real-Time Performance

Time management can provide support for real-time response of the application system to ensure it is running in real time and orderliness. In each operating system, a timer is designed to generate periodic clock interrupt signals and whenever a clock interrupt arrives, the counter value is incremented by 1. The time interval between two clock interrupts is the smallest timing unit of the system clock, called the clock granularity. The granularity size of the clock determines the time accuracy of the operating system and the response speed to events. It is one of the factors that affect the real-time performance of the operating system.

B. Task Management and Its Impact on Real-Time Performance

Task management is the core part of the kernel. It has functionalities such as creating task, deleting task, suspending task, restoring task, task scheduling and setting task properties. Creating a task is to allocate and initialize associated data structure. Deleting a task is to release the task control block corresponding to the task. Suspending a task is to make the task in a waiting state. Restoring a task is to wake up a suspended task to make it in a ready state. Task attribute setting is used to set task preemption time, time slice and other attributes. Task scheduling refers to the process of determining when each task consumes system resources after giving a set of real-time tasks and system resources. In several functional modules of task management, task creation, task deletion, task suspension and restoration, tasks property setting and other functions are simply processed and used in a relatively low frequency. They are not the main factors affecting the operating system real-time performance. In order to accurately manage system resources to achieve requirements of real-time performance and predictability, it is necessary for task scheduling module to take task scheduling policy analysis and schedulability analysis. Therefore, the process of scheduling functions is more complex. It is necessary to perform scheduling functions at scheduling points such as, when an interrupt service routine is over, when a task is in a waiting state due to waiting for resources, when a task is in a ready state and when the task time slice is used up. Therefore, using the frequency of scheduling function is very high. So scheduling algorithm is one of the core elements to determine real-time performance of the operating system.

C. Interrupt Management and Its Impact on Real-Time Performance

Interrupt management is an important part of real-time operating system kernel, used for management and real-time processing of all kinds of events. Interrupt is a hardware mechanism for asynchronous event notification for the CPU. Once the interrupt is recognized, the CPU saves part or all of the context which is some or all of the values of the registers and jumps to a special subroutine called interrupt service routine (ISR). Interrupt service routine processes the event and after processing is complete, the program returns to the interrupted task or the task which is with the highest priority and in a ready state.

Interrupt latency is one of the key factors that determine the operating system real-time performance. The main reasons leading to interrupt latency include the following three ones: (1) When system kernel is executing certain critical region of code, interrupts must be disabled to prevent certain common data structures from being multiply accessed.(2) It is likely that the system is handling interrupt with higher priority, which leads to being unable to timely respond to the current interruption.(3) It will spend a fair amount of time for a series of work, from detecting a generated interrupt by the system to the execution of the interrupt service routine, reading interrupt vector from the interrupt controller chip, saving the state of flag register, finding the interrupt vector table and jumping to the interrupt service routine. Because this part of work is often done by hardware, it is also known as hardware response time. To sum up, interrupt latency time consists of three parts which are the maximum disabling interrupt time of the kernel, interrupt nesting time and hardware response time. Hardware response time is generally very short and negligible. Interrupt nesting time is related to specific application so nested layers and ISR execution time of

each nested interrupts are uncertain and therefore there is no way to measure interrupt nesting time. Thus, the largest disabling interrupt time of kernel is one of core elements affecting interrupt latency and thereby real-time performance of the operating system.

D. Synchronization and Communication Mechanisms and Their Impact on the Real-Time Performance

In multi-tasking real-time systems, data or information transmitted is often required between tasks or between tasks and interrupt service in order to achieve secure access to shared resources, synchronous activities of multiple collaborative tasks or simply exchange of information and other functions. General real-time kernels will provide a rich set of synchronization and communication mechanisms, including semaphores, events, mailboxes, message queues, pipes, global variables, shared memory, RPC and etc. In most cases, a shared resource can be used by only one task at one moment and cannot be interrupted by other tasks during occupation. Thus, the shared resource management is the core problem of multi-tasking system. Semaphores are most commonly used as a shared resource management mechanism, so the semaphore management is a major factor affecting the real-time performance of the operating system on communication and synchronization mechanism are concerned.

