Combining Predictable Execution with Full-Featured Commodity Systems

Adam Lackorzynski, Carsten Weinhold, Hermann Härtig

Jul 5th 2016, OSPERT Workshop
Combining a GPOS and Real-Time
Our Goal

- Use GPOS (Linux) for non-RT parts of an RT application
- Run RT part outside of Linux
  - Separate „noisy“ Linux from sensitive parts
- Best of both worlds:
  - Use Linux for features
  - Run RT outside of Linux
- No gap to overcome
  - No application splitting
  - No proxying
L4Re Microkernel System

L4Re Microkernel / Hypervisor

L4Re Services

μApp

RT-App

Security -App

VM
L4Re Virtualization Variants

- Hardware-assisted virtualization
  - Intel VT-x, AMDs SVM, ARM VE, MIPS VZ

- Para-Virtualization
  - No hardware support for Virtualization required
  - L⁴Linux, FreeRTOS, L4OpenBSD, ...
  - Tight integration...
L4Linux

L4Re Microkernel

vCPU

Task

Task

Task

Core

Core

Core
Detaching Linux Threads
• Thread of an application can run undisturbed from Linux activities
• Program can execute L4 system calls directly, without involving the Linux kernel
• Multi-threaded applications can run detached and normal in parallel
Evaluation / Benchmarks

FWQ (Fixed Work Quantum)
• Measure OS Noise
• Measure execution time jitter

Control loop hybrid application

3 Workloads:
• Idle
• Build: Building a Linux kernel
• IO: Stressing disk and network

Intel Core™ i7-4770, Quad-Core at 2993 MHz
• Linux idle

~450,000 cycles
9.6%
FWQ Linux Examples

- Linux build load

~570,000 cycles 11.8%
FWQ Linux Examples

- Linux I/O load

~4,430,000 cycles
51%
• Linux PREEMPT Idle
• Linux PREEMPT build load
• Linux PREEMPT I/O load

~774,000 cycles
18%
FWQ L⁴Linux with Detaching

- Detached build load

CPU Cycles per Work

- ~6,500 cycles
• With idle, build load & I/O load
<table>
<thead>
<tr>
<th>Deltas to minimum in CPU cycles</th>
<th>Linux PREEMPT</th>
<th>Detached Thread in L4Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>434000 (10.2%)</td>
<td>21 (0.0005%)</td>
</tr>
<tr>
<td>Build Load</td>
<td>520000 (12.2%)</td>
<td>6528 (0.15%)</td>
</tr>
<tr>
<td>I/O Load</td>
<td>773000 (18.2%)</td>
<td>1152 (0.027%)</td>
</tr>
</tbody>
</table>
• Looking at interrupt latency

• Use Linux to launch the following code:

```c
next = now() + 1000µs;
while (1) {
    wait_for_time(next);
    tsc = read_tsc();
    histogramm[i++] = tsc - prev_tsc;
    prev_tsc = tsc;
    /* do work */
    next += 1000µs;
}
```

• Linux and L4 implementations of now() and wait_for_time()
Control Loop with Linux

- Linux Idle

+/- ~440,000 cycles
+/- 14%
Control Loop with Linux

- Linux build load

~2,600,000 cycles
+/- 43%
Control Loop with Linux

- Linux I/O load

![Graph showing Linux I/O load with +/- 451,000 cycles and +/- 15% variation.](image)
Control Loop in Detached Thread

- $L^4$Linux detached Idle

+1,129 / -680 cycles
+0.03% / -0.02%
Control Loop in Detached Thread

- L⁴Linux detached build load

![Graph showing CPU cycles per loop with a range of +4,200 / -5,400 cycles and a percentage change of +0.14% / -0.18%](image-url)
• $L^4$Linux detached I/O load
<table>
<thead>
<tr>
<th>Maximum Deviation in CPU cycles</th>
<th>Linux</th>
<th>L^4Linux Detached Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>862000 (+14.7% / -14.1%)</td>
<td>1809 (+0.03% / -0.02%)</td>
</tr>
<tr>
<td>Build Load</td>
<td>2599000 (+/- 43.4)</td>
<td>9633 (+0.14% / -0.18%)</td>
</tr>
<tr>
<td>I/O Load</td>
<td>903000 (+/- 15.1%)</td>
<td>3339 (+0.06% / -0.053%)</td>
</tr>
</tbody>
</table>
Conclusion

- Separating a part of a program from a GPOS is possible
- Avoids separation gap
- Real-time and HPC have similar goals
l4re.org