**Table of Contents**

[**Steps to Create and Test Viomb device and driver on x86**](#_f2pyj7676xxu) **3**

[**Installing QEMU**](#_9qq2ikhpuhst) **3**

[**GDB Qemu**](#_4igxkqb7n7u3) **4**

[**GDB OpenBSD**](#_z6lo4ry4xx2d) **5**

[**virsh**](#_rfphu2a7vs7) **6**

[**Switch to Old Qemu(virt Manager) from Custom Qemu**](#_fdmg0cuckftm) **7**

[**Switching from Virt MAnager to customQemu**](#_uj9rem1iejc5) **7**

[**Enable SSH into outer VM**](#_hpoqh5sy4rcq) **7**

[**SSH into outer VM**](#_oqrik9wc0nxt) **9**

[**Creating “Outer”**](#_x7cc6uj8mhoq) **9**

[**Driver/Host Changes**](#_oa3ak0w7uf1j) **10**

[**Device Changes and Running the new VM**](#_orzh5o4b3jp2) **12**

[**GDB Debugging**](#_5fhl6bomb8cc) **12**

[**See Attached Devices to PCI Bus**](#_rj76yd81ul0n) **12**

[**Inner VM commands**](#_15030j94annd) **12**

[**Outer VM commands**](#_axcrofx6jltc) **12**

[**Start VMD in debugging Mode**](#_kkjfcuhuc9dr) **12**

[**Github**](#_hp6tiuh5hxq5) **13**

[**Inner VM Error**](#_pdxiu9th9qfg) **13**

[**Make changes and compile vmctl (usr.sbin->vmctl folder):**](#_fjq6yhxqa22g) **13**

[**GRUB Error**](#_hh7eoj5kuiyb) **13**

[**Steps to test Virtio viomb device and driver stats queue functionality on arm64 device**](#_exlbmqejviv9) **16**

[**Step 1: Download pre-reqs**](#_dov52g7cyy4m) **16**

[**Step 2: Create guest VM**](#_xls8yipnd2wq) **16**

[**Step 3: Boot guest VM**](#_p0pji9yc9ilz) **17**

[**Step 4: Optimization**](#_42df3530epv0) **17**

[**Step 5: Attach Viomb device**](#_lqx0mjxw3sob) **18**

[**Step 6: Analyze logs in guest VM**](#_j9egppaq8n8o) **18**

[**Step 7: Debug from Qemu**](#_849c02ragcoe) **19**

[**ERRORS AND SOLUTIONS:**](#_bkoceufovasn) **20**

[**How to resolve RKDRM error?**](#_ucxurkj8jzed) **20**

# 

# 

# Steps to Create and Test Viomb device and driver on x86

## **Installing QEMU**

<https://en.wikibooks.org/wiki/QEMU/QEMU_Hello_World:_Installing_QEMU_and_getting_it_up_and_running>

* 1. **git clone https://github.com/ruchikahazariwal/qemu.git**
  2. **cd qemu**
  3. **git checkout master**
  4. **git submodule init**
  5. **sudo git submodule update --recursive**
  6. **git submodule status --recursive**
  7. **sudo apt install make -- if make is not installed before**
  8. **sudo make clean**
  9. **mkdir build**
  10. **cd build**
  11. **sudo apt-get update**
  12. (<https://github.com/Cisco-Talos/pyrebox/issues/41>)

**sudo apt-get install build-essential zlib1g-dev pkg-config libglib2.0-dev binutils-dev libboost-all-dev autoconf libtool libssl-dev libpixman-1-dev libpython-dev python-pip python-capstone virtualenv libsdl2-dev libusbredirhost-dev flex bison libspice-server-dev libspice-protocol-dev**

* 1. (from within build folder)

**(original version)**

**../configure --enable-spice --enable-kvm --enable-usb-redir --target-list=x86\_64-softmmu --enable-sdl**

**(only if you want to do debugging)**

**../configure --enable-spice --enable-kvm --enable-usb-redir --target-list=x86\_64-softmmu --enable-sdl --enable-debug --extra-cflags="-g3"**

* 1. **(0. AFTER GIT PULL) change directory in build folderVIRTIO\_BALLOON\_F\_MUST\_TELL\_HOST**
  2. **(1. AFTER GIT PULL) sudo make -j 8**
  3. **(2. AFTER GIT PULL) sudo make install**
  4. **\*(JUST ONCE, if it’s already there, skip this step)**

**sudo mv /usr/bin/qemu-system-x86\_64 /usr/bin/qemu-system-x86\_64.old**

* 1. **(3. AFTER GIT PULL) sudo rm /usr/bin/qemu-system-x86\_64**
  2. **(4. AFTER GIT PULL)**

**sudo ln -s /usr/local/bin/qemu-system-x86\_64 /usr/bin/qemu-system-x86\_64**

* 1. **(5. AFTER GIT PULL) sudo chmod 755 /usr/bin/qemu-system-x86\_64**
  2. (**ONCE**)

**sudo ln -s /etc/apparmor.d/usr.sbin.libvirtd /etc/apparmor.d/disable/usr.sbin.libvirtd**

* 1. (**ONCE**)

**sudo /etc/init.d/apparmor restart**

* 1. **(REBOOT YOUR LAPTOP)**
  2. **(JUST ONCE) qemu-img create -f qcow2 xx 5G**
  3. **Create VM without virtmanager**

1. **(basic) sudo qemu-system-x86\_64 -enable-kvm -hda xx -sdl**
2. **(Internet access) sudo qemu-system-x86\_64 -smp 4 -enable-kvm -hda /var/lib/libvirt/images/openbsd.qcow2 -sdl -device virtio-balloon-pci -net nic,model=rtl8139 -net user**
3. **(Ruchika)** 
   1. **sudo /usr/local/bin/qemu-system-x86\_64 -smp 4 -enable-kvm -hda /var/lib/libvirt/images/openbsd.qcow2 -sdl -device virtio-balloon-pci -net nic,model=rtl8139 -net user -serial mon:stdio -s -qmp tcp:localhost:4444,server,nowait**
   2. **(Run in different linux terminal) telnet localhost 4444**
4. **(David’s OpenBSD)**
   1. **sudo qemu-system-x86\_64 -enable-kvm -hda /var/lib/libvirt/images/outer.qcow2 -sdl -device virtio-balloon-pci**
5. **(David)** 
   1. **sudo /usr/local/bin/qemu-system-x86\_64 -smp 8 -enable-kvm -hda /var/lib/libvirt/images/outer.qcow2 -sdl -device virtio-balloon-pci -net nic,model=rtl8139 -net user -serial mon:stdio -qmp tcp:localhost:4444,server,nowait**
   2. **(Run in different linux terminal) telnet localhost 4444**
6. **(David’s centOS) sudo qemu-system-x86\_64 -smp 8 -enable-kvm -hda /var/lib/libvirt/images/centos7.0.qcow2 -sdl -device virtio-balloon-pci -net nic,model=rtl8139 -net user**
7. **(Hema/Anjala) (Internet access) sudo qemu-system-x86\_64 -smp 8 -enable-kvm -hda /var/lib/libvirt/images/generic.qcow2 -sdl -device virtio-balloon-pci -net nic,model=rtl8139 -net user**

## **GDB Qemu**

(worked in my system. Still doing troubleshooting though.)

