TOSHIBA

Leading Innovation >>>

Applying Jailhouse to the Civil Infrastructure System

Masaki Miyagawa
Toshiba Corporation

2017 June 23

Agenda

- **Civil Infrastructure System**
- **Jailhouse**
  - Demonstration in QEMU/KVM
  - IVSHMEM
  - Applying Civil Infrastructure System
- **Conclusion**

Agenda

- **Civil Infrastructure System**
- **Jailhouse**
  - Demonstration in QEMU/KVM
  - IVSHMEM
  - Applying Civil Infrastructure System
- **Conclusion**

Civil Infrastructure System

- **Why we use Linux?**
  - It is easy to use many types of communication library.
  - Many types of CPU architecture are supported.
  - There are many types of distribution that can be used for commercial use.
- **Open Source Summit 2017**
  - We demonstrated power plant controller that uses CIP Kernel.
  - https://www.cip-project.org/blog/2017/06/07/event-report-open-source-summit-japan-2017

Agenda

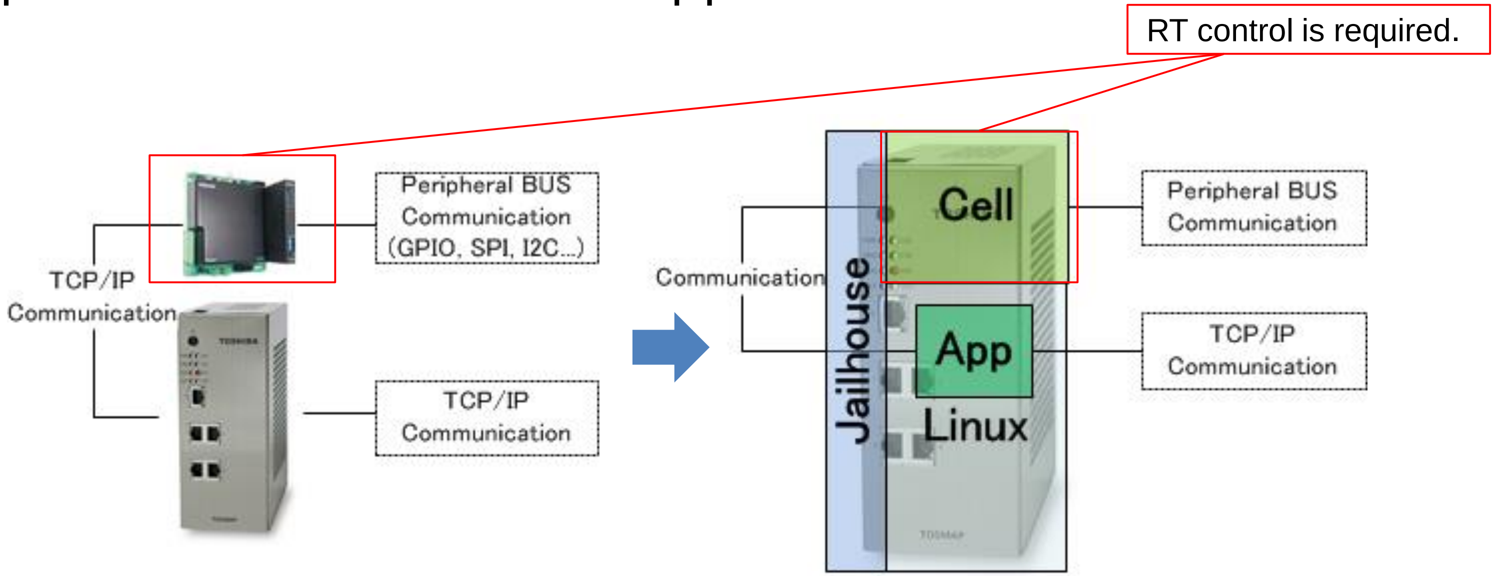
- Civil Infrastructure System
- **Jailhouse**
  - Demonstration in QEMU/KVM
  - IVSHMEM
  - Applying Civil Infrastructure System
- Conclusion

Jailhouse

- **Jailhouse**
  - is Linux-based partitioning hypervisor.
    - https://github.com/siemens/jailhouse
  - version 0.6 is released.
  - supports x86_64, ARM v7 and ARM v8.
  - manages guest application as Cell.
    - Cell occupies some hardware specified in configuration.
    - This solution is called AMP(Asymmetric Multi-Processing).

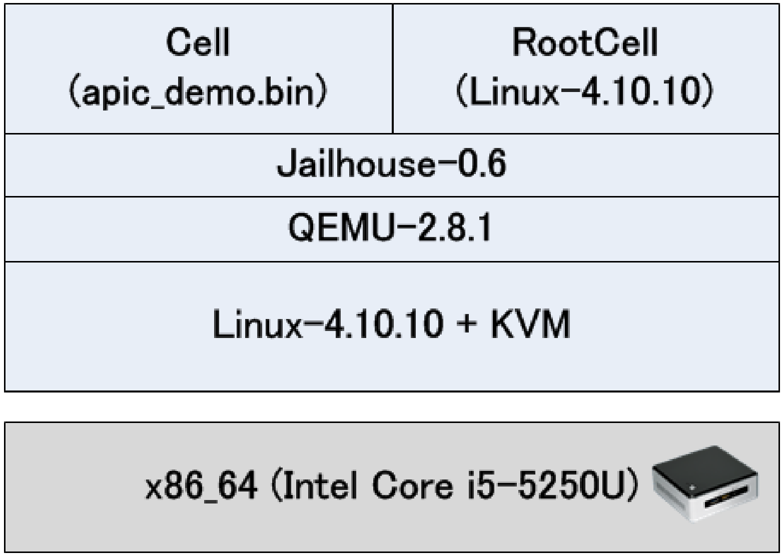
Jailhouse

- **Motivation**
  - We want to integrate the functionality of basic controller and special modules.
  - Special modules are used for RT control.
    - e.g. Turbine, Generator …
  - Traditionally, special modules are developed as a bare-metal application or RTOS based application.


RT control is required.

Jailhouse

- **How it works?**
  - “inmate” is a program running in the Cell.
  - Jailhouse v0.6 provide some “inmates” for demonstration.
  - I studied start up process of “inmates” and memory management system of Jailhouse.
- **Demo application provided by Jailhouse**
  - Demonstration in QEMU/KVM @Jailhouse-0.6/README.md



Cell
(apic_demo.bin)

RootCell
(Linux-4.10.10)

Jailhouse-0.6

QEMU-2.8.1

Linux-4.10.10 + KVM

x86_64 (Intel Core i5-5250U)

Jailhouse

- **Preparation**
  - Boot up host Linux.
    - To modify KVM parameter is needed.

```
#cat /etc/modprobe.d/kvm-nested.conf
options kvm_intel_nested=1
```

  - Execute QEMU.

```
#!/bin/sh
./qemu-system-x86_64 -machine q35,kernel_irqchip=split -m 1G -enable-kvm ¥
        -vnc 133.113.27.94:0 -k ja ¥
        -smp 4 -device intel-iommu,intremap=on,x-buggy-eim=on ¥
        -cpu kvm64,-kvm_pv_eoi,-kvm_steal_time,-kvm_asyncpf,-kvmclock,+vmx ¥
        -drive file=$1,format=qcow2,id=disk,if=none ¥
        -device ide-hd,drive=disk -serial stdio -serial vc ¥
        -netdev user,id=net -device e1000e,addr=2.0,netdev=net ¥
        -device intel-hda,addr=1b.0 -device hda-duplex
```

  - Build and install jailhouse at guest Linux(Root Cell).

```
#cd jailhouse-0.6
#make; make firmware_install
```

Jailhouse

- **Starting Jailhouse**
  - Modify grub.cfg and reboot RootCell.

```
#cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-4.10.10
root=UUID=*********************
memmap=66M$0x3b000000 ro quiet
```

