VFIO based Channel I/O passthrough on IBM z Systems

An introduction to Channel I/O and vfio-ccw

Dong Jia Shi / bjsdjshi@cn.ibm.com

Agenda

- Basic Concepts
- Initiating I/O
- Linux Support for Channel I/O
- Virtualization Support
- Channel I/O Passthrough

- **Basic Concepts**
- Initiating I/O
- Linux Support for Channel I/O
- Virtualization Support
- Channel I/O Passthrough

Basic Concepts

Things we need to know about Channel I/O.

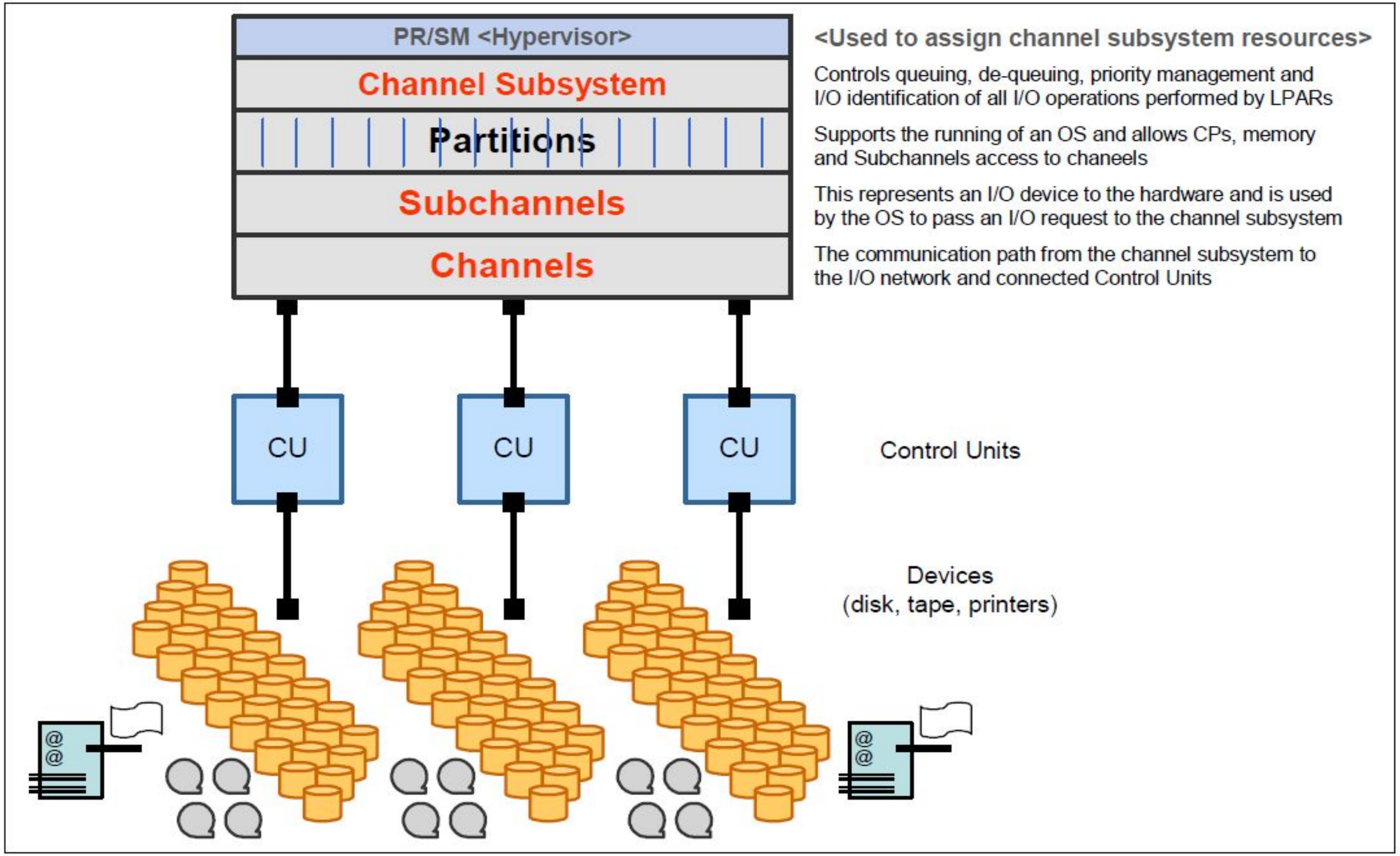
Basic Concepts - CSS & Subchannel

- Channel Subsystem (CSS)
  - Provides I/O mechanism
  - Processors dedicated to I/O
- Channel Subsystem Image
  - Comprised of *subchannels* and *channel paths*
  - Up to 4 images per machine
- Subchannel
  - Logical communication path to and from device
  - Collects status for I/O, connections and device
  - Organized into up to 4 sets of up to 64k subchannels

