SOSP: U: EnergyTimers — Integrating Physical Energy Measurement Devices into Operating System Kernels

Luis Gerhorst
Friedrich-Alexander University Erlangen-Nürnberg (FAU)
luis.gerhorst@fau.de

Abstract
High energy consumption is a problem both in mobile computing systems due to their limited battery capacity, as well as server systems with a constrained cooling capacity. Application developers can improve the energy efficiency of their software if they are able to monitor and profile the energy consumption to discover energy bugs. To support this, I have integrated an external energy measurement device into the standard perf_event subsystem of the Linux kernel, which allows the analysis of the energy consumption of applications running on the system under test (SUT). To enable low-overhead self-monitoring, my implementation employs energy timers: the measurement device only notifies the host system when an energy budget has been consumed. My evaluation shows that energy timers can be used to accurately measure the energy consumed by a workload and allow for an overhead-reduction on the SUT by 20 % to 51 % in comparison to regular timers, all while guaranteeing the same level of precision.

1 Problem and Motivation
In modern operating systems (OSs) there exist a variety of components dedicated to measuring and quantifying time. Processors offer counters that can be read to identify the current point in time, and also programmable chips (i.e., timers) that notify the processor, usually using an interrupt, when a specified amount of time has elapsed. For energy, however, which is a critical system resource for both small battery-powered devices [16, 27] as well as large scale clusters with thermal constraints [24], only a very limited set of built-in facilities exist to manage it. This is true, especially on the software side [11]. The Portable Operating System Interface (POSIX) defines a variety of interfaces that are related to time, but so far no interface is related to energy [12]. This is due to the fact that only few platforms offer a way to monitor the energy consumed by the system.

On recent Intel processors, the Running Average Power Limit (RAPL) interface allows monitoring of the system’s energy consumption with the help of hardware counters [6]. Using tools such as the Linux kernel’s perf [26], this allows userspace to determine how much energy an application consumes. The RAPL interface, however, is only available on recent x86 processors [17]. On embedded platforms and older Intel CPUs, energy measurements are usually not integrated into the hardware at all. And even on Intel processors that support RAPL, external devices are regularly required to monitor the system’s physical energy consumption, because only recently, RAPL has switched from model-based energy estimations to actual physical measurements [7]. To summarize, for accurate physical energy analysis custom hardware and software is the dominant measurement setup [4, 5, 9, 13, 18, 23]. This however, leads to results that are hard to reproduce and vulnerable to bugs, ultimately leading to measurement errors. As an universally available RAPL is likely to not exist soon, external energy measurement devices have to be integrated into operating system kernels to allow for reproducible, accurate and low-overhead physical energy measurement.

In this paper, I propose the simple but powerful EnergyTimer protocol, which enables OSs to implement a variety of services employing external energy measurement devices. I have integrated my protocol into the Linux kernel’s perf_event subsystem. My work allows measuring an application’s physical energy consumption with an external device, using the standard perf utility as interface.

2 Approach and Uniqueness
A generic measurement setup for a system that monitors its own energy consumption using an external device is shown in Figure 1. The device determines the power consumption by intercepting the power supply of the host system. The digitized power values are then integrated, producing the amount of energy consumed by the system. In naive custom hardware and software setups, the digital channel from the measurement device to the host system may, for example, continuously stream power samples to the host system. This however, leads to continuous overhead on the host system as it either has to integrate over power samples in real-time,
or store all power values, which can in turn induce noticeable overhead due to disk accesses. To avoid this problem, a dedicated system can be used to store the samples without generating overhead on the system under test (SUT) [9]. However, this requires complicated synchronization of the two systems to correlate software events with power profiles [18]. I propose that instead the following protocol is used:

- The host system instructs the measurement device to notify it whenever a certain energy budget has been consumed. It sets an energy timer.
- The measurement device continuously integrates over the power values and only notifies the host system when the previously defined energy budget has been consumed [5, 13].

The communication between the host system and the measurement device is illustrated by example in Figure 2. The protocol allows for a simple trade-off between precision, where the set energy timer is small, leading to a frequent interrupt, and accuracy, where the interrupt is infrequent, causing less overhead. This allows it to be applied in a variety of domains each with different classes of workloads. Besides being broadly applicable, energy timers also greatly simplify the budgeting of energy usage with regard to regular timers, where the user has to frequently poll whether the budget has been consumed already.

3 Results and Contribution

To demonstrate the soundness and practicality of my approach, I have implemented it on the SAMA5D3 Xplained [25] embedded Linux system in which the measured energy consumption is made available to userspace using the perf tool. The design and usage of the measurement system is described in Section 3.1. Thereafter, Section 3.2 evaluates the overhead and accuracy of the physical energy measurements performed using my tool.

