Operating Systems: are we finally ready to move forward after 30 years of stagnation?

Antônio Augusto Fröhlich
GMD-FIRST
Kekuléstraße 7
12489 Berlin, Germany
guto@first.gmd.de

Wolfgang Schröder-Preikschat
University of Magdeburg
Universitätsplatz 2
39106 Magdeburg, Germany
wosch@ivs.cs.uni-magdeburg.de

Abstract

If there are areas of computer science that were left behind by the market, operating systems is certainly one of them. The products currently available in the area, namely Microsoft WINDOWS and UNIX, owe their designs to projects such as THE [1] and MULTICS [4], which are at least 30 years old. In the meanwhile, several new ”hacks” have been proposed, but very few systems brought about revolutionary designs. The partitioning of the problem domain in abstractions like `process` and `file` is so old that some people regard it as a ”canonical” one. The scene is even worse if one takes in consideration that only 2% of the microprocessors produced in the year 2000 targeted the interactive market [6], for which those operating systems have been designed. The software industry spent 30 years saying that there are not many things to be improved in the operating system area [5], but left 98% of market without a choice!

Innumerable software engineering techniques have been proposed for the development of applicative software, many of them are now mature and widely used in production. Nevertheless, not many operating system developers have tried to deploy them, or to adapt them, to the development of their products. We believe that several of these techniques can be successfully deployed in the area of operating systems with minor adjusts, as long as we reformulate our view of the corresponding domain. Of course we are dealing with a very special field. A field where words such as asynchronism and determinism have very special meanings. A field pressed to squeeze the last bit of performance and to blow overhead away without relaxing on correctness. Anyway, by now we should be ready to make it up with software engineering and move forward with operating system design.

We developed a novel operating system design method that addresses many of the questions raised above. Deeply influenced by *object-oriented design*, but also by *family-based design*, *collaboration-based design*, *aspect-oriented programming* and *generative programming*, our method enables the development of operating systems as an assemblage of reusable and adaptable components. Our method produces systems that can be tailored to fulfill the requirements of any particular application, without disregarding any of the quality metrics compulsory to the field. Indeed, applications play such a major role that we decided to name the method *application-oriented system design*.

In summary, our method conducts the partitioning of the problem domain in *scenario-independent, application-ready abstractions* that will shape the components of the resulting system. Because these abstractions know very little about the execution scenario they will join, they can be adapted to join several scenarios. When performing a scenario, abstractions are wrapped by *scenario-adapters*, which know details of both scenario and abstractions. In order to avoid overloading users with system-level decisions, all implementations of an abstraction are made visible through the same *inflated interface*. If the system is proper designed, tools can automatically select the best implementation via syntactical analysis of the application’s source code.

This combination of objects, collaborations, and aspects may sound low-performing. However, our method proposes abstractions to be arranged in a *statically metaprogrammed component framework*, so that compositions are carried out during compilation, resulting in virtually no run-time overhead. Such frameworks define the relationships between abstractions in terms of scenario-adapters, hence capturing a reusable system architecture. If the target application does not need this or that abstraction, the corresponding scenario-adapters will contribute for a scenario free of them. In this way, each application gets

exactly the operating system it needs. Yet, system developers do not need to redesign or reimplement the system several times, they can play with components, adapters and frameworks to deliver a vast range of application-oriented systems.

We deployed this design method in the EMBEDDED PARALLEL OPERATING SYSTEM (EPOS) project under development at GMD-FIRST [2]. The current implementation targets parallel applications running in a cluster of PCs interconnected through a MYRINET high-speed network. Although the number of components in the repository is still small, and the tools are relatively primitive, EPOS first results are very encouraging, not only in terms of performance, but of quality in general. As far as we are concerned, application-to-application communication over MYRINET in EPOS has the best performance ever reported [3]. Besides continuing the development for the cluster domain, we will soon deploy application-oriented system design to the embedded systems domain.

References

[1] Edsger Wybe Dijkstra. The Structure of the THE-Multiprogramming System. *Communications of the ACM*, 11(5):341–346, May 1968.

[2] Antônio Augusto Fröhlich and Wolfgang Schröder-Preikschat. High Performance Application-oriented Operating Systems – the EPOS Aproach. In *Proceedings of the 11th Symposium on Computer Architecture and High Performance Computing*, pages 3–9, Natal, Brazil, September 1999.

[3] Antônio Augusto Fröhlich, Gilles Pokam Tientcheu, and Wolfgang Schröder-Preikschat. EPOS and Myrinet: Effective Communication Support for Parallel Applications Running on Clusters of Commodity Workstations. In *Proceedings of 8th International Conference on High Performance Computing and Networking*, pages 417–426, Amsterdam, The Netherlands, May 2000.

[4] Elliott Organick. *The Multics System: an Examination of its Structure*. MIT Press, Cambridge, U.S.A., 1972.

[5] Rob Pike. Systems Software Research is Irrelevant. Online, February 2000. [http://cm.bell-labs.com/who/rob/utah2000.ps].

[6] David Tennenhouse. Proactive Computing. *Communications of the ACM*, 43(5):43–50, May 2000.