Real Time vs. Real Fast

Paul E. McKenney
IBM Distinguished Engineer
Overview

- Confessions of a Recovering Proprietary Programmer
- What is “Real Time” and “Real Fast”, Anyway???
- Example Real Time Application
- Example Real Fast Application
- Real Time vs. Real Fast
- How to Choose
Proprietary Programming: Requirements
Proprietary Programming: “Solution”
FOSS Programming: Requirements
Just Another Day on LKML...
But Sometimes Consensus is Achieved
And a Good Solution Produced Thereby
What is “Real Time”, Anyway?

Review of Definitions

(Taken from January 2007 Linux Journal article.)
What is “Real Time”, Anyway? (Definition #1)

A hard realtime system will always meet its deadlines
Problem With Definition #1

If you have a hard realtime system...

I have a hammer that will make it miss its deadlines!
What is “Real Time”, Anyway? (Definition #2)

A hard realtime system will either:
(1) meet its deadlines, or
(2) give a timely failure indication
Problem With Definition #2

I have a “hard realtime” system
It simply always fails!
What is “Real Time”, Anyway? (Definition #3)

A hard realtime system will meet all its deadlines!!!(But only in absence of hardware failure.)

(Never mind that handling hardware failures is an important software task!!!)
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!!!”
What is “Real Time”, Anyway? (Definition #4)

A hard realtime system will pass a specified test suite.

(This definition can cause purists severe heartburn.)

(But is actually used in real life.)
But One Other Question on Definitions 1-3...

What is the Deadline???

What guarantees can an RTOS make?
Real Time and Real Fast: 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!
Example Real Time Application: Fuel Injection
Example Real-Time Application: Fuel Injection

Mid-sized industrial engine

- Fuel injection within one degree surrounding “top dead center”

1500 RPM rotation rate

- $1500 \text{ RPM} / 60 \text{ sec/min} = 25 \text{ RPS}$
- $25 \text{ RPS} \times 360 \text{ degrees/round} = 9000 \text{ degrees/second}$
- About 111 microseconds per degree
- Hence need to schedule to within about 100 microseconds
Fuel Injection: Conceptual Operation

Top Dead Center

Bottom Dead Center
Fuel Injection: Too Early and Too Late Are Bad

Injecting Too Early (Exaggerated)

Injecting Too Late (Exaggerated)
Fuel Injection: 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();
```
Bad results, even on -rt kernel build!!! Why?
Fuel Injection: 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?
Fuel Injection: 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?
Fuel Injection: 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 with RT firmware – your mileage will vary)

More than 3 milliseconds on non-realtime kernel!!!
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: Kernel Build
Real-Time Magic to Non-Real-Time Application

Kernel build

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>1332.6</td>
<td>1556.2</td>
<td>0.86</td>
</tr>
<tr>
<td>Std. Dev.</td>
<td>14.6</td>
<td>22.4</td>
<td></td>
</tr>
<tr>
<td><strong>user</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>3012.2</td>
<td>2964.7</td>
<td>1.02</td>
</tr>
<tr>
<td>Std. Dev.</td>
<td>12.7</td>
<td>17.5</td>
<td></td>
</tr>
<tr>
<td><strong>sys</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>316.6</td>
<td>657</td>
<td>0.48</td>
</tr>
<tr>
<td>Std. Dev.</td>
<td>1.4</td>
<td>9.2</td>
<td></td>
</tr>
</tbody>
</table>

Smaller is better, real-time kernel *not* helping...
Comparison of Real Time vs. Real Fast
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!!!
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!!!
Real Time vs. Real Fast Throughput: No Penalty

For example, heavy floating-point workloads
Real Time vs. Real Fast Throughput: “real” Penalty

Mixed workloads
Real Time vs. Real Fast Throughput: “sys” Penalty

Filesystem I/O workloads: “don't do that”
Real-Time Latency vs. CPU Utilization

CPU Utilization by Real-Time Tasks
Can be avoided by time-slotting
Which happens naturally in piston engines
Sources of Real-Time Overhead

Memory utilization due to 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 overhead of scheduling real-time tasks
  Global distribution of high-priority real-time tasks
High-resolution timers
Real Time vs. Real Fast: How to Choose
Real Time vs. Real Fast: How to Choose

- Throughput only goal? (Y/N)
  - Peak Loads Degrade Response? (N/Y)
    - Memory Pressure? (N/Y)
      - Real Fast
    - Real Time
  - Virtualization Required? (RT Guests) (Y/N)
    - Basic Work Item > 100ms? (Y/N)
      - Real Fast
    - Real Time
Longer Term: Avoiding 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 MV work)

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!
Acknowledgments

Ingo Molnar
Thomas Gleixner
Sven Deitrich
K. R. Foley
Gene Heskett
Bill Huey
Esben Neilsen

Nick Piggin
Steve Rostedt
Michal Schmidt
Daniel Walker
Karsten Wiese
Gregory Haskins

And many many more...
Legal Statement

This work represents the views of the authors and does not necessarily represent the view of IBM.

Linux is a copyright of Linus Torvalds.

Other company, product, and service names may be trademarks or service marks of others.
Use the right tool for the job!!!