E. Memory Management and Its Impact on the Real-Time Performance

Memory holds code and data that the CPU can directly access and are an important part of real-time systems. In order to ensure real-time and reliability of the operating system, efficient and sophisticated memory management ways are an important part of real-time operating system. Real-time performance requirements determine that real-time operating system must use quick and definite memory management and cannot use virtual storage technology. Therefore, memory management methods used in the real-time operating system kernel are very simple. And the memory management is one of the major components of real-time operating system kernel, but not the major factor affecting the operating system real-time performance.

III. Real-Time Performance Testing Program Of Operating System Based Mixed Load

A. Timing Method

The main method of real-time performance test is to obtain the execution time of some system behavior. Therefore, the primary task of performing real-time operating system performance testing is to solve the timing issue. Due to the various timing method itself will spend part of the CPU time and in order to improve the accuracy of the test results, calculating the cost of the chosen timing method should be as low as possible based on guaranteeing the effective timing accuracy. Commonly used methods include hardware timing and software timing.

Hardware timing method mainly uses two ways that are the PCI bus analyzer and GPIO pin output timer. The PCI bus analyzer is a PCI plate embedded into the measured target system. It stubs before and after the test code and writes data to the PCI bus address during system execution. PCI bus analyzer stores data written to the bus and makes the timestamp. After the test is completed, the data is downloaded from the bus analyzer to the development machine. By analysis of these data, we can get the time spent on performing the measured operation. The timing method of GPIO pin output is similar to a timing method by the PCI bus analyzer. It is also necessary to stub test code. An instruction is inserted at the beginning and the end of the test code. The signal is output to a GPIO pin. The test plates complete the timing work by capturing these signals.

The software timing ways also include two ways that are system calls and autonomous obtaining way. RTOS typically offers system calls with higher precision. By calling function *gettimeofday* () provided by RTLinux we can achieve us-level timing. The *clock_gettime* () function can provide ns-level timing. Autonomous obtaining way uses assemble instruction of *rdtsc* to read the CPU time counter. It returns the elapsed CPU cycles as of booting by EDX: EAX register. In the case of determining CPU frequency, a more precise and absolute time can be achieved by the return value.

Hardware Timing can obtain high timing accuracy in the case of almost no consuming CPU time. However, it requires additional hardware and the cost is higher. Meanwhile, it can not be used in software testing for closed-source code due to the need for source code stub. Software timing method of autonomous obtaining has strong versatility and the consumption of CPU time is relatively small. However, development trend of the current processors is multi-core, energy-saving and intelligent. In this background, the accuracy of the *rdtsc* instruction is greatly weakened for three reasons: (1) It cannot be guaranteed that TSC of each core on the same board is synchronized.(2) The clock frequency of the CPU may change. For example, laptops reduces the clock frequency in power-saving mode.(3) In order to reduce performance missing caused by problems such as data-dependency, to avoid certain types of delayed consumption and to improve the execution efficiency of the processor instruction, the out-of-order execution scheme is used in the process. This scheme leads directly to a result that the number of cycles measured using *rdtsc* is inaccurate.

Based on the above analysis, we use a software timing method based on system calls to complete timing work in the real-time performance testing program of operating system based on a mixed load.

B. Shielding Effects of Hardware Performance on Test Results

Real-time performance of the operating system is often evaluated by the length of time required to complete certain services. However, the time spent on completion of certain operations is subject to the hardware platform performance on which the operating system is running, in

addition to the efficiency of the operating system itself. In order to shield the impact of hardware platform on the test results, we prepare a calibration procedure in the operating system real-time performance testing program based on a mixed load. The average execution time of the program is taken as a standard time unit (hereinafter referred to as calibration reference T_Standar) to standardize test results for each indicator. The hardware which is close with the execution efficiency is the CPU and memory. Thus, the calibration procedure should not only reflect the CPU speed but also can roughly assess reading and writing speed of memory. Meanwhile, the execution time of the calibration program as a reference and the execution time of calibrated objects should not have too much difference. To simultaneously achieve these objectives, the reference operation is repeatedly executed 100 times in the calibration program and we record the time to complete the reference operation, T1, T2, ... T100. We take the average of T1-T100 as the calibration reference T_Standar. The reference operations perform multiplication by two 2×2 order double- type matrixes. The processing flow of the calibration procedure is shown in Figure 1.

