Virtualization Solutions in Freescale Linux SDK(1) – Hypervisor(topaz)

Virtualization solutions are hardware and software technologies which provide an abstraction layer that enables running multiple operating systems on a single system. There are three kinds of virtualization solutions provided in Freescale Linux SDK, they are KVM/QEMU, Hypervisor(Topaz) and Linux Container. This document introduces Freescale Hypervisor(Topaz) technology.

1 Freescle Hypervisor overview

The Freescale hypervisor is a layer of software that enables the efficient and secure partitioning of a multi-core system.

Hypervisor is focused on static partitioning (supervised AMP), a system's CPUs, memory and I/O devices can be divided into logical partitions. These partitions are isolated from one another, and the configuration is fixed until a reconfiguration and the system reboot. So far, hypervisor is supported on e500mc/e5500/e6500 platform.

1.1 Hypervisor Boot sequence

Hypervisor must be booted with an ePAPR compliant boot program such as u-boot. U-boot loads the hypervisor program image into memory, configures the hardware device tree and loads it into memory with the bootargs property on /chosen node set to specify the address of the hypervisor configuration tree. Then u-boot transfer control to the hypervisor image on the boot CPU.

Hypervisor initializes each partition and boots guest OS

1. Create a guests device tree and copy to the partition's memory.
2. Load all images into the partition's memory as specified by the load-image-table, guest-image, and linux-rootfs properties.
3. Initializes the virtual CPU
4. Transfer control to the guest operating system on the boot CPU for that partition.

1.2 Guest Physical Address

Guest software executing in a partition sees a physical address space created by the hypervisor that may not directly correspond to the true physical addresses in the system hardware. Hypervisor maintains this mapping of the guest physical to the true physical addresses, which is completely transparent to guest software. Hardware TLBs contains mapping of guest virtual addresses to true physical addresses.

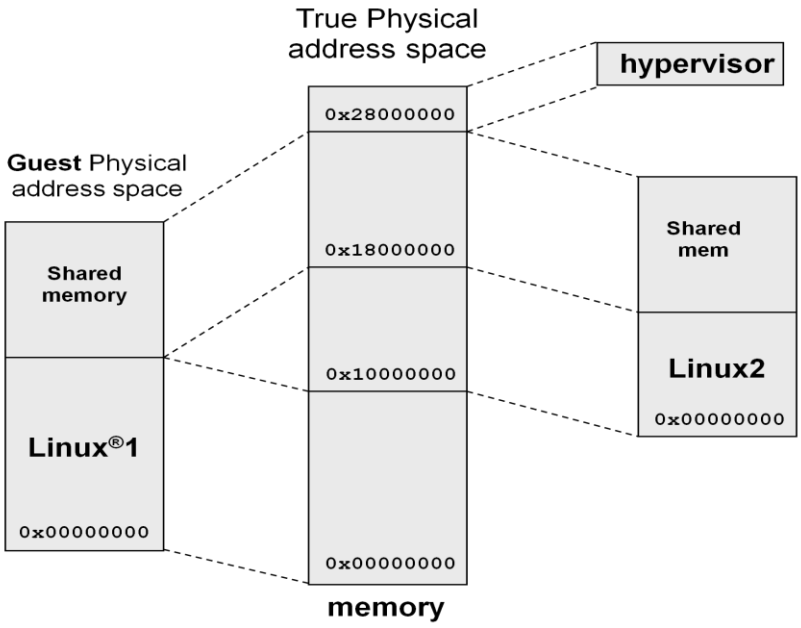

Figure 1-1 guest physical and true physical addresses

1.3 Hypervisor device trees

The hypervisor configuration tree contains all hypervisor configuration parameters and partition definition information. Based on this configuration tree, the hypervisor dynamically creates ePAPR-compliant guest device trees for each partition. A guest device tree describes the resources available to the partition including—CPUs, memory, I/O devices, and other virtual resources.

