Research on Linux Real-time and performance Evaluation for Loongson 3A3000 processor

Xuebing Chen *
Jiangsu Automation Research Institute, Lianyungang, Jiangsu, 222006, China
*Corresponding author’s e-mail: 99091cxb@163.com

Abstract. In order to solve the real-time problem of Linux system for loongson3A, the real-time enhancement design of Linux kernel is carried out by using the idea of PREEMPT_RT patch, the performance analysis support based on ftrace tool is improved, and the real-time optimization method based on reserved core scheduling is proposed. The real-time performance of the system is tested from four aspects: context switching, interrupt response, scheduling latency and network latency. Five context switching methods are proposed, including active switching, priority preemptive switching, semaphores, message queuing and signal distribution latency. A test method based on the internal timer of the PCI board is designed. The test results show that the maximum latency of real-time Linux system based on Loongson3A3000 is less than 30us, the reserved core scheduling latency is the minimum, and the maximum jitter of network communication latency is less than 120us. In conclusion, the system based on loongson3A3000 can meet the demand for hard real-time applications at millisecond level.

1. Introduction
Currently, there are two kinds of hard real-time solutions based on Linux. One is similar dual-kernel solution, which uses Nano or MicroKernel as the real-time kernel and Linux kernel as a non-real-time task, such as rt-linux [1], RTAI [2-3] and Xenomai [4-6]. The other is the kernel PREEMPT_RT patch (also known as RT patch) scheme [7-8], which makes the Linux kernel fully preemptible by modifying it. The main idea of real-time PREEMPT_RT patch is to minimize the non-preemptive area of the kernel as far as possible. The main means include pre-emption of spin lock, real-time mutual exclusion, priority reversal prevention, high-precision clock, etc. By contrast, the latter can fully inherit Linux’s original API and system resources, making user development easier to get started.

The loongson core series CPU has been widely used in many fields. The 3A3000 processor [9] is the latest generation of high-performance quad-core CPU introduced by Loongson Technology Corporation Limited.

In this paper, based on Linux 3.10 kernel, the PREEMPT_RT patch is adapted to the 3A3000 processor + 7A1000 bridge, aiming to provide a real-time and efficient solution for the loongson platform, which can meet the requirements of hard real-time application fields such as industrial control, robot and the like.

2. Real-time optimization method

2.1. Interrupt threading
In Linux system, interrupt takes precedence over task execution. Due to the uncertainty of hard interrupt (off interrupt execution) and soft interrupt execution, the scheduling latency of real-time task is uncertain. To solve this problem, the PREEMPT_RT patches thread hard interrupts and soft interrupts by prioritizing them from the execution context of an interrupt to the execution context of a task, becoming schedulable entities. In this way, real-time urgent tasks can be given higher priority to preempt the execution of interrupt task and reduce the time of system shutdown interruption [10].

But for system clock interrupts and handling very short interrupts, there is no threading. For example, for the cascade interrupt of no_action in loongson3A, the IRQF_NO_THREAD flag is set to prohibit the interruption of threading.

2.2. Spinlock preemptive

There are a large number of spin locks in Linux kernel, and spinlock is a kind of encapsulation of original spin lock (raw_splinlock). Spinlock prohibits preemption by default in the implementation, which greatly affects the real-time response ability of the system. In order to eliminate this non-preemptible area in the kernel, the PREEMPT_RT patch encapsulates the implementation interface of spinlock into preemptible real-time mutex (RT-Mutex), and adopts priority inheritance protocol to prevent priority reversal [8].

However, for the access protection of system critical hardware registers (off interrupt) and non-threaded interrupts, spinlock cannot be converted into RT-Mutex, and directly modified to raw_splinlock, for example, the 7A1000 interrupt controller and CPU serial port driver.

2.3. Ftrace performance analysis tool

With the aid of ftrace [8] in the kernel, it can assist in analyzing the largest shutdown and preemptive area in the kernel. The ftrace tool uses the function of sched_clock() provided by the kernel for sampling timing. The default sched_clock function returns the time unit based on jiffies, and the precision is millisecond level, which does not meet the sampling analysis of microsecond level, as shown in figure 1.