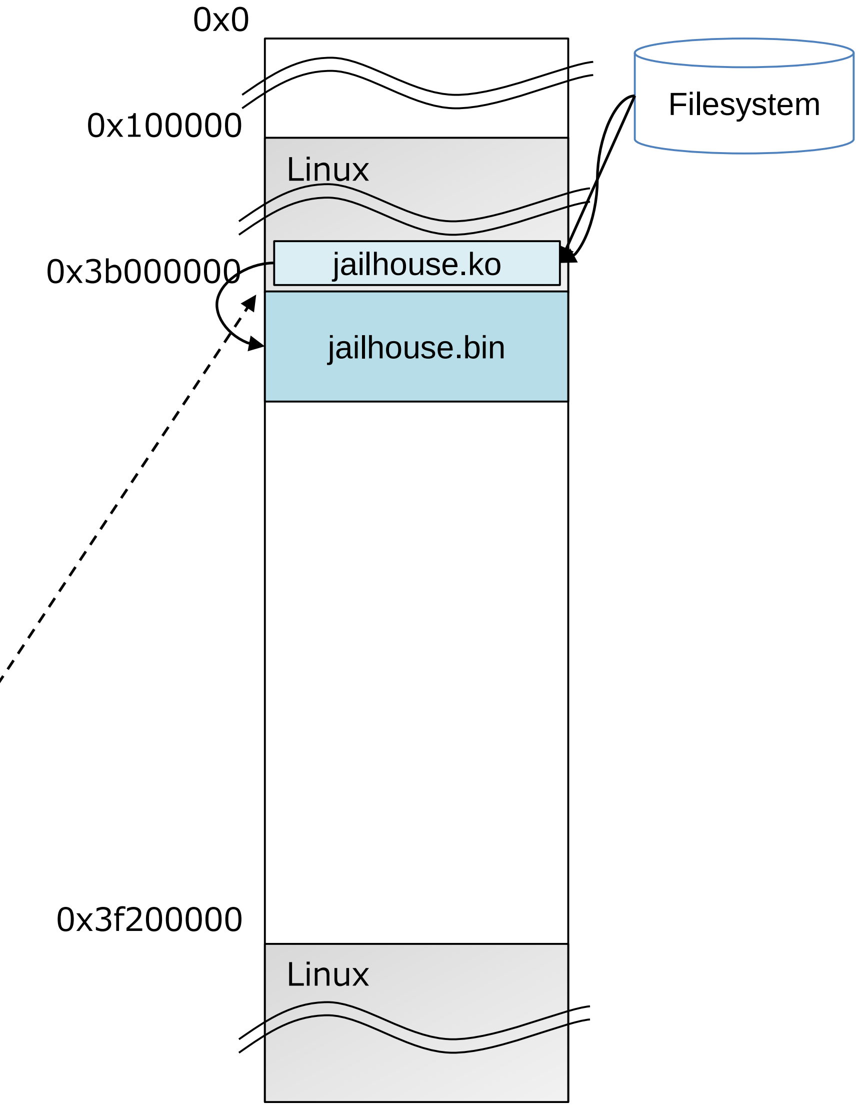
  - Load “jailhouse.ko.”

```
#insmod jailhouse-0.6/driver/jailhouse.ko
```

  - Load “jailhouse.bin.”

```
#jailhouse-0.6/tools/jailhouse enable jailhouse-
0.6/configs/qemu-vm.cell
```

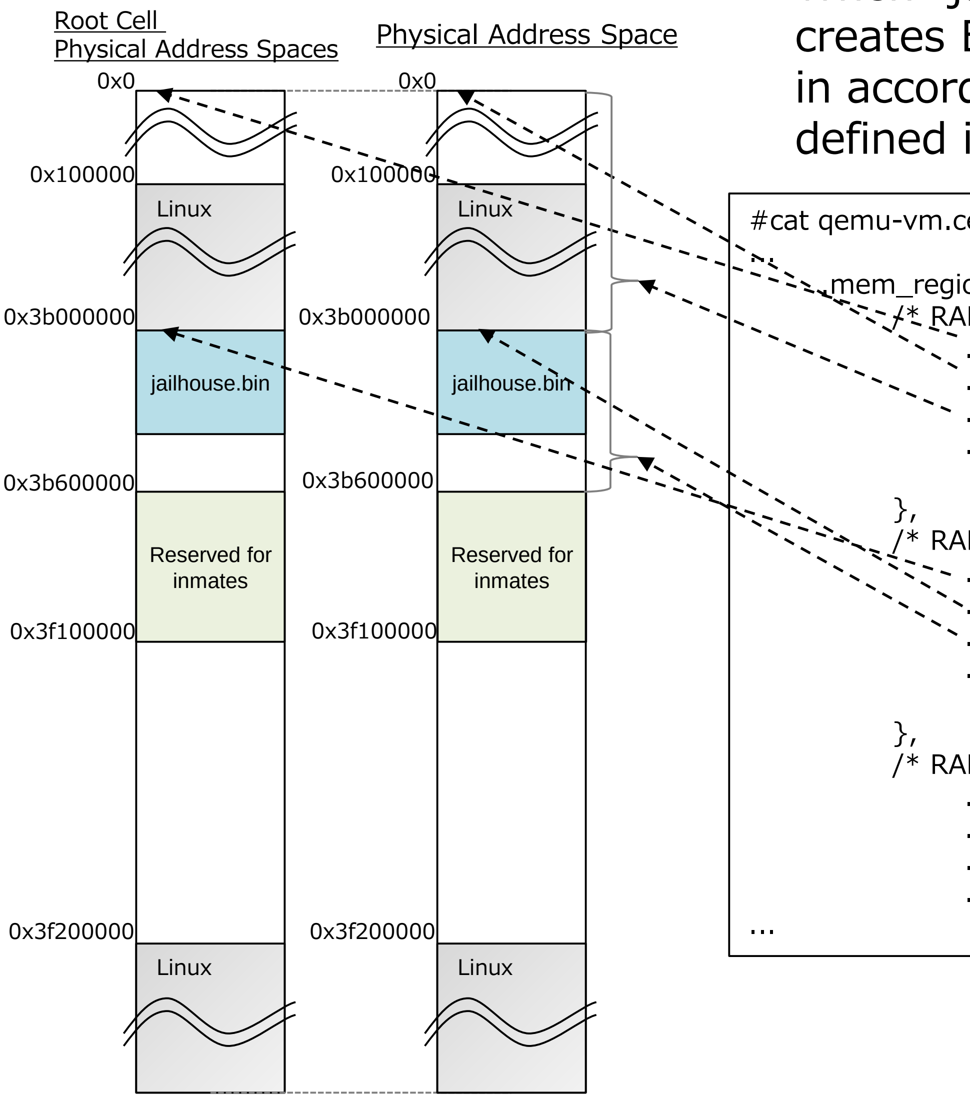
```
#cat qemu-vm.cell
...
		.hypervisor_memory = {
				.phys_start = 0x3b000000,
				.size = 0x4000000,
		},
...
```