Steps to follow:

1. **sudo gdb**
2. Inside gdb run: **file "/usr/local/bin/qemu-system-x86\_64"**
3. **break virtio\_balloon\_receive\_stats**
4. (Ruchika) r -enable-kvm -hda /var/lib/libvirt/images/openbsd.qcow2 -sdl -device virtio-balloon-pci -net nic,model=rtl8139 -net user -serial mon:stdio -qmp tcp:localhost:4444,server,nowait

(David) **r -enable-kvm -hda /var/lib/libvirt/images/centos7.0.qcow2 -sdl -device virtio-balloon-pci -net nic,model=rtl8139 -net user -serial mon:stdio -qmp tcp:localhost:4444,server,nowait**

1. Run **“handle all nostop”**. Keep pressing Enter until back to the (gdb) terminal
2. In a different linux terminal - **telnet localhost 4444, Follow steps of 3.Virsh 3. polling**
3. Keep pressing **next** or **continue** infinite times until qemu starts. Custom printf will start appearing that means you are on the right path.
4. Keep checking the stack by running: **bt**

## **GDB OpenBSD**

* 1. Qemu try to find qemu executable in /usr/bin. If it is not there, it considers qemu exe file from /usr/local/bin
  2. Cd /usr/local/bin
  3. Build the custom openbsd kernel via virt-manager
  4. Copy the bsd gdb kernel from the outer VM to your local host (linux desktop for example)

**scp root@192.168.122.41(your ip):/usr/obj/openbsd/sys/arch/amd64/compile/GENERIC.MP/obj/bsd.gdb ~/Documents/**

* 1. Now stop the virt-manager
  2. Start the qemu from terminal with command **sudo /usr/local/bin/qemu-system-x86\_64 -smp 4 -enable-kvm -hda /var/lib/libvirt/images/openbsd.qcow2 -sdl -device virtio-balloon-pci -net nic,model=rtl8139 -net user -serial mon:stdio -s -qmp tcp:localhost:4444,server,nowait**
  3. **NOTE**: -s will start gdb stub on localhost:1234 ==-gdb tcp::1234
  4. **sudo gdb**
  5. **file “~/Documents/bsd.gdb”**
  6. **target remote :1234**
  7. **break viomb\_attach**
  8. **until viomb\_attach**
  9. **handle all nostop**
  10. next/continue

The flow of the code:

hw/virtio/virtio-balloon.c: Virtio\_balloon\_receive\_stats

hw/virtio/virtio.c: Virtio\_queue\_notify

hw/virtio/virtio-pci: Virtio\_pci\_notify\_write

To run debugging in openbsd:

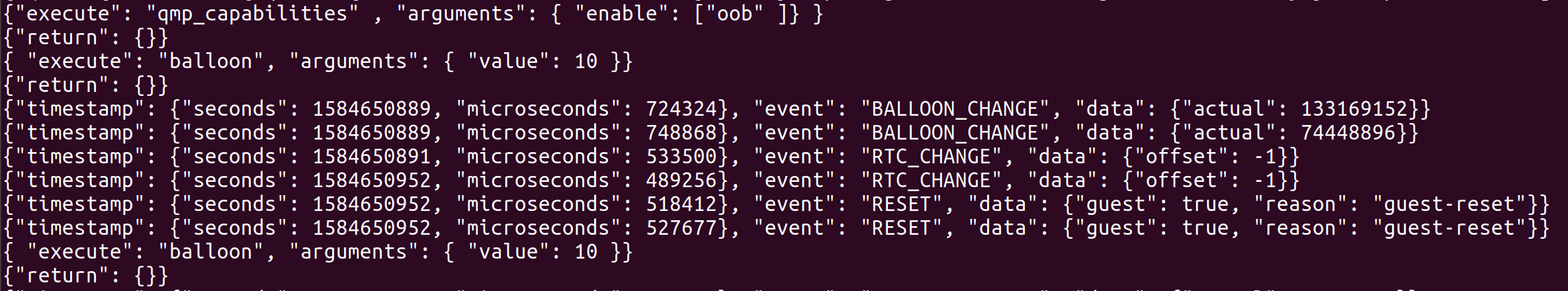
gdb obj/bsd.gdb

Linux was able to call virtio\_balloon\_receive\_stats because while attaching driver to the qemu balloon device, it updated the stats queue and sent the interrupt to the device. Openbsd is not doing that, it is still waiting for the interrupt from the device.

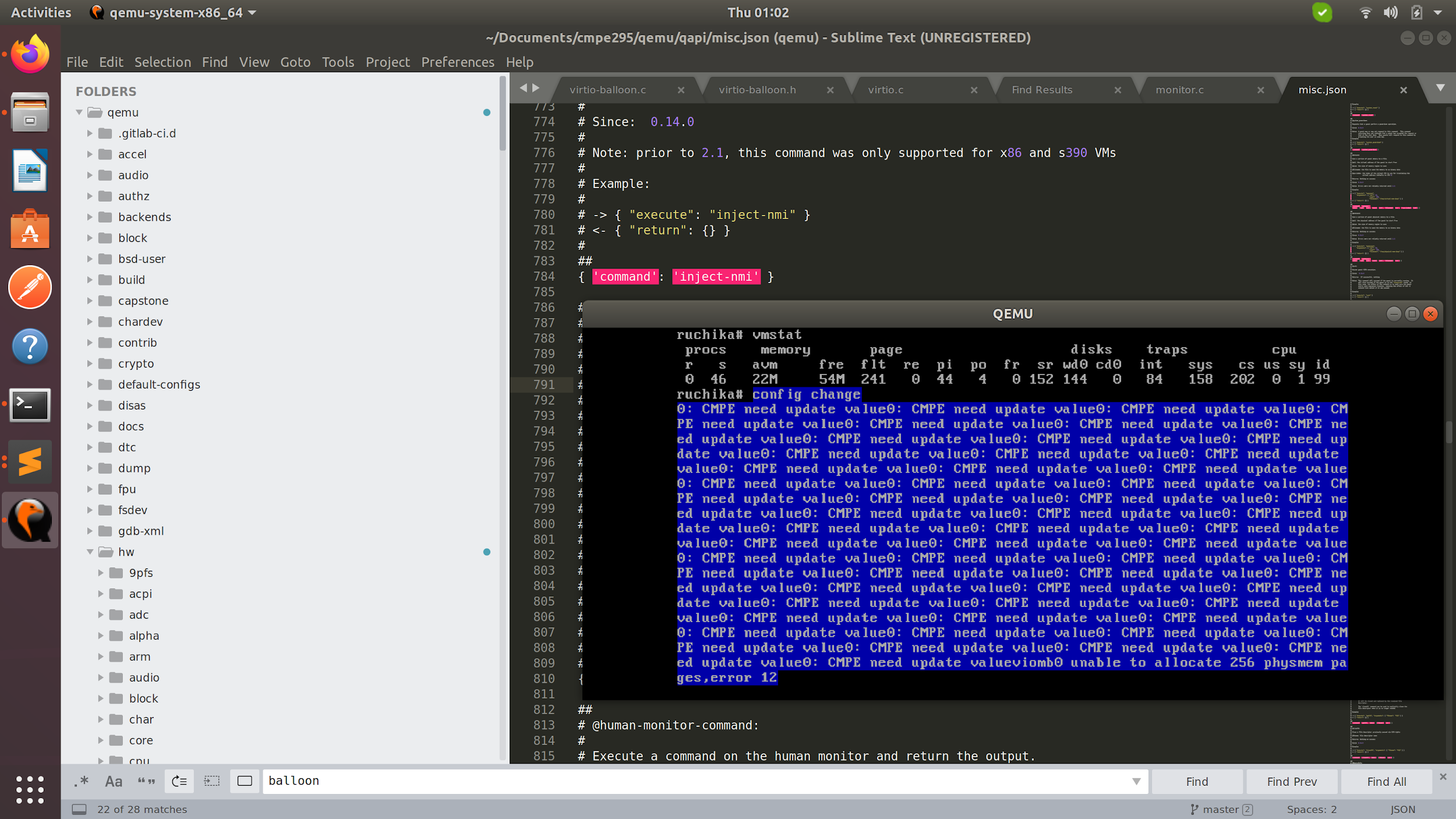
Two ways to solve this problem. Can you suggest which option is better for now:

1. Either we can also update the stats while attaching the qemu device with the openbsd driver.
2. We need to send the interrupt from the qemu device with the help of monitor command “guest-stats-polling-interval”, so that openbsd starts updating the stats queue.