The 3A3000 processor mainly provides three kinds of clocks [9], namely count, node_counter and hpet. The count register is provided for each CPU, and hpet and node_counter are both global clocks. Here, 64-bit node_counter is used for timing, and the overflow of 32-bit counter is not considered. In order to implement the high-precision sched_clock(), it needs to be reloaded as follows:

```c
unsigned long long notrace sched_clock(void) {
    cycles = (node_counter_read());
    return (cycles * clock2ns_scale) >> CLOCK2NS_SCALE_FACTOR;
}
```

Since the kernel does not support floating point operation, to improve the accuracy, clock2ns_scale is based on the following calculation:

```c
ns= cycles / (cpu_freq / ns_per_sec)
    = cycles * (ns_per_sec / cpu_freq)
    = cycles * (10^9 / (cpu_khz * 10^3))
    = cycles * (10^6 / cpu_khz)
    = cycles * (10^6 * SC / cpu_khz) / SC
    = cycles * clock2ns_scale / SC
```

Here, if CLOCK2NS_SCALE_FACTOR is 10 and SC is 1 << CLOCK2NS_SCALE_FACTOR, then clock2ns_scale is equal to (10^6 << CLOCK2NS_SCALE_FACTOR) / cpu_khz. The result of ftrace is shown in figure 2.
2.4. Scheduling Optimization based on reserved CPU

To ensure the response and processing of time-sensitive real-time events in multi-core systems, real-time critical tasks are assigned to the reserved CPUs to prevent them from interfered by other tasks and peripheral interrupts. Scheduling Optimization based on reserved CPU is shown in figure 3.

Before the kernel starts, the reserved CPU number is set through the boot parameter of isolcpus. By default, tasks are not scheduled to execute on isolated CPUs after the system is started. Users can bind real-time critical tasks to an isolated CPU through the taskset command. However, the isolcpus parameter does not isolate interrupts. The interrupt load balancing mechanism is adopted by default in loongson Linux kernel, and the peripheral interrupts are allocated to each CPU in turn for processing. Using the affinity setting between interrupt and CPU, the peripheral interrupt is bound to run on unreserved CPUs by setting the interrupt mask, as shown below:

```bash
for irq in /proc/irq/*; do
    if [ -f ${irq}/smp_affinity ]; then
        echo ${no_isolcpu_mask} > ${irq}/smp_affinity
    fi
done
```

3. Real-time performance evaluation

Based on the literature [8] and [13-15], the performance indexes of real-time evaluation mainly include the following task switching, interrupt response and scheduling latency. There are various reasons for task context switching, such as task sleep or actively abandon scheduling, high-priority preemption of low-priority tasks, task blocking on semaphore, message queue, signal and so on. In order to fully test and verify the system, this paper proposes the above five context switching tests. Considering the application of network communication, the latency test of network communication is proposed.

---

Figure 1. irqsoff latency trace with millisecond level

Figure 2. irqsoff latency trace with microsecond level

Figure 3. Scheduling Optimization based on reserved CPU
The system test is carried out in two cases: one is tested under the condition of no load, the other is tested under the condition of full load. Under full load, multiple processes perform graphics rendering (x11perf), multitask communication (hanckbench [11]), file IO operation (find) and network communication operations (ping). The hardware environment of the test is as follows: 3A3000 processor + 7A1000 bridge, CPU frequency 1.2GHz, memory size 4GB. The rootfs system used is MIPS O32 ABI, which is built on the Yocto framework [12].

3.1. Context switching test

3.1.1. Active switching latency

Two real-time tasks with the same highest priority are initiated, each time task1 switches to task2 by calling sched_yield(), and task2 switches to task1, again by calling sched_yield().

Then the context switching time is (T2-T1) / 2. Test 200000 times under no load and full load respectively, and the results are shown in the figure 4 and figure 5. It can be seen that the maximum is 6us under no load and 12.1us under full load.

3.1.2. Switch latency based on task preemption

Set two real-time tasks, task1 with High priority and task2 with low priority. Task 1 executes first, raising task2’s priority to High+1 on each loop, while task2 continually sets its own priority to low.

The final result is (T2-T1) / 2. Test 200000 times under no load and full load respectively. The results are as follows: the maximum value under no load is 10.8us, and the maximum value is 15us under full load. See in figure 6 and figure 7.
3.1.3. Task switching latency based on semaphore

Set two tasks, task1 with High priority and task2 with low priority. Task 1 executes first, waiting for semaphore in each loop; Task2 releases semaphore in loop. Make sure the semaphore is empty while task 1 waits for it.