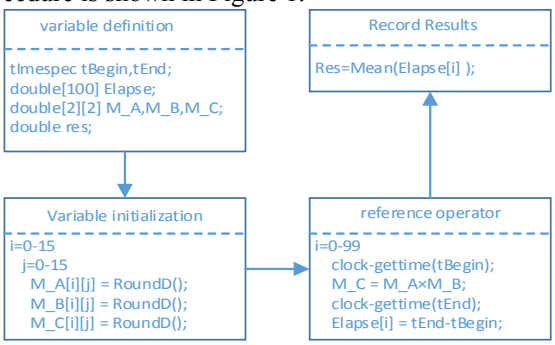

Figure 1. The processing flow of the calibration procedure

C. Test Indicators and Test Methods

According to the analysis in section II, the size of clock granularity, the efficiency of the task scheduling algorithm, speed of interrupt response, communication and synchronization mechanisms and memory management scheme and other factors, all of these are factors influencing real-time performance of the operating system, but these factors vary in the influence on the real-time performance of the operating system. RTOS uses relatively simple memory management in order to achieve fast and clear memory management objectives. At the point of memory management technology, each RTOS is of no much difference, so memory management is not taken as a test indicator in the operating system real-time performance testing program based mixed load. The smaller clock granularity is the basis of the RTOS. The clock granularity of any RTOS can meet real-time requirements. Although size of clock granularity will have some impact on real-time performance of the operating system, it is not the main factor affecting operating system real-time performance. Moreover, the impact on the operating system by clock granularity can be reflected by testing task preemption time and thus it is unnecessary to take it as an independent test indicator. Communication and synchronization mechanisms are one of the important factors affecting the real-time performance of the operating system. The semaphore is one of the most commonly used means of communication and synchronization and thus the semaphore shuffling time is taken as one of test indicators. Task scheduling algorithm and interrupt management capabilities are key factors affecting operating system real-time performance. Efficiency of scheduling algorithm is mainly reflected in the task switching time and task preemption time. Interrupt management capabilities can be evaluated through the response time. Therefore, task switching time, task preemption time and interrupt response time is taken as the test indicators.

- *Task switching time and its test methods*

Task switching time refers to the period during which the system switches from one task to another task with the same priority. The task switching time reflects the efficiency of task round robin schedule with the same priority task. The task switching process includes three steps that are context saving the current task, selecting tasks to be performed and restoring the context of the new task.

The test principle of task switching time is as follows: create and run multiple threads with the same priority. The execution of these threads is very simple, which just record the current time and give up the right to use CPU and thus its execution time is negligible. On the basis of ignoring the time that the thread itself consumes, we can get the approximate time of task switching by recording running time points of various threads for some time. As shown in Figure 2, t2-t1, t3-t2, etc. can be approximately considered as task switching time.

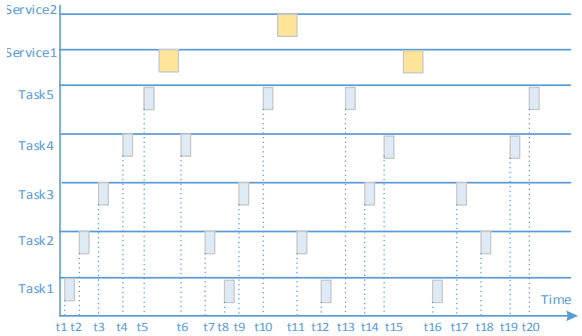

Figure 2. The test principle of task switching time

When using this method for testing, because the operating system itself is also running some service threads (service 1 and service 2 in Figure 3-3), it will spend part of the CPU time, which will lead to such large values as t6-t5, t11 -t10, t16-t15. At this point these large values can be regarded as gross error values which are removed by an improved Pauta criterion. The improved Pauta criterion is as below: When the residual error of a value $v > 3\sigma$, the value is considered a gross error which should be removed. Specific test methods are as follows: Open a monitor thread with the highest priority and 5 test threads with the next highest priority. Testing

thread circularly performs the following actions: (1) Get the current absolute time ti and save it into the switching time array ListTime. (2)Give up occupation of the CPU. Monitoring thread mainly completes the following operations: (1) Sleep for a period of time (e.g. 30 seconds). The length of time period is set by the testers based on their own test needs. (2) Kill all test threads.(3) The absolute time recorded by each test thread in ListTime is converted into approximate scheduled time and stored in the ListRes array.(4) Using improved Pauta criterion to exclude gross error data in ListRes.(5) The maximum value in ListRes is fed back to the testers as the test results. The test process is shown in Figure 3.

