Unikraft – The "Unikernel Core"

The high level goal of Unikraft 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. The main idea behind Unikraft is depicted in Figure 1 and consists of two basic components:

Library pools would contain libraries that the user of Unikraft 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 Unikraft 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, a user wanting to create a network driver domain would start the tool ("make menuconfig") and (steps 1 in Figure 1). 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). In the case of an application-based unikernel (e.g., lighthttpd) the process is the same, except the user runs "make menuconfig" from the application directory; the Makefile there would set a variable to indicate what the application is, and would include the main Unikraft Makefiles so that the unikernel can be built.

An additional goal (or hope) of Unikraft 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 today.

A note on the ABI/API: because Unikraft 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.

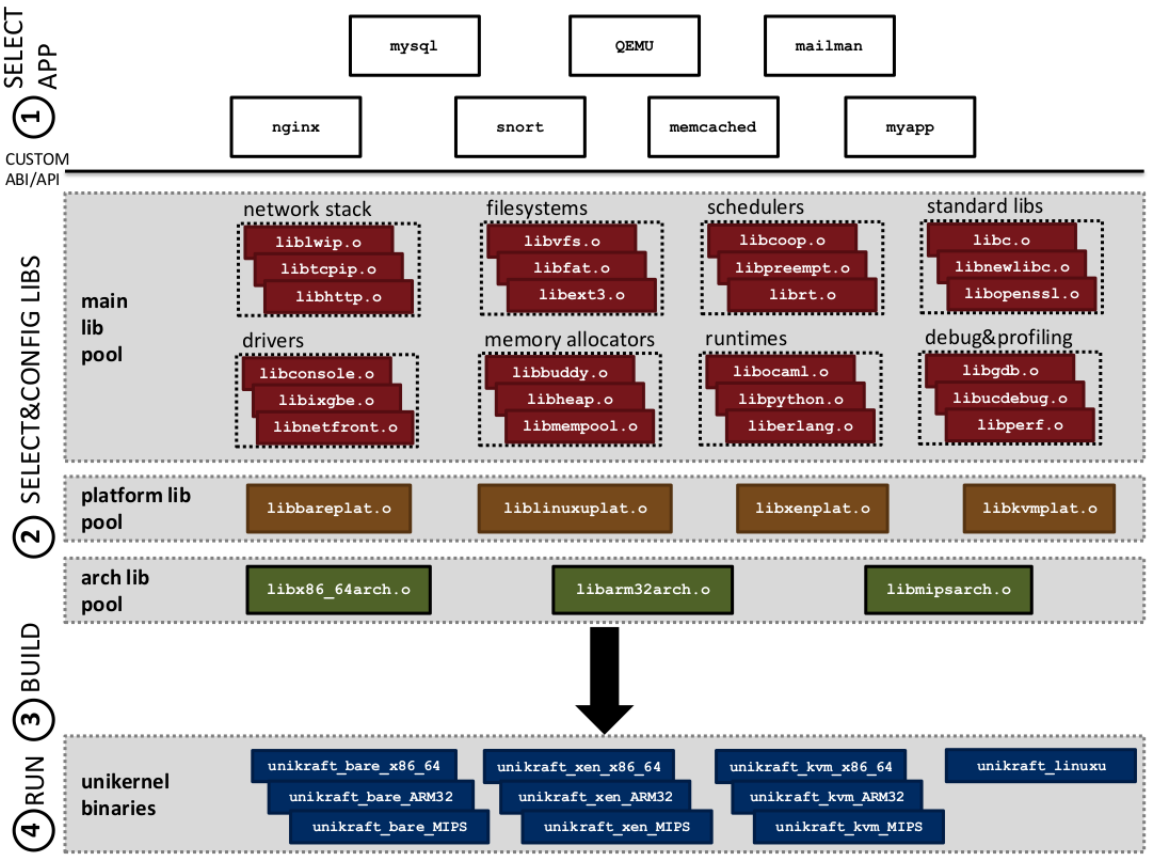

Figure 1. Unikraft architecture.