Heart ticking for a guest running on FreeBSD ARM hypervisor

Mihai Carabas
mihai@freebsd.org

BSDCan 2019
University of Ottawa
Ottawa, Canada
May 17 – 18, 2019

About me

- University POLITEHNICA of Bucharest
  - PhD: virtualization on embedded devices
  - Lector: operating systems, systems architecture, networks

About me

- University POLITEHNICA of Bucharest
  - PhD: virtualization on embedded devices
  - Lector: operating systems, systems architecture, networks
- BSD world
  - DragonFly BSD: SMT aware scheduler - 2012, Intel EPT for vkernels - 2013
  - FreeBSD - bhyve: instruction caching - 2014, porting bhyve on ARM - 2015 (and present), coordinating save-restore and migration projects for bhyve

About me

- University POLITEHNICA of Bucharest
  - PhD: virtualization on embedded devices
  - Lector: operating systems, systems architecture, networks
- BSD world
  - DragonFly BSD: SMT aware scheduler - 2012, Intel EPT for vkernels - 2013
  - FreeBSD - bhyve: instruction caching - 2014, porting bhyve on ARM - 2015 (and present), coordinating save-restore and migration projects for bhyve
- Promoting bhyve through some diploma and master projects (e.g. ATA emulation, NE2000 emulation)
- Coordinating these diploma and master projects

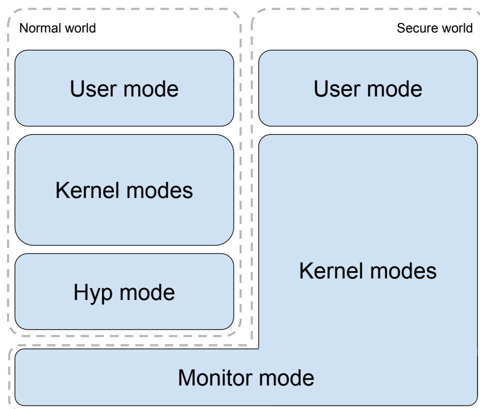
Hardware Assisted Virtualization

- A new CPU privilege level
  - On Intel/AMD: extends the current kernel mode (root/non-root)
  - On ARM (v7 and v8): a brand new level called Hyp-mode

FreeBSD

Hardware Assisted Virtualization

- A new CPU privilege level
  - On Intel/AMD: extends the current kernel mode (root/non-root)
  - On ARM (v7 and v8): a brand new level called Hyp-mode



FreeBSD

Type-2 Hypervisor on ARM

- Is more difficult to achieve

FreeBSD

Type-2 Hypervisor on ARM

- Is more difficult to achieve
- Have to rewrite significant parts of the base OS to use the new registers

Type-2 Hypervisor on ARM

- Is more difficult to achieve
- Have to rewrite significant parts of the base OS to use the new registers
- Even then you can't run userspace apps directly over it

Type-2 Hypervisor on ARM (2)

- We need to leverage the FreeBSD management mechanisms
- Don't want to write a full hypervisor from scratch

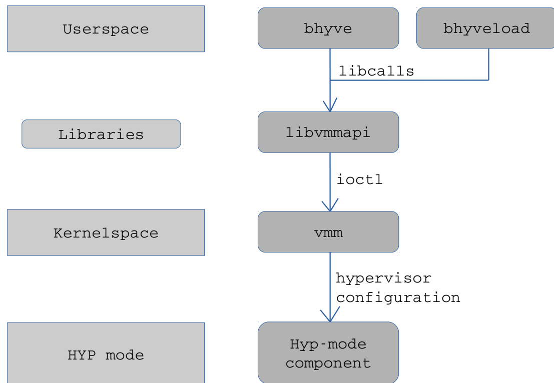
Type-2 Hypervisor on ARM (2)

- We need to leverage the FreeBSD management mechanisms
- Don't want to write a full hypervisor from scratch
- Insert only a small code into Hyp-mode
  - Bridge between the host OS and the hardware
  - It's called when doing hypervisor operations

Type-2 Hypervisor on ARM (2)