Jailhouse

- **EPT setting for Root Cell**

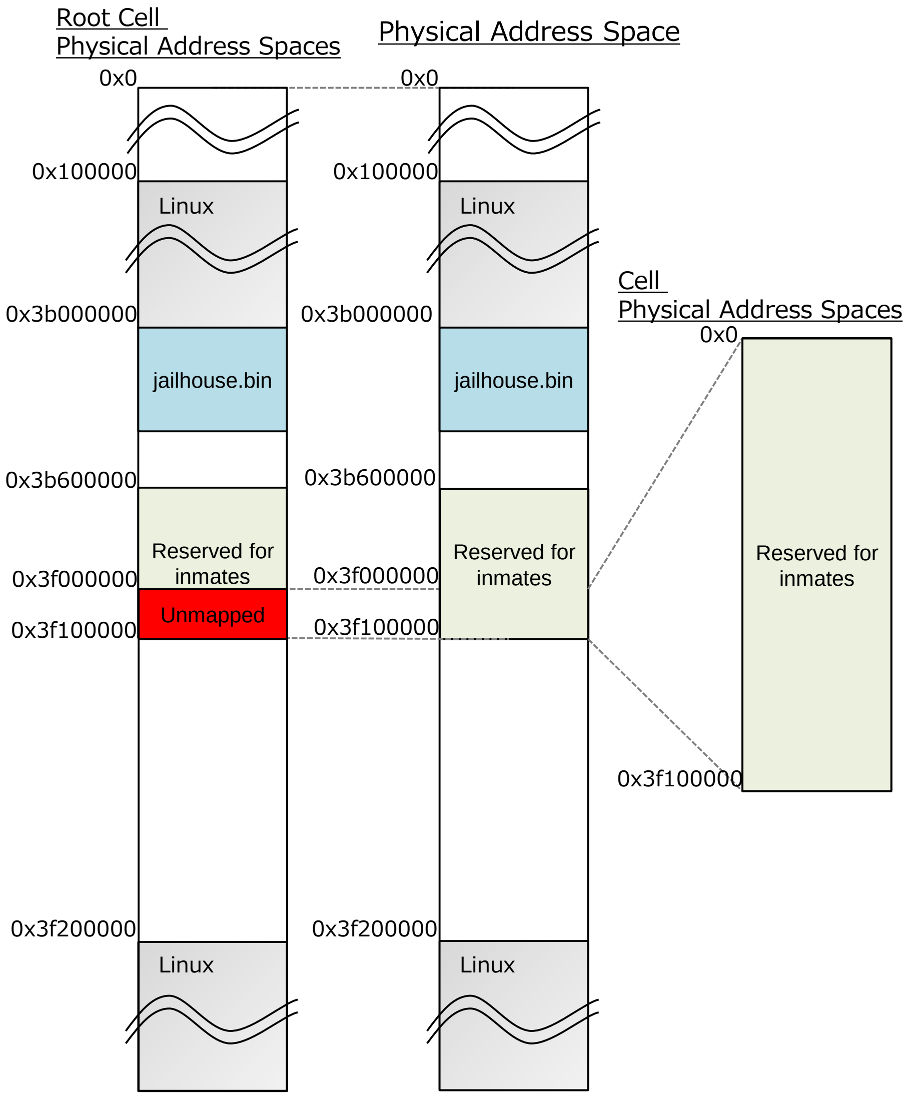
When “jailhouse.bin” is loaded, “jailhouse.bin” creates EPT for RootCell. This process is done in accordance with “struct jailhouse_memory” defined in Root Cell configuration.



```
#cat qemu-vm.cell
...
.mem_regions = {
        /* RAM */ {
                .phys_start = 0x0,
                .virt_start = 0x0,
                .size = 0x3b000000,
                .flags = JAILHOUSE_MEM_READ | JAILHOUSE_MEM_WRITE |
                        JAILHOUSE_MEM_EXECUTE | JAILHOUSE_MEM_DMA,
        },
        /* RAM (inmates) */ {
                .phys_start = 0x3b600000,
                .virt_start = 0x3b600000,
                .size = 0x3b00000,
                .flags = JAILHOUSE_MEM_READ | JAILHOUSE_MEM_WRITE |
                        JAILHOUSE_MEM_EXECUTE | JAILHOUSE_MEM_DMA,
        },
        /* RAM */ {
                .phys_start = 0x3f200000,
                .virt_start = 0x3f200000,
                .size = 0xddf000,
                .flags = JAILHOUSE_MEM_READ | JAILHOUSE_MEM_WRITE |
...
```

Jailhouse

- **EPT setting for Cell**



- **Creating Cell for “inmate”**

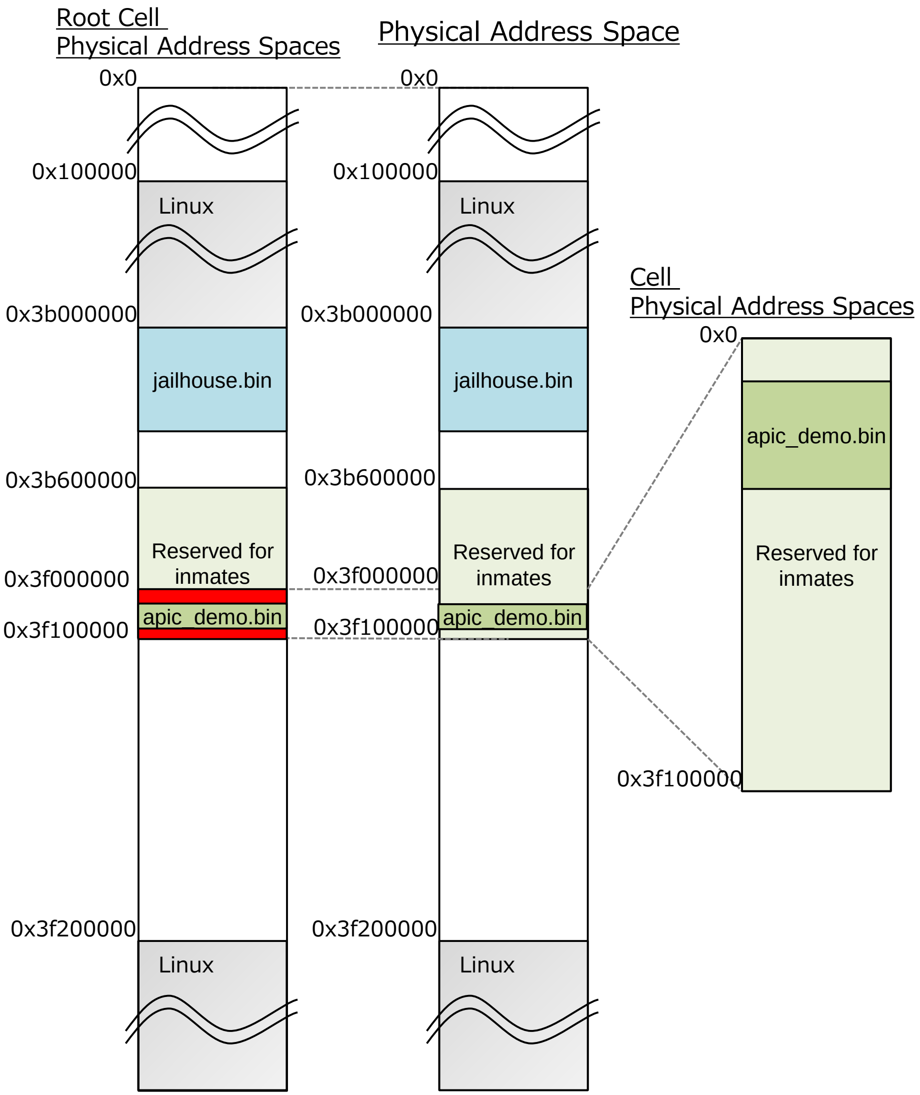
```
#jailhouse-0.6/tools/jailhouse cell create
jailhouse-0.6/configs/apic-demo.cell
```

When “jailhouse.bin” creates Cell for “inmate,” EPT setting is also created by “jailhouse.bin” in accordance with Cell configuration.

In this time, “jailhouse.bin” modifies EPT setting for Root Cell to unmap the area occupied by “inmate.”

