Linux scheduling in embedded systems

Seminar paper

The final version of the seminar paper is here: scheduling_final.pdf
And the presentation is here: presentation.pdf
Source code for the experiment is here: rt_driver.zip

Scheduling in Linux kernel

The default model for scheduling processes for use of the CPU is round-robin time-sharing. This scheduling satisfies two important requirements for interactive multitasking system /2/:

- Fairness, each process gets a share of the CPU
- Responsiveness, a process doesn't need to wait too long to get a share of the CPU

MORE INFO: nice values, kernel configuration, POSIX real-time

What is real-time?

In /10/ real-time system is defined as follows:

"A real-time system is one in which the correctness of computations not only depends upon the logical correctness of the computation but also upon the time at which the result is produced. If the timing constraints are not met, system failure is said to have occurred."

Real-time implementations are usually divided into two categories: hard real-time and soft real-time /1/ and /2/.

Soft real-time system is defined as follows in /11/:

A soft real-time system is one in which performance is degraded but not destroyed by failure to meet response-time constraints.

Hard real-time system is defined as follows in /11/:

A hard real-time system is one in which failure to meet even a single deadline may lead to complete or catastrophic system failure.

Real-time applications are used in, /10/ and /11/:

- the measurement and control industry
- the aerospace industry
- the financial service industry
- the multimedia business
- the medical business

Real-time in Linux

Real-time behaviour in Linux is implemented either using CONFIG_PREEMPT_RT patch set which allows nearly all of the kernel to be preempted, /3/ and /4/. Another way of achieving real-time performance in Linux is to use a dual kernel approach. The real-time microkernel runs Linux as a fully preemptive process. The following real-time implementations use this approach:

- RTAI - Real Time Application Interface, /5/
- RTLinux - Real-Time Linux, /6/ and /7/
- Xenomai - Real-Time Framework for Linux, /8/

Definitions for real-time concepts: deadline, latency, jitter, predictability, periodic task /1/, /10/ and /11/.

Also POSIX real-time vs. hard real-time in /2/.

Testing

I downloaded a ready made image including Linux kernel patched with Xenomai /9/. The only thing missing was user space support for Xenomai. After downloading and compiling the newest Xenomai version, I was able to run initial tests. Running an 1 ms periodic task for 15 minutes shows that the minimum latency was 2 us and the worst was 63 us. The average latency was 19 us.
The 433 MHz transmitter and receiver pair turned out to be a little more challenging to get working. My initial goal was to simply use a function generator and an oscilloscope to just make sure that the RF-link is working. The receiver is however outputting data all the time, even when the transmitter is powered down. I'm assuming that the output should be 0 when there is no carrier and 1 when the transmitter is actually transmitting. The receiver seems to be picking up some random noise and outputting just that. I found some discussions mentioning receiver instability and the solution was to add some filtering to the power supply of the receiver. I did that (capasitors + common mode choke) but it didn't help.

Hello World!

First version of a user space program is working. It is blinking a LED. The real-time task sets GPIO pin to "1" for 220 us and then switches it off. This is done 500 times per second. The measured pulse width is 225 us and there is a 27 us variation in pulse width. The interface to GPIO pins uses memory mapped IO /12/.

This screenshot was took after compiling the test program a few times thus keeping the RPi busy. When the RPi is idle the variation in pulse width is about 15 us. The oscilloscope used for the measurements was a BitScope BS310N.

More blinking

Switching light bulbs on and off using the transmitter and remote controlled sockets is now working. The transmitter is connected to a GPIO pin and the bit sequence to control these sockets is generated using real-time timers. These sockets are divided in four address groups and in each group there can be three units. There is a configuration switch on each socket to select the addressing. The real-time task opens a pipe and by writing to the pipe it is possible to control these sockets. To switch unit A 2 on, write 'a21' to the pipe. And to turn it off write 'a20'.

More testing

- try to get the RF link to work
- write some user space program to utilize real-time
- implement the message structure and verify that the timing is correct
- use the message structure to form a meaningful message to be sent to the receiver
- try to apply real-time patches to vanilla kernel and make it work instead of using ready made image

References

/5/ RTAI - Real Time Application Interface Official Website. https://www.rtai.org/
Intended table of contents:

ABSTRACT

1. INTRODUCTION
   • scheduling in Linux kernel
   • what is working well and are there special cases when the basic scheduling isn’t enough?
   • which leads to introduction to real-time
   • why is real-time needed
   • real-time use cases

2. Real-time implementations in Linux
2.1. Definition of real-time
   • what is not real-time
   • soft real-time
   • hard real-time

2.2. Different approaches for achieving real-time in Linux
   • CONFIG_PREEMPT_RT patch set
   • POSIX real-time
   • micro kernel approach

2.3. Available hard real-time solutions
   • RTAI
   • RTLinux
   • Xenomai

2.4. Expectations
   • performance, both timeliness and response times
   • maybe comparing x86 and RPi architectures

2.5. Roadmap and/or future trends
   • what’s next?
   • is there something new and revolutionary in the horizon?

3. Building real-time environment
3.1. Tools needed
   • toolchain to compile kernel
   • realtime “package”

3.2. Building options
   • kernel configuration options

3.3. Testing the installation
3.4. What services or APIs are available

4. Example application utilizing real-time
4.1. Goals for the application
   • use real-time API
   • make measurements about timing accuracy

4.2. Description of the application
   • sending commands to remote controlled sockets
   • transmitter timing is implemented using real-time task by directly controlling GPIO pin

4.3. Used real-time APIs and methods
- different priorities
- timers
- watchdog

4.4. Results

- some kind of an evaluation how much work was needed to set up a test application
- was the achieved timing accuracy good enough?

5. CONCLUSIONS

6. REFERENCES