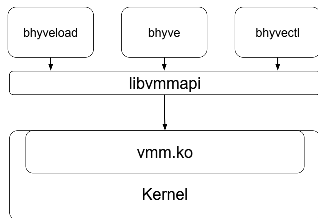
- We need to leverage the FreeBSD management mechanisms
- Don't want to write a full hypervisor from scratch
- Insert only a small code into Hyp-mode
  - Bridge between the host OS and the hardware
  - It's called when doing hypervisor operations
- Other type-2 implementation - KVM
  - VirtualOpenSystems did the same thing

bhyve components



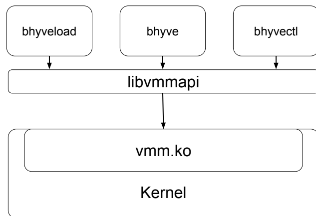
FreeBSD

From bhyve-x86 to bhyve-arm



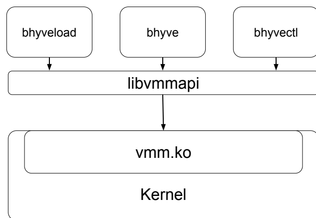
FreeBSD

From bhyve-x86 to bhyve-arm



- Created a new `sys/arm/vmm` by copying the VMM interface from `sys/amd64/vmm`

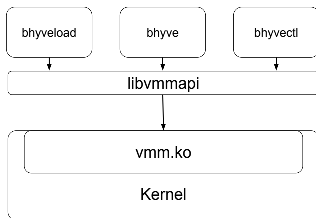
From bhyve-x86 to bhyve-arm



- Created a new `sys/arm/vmm` by copying the VMM interface from `sys/amd64/vmm`
  - **vmm-arm** - kernel module, manages Hyp state, context switching, guest physical memory and other aspects

FreeBSD

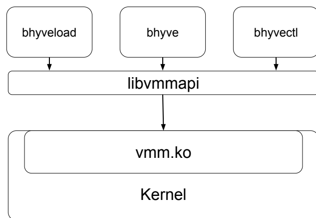
From bhyve-x86 to bhyve-arm



- Created a new `sys/arm/vmm` by copying the VMM interface from `sys/amd64/vmm`
  - **vmm-arm** - kernel module, manages Hyp state, context switching, guest physical memory and other aspects
  - **libvmmapiarm** - userland API for ARM

FreeBSD

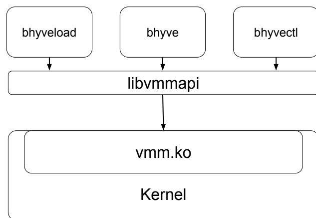
From bhyve-x86 to bhyve-arm



- Created a new `sys/arm/vmm` by copying the VMM interface from `sys/amd64/vmm`
  - **vmm-arm** - kernel module, manages Hyp state, context switching, guest physical memory and other aspects
  - **libvmmapiarm** - userland API for ARM
  - **bhyveloadarm** - userspace bootloader, creates vm and does initial setup, loads guest operating system into memory

FreeBSD

From bhyve-x86 to bhyve-arm



- Created a new `sys/arm/vmm` by copying the VMM interface from `sys/amd64/vmm`
  - **vmm-arm** - kernel module, manages Hyp state, context switching, guest physical memory and other aspects
  - **libvmmapiarm** - userland API for ARM
  - **bhyveloadarm** - userspace bootloader, creates vm and does initial setup, loads guest operating system into memory
  - **bhyvearm** - userspace run loop, emulates stdin/stdout

FreeBSD

Low-level Boot-up

- Crafted an init code placed in `locore`
  - It jumps to a routine where it checks if the platform booted in Hyp-mode
  - Install some stub exception vector for Hyp-mode
  - Marks the virtualization available

FreeBSD

Low-level Boot-up

- Crafted an init code placed in `locore`
  - It jumps to a routine where it checks if the platform booted in Hyp-mode
  - Install some stub exception vector for Hyp-mode
  - Marks the virtualization available
- Created some low-level routines for installing the exception vector for Hyp-mode
  - The most important entry is the `Hypervisor` one
  - It jumps there whenever `hvc` instruction is called or a VM raises an exception

FreeBSD

Low-level Boot-up

- Crafted an init code placed in `locore`
  - It jumps to a routine where it checks if the platform booted in Hyp-mode
  - Install some stub exception vector for Hyp-mode
  - Marks the virtualization available
- Created some low-level routines for installing the exception vector for Hyp-mode
  - The most important entry is the `Hypervisor` one
  - It jumps there whenever `hvc` instruction is called or a VM raises an exception
- Implement the low-level code which is doing context switching between the host OS and the VM
  - Save and restore the context (e.g. registers, co-proc registers)

FreeBSD

How the Host OS is Making Hypervisor Calls?

- Executes the `hvc` instruction
- First parameter indicates the address of a routine
- In Hyp-mode the code checks that the call came from the host OS

FreeBSD

Interrupt vector for Hypervisor mode

```
hyp_hvc:
 ...
 ...
```

Interrupt vector table

0x14	b hyp_hvc

Hypervisor

Kernel

FreeBSD

Low-level Interface for Type-2 Hypervisor

- `vmm_stub_install` - change the Hyp-mode exception vector with the empty one;

FreeBSD

Low-level Interface for Type-2 Hypervisor

- `vmm_stub_install` - change the Hyp-mode exception vector with the empty one;
- `hyp_init_hvc` - init the Hyp-mode environment.

