

Bill Mills

2017/09/28

TI Information - Selective Disclosure



User space API for rpmsg

- Basic message passing
 - Complete Rpmsg char driver
- Name server access
 - Sysfs entries for current name server contents
 - Udev rules fire on Name add/delete

Bulk Data Buffer Exchange

- ION memory allocation & export as DMABUF
- Rproc or rpmsg DMABUF import
 - Model: Done once at application start or per buffer
- Rpmsg userspace I/F extension for address translation and buffer cache maintenance
 - Rpmsg messages have pointers to buffers and I/F gives enough info to know where these are
 - Buffer info includes DMABUF handle, offset and size
 - Cache maintenance is done for each Buffer based on direction (if needed)
 - Address translation is done on buffer addresses
 - Exchange multiple rpmsgs both TX & RX each with one or more buffers in a single kernel call

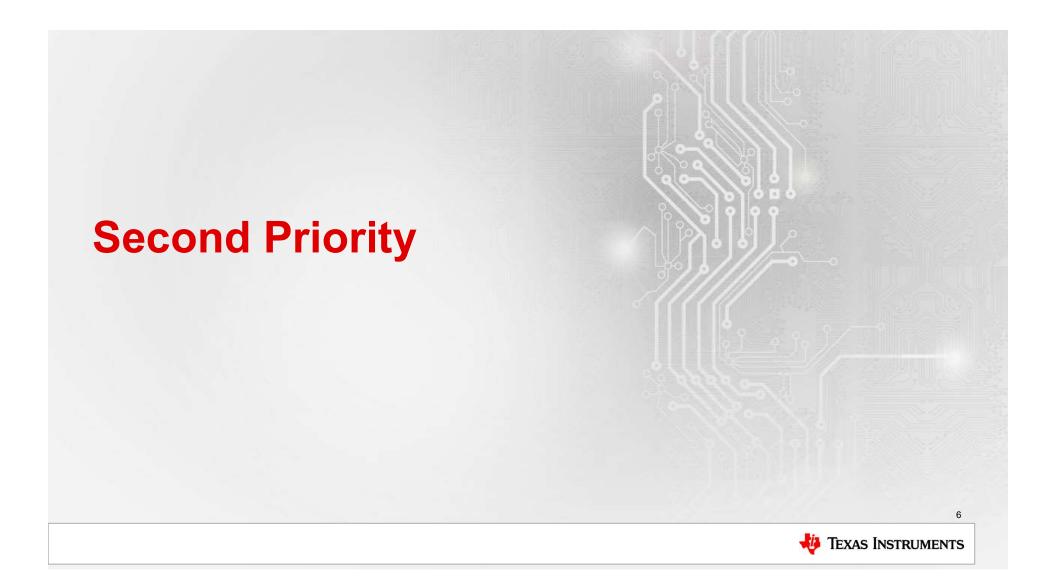
Universal support for Late attach / Detach

- Bootloader or hypervisor loads a "co-processor" first before Linux starts
- Co-processor needs to survive Linux shutdown
- Examples using AM572x as an example (BeagleBoard-X15)
 - M4 comes up first, A15 does not touch the M4 program but does do rpmsg
 - A15 may or may not be allowed to reboot / reload M4
 - A15 can reboot w/o M4 reboot
 - A15 starts M4 but then detaches so that M4 can stay alive if/while A15 reboots
 - A15 Linux to A15 RTOS rpmsg
- Rproc / rpmsg independence is important for many of these cases



RPROC w/ Hypervisor

- How does RProc work in a hypervisor based system
- Model 1: Guest kernel based
 - ARM hypervisor sets up SOC firewalls and/or IOMMUs so that Co-proc can only touch resources owned by Guest A
 - Hypervisor lets Guest A kernel run its normal rproc driver
 - Hypervisor may still need to do low level things like control reset etc
- Model 2: Guest has rpmsg and hyper-proxy rproc
 - Hypervisor (or even secure space) needs to control the rproc. Examples:
 - Need to authenticate firmware before allowed to run
 - Need to first load a kernel/monitor before firmware image
 - Guest kernel has a rproc driver that just delegates most things to the hypervisor



Standardize Rproc lifetime management

- Can we standardize any of the following:
 - suspend/resume
 - HW resource attach / detach
 - Create/destroy new context on co-proc
 - Coredump
 - Logbuffer access
 - Console
 - Gdb proxy
 - CPU utilization, rproc owned memory utilization

RPROC user mode helper

- Certain use cases are out-of-scope for the kernel rproc subsystem
 - ELF fixups
 - Executables in Shared memory used by independent co-processors
 - Shared libraries, Overlays
 - Loading of memories not addressable by master
 - Multiple contexts on co-processor
 - Loading a monitor/supervisor/executive etc before the firmware
- Want a system that can be triggered by the firmware file contents that will cause rproc driver to:
 - expose rproc memories as mmap'able
 - send an event to userspace helper (udev etc)
 - wait for user space to load firmware and then send info to rproc
 - info to include: resource table, start address? others?

Dynamic Virtio

- Rpmsg channel can be used to create and destroy new VirtlO devices
- Can be used like VirtioPCI or MMR driver but is message based so does not require magic memory
 - Magic memory causes side effects when read or written.
 - This is doable in HW MMRs and in a trap & emulate based emulation system.
 - This is impossible to implement when two cores are talking directly

OpenAMP Linux userspace ecosystem

- OpenAMP Linux client library
 - Library to be used by OpenAMP based applications
 - Can go direct to driver resources if it has permission
 - Otherwise (for shared resources etc) it talks via DBus to the OpenAMP Deamon
 - Provides easy API for message exchange and memory buffer area allocation
- OpenAMP daemon
 - In cases where rprocs are a shared resource, daemon coordinates, has DBus interface
 - Can be used by OpenAMP clients to allocate access to a whole rproc or to start an
 executive and allocate a new context
- OpenAMP proxy
 - From the command line, run an elf file on a rproc w/ semi-hosting
 - Can detect is elf supports multiple contexts (via INTERP) and can use Daemon to create a sub-context