

LUO support in systemd

Luca Boccassi, Microsoft

Agenda

- Refresher about systemd's File Descriptor Store feature
- Simple use case: preserve individual FDs
- Complex use case: manually handle LUO Sessions
- Nesting in containers and/or user services
- Internal implementation details
- Discussion on what's next

File Descriptor Store in systemd

- systemd can hold resources for services in the [FD Store](#)
- Added for sockets, but can be used for any FD resource, including [memfds](#)...
- ...which means arbitrary memory can be safely stored and reused across a service restart
- Set [FileDescriptorStoreMax=](#) in the unit
- Use [sd_pid_notify_with_fds](#) to hand over the FD(s) before stopping
- Use [sd_listen_fds](#) to get the FD(s) back after starting
- A service that requires large amount of hard-to-derive data in memory can use a memfd to shortcut restarts
- Caveat: cgroup memory accounting for the memfd is “lost”
 - Original process is the owner, and once it's gone, it's gone

LUO + FD Store = happiness

- Set [FileDescriptorStorePreserve=yes](#)
- Use FDNAME= when sending and receiving FD(s)
- ...and that's pretty much it already!
 - No semantic difference with service restart - intentionally!
- Once the service starts on boot after a kexec, it will get its memfds handed back
- Available since v261, with a supported kernel

LUO Sessions available too

- First-class support for managing local LUO Sessions is available too
- A session allows grouping resources, and also managing FDs that do not support SCM_RIGHTS
- Privileged processes can call LUO APIs at /dev/liveupdate directly
- Since v262, units can get sessions assigned via [LUOSession=](#)
 - Note: for system units only
- In either case, send and retrieve the LUO Session FD to/from the FD Store using the usual APIs

Nesting in user sessions and/or containers

- Nesting works out of the box too!
- FD Stores since v261 are automatically propagated up and down the managing stack, with systemd system and user instances, and nspawn, all supporting forwarding up and down the chain, and across kexec
- System service -> nspawn -> container service
- System service -> user session -> user service
- System service -> user session -> nspawn -> container service
- ...

Implementation details (1/2)

- FDs are stored in the manager at runtime, one store per unit
- On shutdown, the FDs are transferred from the system manager to systemd-shutdown, a separate binary
- sd-shutdown performs the final kill spree of any remaining processes first
- Then it creates a private LUO Session, storing all the stores plus metadata
 - E.g.: shutdown timestamps, kexec counter are also stored and passed to the next boot
 - Kernel rule: process must be alive and running, holding LUO Sessions, on kexec syscall
- The session is named “systemd” (duh!) and it’s special-cased: no unit is allowed to place a session with this name in the FD Store
 - This guarantees the name is available when sd-shutdown creates it
- Finally reboot syscall is invoked

Implementation details (2/2)

- On boot, in the initrd the private LUO Session is opened and data extracted
- All FD Stores are restored for all units, even those that only exist in the full system
- Internal metadata is parsed and stored
- The private LUO Session is closed
- Then units are started and normal boot continues

What's next?

- Is what is currently provided sufficient for simple use cases?
- What about VMMs?
- Is there a concrete use case for unprivileged + nested container/user session + custom LUO session(s)?
 - Only case currently not explicitly covered, as LUOSession= works for system units only

Thanks

Questions?