PROPOSAL: Unicore
==========

Roles

Project Leads: Simon Kuenzer <simon.kuenzer@neclab.eu> (main lead)
Felipe Huici <felipe.huici@neclab.eu> (co-lead)
Florian Schmidt <florian.schmidt@neclab.eu> (co-lead)
Project Mentor: Lars Kurth <lars.kurth@citrix.com>
Project Sponsor: -To be found-

Background

In recent years, several papers and projects dedicated to unikernels have shown the immense potential for performance gains that these have. By leveraging specialization and the use of minimalistic OSes, unikernels are able to yield impressive numbers, including fast instantiation times (tens of milliseconds or less), tiny memory footprints (a few MBs or even KBs), high network throughput (10-40 Gb/s), and high consolidation (e.g., being able to run thousands of instances on a single commodity server), not to mention a reduced attack surface and the potential for easier certification. Unikernel projects worthy of mention include MirageOS, ClickOS, Erlang on Xen, OSv, HALVM, and Minicache, among others.

The fundamental drawback of unikernels is that they require that applications be manually ported to the underlying minimalistic OS (e.g. having to port nginx, snort, mysql or memcached to MiniOS or OSv); this requires both expert work and often considerable amount of time. In essence, we need to pick between either high performance with unikernels, or no porting effort but decreased performance and decreased efficiency with standard OS/VM images.
The goal of this proposal is to change this status quo by providing a highly configurable unikernel code base; we call this base Unicore.

This project also aims to concentrate the various efforts currently going on in the Xen community regarding minimalistic

OSes (essentially different variants of MiniOS). We think that splitting the community across these variants is counter-productive and hope that Unicore will provide a common place for all or most improvements and customizations of minimalistic OSes. The long term goal is to replace something like MiniOS with a tool that can automatically build such a minimalistic OS.

Unicore - The "Unikernel Core"

The high level goal of Unicore is to be able to build unikernels targeted at specific applications without requiring the time-consuming, expert work that building such a unikernel requires today. An additional goal (or hope) of Unicore is that all developers interested in unikernel development would contribute by supplying libraries rather than working on independent projects with different code bases as it is done now. The main idea behind Unicore is depicted in Figure 1 and consists of two basic components:

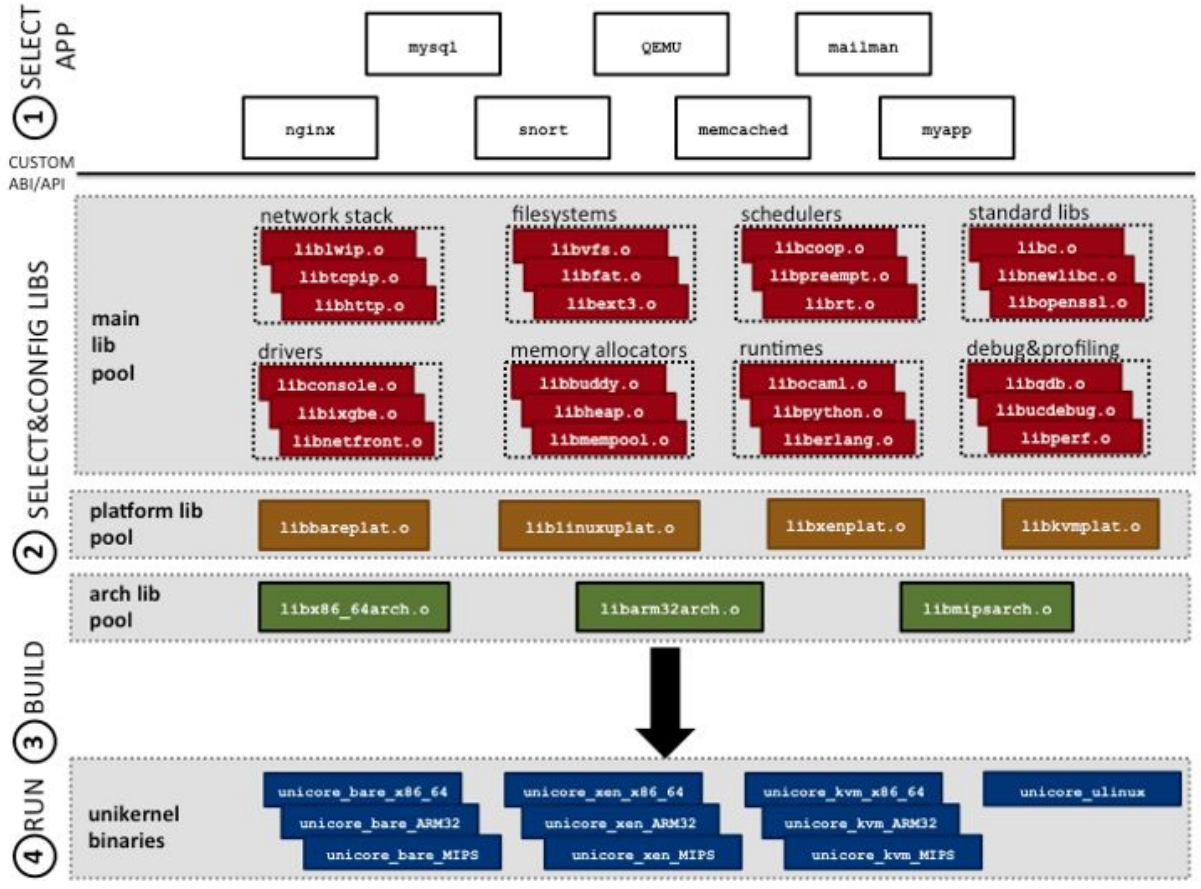

Figure 1. Unicore architecture.