Low-level Interface for Type-2 Hypervisor

- `vmm_stub_install` - change the Hyp-mode exception vector with the empty one;
- `hyp_init_hvc` - init the Hyp-mode environment.
- `vmm_call_hyp` - is called from kernel-mode to request a service from Hyp-Mode;

FreeBSD

Low-level Interface for Type-2 Hypervisor

- `vmm_stub_install` - change the Hyp-mode exception vector with the empty one;
- `hyp_init_hvc` - init the Hyp-mode environment.
- `vmm_call_hyp` - is called from kernel-mode to request a service from Hyp-Mode;
- `vmm_set_get_hvbar` - sets or gets a given exception vector;

FreeBSD

Low-level Interface for Type-2 Hypervisor

- `vmm_stub_install` - change the Hyp-mode exception vector with the empty one;
- `hyp_init_hvc` - init the Hyp-mode environment.
- `vmm_call_hyp` - is called from kernel-mode to request a service from Hyp-Mode;
- `vmm_set_get_hvbar` - sets or gets a given exception vector;
- `hyp_enter_guest` - make the context switch between the host and the next guest that needs to be run;

FreeBSD

Low-level Interface for Type-2 Hypervisor

- `vmm_stub_install` - change the Hyp-mode exception vector with the empty one;
- `hyp_init_hvc` - init the Hyp-mode environment.
- `vmm_call_hyp` - is called from kernel-mode to request a service from Hyp-Mode;
- `vmm_set_get_hvbar` - sets or gets a given exception vector;
- `hyp_enter_guest` - make the context switch between the host and the next guest that needs to be run;
- `hyp_exit_guest` - make the context switch from the guest to the host by restoring the host state;

FreeBSD

Memory Mapping

- Hyp-mode is basically another address space with its own mappings
- New translation level (Stage-2 translation) for VM isolation

FreeBSD

Memory Mapping

- Hyp-mode is basically another address space with its own mappings
- New translation level (Stage-2 translation) for VM isolation
- Issue: only LPAE is supported for both translations
- FreeBSD doesn't support LPAE and we cannot leverage on its memory management

LPAE Support

- Implement LPAE support in the VMM code
- Support for 40bit PA
- 3-level pagetables support (other formats are available but I've simplified the implementation)

LPAE Support

- Implement LPAE support in the VMM code
- Support for 40bit PA
- 3-level pagetables support (other formats are available but I've simplified the implementation)
- Issue: On 32-bit we don't have the DMAP mechanism (we need the virtual address of each entry to be able to write on it)

FreeBSD

LPAE Support

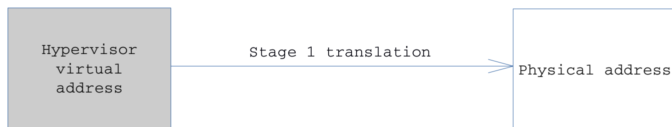
- Implement LPAE support in the VMM code
- Support for 40bit PA
- 3-level pagetables support (other formats are available but I've simplified the implementation)
- Issue: On 32-bit we don't have the DMAP mechanism (we need the virtual address of each entry to be able to write on it)
- Created a shadow pagetable for each level 1 and level 2 pagetables which have the VAs

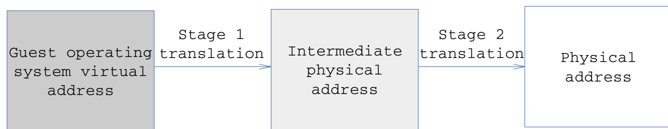
FreeBSD

Memory Mapping Considerations

- Mapped the hypervisor code at the same address in Hyp-mode and in host OS
- All the pointers passed between modes needs to be consistent
- The Hyp-mode works with the MMU enabled using a normal stage-1 translation using it's own pagetables

Type of translations





FreeBSD

Device Emulation

- Implement MMIO emulation using traps in a Stage-2 translation

FreeBSD

Device Emulation

- Implement MMIO emulation using traps in a Stage-2 translation
- Implement the paravirtualized serial console (bvm from amd64)

FreeBSD

Is the guest alive?

- The guest boots up asyncroniously

FreeBSD

Is the guest alive?

- The guest boots up asyncroniously
- In order to interract with it, one needs reponse to user input

FreeBSD

Is the guest alive?

- The guest boots up asyncroniously
- In order to interract with it, one needs reponse to user input
- The responses are coming though **interrupt controller**

FreeBSD

Is the guest alive?

