Real-Time Linux

Christopher Schuster

Project Seminar: Distributed Real-time Systems (summer term 2010)
Prof. Dr. Andreas Polze, Jan-Arne Sobania, M.Sc., Uwe Hentschel, M.Sc.
Hasso-Plattner-Institut, Universität Potsdam

June 22, 2010
Outline

1. Introduction
Outline

1. Introduction
2. History
Outline

1. Introduction
2. History
3. RTCore
Outline

1. Introduction
2. History
3. RTCore
4. PREEMPT_RT
Outline

1. Introduction
2. History
3. RTCore
4. PREEMPT_RT
5. Demo
Introduction

Motivation

- Complex embedded systems
- Re-use of Linux tools and libraries
- Professional low-latency audio/multimedia
- Mixture of real-time tasks and high-level code like user interfaces, network protocols or databases
## Introduction

### Motivation
- Complex embedded systems
- Re-use of Linux tools and libraries
- Professional low-latency audio/multimedia
- Mixture of real-time tasks and high-level code like user interfaces, network protocols or databases

### Implementations
- RTLinux (1996)
- RTAI (1999)
- and several others
Origin of RTLinux

- Real-time operating systems research at New Mexico Institute of Mining and Technology
- Free (GPL) real-time implementation
- Michael Barabanov
  - Did most of the conceptional work and implementation
  - Resulted in his Master’s Thesis in 1996
- Victor Yodaiken
  - Academic advisor
FSMLabs founded in 1996
   - Victor Yodaiken (CEO)
   - Michael Barabanov (Principal Engineer)
   - Cort Dougan (Director of Engineering)

US Patent issued in 1997

Renamed to RTCore in 2003 and release of RTCore/BSD

RTCore technology was sold to Wind River Systems in 2007
RTAI

Origin of RTAI

- Real-time services for DOS written at DIAPM
- DIAPM-RTOS Services ported to Linux on top of RTLinux

1Dipartimento di Ingegneria Aerospaziale - Politecnico di Milano
Origin of RTAI

- Real-time services for DOS written at DIAPM\(^1\)
- DIAPM-RTOS Services ported to Linux on top of RTLinux

**Feb/Mar 1999**

- Linux 2.2.x.x introduces Symmetric Multi-Processing (SMP)
- DIAPM-RTLinux wants to adopt early
- Very slow progress with RTLinux

\(^1\)Dipartimento di Ingegneria Aerospaziale - Politecnico di Milano
Origin of RTAI

- Real-time services for DOS written at DIAPM\(^1\)
- DIAPM-RTOS Services ported to Linux on top of \texttt{RTLinux}

Feb/Mar 1999

- Linux 2.2.x.x introduces Symmetric Multi-Processing (SMP)
- DIAPM-RTLinux wants to adopt early
- Very slow progress with RTLinux

Apr 1999

- First release of DIAPM-RTLinux fork with SMP support
- Renamed to \texttt{RTAI}

\(^1\)Dipartimento di Ingegneria Aerospaziale - Politecnico di Milano
Problems in Adoption

- Problems with RTLinux patent
  - Patent promise for GPL code, but RTAI was striving for commercialization
  - Switched to Adeos kernel patch in 2004
- Dispute on Linux Kernel Mailing list in 2002
  - RTAI was pushing patches to mainline Linux
  - Linux developers not very “excited” about the intrusive approach of RTAI
PREEMPT_RT patch

- Low Latency Patch originally by Ingo Molnar in 2000
- Intensive work started on Linux 2.6.9 in 2004
- Continuously pushing small patches to improve real-time and multi-processor behavior to mainline Linux
- Still moving -rt into mainline
  - Initial roadmaps targeted 2.6.32
  - Perhaps one of the next versions
Dual kernel approach

- **Real-time microkernel** for interrupts handlers and scheduling
- **slightly customized Linux Kernel** that cannot disable interrupts anymore

Disabling interrupts in Linux sets flag in microkernel

Real-time tasks might still get those interrupts

Linux can only run and process pending interrupts if no real-time tasks runnable
Linux Process 1 ➔ Scheduling, System Calls, etc. ➔ Linux Kernel ➔ Software Interrupts ➔ Real-Time Kernel ➔ Hardware Interrupts ➔ RT Task ➔ RT FIFO ➔ user mode

