“Real Time” vs. “Real Fast”: How to Choose?

Paul E. McKenney
IBM Distinguished Engineer & CTO Linux
Linux Technology Center
Overview

- What Is “Real Time”?
- What Real-Time Applications?
- Example Real-Time Application
- Example Real-Fast Application
- Real Time vs. Real Fast: How to Choose
- Summary
What is “Real Time”? 
Hard or Soft Real Time?

<table>
<thead>
<tr>
<th>Required Response Time</th>
<th>Probability of Meeting Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>1us</td>
<td>0%</td>
</tr>
<tr>
<td>10us</td>
<td>50%</td>
</tr>
<tr>
<td>100us</td>
<td>100%</td>
</tr>
<tr>
<td>1ms</td>
<td>100%</td>
</tr>
<tr>
<td>10ms</td>
<td>100%</td>
</tr>
</tbody>
</table>
Definitions of Hard Real Time

1. A Hard Real-Time System Will Always Meet its Deadlines
2. A Hard Real-Time System Will Either (1) Meet its Deadlines, or (2) Give Timely Failure Indication
3. A Hard Real-Time System Will Always Meet its Deadlines (in Absence of Hardware Failure)

- Which definition is appropriate? Why, and under what conditions?
Hard Realtime: Problem With Definition #1

- If you have a hard realtime system...
  - I have a hammer that will make it miss its deadlines!
Hard Realtime: Problem With Definition #2

- I have a “hard realtime” system
  - It simply always fails!
Hard Realtime: Problem With Definition #3

- “Rest assured, sir, that if your life support fails, your death will most certainly not have been due to a software problem!!!”
Hard Realtime: Problem With Definition #4

- This definition can cause purists severe heartburn and cognitive dissonance
- But this definition is what is used in “real life”
- Real systems are too complex to admit first-principles mathematical analysis
  - Perhaps this will change with improved tooling
  - Or perhaps the effects noted by Peter Okech and Carsten Emde in their talks yesterday will favor a continued testing-oriented approach
Real Time and Real Fast: **Useful** Definitions

- **Real Time**
  - OS: “how long before work starts?”

- **Real Fast**
  - Application: “once started, how quickly is work completed?”

- This Separation Can Result in Confusion!
What Real-Time Applications?
What Real-Time Applications?

... In Search of Life ...

In Search of Death ...

... In Search of Money
What Real-Time Applications?

- Industrial control
- Embedded devices
  - PDAs, cellphones, TVs, refrigerators, cars, ...
- Military
- Scientific
- Financial
- Commercial
  - Break with traditional practice: real-time systems no longer standalone, but tied into enterprise IT systems
Example Real-Time Application
Example Real-Time Application: Fuel Injection

- **Mid-sized industrial engine**
  - Fuel injection within one degree surrounding “top dead center”
- **1500 RPM rotation rate**
  - 1500 RPM / 60 sec/min = 25 RPS
  - 25 RPS * 360 degrees/round = 9000 degrees/second
  - About 111 microseconds per degree
  - Hence need to schedule to within about 100 microseconds
Fuel Injection: Conceptual Operation

Top Dead Center

Bottom Dead Center
Too Early and Too Late Are Bad

Injecting Too Early (Exaggerated)

Injecting Too Late (Exaggerated)
Fanciful Code Operating Injectors

```c
struct timespec timewait;

angle = crank_position();
timewait.tv_sec = 0;
timewait.tv_nsec = 1000 * 1000 * 1000 * angle / 9000;
nanosleep(&timewait, NULL);
inject();
```
Fuel Injection Test Program

```c
if (clock_gettime(CLOCK_REALTIME, &timestart) != 0) {
    perror("clock_gettime 1");
    exit(-1);
}
if (nanosleep(&timewait, NULL) != 0) {
    perror("nanosleep");
    exit(-1);
}
if (clock_gettime(CLOCK_REALTIME, &timeend) != 0) {
    perror("clock_gettime 2");
    exit(-1);
}
```

Bad results, even on -rt kernel!!! Why?
Test Program Needs MONOTONIC

```c
if (clock_gettime(CLOCK_MONOTONIC, &timestart) != 0) {
    perror("clock_gettime 1");
    exit(-1);
}
if (nanosleep(&timewait, NULL) != 0) {
    perror("nanosleep");
    exit(-1);
}
if (clock_gettime(CLOCK_MONOTONIC, &timeend) != 0) {
    perror("clock_gettime 2");
    exit(-1);
}
```

Still bad results, even on -rt kernel build!!! Why?
Test Program Needs RT Priority

```c
struct sched_param sp;

sp.sched_priority = sched_get_priority_max(SCHED_FIFO);
if (sp.sched_priority == -1) {
    perror("sched_get_priority_max");
    exit(-1);
}
if (sched_setscheduler(0, SCHED_FIFO, &sp) != 0) {
    perror("sched_setscheduler");
    exit(-1);
}
```

Still sometimes bad results, even on -rt kernel build!!! Why?
Test Program Needs mlockall()

```c
if (mlockall(MCL_CURRENT | MCL_FUTURE) != 0) {
    perror("mlockall");
    exit(-1);
}
```

Better results on -rt kernel: nanosleep jitter < 20us, 99.999% < 13us
(4-CPU 2.2GHz x86 system w/SMI remediation – your mileage will vary)

More than 3 milliseconds on non-realtime kernel!!!
(Though improved on more recent kernels with high-resolution timers.)
Timer Jitter on Recent Kernel

