Research on Test Method of Domestic Embedded Multi-core Operating System

Ying ZHANG Jiasi WANG Songyang DU Liang XIANG

Keywords: Embedded Operating System, Software Testing, multi-core Operating System

Abstract: Based on domestic embedded multi-core operating system testing, this paper describes the characteristics of the multi-core operating system and existing test contents and gives an effective software function and performance function test method for application focus of the domestic embedded multi-core operating system so as to provide references for testing of domestic embedded software and embedded operating system.

Introduction

Recently, with the promotion of domestic embedded software, domestic multi-core operating system has been gradually applied in military, aerospace and other important areas in order to lay a solid foundation for development and application of China's independent software. The domestic embedded multi-core operating system is an embedded real-time operating system independently developed based on special application background and operating environment to meet special requirements for the above areas. The operating system adopts REDE development environment to realize editing, compiling, debugging and other functions and promote sustainable development of domestic independent controllable computing platform technology.

Introduction to Embedded Multi-core Operating System

In the single-core operating system, the inner core performs task management, resource management, interrupt management, event processing and communication management. These functions are still available in the multi-core operating system. In addition, the multi-core operating system also can perform inter-core communication and synchronization, inter-core task schedule management, resource sharing and equipment management. Task switching time, interrupt response time, operational performance, semaphore delay time, memory RW performance and other conventional embedded real-time operating system performances need to be considered. In addition, inter-core synchronizing time needs to be tested for the multi-core operating system, graphics performance for graphic operating system and file RW performance for file system.

Testing Contents of Multi-core Operating System

Based on the above characteristics of the multi-core operating system, the software testing for the domestic embedded multi-core operating system is carried out in three aspects:
(1) Multi-core management
The operating system can guide all available processor cores at the system initialization stage. It can support processor set operation and test on-off status enquiry function of the processor core. By charging the target machine loaded with the tested system, the operating system can monitor the

on/off status of all processor cores to verify correctness of multi-core management function.
(2) Multi-core synchronization
Spinlock, atomic operation, memory barrier and other mechanisms are available for testing of inter-core data reentrance for multiple processors. The correctness of critical resource access (consistent with data reentrance) is monitored to verify the correctness of multi-core synchronization function by running the test program, building multiple tasks and comparing shared critical resource access between spinlock and non-spinlock. The memory barrier is verified to ensure the correctness of data operation sequence based on assembly code and running results generated from comparison and compiling under the condition that multiple tasks have concurrent access to the same global variable when the memory barrier is available. The program running results of atomic operation enabling and disenabling are compared by running the testing program to verify that the enabled atomic operation can guarantee the accuracy of running results under multi-core conditions.
(3) Multi-core scheduling
It can support multi-core parallel operation, core preemption, priority-based preemptive scheduling and round-robin scheduling to avoid task failure. Unified task queue management is adopted to achieve unified scheduling of multi-core tasks and enable the CPU for scheduling local ready queue under multi-core conditions on the basis of efficient inter-core interrupt mechanism. It can support processor affinity and provide a programming interface for interrupting the function bound to specified processor core and users can bind tasks to the specified processor core for operation based on application needs. The test program is operated to monitor task scheduling and check CPU loads of the system when multiple priority tasks are combined for operation, thus verifying the correctness of multi-core scheduling function.

Testing Technology and Method of Multi-core Operating System

Task switching time test

Task switching time refers to the time when CPU control right is actively transferred from the running task to the other ready task. It includes the time for saving running task context, that for selecting next task for scheduling and that for context restoration of ready task.
A task switching event needs to be generated to test the task switching time. Under non-preemptive condition, the running task could be used for activating the other task and then get suspended. The activated task will be in the ready state so as to cause scheduling. The switching process involves a series of operations such as running task context saving, new task context restoration and so on. The time cost for the switching process is task switching time.
The running task could be set as TASK[i] and the ready task as TASK[i+I]. T1 indicates the activation time of TASK[i+I] and T2 refers to the operation start time of TASK[i+I]. Thus, the task switching time could be approximately expressed by T2.T1.
Test case design: build the my_test_performance, call pthread_suspend in the task 1 to suspend the task 2 and then call the pthread_resume to restore execution of task 2 and obtain the time from pthread_resume calling to the execution of task 2 by instrumentation in order to test the task switching time under multi-core conditions.

Task preemption time test

The task preemption time refers to the time of system control right transferring from low priority task to high priority task. It involves identification of the event enabling high priority task readiness,

