Lab 4: Energy Management

18-648: Embedded Real-Time Systems

Out: November 7, 2016, 12:00:00 AM EST
Due: November 30, 2016, 11:59:59 PM EST

1 Introduction

You should learn the following from this lab:

- Tracking energy consumption
- Exploiting processor sleep states to reduce energy consumption
- Exploiting processor frequency scaling to reduce energy consumption
- Evaluation of energy-aware task partitioning heuristics

Before you start hacking:

- Read the entire lab handout before starting work: writeup questions and the tips section aim to give you hints about design and implementation.
- If you complete any work separately, make sure to explain to each other how things work in detail, incl. the writeup questions. During demo we may ask any teammate about any part of the lab.
- Whenever the lab asks you to do something without explaining how to accomplish it, please consult references on kernel development. Thats by design.
- To be awarded full credit, your kernel must not crash or freeze under any circumstances. You are an RTOS vendor!

1.1 Background Information

1.1.1 Power and Energy Model

Recall from lecture that power consumption of a CMOS circuit can be modeled analytically as

$$P(f) = \kappa f^{\alpha} + \beta$$

where the parameters κ, α, β are fitted to measured power consumption data. Energy consumed by a task running for t at frequency f is

$$E(f, t) = P(f)t$$

Prior measurements for the large cores of the Tegra 3 SoC in Nexus 7[1] have determined the following parameter values:

κ	0.00442	mW/MHz
α	1.67	
β	25.72	mW

[1]Power was measured with a current-sense resistor inserted into the feedback path of the voltage regulator supplying the large core cluster.

1.1.2 List Scheduling

List-scheduling is a widely-used generic job scheduling policy. When a job, such as a web server request, arrives, it is assigned to the resource provider with the shortest queue. An attractive property of the resulting assignment is a balanced load. A corollary of a balanced load is that the load on each service provider is at the minimum.

The latter property is particularly useful for energy-efficient task partitioning. Energy of a processor is determined by its frequency, which is constrained by the load allocated to the processor. The list policy can be naturally adapted into a bin packing or real-time task partitioning policy: assign a task to the core with the smallest load so far. The cores that currently have no tasks assigned to them have zero load and always participate as candidate bins. This differs from Worst-Fit-Decreasing policy which opens a new bin only when the current bins cannot accommodate the object.

Since the *list* policy minimizes the load and hence the frequency on individual processors, it should save energy. However, in Section 2.4.3 you will put this claim to the test and identify the assumptions made.

1.1.3 Energy management in Linux

The kernel subsystems in charge of energy management are `cpuidle` [PLB07] and `cpufreq` [PS06]. The former transitions the processor into sleep states and the latter sets its frequency.

Both subsystem abide by the principle of separating *mechanism* from *policy*. The platform-specific mechanism of transitioning the hardware into sleep states and changing frequencies along with the definitions of states and frequency tables is implemented as a `cpufreq/cpuidle` *driver* in the platform-specific directory. The policies, also known as *governors* are platform-independent and implement several heuristics based on system load.

2 Assignment

2.1 Writeup (10 points)

Submit a written report with the answers to the following questions. Please be brief and to the point.

1. (2 points) When and why would you want to compile your kernel in `NO_HZ` configuration?
2. (2 points) What kernel subsystem decides which sleep state the processor enters and what parameters does it use to make the decision?
3. (2 points) What are advantages and disadvantages of the *wake locks* solution adopted by the Android fork of the Linux kernel?
4. (2 points) In the kernel you are working on for Nexus 7, does voltage always change when frequency changes?
5. (2 points) In ARM architecture, what instruction can be used to enter a low-power processor state?

2.2 Energy Tracking (20 points)

2.2.1 Kernel Support for Energy Tracking (10 points)

Source code location: `kernel/rtes/kernel`

For each task with an active reservation keep track of the energy it used since a reserve was set on it. Enable this energy monitoring functionality only if at the time of setting the reserve the `/sys/rtes/config/energy` switch was enabled. Compute an estimate for the power and energy consumption using the model from Section 1.1.1.

