Manifesto

Written by Alexandre Becoulet, Joël Porquet and Nicolas Pouillon.

What is MutekH?

MutekH is the kernel part of an Operating System. Its major characteristics are portability and modularity, which are both necessary for handling multiprocessor platforms with processor heterogeneity in a native manner.

MutekH is first and foremost a research project: it must remain a tool for experimentation. Nowadays, thinking it is possible to replace kernels such as Linux would be crazy as its development costs are estimated at ?several hundreds millions dollars. Even though, this is not completely true for embedded systems yet.

Whether MutekH should become something other than a research project is not the point. For now, it is a research project because people use it this way. But as we make efforts to maintain a high quality code, documentation and website, the choice is no longer ours. The situation can change tomorrow, or never. This is actually a matter of users rather than design. Besides, it is actually possible MutekH is used somewhere else without our knowledge.

Its exo-kernel architecture was first suggested by Frédéric Pétrot, early in the development process. It was chosen because it was well-adapted to the modularity requirement and because it remains an interesting research field.

Several PhD students soon started using MutekH for their own research works. MutekH definitely doesn't belong to the sole original author, thanks to the number of contributions others integrated. Lot of additional work has also been achieved by the means of internships. This has increased the number of services offered by the kernel, opening new usage perspectives and facilitating various aspects of development.

Why yet a new kernel project?

MutekH development was started as an attempt to write the first kernel capable of handling processors heterogeneity in shared memory platforms. This means MutekH should run on multiprocessor platforms composed of different processors types, with possibly different instruction sets, and still share all global variables and in memory data, allowing to run classical multi-threaded applications without relying on a specific API.

Heterogeneity constraints implied a lot of attention to the development of the hardware abstraction layer (HAL). MutekH was first developed to end up being a tiny project, that's why it appeared it was a better approach to write a new HAL from scratch to handle heterogeneity with extreme portability constraints.

Unexpectedly, MutekH has been adopted by others, as explained above, for various research projects, not related to processor heterogeneity. That is why the project really started to grow.

Why is MutekH well-suited for research?

Large projects are sometimes too complex to be used as experimentation playgrounds. As a researcher, you generally want to quickly setup new modules and features on a preexisting base project and focus on your research goal. This base is typically provided by modular projects such as MutekH because you can choose interesting modules and extend them with new ones, to get the minimal suited set of features to perform experiments. The integration effort in large and widespread projects can then be considered later when the experiment cycle is over and results are conclusive.

We believe this approach is well suited when applied to research and kernel concepts. The older OSKit project followed the same idea and actually had a great success. In opposition, many have tried to compete with existing

kernels, and have only ended up adding their attempt to the endless "Yet Another..." list.

Furthermore, the Lip6-SoC laboratory where MutekH has originated is quite familiar with development of research tools. The ?Alliance CAO suite and ?Disydent project were good examples of such free software tools. More recently the development of ?SoCLib, a hardware simulator and components library, was initiated in an attempt to provide a modular framework for building multiprocessor System-on-Chip virtual prototypes. The modular design is the key point of the SoCLib project as it enables researchers to build their own platforms by picking needed hardware component models. MutekH aims at genericity and modularity with the exact same goals, but applied to kernels design.

Is MutekH a pedagogic tool?

MutekH is not a kernel for teaching beginners.

Should MutekH specialize?

No. MutekH should not specialize in a particular field, for several reasons:

- We are fortunate enough to have developed a kernel with a very modular and configurable design. This is partially a side-effect of the mandatory portability needed for heterogeneity but it is also due to the great diversity of other initial developers' research work.

- We need to keep this broad-spectrum experimentation platform aspect. This way it can be of interest to most people as it has been the case until now. When you need a kernel to perform research, you generally try avoid loosing time reinventing the wheel and want to reuse existing facilities, such as a buildsystem, drivers for peripheral devices, libc, services, etc. It is likely that these key elements have already been tested, debugged and validated by others, or even simply imported from other well-tried projects.

- In a research context, the versatility, genericity and collaborative aspect of MutekH guarantees that you are working in realistic conditions. Otherwise, you could end up solving problems in conditions unlikely to arise in the real world or with a demonstrator only usable for a specific purpose.

- A system intended for research should above all show interest in the generic programming issue. This opens numerous research perspectives in terms of kernel architecture or software design patterns, even though it is not the main topic in our lab. The exo-kernel architecture is a real goldmine in this prospect and yet remains not fully investigated.