Basic Concepts - Channel Path & CU

- Channel Path
  - Corresponds to machine ↔ control unit connection
  - Shared between subchannels (up to 8 channel paths per subchannel)
  - Up to 255 channel paths per channel subsystem image
- Control Unit
  - Accepts a set of *channel commands (CCW)*
  - May be integrated with the I/O device
  - Self-descriptive (e.g. SenseID channel command)
  - Responsible for translating between CCW and device-specific actions

Basic Concepts - Channel I/O Architecture



<Used to assign channel subsystem resources>

Controls queuing, de-queuing, priority management and I/O identification of all I/O operations performed by LPARs

Supports the running of an OS and allows CPs, memory and Subchannels access to chaneels

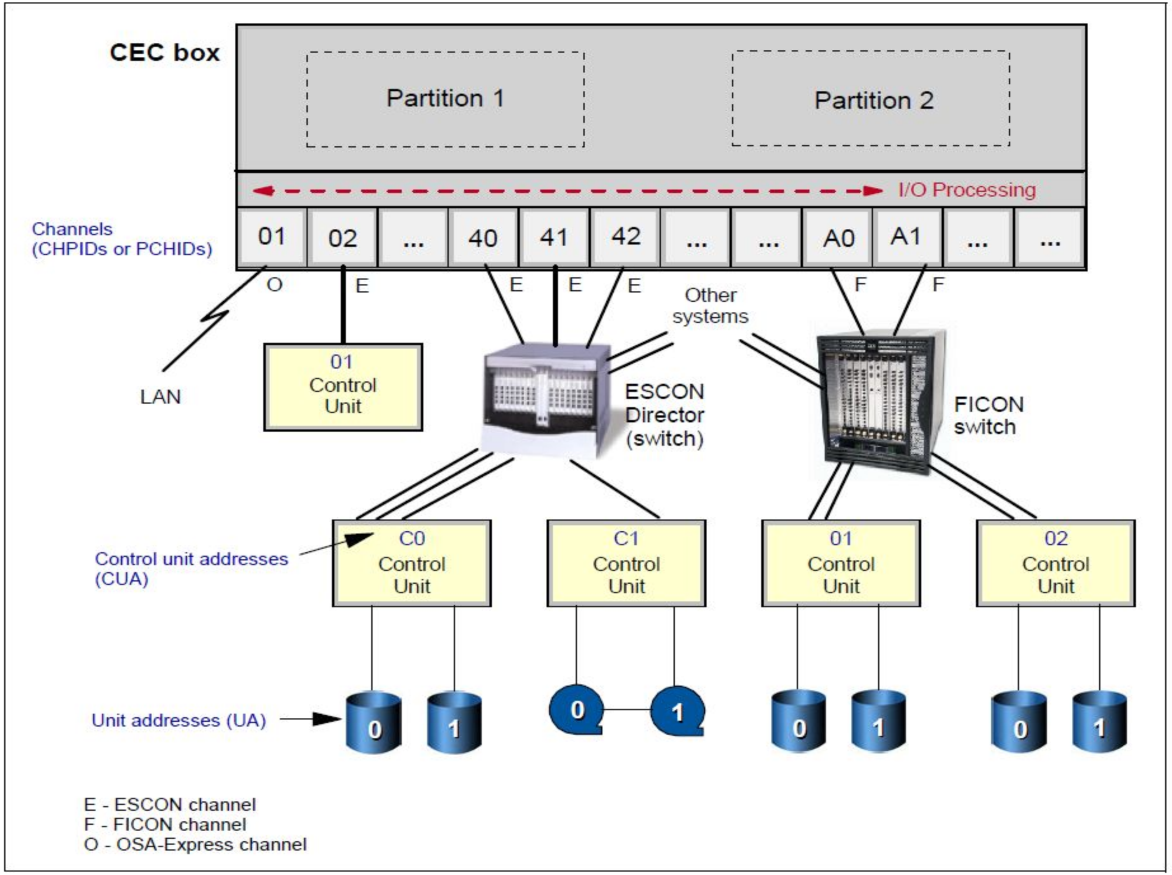
This represents an I/O device to the hardware and is used by the OS to pass an I/O request to the channel subsystem

The communication path from the channel subsystem to the I/O network and connected Control Units