Linux Process 2 ➔ Scheduling ➔ Linux Kernel ➔ Software Interrupts ➔ Real-Time Kernel ➔ Hardware Interrupts ➔ RT Task ➔ RT FIFO ➔ kernel mode
- Kernel mode and kernel address space
  - Real-time tasks installed as loadable modules
- Absolutely no system calls allowed
- Communication with non-real-time code possible
  - RT FIFOs (originally from RTLinux)
  - RT Shared memory, RT message queues (added by RTAI)
<table>
<thead>
<tr>
<th></th>
<th>Introduction</th>
<th>History</th>
<th>RTCore</th>
<th>PREEMPT_RT</th>
<th>Demo</th>
<th>Usage</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>History</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>RTCore</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>PREEMPT_RT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Demo</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Usage</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>References</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Concept

- Patch to standard Linux kernel
- Implements priority inheritance
Concept

- Patch to standard Linux kernel
- Implements **priority inheritance**

**Interrupt Handlers**

-Interrupts handlers are **preemptable**
- Kernel threads/tasklets
  - Network
  - File system
Concept

- Patch to standard Linux kernel
- Implements **priority inheritance**

**Interrupt Handlers**

- Interrupts handlers are **preemptable**
- Kernel threads/tasklets
  - Network
  - File system

**Kernel locks**

- Spinlocks replaced with **preemptable** RT mutexes
- Most of the Big Kernel Locks (BKLs) removed
  - Used to protect not thread-safe Kernel code
Towards a full preemptable Kernel

Kernel Evolution: Preemptible Code

- Kernel 2.0
  - BKL

- Kernels 2.2-2.4
  - Preemption points

- Kernel 2.6

- Real-Time Kernel 2.6

**Preemptible**

**Non-Preemptible**
Current State

Products

- Red Hat Enterprise MRG Realtime
  http://redhat.com/mrg

- SUSE Linux Enterprise Real-Time (SLERT)
  http://novell.com/products/realtime

Novell.
Terminology

- **ALSA**
  - Advanced Linux Sound Architecture
  - Kernel component for audio drivers

- **JACK**
  - JACK Audio Connection Kit
  - Client/Server system for handling real-time, low latency audio

- **ZynAddSubFX**
  - Software synthesizer
  - JACK client
Double Buffering
Buffer size defines minimal latency
Software synthesizer tool chain
Kernel/User mode

Kernel

ZynAddSubFX

DSP

jackd

ALSA

Buffer

Sound card
Other applications occupy Kernel

![Diagram of kernel components]

- ZynAddSubFX
- Flash plugin
- ... (Other applications)
- Kernel
  - ALSA
  - Buffer
  - Sound card
- DSP
- jackd
Real-time scheduling with a preemptive kernel

Diagram

- ZynAddSubFX
- Flash plugin
- ...
Demo!
1. Introduction
2. History
3. RTCore
4. PREEMPT_RT
5. Demo
6. Usage
7. References
**POSIX.1b-1993**

- **Inter-Process Communication**
  - Real-time signals
  - Message queues
  - Semaphores
- Prevent process from blocking
  - Asynchronous I/O
  - Memory lock
- Controlling scheduling policy/parameters
- Precise timers and `nanosleep`
- POSIX.1b compliant Operating Systems
  - QNX
  - LynxOS
Pitfalls

- Blocking System calls
  - printf
Pitfalls

- Blocking System calls
  - `printf`
  - `malloc (C)`
Pitfalls

- Blocking System calls
  - printf
  - malloc (C)
  - new (C++)
Pitfalls

- Blocking System calls
  - `printf`
  - `malloc (C)`
  - `new (C++)`

- Interrupts
  - Non-maskable Interrupts (NMI)
Pitfalls

- Blocking System calls
  - `printf`
  - `malloc (C)`
  - `new (C++)`

- Interrupts
  - Non-maskable Interrupts (NMI)
  - Disabling System Management Interrupts (SMIs) may harm the hardware
Pitfalls

• Blocking System calls
  • `printf`
  • `malloc (C)`
  • `new (C++)`

• Interrupts
  • Non-maskable Interrupts (NMI)
  • Disabling System Management Interrupts (SMIs) may harm the hardware