Other kernels such as Linux or NetBSD are able, by taking care of the software modeling, to run on very various systems, from Set-top boxes without any MMU to Cray, equipped with 224556 processors and 62TiB memory. Though, this achievement is reached with the same source code, an huge list of features and without any trade-off on performance. De facto, it has never even been considered to rewrite kernel projects from scratch just to solve new issues that arise with hardware evolution.

Therefore, it is definitely realistic to achieve the exact same performance with our small kernel.

What is the development process?

Lots of various problems come up. Basically, algorithms to solve them are implemented accordingly to the complexity and specificity of the underlying hardware.

For a kernel such as MutekH which aims at covering a broad-spectrum of platforms, we need to act this way:

- Use a source code configuration tool that manages conflicts and dependencies efficiently.
- Team up, and consider issues and constraints of all developers to converge on the most suitable API which thus will guarantee the best abstraction. MutekH would certainly not be as it is now, if it has been a one man project with an inevitable narrower vision.
- For the parts which deal with the hardware abstraction layer, make sure solutions are portable for every different architecture we have at our disposal. That is why we insisted right at the beginning of the project to target x86 CISC as well as RISC processors.
- Rely on advanced software engineering and modeling to implement all these abstractions without penalizing performance: inlining, conditional compilation, object oriented approach with function pointers, preprocessor meta-programming, etc.
- Iterate several times before finding the best solution, as needed. It is unreasonable to believe one can solve an issue definitively and freeze an abstraction API. Indeed, Linux kernel hackers ?explain it clearly.

How does MutekH compete with other embedded OSes?

There are certainly many other comparisons that could be done, and many valuable kernel projects out there, but here are a few interesting facts:

- MutekH is a free project, fundamental for public research, contrary to VxWorks or QNX.
- MutekH provides native and classical support for shared memory multi-processor platforms, in opposition to embedded kernels like RTEMS or FreeRTOS.
- MutekH has a native support for processors heterogeneity. This is an **exclusive** feature.

How does MutekH compete with other monolithic UNIX-kernels?

It probably doesn't compete and is quite different:

- MutekH can run applications in kernel mode, without any protection or syscall API: neither Linux, nor uCLinux, nor *BSD are designed this way.
- Exo-kernel approach enables implementation of syscall API via optional "libOS", for example for Unix/Posix?. Others can be considered and implemented, such as L4 for instance.
- Exo-kernel approach even allows to co-host kernel applications with user applications.
- MutekH is still a rather small kernel and is not intended to be used on a workstation. This allows newcomers to get ready quickly, and researchers to quickly validate experimental concepts. This may not be the case for typical big kernels depending on research topic. Even the original author of Linux kernel considers now, that his project ?is getting bloated.

Who can get involved in MutekH?

"Qui m'aime me suive !" (Let he who loves me follow me)

Anyone with sufficient technical skills in System & C programming, who likes collaborative work, is most welcome.

Where does original author choices come from?

My choice are certainly influenced by my experience in multiple areas. Of course, I won't try to write my resume here but I'll still try to explain where my ideas come from:

- In the context of collaborative and open-source software development, MutekH is not my first shot. I've authored several other open source projects which sometimes still draw more external contributors and users than MutekH. I know which causes produce which effects in this kind of project management, because I experimented myself and I also follow other projects' hazards.

- In the specific frame of kernel development, this is not my first shot either. In the past, I already implemented a OS for x86 which was approximately the same size as MutekH's, wrote a few drivers and hacks for Linux, wrote several significant patches for OpenBSD (nothing published), and recently, I ported few other OSes such as RTEMS and eCos on the SoCLib hardware simulator. All those achievements allowed me to experience kernel designs more closely and thus to understand common criticisms and praises.

- On a technical point of view, I pay close attention to every advanced programming techniques which empower abstraction, extreme factorization, generic programming and code reuse. I did not invent anything in this area but I'm just trying to apply all these rules which are described by software engineering research topics, but are unfortunately way too much ignored by most developers. I have -- and will always prefer to -- spent the same time writing generic and compact code rather than writing several times the same code with small variations. In the same way, I would rather rewrite a code several times for it to be finally generic than modify it each time I need a variant. It's the insurance of settling problems on the long run, despite leaving beginners on the side. The source code must come out crystal-clear for any qualified developer.

- More generally and to insure the generic behavior of implemented solutions, core developers and I take time to query others' thoughts and advices. We always try to give the maximum latitude to change proposals, even significant ones.