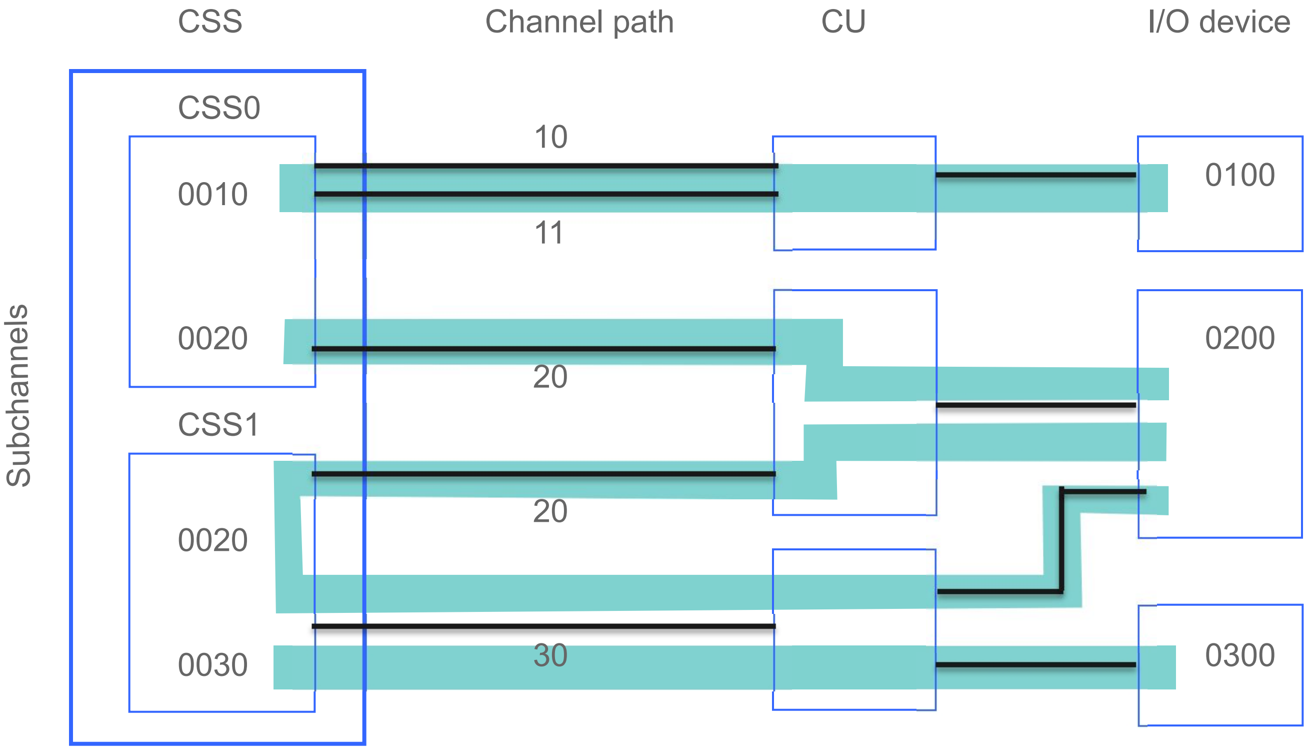
Control Units

Devices
(disk, tape, printers)

Basic Concepts - System Configuration



Basic Concepts - Logic View



- Basic Concepts
- **Initiating I/O**
- Linux Support for Channel I/O
- Virtualization Support
- Channel I/O Passthrough

Initiating I/O

How to issue an I/O instruction and receive its result?

