

RTLinux Manifesto

Victor Yodaiken*

Presented by

Morgan Deters

`mdeters@cs.wustl.edu`

`www.cs.wustl.edu/~mdeters/projects/seminar/`



Washington
University in St. Louis

Fall 2001 Seminar on
Real-Time Programming Languages

26 September 2001

* Work of Michael Barabanov is also presented

Introduction

- *Abstract*

RTLinux is the *hard realtime* variant of Linux that makes it possible to control robots, data acquisition systems, manufacturing plants, and other time-sensitive instruments and machines.

- *Applications*

- NASA
- Jim Henson Creature Shop
- video editors
- PBXs
- robot controllers
- medical*

* *No warranty express or implied*

RTLinux

- RTLinux extends UNIX to realtime
- RTLinux interrupt handlers communicate with non-realtime Linux processes via
 - device interface/FIFOs
 - shared memory
- “[O]bvious methods—like making the standard kernel directly support realtime—are doomed to failure.”
- RTLinux is for hard realtime programs at present
 - Will support soft realtime systems in the future
- $< 15\mu\text{s}$ interrupt latency ($600\mu\text{s}$ under standard Linux)
- $< 35\mu\text{s}$ periodic scheduling latency (20ms under Linux)

Realtime Systems

- Realtime categories
 - *non*-realtime - GUI's, etc.
 - *soft* realtime - video displays
 - *hard* realtime - rocket shutdown sequence
- Hard realtime systems cannot use average case to compensate for worst case
- Traditional (old) realtime systems were hand made
 - Often tight control loops
 - This does not scale
 - A nightmare to maintain for complex systems
- RTLinux relegates non-realtime “fast” applications to Linux

“You Can Put Racing Stripes on a Bulldozer, But it Won’t Go Any Faster”

- Linux + POSIX 1003.13 “Multi-Purpose Realtime System Profile” (PSE54) not precise enough
- General purpose OSes optimize the common case
- Linux is in contradiction to realtime
 - Coarse-grained synchronization around core data structures
 - Fair scheduling is unfair for realtime – high-priority tasks should never wait for lower-priority tasks
 - Efficient use of hardware – *e.g.*, optimized disk access (*w.r.t.* the disk)
 - Lazy batching of operations – *e.g.*, clearing out multiple memory pages at once

On Racing Stripes and Bulldozers (continued)

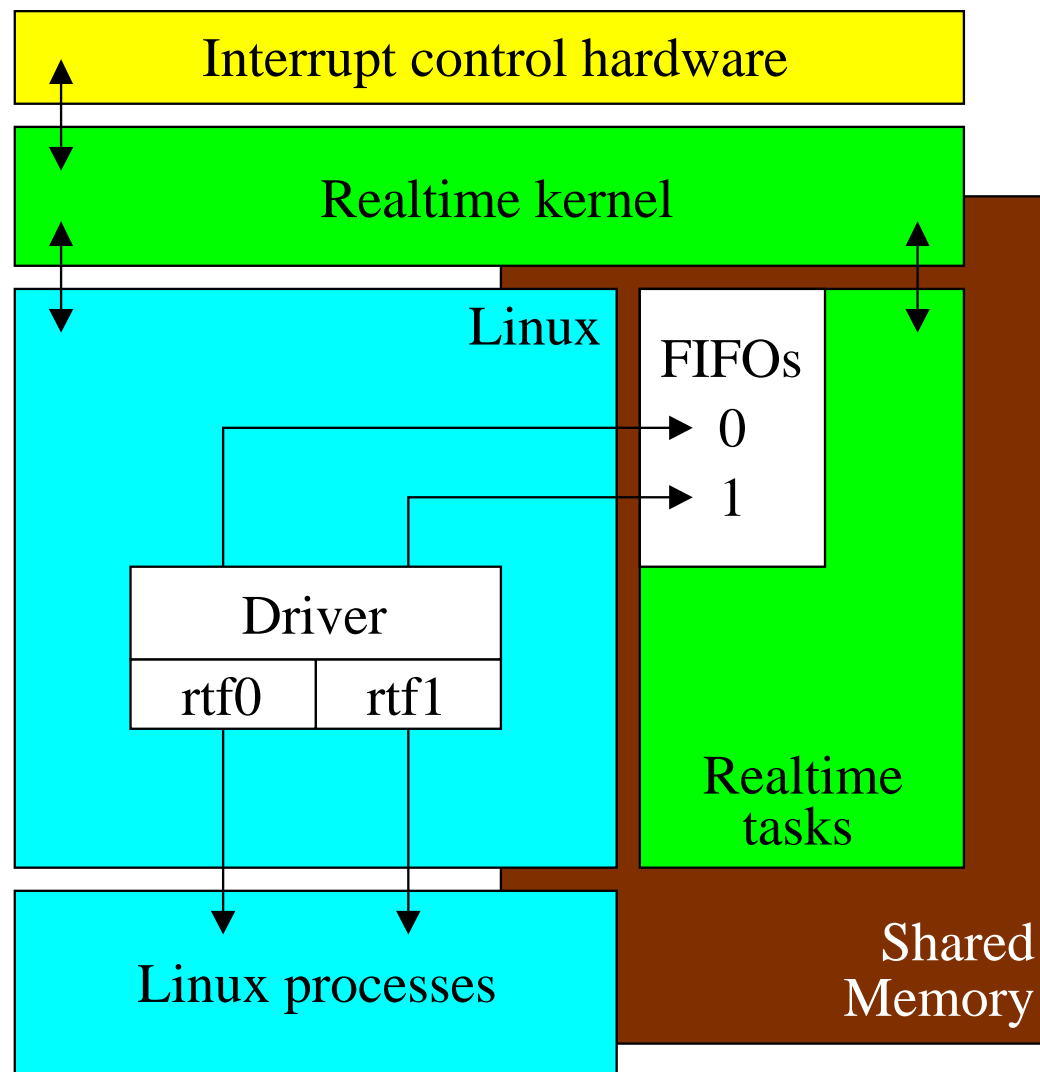
- Linux is in contradiction to realtime *continued...*
 - System calls are not preëemptible... even in low-priority tasks
 - High-priority tasks must wait for low-priority tasks to release resources
- There are OSes that mix realtime and non-realtime (Solaris RT)
 - These are “complicated and quite slow”
- Design goals for general purpose OSes and realtime OSes are contradictory
- Therefore, Linux can't do both realtime and general purpose well