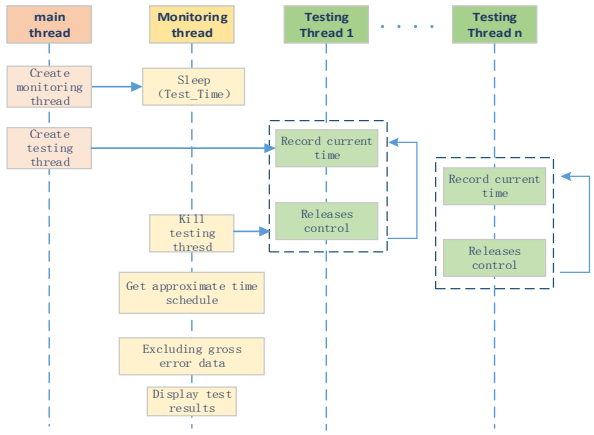

Figure 3. The test process of task switching time

- *Task preemption time and its test methods*

Task preemption time is a time interval from a high-priority task ready to start running by depriving CPU occupation right of low priority tasks, mainly including the prohibition preemption time and task switching time, as show in Figure 4. Prohibition preemption time refers that high-priority task has not yet obtained the right to occupy the CPU immediately due to some reason. The reasons causing prohibition preemption is mainly in the following two: (1) The next, preemption point has not yet arrived. (2) The system is in a critical code lock phase. Task preemption time was the main factor affecting the high-priority real-time task response time. In Figure 3-4, a high priority task B is in the ready state at time T1. At this moment, the system is in a preemption disabled state so an immediate task switch does not happen. The system allows preemption beginning at time T2 and then performs task switching so the task B starts executing by depriving the CPU occupation right of task A.

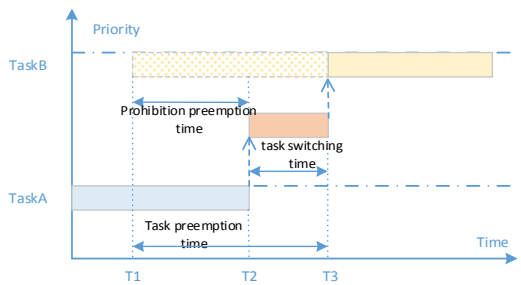

Figure 4. Task preemption time

In the task preemption time test, it is necessary to create and launch four different threads which are background thread, control thread, testing thread and monitoring thread. Background threads consist of five work threads with different priority and their priorities are lower than the highest priority. They circularly perform some operations which may trigger prohibition preemption to simulate the system load until they are terminated by monitoring thread. Control thread has medium priority and circularly performs the following operations: Apply for and wake up semaphore SWakeUp. If successful, it will record the current absolute time and save it into the array ListTime and wake up testing thread. If the semaphore is not applied for successfully, sleep for 100 milliseconds. The state of the testing thread is circularly detected. When the testing thread is detected in a suspended state, it will be woken up. Testing thread with the highest priority, performs three operations :(1) Save the current absolute time in the array ListTime. (2) Release and wake up semaphore SwakeUp. (3) Hang themselves. Monitor thread has the second highest priority and mainly completes the following operations :(1) Sleep for a period of time (e.g. 30 seconds). The length of time period is set by the testers based on their own test needs. (2) Force quitting other threads. (3) The absolute time recorded by each test thread in ListTime is converted into preemption time and stored in the array ListRes. The conversion method is ListRes[i] = ListTime [2*i+1] - ListTime [2*i]. (4) The maximum value in ListRes is fed back to the testers as the test results. The testing process of preemption time is shown in Figure 5.