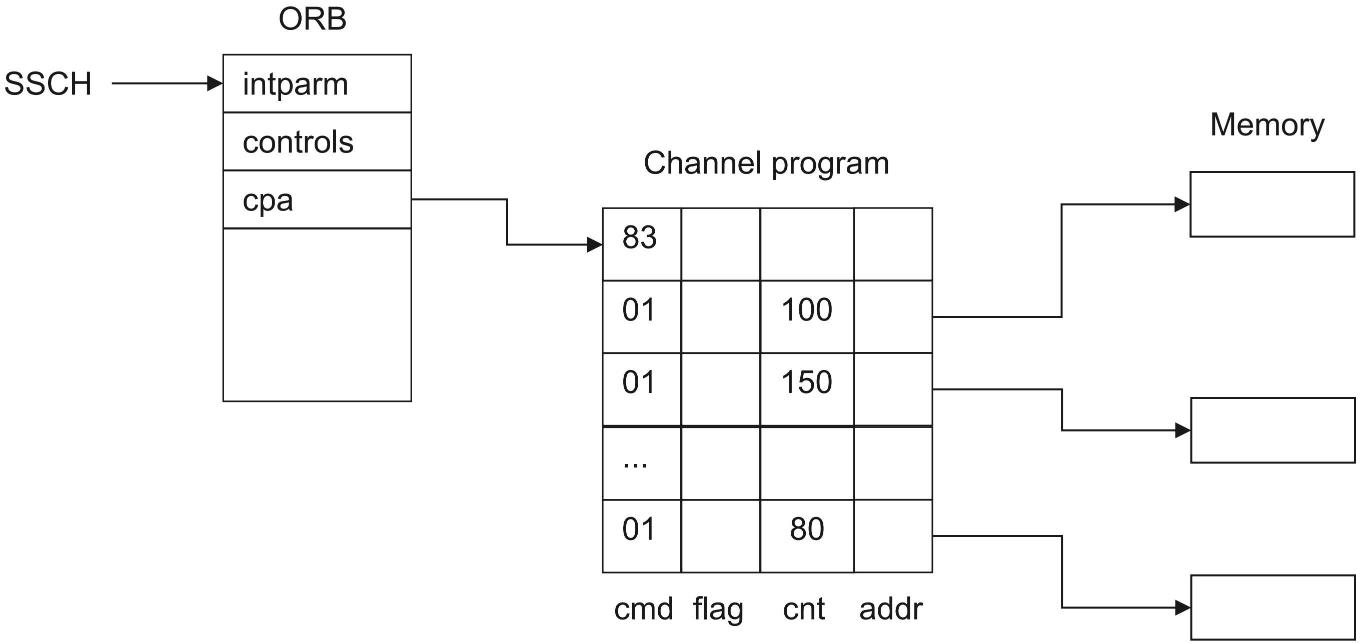
Initiating I/O - I/O Instructions

- START SUBCHANNEL (SSCH)
  - Provides a channel program and parameters (*ORB*) to CSS
  - *Channel program* is performed asynchronously by CSS
  - Upon conclusion, error or caller's request, the subchannel is made status pending and an *I/O interrupt* is generated

Initiating I/O - Channel Program

- Channel program
  - Consist of channel command words (*CCWs*)
  - Each ccw refers a specific command (e.g. read, write) and may refer to a memory area
  - Multiple ccws may be chained (e.g. multiple reads) and started by a single SSCH
  - Running channel programs may be modified in-flight
  - Special features: TIC (GOTO equivalent), suspend marker, program controlled interrupts

Initiating I/O - SSCH and Its Paramaters



ORB:

- intparm
- controls
- cpa

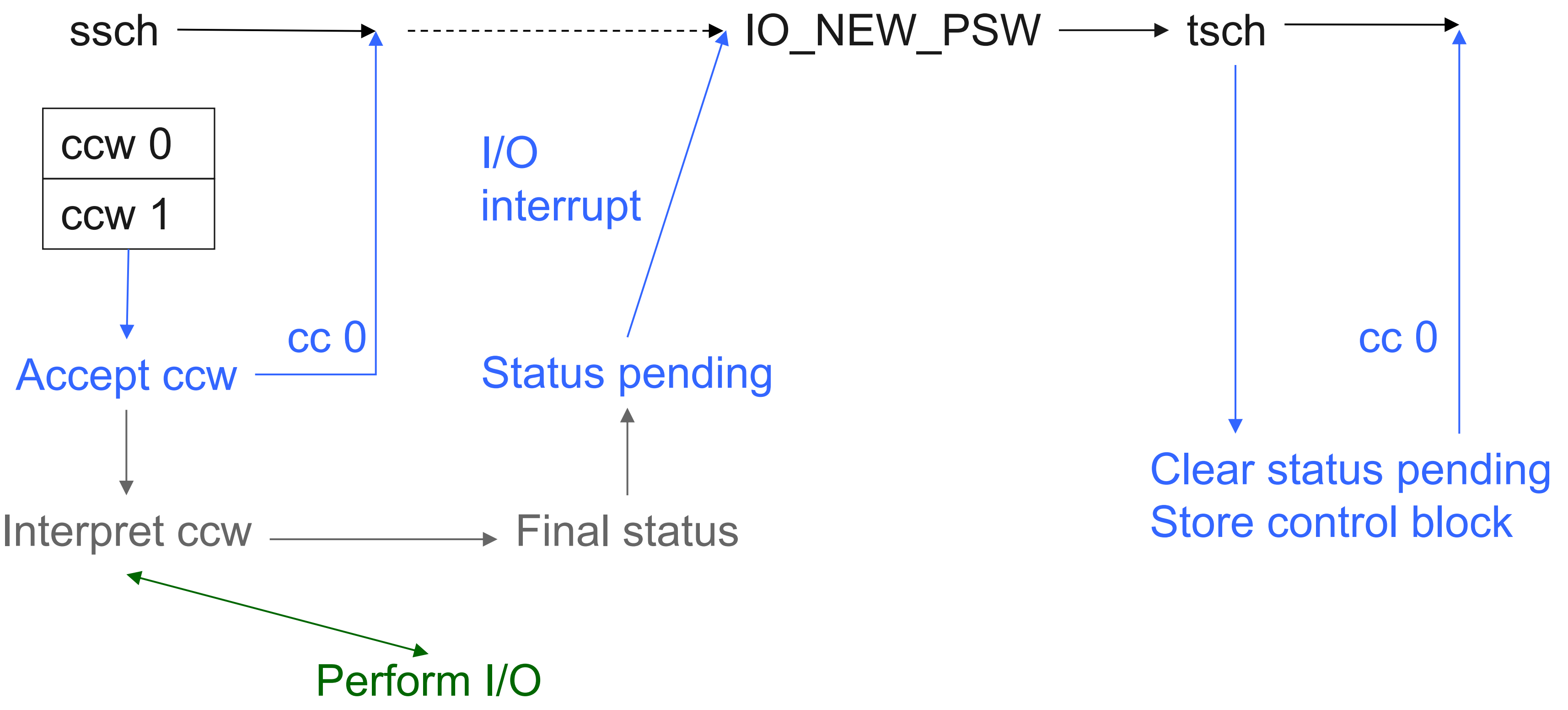
Channel program:

cmd	flag	cnt	addr
83			
01		100	
01		150	
...			
01		80	

Initiating I/O - Interrupts

- I/O Interrupts
  - Floating interrupt
  - Payload is saved into CPU's lowcore
  - Pending I/O interrupts may be removed by I/O instructions
    - TPI – test pending interruption
    - TSCH – test subchannel
  - Usually triggers a TSCH by the program to collect subchannel status

Initiating I/O - A Typical Process



ssch

IO_NEW_PSW

tsch

ccw 0

ccw 1

I/O interrupt

cc 0

Accept ccw

Status pending

cc 0

Clear status pending
Store control block

Interpret ccw

Final status

Perform I/O

- Basic Concepts
- Initiating I/O
- **Linux Support for Channel I/O**
- Virtualization Support
- Channel I/O Passthrough

Linux Support for Channel I/O

The Common I/O layer, driver model and sysfs.

Linux Support for Channel I/O - Common I/O Layer

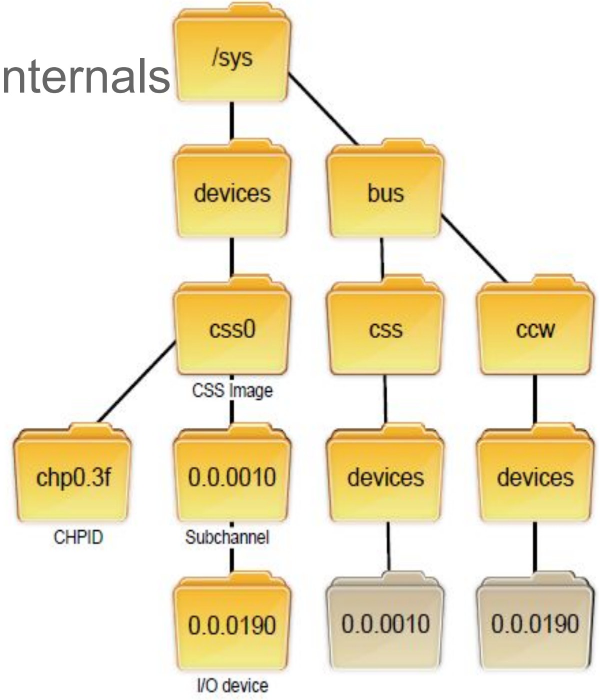
- Common I/O Layer
  - Provides wrapper around low-level channel I/O
  - Handles basic channel I/O and I/O interrupts

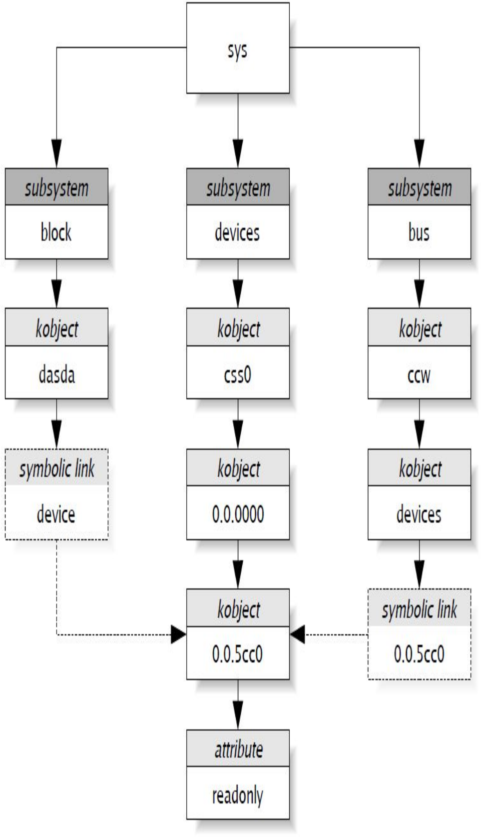
Linux Support Channel I/O - Driver Model