Jailhouse

- **EPT setting**



- **Loading "inmate"**

```
#jailhouse-0.6/tools/jailhouse cell load jailhouse-0.6/inmates/demo/x86/apic-demo.bin –a 0xf0000
```

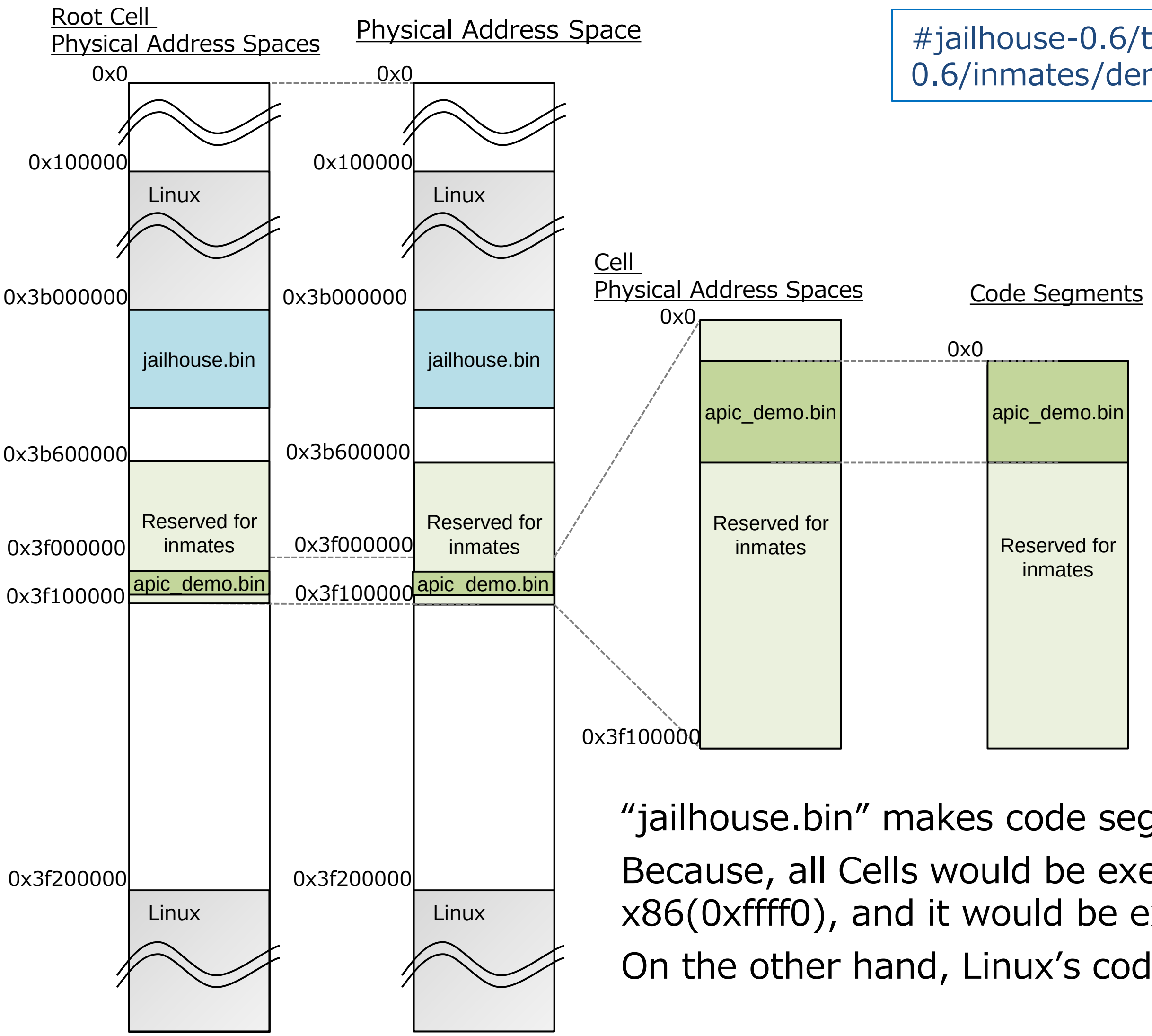
"jailhouse.ko" calls hypercall. The hypercall is handled by "jailhouse.bin" then "jailhouse.bin" remaps physical memory occupied by "inmate" to allow writing a image of "inmate" by Linux.

The image of "inmate" is loaded by Linux.

TOSHIBA
Leading Innovation >>>

Jailhouse ~Trying to run demo application~

- **EPT setting**



- **What "-a" option for?**

```
#jailhouse-0.6/tools/jailhouse cell load jailhouse-0.6/inmates/demo/x86/apic-demo.bin –a 0xf0000
```

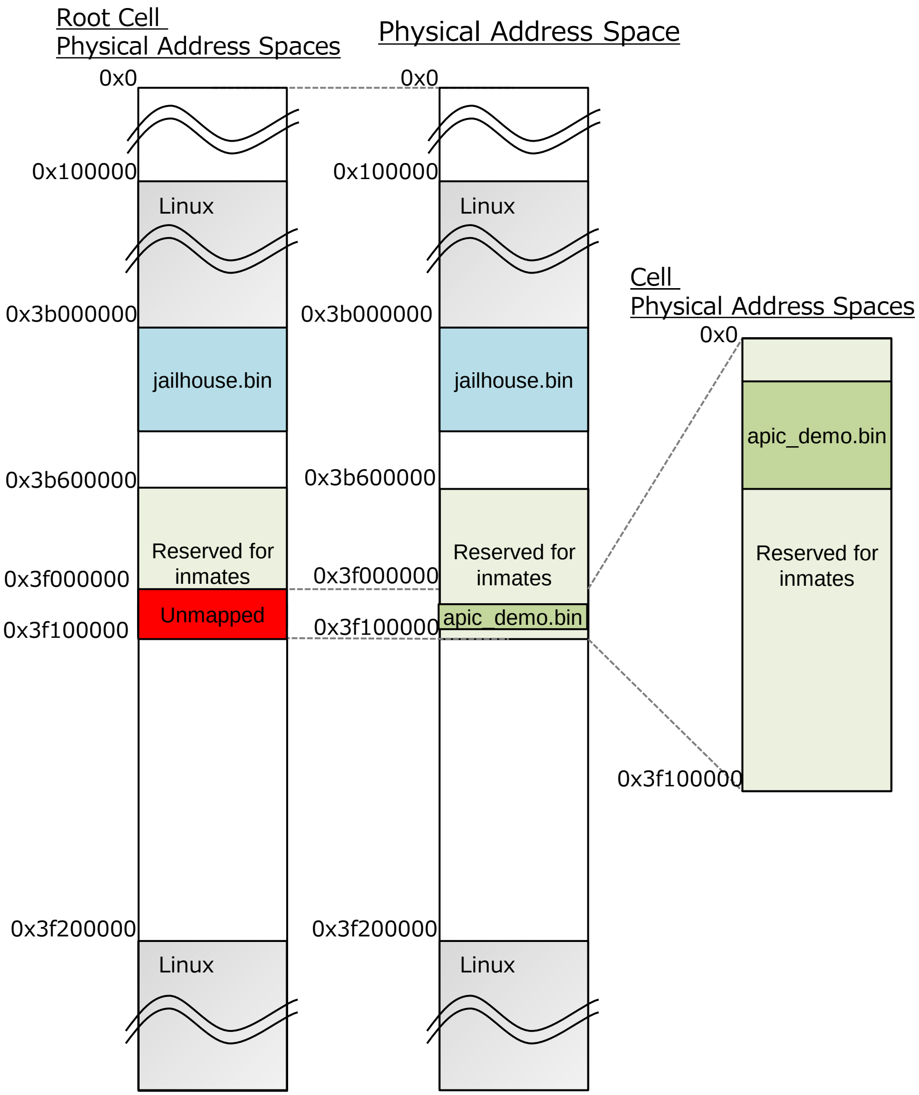
"jailhouse.bin" makes code segment of Cell starts from 0xf0000.

Because, all Cells would be executed from reset vector of x86(0xffff0), and it would be executed as real mode.

On the other hand, Linux's code segment starts at 0x0.