We were able to cause interrupt to openbsd driver for inflating the balloon



This interrupt is calling the viomb\_config\_change function. After this, viomb\_worker got activated.



How can we send the interrupt for the stats queue and call the viomb\_config\_change function.

## virsh

Note: 18446744073709551615 == -1 or FFFFFFF

Virsh

1. To view the guest stats

qemu-monitor-command openbsd '{ "execute": "qom-get", "arguments": { "path": "/machine/peripheral/balloon0", "property": "guest-stats"} }'

1. To set the polling from guests

qemu-monitor-command openbsd '{ "execute": "qom-set", "arguments": { "path": "/machine/peripheral/balloon0", "property": "guest-stats-polling-interval", "value": 10} }'

{"return":{},"id":"libvirt-42"}

1. To connect with monitor console via telnet:

**telnet localhost 4444**

MUST be run FIRST : {"execute": "qmp\_capabilities" , "arguments": { "enable": ["oob" ]} }

{ "execute": "balloon", "arguments": { "value": 10 }}

{ "execute": "query-balloon", "arguments": { "value": 10} }

{ "execute": "human-monitor-command", "arguments": { "command-line": "info pci" } }

{ "execute": "human-monitor-command", "arguments": { "command-line": "info qtree" } }

{ "execute": "query-pci" }

{ "execute": "qom-get", "arguments": { "path": "/machine/unattached/device[0]", "property": "guest-stats"} }

{ "execute": "**qom-list-properties**", "arguments": { "typename": "balloon" }

qemu-monitor-command openbsd --hmp --cmd "qom-get"

qemu-monitor-command openbsd --hmp --cmd "info pci" -- for running the device

## **Switch to Old Qemu(virt Manager) from Custom Qemu**

1. Assuming that the CURRENT “qemu-system-x86\_64” is the CUSTOM version. (The custom is ~50MB)
2. Save CURRENT as the updated CUSTOM:

**sudo cp qemu-system-x86\_64 qemu-system-x86\_64.custom**

1. Overwrite CURRENT with OLD:

**sudo cp qemu-system-x86\_64.old qemu-system-x86\_64**

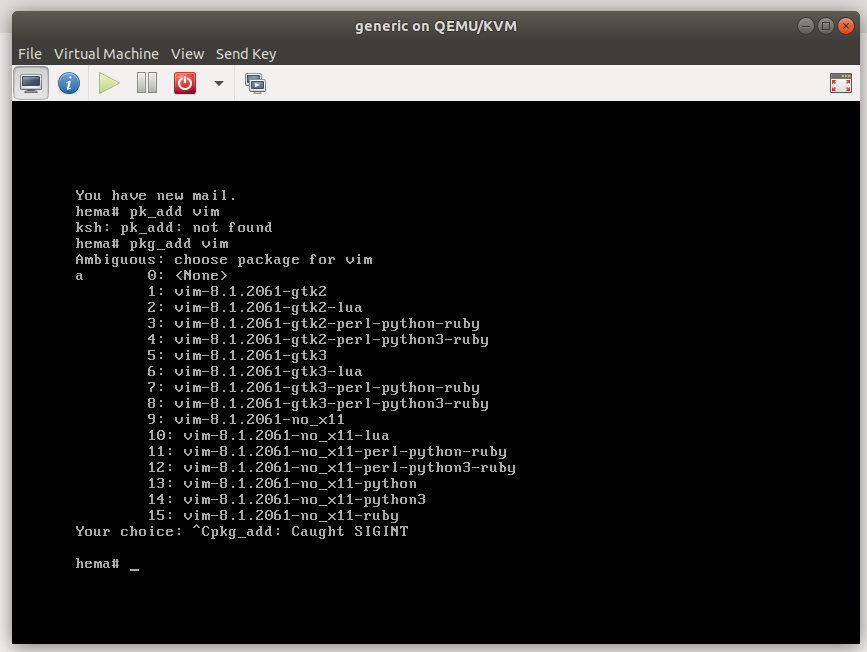
## **Switching from Virt MAnager to customQemu**

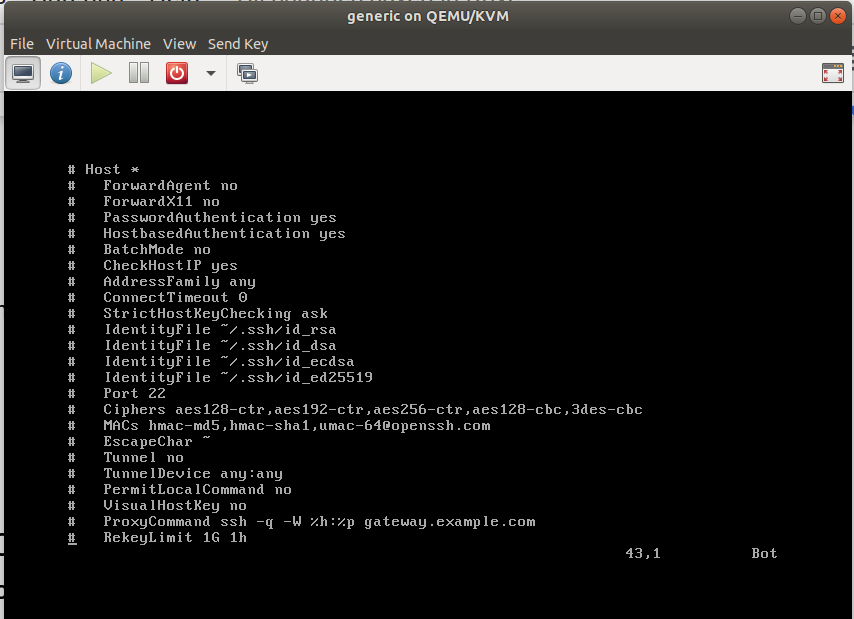
* 1. Assuming that the CURRENT “qemu-system-x86\_64” is the OLD version. (The old is ~11MB)
  2. Overwrite CURRENT with CUSTOM:

**sudo cp qemu-system-x86\_64.custom qemu-system-x86\_64**

## **Enable SSH into outer VM**

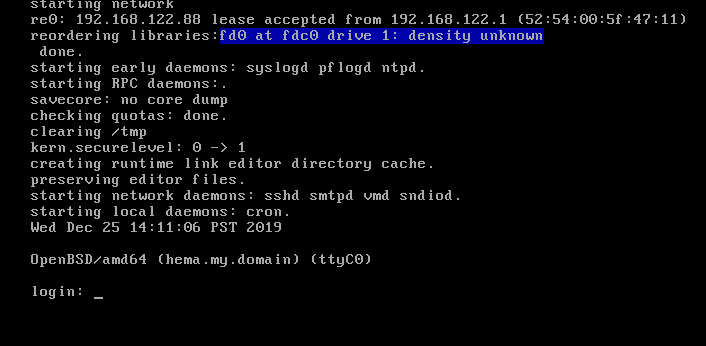
1. Start Outer VM
2. Command to install vim: pkg\_add vim
3. Select 6 as in the given options



1. Command: vim /etc/ssh/ssh\_config
2. Select “i” to insert
3. “ Host based authentication” should be yes as below
4. Select etc->:wq! To exit the file with saving the changes
5. Go to the terminal in linux
6. Type the following command: ssh root@128.162.122.88(hema-192.168.122.88)

Note: How to know the ip address of my vm?