- CSS device drivers
  - subchannels.. I/O, message, eadm...
- CCW device drivers
  - Support for various devices and control units
  - Channel commands specific to device types
  - Examples: dasd, channel attached tapes
- Driver model integration
  - Channel paths as simple objects
  - Channel subsystem image css0 as root
  - Configuration via sysfs attributes
  - Hotplug events generate uevents

Linux Support Channel I/O - sysfs

- sysfs: virtual filesystem
- Representation of driver model internals
  - Hierarchy
  - Attributes
  - Symbolic links (views)





Linux Support Channel I/O - s390tools

- Example of a guest running on LPAR:

```
[root@s38lp25 ~]# lscss
Device   Subchan.  DevType CU Type Use  PIM PAM POM  CHPIDs
----------------------------------------------------------------------
0.0.1220 0.0.000a  1732/01 1731/01      80  80  ff   85000000 00000000
0.0.1221 0.0.000b  1732/01 1731/01      80  80  ff   85000000 00000000
0.0.1222 0.0.000c  1732/01 1731/01      80  80  ff   85000000 00000000
0.0.1240 0.0.000d  1732/01 1731/01      80  80  ff   95000000 00000000
0.0.1241 0.0.000e  1732/01 1731/01      80  80  ff   95000000 00000000
0.0.1242 0.0.000f  1732/01 1731/01      80  80  ff   95000000 00000000
0.0.1700 0.0.0010  1732/03 1731/03      80  80  ff   50000000 00000000
0.0.1701 0.0.0011  1732/03 1731/03      80  80  ff   50000000 00000000
0.0.1780 0.0.0012  1732/03 1731/03      80  80  ff   52000000 00000000
0.0.1781 0.0.0013  1732/03 1731/03      80  80  ff   52000000 00000000
0.0.1800 0.0.0014  1732/03 1731/03      80  80  ff   58000000 00000000
```

- Basic Concepts
- Initiating I/O
- Linux Support for Channel I/O
- **Virtualization Support**
- Channel I/O Passthrough

Virtualization Support

How we do I/O interception and handling?

Virtualization Support - Hardware

- SIE: Virtualization instruction on s390
- I/O instructions get SIE exits
  - Instruction intercept for most I/O instructions
  - Additionally I/O intercept for SSCH
  - Special intercepts for passthrough of real channel devices

Virtualization Support - Software

- Handling I/O
    - Perform path-related operations
    - Interpret channel programs
        - Doing this for arbitrary channel programs is the most complex part!
    - Actually do I/O
        - Either on virtual backend (virtio, …)
        - Or on real (passthrough) I/O device
- Keep subchannel control blocks up to date

Virtualization Support - Software (Cont.)

- Interception requests for injecting I/O interrupts
  - Drop VCPU out of SIE
- I/O interrupts may be cleared by tsch/tpi
- Hypervisor needs to keep track of interrupt payload
  - subchannel ID, interruption parameter

Virtualization Support - Current Status

- Current status for KVM and QEMU:
  - Support for I/O interrupts and related I/O instructions (tsch, tpi) in KVM
  - Support for I/O instructions on virtual subchannels in QEMU (virtual css)
    - i.e. only emulated devices in the virtual channel subsystem 0xfe
  - Support virtio-ccw as the transport
  - vfio-ccw is in progress

- Basic Concepts
- Initiating I/O
- Linux Support for Channel I/O
- Virtualization Support
- **Channel I/O Passthrough**

Channel I/O Passthrough

vfio-ccw: based on vfio-mdev.