- The guest boots up asyncroniously
- In order to interract with it, one needs reponse to user input
- The responses are coming though **interrupt controller**
- Also guest must respond to user input

FreeBSD

Is the guest alive?

- The guest boots up asyncroniously
- In order to interract with it, one needs reponse to user input
- The responses are coming though **interrupt controller**
- Also guest must respond to user input
- There has to be a mechanism that interrupts the guest from time to time to check if it has anything from the user

FreeBSD

Is the guest alive?

- The guest boots up asyncroniously
- In order to interract with it, one needs reponse to user input
- The responses are coming though **interrupt controller**
- Also guest must respond to user input
- There has to be a mechanism that interrupts the guest from time to time to check if it has anything from the user
- **Timer!**

FreeBSD

Is the guest alive?

- The guest boots up asyncroniously
- In order to interract with it, one needs reponse to user input
- The responses are coming though **interrupt controller**
- Also guest must respond to user input
- There has to be a mechanism that interrupts the guest from time to time to check if it has anything from the user
- **Timer!**
- This presentation covers the virtualization of
  - Interrupt controller
  - Timer

FreeBSD

Why do we need interrupts?

- We don't want to do polling
- We have a new *device* called interrupts controller
- The interrupt controller is a connector between CPU and any device which takes care of notifying

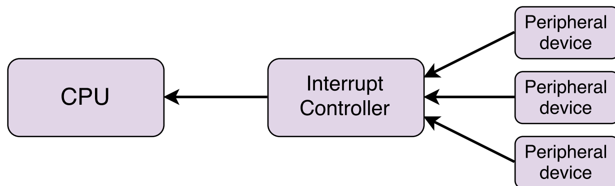


Figure: A CPU using interrupts

FreeBSD

Types of interrupt controllers

- Advanced Programmable Interrupt Controller (APIC) - on x86 (Intel and AMD)

FreeBSD

Types of interrupt controllers

- Advanced Programmable Interrupt Controller (APIC) - on x86 (Intel and AMD)
- Generic Interrupt Controller (GIC) - on ARM

FreeBSD

Generic Interrupt Controller

- Is a system present in ARM processors
- Centralizes interrupt support and management
- Provides a set of registers which enable management of interrupt sources and behavior
- May provide support for the following:
  - Security Extension
  - Virtualization Extension
  - Software Generated Interrupts
  - Managing and generating interrupts from hardware sources
  - Interrupt masking and prioritization
  - Uniprocessor and multiprocessor events

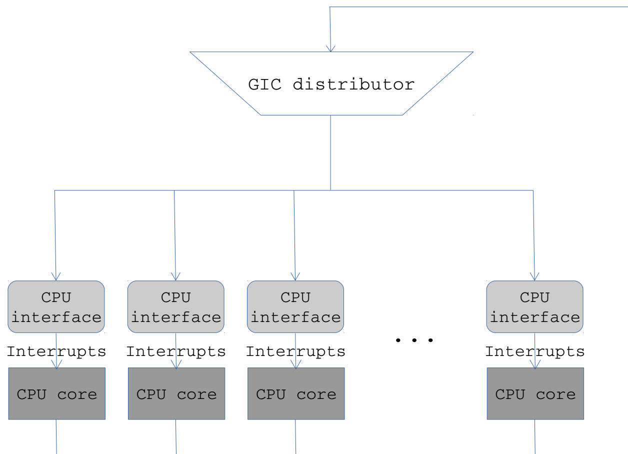
FreeBSD

Basic components of GIC

- **Distributor** - prioritizes and distributes interrupts to the CPU interfaces
- **CPU interfaces** - provides priority masking and preemption handling for processors

FreeBSD

GIC design



FreeBSD

GIC specific registers

- GICD_ISENABLERn / GICD_ICENABLERn - Interrupt Set/Clear-Enable Registers - enable/disable forwarding of the corresponding interrupt from the distributor to the CPU interfaces
- GICD_ISPENDRn / GICD_ICPENDRn - Interrupt Set/Clear-Pending Registers - sets/clears the pending state of respective interrupt
- GICD_ISACTIVERn / GICD_ICACTIVERn - Interrupt Set/Clear-Active Registers - activates/deactivates an interrupt; used when saving and restoring the GIC state
- GICC_IAR - Interrupt Acknowledge Register - contains the interrupt ID for the processor to read
- GICC_EOIR - End of Interrupt Register - signals the completion of handling an interrupt

FreeBSD

Interrupt lifecycle

- Transitions 1 and 2 - the interrupt becomes pending due to being generated by a peripheral or software