priority comparison and task switching.
Test case design: build the my_test_performance, call pthread_suspend in the task 1 to suspend the task 2 and then call the pthread_resume to restore execution of task 2 and obtain the time from pthread_resume calling to the execution of task 2 by instrumentation in order to test the task response time under multi-core conditions.
Interrupt response time test
Interrupt response time refers to the time interval from interrupt generation to the first command for execution of interrupt handler. It mainly includes system lock interrupt time and interrupt execution preparation time.
System interrupt is an important asynchronous event, aiming to improve system efficiency and avoid resource occupation caused by CPU roll polling of some events. In the interrupt driven system, CPU can run normal executive routine. When the input and output equipment requires a service, it will notify the CPU by interrupt and the CPU will make a quick response by means of interrupt service routine.
In the software test process, run the IntrResp_test; install user - defined exception handler and then trigger interrupt to obtain the time from interrupt trigger time to entering exception handler by sys_timestamp, thus getting the interrupt response time under multi-core conditions; and finally take the maximum value of multiple test results.
Operational performance test
Different CPU cores are designed for typical calculation to compare the operation time and test CPU computing performance under multi-core conditions. Test methods include PI operation, n queen algorithm and Fibonacci sequence algorithm.
Linux platform testing tool cpu 2006 is transplanted in the testing process to run in the Reworks operating system, to obtain bzip, specrand, libqua, hmmer and other operation processing time.
Inter-core synchronization time test
Inter-core synchronization of the operating system mainly includes the synchronization of semaphore and mutex and the events are generally of one-to-many. The semaphore means a resource is executed by multiple tasks sequentially. The mutex means a resource is pre-empted by multiple tasks without sequence.
In the test process, run the my_test_performance; by enabling two tasks at the same priority level, task 1 terminates task preemption by calling spin_lock and activates task preemption by calling spin_unlock and task 2 terminates task preemption by calling spin_lock; adopt Testbed RtInsight to obtain the time from task 1 activating task preemption by calling spin_unlock to task 2 terminating task preemption by calling spin_lock, thus testing the inter-core synchronization time under multi-core conditions; and finally take the maximum value of multiple test results.
Memory R/W speed
Program codes are stored in the ROM by pre - allocated address space and no address is re-allocated. Final program load address must be re-allocated when the program is generated. As the data is stored in the RAM, the system needs to respectively position code and data. In addition, read - only data and variable data of the ROM needs to be determined and a number of read - only data is not transmitted to the RAM to save the expensive RAM. Embedded system storage management includes RAM (on-chip RAM, SRAM) management, ROM (MASK ROM, FLASH) management and virtual memory management (only for MMU included MCU/CPU).
During the process of test, adopt mbw test program for tesing of user memory RW speed, including

MEMCPY, DUMB and MCBLOCK (MEMCPY test of fixed block size (default 262144); import mbw engineering and call mbw for testing by the parameter 50 to get operation results.

File R/W rate

The simple embedded system can run well without file system support. But the modern embedded system needs management and transmission of file data on the SSD or USB disk, for which the file system is necessary. The file system shall provide persistent large - capacity data storage management support function and file portability function. This file system could be stored in the flash memory and the bootstrap routine can copy the file system to the RAM for running when the system starts.

An embedded multi-task file system mainly performs the following functions:

- Performing file naming, access, update and data protection.
- Transmit data between the embedded system and other systems, handle multi-user file I/O access and perform file sharing function.
- Be able to logically organize data and take random access data to quickly find information.
- Support device independence (the application program file operation is independent of storage medium and the file operation generates the same result on the compatible device).
- To be compatible with multiple file systems (FAT16,FAT32), standard I/O interface (ANSI C, POSIX or WIN32).
- Support reliable storage on multiple storage media (such as SSD and U disk)

Realization of the file system requires storage of file and directory and storage medium space management. The embedded file system test focuses on performance testing by means of iozone tool.

U disc RW rate:

Adopt performance_test to test the file RW rate respectively by calling iozone("-i 0 r 32k s 32m") and iozone("-i 1 r 32k s 32m")

(-i #n is the specified test item and n is 0 to 12 (0=write/rewrite, 1=read/re-read); -r is the specified file block size, in KB; -s is the specified file size, in MB)）

performance_test: running iozone in C disk:

mount("dosfs","/dev/umass0p1","/c")

sp iozone,"-i 0 -i 1 -r 32k -s 32m"

Graphics performance test

GUI graphics system test adopts universal performance tool DMA. Test methods include:

a) Focus on testing of the time for the system drawing rectangle and line and filling rectangle:

Load gtk_demo_new and call do_canvas_test1 function in canvas.c file to adopt surface->DrawRectangle function on the canvas to get the time for drawing the rectangle.

Load gtk_demo_new and call do_canvas_test1 function in canvas.c file to adopt surface->DrawLine function on the canvas to get the time for drawing the line.

Load gtk_demo_new and call do_canvas_test1 function in canvas.c file to adopt surface->FillRectangle function on the canvas to get the time for filling the rectangle.

b) In addition, memory usage of GUI graphics system is tested. Comparison of memory usage between acceleration and non-acceleration:
Load gtk_demo_new, set do_canvas2 fucntion of canvas.c file (window size 400*300), compile and download it to the target machine, call do_canvas_test2 and mi function to record memory usage after window opening; close 400*300 window, call mi function to record memory usage after window opening; compare memory difference.
Load gtk_demo_new, set do_canvas2 function of canvas.c file (window size 800*600), compile and download it to the target machine, call do_canvas_test2 and mi function to record memory usage after window opening; close 800*600 window, call mi function to record memory usage after window opening; and compare memory difference.
Load gtk_demo_new, set do_canvas2 function of canvas.c file (window size 1024*768), compile and download it to the target machine, call do_canvas_test2 and mi function to record memory usage after window opening; close 1024*768 window, call mi function to record memory usage after window opening; and compare memory difference. Record the difference between opening and closing of 1024*768 window.
Load GTK_DMA, confirm selection of graphics system configuration and display drive in the configuration file, set PRIMARY of df_dok_2.c to 0 (start DMA acceleration) and execute df_dok_demo2（0,0） to obtain test results; and set the PRIMARY to 1 (terminate DMA acceleration) and obtain test results. Obtain data comparison between start and stop of hard acceleration)

Summary

Independent software development is the only route to software development, but independent development and application of the embedded operating system remain at the initial stage. The test technology and method used for the software are continuously improved in the exploration and trial. With more application and promotion of domestic embedded software, quality and test level of China’s independent software will gradually rise. More and better independent application software products will be realized to provide supports for development of China’s software business.

References

[1] Embedded Software Test, Kang Yimei, China Machine Press, 2011;

[2] Embedded Operating System, Ji Jinshui and An Hongxin, Lanzhou University Press, 2009;

[3] Embedded Real-time Operating System, Gong Hui, Science Press, 2003.