Channel I/O Passthrough - Concepts

- Assign host device (subchannels) to a specific guest
  - Usage is exclusive
- Guest sees a normal channel device (e.g. dasd)
  - As opposed to a virtio-ccw proxy device
- Similar to PCI passthrough

Channel I/O Passthrough - VFIO

- VFIO
    - Virtual Function I/O
    - A secure, userspace driver framework
        - Secure and IOMMU protected environemnt
        - Expose direct device access to userspace
        - Allow safe, non-privileged, userspace drivers
- Why VFIO?
    - The means to do I/O passthrough in Linux.
    - --> vfio-ccw (same level with vfio-pci)

Channel I/O Passthrough - TYPE1 IOMMU

- VFIO IOMMU models:
  - IOMMU API (type1)
  - ppc64 (SPAPR)
  - NOIOMMU
- Why TYPE1?
  - NOIOMMU is not the case
    - Althought Channle I/O doesn't have a hardware IOMMU support
  - We are definitely not SPAPR
  - We don't want/need/allow to invent a new type:
    - zPCI adapted itself to type1 already
    - We do not have a special requirement that beyond the offerings from type1
  - --> vfio_iommu_type1

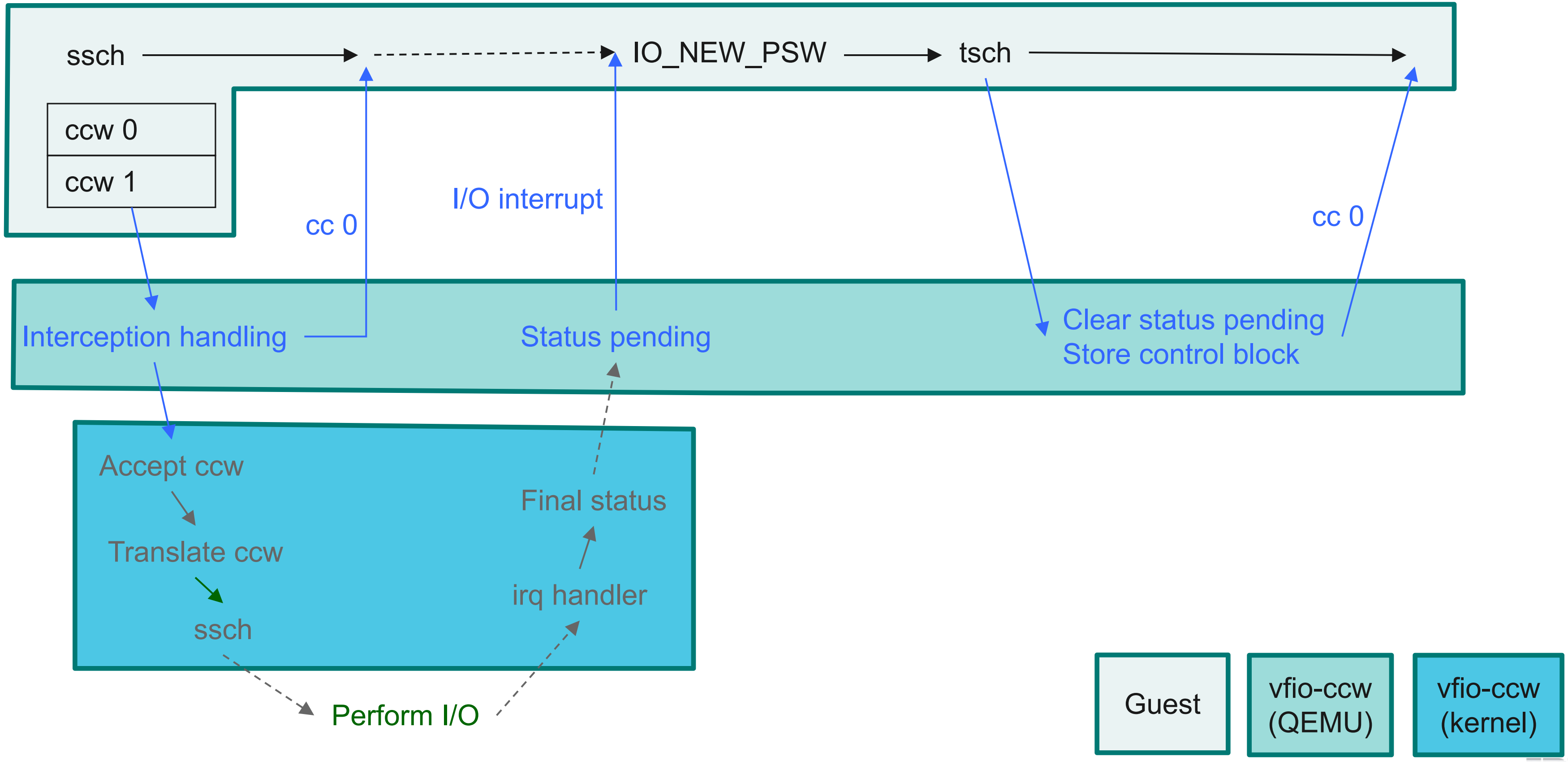
Channel I/O Passthrough - Mediated Device

