Evaluation of RT-Linux on different hardware platforms for the use in industrial machinery control

Thomas Gisenleitner, DI(FH)
ENGEL Austria GmbH
Ludwig-Engel-Strasse 1, A-4311 Schwertberg
thomas.gusenleitner@engel.at

Gerhard Lettner, BSc
ENGEL Austria GmbH
Ludwig-Engel-Strasse 1, A-4311 Schwertberg
gerhard.lettner@engel.at

Abstract

Using Linux and Open Source in an industrial environment is becoming more and more common, in part to ensure participation with daily improvements and compatibility with future development. One of the most important requirements in the environment of industrial machinery control is realtime, so we decided to evaluate RT-Linux on different hardware platforms. To generate a realistic load which is comparable to the real machinery control a simplified version of machinery control, called testplc, was developed and used in the hardware assessments conducted. The results of this evaluation should give a clear statement about the applicability of each hardware platform for the machinery control area.

1 Introduction

Repeatable cycles is the main goal of our machinery control, so one of the preconditions is an exact and reproducible timing of the scheduler. The machinery control in real is a bit too complex for this evaluation so we decided to implement a little test control application with a comparable core design.

To get a concrete appreciation of the interaction between RT-Linux, CPU, chipset and board design we evaluate the same application with the same RT-Linux kernel on different hardware platforms.

2 Method

To get a wide range of results we take some really different hardware platforms from old industrial over modern energy saving up to high performance and consumer electronics.

In this environment its impossible to take the real control application for evaluation on these different hardware platforms, the used test control application is explained in the next section.

The used operation system is based on debian lenny, the kernel version is 2.6.33.9\(^1\) which is patched with the realtime patch rt31\(^2\). More details see chapter 2.2.

The aim of this evaluation work is to get more into details using RT-Linux for industrial machinery control. It’s very interesting to implement a small test control application designed towards the real control application. The concrete results give us a better understanding which influences lead to a scheduling malfunction and the properties of multi core cpus are displayed in the latency histograms.

Indeed a machinery control application in real

\(^1\)http://www.kernel.org/pub/linux/kernel/v2.6/longterm/v2.6.33
\(^2\)http://www.kernel.org/pub/linux/kernel/projects/rt/
life has to deal with locking, interrupts, priority inheritance and other complicated stuff, but if this simple test control application shows problems in scheduling and runtime a real machinery control application will never work.

2.1 Test Control Application

The test control application is implemented in pure C with posix threads. Five of these threads build the core of this control application with different cycle times and priorities to simulate the real machinery control. The timing is controlled with absolute timestamps starting with a global start time. In every cycle the period time is added to the absolute time. Each thread greps the actual system time after returning from clock_nanosleep before it calculates the difference between scheduled time and the actual time. We call this difference latency and inserted the value in a histogram. The parameters for clock_nanosleep are clockId=CLOCK_MONOTONIC and flags=TIMER_ABSTIME. For security reasons we introduce a latency threshold to detect a scheduling fault.

Every thread has a number of functions to execute, for example calculate 100 double multiplications, 500 double multiplications, sorting lists, calculating pi and communicate 1024 bytes over udp to a server. The udp communication is only done in the 10000 micro second cycle time task to avoid network problems. This communication generates a huge amount of interrupts to stress the scheduler.

For measurement we take the time of 500 double multiplications in nano seconds which are also stored in a histogram. The timing and priority configurations of these threads are

1. 100 us cycle time with priority 80
2. 500 us cycle time with priority 75
3. 1000 us cycle time with priority 70
4. 2000 us cycle time with priority 65
5. 10000 us cycle time with priority 60

This test control application gives us two histograms as result. The first one is a latency histogram of all five threads within a range of 0 to 300 micro seconds. The second one is a timing histogram of the 500 double multiplications within a range of 0 to 20000 nano seconds.

2.2 RT-Linux

To get comparable results it's evident to use equal Kernel configurations for the evaluation on each hardware platform. As already mentioned in chapter 2 the used Kernel was patched using the required RT-Patch which is available at [www.kernel.org](http://www.kernel.org). To ensure the real time behaviour the kernel configuration (see section 2.2.1) as well as the runtime configuration (see section 2.2.2) is important and was adopted.

