Virginia Tech ❖ Bradley Department of Electrical and Computer Engineering

ECE 5984 Virtualization Technologies

Project 2: Inter-VM communication channels

Introduction

The goal of this project is to implement different communication mechanisms between two or more VMs and to study their performance. Three implementations are required form the student:

- a naive implementation that use IN/OUT port instructions;
- an implementation that use shared memory without interruptions;
- an implementation that use shared memory with interruptions;

Please note that the use of kvmtool/KVM is a must since such communication mechanism already exist in Qemu (see section 2).

Related Work

Qemu has a support for such mechanism. It is called *ivshmem*. Students are free to inspire/reuse the code from Qemu. An even better place to get inspiration can be found in the source of KVM, where an attempt to replicate *ivshmem* has been attempted (see *kvmtool/hw/pci-shmem.c*). However, this implementation is not functional. Of course, it is not mandatory to replicate *ivshmem* as long as all the projects three requirements are met.

Project steps

The required steps have been described in section 1. Here we will give more details on them.

IO Ports Implementation

This step requires modifying the kvmtool and adding a program to the guest VM in userspace. Modifying the guest kernel should not be necessary.

- Kvmtool need to be modifyed to catch IN/OUT instructions on some given ports;
- The same ports will be used by the userspace program to communicate with kvmtool. To access these ports in userspace, one can use: *ioperm(), iopl(), inb(), outb() and similar functions*;

Shared Memory w/o Interrupt

This implementation requires setting a shared memory between "root mode" and "non-root mode". It is probably possible to implement it without modifying the guest kernel as long as one can find the physical address of a memory region in userspace. However it is recommended to use a kernel module as in the next step.

Shared Memory with Interrupt

For this step we need to implement a device driver in Linux. It may simpler to use the UIO device driver model for this task. See Linux kernel documentation in: *Documentation/driver-api/uio-howto.rst*

Note: even when using shared memory, you may need to use the IO port implementation to do some initialization and so on.

Project Report

The report should be 10 pages and should at least contain:

- An introduction: describe the problem, a quick about the solutions, a description of the results, etc;
- A clear description of the software architecture of each implementation (figures can be used to facilitate the understanding):
  - What are the different layers involved in the communication;
  - A detailed description of how the data traverse the different; memory layers, CPU modes, software layer and so on;
  - Also, if multiple solutions where available write them down and explain you choice;
- Implementation details;
- A performance analysis:
  - comparison between the different implementations. For example, the time to transfer 4KB, 2MB, 1GB;
  - a break down of the overhead of each software layer within the most performing implementation.

Remark: A concise report, even if it contains less pages than requested, is better than an empty 10 pages report.

Group work

This project can be accomplished in groups of maximum 2 students.

Results to be handed - Deadline: 2018-03-28 11:59PM

The following is expected to be handed by 2018-03-28, 11:59PM:

1. Sources and patches for each implementations.
   Remark: avoid putting the code of kvmtool or Linux or other existing big project sources: put patches;
2. README files explaining:
   - the content of each source and patch;
   - how to compile and run the implementations;
3. Shell scripts to reproduce the experiments, if any;
4. The project report in PDF format.

All of this should be contained in a compressed archive. One archive per group should be submitted.

Grading

This project will be graded on 100 which are distributed as follow:

1. IO port implementation = 25 points
2. Shared memory w/o interrupts = 25 points
3. Shared memory with interrupts = 25 points
4. Report = 25 points

Don't forget to earn extra points by doing the homework. Hopefully everybody will have 100 points at the end.

Remark: This grading maybe subject to change.