Jailhouse ~Trying to run demo application~

- **EPT setting**

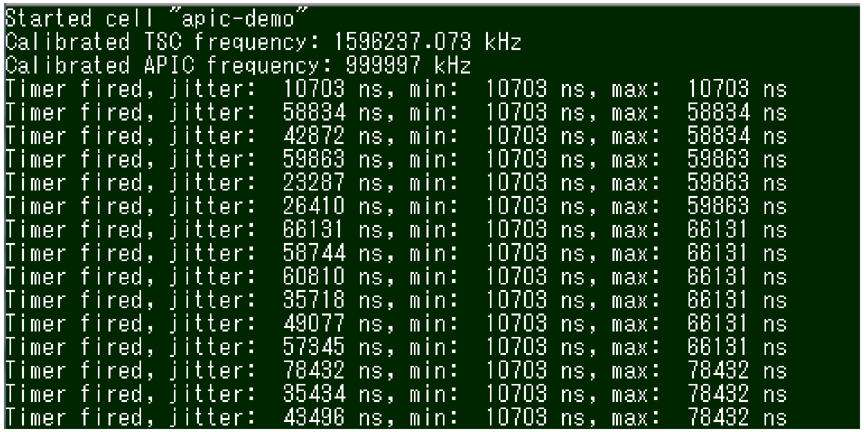


- **Starting inmate**

```
#jailhouse-0.6/tools/jailhouse cell start 1
```

“jailhouse.ko” calls hypercall. The hypercall is handled by “jailhouse.bin”, and “jailhouse.bin” unmap physical memory occupied by inmate to avoid to modify the image of “inmate” from Linux.

“apic-demo.bin” is started.



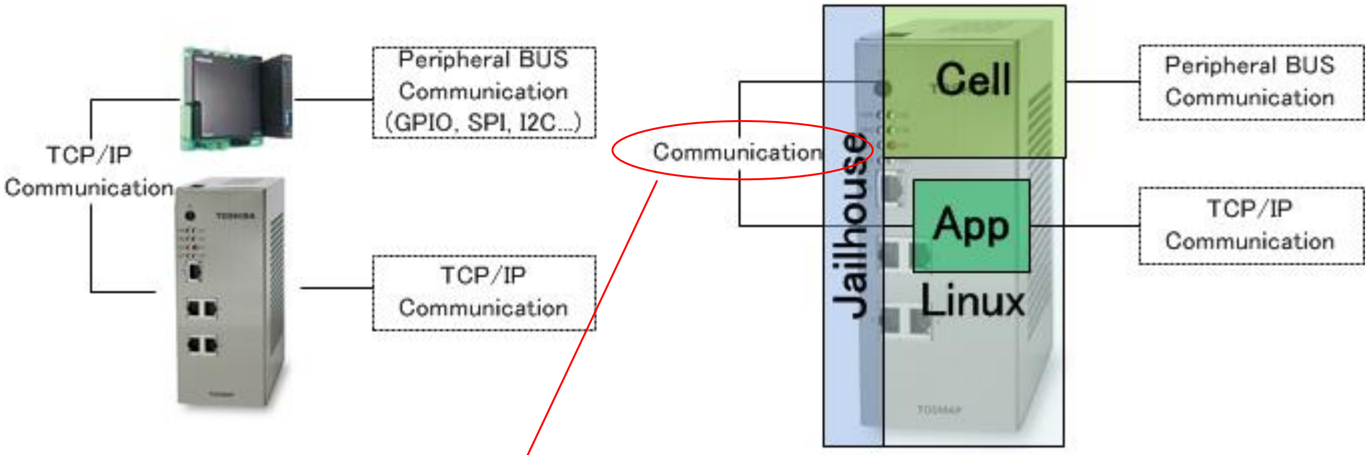
```
Started cell "apic-demo"
Calibrated TSC frequency: 1596237.073 kHz
Calibrated APIC frequency: 999997 kHz
Timer fired, jitter:  10703 ns, min:  10703 ns, max:  10703 ns
Timer fired, jitter:  58834 ns, min:  10703 ns, max:  58834 ns
Timer fired, jitter:  42872 ns, min:  10703 ns, max:  58834 ns
Timer fired, jitter:  59863 ns, min:  10703 ns, max:  59863 ns
Timer fired, jitter:  23287 ns, min:  10703 ns, max:  59863 ns
Timer fired, jitter:  26410 ns, min:  10703 ns, max:  59863 ns
Timer fired, jitter:  66131 ns, min:  10703 ns, max:  66131 ns
Timer fired, jitter:  58744 ns, min:  10703 ns, max:  66131 ns
Timer fired, jitter:  60810 ns, min:  10703 ns, max:  66131 ns
Timer fired, jitter:  35718 ns, min:  10703 ns, max:  66131 ns
Timer fired, jitter:  49077 ns, min:  10703 ns, max:  66131 ns
Timer fired, jitter:  57345 ns, min:  10703 ns, max:  66131 ns
Timer fired, jitter:  78432 ns, min:  10703 ns, max:  78432 ns
Timer fired, jitter:  35434 ns, min:  10703 ns, max:  78432 ns
Timer fired, jitter:  43496 ns, min:  10703 ns, max:  78432 ns
```

Agenda

- Civil Infrastructure System
- Jailhouse
  - Demonstration in QEMU/KVM
  - IVSHMEM
  - Demonstration : Applying Civil Infrastructure System
- Conclusion

Jailhouse

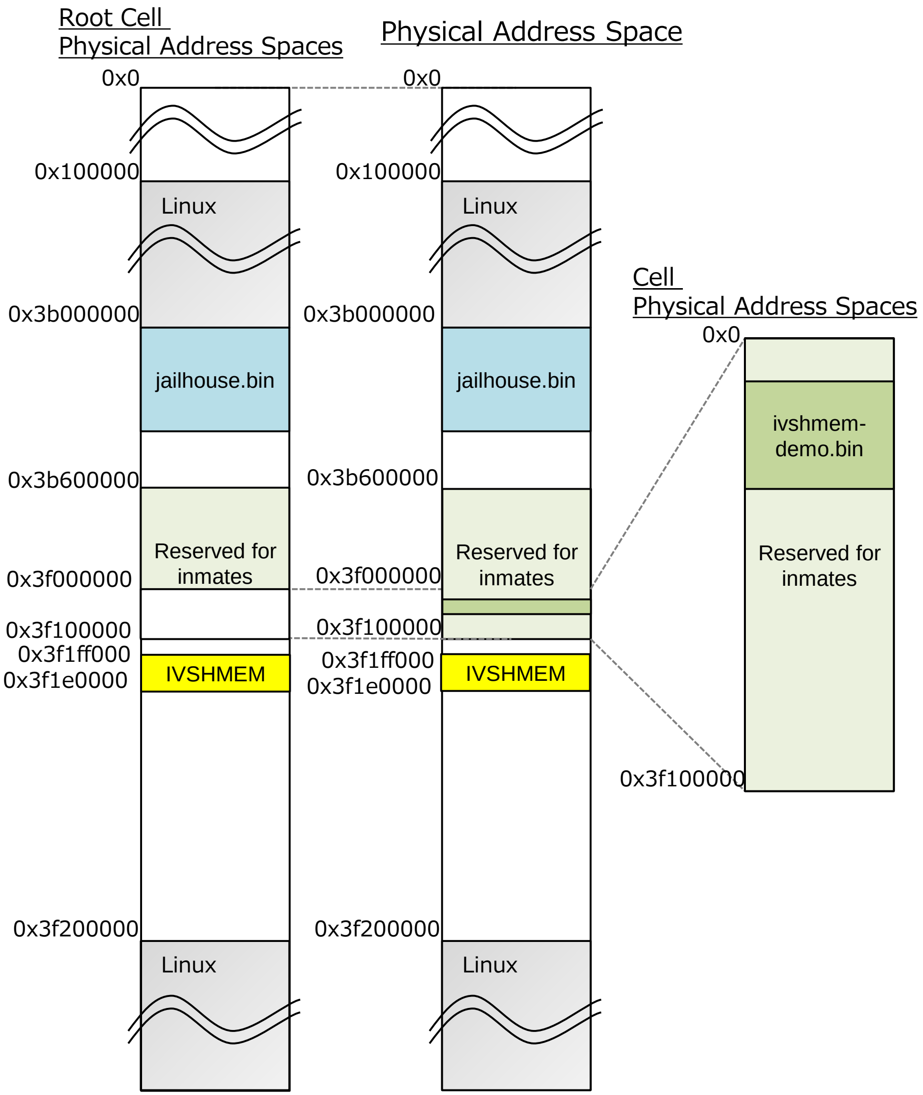
- **How can we make communication between inmate and Linux application?**
  - IVSHMEM provides Inter-Cell Communication.