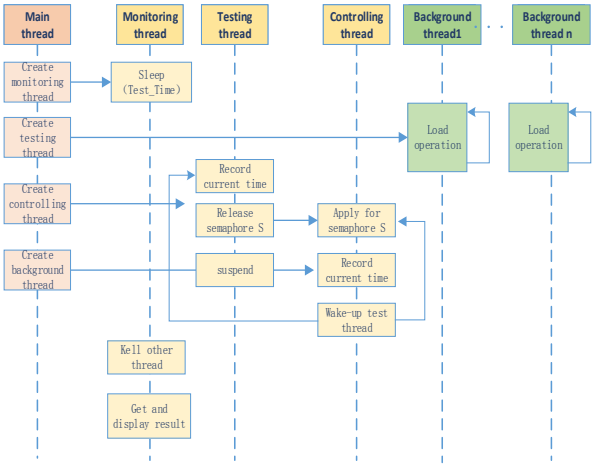

Figure 5. The test process of task preemption time

- *Interrupt response time and its test methods*

Interrupt response time refers to the time period from interrupt occurrence to the beginning of the execution of the first instruction of the users interrupt service routine, including two parts that are terminal latency and execution time of kernel interrupt service routine. The maximum disabling interruption time of kernel is the main factor affecting the interrupt response time. In order to give full consideration to the effect of the maximum disabling interruption time of the kernel on the test results

in testing, we create and start three types of threads in testing methods of interrupt response time: background thread, testing thread and monitoring thread. Background thread has a medium priority. They circularly carry out all types of system calls which may trigger disabling interrupts. Testing thread circularly does the following operations: (1) Apply for semaphore S. (2) If the application is successful, it will obtain the current absolute time and include it in the array ListTime. (3) The interrupt is simulated by calling the soft interrupt instruction. User interrupt service routine mainly do the following: Get the current absolute time and include it in the array ListTime. Release semaphore S. Monitoring thread has the second highest priority and mainly completes the following operations: (1) Sleep for a period of time (e.g. 30 seconds). The length of time period is set by the tester based on their own test needs. (2) Force quitting other threads. (3) The absolute time recorded by each test thread in ListTime is converted into interrupt response time and stored in the array ListRes. The conversion method is ListRes[i] = ListTime[2*i+1]-ListTime[2*i]. (4) The maximum value in ListRes is fed back to the testers as the test results. The testing process of interrupt response time is shown in Figure 6.

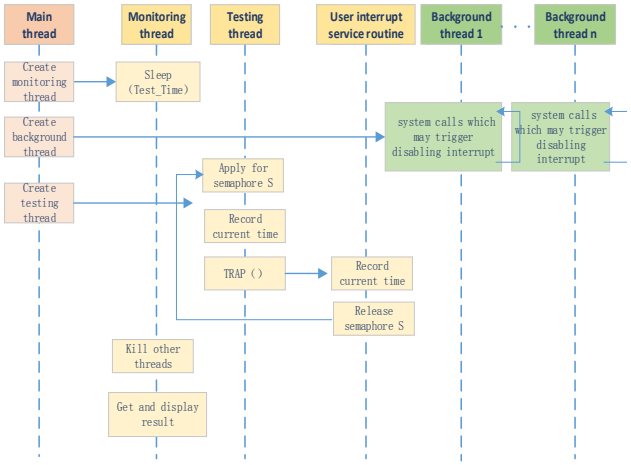

Figure 6. The test process of interrupt response time

- *Semaphore shuffling time and test methods*

Semaphore shuffling time refers to the time period from releasing signal by one task to another task waiting for the semaphore be activated with it, as shown in Figure 7.