Somewhat worse: 45us max of 10M samples, 99.999% < 27us
Same machine, but 18 month older
Fuel Injection: Further Tuning Possible

- On multicore systems:
  - Affinity time-critical tasks onto private CPU
    - (Can often safely share with non-realtime tasks)
  - Affinity IRQ handlers away from time-critical tasks

- Carefully select hardware and drivers
- Carefully select kernel configuration
  - Depends on hardware in some cases
Example Real-Fast Application
Bring RT Magic to Non-Real-Time Application

```
tar -xjf linux-2.6.24.tar.bz2
cd linux-2.6.24
make allyesconfig > /dev/null
time make -j8 > Make.out 2>&1
cd ..
rm -rf linux-2.6.24
```
# Kernel Build: Performance Results

<table>
<thead>
<tr>
<th></th>
<th>Real Fast (s)</th>
<th>Real Time (s)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>real</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>828.7</td>
<td>904.1</td>
<td>0.92</td>
</tr>
<tr>
<td>Std. Dev.</td>
<td>5</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td><strong>user</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>2337.9</td>
<td>2510.5</td>
<td>0.93</td>
</tr>
<tr>
<td>Std. Dev.</td>
<td>4.7</td>
<td>1.4</td>
<td></td>
</tr>
<tr>
<td><strong>sys</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>323.4</td>
<td>430.7</td>
<td>0.75</td>
</tr>
<tr>
<td>Std. Dev.</td>
<td>4.9</td>
<td>11.1</td>
<td></td>
</tr>
</tbody>
</table>

Smaller is better, real-time kernel *not* helping...
(But much better than 18 months ago.)
Real Time vs. Real Fast: Role of Task Duration

![Graph showing the role of task duration](image-url)
Real Time vs. Real Fast: Throughput Comparison

- **Real-time system starts more quickly**
  - Proverbial hare
- **Real-fast system has opportunity to catch up**
  - Proverbial tortoise
- **Tradeoff based on task duration**
The Dark Side of Real Time

So, have you finished yet?

Nope!

But I gave you this task two weeks ago!!!

Hey, I started work on it immediately!!!

© 2008 Sarah McKenney Creative Commons (Attribution)
The Dark Side of Real Fast

So, have you finished yet?
Nope!

But I gave you this task two weeks ago!!

Chill! As soon as I start, I will be done in no time!!!
Sources of Real-Time Overhead

- Memory utilization due to mlockall()
  - Hence Clark's reluctance to recommend mlockall()
- Increased locking overhead
  - Context switches, priority inheritance, preemptable RCU
- Increased irq overhead
  - Threaded irqs (context switches)
  - Added delay (and interactions with rotating mass storage)
- Increased real-time scheduling overhead
  - Global distribution of high-priority real-time tasks
  - Peter Zijlstra is working on this issue
    - Might need combined approach: partition + placement assist
- High-resolution timers
  - Pretty hard to beat timer-wheel throughput!!!
Real Time vs. Real Fast: How to Choose
Real Time vs. Real Fast: How to Choose

- Throughput only goal?
  - Yes → Real Fast
  - No → Peak Loads Degrade Response?
    - Yes → Real Fast
    - No → Memory Pressure?
      - Yes → Real Fast
      - No → Real Time

- Virtualization Required? (RT Guests)
  - Yes → Real Fast
  - No → Basic Work Item > 20ms?
    - Yes → Real Fast
    - No → Real Time
Longer Term: Avoid the Need to Choose

- Reduce Overhead of Real-Time Linux!
  - Easy to say, but...
  - Reduce locking overhead (adaptive locks)
  - Reduce scheduler overhead (ongoing work)
  - Optimize threaded irq handlers
  - Eliminate networking reader-writer-lock bottlenecks (ongoing work)
  - Beat up HW people to improve HW latencies
    - I nominate Thomas Gleixner to administer the beatings

- Note that partial progress is beneficial
  - Reduces the need to choose

- Harvest the low-hanging fruit!!!
Low-Hanging Fruit

Harvest it.
Don't trip over it!

© 2008 Sarah McKenney Creative Commons (Attribution)
Summary

Use the right tool for the job!!!

Image copyright © 2004 Melissa McKenney
Acknowledgments

- Ingo Molnar
- Thomas Gleixner
- Sven Dietrich
- K. R. Foley
- Gene Heskett
- Bill Huey
- Esben Neilsen
- Peter Zijlstra

- Nick Piggin
- Steve Rostedt
- Michal Schmidt
- Daniel Walker
- Karsten Wiese
- Gregory Haskins
- Darren Hart
- John Stultz

- And many many more...
Legal Statement

- This work represents the view of the author and does not necessarily represent the view of IBM.
- IBM and IBM (logo) are trademarks or registered trademarks of International Business Machines Corporation in the United States and/or other countries.
- Linux is a registered trademark of Linus Torvalds.
- Other company, product, and service names may be trademarks or service marks of others.
Questions?

To probe further:

- **Applications:**
  - [http://www.linuxjournal.com/article/9361](http://www.linuxjournal.com/article/9361) (Enterprise real-time)
  - [http://www.b-eye-network.de/view-articles/3365](http://www.b-eye-network.de/view-articles/3365) (Time value of information)
  - [http://searchenterpriselinux.techtarget.com/news/article/0,289142,sid39_gci1309889,00.html](http://searchenterpriselinux.techtarget.com/news/article/0,289142,sid39_gci1309889,00.html) (Order of magnitude decrease in response time required over 5 years time)

- **Extreme Real Time:**

- **Rants:**