

Multiprogramming Microcontrollers Safely and Effectively:

An Analysis on Multiprogramming a 64 kB Computer Safely and Efficiently

Grace E. Deehl

Florida International University – College of Engineering and Computing

COP 4610 - Principles of Operating Systems

Professor Jose Osorio

Summer C 2018

*Multiprogramming Microcontrollers Safely and Effectively:**An Analysis on Multiprogramming a 64 kB Computer Safely and Efficiently*

The following report will detail a thorough rundown of the paper *Multiprogramming a 64 kB Computer Safely and Efficiently*. The paper, presented by several research students at Stanford University, the University of Virginia, and the University of California, Berkeley, showcases a new operating system Tock. The operating system Tock was designed to provide “a rich multiprogramming environment” free of the issues that ordinarily arise in currently implemented systems. (Levy 2013)

Introduction

First and foremost, the paper does an excellent job at communicating what Tock does that other embedded operating systems haven’t been able to do that optimize its ability to perform safely and efficiently. The paper provides data-driven comparisons to the currently available systems and compares and contrasts each’s strengths, weaknesses, and sacrifices made. The comparisons are made by dividing system functionalities into five attributes that work hand-in-hand to support multiple applications in very small memory spaces: concurrency, memory efficiency, dependability, fault isolation, and loadable application. Often times, as shown in the paper in several other operating systems, the implementation of one of the functionalities will lead to tradeoffs that sacrifice other desirable and often essential features. After describing what is being done correctly and ineffectively, this analysis will detail what Tock does to work around any pitfalls and tradeoffs. Once it is clear what attributes need to be supported to make Tock a desirable embedded operating system, the authors go on to detail what ensures the safety of the application data that is being run through Tock.

In order to analyze the implementation of Tock, I will discuss each shortcoming of small computing systems that it aims to overcome and each problem they are trying to circumvent by addressing whether or not it is indeed overcome. If the particular shortcoming is overcome, I will discuss why their method is effective. If it is not, I will discuss why and include potential solutions that would overcome the limitation successfully.

The paper claims that Tock eliminates the need for a trade-off for effective system features. Unlike other computing systems designed for small memory sized machines, Tock provides for all five embedded operating system criterion. The subsequent sections will detail each of the operating system properties, describing what they are, how Tock claims to account for each, and whether or not it successfully does so and why. Afterwards, I will discuss the intention of potential Tock implementations, showcasing its versatility, while touching upon what could be done in the future to increase all functionalities required by society's ever-changing technology.

Key Features of Embedded Operating Features

Fault Isolation. Having such limited resources makes it hard to rely on hardware for general purpose computing. Embedded operating systems in microcontrollers traditionally do not provide fault isolation. In such low memory bearing microcontrollers, where such a small address space is shared by so many different components, standard fault isolation techniques aren't effective as they lack memory protection from applications corrupting other applications' states. Dealing with the allocation of any amount of memory can lead to system instability. Driver isolation techniques that are found in current general use machines are found to be far too resource-heavy.

In order to implement isolation for safety on a system of this size it must come at little to no overhead cost. Each application must be inherently distrusting of all other applications to stay secure and stable. Tock provides memory safety through the use of capsules in its kernel. Capsules are instances of a Rust struct which can be scheduled concurrently. Capsules can share a single stack. Isolation is forced by the type-system provided by Tock at time of compile. The safety of capsules come from their innate trust for being active. However each capsule is distrustful of others, keeping component safety. They are run from system calls and hardware events. The capsules are independent on a shared stack. This removes the need for a heap.

Memory Efficiency. As resources are so limited in these types of microcontrollers, it is evident that conservative resource management is essential for running without any unnecessary delay or bottlenecks. Operating systems tend to opt for dependability by allocating an adequate amount of memory which ultimately tends to lead to a great waste of it, as the paper details that “in typical use cases, 80% of RAM is wasted.” (Levy 2017) Tock’s Grants provide for memory efficiency by requiring small, fixed overhead for the kernel and only providing memory to capsules via ‘granted’ requests. Grants ensure the independence of memory for one process versus other allocated process memory. They’re the focal point of eliminating the tradeoff of safety versus flexibility.

Concurrency. A system with multiple cores can provide for threads to run in parallel and work to completion while interleaving overtime. (Gagne 2013) In order to provide concurrency, Tock implements Capsules and Grants. Capsules communicate with each other straightly through global variables and functions. Capsules follow concurrency by addressing the kernel directly

through “normal control flow”. (Gagne 2013) As well, Tock assesses the number of events present the amount of time needed to statically establish the scheduling queue. This method addresses both memory efficiency and concurrency.

Avoiding Tradeoffs. / Efficiency vs Concurrency. Tock allows the kernel to use granted portions of capsule memory, which act as “dynamic kernel heap” that prevents starvation. Donation of available memory. Both Grants and Capsules eliminate the need for tradeoffs in performance quality.

Loadable Applications. In Tock, applications are able to be dynamically loaded. There is a region granted (in a grant) for each process which is dynamically allocated depending on requirements of the specific process.

Dependability. It’s difficult to sign off on any system that isn’t dependable in that it will run without needing any mid-process involvement. Tock’s dependability is ensured by following the previous four system fundamental attributes as each provides stability and reliability in resource and software management. Dependability comes with the guarantee of safety. Using Rust as a programming language allows for memory and type safety. Processes are allocated dynamically and have a dedicated memory area and use context switch for communication which ensures safety.

Potential Issues. The security issues I feel are not mentioned are multiple kernels, security protocols. I feel that Tock meets the need for niche uses as indicated by the authors and

their lack of tradeoffs aids in the tasks mentioned but do not provide a broader scope as to the continued implementation of Tock.

References

- Butta, Prabal., Campbell, Bradford., Ghena, Branden., Giffin, Daniel. B., Levis, Philip. Levy, Amit., Pannuto, Pat. (2017). Multiprogramming a 64 kB Computer Safely and Efficiently. *Proceedings of SOSPP '17*, Pages 1-18.
- Gagne, Greg., Galvin, Peter B., Silberschatz, Abraham. (2013). Operating System Concepts, Ninth Edition. *John Wiley & Sons, Inc.*