This time, we studied usage of IVSHMEM.

Jailhouse

- **IVSHMEM is provided as a virtual PCI devices**



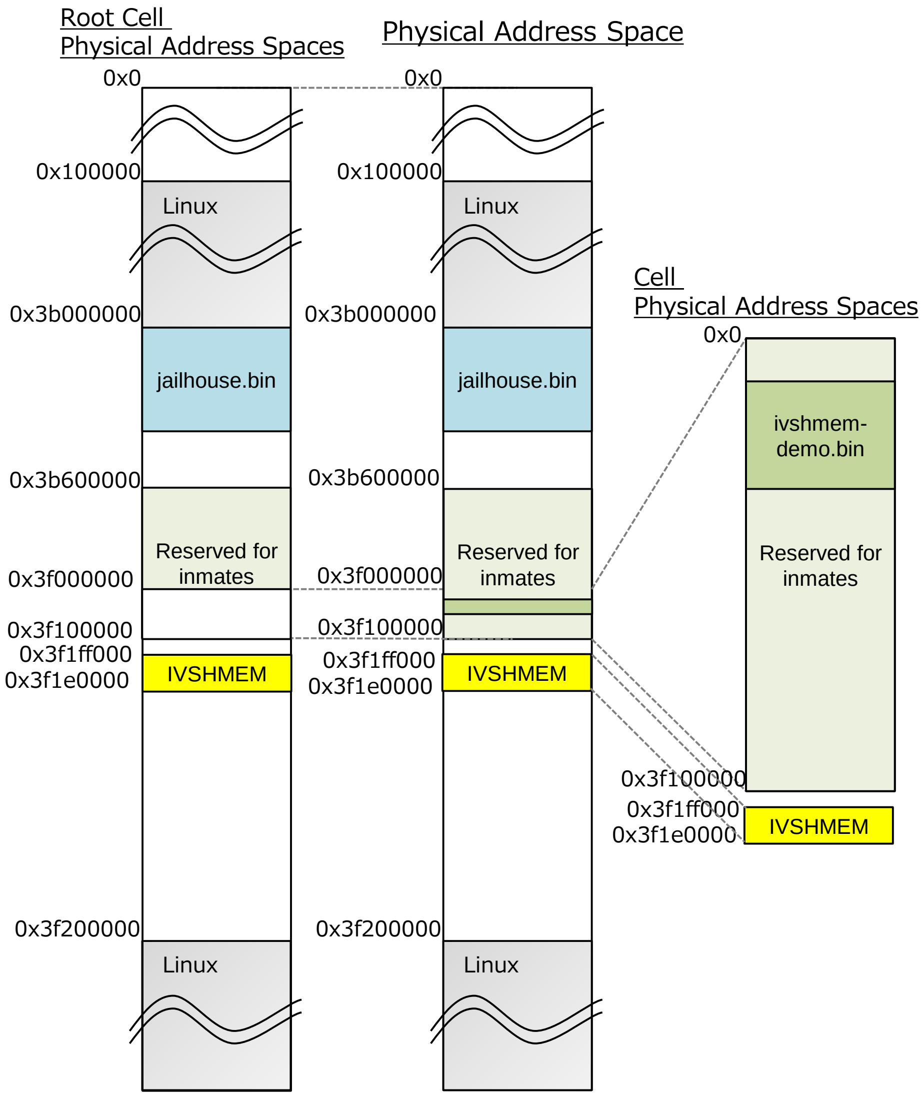
Configuration for RootCell

```
#cat jailhose-0.6/configs/qemu-vm.cell
...
        .mem_regions = {
                {
                .phys_start = 0x3f1ff000,
                .virt_start = 0x3f1ff000,
                .size = 0x1000,
                .flags = JAILHOUSE_MEM_READ |
                                JAILHOUSE_MEM_WRITE,
                },
        },
...
        .pci_devices = {
                {
                .type = JAILHOUSE_PCI_TYPE_IVSHMEM,
                .domain = 0x0000,
                .bdf = 0x0f << 3,
                .bar_mask = {
                        0xffffff00, 0xffffffff, 0x00000000,
                        0x00000000, 0xffffffe0, 0xffffffff,
                },
                .num_msix_vectors = 1,
                .shmem_region = 15,
                .shmem_protocol =
                JAILHOUSE_SHMEM_PROTO_UNDEFINED,
                },
        },
...
```

TOSHIBA
Leading Innovation >>>

Jailhouse

- **IVSHMEM is provided as a virtual PCI devices**

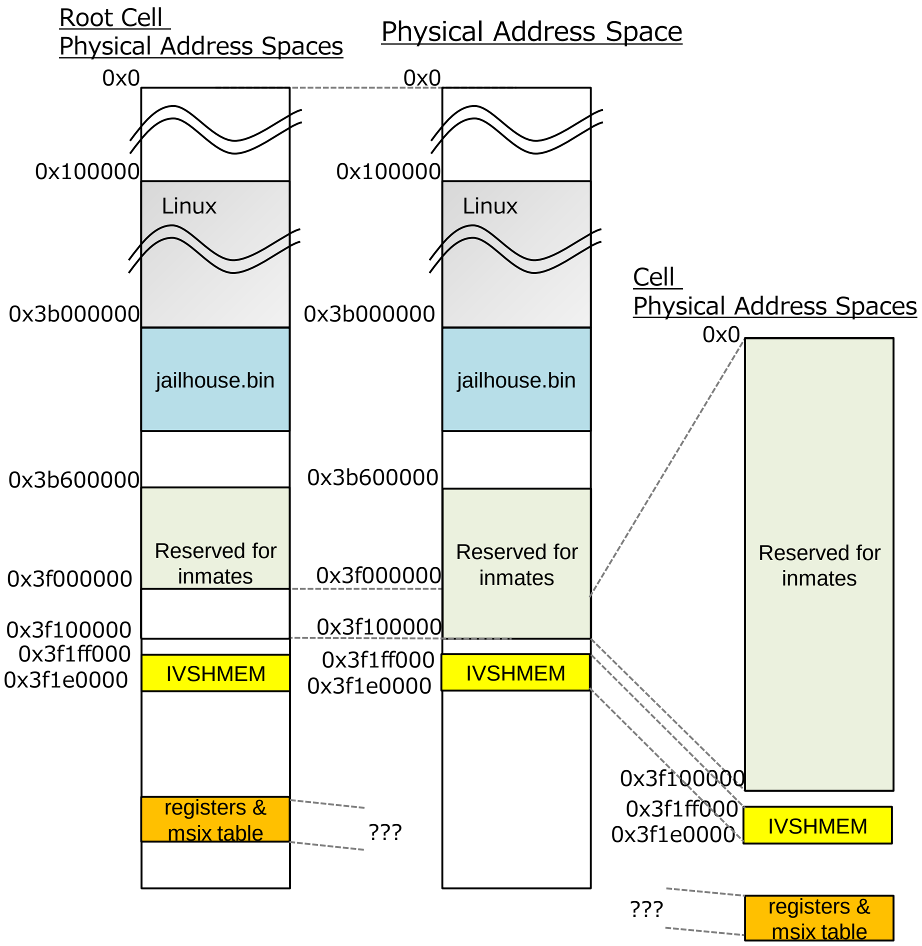


Configuration for Cell

```
#cat jailhose-0.6/configs/ivshmem-demo.cell
…
        .mem_regions = {
…
                /* IVSHMEM shared memory region */
                {
                        .phys_start = 0x3f1ff000,
                        .virt_start = 0x3f1ff000,
                        .size = 0x1000,
                        .flags = JAILHOUSE_MEM_READ |
                        JAILHOUSE_MEM_WRITE |
                        JAILHOUSE_MEM_ROOTSHARED,
                },
        },
…
        .pci_devices = {
                {
                        .type = JAILHOUSE_PCI_TYPE_IVSHMEM,
                        .domain = 0x0000,
                        .bdf = 0x0f << 3,
                        .bar_mask = {
                                0xffffff00, 0xffffffff, 0x00000000,
                                0x00000000, 0xffffffe0, 0xffffffff,
                        },
                        .num_msix_vectors = 1,
                        .shmem_region = 2,
                },
        },
…
```

