

Open-Source Hard Real-Time for Linux

Nicholas Mc Guire

OpenTech EDV-Research GmbH

Mistelbach - AUSTRIA

www.opentech.at

Schedule

- What is real-time
- The RTOS design dilemma
- Open-Source real-time extensions available for Linux
- hard real-time for Linux core concepts
- example code - its really easy !
- where to get it - how to contribute.

What is real-time

Classes of real-time systems

- Hard Real-time
- Soft Real-time
- Non Real-time

Hard real-time

- guarantee every deadline is met
- deterministic system
- temporal behavior not load dependency
- very limited access to dynamic resources
- High resolution timers

Soft real-time

- gives statistic guarantees
- Quality guarantees (QOS)
- dynamic resources can be used with care
- High resolution timers
- low average jitter and interrupt response

Non real-time

- Absolutely no guarantees at all
- Delays are load dependant
- dynamic resources are fully available
- timers are coarse grain

Typical Timings

- Multimedia data 10ms frame rate in video
- Networking 1us frame gap on gigabit Ethernet
- Time management 1s - 10us global time sync
- Control systems 10s - 1us
- Telecom systems 10ms - 10us

General performance notes

- hard real-time comes at a cost
- Optimizations and real-time demands
- Complexity of programming
- Resource management becomes complex

Non real-time might do

- If throughput is your main concern
- You can live with good average performance without guarantees
- Missing an event from time to time will not kill any body
- Low time resolution does not cause any problems for your application

Hard real-time is a requirement

- If you need high time-stamp precision
- If missing an event will cause problem
- If time resolution below the millisecond is needed
- The correctness of results depends on time as well as value

The RTOS design dilemma

Determinism and general purpose OS demands contradict each other

- Only a small and simple OS will be deterministic
- Users want TCP/IP CUI and compatibility with a desk-top system

Why not make a general purpose OS into a RTOS ?

Why can't one simply add all the real-time features to a GPOS ?

- GPOS are event driven not time triggered
- Introducing determinism means modifying the kernel AND applications
- Maintenance costs of such a system would be too high and offer little utility for most users
- No simple integration of existing applications possible

Expand a RTOS to become a GPOS ?

So lets simply put all the features requested into a small and deterministic RTOS ?

- Core capabilities would need to be preemptable
- optimizations would be restricted - no "large" atomic instructions
- priority inversion issues
- Existing software packages would again need modification at the source level

Software validation issues

- error containment
- temporal debugging
- modularity issues
- system level analysis

Dual kernel approach

A possible solution to the above dilemma is to use a dual-kernel approach

Dual kernel RTOS/GPOS concept

Have a small deterministic RTOS with full hardware control coexist with a non-deterministic GPOS striped of any direct hardware control run on one platform.

Hard real-time

Open-source Dual kernel solution available

- RTLinux/GPL
- RTAI/RTHAL
- RTAI/ADEOS

Open-source Soft real-time solutions

- Librtos (let see how it does ...)
- preemptive kernel
- low latency patches

Addon components

Addon components available for hard real-time variants

- RTNet - real-time networking on Ethernets
- Comedi - data acquisition board support
- RT-safe libm - real-time safe math library

hard real-time for Linux

- disable interrupts in the GPOS
- interrupt emulation through the RTOS
- run Linux as the RTOS idle task
- manage resources in the GPOS

Interrupt abstraction

```
if(there is a RT-handler)
    execute it immediately
else if(there is a Linux kernel and no RT-request)
    execute it
else
    mark the interrupt pending
```

High resolution time

- directly use hardware timers in one-shot mode
- timer granularity in the nanosecond range
- time stamp precision in the low microsecond range

Scheduling

- priority driven preemptive scheduling
- POSIX SCHED_FIFO policy
- Run the GPOS as the idle task of the RTOS
- CPU utilization is not the core issue as the idle task consumes the rest

Resource management

- Let the GPOS do all the messy jobs (pci-initialization, etc.)
- statically allocate all resources in GPOS context and then use them in the RTOS

RTLinux/GPL

RTLinux/GPL is one of the dual-kernel implementations of hard real-time for Linux.

- RTLinux/GPL is a hard real-time extension to Linux
- RTLinux/GPL follows the POSIX PSE51 1003.13 RT profile
- Its a small deterministic layer below Linux
- RTLinux/GPL manages the resource time and CPU - thats it.

<http://www.rtlinux-gpl.org>

What happens to Linux in RTLinux/GPL ?

- Linux is only a thread managed by the real-time scheduler
- RTLinux/GPL runs Linux as its idle task by default
- Linux is fully preemptable from rt-context

What does RTLinux/GPL provide to Linux ?

- A small resource constraint but deterministic processing context
- High resolution timing functions
- Advanced interrupt control functions
- basic IPC is provided to Linux to allow communication with rt-tasks

RTLinux - hello.c

A simple example - lets run "hello world!" in real-time

- `init_module` - set up all resources in Linux
- `thread_code` - the rt-executable
- `cleanup_module` - manage the clean shutdown and release of resources in Linux again.

init_module

```
int
init_module(void)
{
    return pthread_create (&thread, NULL, start_routine, 0);
}
```

```
void * start_routine(void *arg) {
    struct sched_param p;
    hrtime_t next = clock_gettime(CLOCK_REALTIME) + 1000000000;
    p . sched_priority = 1;
    pthread_setschedparam (pthread_self(), SCHED_FIFO, &p);

    while (1) {
        clock_nanosleep (CLOCK_REALTIME, TIMER_ABSTIME,
            hrt2ts(next), NULL);
        next += 500000000;
        rtl_printf("I'm here; my arg is %ld\n", (long) arg);
    }
    return 0;
}
```

cleanup_module

```
void
cleanup_module(void)
{
    pthread_cancel (thread);
    pthread_join (thread, NULL);
}
```

Where to get it

- www.rtlinux-gpl.org
- www.rtai.org
- home.gna.org/adeos
- kernel preemption in the 2.6 kernels at kernel.org

How to contribute

- Use free software !
- help out on the mailing list - answer questions
- expand the code base - make your project open-source
- Support the free-software-foundation www.fsf.org

Acknowledgment

The dual kernel concept was proposed by Victor Yodaiken and first implemented by him together with Michael Barabanov in 1996 at the University of New Mexico, NM, USA. These foundations have become the core of all existing hard real-time dual kernel variants for Linux to date. Unfortunately the original authors of the RTLinux core have decided no longer to support the open-source model in 2001.