Export the following values to userspace:

`/sys/rtes/freq`	MHz	processor frequency
`/sys/rtes/power`	mW	total power consumption
`/sys/rtes/tasks/`*pid*`/energy`	mJ	energy consumed by each task
`/sys/rtes/energy`	mJ	total energy consumed by the system (all tasks present and past)

Reading each of the above virtual `sysfs` files should return the latest value of the respective quantity, e.g.

```
$ cat /sys/rtes/power
250
```

The values should be up-to-date at any time when a task is not actively executing on the processor. Writing any value to `/sys/rtes/energy` should reset the total energy accumulator to zero.

2.2.2 Energy Monitor Native Application (5 points)

Source code location: `kernel/rtes/apps/energymon/`

Write a native application that prints to standard output the frequency, power, and energy quantities collected by the kernel (listed in Section 2.2.1) every second. It accepts one argument, `tid`. If `tid` is not zero, then the energy displayed should be the energy consumed by the respective thread; otherwise the energy displayed should be the total consumed by the system.

```
$ ./energymon 0
FREQ (MHZ)   POWER (mW)   ENERGY (mJ)
500000              500           500
500000              500          1000
250000              125          1275
```

2.2.3 Energy Monitor UI (5 points)

Display the frequency, power, and energy quantities collected by the kernel (listed in Section 2.2.1) in a table in your taskmon Android application. Refresh the values every second.

2.3 Energy-Efficient Frequency Assignment (20 points)

2.3.1 SysClock frequency (20 points)

Source code location: `kernel/rtes/kernel/sysclock.c` Implement a new `cpufreq` governor, `sysclock`, which would carry out the frequency assignment according to the *SysClock* algorithm. The frequency should be calculated and applied to all active cores at the time new reservation is added to or removed from the system, as well as when the user selects this governor.

2.4 Energy-Efficient Task Partitioning (40 points)

2.4.1 Turn off unused processors (10 points)

After a reservation is added to or removed from the system, bring offline all processors that are unused by tasks with active reservations. A processor should be brought online when the task partitioning heuristic cannot fit a task on currently online processors.

2.4.2 List partitioner (10 points)

Implement a fifth partitioning heuristic, LST, that assigns tasks to cores according to the *list-scheduling* policy outlined in Section 1.1.2.

2.4.3 Comparison experiment (15 points)

Using SysClock compare the total energy consumption over an interval of one minute for the WFD and BFD policies with that of LST policy for at least five randomly-generated tasksets of varying total utilization with at least five tasks in each.

For the same tasksets, also compare the total energy savings using ES-RHS+ and LST. This can be done by setting $\beta = 0$ and f as the maximum possible CPU frequency, in the model from Section 1.1.1. For the sake of simplicity assume that when the processor is in deep sleep mode, power consumption is zero, and each core can perform ES-RHS+ independently (may not be true in practice)

Present your results in the writeup as a plot of average energy vs total taskset utilization with at least five utilization levels. Each data point would be an average of total energy across five tasksets. Include an interpretation of your results.

2.5 Energy-Efficient Sleep Scheduling (Bonus 15 points)

Source code location: `kernel/rtes/kernel/es_sched/`

Note: Please refer to [Cor10] and [Doc] to get some background before attempting 2.5.2 and 2.5.3

2.5.1 Deep Sleep Governor (5 points)

Implement a new `cpuidle` governor, `deepsleep`, which provides the capability to transition the processor into *deep sleep*, every-time the OS requests the processor be put to sleep. Provide a brief justification in the writeup, whether this is a good choice for practical systems (both general purpose and real-time).

2.5.2 ES-RMS Schedulability (5 points)

Implement a sysfs based switch at `/sys/rtes/esrms`. When 1 is written to this file, perform admission control using ES-RMS, considering the deep sleep duration $C_{sleep} = 5ms$ and deep sleep period $T_{sleep} = 100ms$. Writing a 0 to the file should use the original RM-based schedulability tests (from Lab 3). The default value of the file should be 0.

2.5.3 ES-RMS Governor (5 points)

Implement a new `cpuidle` governor, `esrms`, which provides capability to transition the processor into *deep sleep* for 5 milliseconds every 100 milliseconds. Note that the sleep must be *uninterruptible.* Provide a brief discussion in the writeup, explaining how you implemented an uninterrupted periodic deep sleep. Ensure that when the `esrms` governor is enabled, the clock frequency always stays at the maximum supported frequency. You may also want to refer to the Tegra 3 cpuidle driver which can be found in the file `kernel/arch/arm/mach-tegra/cpuidle-t3.c`.