1. Command: reboot in your outer vm
2. You should see reo ip address lease accepted from…… as shown below



## **SSH into outer VM**

* 1. **ssh root@192.168.122.14**

## **Creating “Outer”**

* 1. **Install virt-manager**
     1. **sudo apt-get install qemu-kvm**
     2. **sudo apt-get install qemu-system**
     3. **sudo apt-get install virt-manager**
  2. **Launch virt-manager**
     1. **sudo virt-manager**
  3. **Create “Outer” VM host**
     1. Use the virt manager’s GUI to create this VM
     2. The settings can be as follows:
        1. RAM 13000
        2. CPU 8
        3. Disk image disk size 33
        4. Directory does not contain SHA256 - yes
        5. Set “copy host CPU configuration” checkbox. (copies the properties from Linux)
     3. Restart VM
  4. **Download the modified OpenBSD code to the outer VM**
     1. Run the “outer” vm
     2. **pkg\_add git**
     3. **cd /usr/obj/**
     4. **git clone** [**https://github.com/ruchikahazariwal/openbsd.git**](https://github.com/ruchikahazariwal/openbsd.git)

or **git clone git://github.com/ruchikahazariwal/openbsd.git**

* 1. **Create object files**
     1. From within “outer” vm
     2. **cd openbsd/sys/arch/amd64/compile/GENERIC.MP/**
     3. **(only do “once in a while...” )make clean**
     4. **make obj**
     5. **make config**
     6. **make -j #** //# number of cpu’s assigned to the VM, use the max amount possible, like 8 if 8 are available from the KVM gui configuration
  2. **Use the new Object files to install OpenBSD** 
     1. From within “outer” vm
     2. **cp /usr/obj/openbsd/sys/arch/amd64/compile/GENERIC.MP/obj/bsd /bsd0**
     3. **reboot** // or Reboot using the GUI
     4. enter **bsd0** when it reboots
  3. **Allocate the space for the inner VM**
     1. **mkdir usr/obj/xx**
     2. **vmctl create -s 5g /usr/obj/xx/inner\_vm.qcow2**

**If BIOS\_FIRMWARE error, try: “fw\_update”**

1. **(ONLY FIRST TIME) (inner VM)Download iso image for inner VM**

**Cd to /usr/obj**

**ftp** [**https://cdn.openbsd.org/pub/OpenBSD/snapshots/amd64/install66.iso**](https://cdn.openbsd.org/pub/OpenBSD/snapshots/amd64/install66.iso)

**OR download install67.iso from google drive**

**Or**

**scp install67.iso root@192.168.122.74:/usr/obj/xx**

## **Driver/Host Changes**

* 1. Push changes
  2. (outer) Pull changes
  3. (outer) Rebuild outer’s kernel
     1. **cd /usr/obj/openbsd/sys/arch/amd64/compile/GENERIC.MP/**
     2. **make clean**
     3. **make obj**
     4. **make config**
     5. **make -j #** //# number of cpu’s assigned to the VM, use the max amount possible, like 8 if 8 are available from the KVM gui configuration

Note: the step c is required only if you change the device related code else you can skip step c.

* + 1. **cd obj**
    2. **cp bsd /bsd0**
    3. **reboot**
    4. **bsd0** //when restarting
  1. (outer) Start/Reset VMD
     1. **cd YOUR\_OPENBSD/usr.sbin/vmd**
     2. **git pull**
     3. **make clean**
     4. **make -j 8**
     5. **make install**
     6. **pkill -9 vmd**
     7. **rcctl enable vmd**
     8. **rcctl start vmd**
        1. **rcctl -d start vmd**
     9. **rcctl check vmd**
  2. (outer) Start Inner (if already inner running, skip)
     1. **(First time?)**

**vmctl start -c -r /usr/obj/install67.iso -d /usr/obj/xx/inner\_vm.qcow2 -L test**

1. (I)nstall **I**
2. Terminal type? **Enter**
3. System hostname? **“Inner”**
4. Network? [vio0] **Enter**
5. IPv4 [dhcp] **Enter**
6. IPv6 [none] **Enter**
7. Which network interface? [done] **Enter**
8. DNS [my.domain] **Enter**
9. DNS namesevers? [none] **Enter**
10. Password for root **david**
11. Start sshd [yes] **Enter**
12. Change console [yes] **Enter**
13. Speed [115200] **Enter**
14. User? [no] **Enter**
15. Allow root ssh login? [no] **YES**
16. Which disk is the root disk? [sd0] **Enter**
17. Use MBR [whole] **Enter**
18. Use (A)uto layout [a] **Enter**
19. Location of sets? [cd0] **Enter**
20. Pathname to the sets? [6.6/amd64] **Enter**
21. Set name(s)? [done] **Enter**
22. Directory does not contain SHA256.sig. [no] **YES**
23. Location of sets? [cd0] **Enter**
24. Pathname to the sets? [6.6/amd64] **Enter**
25. Set name(s) [done] **Enter**
26. Directory does not contain [no] **YES**
    * 1. **(After first time -for local network)**

**vmctl start -c -d /usr/obj/xx/inner\_vm.qcow2 -L test**

* 1. (inner) Exit inner
     1. Leave inner vm running.Use ~. (both simultaneously) to exit the inner vm without stopping it.
  2. (outer) Copy outer’s new kernel to inner
     1. **cd /usr/obj/openbsd/sys/arch/amd64/compile/GENERIC.MP/obj**
     2. **scp bsd root@100.64.1.3:~/**
  3. (outer) Go inside inner
     1. **Command: vmctl console test**
  4. (inner) Creating a new pointer for inner to reference new kernel
     1. **cd ~/**
     2. **//wait until inner vm is initialized**
     3. **cp bsd /bsd0**
  5. (inner) Restart inner
     1. **MUST Reboot after this**
     2. **After rebooting to start the inner vm with new code given bsd0 after rebooting**

## **Device Changes and Running the new VM**

* 1. Push changes
  2. (outer) Pull changes
  3. (outer) start/reset vmd
     1. **cd /usr/obj/openbsd/usr.sbin/vmd**
     2. **make clean**
     3. **make**
     4. **make install**
     5. **pkill -9 vmd**
     6. **rcctl enable vmd**
     7. **rcctl start vmd**
        1. **rcctl -d start vmd**
     8. **rcctl check vmd**
  4. (outer)Run inner

**vmctl start -c -r /usr/obj/install66.iso -d /usr/obj/xx/inner\_vm.qcow2 -L test**

(In the above command: install66.iso is in usr/obj folder and new inner vm space is allocated in the usr/obj/xx/ folder)

## **GDB Debugging**

1. **pkg\_add GDB**
2. **gdb -g test.c**
3. **gdb ./a.out**
4. **B 10 //line number of breakpoint**
5. **Run //runs program**
6. **S //**

## **See Attached Devices to PCI Bus**

1. **Pcidump**
   1. **Dmesg** // this command will give information that shows drivers and devices.

