Migrate a RTAI Application to RT-Preempt

-

A case study

linu
tronix

Initial position

- Linux-2.4.26 + RTAI
- Existing Application (Automated Test Equipment)
  - Audio stimulation
  - Frequency generator (1 - 1000Hz)
  - Power fail scenarios
  - Serial data stream capture and analysis

Goal

- Linux-2.6 + Realtime Extension
- Make the application code more modular and portable. Code must be usable on an unmodified kernel
  - ➔ Application Programming Interface: POSIX

Which real-time extension to use ?

Available choices:

- RT-Linux
  - Excluded due to licensing
- RTAI/Xenomai
- RT-Preempt

Xenomai vs. RT-Preempt

Xenomai:

- Dual Kernel approach
- Separate libraries
- RTAI migration skin allows easy transition of the existing code

RT-Preempt

- Single Kernel
- Standard glibc (+ patches)
- Reimplementation of the code necessary

Xenomai first choice ?

- POSIX interface not fully implemented
- One to one usage of existing code did not work out
- Adds 300k+ binary code size and 250k+ data size to the kernel
- Code has to be modified / recompiled to use on vanilla Linux
- Unclear project situation after the RTAI split

What about RT-Preempt ?

- POSIX interface fully implemented
- Small increase in kernel size (36k code, 15k data)
- Strong and mainline visible development
- Code runs unmodified on vanilla Linux
- Performance evaluation positive

How to migrate smoothly and fast ?

Stub device drivers implemented first, so application and device driver development can go in parallel

Userspace:

- Complete redesign of the application
- Reuse of code evaluated
- Modular functionalities implemented in parallel

Kernel:

- Usage of a framework for industrial I/Os
- Driver implementation restricted to absolute necessary low level functions

What's the result ?

Two weeks of implementation time. (4 engineers)

- Functional prototype
- Shiny new modular design
- Impressive decrease of code lines:

Original RTAI code: 12200

New implementation: 7300

Performance results

Environment:

- Pentium M 1,4GHz
- Custom made interface hardware (Audio stimulation, Frequency stimulation, Fieldbus interface)

Performance results

Audio: Interrupt driven refill of the D/A buffer (Period: 720us)

Hardware

ISR (Low level ACK)

Wakeup UT

mmap

Buffer Refill Task

- Fill FIFO with pre calculated values
- Calculate next buffer or read from mmapped file

Maximum Latency

(Hardware interrupt -> Last FIFO entry written): 220us

Performance results

Frequency stimulation: 1-8 Timers toggle an output pin

Hardware

Timer callback

Driver

High Resolution timers

Frequency Control Task
- Command pattern processing

Frequency jitter: max. 60µs

Performance results

Data stream capture from the device under test

Hardware

ISR (Low level ACK)

Wakeup UT

mmap

Buffer Readout Task

- Read FIFO
- Store values into a Ring Buffer

Signal

Data processing task

- Read and process values from the Ring Buffer
- Store result

Performance results

Data stream capture from the device under test

Data Rate: 115200 Baud

Interrupt Rate: 1.3 ms

Maximum Latency (Interrupt -> Readout last byte): 380us

Ringbuffer size: 256kiB

Max. Filllevel: 160 kiB

Is Preempt-RT production ready ?

It depends.

- The development process has stabilized
- Productized versions are necessary (not every single -rt release is usable)
- Used already in products:
  - Laser control
  - Wood working machines (multi axis servo control)
  - Soft-PLC (ARM, PPC based)