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
- Thoth was the earliest example back in the late 70s
- QNX started in the early 80s along with a few others but most of those abandoned by the early 90s
- VSTa is the most recent complete example of such an OS other than QNX itself

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

- General-purpose dynamic OS, with good support for both desktop/server and embedded use cases
- Lightweight real-time microkernel with fast IPC
- Lightweight IPC transport layer without marshalling
- Natively Unix-like, with no built-in concept of multiple personalities
- Limited vertical modularity; each instance of most subsystems is a single process
- Filesystem as the only service lookup mechanism

General-purpose OS with good support for both desktop/server and high-end embedded use cases

- Somewhat neglected by current research OS development (Genode being one notable exception)
- More active in the hobby OS world
- Direct compatibility with a wide range of applications and hardware is a high priority for UX/RT, unlike most other research and hobby general-purpose OSes

Lightweight real-time microkernel with fast IPC

- Performance can be competitive with monolithic kernels
- Smaller TCB
- Real-time support easier than with a more heavyweight kernel

Lightweight IPC transport layer; marshalling limited or absent

- Many types of APIs are simple enough to not need complex marshalling
  - Especially when implemented as special filesystems
  - Includes services that need high throughput to move bulk data like disk filesystems and network stacks
  - Marshalling shouldn't be included in the TCB of services with simple enough APIs
- Marshalling can still be implemented in an optional library
  - Can be made easier with a message-boundary-preserving special file type

Natively Unix-like, with no concept of multiple personalities

- Multiple personality support adds complexity and often also overhead
- Win32 is the only non-Unix OS API still relevant
  - Wine already implements Win32 on top of Unix
- A lot can be improved by simply cleaning up and enhancing Unix
- Linux binary compatibility is better integrated
- Easier to use LKL for access to Linux drivers

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 can add two context switches per layer
- 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

Filesystem as the only service lookup mechanism

- A VFS that dispatches requests to multiple servers is already present
  - Adding any kind of parallel lookup mechanism just adds unnecessary complexity
- Security is mostly reduced to just file security
- Containerization is mostly just reduced to constructing filesystem namespaces

Various UX/RT 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 C code
- Everything truly is a file; even process memory and state are accessed through special files
  - Alternative file I/O APIs will provide direct access to message registers and a shared buffer
  - These will interoperate with traditional file APIs
- Security model based on per-process permission lists and capability transfers
  - Role-based access control implemented on top of this
  - Privilege escalation managed through a server that associates manifests with binaries
  - SUID/SGID binaries and the requirement to be root for privileged APIs both absent

Various UX/RT features

- Disaggregated process state model; all process state associated with context objects
  - Some of these correspond to underlying seL4 context objects
  - Threads within a process may have different context objects
  - Includes process creation; processes 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, a bit like SMF
  - Uses cgroups sort of similar to those in Linux
  - "Event superserver" to replace the many ad-hoc event daemons found in most other modern OSes
  - Login/session management integrated with the init system and cgroups

Various UX/RT 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 shims for Wayland clients provided
- Lightweight desktop environment that is Mac/NeXT-like by default but is highly configurable
- Support for multiple distributions
  - More analogous to Illumos than to Linux
  - Reference distribution with a distinct identity
  - Toolchain for easily building custom distributions (both embedded and 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

Roadmap

- Locking support
- IPC transport layer
- VFS
- Root server internal filesystem drivers (procfs, memfs, null, etc.)
- VSpace deallocation
- User process support

Links

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)