DYNAMIC DEVICE MANAGEMENT FOR LXC

Michael J Coss
Oct 15, 2014

OUR ORIGINAL GOAL

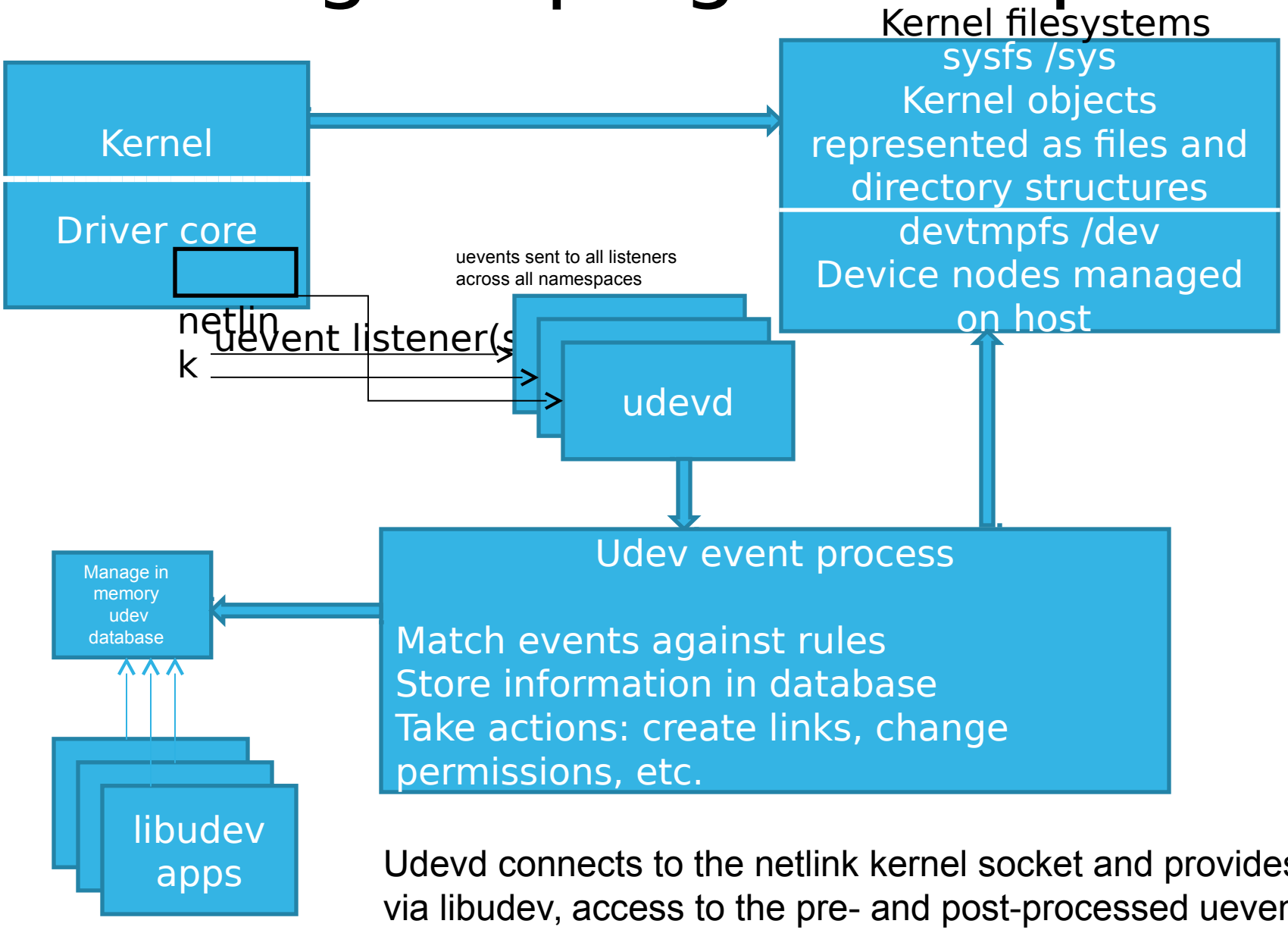
To provide a virtual desktop environment that

- Has performance as close as possible to the non-virtualized environment
- Lets the user dynamically add and remove I/O devices to/from the virtual desktop
  - Keyboard and mouse
  - Display
  - Audio (speaker, microphone)
- Supports 3d hardware-accelerated graphics

DYNAMIC DEVICE MANAGEMENT

- Device management is currently done via a combination of devtmpfs, sysfs and udevd
  - Not namespace aware
  - Not container aware
- Most LXC documentation said not to run udevd in containers at all
- Many users simply mount devtmpfs in the container
  - Works but exposes all devices to the container

Existing hotplug event path



Udevd connects to the netlink kernel socket and provides, via libudev, access to the pre- and post-processed uevent stream.