## **Inner VM commands**

1. **~.** // tilde dot

## **Outer VM commands**

1. **vmctl status**
2. **vmctl console <name>**
3. **vmd -dvvv**

## **Start VMD in debugging Mode**

1. **rcctl -d start vmd**
2. **dmesg | egrep '(VMX/EPT|SVM/RVI)'**

**Note : to see all the vmd processes running the command is “ps ax | grep vmd”**

**Note: to see all the printfs the command is “vmd -dvvvv” this will kill the existing vm and create a new vmd instance named “ vmd -dvvvvv”**

## **Github**

1. Delete Branch

I. Switch to another branch(other than master)

**ii. git branch -d <branch\_name>**

**iii. git push origin --delete <branch\_name>**

1. Delete a directory?

I. If empty: rm directory\_name

ii. If the directory is not empty: rm -r directory\_name

## **Inner VM Error**

**“/dev/sd0d (3518cd3e4f4b06ca.d): UNEXPECTED INCONSISTENCY; RUN fsck\_ffs MANUALLY.”**

1. **fsck\_ffs sd0d**
2. **Say all “yeses”**
3. **Reboot**

**Note: Command to remove a directory - “rm -r directoryname”**

## **Make changes and compile vmctl (usr.sbin->vmctl folder):**

1. make

2. make install

## **GRUB Error**

1. Go to command prompt as administrator
2. command: diskpart
3. list disk
4. select disk 0
5. list partition
6. select partition 2
7. assign letter=x
8. exit
9. x:
10. dir
11. cd efi
12. dir
13. rd ubuntu /s
14. Y
15. exit
16. **Update the kernel of init when bogus syscall occurs**
    1. Login with normal bsd.
    2. Run “sysupgrade”. It will download the folder “\_sysupgrade” in your home directory.
    3. After installation, it will reboot itself (**DON’T WRITE ANYTHING IN BOOT OPTION**) and it will automatically call “upgraded kernel”.
    4. Reboot again when it is done and select your custom bsd this time.
17. **If we change anything in vmmvar.h**
18. su
19. find the ~YOUR\_BSD/sys/arch/amd64/include/vmmvar.h that you changed, and cp that to /usr/include/machine

**cp ~YOUR\_BSD/sys/arch/amd64/include/vmmvar.h /usr/include/machine**

1. exit (exit from su)

vmm.c (the kernel code) picks up "arch/amd64/include/vmmvar.h" but vmd.c picks up /usr/include/machine/vmmvar.h

1. Now you can build VMD
2. Compile after changing uvmexp:

Run below commands in each folder

* make
* make install

sysctl - > sbin/sysctl

systat -> **cp ~YOUR\_BSD/sys/uvm/uvmexp.h /usr/include/uvm**

cd **~YOUR\_BSD/**usr.bin/systat

vmstat - > usr.bin/vmstat

top - > usr.bin/top

vmctl -> usr.sbin/vmctl

1. If it gives known\_hosts error: host already exists

First run “rm /root/.ssh/known\_hosts/”

# Steps to test Virtio viomb device and driver stats queue functionality on arm64 device

Note: Viomb device is also called as memory ballooning

## **Step 1: Download pre-reqs**

Download

1. miniroot68.img

2. QEMU\_EFI.fd

from<https://0x16h.github.io/OpenBSD_arm64_qemu.html>

## **Step 2: Create guest VM**

In first terminal execute the following command:

Command 1: apt update && install qemu virt-manager

Now create a root.qcow2 file using command 2

Command 2:

vmctl create -s 10g root.qcow2

Create a go.sh file, this file is used to launch qemu on linux

Command 3: vi go.sh

Note: go.sh should contain the following script:

#!/bin/sh

qemu-system-aarch64 -m 2048 -M virt -cpu cortex-a57 -bios QEMU\_EFI.fd -device virtio-rng-device -drive file=miniroot68.img,format=raw,id=drive1 -net user -drive file=root.qcow2,if=none,id=drive0,format=qcow2 -device virtio-blk-device,drive=drive0 -nographic -serial tcp::4450,server,telnet,wait -netdev user,id=net0 -device virtio-net-device,netdev=net0

save the file (esc + :wq!)

Give permission to go.sh file

Command 4: chmod +x go.sh

Execute the file

Command 5: ./go.sh

## **Step 3: Boot guest VM**

In second terminal

Command 6: telnet localhost 4450

Other than the below questions just press enter whenever prompted for an answer

(I)nstall, (U)pgrade, (A)utoinstall or (S)hell? I

System hostname? (short form, e.g. 'foo') hema

Setup a user? (enter a lower-case loginname, or 'no') [no] no

DNS domain name? (e.g. 'example.com') [my.domain] local

Password for root account?

Allow root ssh login? (yes, no, prohibit-password) [no] yes

Use (W)hole disk or (E)dit the MBR? [whole] W

Use (A)uto layout, (E)dit auto layout, or create (C)ustom layout? [a] A

HTTP Server? (hostname, list#, 'done' or '?') cdn.openbsd.org

Note: To take a break, we can stop the environment by ctrl+C in the first terminal.

To again start the project:

In first terminal execute the command 5 (ie., running the go.sh file if you are doing for the first time otherwise you can run godel.sh file)

## **Step 4: Optimization**

Stop the first terminal by (ctrl+C) and execute the following commands:

Command 7: cp go.sh godel.sh

Replace the text in godel.sh by following:

#!/bin/sh

qemu-system-aarch64 -m 2048 -M virt -cpu cortex-a57 -bios QEMU\_EFI.fd -device virtio-rng-device -net user -drive file=root.qcow2,if=none,id=drive0,format=qcow2 -device virtio-blk-device,drive=drive0 -nographic -serial tcp::4450,server,telnet,wait -netdev user,id=net0 -device virtio-net-device,netdev=net0

Command 8: chmod +x godel.sh

Command 9: ./godel.sh

## **Step 5: Attach Viomb device**

Now we need to tell Qemu to attach the virtio balloon device in the .sh script so let’s append the below

script at the end of the godel.sh script after a space.

-device virtio-balloon-pci -net nic,model=rtl8139

For convenience I create a new file and called as godel-balloon.sh and gave the permission usng chmod +x similar to command 4

The content of godel-balloon.sh file is below:

#!/bin/sh

qemu-system-aarch64 -m 2048 -M virt -cpu cortex-a57 -bios QEMU\_EFI.fd -device virtio-rng-device -net user -drive file=root.qcow2,if=none,id=drive0,format=qcow2 -device virtio-blk-device,drive=drive0 -nographic -serial tcp::4450,server,telnet,wait -netdev user,id=net0 -device virtio-net-device,netdev=net0 -device virtio-balloon-pci -net nic,model=rtl8139

## **Step 6: Analyze logs in guest VM**

Run qemu using godel-balloon.sh in the first terminal, forget about go.sh and godel.sh.

In the second terminal execute command 6. While booting give bsd0 since we compiled the our code into bsd0.

Logs related to balloon is attached give in the viomb\_attach functions, and stats queue is initiated and empty req is placed by the driver with all the 6 tags are set to 0x0, all these logs will be visible while booting. So now we successfully initialized an empty stats queue in guest openbsd running on qemu which emulate the arm64 hardware.

## **Step 7: Debug from Qemu**

Let’s verify the logic of obtaining the guest statistics using stats queue from device side. For this the device has to make a request.

stats\_poll interval needs to be more than zero to put a request in stats queue hence this need to be set. Here we are doing it manually i.e., giving a hard coded value to test.

Code: on linux, clone qemu from<https://github.com/Hemaprasanthi/qemu/commits/master> modify the file hw/virtio/virtio-balloon.c in location static void virtio\_balloon\_device\_realize(DeviceState \*dev, Error \*\*errp)

add the following code at the end of the function.

s->stats\_timer = timer\_new\_ms(QEMU\_CLOCK\_VIRTUAL, balloon\_stats\_poll\_cb, s);

s->stats\_poll\_interval = 10;

/\* XXX \*/

reset\_stats(s);

Before compiling, we need add a target to create qemu-system-x86 object, because qemu emulated the arm64 hardware using that object. By default it creates x86.