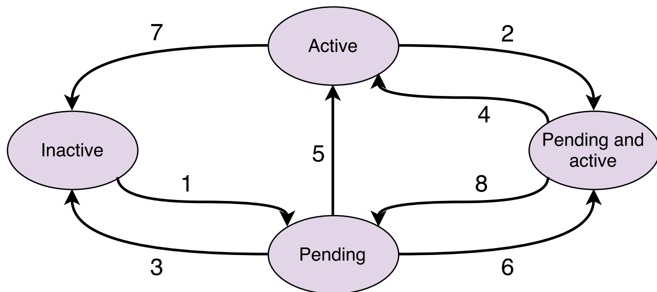


Figure: Interrupt Lifecycle

FreeBSD

Interrupt lifecycle(2)

- Transitions 3 and 4 - the pending state is removed either because the interrupt was deasserted, in case it is level triggered, or due to software modifying the state

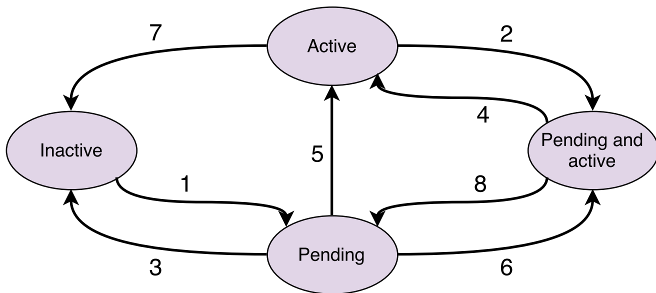


Figure: Interrupt Lifecycle

FreeBSD

Interrupt lifecycle(3)

- Transition 5 - applies to edge triggered interrupts upon acknowledgement

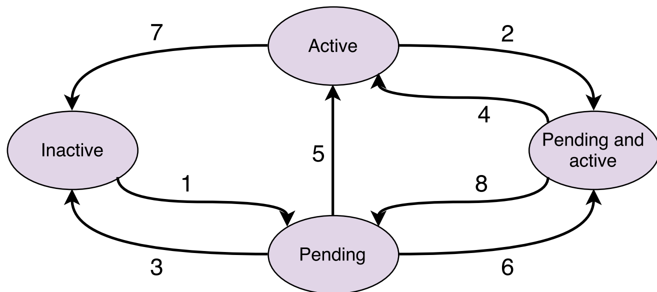


Figure: Interrupt Lifecycle

Interrupt lifecycle(4)

- Transition 6 - same as 5 for level triggered interrupts

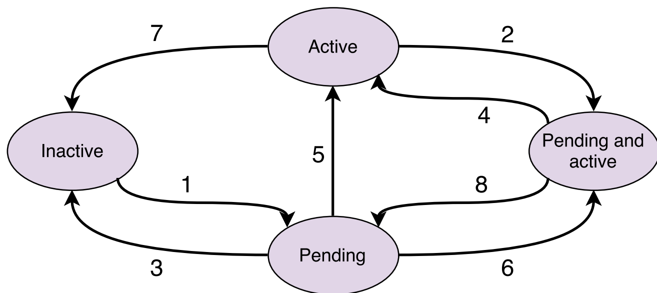


Figure: Interrupt Lifecycle

FreeBSD

Interrupt lifecycle(5)

- Transitions 7 and 8 - software deactivates the interrupt

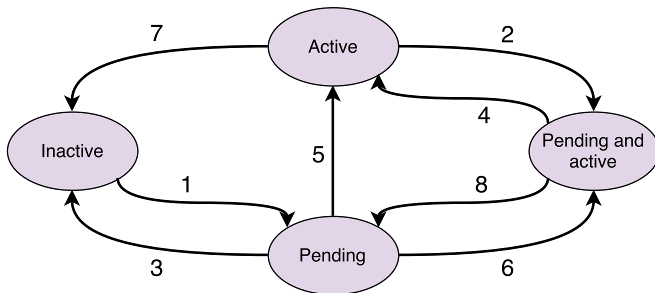


Figure: Interrupt Lifecycle

Interrupt Controller Virtualization

- 2 components: Distributor and CPU Interface

FreeBSD

Interrupt Controller Virtualization

- 2 components: Distributor and CPU Interface
- ARM provides a CPU Virtual Interface which can be used directly by the VM
  - Mapped CPU Interface over the CPU Virtual Interface
  - **Virtual CPU interfaces** - the virtual counterpart to the physical CPU interfaces, provide the same functionality for use in virtualized systems; also contain a control block through which it can be controlled