Jailhouse

- **The mechanism of IRQ Sending**



“imate” defines PCI configuration registers address and set it to BAR[0] of the virtual PCI device. This physicall adderss shall not exist in EPT.

When “inmate” writes PCI configuration register, EPT violation is handled by “jailhouse.bin. “

“jailhouse.bin” sends IRQ to budy of the “inmate.”

Jailhouse

- **Prepare the IVSHMEM driver for Linux**
  - https://github.com/henning-schild/ivshmem-guest-code/tree/jailhouse

```
#cat jailhouse-0.6/Documentation/inter-cell-communmication.txt
...
You can go ahead and connect two non-root cells and run the ivshmem-demo. They
will send each other interrupts.
For the root cell you can find some test code in the following git repository:
https://github.com/henning-schild/ivshmem-guest-code
Check out the jailhouse branch and have a look at README.jailhouse.
```

Jailhouse

- **Prepare a test application for Linux**
  - PCI configuration registers area.
    - This area related to uio_info->mem[0].
    - This area will be used to send IRQ to “inmate.”
    - When mmap() is applied to uio, physical address of uio_info must be aligned to PAGE address, however it depends on location of PCI devices.

```
#cat /proc/bus/pci/devices
0078  1af41110  0  c0000004   0  0  0  c0000104   0   0  100  0  0  0  20  0  0 uio_ivshmem
0070  1af41110  0  c0000204   0  0  0  c0000124   0   0  100  0  0  0  20  0  0 uio_ivshmem
```

```
#cat linux-4.10.10/drivers/uio/uio.c
static int uio_mmap_physical(struct vm_area_struct *vma)
{
        struct uio_device *idev = vma->vm_private_data;
        int mi = uio_find_mem_index(vma);
        struct uio_mem *mem;
        if (mi < 0)
                return -EINVAL;
        mem = idev->info->mem + mi;

        if (mem->addr & ~PAGE_MASK)
                return -ENODEV;
```

BAR0 (registers).
0x4 is masked by Linux, however we were not able to mmap 0xc0000200.
We just changed order of the IVSHMEM definition in the configuration.

Jailhouse

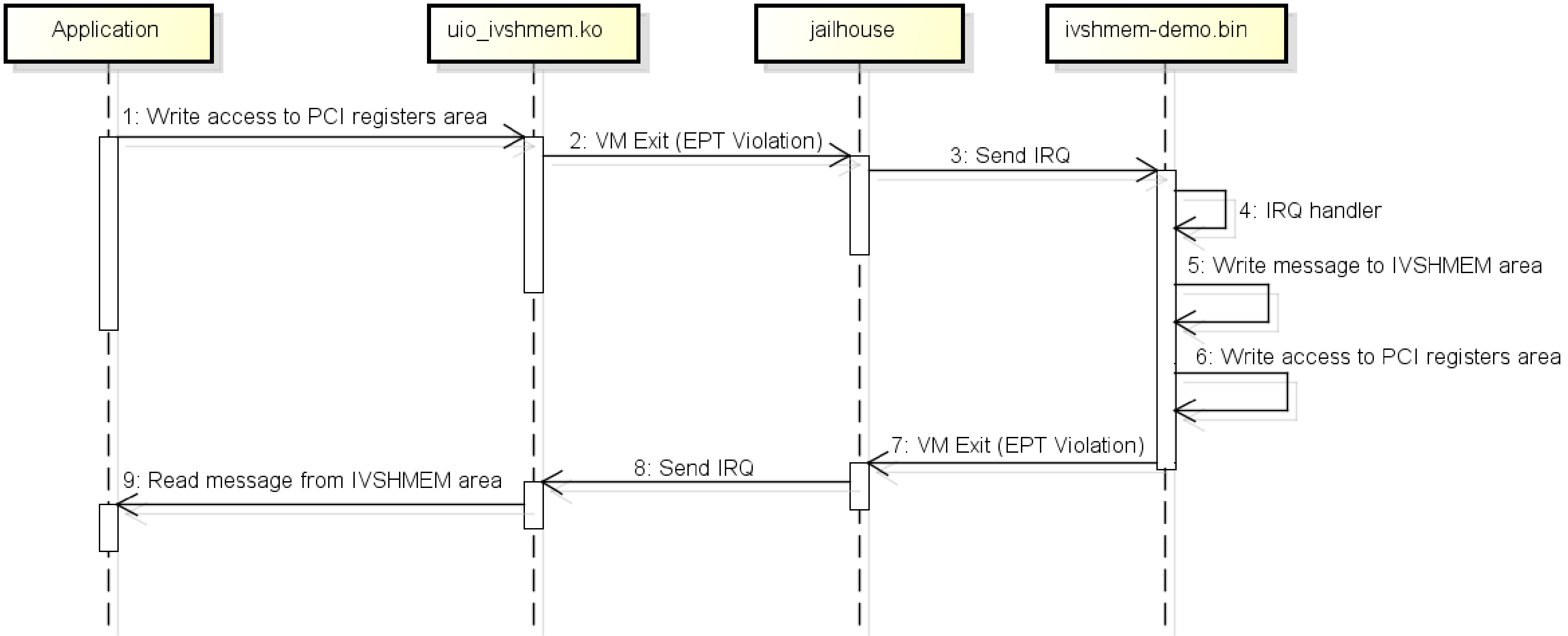
- **Prepare a test application for Linux**
  - To send IRQ to “imate”.

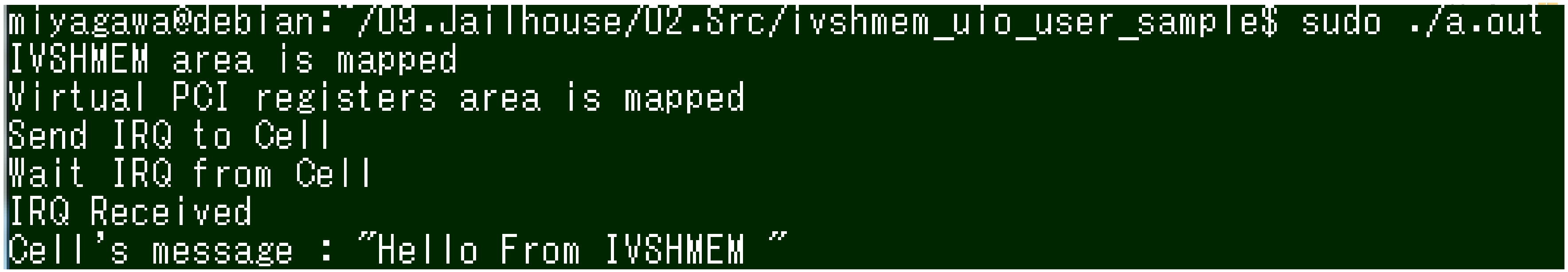
```
static void jh_ivshmem_mmio_write(void *addr, uint32_t value)
{
        asm volatile("movl %0,(%1)" : : "r" (value), "r" (addr));
}

void jh_ivshmem_send_irq(uint32_t *registers)
{
        jh_ivshmem_mmio_write((registers + 3), 1);
}
```

  - IVSHMEM area.
    - This area related to uio_info->mem[1].