Command 10: cd qemu (this qemu is not the one inside openbsd folder)

if it is first time run commands from 11 to 18 else skip

Command 11: git submodule init

Command 12: sudo git submodule update –recursive

Command 13: git submodule status –recursive

Command 14: sudo apt install make -- if make is not installed before

Command 15: sudo make clean

Command 16: mkdir build

Command 17: cd build

Command 18: sudo apt-get update

1. (<https://github.com/Cisco-Talos/pyrebox/issues/41>)

**sudo apt-get install build-essential zlib1g-dev pkg-config libglib2.0-dev binutils-dev libboost-all-dev autoconf libtool libssl-dev libpixman-1-dev libpython-dev python-pip python-capstone virtualenv libsdl2-dev libusbredirhost-dev flex bison libspice-server-dev libspice-protocol-dev**

Command 19:

../configure --enable-spice --enable-kvm --enable-usb-redir --target-list=aarch64-softmmu --enable-sdl --enable-debug –extra-cflags="-g3"

Command 20: sudo make config

Command 21: sudo make -j8

Command 22: sudo make install

Command 23: sudo mv /usr/bin/qemu-system-aarch64 /usr/bin/qemu-system-aarch64-old

Command 24: sudo rm /usr/bin/qemu-system-aarch64

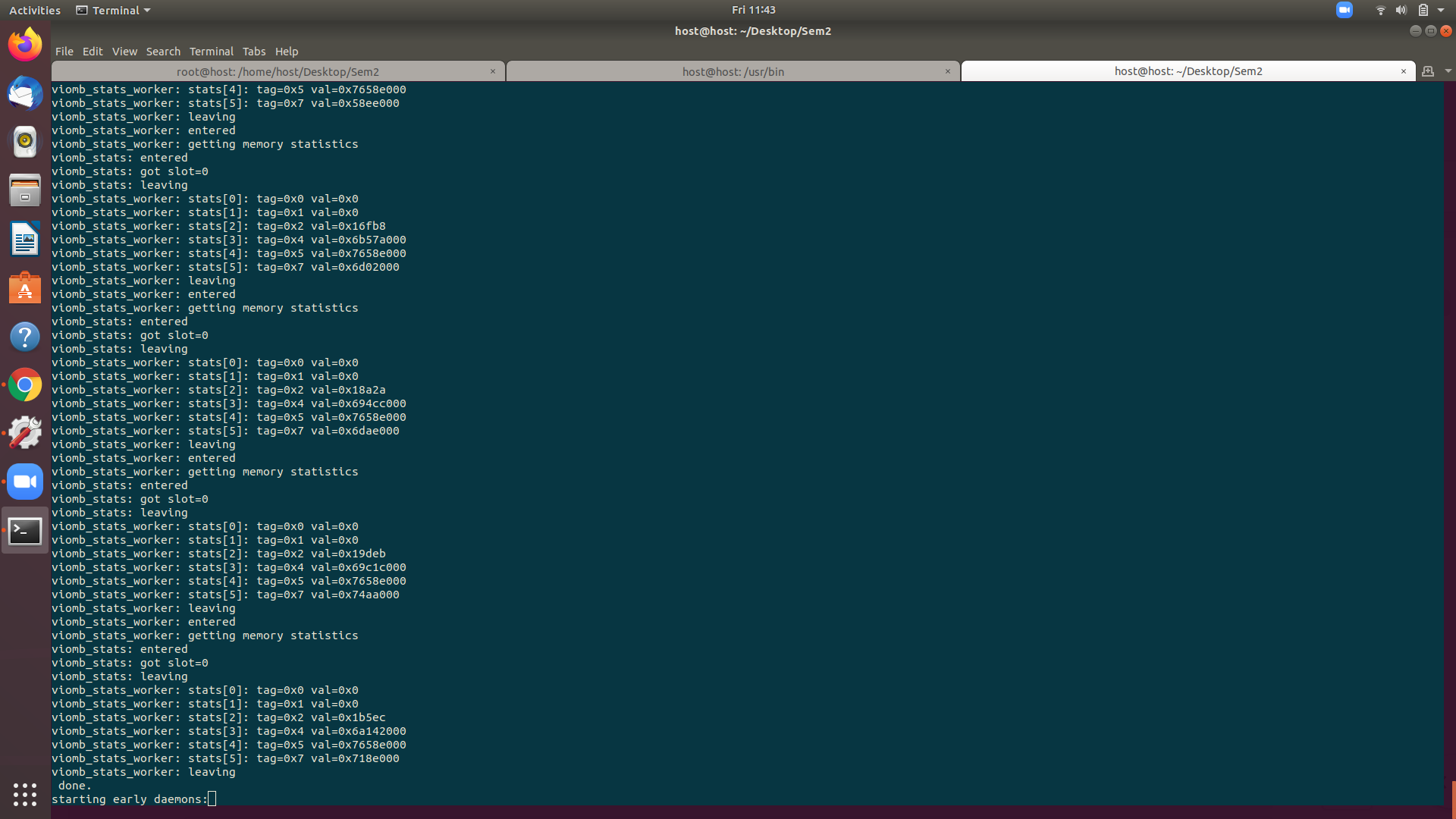
Command 25: sudo ln -s /usr/local/bin/qemu-system-aarch64 /usr/bin/qemu-system-aarch64

