Proposal – 2023

Title:

- **Micro-kernalisation de Linux/ Micro-Linux Kernel: Building a Message-Oriented Middleware for Kernel space – User space communication**

Advisors:

- Alain Tchana (Grenoble INP – alain.tchana@grenoble-inp.fr), ERODS team, LIG lab, Grenoble
- Renaud Lachaize (UGA – renaud.lachaize@univ-grenoble-alpes.fr), ERODS team, LIG lab, Grenoble

General context and problem statement

The scheduler is one of the main components of an operating system (OS).
Its role is to make frequent decisions about the allocation of the CPU cores to the different threads (execution flows) that compete for them. The efficiency of OS scheduler has a critical impact on the overall performance of the applications. Several works have shown that it is not possible to design a one-size-fits-all scheduling policy that is well-suited for all the workloads. This is why the Linux OS (like many others) provides several "scheduling classes" (e.g., "Fair", "RR", "Deadline", etc.). However, even such a hierarchy of classes is not sufficient to accommodate the needs of all applications – within the same class, different policies can be envision to take into account the specific characteristics of various workloads. CFS is the implementation of the "Fair" scheduling class in Linux. CFS has been designed to accommodate the requirement of general-purpose applications, which makes it sometimes inefficient for other types of emerging applications such as, for example, data-intensive & I/O-sensitive applications. Moreover, new types of workloads are appear on a quite regular basis.
In order to take into account the diversity of existing workloads and their needs, the CFS code base is modified regularly. Researchers propose new optimizations or even completely new schedulers (designed from scratch). However, most of the new proposal are never integrated into the "vanilla" Linux code base (the version used/maintained for production) given the very complex integration and quality-control procedures that are required by such a popular and general-purpose operating system. Moreover, the development and debugging of the code base of a new/modified scheduler are difficult and error-prone, as it is typically written in C and running in kernel mode. In addition, deploying a new scheduler (or new version/release) requires to reboot the machine, which is problematic in the context of a large-scale datacenter: the downtime of machines introduces management burden, increased resource consumption and risks, which the system administrators would like to reduce.

Solution

In order to facilitate the integration of new schedulers in production-grade operating systems, two main approaches have been proposed: user-defined kerner-level schedulers and user-defined user-level schedulers.
In both cases, the datacenter operator is in charge of choosing/developing and deploying its own scheduling policy. In the first cases, the policy is injected (as a plug-in) into the OS kernel. In the second case, the custom scheduler runs in user mode (like an application) and interacts with the kernel to leverage its internal mechanisms (e.g., context switching) – only the policy is delegated to user space. The project described in the present document takes place in the context of the second approach, because it allows the following benefits: (1) leveraging conventional debugging tools, (2) easily replacing the current scheduling (with a simple process launch/restart), (3) using a high-level programming language for the development of the scheduler code.
This year, researchers form Google and Stanford have published a new contribution named ghOSt[1] at SOSP, one of the most prestigious conferences in the field of operating systems.

GhOSt is a framework allowing to build scheduler running in user space for the Linux operating system. This framework has been heavily used in production by Google and can hence be considered as reliable. The code of ghOSt is available as open source [2][3]. ghOSt opens the door to the study and exploration of many custom scheduling strategies.

Generalization

This approach has been explored in the past in the context of micro-kernels[11]. It is revisiting in the context of Linux, a real functioning and massively popular monolithique OS. Remzi and Andrea Arpaci-Dusseau published in 2021 uFS[7], a userland file system relying on SPDK[8]. Other recent works following this trends mainly focused on the network stack (lwIP[9]+DPDK[10]).

In the remaining of the document we call MiLK this approach.

Goals

The implementation of MiLK requires a key component, which is a communication framework between the kernel space (where mechanisms lie) and the userspace (where the new policies run). The purpose of this project is twofold:

1. Benchmarking of existing kernel space – userspace systems.
2. Identify the key properties of a Message-Oriented Middleware (MOM) for kernel space – userspace communication in the context of MiLK.
3. Design and implementation of a MOM for the purpose of MiLK.

Bibliography

[1] Jack Tigar Humphries, Neel Natu, Ashwin Chaugule, Ofir Weisse, Barret Rhoden, Josh Don, Luigi Rizzo, Oleg Rombakh, Paul Turner, Christos Kozyraki. ghOSt: Fast & Flexible User Space Delegation of Linux Scheduling. SOSP 21.

[2] ghOSt kernel: https://github.com/google/ghost-kernel

[3] ghOSt user space components: https://github.com/google/ghost-userspace

[4] Cody Cutler, M. Frans Kaashoek, and Robert T. Morris. The benefits and costs of writing a POSIX kernel in a high-level language. OSDI 18.

Additional references

[5] Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, Nathan C. Burnett, Timothy E. Denehy, Thomas J. Engle, Haryadi S. Gunawi, James A. Nugent, and Florentina I. Popovici. Transforming Policies into Mechanisms with Infokernel. SOSP 03.

[6] Georgios Chatzopoulos, Rachid Guerraoui, Tim Harris, and Vasileios Trigonakis. Abstracting Multi-Core Topologies with MCTOP. EuroSys 17.

[7] Jing Liu, Anthony Rebello, Yifan Dai, Chenhao Ye, Sudarsun Kannan, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau. Scale and Performance in a Filesystem Semi-Microkernel. SOSP 2021.

[8] https://spdk.io/

[9] https://savannah.nongnu.org/projects/lwip/

[10] https://www.dpdk.org/

[11] https://os.itec.kit.edu/downloads/da_2008_reichelt-sebastian_os-decomposition.pdf