• Page faults
Enabling real-time scheduling

```c
int main(int argc, char* argv[]) {
    struct sched_param param;
    param.sched_priority = 50;
    pthread_setschedparam(pthread_self(), SCHED_FIFO, &param);
}
```
Preventing page faults on accessing the stack

```c
#define MAX_SAFE_STACK (8*1024) // 8 KB

void stack_prefault() {
    unsigned char dummy[MAX_SAFE_STACK];
    memset(&dummy, 0, MAX_SAFE_STACK);
}

int main(int argc, char* argv[]) {
    struct sched_param param;
    param.sched_priority = 50;
    pthread_setschedparam(pthread_self(), SCHED_FIFO, &param);
    mlockall(MCL_CURRENT|MCL_FUTURE);
    stack_prefault();
}
```
#define MAX_SAFE_STACK (8*1024) // 8 KB
#define MAX_ALLOCS_MEM (100*1024*1024) // 100 MB

void stack_prefault() {
    unsigned char dummy[MAX_SAFE_STACK];
    memset(&dummy, 0, MAX_SAFE_STACK);
}

int main(int argc, char* argv[]) {
    struct sched_param param;
    param.sched_priority = 50;
    pthread_setschedparam(pthread_self(), SCHED_FIFO, &param);
    mlockall(MCL_CURRENT|MCL_FUTURE);
    stack_prefault();
    malloc (M_TRIM_THRESHOLD, -1); // Turn off malloc trimming.
    malloc (M_MMAP_MAX, 0); // Turn off mmap usage.
    char* buffer = malloc(MAX_ALLOCS_MEM);
    memset(&buffer, 0, MAX_ALLOCS_MEM);
    free(buffer);
    // now it’s safe to do RT stuff
    }
SCHED_OTHER

- Use default, non-real-time scheduling
SCHED_OTHER
  - Use default, non-real-time scheduling

SCHED_RR
  - Use round-robin real-time scheduling
Policies

**SCHED_OTHER**
- Use default, non-real-time scheduling

**SCHED_RR**
- Use round-robin real-time scheduling

**SCHED_FIFO**
- Use FIFO real-time scheduling
### Policies

**SCHED_OTHER**
- Use default, non-real-time scheduling

**SCHED_RR**
- Use round-robin real-time scheduling

**SCHED_FIFO**
- Use FIFO real-time scheduling

**SCHED_SPORADIC**
- With a known *time budget* and *period* this scheduling policy allows Rate Monotonic Scheduling with other periodic or sporadic tasks
## Scheduling

### Priorities

<table>
<thead>
<tr>
<th>API</th>
<th>nice/renice</th>
<th>pthread_setschedparam</th>
</tr>
</thead>
<tbody>
<tr>
<td>+19 ... 0</td>
<td>-1 ... -20</td>
<td>1 ... 49</td>
</tr>
<tr>
<td></td>
<td></td>
<td>50 ... 99</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Low Prio</th>
<th>High Prio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fairly Scheduled</td>
<td>Below Kernel RT</td>
<td>Round-Robin/FIFO Scheduled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>39 ... -100</td>
</tr>
</tbody>
</table>

|             | High Prio |
| Kernel      |           |
|             | 139 ... 0 |

- **Kernel**: 139 ... 0
- **Top**: 39 ... -100
- **Low Prio**: +19 ... 0
- **High Prio**: -1 ... -20
- **Round-Robin/FIFO Scheduled**: 1 ... 49
- **Below Kernel RT**: 50 ... 99

**Usage**

- **nice/renice**
- **pthread_setschedparam**

**API**

- `top`
References

Michael Barabanov.
A linux-based real-time operating system.

Bryan Che.
Introduction to realtime linux.
SCALE 7x, February 2009.

IEEE/ISO/IEC.
ISO/IEC IEEE Std. 1003.1b-1993 real time extensions.
1993.

Ingo Molnar and Thomas Gleixner.
Interview: the return of the realtime preemption tree.
LWN.net.

Theodore Ts’o, Darren Hart, and John Kacur.
Real-time linux wiki.

Klaas van Gend.
Real time linux patches: history and usage.