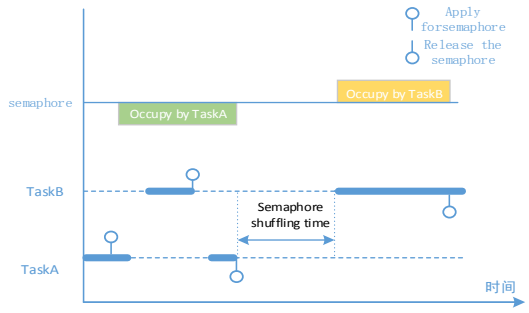

Figure 7. Semaphore shuffling time

Semaphore shuffling time and time overhead associated with mutex is an important indicator of RTOS real-time performance. In semaphore shuffling time measurement, we use three threads which are thread A, B and monitoring thread. The thread A which has the second highest priority, circularly carries out the following operations: (1) Get the current absolute time and include it in the array ListTime. (2) Release the semaphore S. (3) Apply for the semaphore P. Thread B has the highest priority, circularly carrying out the following operations: (1) Apply for semaphore S. (2) Get the current absolute time and include it in the array ListTime. (3) Release the semaphore P. Monitoring thread has the second highest priority and mainly completes the following operations: (1) Sleep for a period of time (e.g. 30 seconds). The length of the time period is set by the tester based on their own test needs.(2) Force quitting other threads.(3) The absolute time recorded by each test thread in ListTime is converted into semaphore shuffling time and then stored in the array ListRes. The conversion method is ListRes[i]=ListTime[2*i+1]-ListTime[2*i]. (4) The maximum value in ListRes is fed back to the testers as the test results. The testing process is shown in Figure 8.

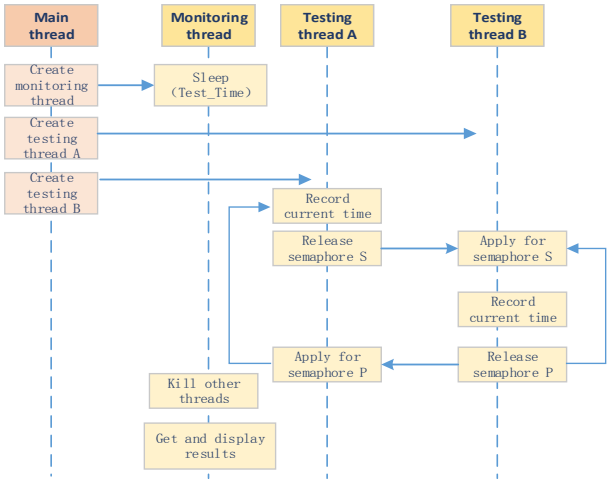

Figure 8. The test process of semaphore shuffling time

D Scoring Standard

In the real-time operating system performance testing program based on mixed loads, each test index score is the ratio of the key code running time and run-time of calibration procedures fed back by the test cases corresponding to the index. The greater the value of the index indicates that consuming time of the index is more. Therefore, for the same test indicator, the lower the score the better the real-time performance of the operating system is. In testing, we first run the calibration procedure and record the average execution time of the program, RT_S, as a calibration reference T_Standar. We run test cases of each index described in section 3.3.1-3.3.4 and record the times indicating the efficiency executing key code by the index, which are task

switching time T_TaskSW, task preemption time T_TaskPM, interrupt response time T_InterruptRP and semaphore shuffling time T_SemaphoreSF. We obtained test scores of indicators which are task switching score (T_TaskSW/RT_S), task preemption score (T_ TaskPM/RT_S), interrupt response score (T_InterruptRP/RT_S) and semaphore shuffling score (T_ SemaphoreSF/RT_S).

IV. Test Results

Real-time performance of the operating system is determined by both the responsiveness and determinist. Response performance refers to the response speed of the system to external events are identified. Deterministic refers to whether under what circumstances, the response of the system is predictable. Namely performance indicators are independent of the system load, the influence of system factors such as the current state.

A. Response Performance Test Results

In order to investigate response performance of RGMP-ROS operating system, we use real-time performance testing program based on a mixed load to conduct real-time performance testing on RGMP-ROS and commonly used open-source real-time operating system RTLinux.

The main hardware test environment is configured as follows: Intel® Core™ i5-2540 processor; 4GB DDR3 memory.

The test results are shown in Table 1.

TABLE I.
THE TEST RESULTS

	RTLinux	RGMP-ROS
Task switching score	3.21774	1.73517
Task preemption score	5.68273	3.40269
Interrupt response score	6.33027	2.97413
Semaphore shuffling score	8.20751	6.49854

As can be seen from Table 4-1, RGMP-ROS real-time performance is better than that of RTLinux in task switching, task preemption, interrupt response, semaphores shuffling, etc.

