Greening the OpenSolaris Kernel

OSDevCon 2009, Dresden

Eric Saxe <eric.saxe@sun.com>

Solaris Kernel Development
Sun Microsystems, Inc.
http://www.opensolaris.org/os/project/tickless
Intro and Overview

- Power Management Feature Background
- Greening the System
  - Power Efficient Resource Management
  - Efficient Resource Consumption
- Tickless Kernel Project
  - Overview
  - Progress
- Getting Involved
Resource Power Management

- **Active Resource Power States**
  - Trade off: performance vs. power
    - CPUs: Dynamic Frequency, Voltage Scaling (DVFS)
    - Memory, CPUs: Clock Throttling
    - CPUs: Dynamic Frequency Overclocking

- **Idle Resource Power States**
  - Trade off: power vs. recovery latency
    - CPUs: ACPI C-states
    - Memory: Self-Refresh
    - Systems: Suspend to RAM, Suspend to Disk
CPU Power Management (then)

- The CPUPM Subsystem and the dispatcher don't necessarily get along.
- Architecture relies on polling, need to periodically look at CPU utilization statistics, even on an idle system.
Dispatcher Integrated CPUPM (now)

- Event based architecture driven by thread scheduling activity (no polling)
- Enables power aware thread placement, and thread aware CPU power management
- Dynamic Frequency and Voltage Scaling, and multi-level C-states
But None of it Matters....

… If consumers are wasteful (or just broken) with respect to resource utilization.
But None of it Matters....

… If consumers are wasteful (or just broken) with respect to resource utilization.

There's limits to what can be done with respect to optimizing resource management efficiency...

“throttling” requests (where possible) generally detrimental to performance

Imposing “active PM” residency at the expense of “idle PM” residency generally not good trade-off
But None of it Matters....

... If consumers are wasteful (or just broken) with respect to resource utilization.

There's limits to what can be done with respect to optimizing resource management efficiency

“throttling” requests (where possible) generally detrimental to performance

Imposing “active PM” residency at the expense of “idle PM” residency generally not good trade-off

Good resource management ultimately cannot compensate for wasteful resource consumption.
Profiles of Inefficient Software

- Resource consumption non proportional with respect to useful work performed...

- At higher utilizations with poor scaling...
  - Too many threads, memory leaks, etc.

- At low/zero utilization, by not yielding (or continuing to use) resources
  - e.g. periodic “polling” for a condition
Observing Inefficiency

- A simple approach for the low utilization case...
  - At system idle no useful work is being performed...
  - So watch who's using resources (they are being bad).
Observing Inefficiency

- A simple approach for the low utilization case...
  - At system idle no useful work is being performed...
  - So watch who's using resources (they are being bad).

- Optimizing for the low utilization case makes sense, due to effectiveness of idle power management features.
  - In many ways, high utilization case already pursued though performance (scalability) efforts.
PowerTOP(1M)

PowerTOP for OpenSolaris v1.1

Cn Avg residency P-states (frequencies)
C0 (cpu running) (59.9%) 1998 Mhz 100.0%
C1 0.1ms (40.1%) 2997 Mhz 0.0%

Wakes from idle per second: 3477.2 interval: 1.0s
no ACPI power usage estimate available

Top causes for wakes:
86.3% (3000.0)
4.3% (148.5)
3.5% (121.6)
3.4% (116.8)
3.1% (107.9)
2.9% (100.0)
2.8% (96.0)
1.9% (66.3)
1.5% (53.5)
0.5% (16.8)
0.3% (10.9)
0.3% (9.9)
0.2% (7.9)
0.2% (7.9)
0.2% (5.0)
0.1% (5.0)
0.1% (4.0)
0.1% (4.0)
0.1% (3.0)
0.1% (3.0)

Suggestion: run as root to get suggestions for reducing system power consumption

Q - Quit  R - Refresh
Greening the System
Starting with the Kernel...

Why?