Command 26: sudo chmod 755 /usr/bin/qemu-system-aarch64

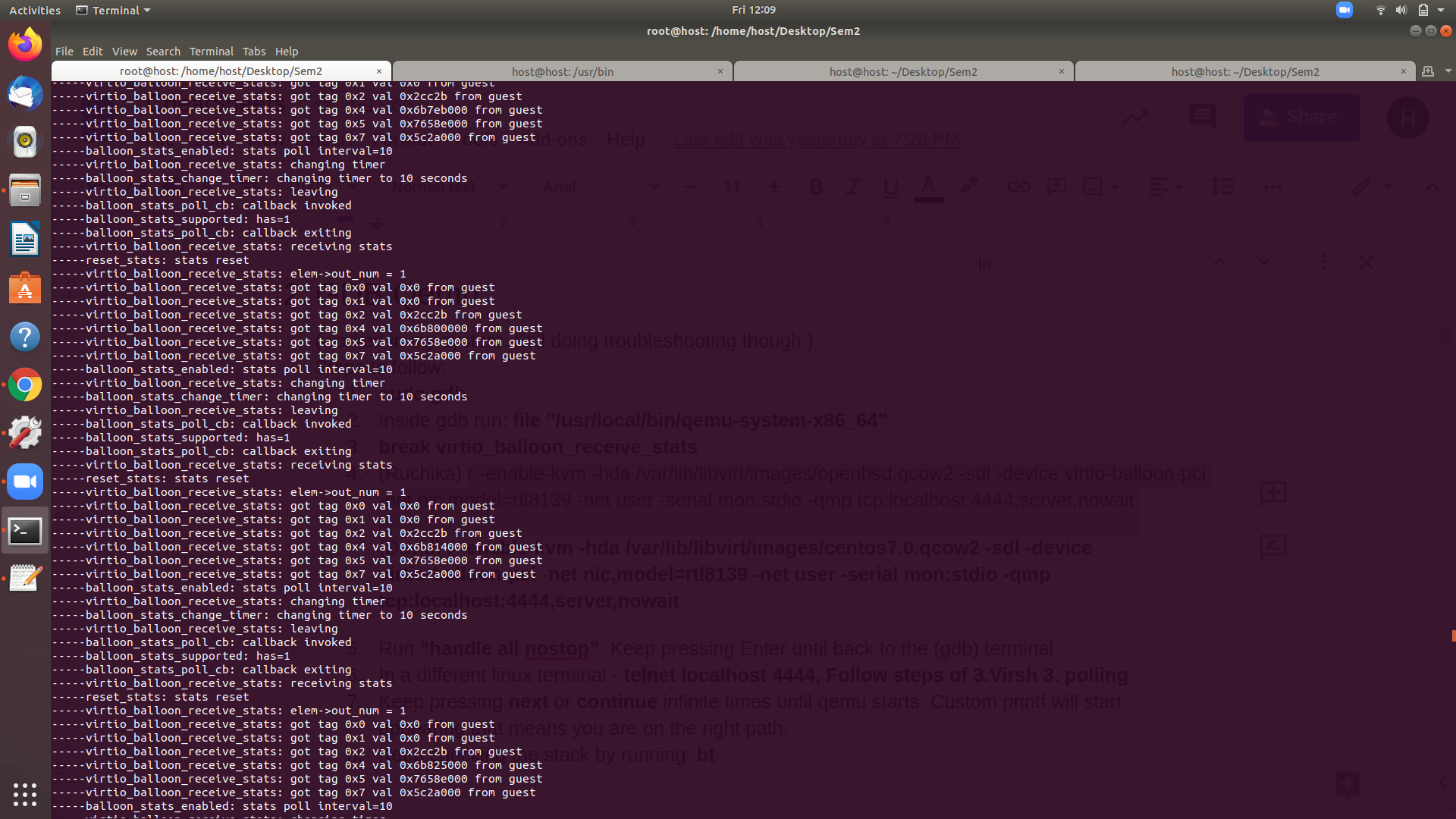
Command 27: ./godel-balloon.sh

Execute command 6 in the second terminal

Guest VM(openbsd) logs about stats queue

****

Host(Qemu) logs about stats queue



# ERRORS AND SOLUTIONS:

## **How to resolve RKDRM error?**

Error:

o '\_\_asm(".section .rodata,\"a\"");' > gapdummy.c

cc -g -Werror -Wall -Wimplicit-function-declaration -Wno-uninitialized -Wno-pointer-sign -Wframe-larger-than=2047 -Wno-address-of-packed-member -Wno-constant-conversion -mcmodel=kernel -mno-red-zone -mno-sse2 -mno-sse -mno-3dnow -mno-mmx -msoft-float -fno-omit-frame-pointer -ffreestanding -fno-pie -msave-args -O2 -pipe -nostdinc -I/home/openbsd/sys -I/home/openbsd/sys/arch/amd64/compile/GENERIC.MP/obj -I/home/openbsd/sys/arch -I/home/openbsd/sys/dev/pci/drm/include -I/home/openbsd/sys/dev/pci/drm/include/uapi -I/home/openbsd/sys/dev/pci/drm/amd/include/asic\_reg -I/home/openbsd/sys/dev/pci/drm/amd/include -I/home/openbsd/sys/dev/pci/drm/amd/amdgpu -I/home/openbsd/sys/dev/pci/drm/amd/display -I/home/openbsd/sys/dev/pci/drm/amd/display/include -I/home/openbsd/sys/dev/pci/drm/amd/display/dc -I/home/openbsd/sys/dev/pci/drm/amd/display/amdgpu\_dm -I/home/openbsd/sys/dev/pci/drm/amd/powerplay/inc -I/home/openbsd/sys/dev/pci/drm/amd/powerplay/smumgr -I/home/openbsd/sys/dev/pci/drm/amd/powerplay/hwmgr -I/home/openbsd/sys/dev/pci/drm/amd/display/dc/inc -I/home/openbsd/sys/dev/pci/drm/amd/display/dc/inc/hw -I/home/openbsd/sys/dev/pci/drm/amd/display/dc/clk\_mgr -I/home/openbsd/sys/dev/pci/drm/amd/display/modules/inc -I/home/openbsd/sys/dev/pci/drm/amd/display/modules/hdcp -I/home/openbsd/sys/dev/pci/drm/amd/display/dmub/inc -I/home/openbsd/sys/dev/pci/drm/i915 -DDDB -DDIAGNOSTIC -DKTRACE -DACCOUNTING -DKMEMSTATS -DPTRACE -DPOOL\_DEBUG -DCRYPTO -DSYSVMSG -DSYSVSEM -DSYSVSHM -DUVM\_SWAP\_ENCRYPT -DFFS -DFFS2 -DFFS\_SOFTUPDATES -DUFS\_DIRHASH -DQUOTA -DEXT2FS -DMFS -DNFSCLIENT -DNFSSERVER -DCD9660 -DUDF -DMSDOSFS -DFIFO -DFUSE -DSOCKET\_SPLICE -DTCP\_ECN -DTCP\_SIGNATURE -DINET6 -DIPSEC -DPPP\_BSDCOMP -DPPP\_DEFLATE -DPIPEX -DMROUTING -DMPLS -DBOOT\_CONFIG -DUSER\_PCICONF -DAPERTURE -DMTRR -DNTFS -DHIBERNATE -DPCIVERBOSE -DUSBVERBOSE -DWSDISPLAY\_COMPAT\_USL -DWSDISPLAY\_COMPAT\_RAWKBD -DWSDISPLAY\_DEFAULTSCREENS="6" -DX86EMU -DONEWIREVERBOSE -DMULTIPROCESSOR -DMAXUSERS=80 -D\_KERNEL -MD -MP -c /home/openbsd/sys/conf/swapgeneric.c

cc -c -g -Werror -Wall -Wimplicit-function-declaration -Wno-uninitialized -Wno-pointer-sign -Wframe-larger-than=2047 -Wno-address-of-packed-member -Wno-constant-conversion -mcmodel=kernel -mno-red-zone -mno-sse2 -mno-sse -mno-3dnow -mno-mmx -msoft-float -fno-omit-frame-pointer -ffreestanding -fno-pie -msave-args -O2 -pipe -nostdinc -I/home/openbsd/sys -I/home/openbsd/sys/arch/amd64/compile/GENERIC.MP/obj -I/home/openbsd/sys/arch -I/home/openbsd/sys/dev/pci/drm/include -I/home/openbsd/sys/dev/pci/drm/include/uapi -I/home/openbsd/sys/dev/pci/drm/amd/include/asic\_reg -I/home/openbsd/sys/dev/pci/drm/amd/include -I/home/openbsd/sys/dev/pci/drm/amd/amdgpu -I/home/openbsd/sys/dev/pci/drm/amd/display -I/home/openbsd/sys/dev/pci/drm/amd/display/include -I/home/openbsd/sys/dev/pci/drm/amd/display/dc -I/home/openbsd/sys/dev/pci/drm/amd/display/amdgpu\_dm -I/home/openbsd/sys/dev/pci/drm/amd/powerplay/inc -I/home/openbsd/sys/dev/pci/drm/amd/powerplay/smumgr -I/home/openbsd/sys/dev/pci/drm/amd/powerplay/hwmgr -I/home/openbsd/sys/dev/pci/drm/amd/display/dc/inc -I/home/openbsd/sys/dev/pci/drm/amd/display/dc/inc/hw -I/home/openbsd/sys/dev/pci/drm/amd/display/dc/clk\_mgr -I/home/openbsd/sys/dev/pci/drm/amd/display/modules/inc -I/home/openbsd/sys/dev/pci/drm/amd/display/modules/hdcp -I/home/openbsd/sys/dev/pci/drm/amd/display/dmub/inc -I/home/openbsd/sys/dev/pci/drm/i915 -DDDB -DDIAGNOSTIC -DKTRACE -DACCOUNTING -DKMEMSTATS -DPTRACE -DPOOL\_DEBUG -DCRYPTO -DSYSVMSG -DSYSVSEM -DSYSVSHM -DUVM\_SWAP\_ENCRYPT -DFFS -DFFS2 -DFFS\_SOFTUPDATES -DUFS\_DIRHASH -DQUOTA -DEXT2FS -DMFS -DNFSCLIENT -DNFSSERVER -DCD9660 -DUDF -DMSDOSFS -DFIFO -DFUSE -DSOCKET\_SPLICE -DTCP\_ECN -DTCP\_SIGNATURE -DINET6 -DIPSEC -DPPP\_BSDCOMP -DPPP\_DEFLATE -DPIPEX -DMROUTING -DMPLS -DBOOT\_CONFIG -DUSER\_PCICONF -DAPERTURE -DMTRR -DNTFS -DHIBERNATE -DPCIVERBOSE -DUSBVERBOSE -DWSDISPLAY\_COMPAT\_USL -DWSDISPLAY\_COMPAT\_RAWKBD -DWSDISPLAY\_DEFAULTSCREENS="6" -DX86EMU -DONEWIREVERBOSE -DMULTIPROCESSOR -DMAXUSERS=80 -D\_KERNEL -MD -MP gapdummy.c -o gapdummy.o