Enter: *RTLinux!*

(applause!)

- Based on *MERT* design of decoupled but cooperative realtime and general purpose OSes running in parallel
 - Linux is an idle task under a small realtime OS
- Software emulation of hardware interrupts – CLI & STI
- The “enslaved” Linux never actually disables or enables hardware interrupts
 - * Requests are governed by realtime kernel
 - * Non-realtime (Linux) ISRs are only executed when no realtime action is pending
 - * No matter what Linux is doing, it’s preemptible
 - * No matter how hard it tries, Linux *cannot* add latency to RTLinux interrupt response time

RTLinux: *A Bird's Eye View*



Using RTLinux

- **Module-oriented** – load kernel modules for what you want, including scheduler & FIFO operations
- **pthread interface with some non-portable extensions**
 - `int pthread_create(pthread_t* thread, pthread_attr_t* attr, void*(*start_routine)(void*), void* arg)`
 - `int pthread_make_periodic_np(pthread_t thread, hrtime_t start_time, hrtime_t period)`
 - `int pthread_suspend_np(pthread_t thread)`
- **Realtime FIFO interface**
 - `int rtf_create(unsigned fifo, int size)`
 - `int rtf_get(unsigned fifo, char* buf, int count)`
 - `int rtf_put(unsigned fifo, char* buf, int count)`

Hello, Periodic World!

```
#include <rtl.h>
#include <time.h>
#include <pthread.h>

pthread_t thread;
void* start_routine(void* arg) {
    struct sched_param p;
    p.sched_priority = 1;
    pthread_setschedparam(pthread_self(),
        SCHED_FIFO, &p);
    pthread_make_periodic_np(pthread_self(),
        gethrtime(), 500000000);
    while(1) {
        pthread_wait_np();
        rtl_printf("I'm here; my arg is %x\n",
            (unsigned)arg);
    }
    return 0;
}

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

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

- This code is compiled as a kernel module
- `init_module()` is executed *in a Linux kernel thread* when the module is loaded and spawns a new RTLinux thread
- `start_routine()` runs *in an RTLinux thread* and sets up periodic execution, then blocks
- `pthread_wait_np()` returns at the start of each 500-millisecond period (the task runs at 2 Hz)
- The `rtl_printf()` call prints “I’m here; my arg is 0\n” to the kernel ring buffer
- `cleanup_module()` is called when the module is unloaded

Some Remarks (What the Manifesto Doesn't Tell You)

- NetBSD also supported as a slave general purpose OS
- Alpha and PowerPC ports (even SMP) now available
- Version 3.0 out since February 2001
- “[Priority inheritance] is a dangerous method that we do not support in RTLinux.” –RTLinux FAQ
- You must be root to run realtime programs
(insmod/rtdlinux)
- Because all realtime tasks execute in kernel space, a buggy program can easily bring down the system or cause data loss (!)

Technically Speaking...

- CLI, STI, & IRET instructions replaced with S_CLI, S_STI, & S_IRET macros that implement *soft interrupts*
- Realtime kernel modules are actually the second design—originally arbitrary ELF executables were run in realtime, each in their own memory page, but this suffered from severe performance problems
 - Invalidation of the cache's translation lookaside buffer (TLB) for each realtime context switch
 - System call overhead: trapping to a more privileged level is expensive
 - These problems are addressed by placing all realtime tasks in the kernel space

Still Technically Speaking...

- Different schedulers available
 - Basic scheduler is a priority-based preëmptive scheduler
 - * Each task has a unique priority
 - * Handles periodic tasks
 - * Linux is the lowest priority realtime task
 - * Programs clock in one-shot (slow on x86) or periodic mode
 - Rate-Monotonic scheduler (RMS)
 - Earliest Deadline First (EDF) scheduler
- Schedulers are implemented as kernel modules
- Can “easily” implement other schedulers

Finè

RTLinux Manifesto:
The Continuing Discussion...

`www.cs.wustl.edu/~mdeters/projects/seminar/`



Washington
University in St. Louis

Fall 2001 Seminar on
Real-Time Programming Languages

26 September 2001