Real-time task1 executes:
for (i=0; i<count;i++){
    T1=GetTime ();
    sem_wait ();
}

Real-time task2 executes:
While (1) {
    T2=GetTime ();
    sem_post ();
};

The final result is (T2-T1). The test results were shown in the figure below. The maximum value is 17.7us under no load and 18.2us under full load.

3.1.4. Task switching latency based on message queue

Set two tasks, task1 with High priority and task2 with low priority. Task 1 executes first, waiting to receive messages on the message queue in each loop, task2 sends messages in each loop. Make sure the message queue is empty while task 1 is waiting to receive messages.

Real-time task 1 executes:
for (i=0; i<count;i++){
    T1=GetTime ();
    mq_receive ();
}

Real-time task 2 executes:
While (1) {
    T2=GetTime ();
    mq_send ();
};

The final result is (T2-T1). Test 200000 times under no-load and full load respectively. The test results show that the maximum value is 17.7us with no load and 18.9us with full load. See in figure 10 and figure 11.
3.1.5. Task switching latency based on signal

Set two real-time tasks, task1 with High priority and task2 with low priority. Task 1 execute first, wait for signal in for loop, task2 send signal in loop.

Real-time task 1 executes:
For (i=0; i<count;i++)
    T1=GetTime ();
    sigwait ();

Real-time task 2 executes:
While (1) {
    T2=GetTime ();
    pthread_kill (task1, sig);
}

The final result is (T2-T1). The test results showed that the maximum value is 10.1us under no load and 10.8us under full load. See in figure 12 and figure 13.

3.2. Interrupt response latency test

The interrupt response latency is tested by PCI time board card. Modify the hardware logic of the PCI board so that it periodically produces interruptions. Whenever the interrupt is generated, the internal counter of that board clears 0 and begins to time. By reading the internal counter value T of the time board in the registered interrupt handling function, the interrupt response time is obtained.

The test results are shown in figure 14 and figure 15 respectively. The maximum latency is 17.7us and the average value is 8.8us under no load. The maximum is 27.5us and the average is 10.0us under full load.
3.3. Scheduling latency test

Cyclitest[11] is a test tool in rt-tests, which comprehensively reflects the clock accuracy and scheduling latency of the system. The basic principle of it is:

```c
clock_gettime(CLOCK_MONOTONIC, &t1);
clock_nanosleep(CLOCK_MONOTONIC, TIMER_RELTIME, &interval, NULL);
clock_gettime(CLOCK_MONOTONIC, &t2);
diff = calcdiff_us(t2 - t1 - interval);
```

The Diff value is the scheduling latency. 150000 times are tested with no load, full load and reserved CPU (core 2) respectively, and the test results are shown in figure 16, figure 17 and figure 18 respectively.

The test results show that the maximum latency time of cyclitest without load is 20us, the average is 6.6us and the jitter is 15us. With full load, the maximum is 23 us, the average is 8.2 us, and the jitter is 18 us. The average value of the latter has increased significantly.

When running on reserved CPU, the maximum latency is 10us, the jitter is 4 us, and the average value is the smallest. It can be seen that this kind of real-time situation is the best.
3.4. Network communication latency test

Network communication latency is tested by UDP message. Host A and host B are both loongson3A hosts, which are connected by switches. Test as follows.

Host A obtains the current time stamp T1, organizes it into a message, and sends it to host B; Host B will forward the message to host A immediately after receiving the message. At this time, host A will obtain the local time stamp T2 after receiving the forwarding message, subtracting the time stamp T1 recorded in the message to obtain the latency time of the message back and forth. The length of the above UDP message is 1024. The test is conducted for 200,000 times, and the results are shown in figure 19, with the maximum value of 0.252ms, the minimum value of 0.147ms and the jitter of 105us.

4. Conclusion and Future work

Based on the idea of PREEMPT_RT patch, this paper completed the real-time design of the Linux kernel on the architecture of loongson3A3000+7A1000, and conducted a comprehensive test on the real-time performance of the system from the four aspects of context switching, interrupt response, scheduling latency and network latency.

In context switching test, five scenarios are proposed, including active switching, priority preemptive switching, semaphore, message queue and signal distribution latency. Among them, the overhead of active switching latency is the least.