Library pools would contain libraries that the user of Unicore can select from to create the unikernel. From the bottom up, library pools are organized into (1) the architecture library tool, containing libraries specific to a computer architecture (e.g., x86_64, ARM32 or MIPS); (2) the platform tool, where target platforms can be Xen, KVM, bare metal (i.e. no virtualization) and user-space Linux; and (3) the main library pool, containing a rich set of functionality to build the unikernel from. This last library includes drivers (both virtual such as netback/netfront and physical such as ixgbe), filesystems, memory allocators, schedulers, network stacks, standard libs (e.g. libc, openssl, etc.), runtimes (e.g. a Python interpreter and debugging and profiling tools. These pools of libraries constitute a code base for creating unikernels. As shown, a library can be relatively large (e.g libc) or quite small (a scheduler), which should allow for a fair amount of customization for the unikernel.

The Unicore build tool is in charge of compiling the application and the selected libraries together to create a binary for a specific platform and architecture (e.g., Xen on x86_64). The tool is currently inspired by Linux's kconfig system and consists of a set of Makefiles. It allows users to select libraries, to configure them, and to warn them when library dependencies are not met. In addition, the tool can also simultaneously generate binaries for multiple platforms.

As an example, imagine a user wanting to generate a network driver domain unikernel. In this case, we would assume the "application" to be the netback driver. To select this application, the user would first run "make menuconfig" from within the netback application folder. The Makefile there would set a variable to indicate what the application is, and would include the main Unicore Makefiles so that the unikernel can be built (Step 1 in the figure). Using the menu-based system, the user chooses the relevant libraries; for a Xen driver domain this would include a physical network driver, the netback driver, the libxenplat library and a library from the architecture library pool such as libx86_64arch (Step 2 in the figure). With this in place, the user saves the configuration and types "make" to build the unikernel (Step 3) and xl create to run it (Step 4).

A note on the ABI/API: because Unicore allows for customization of the unikernels, the ABI (or API since there is no kernel) would be custom, that is, defined by the libraries the user selected.

Having said that, it would be perfectly possible, for instance, to build POSIX-compliant unikernels with it.

Relevance to Xen and its Community

Unikernels are important to a number of areas relevant to the Xen community, including IoT, automotive, stub domains and driver domain disaggregation. Unicore could help boost the progress in all of these areas by quickly providing the necessary tools to create unikernels for them. For instance, for a driver domain, the user would include the “library” containing the relevant hardware driver and corresponding back-end driver, and in principle Unicore would take care of the rest.

In addition, Unicore could eventually replace Mini-OS, providing a cleaner, more stable and flexible base from which to build unikernels for projects (the modularization of Mini-OS is in fact already taking place).

Current Status

Unicore is at an early stage. For now it includes some base libraries with code extracted from Mini-OS as well as a build tool inspired by Linux's KConfig system. Unicore is currently able to build "hello world" unikernels for Xen and Linux user space on x86_64 and ARMv7.

Incubation

The reason behind making Unicore a Xen sub-project project is to (1) bring the existence of Unicore to the attention of the Xen community and to outside world; (2) to attempt to harness interest and potentially development cycles from people and companies interested in unikernels; and (3) to concentrate maintenance resources from people interested in unikernels within the community.

License

The main license of the run-time components of Unicore will be a
3-clause BSD license, unless there is a good reason not to use it
(e.g. we may import 2-clause BSD licensed code from Mini-OS, which
we would *not* anticipate to change). The Makefile system would be
licensed under GPL v2 or later as we want to be able to use
KConfig functionality from Buildroot/Linux.

Required Infrastructure

The official repositories should be created on
[http://xenbits.xenproject.org/] under `unicore.git`. There should
be a main repository for the core unicore implementation and
additional repositories for some more advanced extension libraries
(e.g., lwIP, newlib).

Main repository

`unicore.git`

Repositories for extension libraries

Repositories for additional libraries that are supported by the
Unicore
project should exist under a separate directory:

`unicore-libs/`

For example:

`unicore-libs/lwip.git`
`unicore-libs/newlib.git`

Mailing list

In the beginning we would use the MiniOS mailing list
(minios-devel@lists.xenproject.org). When we get traction with
Unicore we could consider splitting that traffic onto a unicore
mailing list.