3.1 Implementation

My implementation of the EnergyTimers protocol includes both a kernel module that communicates with the external measurement device, as well as a simple firmware for an AVR microcontroller which constitutes the measurement device. For current and voltage measurements, the microcontroller employs the LTC2991 current and voltage analog-to-digital converter (ADC) [8]. In my prototype, the microcontroller and the host system communicate using a serial line. This has the advantage that the overhead is kept to a minimum in comparison to, for example, a network connection, while the interface is still available on a variety of devices. The communication protocol executed over the serial line employs tested and well-documented methods whenever possible, for example, the Serial Line Internet Protocol (SLIP) is used for framing [19], and XMODEM’s CRC-16 is used to ensure data integrity [3, 10]. To minimize the amount of data, and therefore overhead, data is encoded in binary form. Figure 3 displays an overview over the interacting subsystems when my kernel module is used for energy measurements. The module registers both a line discipline as well as a perf performance monitoring unit (PMU). When a new measurement device is connected to a serial port on the host system, the user simply attaches the custom line discipline to that serial port, using for example, ldattach [20]. This causes data received on this port to be interpreted as EnergyTimer packets which constitute the notifications from the measurement device about consumed energy budgets. In response to these, the perf counter is updated. Note that the bytes received on this particular serial line are not copied to userspace. The

---

1The source code is available at https://gitlab.cs.fau.de/i4/pub/energytimers under an open-source license.
module exports sysfs files that allow userspace to control how often the system is notified by the measurement device. Notifications can either be energy or time-driven, that is, the system is either notified whenever a set timer elapses or whenever a certain amount of micro joules has been consumed. To ensure portability, the code implementing the protocol is encapsulated in a library which at this point is already used by both the kernel module and the AVR firmware. Using my work, an application developer can measure the energy consumed by their application simply by entering the command shown in Listing 1.

Listing 1. Using `perf stat`, application developers can easily determine the amount of energy a task consumed using the EnergyTimers (et) PMU.
```
perf stat -e et/energy/ my_app
# -> e.g., 1.664 joule
```

3.2 Evaluation
To confirm the correctness of my implementation, I have measured the power consumption of known workloads on the SAMASD3 Xplained board. Figure 4 shows the energy consumed by the processor, the embedded memories and the peripherals\(^2\) while idle and during the Dhrystone benchmark, measured using an energy timer of 8.0 mJ. Each measurement presented in this section represents the mean value calculated from 10 samples. The variation was always below the displayed resolution (i.e., point size for graphs and number of digits for numbers). While idle, my tool reports a mean energy consumption of 68.2 mW. During the Dhrystone benchmark a mean power consumption of 166.4 mW is reported. This closely matches the power consumption during Dhrystone reported by Atmel in the board’s datasheet, which is 162.5 mW \([2]\).

To guarantee the same level of precision when keeping track of the energy consumed, energy timers require less overhead on the host system than regular timers. Energy timers guarantee to the processor, that since the last notification, at most the set energy budget has been consumed. When a regular timer is used instead, the processor can not know whether the pending interval is one with large or small energy consumption. Therefore, to guarantee that at most a certain amount of energy was consumed in the meantime, the user has to assume that the maximum power draw possible occurred since the last notification. This causes the timer interval derived to be unnecessary small as the device rarely consumes that much power during a typical workload. The small interval causes more frequent interruptions to the SUT and therefore hurts measurement trueness. To demonstrate this advantage of energy timers, I have measured the energy consumption of a fixed workload using both energy timers between 0.8 mJ and 40.0 mJ, as well as regular timers between 4 ms and 200 ms. Unfortunately, Microchip does not list any information on the maximum power draw of the core components in the datasheet or user guide of the SAMASD3 Xplained board \([1, 2, 25]\). Still, this information is required to determine the guaranteed level of precision for energy measurements when regular timers are used. Therefore, the user has to estimate the value based on additional measurements.

\(^2\)Measured by intercepting VDDCORE using JP1. As described in the board’s user guide \([1]\), the incorrect JP1 routing was fixed.
Energy timers in turn, do not require this information in the first place which is another advantage. My measurements show a maximum power consumption for the board of at least 178.0 mW. Given some headroom I therefore estimate 200.0 mW to be the upper limit. Assuming this maximum power draw, Figure 5 displays how energy timers compare to regular timers regarding the overhead required for a given guaranteed level of measurement precision. Overhead constitutes itself in the number of notifications to the host system that update the energy counter maintained by perf. The guaranteed precision of the measurement is simply the set energy budget, or if regular timers are used, the timer interval multiplied with the previously estimated maximum power draw. Because of the overestimation that is required for regular timers, energy timers allow for lower overhead than regular timers while guaranteeing the same level of precision.

4 Future Work

In future work EnergyTimers may be used to implement a variety of other OS services besides simple energy measurements. Using EnergyTimers, a scheduler can, for example, preempt applications based on energy budgets not CPU time budgets as illustrated in Figure 6. Also, interrupts occurring every N joules can be used to determine where in an application most of the energy is consumed [5, 21]. Figure 7 illustrates how increased energy consumption in a subroutine causes a greater number of profiling samples be collected for the code section when the energy timer is sufficiently small. For this, the interrupt handler must record the current context, for example, the current value of the instruction pointer, for later analysis which may be implemented using perf’s sampling capabilities.

5 Related Work

Le Sueur and Heiser demonstrated that generic processor features like dynamic voltage and frequency scaling (DVFS) as well as C states (sleep modes) can save power [15], but also that the power savings drawn from DVFS are limited [14].
6 Conclusion

This paper presented energy timers, which are a powerful concept that allows for the integration of external energy measurement devices into OS kernels. I have integrated energy timers into the Linux perf_event subsystem, giving users a convenient and powerful interface to perform physical energy measurements. My evaluation shows that energy timers both allow for accurate measurements, but also that they outperform traditional timers regarding their guaranteed precision. In future work, energy timers may be used for energy profiling and also energy-driven scheduling.

References