In the interrupt response test, a test method based on the internal timer of the PCI board is designed, and the results show that the maximum latency of the system is less than 30us even under heavy load. In order to ensure the response and processing of time-sensitive real-time events, a scheduling scheme based on reserved CPU is proposed for multi-core real-time applications.

Finally, combined with the application scenario of network communication, the test is carried out from the UDP message back and forth communication latency. The results show that the maximum jitter of network communication latency is less than 120us, which can meet the demand for hard real-time applications at millisecond level. The work in this paper provides an important reference for the application of loongson3A real-time Linux.

Future work and considerations:

a) There are various reasons for system jitter. In the future, the jitter latency of the system is further analyzed and optimized by combining with performance analysis tools.

b) The current application system used is MIPS O32 ABI, which is worse than MIPS N64 or MIPS N32 ABI. Later we will upgrade the system to MIPS N64.

c) Robot operating system (ROS) is applied more and more widely, but its real-time performance has not been solved effectively [5]. In the future, we plan to further combine the research results of this paper with the ROS communication mechanism, and further verify its application in the field of robotics.
5. Appendices
System load script code:

```
#!/bin/bash
while true; do x11perf -all; done &
while true; do hackbench -l -l -p -g 10 -f 10; done &
while true; do ping -s 10 182.16.20.1; done &
while true; do ping -s 10000 182.16.20.1; done &
while true; do find / ; done &
```

Acknowledgments
This work was supported by the National Key R&D Program of China (No. 2018YFB1305900). In addition, I would like to thank the folks at Loongson Technology Corporation Limited for supporting this effort, notably Yuqian Wang. Further I thank the members of Basic Software Department for discussions and support for this undertaking, notably Zhihong Sun, Guohua Chen and Tao Zhu.

References
[1] Wu, Y.M. (2002) Analysis of real-time mechanism of RT-Linux. Journal of Computer Applications, 22(12): 110-112.
[2] Mantegazza, P., Dozio, L., Papacharalambous, S. (2000) RTAI: Real time application interface. Linux Journal, 2000(72): 10.
[3] Arm, J., Bradac, Z., Kaczmarczyk, V. (2016) Real-time capabilities of Linux RTAI. IFAC PapersOnLine, 25: 401-406.
[4] Gerum, P. (2019) How does Xenomai deliver real-time. http://xenomai.org.
[5] Delgado, R., You, B.J., Choi, B.W. (2019) Real-time control architecture based on Xenomai using ROS packages for a service robot. The Journal of Systems & Software, 151: 8-19.
[6] Su, Y., Zhang, T., Sun, L. (2013) Research on Real-time Linux System Based on Xenomai. Computer Technology and Development, 23(10): 1-5.
[7] Wiki. (2019) Technical details of PREEMPT_RT patch. https://wiki.linuxfoundation.org/realtime/documentation/technical_details/start.
[8] Wu, Z.J. (2010) Research and practice on preempt-RT patch of linux. Lanzhou: Lanzhou University.
[9] Loongson Technology Corporation Limited. (2017) usr manual for loongson3A/3B. www.loongson.cn/uploadfile/cpu/3A3000/Loongson3A3000_3B3000user1.pdf.
[10] Wang, N., Li, Y.F., Sun, F.Y. (2018) The analysis of threaded interrupts and interrupt latency test method. Intelligent Computer and Applications, 8(06): 14-17.
[11] Wiki. (2019) RT-tests. https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/rt-tests.
[12] Linux Foundation. (2019) Yocto Project Quick Build. https://www.yoctoproject.org/docs/current/brief-yoctoprojectqqs/brief-yoctoprojectqqs.html.
[13] Zhang, X.L., Guo, R.F., Tao, Y.D. (2014) Research of linux realtime peemption patch and realtime performance measure. Computer Engineering, 40(10): 304-307.
[14] Wang, S.H., Yang, D.S., Wang, Y.S. (2015) Research and Implementation on RT-Preempt Patch of Linux Based on ARM. Modular Machine Tool & Automatic Manufacturing Technique, 09: 1-4.
[15] Betz, W., Cereia, M., Bertolotti, I.C. (2009) Experimental Evaluation of the Linux RT Patch for Real-time Applications. IN: Proc of IEEE Conference on Emerging Technologies & Factory Automation. Mallorca, Spain, pp: 1-4.