Jailhouse

- **Communication between Linux and Cell**


Application
uio_ivshmem.ko
jailhouse
ivshmem-demo.bin
1: Write access to PCI registers area
2: VM Exit (EPT Violation)
3: Send IRQ
4: IRQ handler
5: Write message to IVSHMEM area
6: Write access to PCI registers area
7: VM Exit (EPT Violation)
8: Send IRQ
9: Read message from IVSHMEM area



```
miyagawa@debian:~/09.Jailhouse/02.Src/ivshmem_uio_user_sample$ sudo ./a.out
IVSHMEM area is mapped
Virtual PCI registers area is mapped
Send IRQ to Cell
Wait IRQ from Cell
IRQ Received
Cell’s message : “Hello From IVSHMEM ”
```

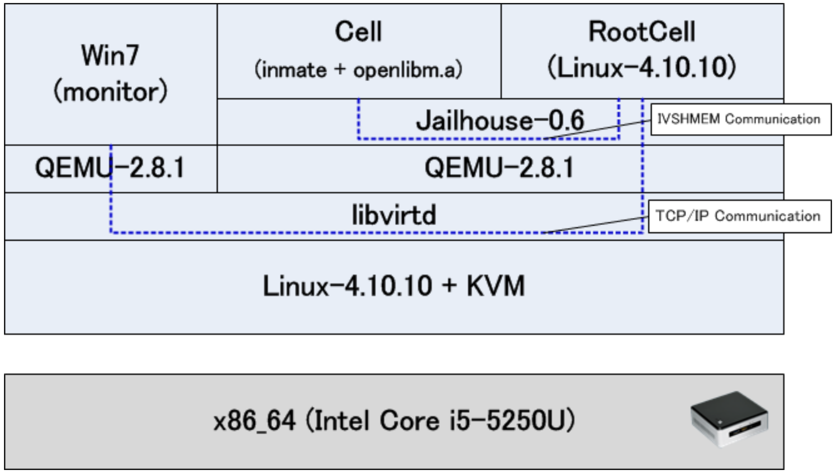
Agenda

- Civil Infrastructure System
- Jailhouse
  - Demonstration in QEMU/KVM
  - IVSHMEM
  - Applying Civil Infrastructure System
- Conclusion

Applying Civil Infrastructure System

- **Applying power plant control application.**
  - Traditionally, power plant control application uses libm.
  - In this demonstration, we use sin() function to generate sin wave.
  - openlibm is linked to “inmate”.
    - https://github.com/JuliaLang/openlibm

Configuration

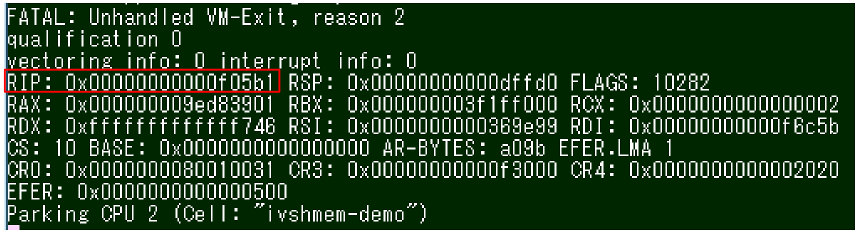


Makefile.lib

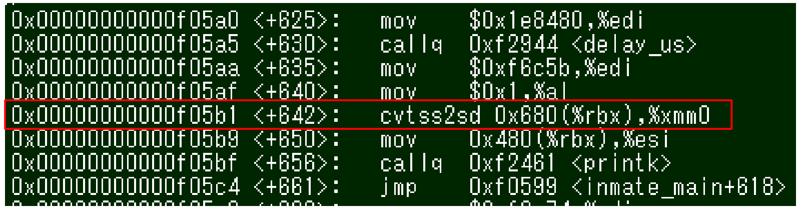
```
Index: inmates/lib/x86/Makefile.lib
===================================================================---
inmates/lib/x86/Makefile.lib        (リビジョン 1366)
+++ inmates/lib/x86/Makefile.lib        (作業コピー)
@@ -12,6 +12,7 @@

 KBUILD_CFLAGS += -m64 -mno-red-zone
 GCOV_PROFILE := n
+OPENLIBM := /home/miyagawa/09.Jailhouse/02.Src/openlibm-
master/libopenlibm.a

 define DECLARE_TARGETS =
  _TARGETS = $(1)
@@ -27,7 +28,7 @@
 # obj/NAME-linked.o: ... obj/$(NAME-y) lib/lib[32].a
 .SECONDEXPANSION:
 $(obj)/%-linked.o: $(INMATES_LIB)/inmate.lds $$(addprefix
$$(obj)/,$$($$*-y)) ¥
-                $(INMATES_LIB)/$$(if $$($$*_32),lib32.a,lib.a)
+                 $(INMATES_LIB)/$$(if $$($$*_32),lib32.a,lib.a) $(OPENLIBM)
       $(call if_changed,ld)

 $(obj)/%.bin: $(obj)/%-linked.o
```

Applying Civil Infrastructure System

- **TIPS**
  - When we tried this inmate, triple fault was invoked.



```
FATAL: Unhandled VM-Exit, reason 2
qualification 0
vectoring info: 0 interrupt info: 0
RIP: 0x00000000000f05b1 RSP: 0x00000000000dffd0 FLAGS: 10282
RAX: 0x000000009ed83901 RBX: 0x000000003f1ff000 RCX: 0x0000000000000002
RDX: 0xfffffffffffff746 RSI: 0x0000000000369e99 RDI: 0x00000000000f6c5b
CS: 10 BASE: 0x0000000000000000 AR-BYTES: a09b EFER.LMA 1
CR0: 0x0000000080010031 CR3: 0x00000000000f3000 CR4: 0x0000000000002020
EFER: 0x0000000000000500
Parking CPU 2 (Cell: "ivshmem-demo")
```



```
0x00000000000f05a0 <+625>:   mov    $0x1e8480,%edi
0x00000000000f05a5 <+630>:   callq  0xf2944 <delay_us>
0x00000000000f05aa <+635>:   mov    $0xf6c5b,%edi
0x00000000000f05af <+640>:   mov    $0x1,%al
0x00000000000f05b1 <+642>:   cvtss2sd 0x680(%rbx),%xmm0
0x00000000000f05b9 <+650>:   mov    0x480(%rbx),%esi
0x00000000000f05bf <+656>:   callq  0xf2461 <printk>
0x00000000000f05c4 <+661>:   jmp    0xf0599 <inmate_main+618>
```

  - To enable SSE (Streaming SIMD Extensions) instruction is needed.

```
Index: inmates/lib/x86/header.S
===================================================================
--- inmates/lib/x86/header.S    (リビジョン 1365)
+++ inmates/lib/x86/header.S    (作業コピー)
@@ -46,7 +46,15 @@

        .code32
 start32:
+       mov %cr0,%eax
+       and 0xFFFB,%ax
+       or  0x2,%ax
+       mov %eax,%cr0
        mov %cr4,%eax
+       or  3 << 9,%ax
+       mov %eax,%cr4
+
+       mov %cr4,%eax
        or $X86_CR4_PAE,%eax
        mov %eax,%cr4
```

Conclusion

- **Lessons Learned**
  - Jailhouse provides strict isolation.
  - IVSHMEM is easy to use.
  - Debugging “inmate” is difficult.
    - We hope useful tool would be provided.
- **Our Future Plan**
  - To watch development of Jailhouse.
  - Try to run Jailhouse on real hardware.
  - continue to learn Jailhouse.

TOSHIBA

Leading Innovation