Introduction to UX/RT

Andrew Warkentin
QNX/Thoth-like architecture (with a few ideas from Plan 9)

- Uncommon architecture, despite seemingly being optimal in several ways
- General-purpose dynamic OS, with good support for both desktop/server and embedded use cases
- Natively Unix-like, with no built-in concept of multiple personalities
- Limited vertical modularity; each instance of most subsystems is a single process containing all layers of that subsystem
- Filesystem as the primary or only service lookup mechanism
- Lightweight IPC transport layer; marshalling limited or absent
General-purpose OS with good support for both desktop/server and high-end embedded use cases

- Seems to be somewhat neglected by current research OS development (Genode being one notable exception)
- More active in the hobby OS world
- However, broad appeal and direct compatibility with existing applications don’t seem to be particularly high priorities for most research and hobby general-purpose OSes
Natively Unix-like, with no concept of multiple personalities

- Win32 is the only significant non-Unix OS API that hasn't been relegated to legacy status
  - Wine already implements Win32 on top of Unix, so no need to implement a separate Win32 personality
- Multiple personality support adds complexity and usually also overhead
- A lot can be improved by simply cleaning up and enhancing Unix, as opposed to making an entirely new environment
Limited/optional vertical modularity; each instance of most subsystems is a single process containing all layers of that subsystem

- Separation of subsystems into one process per layer/component adds at least two context switches per layer handling a request (assuming all layers use RPC-like IPC)
- Protection domains often correspond more to entire subsystem instances rather than components/layers
  - e.g. all the layers of a storage stack often just deal with the same data at different levels
- Where necessary, vertical disaggregation can still be supported by plugin architectures within servers and tap-type drivers that allow an upper server to call a lower server
  - e.g. where partition-level encryption is in use, there could be separate disk servers for the encrypted and decrypted layers
A VFS that dispatches requests to multiple servers is already present
  - Adding any kind of parallel lookup mechanism just adds unnecessary complexity with no real benefit

All security is reduced to file security

Similarly, containerization is mostly just a matter of constructing filesystem namespaces
Many types of APIs are simple enough to not need complex marshalling, especially when implemented as special filesystems.

- This includes services that need high throughput to move bulk data like disk filesystems and network stacks.
- Similarly, marshalling shouldn't be included in the TCB of services that don't need it.

Marshalling can still be implemented in an optional library for services that need it.

- A message-boundary-preserving special file type can be supported to make this a lot easier.
Various features

- seL4 microkernel (may switch to a fork at some point due to differing priorities)
- Extensive use of safer languages like Rust and V for new code, although a lot of C will still be present
  - Third-party code will be used wherever it makes sense; this will account for most of the C code
- Everything truly is a file; even process memory and state are accessed through special files in addition to all
  IPC being file-based
  - Alternative file I/O APIs providing direct access to message registers and a shared buffer provided to
    avoid unnecessary copying
- Security model based on per-process permission lists and capability transfers
  - Role-based access control implemented on top of this
  - SUID/SGID binaries and the requirement to be root for privileged APIs both absent
Various features

• Disaggregated process state model; state that is associated directly with processes in other Unix-like OSes is associated with context objects
  - Some of these correspond to underlying seL4 context objects
  - This includes process creation; processes are completely empty when created, and fork() and spawn() are wrapper functions
• LKL for device drivers, disk filesystems, and network stack
• Linux compatibility environment based on a library/loader and several special filesystems
• Modular init system with delegated restarters similar in architecture to SMF (but without the databases and XML)
  - Uses cgroups sort of similar to those in Linux
  - "Event superserver" daemon to replace the many ad-hoc event daemons found in most other modern OSes (e.g. NetworkManager, udev, udisks) with command-invoking units
  - Login/session management integrated with the init system and cgroups
Various features

- dpkg/apt-based package manager with extensions for containerization and functional package management
- Policy-free compositing window server, with support for an external X11-style reparenting window manager
  - Compatibility library for Wayland clients provided
- Lightweight desktop environment that is Mac/NeXT-like by default but is highly configurable
- Toolchain for easily building custom distributions (either embedded or desktop/server)
Current status

- Utility to create boot filesystem images
- In-memory bootloader
- Patch to seL4 to pass through Multiboot2 header to root server
- Low- and mid-level kernel bindings for Rust (based on those from Ferros and Robigalia)
- Allocator stack including a heap allocator and a complete set of kernel object allocators
  - Kernel object allocators derived from Robigalia and heavily enhanced
  - Heap allocator is a heavily extended version of the one used in Redox
  - Only feature required for user processes currently missing is deinitialization of VSpaces
- Root server that checks the Multiboot info and performs some allocator tests
https://gitlab.com/uxrt - GitLab group

https://gitlab.com/uxrt/uxrt-toplevel/-/blob/master/architecture_notes - Further notes on the architecture (currently a bit disorganized)