- Improve ability to leverage idle power management features (especially on small systems).
- Lessen guest performance overhead at zero utilization (when sharing system with other guests).
- Lessen jitter, to improve RT latency/determinism and barrier synchronization performance (HPC).
- Improve kernel service scalability.
- Set the example for all software in the ecosystem, and learn (while providing missing mechanism) along the way...
Greening the System

Approach

- Consider PowerTOP(1M) an “todo” list.
  - Being “tickless” is a matter of degree (not binary)
    - e.g. average duration of system quiescence
- Begin by eliminating the 100 Hz clock() cyclic
  - Decompose it into component tick based services.
    - For each service:
      - Provide an event based (tickless) implementation
      - Where this isn't possible, make it less painful.
- Provide the architecture / interfaces needed to facilitate event based programming practices (and more efficient polling) throughout the system.
Tickless clock() Overview

- Core tick-based clock() services
  - Expire callouts / timeouts (timers)
  - Perform CPU utilization accounting for running threads, and expire time slices
  - Bump lbolt variable (tick resolution time source)
  - Time-of-day / hires time sync up
  - ...and other stuff that's crept in.
Tickless Timeouts / Callouts

Historical Implementation

- `clock()` invoked a routine that would inspect callout table heaps, expiring due timers.
- Inherently non-scalable and inefficient (as tables frequently empty on idle systems)
Tickless Timeouts / Callouts

- **Historical Implementation**
  - `clock()` invoked a routine that would inspect callout table heaps, expiring due timers.
  - Inherently non-scalable and inefficient (as tables frequently empty on idle systems)

- **Tickless Implementation**
  - Re-programmable cyclics introduced
  - Per CPU timer heap(s), driven by a re-programmable cyclic whose firing is set for when the next timer is due.

- Status: Integrated into Nevada build 103
Tickless Ibolt

- Ibolt - “lightning bolt”
  - “tick” counter (global kernel variable) incremented by clock()
  - Used extensively throughout the kernel
    - as a low resolution, yet cheap to read (and convenient) time source
    - as arguments for cv_timedwait() and friends
  - Likely used in 3rd party kernel modules

- Approach
  - Replace the variables with a routine backed by a hardware time source
    - Leverage existing ddi_get_lbolt()
  - Change where Ibolt comes from, not how it is used

- Status
  - Preparing to integrate (next few builds)
Tickless Thread Accounting (TAC)

Approach
- Per thread heap of timers maintained that fire when various amounts of thread CPU time have elapsed
  - time slice expiration, CPU time resource limits, etc.
- Builds upon “reprogramable cyclics” feature

Implementation
- A TAC omni-cyclic processes the per CPU timer heaps.
- Each CPUs cyclic is programmed at context switch time to the earliest timer in the heap
- On cyclic expire, accounting is done and the cyclic is reprogrammed to the next timer
- If the cyclic detects a kernel thread, it switches itself off

Status
- In development. Design document available for review.
Tickless OpenSolaris Project
Getting Involved

- Primary mailing list: tickless-dev@opensolaris.org
- Source repositories hosted on hg.opensolaris.org
  - One “gate” per clock() sub project
  - Will likely maintain a repo that is also the merge of the sub-projects

Bug Tracking

- Bugzilla: http://defect.opensolaris.org/
  - Track bugs under: Development/power-mgmt/tickless*
    - tickless tick accounting, tickless lbolt, tickless time sync, tickless clock misc
  - All bug updates currently go to tickless-dev as well

Dev Team Meetings

- Tuesdays 10:30AM Pacific
- Concall info on project page
Tickless OpenSolaris Project

OpenSolaris Project: Tickless Kernel Architecture

View the leaders for this project
Project Observers

Endorsing communities
Power Management

Tickless Kernel Architecture

Overview
By default, the clock cyclic fires at 100 Hz, regardless of whether or not any timeouts/callsouts are scheduled to fire/expire. This is suboptimal from a power efficiency standpoint, as at least one of the system's CPUs never become quiescent/die enough to be brought into a low power state.

This work involves re-implementing the services presently provided by clock() in a tickless (or event based) fashion, eliminating the need for the system to "wake up", only to realize that there's nothing to do on an otherwise idle system.

Tasks
Please have a look at the Tasks page and it's child pages for more information on the implementation of this project.

Getting Involved
If you would like to be involved with this project, the best way to get started is to join the project development mailing list, and introduce yourself (who you are, what you do, what interests you about this project, etc.). If there is something in particular you are looking for, please feel free to ask.

Project Mail Aliases
tickless-dev at opensolaris dot org is the project's primary development mailing list. Please feel free to subscribe.

- Subscribe
- Unsubscribe
- Change your subscription options
- View the list archives

CRs
Currently, this project and its subtasks are tracked by the following change requests:
6551399 clock efficiency optimizations ("tickless clock")

Announcements

16 Mar 2009 "Kick-off" meeting announcement

Blogs
rv - Split views
References

- Tickless Project Page
  - http://www.opensolaris.org/os/project/tickless

- Power Management Community
  - http://www.opensolaris.org/os/community/pm