FreeBSD

Interrupt Controller Virtualization

- 2 components: Distributor and CPU Interface
- ARM provides a CPU Virtual Interface which can be used directly by the VM
  - Mapped CPU Interface over the CPU Virtual Interface
  - **Virtual CPU interfaces** - the virtual counterpart to the physical CPU interfaces, provide the same functionality for use in virtualized systems; also contain a control block through which it can be controlled
- One needs to emulate the accesses to the distributor

GIC virtualization specific registers

- GICV_IAR - Interrupt Acknowledge Register - contains the interrupt ID for the processor to read
- GICV_EOIR - End of Interrupt Register - signals the completion of handling an interrupt

FreeBSD

GIC virtualization specific registers

- GICV_IAR - Interrupt Acknowledge Register - contains the interrupt ID for the processor to read
- GICV_EOIR - End of Interrupt Register - signals the completion of handling an interrupt
- GICH_LRn - List Registers - contain interrupt context information to be used by the virtual CPU interfaces
- GICH_ELSRn - Empty List Register Status Registers - can be used to identify which List Registers are available to deliver an interrupt

FreeBSD

GIC Memory Mappings

- In order to enable interaction with vGIC similar to the normal GIC it is required to map the memory region used by the interrupt controller for memory-mapped IO in the address space of the guest
- `GICC_` on `GICV_`

FreeBSD

GIC Memory Mappings

- In order to enable interaction with vGIC similar to the normal GIC it is required to map the memory region used by the interrupt controller for memory-mapped IO in the address space of the guest
- `GICC_` on `GICV_`
- Additionally, access to this region must be trapped within the hypervisor to take appropriate action - (`GICD_`)

FreeBSD

Distributor Emulation

- Register the Distributor address range accesses for in-kernel emulation

Distributor Emulation

- Register the Distributor address range accesses for in-kernel emulation
- Created internal structure to retain the state of the distributor for each VM

FreeBSD

Distributor Emulation

- Register the Distributor address range accesses for in-kernel emulation
- Created internal structure to retain the state of the distributor for each VM
- Basically configs of each interrupt (e.g. irq_enable, irq_active, irq_state, irq_conf)

FreeBSD

Distributor Emulation

- Register the Distributor address range accesses for in-kernel emulation
- Created internal structure to retain the state of the distributor for each VM
- Basically configs of each interrupt (e.g. irq_enable, irq_active, irq_state, irq_conf)
- All reads and writes to Distributor registers are handled

FreeBSD

Distributor Emulation

- Register the Distributor address range accesses for in-kernel emulation
- Created internal structure to retain the state of the distributor for each VM
- Basically configs of each interrupt (e.g. irq_enable, irq_active, irq_state, irq_conf)
- All reads and writes to Distributor registers are handled
- Need to populate the LR register acordingly to the state of the distributor

FreeBSD

Distributor Emulation

- Register the Distributor address range accesses for in-kernel emulation
- Created internal structure to retain the state of the distributor for each VM
- Basically configs of each interrupt (e.g. irq_enable, irq_active, irq_state, irq_conf)
- All reads and writes to Distributor registers are handled
- Need to populate the LR register acordingly to the state of the distributor
- the LR register contains all the active interrupts that are signaled to the CPU Virtual Interface of the VM

Distributor Emulation

- Register the Distributor address range accesses for in-kernel emulation
- Created internal structure to retain the state of the distributor for each VM
- Basically configs of each interrupt (e.g. irq_enable, irq_active, irq_state, irq_conf)
- All reads and writes to Distributor registers are handled
- Need to populate the LR register acordingly to the state of the distributor
- the LR register contains all the active interrupts that are signaled to the CPU Virtual Interface of the VM
- To leverage existing interrupt controller logic, the internal state of the vGIC is flushed to hardware before switching to the guest and then synced back to software once control is handed over to the host.

FreeBSD

Generic Timer with Virtualization

- An implementation of the Generic Timer with Virtualization Extension provides four timers per CPU
  - Non-secure PL1 physical timer
  - Secure PL1 physical timer
  - Non-secure PL2 physical timer
  - Virtual timer

Virtual Generic Interrupt Controller

Name	Description
CNTV_CTL	Virtual Timer Control register. Used by the guest to interact with the timer hardware
CNTV_CVAL	Virtual Timer CompareValue register
CNTHCTL	Controls access to the physical registers. In particular, the PL1PCTEN and PL1PCEN are used to disable access to the physical timer registers
CNTVOFF	Virtual Offset register - specifies value to be subtracted from physical counter in order to obtain virtual counter