2.2.1 Kernel configuration

The following items in the Kernel configuration were considered:

**Processor type and features**

- High Resolution Timer Support
- Symmetric multi-processing support
- Preemption Mode
- Complext Preemption (Real-Time)
- Generix x86 support
- Timer frequency
  - 1000 HZ

**Power management and ACPI options**

- Power Management Support
- CPU Frequency Scaling

**Kernel Hacking**

- Tracers
  - Scheduling Latency Tracer
  - Scheduling Latency Histogram
  - Missed Timer Offset Histogram

2.2.2 Runtime configuration

To ensure real time behavior during runtime, the Real-Time group scheduling must be modified. Therefore the content of the files `/proc/sys/kernel/sched_rt_period_us` and `/proc/sys/kernel/sched_rt_runtime_us` has to be set equal. The standard content for `sched_rt_period_us` is 1000000 (1s) and for `sched_rt_runtime_us` 950000 (0.95s). Using the standard settings causes to give the non real time tasks a chance in 5% of the CPU time, if a real time task would lock the whole CPU. The described modification has the effect that the real time tasks
get the whole CPU time and so can eventually lock
the system so that no more non real time task can
run. Further details can be found in the Kernel
documentation [3].

3 Results

In the following sections we list the results and de-
scriptions of the used test systems. The following his-
tograms show the latency and multiplication times
in micro seconds on the X axis and the number of
occurrences on the Y axis in a logarithmic scale.

The following figures (Figure 2, 4, 6, 8) with the
multiplication times show the variation of calculation
time in the different realtime tasks and illustrate the
influence of the thread priority.

3.1 Testsystem Z530

3.1.1 Description

The testsystem Z530 is based on a Kontron main-
board with an Intel(R) Atom(TM) CPU Z530
clocked with 1.60GHz and two giga bytes of mem-
ory.

3.1.2 Result

The testplc takes about 80% of the whole CPU per-
formance. In this case the latency times vary over
more than 100 micro seconds, the peaks look like
serialized tasks in order to the configured priority.
Maybe one of the main reasons for this picture of
latency times in Figure 1 is the poor performance
and the missing ability for parallel computing. The
first three, highest priority tasks do not exceed the
maximum worst case latency of 300 micro seconds,
the 2000 and 100000 micro second task exceed the
worst case latency of 300 micro seconds very often,
as shown in the histogram on the right peaks where
the overruns are cumulated.

3.2 Testsystem T7500

3.2.1 Description

The testsystem T7500 is based on a Kontron main-
board with an Intel(R) Core(TM)2 Duo CPU T7500
clocked with 2.20GHz and three giga bytes of mem-
ory.

3.2.2 Result

The testplc takes about 15% of the whole CPU per-
formance. In this case the latency times as shown in
Figure 3 do not vary much and concentrate within 10
micro seconds. In this plot we see the ability of par-
allel computing, the two fastest tasks set their cpu
affinity so that each of them use one core.
3.3 Testsystem D525

3.3.1 Description

The testsystem D525 is based on a Gigabyte GA-D525TUD mainboard with an Intel(R) Atom(TM) CPU D525 clocked with 1.80GHz and two giga bytes of memory.

3.3.2 Result

The testplc takes about 32 % of the whole CPU performance. In this case of Figure 5 the latency times cover a range up to 200 micro seconds. Only a view peaks are detected which exceed the 300 micro seconds.

3.4 Testsystem CP255

3.4.1 Description

The testsystem CP255 is based on a KEBA mainboard with an Intel(R) Pentium(R) M processor clocked with 1.40GHz and one giga bytes of memory.

3.4.2 Result

The testplc takes about 25 % of the whole CPU performance. In this case the latency times in Figure 7 of the three highest priority tasks do not vary much and concentrate within 30 micro seconds. The lower the priority the higher the variation in the latency time, but even the lowest priority tasks latency so far stays below 150 micro seconds.
The results in Figure 7 were produced using a vendor specific rt-kernel based on version 2.6.33.9.

3.5 Testsystem OMAP4

3.5.1 Description

The testsystem OMAP4 is based on a OMAP4 Panda board with an ARMv7 Processor rev 2 (v7l) processor clocked with 1.00GHz and one giga bytes of memory.

3.5.2 Result

The testplc takes about 75 % of the whole CPU performance. In Figure 9 the latency times of the three highest priority tasks vary much and reach 300 micro seconds as well as the lower priority tasks. This behaviour shows that the realtime capability is not given for this testsystem.

4 Conclusion

The results of the recorded histograms show the different real time behaviours of the evaluated hardware platforms. For the Z530 and OMAP testsystem the real time capability can’t be confirmed as the latency time shows a variation more than 300 micro seconds. In contrast to the testsystem Z530 the histograms from the testsystem T7500 show a very small range of latency variation of only 40 micro seconds for the tasks with the lowest priority. Although the testsystem CP255 is optimized for realtime, the results are not as good as the values from testsystem T7500. We assume the cpu performance is the reason for this behaviour. The four higher priority tasks show only a variation of 15 micro seconds. As the implementation considers the requirements of a real machine control and the evaluation gives a good overview about the realtime capability, we consider the acquired evaluation method as an extension to the OSADL QA farm [2] evaluation.
References

