LAS16-TR06
RemotePROC & rpmsg development

Bjorn Andersson
remoteproc
A framework for controlling the lifecycle of secondary processors in an asymmetric multiprocessor system
remoteproc drivers

Implements the hardware interface for starting and stopping a remoteproc

- `rproc_alloc(..., ops, firmware, ...)`
- `rproc_add()`
- `rproc_del()`
- `rproc_put()`

ops struct contains:

- `start()`
- `stop()`

Separate ops struct for fw handling operations
Resource table

- Firmware contains resource table section, listing:
  - Memory carveouts
  - IOMMU mappings
  - Trace buffers
  - Virtio devices

- Loaded and processed by the remoteproc core before loading code and data segments and calling `start()`

- Limited implementation support for controlling physical placement of memory carveouts
  - Resource table entries contain (optional) physical address
  - Dynamic allocation done in the context of the remoteproc driver
Auto-booting a remoteproc

- Traditionally a remoteproc was booted:
  - explicitly by a client, or
  - by the virtio devices registered during early resource table parsing

- First virtio device registered boots the core
  - Resources for additional virtio devices might not yet be registered
  - No ability to control a remoteproc explicitly if it has virtio devices in its resource table

- Change introduced to flag a remoteproc to auto-boot
  - Behavior no longer depends on listed and loaded virtio devices
  - Simplifies resource handling
Amend resource table entries

- Remoteproc driver might need to provide additional data for resources
  - Constraints on carveout allocations
  - Communicate addresses of programmatically registered resources

Suggestion:

- Provide API for remoteproc drivers to register resources
- Merge resources with entries from the resource table
  - FW_RSC_ADDR_ANY can be overridden by more specific data
  - Deny size growth
  - Update resource table, when applicable
Firmware without resource table

- Support for firmware files without resource table
  - Hard coded memory regions in driver
  - Firmware with non-standard data structures
- Still want to reuse core support for carveouts etc

Traditionally remoteproc drivers could override resource table load operation and programmatically inject generated table
  - Resource table is a on-disk structure, making this inconvenient

Suggestion:
- Expose API for remoteproc drivers to register resources with the core
Resource-list and API

- Group different types of memory regions in common list
  - Support carveout, ioremap and other types
- Extend previously suggested APIs to cover additional resources
  - Merge and validate with existing resources
- Remoteproc drivers and resource table parser using the same API
  - Decouples the remoteproc core from version 1 of the resource table format
- Defer allocations until boot time
  - Allows the merging of static and dynamically registered resources before allocation

- Use mechanism for allocating vring memory
  - Matching on device, then physical address and falling back on carveout
  - Allows driver to register ioremap’ed region to be used to vrings
Virtio Device alternatives

- Qualcomm SMD and GLINK follows remoteproc state
  - Like RPMSG and other virtio based devices
- Drivers related to firmware functionality follow remoteproc state
  - State either driven by function driver or function driver following state

- Generalize virtio-device list to allow arbitrary “subdevices”
  - “probe”/“remove” these devices following the RPROC_RUNNING state
  - Provide API to allow remoteproc drivers to register these devices
  - Common types in the core?
Remoteproc core dump files

Beneficial to provide core dumps for post mortem analysis of crashing remotepros

Suggestion:

- Provide API for registering segments to be included in the dump file
  - Can come from remoteproc driver or resource parser
- During crash recovery the core generates ELF structure based on segment list
- Make ELF content accessible through debugfs
- Recover remoteproc after user space finishes acquiring a copy, or timeout

Some data dumped from device memory or registers, would need backup before core is powered down
Custom firmware parser

In current implementation the core reads, parses and operates on the resource table

Inflexible for remoteproc drivers with firmware without resource table
  ● Driver can inject generated resource table during load
  ● Transforming other information into resource table is inconvenient

Suggestion:
  ● Extend fw_ops with a “parse” function
    ○ Allows drivers to supply custom resource parser
    ○ Doesn’t need to generate the on-disk format
  ● Refactor current resource parser, register as default parser
Dealing with huge firmware files

Firmware files often large and residing on secondary partitions

● Firmware files available very late
● Need mechanism to “signal” the availability
● Currently two working mechanisms
  ○ Using kernel modules for remoteproc drivers
  ○ CONFIG_FW_LOADER_USER_HELPER_FALLBACK

Firmware often use case and device specific

● How to distribute firmware files?
rpmsg
A framework and wire format for exchanging messages over a point-to-point channel to a “remote processor”

(rpmsg is not remoteproc)
rpmsg drivers

RPMSG drivers implement communication endpoint

- Registering with kernel interfaces and expose remote functionality
- rpmsg devices instantiated and destroyed based on channel availability
- `rpmsg_send(struct rpmsg_endpoint *)`
- Driver has per-endpoint callback for incoming data
- `rpmsg_create_ept()` and `rpmsg_destroy_ept()` for multi-endpoint drivers
Alternative wire-formats

- Traditionally only VIRTIO_ID_RPMSG with its specific wireformat
- Qualcomm SMD and GLINK channel management and API similar to RPMSG
  - Mainline version of SMD designed to mimic RPMSG
  - Buffer management and packet format differs - not virtio based
- Split rpmsg implementation in virtio-rpmsg and housekeeping + API part
- RPMSG devices presented with one API regardless of backend

- Some generic improvements
  - Send functions now operate on “endpoints”
  - Cleaned up the public header file
User-space interface

● Remote core exposes functionality unrelated to Linux kernel
  ○ E.g debug interfaces and custom application logic
● Expose `rpmsg_endpoint` access to user space
  ○ Prior art based on exposing specific channels over sockets or character devices
● Upstream solution must handle multiple channels from multiple remotes
  ○ Without depending on hard coded channel names in the kernel
  ○ Likely not acceptable to put list in DeviceTree

Suggestion:

● Mechanism for managing endpoints to be exported
  ○ Manager can’t rely on standard rpmsg device handling
● `rpmsg_create_ept()` and `rpmsg_destroy_ept()` tied to open/close of char devices
PCAP sink

RPMSG driver development benefits from ability to dissect communication
- Wireshark support for custom dissectors
- Hack used for wcn36xx development
- Standard solution would aid development of other drivers

Suggestion:
- Develop common pcap sink implementation for all rpmsg transactions
  - Single sink with BPF support?
  - One sink per endpoint?
Thank You

#LAS16
For further information: www.linaro.org
LAS16 keynotes and videos on: connect.linaro.org