Boston University
Electrical and Computer Engineering

Senior Thesis PDRR Report

Author:
Vijay Thakkar

Supervisor:
Dr. Richard West

October 8, 2018

Contents

1 Thesis Statement **2**

2 Summary **2**

3 Motivation **2**

4 Objectives and Methodology **3**

5 Requirements **5**

6 Deliverables **5**

1 Thesis Statement

Design and implement a proof of concept system for low latency and low overhead cross virtual machine communication between an Android OS and a Quest RTOS emulated on x86 multi-core processors while ensuring the timing constrains of the mixed criticality domain.

2 Summary

We intend to design and implement a proof of concept system for the future autonomous car platforms that allow for mixed criticality processing on a consolidated high performance system on chip. We have chosen x86 processors as the platform of choice due to client requirements as well as the ready availability, widespread use and proven functionality of multi-core x86 processors. We believe that time is a first class resource and therefore require that the final platform have a Real Time Operating System that is responsible for the high criticality control processing of the car. In addition, our client requires that we have an Android OS also running on the same platform to serve as the low criticality (info-tianment) processing. However, we still need to enable some cross virtual machine communication to maintain some shared data such as CAN bus messages of the car in a way that does not interfere with the timing guarantees of the RTOS control processing. The goal of the thesis is to design and implement a proof of concept system that allows such cross-VM communication within a mixed criticality domain.

3 Motivation

Traditionally, vehicle control systems have comprised of many dozens of microprocessors that control the car via relays and communicate with each other over a Controller Area Network (CAN) bus. These micro-controllers often have architectures that are not widely supported by mainstream software toolchains and are not powerful enough to run compute heavy software such as Android CarPlay systems. This poses many limitations on modern cars due to their heavy integration of automated navigation systems, infotainment

and even autonomous driving.

The approach has been to include a ARM processor and/or an NVidia GPU in addition to the set of micro controllers specifically to serve these modern functions with the control network completely isolated. This results in a complex design and increased costs that can be eliminated if a single consolidated platform for both control systems as well as infotainment and a host for automated driving were to be used.

The approach taken by Professor Rich West and his team has been to design and build such a platform for automobiles of the future that use a single multi-core x86 processor as a platform for hosting all of the functionality without compromises. The intended design of this plat form has Quest-V as a partitioning hypervisor with an instance of Quest RTOS guest for time-critical control systems processing, a Linux guest OS for other miscellaneous compute and finally Android guest as the host for low criticality processing. For this, we need a method to efficiently deliver messages between the virtual machines, which is the overarching goal of the thesis.

4 Objectives and Methodology

The primary objective of the project is to design and implement a software solution to enable cross VM communication. Preliminary research for the project will asses the feasibility of two different methods to achieve this. Although the final product will utilize Quest-V as the hypervisor, for the purpose of this proof of concept, we relax this requirement to be able to use any hypervisor of our choosing that can successfully demonstrate the validity of the concept. To this end, we have the following two options for the choice of hypervisor:

- Linux as the host operating system with a type 2 hypervisor (KVM, qemu etc.) on top of it.
- Quest-V as the type 1 hypervisor with guests running directly on it.

The first method would be easier to set up as it would require no direct hardware configuration. In the former case, we first intend to start by setting up a RAM disk in the host operating system, create a single file in it, mount that file in both the guest operating systems and then memory map the file to serve as the shared memory region between the two guests. If

this is not fast enough for our requirements due to the file system abstraction overhead, we again have two choices for further optimizations. We can either modify the type 2 hypervisor to map certain virtual memory pages of the host operating system to both the guests as a method to implement cross-VM shared memory, or we revert to using Quest-V type 1 hypervisor. Modifications to existing hypervisors such as qemu would be difficult due to the complexity of the large codebase, and we would likely take the latter approach since Quest-V is the requirement for the final product and both approaches would require substantial modifications anyway. To use Quest-V hypervisor, we would change the configuration of the x86 memory management unit (MMU) mapping at the hardware level to to map a certain region of physical memory to the same virtual memory pages for both guests. Once the MMU is configured, we would then write specific drivers (one for each guest) that can then expose this shared memory region to userland processes through a system call API. In any of these scenarios, the overall architecture of our approach looks functionally similar to the representation in figure 1.

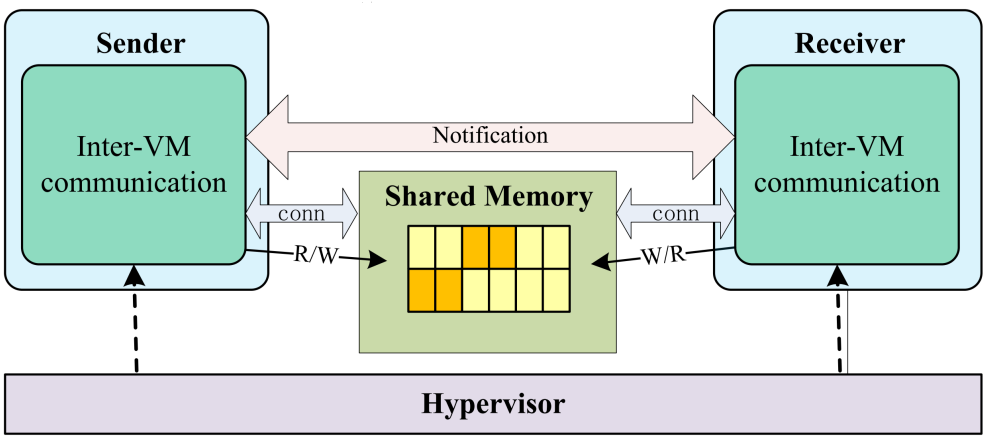

Figure 1: Design diagram of the shared memory style cross-VM communication. The green sections within the VMs represent the driver layer, the shared memory region in the middle represents the implementation of either directly shared memory access through the hypervisor or through a file-system abstraction.[2]

[2]Yi Ren. 2016. Shared-Memory Optimizations for Inter-Virtual-Machine Communication. *ACM Computing Surveys 48, 4* (May 2016).

5 Requirements

The requirements for the thesis will be primarily set over the course of the research phase, however, there are some primary requirements that are binary in nature. Following is a list of requirements that must be met for the project without a single failure:

- Both guest operating systems must have direct read-write access to shared memory across the hypervisor.
- Latency for communication must be less than 60Hz refresh rate of the infotainment system. This translates to a maximum allowable latency of 33.3 ms for the cross-VM communication.
- There must be no CAN packets or messages that are dropped or missed during communication.
- Android OS kernel module and processing must not pose any viable risk to the high criticality.

6 Deliverables

Following is a list of the tentative deliverable for the thesis. Note that depending on design decisions made over the course of the research, some components may be added or removed, therefore the tentative nature of the list.

- Linux or Quest-V (host) driver to set up the shared buffer.
- Linux or Quest (guest) driver as a producer for buffer.
- Android (guest) driver as consumer of buffer.
- Demonstration of the functionality of the above three on hardware (possibly requiring userland programs).
- Final Thesis.