High-Resolution Timing Update

Nick Rasmussen
nick@cs.wisc.edu

Brandon Schendel
schendel@cs.wisc.edu

Computer Sciences Department
University of Wisconsin
1210 W. Dayton St.
Madison, WI 53706-1685
USA
Paradyn High-Resolution Timing Update

© 2000 {nick,schendel}@cs.wisc.edu

Visis    Paradyn Front-end    Paradyn Daemon    Application

RTinst

OS

Hardware
Motivation

• We already have access to high-resolution elapsed time
  • TSC: 20 cycle query cost, ~2ns resolution

• Our only interface to get process time is through the times system call
  • times: 350 cycle query cost, 10ms resolution

• Goal: Provide access to high-resolution process time with low query cost
High-Resolution Timers

• Solaris `gethrtime`, `gethrvtime`

• Timers previously added to other OSs:
  • VAX/BSD, SunOS 2.x, CM5, Sequent
    Symmetry
Linux Kernel Modifications

- Add TSC-based process time bookkeeping to `task_struct`
- Update on context switch
- Update on kernel entry and exit
- Expose to user via `mmap`
  - `/proc/PID/hrtime`
- Allows cycle-accurate query of other processes
hrtime_struct

```c
struct hrtime_struct {
    volatile hrtime_t last_us_dispatch;
    volatile hrtime_t utime;
    volatile hrtime_t stime;
    volatile long in_system;
    long has_ustime;
    hrtime_t start_time;
    volatile hrtime_t last_dispatch;
    volatile hrtime_t vtime;
    volatile long offset_to_cpu0;
    long refcount;
    spinlock_t reflock;
};
```
hrtime_struct

```c
struct hrtime_struct {
    volatile hrtime_t last_us_dispatch;
    volatile hrtime_t utime;
    volatile hrtime_t stime;
    volatile long     in_system;
    long              has_ustime;
    hrtime_t          start_time;
    volatile hrtime_t last_dispatch;
    volatile hrtime_t vtime;
    volatile long     offset_to_cpu0;
    long              refcount;
    spinlock_t        reflock;
}
```
Self-Query Example

// hr points to the mmapped hrtime_struct
// for the current process
do {
    begin = rdtsc();
    vtime = begin - hr->last_dispatch + hr->vtime;
    end   = rdtsc();
} while (end - begin > THRESHOLD)
Subtle Issues

• TSC synchronization on SMP boxes
• Process state change during query
  • Context switch, kernel entry/exit
• TSC $\rightarrow$ nanosecond conversion
  • Drift from real time if conversion factor is not accurate
Libhrt ime

• User library to access timers
• Timer query cost - ~120 cycles
<table>
<thead>
<tr>
<th>Visis</th>
<th>Paradyn Front-end</th>
<th>Paradyn Daemon</th>
<th>Application</th>
</tr>
</thead>
</table>

![Diagram showing the relationship between Visis, Paradyn Front-end, Paradyn Daemon, and Application. The diagram includes OS and Hardware layers.](image)
Motivation

• Sample data pipeline through Paradyn is too narrow to support high resolution counters.

Used with permission of Bob Thaves.
Current Sampling Pipeline

<table>
<thead>
<tr>
<th>Visis</th>
<th>Paradyn Front-end</th>
<th>Paradyn Daemon</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td>float</td>
<td>double</td>
<td>int32</td>
</tr>
<tr>
<td>float</td>
<td>float</td>
<td>int32</td>
<td>int32</td>
</tr>
<tr>
<td>float</td>
<td>float</td>
<td>int64</td>
<td>int64</td>
</tr>
</tbody>
</table>

RTinst

Counters

Timers

seconds and fraction of seconds

microseconds
Future Sampling Pipeline

Visis | Paradyn Front-end | Paradyn Daemon | Application

- int64
- int64
- int64

- int64
- int64
- int64

Counters

Timers

nanoseconds

varying time units
## Data Type Tradeoffs

How long can we exactly count nanoseconds?

<table>
<thead>
<tr>
<th></th>
<th>float</th>
<th>double</th>
<th>int64</th>
</tr>
</thead>
<tbody>
<tr>
<td>bits of precision</td>
<td>23</td>
<td>52</td>
<td>63</td>
</tr>
<tr>
<td>largest counting value</td>
<td>$8 \times 10^6$</td>
<td>$4 \times 10^{15}$</td>
<td>$9 \times 10^{18}$</td>
</tr>
<tr>
<td>time</td>
<td>8 ms</td>
<td>52 days</td>
<td>292 years</td>
</tr>
<tr>
<td>aggr w/ 100 processes</td>
<td>80 $\mu$s</td>
<td>13 hours</td>
<td>2.9 years</td>
</tr>
<tr>
<td>aggr w/ 1000 processes</td>
<td>8 $\mu$s</td>
<td>1.3 hours</td>
<td>106 days</td>
</tr>
</tbody>
</table>
Choosing Time Retrieval Method

- Run time check of best available timer
  - Linux
    - use libhrt ime if available
    - else default to standard system calls
  - Other platforms
    - similar timer selections
Current Status: Sampling Pipeline

• RTinst library and daemon in release 3.0
• Next step
  1) Front-end
  2) Daemon to Front-end RPC calls
  3) Visis
  4) Visi to Front-end RPC calls
Current Status: Linux Timer Support

- Linux kernel support and library completed
  - Library and patches against the latest stable and development kernels at:
    - http://www.cs.wisc.edu/paradyn/libhrtime/
- TSC offset measurement on SMP boxes unimplemented
- Not yet accepted into the main kernel tree