2.6 Demo (10 points)

Each group will also have to *demonstrate* their lab work in action to the TAs. The date and schedule of the demos will be announced via Piazza. Please show up on time with a clean and smooth demo prepared.

3 Tips

3.1 Frequency and sleep state control

- For an example of how to control frequency, take a look at `userspace cpufreq` governor implementation
- Take a look at `/sys/devices/system/cpu/cpufreq` and `/sys/devices/system/cpu/cpu`i`/cpufreq`

- Make sure to keep `auto_hotplug` from interfering

```
$ echo 0 > /sys/module/cpu_tegra3/parameters/auto_hotplug
```

- Make sure the current CPU cluster is the 'G' cluster, not the 'LP' (low power) companion core (see [Nvi11]): `/sys/kernel/cluster`
- The lowest frequency supported by the Tegra 3 System-on-Chip in Nexus 7 is 50 MHz. However, the default kernel that you are using caps the lowest frequency at 340 MHz. It is possible to remove this cap by adjusting a hardcoded value in the Quality-of-Service (QoS) subsystem, however it is not necessary to do so for this lab. Work with the 340 MHz as the lowest available frequency.

3.2 Energy tracking

- Take care not to include power for disabled processors into your calculations.

3.3 Synchronization

- To hold an object alive without having to be inside a critical section for the entire duration during which that object is used, reference counting is helpful. Reference counting consists of a `get` and a `put` operation. The `get` operation atomically looks up the reference to the needed object, increments the reference count and returns the reference. The put operation decrements the count and releases the object when the count reaches zero. Alternatively, the `get` operation might already take a reference and only increment its reference count. In this case, the user must lookup the reference and call `get` in a critical section. For example, see `get_task_struct` and `put_task_struct` and their usage.

4 How to Submit?

You must use `git` to submit all your work for this lab, including the written report. Please make sure all your commits to source trees are pushed to the corresponding repos:

kernel code	`18648/team`*number*`/kernel`
Taskmon app	`18648/team`*number*`/taskmon`
writeup file (incl. a compiled PDF if using markdown or LaTeX)	`18648/team`*number*`/writeups`

For the user-space Android application Java code, do not commit the `.CLASS`, any other auto-generated files, or binaries. Clean the projects before committing. Check the directory tree: `build.xml` should be in `taskmon/TaskMon`.

Make sure you don't miss any files (incl. makefiles) by checking `git status` before and after your commits. Finally, make sure your commits are pushed into the master branch of the remote repo by checking `git log origin/master`. The code that will be graded and demoed has to be in the remote repo master by the deadline. This means: push frequently as you complete each part, but always make sure that what is in the remote master builds successfully. If you want to share unfinished code, push into another branch of your making.

See Section 4.4 in Development Environment Setup handout for an example work flow.

5 CIT Plagiarism Policy

Please read and understand the CMU-CIT Plagiarism Policy. All work submitted for grading are subject to the policy, which will be strictly enforced.

References

[Cor10] Jonathan Corbett. The cpuidle subsystem, https://lwn.net/articles/384146/. 2010.

[Doc] Linux Documentation. Supporting multiple cpu idle levels in the kernel, https://www.kernel.org/doc/documentation/cpuidle/sysfs.txt.

[GMG] T. Gleixner, P. E. McKenney, and V. Guittot. Cleaning up linux's CPU hotplug for real time and energy management.

[Nvi11] Nvidia. Variable SMP: a Multi-Core CPU architecture for low power and high performance. 2011.

[PLB07] V. Pallipadi, S. Li, and A. Belay. cpuidle: Do nothing, efficiently. In *Linux Symposium, June*, 2007.

[PS06] V. Pallipadi and A. Starikovskiy. The ondemand governor. In *Proceedings of the Linux Symposium*, volume 2, page 215–230, 2006.

[SSV$^+$08] Vaidyanathan Srinivasan, Gautham R. Shenoy, Srivatsa Vaddagiri, Dipankar Sarma, and Venkatesh Pallipadi. Energy-aware task and interrupt management in linux. In *Ottawa Linux Symposium*, 2008.