B. Determinist Performance Test Results

To test the determinist performance of RGMP-ROS, Investigated the influence of the number of tasks on the task switching time, task preemption time, interrupt response time and semaphore shuffling time. The test results are shown in Figure 9

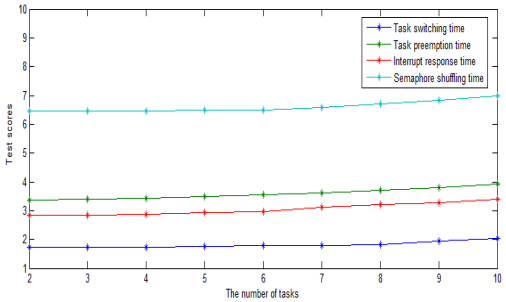

Figure 9. The test result of determinist performance

As can be seen from figure 4-1, although the test scores increase with the increase of the number of tasks, but the increasing amplitude is smaller; That is to say, response performance has very little to do with the number of tasks. Therefore, RGMP - ROS has better certainty.

Acknowledgment

Thanks National High-Tech Research and Development Program of China (863 Program) for its funding of sub-project “Testing of the operating system kernel module and software library” (No. 2012AA041402-4). We are grateful to other members of the project team for their valuable help and suggestions.

References

[1] Comp. realtime: Frequently Asked Questions (FAQs). (Version 3.5) <http://www.faqs.org/faqs/ realtime-computing/faq/>
[2] http://www.evansdata.com/n2/surveys/embedded/2003_1/embedded_xmp1.shtml
[3] R. P. Kar, K. Porter. Rhealstone--A Real Time Benchmarking Proposal [J]. Dr. Dobb's Journal, 1989,14 (2): 14-24
[4] Nelson Weiderman. Hartstone: synthetic benchmark requirements for hard real-time applications. ACM SIGAda Ada Letters 1990,10 (3): 126-136
[5] www.expresslogic.com]
[6] L. C. Briand, Y. Labiche, M. Shousha. Performance Stress Testing of Real-Time Systems Using Genetic Algorithms. GECCO '05, New York, NY, USA 2005: 1021-1028
[7] LU Jun, SHEN Jian, ZHANG Tao Real-time Operation System RTLinux and Technique of Performance Measurements. Journal of Military Communications Technology Vol.28 2007：8-11
[8] Liu Yunsheng, Xu Chao. RTOS Real-time Performance Test. COMPUTER ENGINEERING AND APPLICATIONS 40 (11), 2004:93-95
[9] LI Juan, YE Hong, LI Yun-xi. Time Performance Measurement of Airborne Embedded Real-time Operating System Supporting Partition. Aeronautical Computing Technique, 36 (6) 2006:80-82
[10] SHEN Guo-bao, LIU Song-qiang. Performance evaluation of real time systems. Nuclear Electronics &Detection Technology, 22 (5), 2002:416-419
[11] CHU Wen kui, ZHANG Feng ming, FAN Xiao guang. Measurement of real-time performance of embedded Linux systems. Systems Engineering and Electronics 29 (8), 2007:1385-1401
[12] ZHAO Li-ye, ZHANG Ji, YOU Xia. Analysis and Evaluation of Real-time Operating System Performance. Computer Engineering, 34 (8), 2008:283-285
[13] DONG Jia-liang, LI Yan-feng, YANG Qiu-song. Real-time performance evaluation of embedded operating system in aerospace. Computer Engineering and Design, 34 (1), 2013:114-120
[14] WU Xun, MA Yuan, DONG Qin-peng. Research on Real-time Operating System Performance Measurement. Journal of System Simulation. 25(2),2013:313-316]
[15] JIANG Jianhui, TANG Zhijie, A Novel Method to Measure Real-Time Performance Parameters of Embedded Operating Systems. OURNAL OF TONGJI UNIVERSITY (NATURAL SCIENCE), 36 (9), 2008:1260-1266

[16] Yang Heng, Qin Pinjian, Huang Liansheng. Real- time Performance Test of Poloidal Field Power Supply Operation System. Computer Measurement & Control,18(12),2010:2730-2732