cc: error: argument unused during compilation: '-mno-sse2' [-Werror,-Wunused-command-line-argument]

cc: error: argument unused during compilation: '-mno-sse' [-Werror,-Wunused-command-line-argument]

cc: error: argument unused during compilation: '-mno-3dnow' [-Werror,-Wunused-command-line-argument]

cc: error: argument unused during compilation: '-mno-mmx' [-Werror,-Wunused-command-line-argument]

cc: error: argument unused during compilation: '-msoft-float' [-Werror,-Wunused-command-line-argument]

cc: error: argument unused during compilation: '-msave-args' [-Werror,-Wunused-command-line-argument]

cc: error: argument unused during compilation: '-mno-sse2' [-Werror,-Wunused-command-line-argument]

cc: error: argument unused during compilation: '-mno-sse' [-Werror,-Wunused-command-line-argument]

cc: error: argument unused during compilation: '-mno-3dnow' [-Werror,-Wunused-command-line-argument]

cc: error: argument unused during compilation: '-mno-mmx' [-Werror,-Wunused-command-line-argument]

cc: error: argument unused during compilation: '-msoft-float' [-Werror,-Wunused-command-line-argument]

cc: error: argument unused during compilation: '-msave-args' [-Werror,-Wunused-command-line-argument]

\*\*\* Error 1 in target 'swapgeneric.o'

\*\*\* Error 1 in /home/openbsd/sys/arch/amd64/compile/GENERIC.MP (Makefile:1772 'gapdummy.o')

\*\*\* Error 1 (Makefile:1718 'swapgeneric.o')

Solution:

In file openbsd/[sys/dev/fdt/files.fdt](https://github.com/Hemaprasanthi/openbsd/commit/2a17946f5dbe83c231dffad3aa40aff16d9c9139#diff-c011458f600871f1be3e6ea7d0afaf306dc8783d6d85cc33f3ba63f3d9420867) uncomment the lines:

device rkdrm: drmbase, wsemuldisplaydev, rasops15, rasops16, rasops24, rasops32

attach rkdrm at fdt

file dev/fdt/rkdrm.c rkdrm

file dev/pci/drm/drm\_gem\_cma\_helper.c rkdrm

**Hema:**

Welcome to the OpenBSD/amd64 6.6 installation program

(I) Install ..

Enter I and then press enter

Keep pressing enter for all the question.

Enter a host name : hem

Enter a password for the root:

Re-enter the password for the root:

Directory doesnot contain SHA256.sig. Continue without verification? yes

Exit to (S) Shell ….

Press R for reboot

login: root

password: hema

enter the su command // this will help you to get into your root account.

pkg\_add git // command to install git

git clone git://github.com/openbsd/src.git

go to sys/arch/amd64/compile/GENERIC.MP and type make obj command

then type make config command

then make -j 6(this is number of cpu’s)

after that type the folllowing command:

cp *home*/hema/src/sys/arch/amd64/compile/GENERIC.MP/obj/bsd /bsd0

after that

cd *usr/*obj

here we are going to obj folder because we will run out of space if we are in user.

Type df -h to see all the users and the space it has

type cd *usr*/obj/

mkdir xx

cd xx

vmctl create -s 5g inner\_vm.qcow2

Now download the install 66.iso

reboot

shutdown

go to the details of the host openbsd and go to view→details→CPUs→checkcopy host CPU configuration

start your openbsd vm

df -h

cd *usr/*obj/xx/

// now start your guest

rcctl enable vmd

rcctl start vmd // this should give vmd(ok)

fw\_update

vmctl start -c -r install66.iso -d inner\_vm.qcow2 -i 1 test

References:

https://turbonomic.com/wp-content/uploads/2015/07/memory-management-fundamentals\_ebook2.pdf

https://searchservervirtualization.techtarget.com/feature/How-hypervisors-dynamically-allocate-memory-to-improve-VM-performance

https://www.maketecheasier.com/fix-broken-packages-ubuntu/

http://jeffwouters.nl/index.php/2011/01/dynamic-memory-vs-memory-overcommit/comment-page-1/#comment-74993

https://searchservervirtualization.techtarget.com/definition/memory-ballooning

https://vitux.com/how-to-check-if-your-processor-supports-virtualization-technology/

**How to see the ipaddress of your vm?**

ifconfig

**How to transfer one file from linux to outer vm?**

Scp filename root@ipaddress\_of\_the\_outervm:/path\_of\_the\_location\_in\_outervm

**Anjala**

**In bsd- cd /home/anju/openbsd/sys/arch/amd64/compile/GENERIC.MP**

**vmctl start -c -r /home/anju/xx/install66.iso -d /home/anju/xx/inner\_vm.qcow2 -i 1 test**

**(-to start vm) vmctl start -c -d /home/anju/xx/inner\_vm.qcow2 -L test**

**Steps to run custom qemu and virtmanager**

**Ruchika**

**ps -aux | egrep 'vmd'**

**vmctl start -c -r /usr/obj/install66.iso -d /home/xx/inner\_vm.qcow2 -i 1 test**

**x vmctl start -c -d /home/xx/inner\_vm.qcow2 -L test**

**-------------------------------------------------------------------------------------------------------------------------------------------**

Approach:

1. Request the memory stats from the inner VMs.

Actor: Balloon device

Approach: First, device needs information from every VM that is running on it. For memfree tag, it will update the balloon\_stats’s tag value as 3. It will notify the driver.

Implementation:

Need to trigger the action to retrieve the stats by updating the stats queue:

Task 1:

1. User executes a vmctl command **vmctl getStats** from within the outer VM - **DONE**
2. Update the empty envelope (take from the available ring and put it in the used)
3. Interrupt the driver
4. From inner VM **viomb\_stats\_intr** will handle the request
5. run a function to return “memfree”

Task 2:

1. Stats queue descriptor table needs to be updated
2. Put it in used queue.

Proposed Memory Reclaim Algorithms:

1. Weighted Round Robin with percentage (75)
2. Weighted Round Robin with available free memory
3. Ratio based even distribution

Example 1: VM1 - 1000, VM2 - 100, VM3 - 10, Host Requires 990

Algorithm 1: 750, 75, 7.5 // performance

Algorithm 2: 832.5

Algorithm 3: Assume host requires 900

1110 is the total so 1000 + 100 + 10

VM1 would be asked for (1000/1110 \* 900) = 810.81

VM2 (100/1110) \* 900 = 81.08

VM3 (10/1110) \* 900 = 8.11

VM 1000 total

When it’s 100% of it’s available, it will start swapping?