FreeBSD

Virtual Generic Interrupt Controller

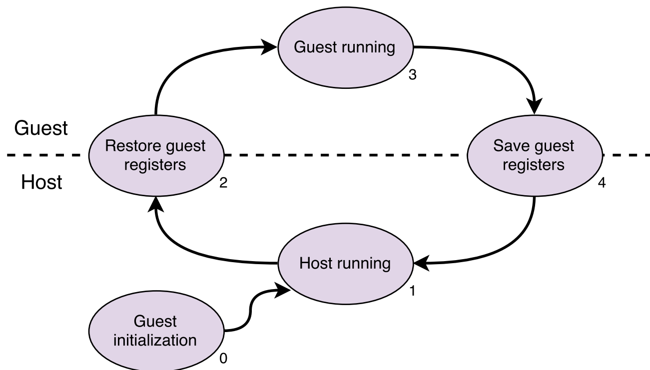


Figure: Virtual Timer Workflow

FreeBSD

Virtual Generic Interrupt Controller

- The workflow of the virtualization process is as follows:
  1. At guest initialization (state 0), the CNTVOFF register is initialized with the current value of the physical timer, rendering the virtual counter 0 for the newly created virtual machine
  2. Before entering the guest (transition from state 1 to state 2), the hypervisor internal state for the virtual timer is checked in order to determine whether any interrupts should have been triggered by the timer and need to be injected by the vGIC
  3. Upon entering the guest (state 2), the hypervisor enables the virtual timer if necessary, disables access to the physical timer, and restores the CNTVOFF register for the current virtual machine, as well as restoring CNTV_CVAL and CNTV_CTL

Virtual Generic Interrupt Controller (2)

4. While running, the guest accesses the virtual timer with no intervention from the hypervisor (state 3); any interrupts triggered here are sent to the vGIC, which will inject them accordingly
5. When exiting the guest, the CNTV_CVAL and CNTV_CTL registers are saved (state 4) and the hypervisor internal state is updated (transition from state 4 to state 1
6. The host continues to use the physical timer until the guest is run again, when the process resumes at step 2

FreeBSD

Results

- Currently, FreeBSD finishes the boot process while running as a guest in bhyve.

Results - listing

```
...
gic0: <ARM Generic Interrupt Controller> mem 0x2c001000-0x2c001fff,0x2c002000-0
      x2c003fff on ofwbus0
gic0: Cannot find Virtual Interface Control Registers. Disabling Hyp-Mode...
intr_pic_register(): PIC 0xc2207100 registered for gic0 <dev 0xc2633b80, xref 1>
intr_pic_claim_root(): irq root set to gic0
generic_timer0: <ARMv7 Generic Timer> irq 0,1,2,3 on ofwbus0
vgic_dist_mmio_write on cpu: 0 with gpa: 2c001100 size: 4 with val: 8000000
Timecounter "ARM MPCore Timecounter" frequency 24000000 Hz quality 1000
Event timer "ARM MPCore Eventtimer" frequency 24000000 Hz quality 1000
cpulist0: <Open Firmware CPU Group> on ofwbus0
cpu0: <Open Firmware CPU> on cpulist0
cryptosoft0: <software crypto>
NULL mp in getnewvnode(9), tag crossmp
Timecounters tick every 1.000 msec
WARNING: WITNESS option enabled, expect reduced performance.
WARNING: DIAGNOSTIC option enabled, expect reduced performance.
md0: Embedded image 18251776 bytes at 0xc0475f94
Trying to mount root from ufs:/dev/md0 []...
warning: no time-of-day clock registered, system time will not be set accurately
#
```

FreeBSD

Development platform for bhyve ARM

- FastModels from ARM emulating an CortexA15 (XX evaluation days, needs license from ARM)

Development platform for bhyve ARM

- FastModels from ARM emulating an CortexA15 (XX evaluation days, needs license from ARM)
- Running bhyve ARM on a real hardware platform
- Runnning bhyve ARM on Samsung Exynos 5250 and Cubie2 (All Winner A20)

FreeBSD

Running bhyve ARM on a real hardware platform

- The code base was very old from April 2015 and the boards weren't booting up

FreeBSD

Running bhyve ARM on a real hardware platform

- The code base was very old from April 2015 and the boards weren't booting up
- Did a rebase with the current HEAD and our development repo
  - Fix locore-v6.S integration of Andrew's LEAVE_HYP and our hypervisor stub install method
  - Fix the problems with vGIC introduced by INTRNG

FreeBSD

Running bhyve ARM on a real hardware platform

- The code base was very old from April 2015 and the boards weren't booting up
- Did a rebase with the current HEAD and our development repo
  - Fix locore-v6.S integration of Andrew's LEAVE_HYP and our hypervisor stub install method
  - Fix the problems with vGIC introduced by INTRNG
- Get the latest u-boot which left the board in Hyp-mode state