- Mediated device support
  - A framwork to manage mediated device (e.g. vGPU)
  - Still under discussion in the community
  - vfio-mdev
- Why MDEV?
  - The the channel I/O instruction handling makes it looking like a MDEV
  - The channel subsystem will access any memory designated by the caller in the channel program directly
    - We have software principling and translating the ccws
    - iova -> hpa
  - --> vfio-ccw should provide a parent device and callbacks to create a mdev

Channel I/O Passthrough - Ideas

- Host needs to treat passed-through devices specially
  - Don't touch devices with regular host drivers
  - Match with special passthrough subchannel driver (vfio_ccw.ko)
- I/O requests for passed-through devices will be intercepted
  - Need to interpret channel program
  - Translate memory addresses and execute on host
- Interrupts will be received by host
  - Need to inject into guest
- Add passthrough device options to qemu
  - A new vfio-ccw device model

Channel I/O Passthrough - Ideas (Cont.)



Channel I/O Passthrough - SSCH Handling Process

Q1-Q7: Qemu side process. **K1-K7: Kernel side process.**

Q1. Gets I/O region info during initialization.
Q2. Setup event notifier and handler to handle I/O completion.

... ...

Q3. Intercept a SSCH instruction.
Q4. Write the guest CCW program and ORB to the I/O region.
K1. Copies from guest to kernel.
K2. Performs address translation on the guest's CCW program.
K3. With the necessary information contained in the ORB passed in by Qemu, issue the CCW program to the device.
K4. Return the SSCH condition code.
Q5. Returns the condition code to the guest.

... ...

K5. Interrupt handler gets the I/O result and writes the result to the I/O region.
K6. Manipulates the IRB, to match the guest's request.
K7. Uses eventfd_signal to report the interrupt to Qemu.
Q6. Gets the event, the event handler reads the IRB from the I/O region.
Q7. Injects the I/O interrupt in the guest

Channel I/O Passthrough - QEMU Device Modelling

- vfio-ccw device modelling:
  - -M s390-ccw-virtio,s390-map-css=on
  - -device vfio-ccw
    - hostid=0.0.013f
    - guestid=0.0.1234
    - mdevid=6dfd3ec5-e8b3-4e18-a6fe-57bc9eceb920

Channel I/O Passthrough - Example

- Usage

```
insmod drivers/vfio/vfio.ko
insmod drivers/vfio/mdev/mdev.ko
insmod drivers/vfio/vfio_iommu_type1.ko
insmod drivers/s390/cio/vfio_ccw.ko
insmod drivers/vfio/mdev/vfio_mdev.ko

echo 0.0."$devno" > /sys/bus/ccw/devices/0.0."$devno"/driver/unbind
echo 0.0."$schid" > /sys/bus/css/devices/0.0."$schid"/driver/unbind
echo 0.0."$schid" > /sys/bus/css/drivers/vfio_ccw/bind

echo "$UUID" \
> /sys/bus/css/devices/0.0."$schid"/mdev_create

qemu-system-s390x \
-M s390-ccw-virtio,s390-map-css=on -enable-kvm -m 2048 -nographic \
-device vfio-ccw,id=pass0,hostid=$schid,guestid=0.0.1234, \
 mdevid=$UUID \
... ...
```

Channel I/O Passthrough - Example (Cont.)

- Example of a guest running under qemu with vfio-ccw:

```
[root@localhost ~]# lscss
Device   Subchan.  DevType CU Type Use  PIM PAM POM  CHPIDs
----------------------------------------------------------------------
0.0.0000 0.0.0000  0000/00 3832/01 yes  80  80  ff   00000000 00000000
0.0.0001 0.0.0001  0000/00 3832/02 yes  80  80  ff   00000000 00000000
0.0.1234 0.0.1234  3390/0c 3990/e9      f0  f0  ff   42434445 00000000
[root@localhost ~]# chccwdev -e 0.0.1234
Setting device 0.0.1234 online
dasd-eckd 0.0.1234: A channel path to the device has become operational
dasd-eckd 0.0.1234: New DASD 3390/0C (CU 3990/01) with 30051 cylinders, 15 heads, 224 sectors
dasd-eckd 0.0.1234: DASD with 4 KB/block, 21636720 KB total size, 48 KB/track,
 compatible disk layout dasda:VOL1/  0X3F3F: dasda1
Done
[root@localhost ~]# lsdasd
Bus-ID     Status      Name      Device  Type  BlkSz  Size      Blocks
==============================================================================
0.0.1234   active      dasda     94:0    ECKD  4096   21129MB   5409180
```

Credits

Thanks and credits goes to:

Cornelia Huck <cornelia.huck@de.ibm.com>

for borrowing me part of the slides.

Thanks!
&
Questions?