

An Analysis of Multiprogramming Microcontrollers

Robert Sunshine

Florida International University

An Analysis of Multiprogramming Microcontrollers

A microcontroller is a small computer on a single integrated circuit. It's not as complex as the standard home computer or smartphone, they usually only deal with kilobytes of data and small amount of power down to nanowatts. Microcontrollers are designed for use in embedded applications, automatically controlled devices that generally only do a single job. These systems are used in such a manner that they will still be functional while in sleep mode running off very miniscule amounts of power, very suited for longevity without needing to be maintained.

While these microcontrollers are suitable for single purpose devices, in recent years newer embedded applications are becoming a platform for multiple programs to run on, and microcontrollers lack the hardware to support these new demands. Examples like sports watches that can possibly install third party software using the same limited hardware or wireless headphones that can sync to devices through Bluetooth and also transmit other information such as battery life and play speed. While many current operating systems for these embedded systems are trying to provide for these needs, they lack major aspects in both efficiency and security. A common security concern is the fact that the hardware often uses the same memory for the OS and applications, this lack of memory isolation means that any errors in code could threaten the whole system. A common theme among operating systems for low-power embedded systems is to optimize software to make up for the limits in hardware. Unlike full multiprocessor systems, microcontroller systems don't advance as quickly over time due to the limitations being different. They aren't able to use large amounts of space or energy due to limits in how much energy the system is able to access during sleep. Because of such limits, low-power embedded systems are lacking in security and efficiency, only being able to provide simpler models to prevent excess overhead. This demand has spawned new solutions, such as the new operating

system Tock that takes advantage of recent hardware improvements to the microcontroller environment to allow multiprocessing while maintaining similar or even improved memory usage compared to monolithic operating systems. Tock is presented as a solution to some common problems of low-power embedded systems by introducing new methods that address the limitations of the system and provide a platform to implement multiprogramming structures.

The way Tock is able to do this is through a combination of the type-safe programming language Rust and new advancements in low-power embedded systems. The Tock OS is able to implement features that were previously not capable such as fault isolation and dynamic memory allocation. Tock also allows the kernel to use portions of the kernel memory, called *grants* that are able to allow processes to be partitioned so that if one process crashes it can easily be killed without threatening the other processes. It's impressive that so many new features were able to be implemented, the current state of the OS for embedded systems seemed to be that each different one had benefits the others lacked, with no single OS having the benefits of all of them. Through the synergy of hardware and software, Tock is able to utilize the limited resources and maximize its efficiency. By having the infrastructure's language be designed in such a way, the OS is able to ignore some of the checks other OS are required to do since Rust's type-safe language prevents the kinds of errors found in other OS

While Tock isn't the only OS that supports new hardware features such as the memory protection unit (MPU), it's designed to use mutually distrustful applications. This allows the kernel to not be forced by any application that may try to read or write kernel memory. Tock is able to isolate the kernel from the applications and the applications from each other, providing a safer and even a more efficient environment.

Newer embedded applications that provide multiprogrammable systems require that the OS support five key features: concurrency, dependability from resource exhaustion, fault isolation, memory efficiency, and application updates at runtime. Although current systems support some of these features, it is touted that Tock is able to support all 5. The Tock architecture is able to do this by using “capsules” and “processes”. Capsules are within the kernel, which is what allows it to be isolated and distrustful of programs. It is able to do so through Rust, creating software based structs that can pass multiple capsules in a single struct. These capsules are untrusted by the system due to their software nature, with a small few being an exception. This type of structure eliminates a lot of overhead and ensures the kernel remains safe. Processes act similarly to other OS processes, being memory-isolated through the hardware and as a result can be written in any language. They have this advantage over capsules, as well as having preemption that allows for long running computations. It seems like the limitations of this kind of hardware has forced a new perspective on how to build this kind of software. Instead of leaving some processes to hardware, having an equivalent in software form can bypass some of the limits of the medium.

Although Tock is able to keep these processes isolated and be more secure, it does not try to stop any single kind of threat that may pose itself to the system. Instead, it provides a platform that creators can tailor in order to fit their needs, making Tock more versatile. With this strict separation between the kernel, drivers, and applications, Tock is able to provide a level of security that hasn't been capable before in other operating systems, and when compared overall it has similar if not better levels of memory and time usage. The lack of security in these embedded systems is a growing concern with low-power embedded systems being implemented in more and more technology. Along with a risk to privacy is a risk for failure that can be

prevented through the use of isolation and implementing flexible multiprogramming. While Tock may be a good platform that provides multiprogramming, any legacy code from other operating systems has to be imported to rust, as well as adjusted to take advantage of the new systems Tock provides such as grants. Although the lack of resources will continue to be a source of struggle for system designers, Tock has shown that it is possible to increase the capability of low-power embedded systems.

References

- Levy, A., Campbell, B., Ghena, B., Giffin, D. B., Pannuto, P., Dutta, P., & Levis, P. (2017). Multiprogramming a 64kB Computer Safely and Efficiently. *Proceedings of the 26th Symposium on Operating Systems Principles - SOSP 17*. doi:10.1145/3132747.3132786
- T. (2018, June 27). Tock/tock. Retrieved from <https://github.com/tock/tock>
- Levy, A., Ghena, B., Andersen, M., & Campbell, B. (n.d.). Tock Operating System Safety wIthout Processes for Embedded Systems. Retrieved from <https://forum.stanford.edu/events/2016/slides/iot/Amit.pdf>
- Bradjc. (2018, June 20). Tock Userland has Graduated. Retrieved from <https://www.tockos.org/blog/2018/libtock-c/>