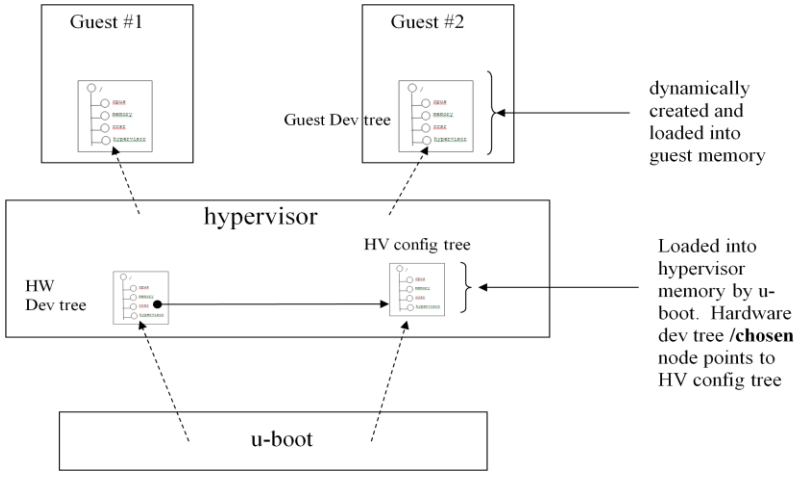

Figure 1-2 Hypervisor configuration trees

The following is a simple example about partition definition in hypervisor device tree. The partition contains two CPUs, one memory region, and a UART I/O device.

```
// ================================================

// Partition 1

// ================================================

Part1 {

     compatible = “partition”;

     cpus = <2 2>;

     guest-image = <0 0xe8200000 0  0  0  0x200000>;

     linux-rootfs = <0 0xe9000000 0  0 0x01300000 0 0x02800000>;

     dtb-window = <0 0x01000000 0 0x10000>;

    gpma {

            compatible = “guest-phys-mem-area”;

                    phys-mem = <&pma1>;

                    guest-addr = <0 0>;

   };

   Serial2 : serial2 {

          Device = “/soc/serial@11d500”;

  };

};
```

The cpus property specifies that the partition has 2 CPUs starting at physical#2, cpu 2-3, those cpus map to virtual cpus 0-1 in the partition. The vcpu(virtual CPU)emulates a physical CPU, and the behavior of instructions, registers, and exceptions on the vcpu is nearly identical to the physical CPU being emulated.

The guest-image property specifies the source address, destination address, and size of an image to be executed when a partition is started. In the example, the guest’s program image to be loaded is located at true physical 0xe2000000. It is to be copied to guest physical 0x0. The max size of the image is 0x200000.

The linux-rootfs property specifies the source/destination and size of Linux root filesystem. The root filesystem image to be loaded is located at physical address 0xe9000000. It is to be copied to guest physical 0x01300000. The max size of the image is 0x02800000.

The dtb-window defines a 64KB window at address 0x01000000 where the guest DTB should be loaded.

The gpma property defines one guest physical memory area corresponding to physical memory area pmal. The guest physical area for the gpma is 0x0.

2 . How to Set up Hypervisor

In Linux SDK, the default configuration is for setting up hypervisor from NOR Flash, this document will introduce how to modify hypervisor device tree and boot hypervisor system from RAM.

2.1 Modify hypervisor device tree

In SDK 1.6 hv.dts, there is a node defining one physical memory area for booting from RAM, so all the images should be deployed inside this area.

```
images_pma: images_pma {
compatible = "phys-mem-area";
addr = <0x0 0x78000000>;          // Used for boot-from-RAM
size = <0x0 0x04000000>;          // 64MB
};
```

Modify the guest images addresses as the following.

```
part1 {

    // Indicates that it is a partition node

    compatible = "partition";

    label = "p1-linux";

    // CPUs #0 to #6 are assigned to this partition

    cpus = <0 6>;

    guest-image = <0x0 0x78020000 0 0 0 0x700000>;

    linux-rootfs = <0x0 0x79300000 0 0x01300000 0 0x02800000>;

    dtb-window = <0 0x1000000 0 0x20000>;
```

```
... ...

part2 {
compatible = "partition";
label = "p2-linux";
// CPU #7 is assigned to this partition
cpus = <7 1>;
guest-image = <0x0 0x78020000 0 0 0 0x700000>;
linux-rootfs = <0x0 0x79300000 0 0x01300000 0 0x02800000>;
dtb-window = <0 0x1000000 0 0x20000>;
… …

}
```

2.2 Deploy hypervisor images

Under u-boot prompt, deploy the following images.

```
=>tftp 78020000  uImage-p4080ds.bin

=>tftp 79300000  fsl-image-core-p4080ds.ext2.gz.u-boot

=>tftp 78700000  hv.uImage

=>tftp 0x78800000  uImage-p4080ds-usdpaa.dtb

=>tftp 0x78900000 hv.dtb
```

Here specify the physical address of hv.dtb from u-boot environment.

```
=>setenv bootargs config-addr=0x78900000 console= ttyS0,115200

=>bootm 0x78700000 - 0x78800000
```

2.3 Use mux_server to connect to partitions

Use the mux_servre to the target board to ensure that individual UART streams are decoded correctly on the host side.

```
mux_server -exec "bft connect P4080DS-1" 18900 18901 18902&
```

Note: “bft connect P4080DS-1” is the command to connect to the UART console.

If the device is attached at /dev/ttyS0 and export mux channels, the command should be as the following.

```
mux_server  /dev/ttyS0  18900 18901 18902&
```

socat -,raw,echo=0 tcp:0:18900

socat -,raw,echo=0 tcp:0:18901

socat -,raw,echo=0 tcp:0:18902