Running bhyve ARM on a real hardware platform

- The code base was very old from April 2015 and the boards weren't booting up
- Did a rebase with the current HEAD and our development repo
  - Fix locore-v6.S integration of Andrew's LEAVE_HYP and our hypervisor stub install method
  - Fix the problems with vGIC introduced by INTRNG
- Get the latest u-boot which left the board in Hyp-mode state
- `hvc` instruction is causing an *undefined instruction* exception

FreeBSD

`hvc` *undefined instruction*

- U-boot problems
  - U-boot loads Linux using `bootm` command, which switches to HYP-mode on entering the kernel
  - FreeBSD kernel is loaded by u-boot with the simpler go command
  - Patched the go flow for ARM in order to enter kernel in HYP-mode

FreeBSD

`hvc` *undefined instruction*

- U-boot problems
  - U-boot loads Linux using `bootm` command, which switches to HYP-mode on entering the kernel
  - FreeBSD kernel is loaded by u-boot with the simpler go command
  - Patched the go flow for ARM in order to enter kernel in HYP-mode
- Bugs in `locore-v6.S`
  - `ldr` pseudo-instruction was used to load addresses of various labels in registers
  - `ldr` requires address to be in the range of a page (4K) relative to the pc

FreeBSD

`hvc` *undefined instruction*

- U-boot problems
  - U-boot loads Linux using `bootm` command, which switches to HYP-mode on entering the kernel
  - FreeBSD kernel is loaded by u-boot with the simpler go command
  - Patched the go flow for ARM in order to enter kernel in HYP-mode
- Bugs in `locore-v6.S`
  - `ldr` pseudo-instruction was used to load addresses of various labels in registers
  - `ldr` requires address to be in the range of a page (4K) relative to the pc
  - Moved the label closer using a temporary symbol table

FreeBSD

`hvc` *undefined instruction*

- U-boot problems
  - U-boot loads Linux using `bootm` command, which switches to HYP-mode on entering the kernel
  - FreeBSD kernel is loaded by u-boot with the simpler go command
  - Patched the go flow for ARM in order to enter kernel in HYP-mode
- Bugs in `locore-v6.S`
  - `ldr` pseudo-instruction was used to load addresses of various labels in registers
  - `ldr` requires address to be in the range of a page (4K) relative to the pc
  - Moved the label closer using a temporary symbol table
  - The address loaded by `ldr` was not the label's address, but the one stored at the label's location, so we replaced `ldr` with `adr`

FreeBSD

Encoutered Issues

- The host interrupts were not disabled before entering guest execution

FreeBSD

Encoutered Issues

- The host interrupts were not disabled before entering guest execution
- Consequently, these would arrive while the guest was running and were identified as spurious.

FreeBSD

Encoutered Issues

- The host interrupts were not disabled before entering guest execution
- Consequently, these would arrive while the guest was running and were identified as spurious.
- The cause was mistakenly assumed to be the incomplete implementation of the vGIC logic.

FreeBSD

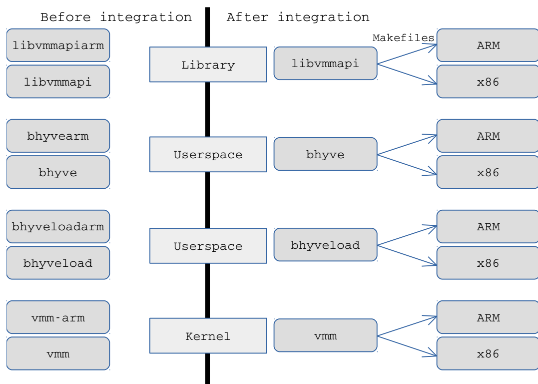
Encoutered Issues

- The host interrupts were not disabled before entering guest execution
- Consequently, these would arrive while the guest was running and were identified as spurious.
- The cause was mistakenly assumed to be the incomplete implementation of the vGIC logic.
- Differences between the emulator and the hardware paltform: some steps were not required for the emulator to behave correctly

FreeBSD

Next Steps

- Merge the code...
- Merge bhyvearm code https://reviews.freebsd.org/D10213 - prereq. MD/MI for bhyve



FreeBSD

Next Steps (2)

- Test on more hardware platforms (only Cubie2 at this moment)
- SMP support in VMM ARM module

FreeBSD

Conclusions

- ARM is offering us support to create a performant virtualized interrupt controller

FreeBSD

Conclusions

- ARM is offering us support to create a performant virtualized interrupt controller
- Pretty tedious to emulate each operation of the distributor and debug the timer

FreeBSD

Conclusions

- ARM is offering us support to create a performant virtualized interrupt controller
- Pretty tedious to emulate each operation of the distributor and debug the timer
- Hard to debug on hardware platforms

Thank you for your attention!
ask questions