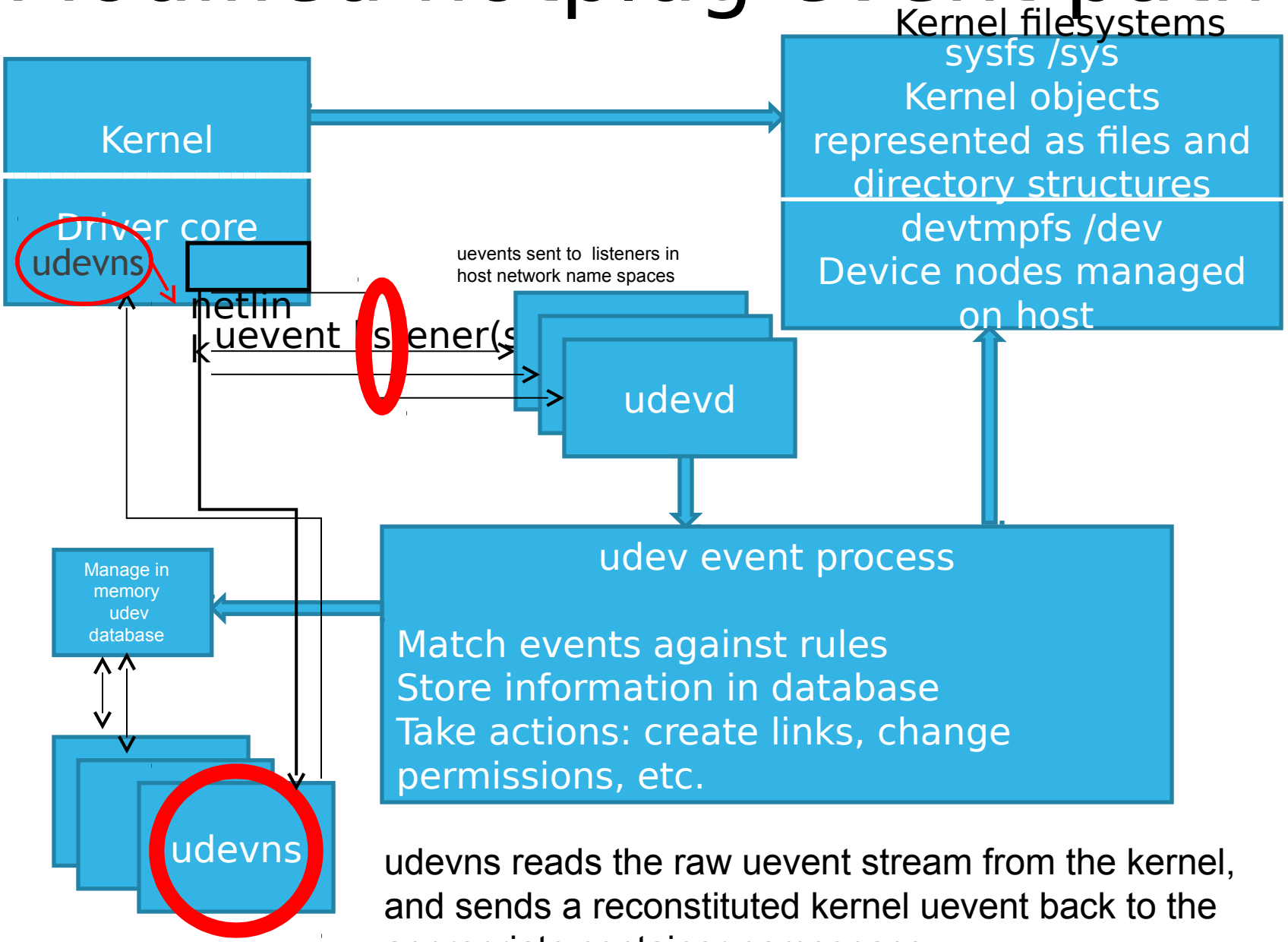
DYNAMIC DEVICE MANAGER ISSUES

- Assuming you can even get udev to run in a container, the process listens on a kernel socket and *all* events are passed to it
- Mounting devtmpfs inside a container grants the container access to *all* devices
  - Can use device controls in lxc.conf to restrict access
    - ideally would like to have only a subset of devices exposed to the container
- Where and how do you apply policy?
  - Who gets what kernel events?
  - Who owns the device?

OUR CHANGES

- The kernel currently broadcasts uevents to any process listening on the kernel socket.
  - Pass uevents only to processes in the server's network namespace.
  - Containers run in a separate network namespace to facilitate isolation
  - A new kernel function is needed to take a uevent targeted for a specific container and route to any listeners in that container's namespace.
- User space daemon (udevns)
  - Policy-control daemon listens for uevents of interest
  - Checks an in-memory database for device information
  - Manages device nodes in container's /dev directory and passes uevent to kernel socket in container's namespace

Modified hotplug event path



udevns reads the raw uevent stream from the kernel, and sends a reconstituted kernel uevent back to the appropriate container namespace

SO HOW DOES IT WORK (KERNEL PERSPECTIVE)

User plugs in a USB keyboard…

- A directed graph of directories, and files as specified by the device driver is generated in /sys for the various kobjects

  Total of 186 entries generated for a keyboard hotplugged on my system

- As devices are added, uevent messages are generated and sent to processes listening on the netlink socket, in the host network namespace

  Total of 6 events generated for the keyboard insertion

SO HOW DOES IT WORK (USERSPACE PERSPECTIVE)

- udevd listens on the netlink socket in the host network namespace
  - Processed as normal, actions taken as specified by rules
- udevns listens to the same uevent message stream that udevd sees
- udevns determines which container is interested in the given event

SO HOW DOES IT WORK (USERSPACE PERSPECTIVE) CONT.

- udevns determines what devices are needed
  - Creates or removes the nodes in the container's local /dev directory
- udevns constructs a uevent message
  - Sent via a simple pseudo device driver and forwarded to the container's udevd via the netlink socket
  - Any other container processes listening on the netlink socket will receive a copy of the uevent

CONCLUSIONS

- udevns is just one example of policy mechanism to manage uevents
- Two key kernel infrastructure mechanisms were needed
  - Stop the broadcast of uevents to all namespaces
  - Facilitate selectively sending uevents to a specific container
- While this addresses devtmpfs by eliminating its use in the container, sysfs is still an issue