Google Summer of Code 2018

From QEMU

Contents

Introduction

QEMU is participating in Google Summer of Code 2018 (http://g.co/gsoc). This page contains our ideas list and information for students and mentors. Google Summer of Code is an open source internship program for university students offering 12-week, full-time, paid remote work from May to August!

Applicants: QEMU is also participating in Outreachy May-August 2018. We recommend applying for the same project idea in both internship programs if you are eligible.

Application Process

After contacting the mentor to discuss the project idea you should fill out the application form at [1] (https://g.co/gsoc/). The form asks for a problem description and outline of how you intend to implement a solution. You will need to do some background research (looking at source code, browsing relevant specifications, etc) in order to form an idea of how to tackle the project. The form asks for an initial 12-week project schedule which you should create by breaking down the project into tasks and estimating how long they will take. The schedule can be adjusted during the summer so don't worry about getting everything right ahead of time.

Candidates may be invited to an IRC interview with the mentor. The interview consists of a 30 minute C coding exercise, followed by technical discussion and a chance to ask questions you have about the project idea, QEMU, and GSoC. The coding exercise is designed to show fluency in C programming.

Here is a C coding exercise we have used in previous years when interviewing students: 2014 coding exercise (http://pastebin.com/6J1wwDhK)

Try it and see if you are comfortable enough writing C. We cannot answer questions about the previous coding exercise but hopefully it should be self-explanatory.

If you find the exercise challenging, think about applying to other organizations where you have a stronger technical background and will be more competitive compared with other candidates.

Find Us

- IRC (GSoC specific): #qemu-gsoc on irc.oftc.net
- IRC (development):
  - QEMU: #qemu on irc.oftc.net
  - KVM: #kvm on chat.freenode.net
- Mailing lists:
  - QEMU: qemu-devel (http://lists.nongnu.org/mailman/listinfo/qemu-devel)
  - KVM: linux-kvm (http://www.linux-kvm.org/page/Lists,_IRC)

Please contact the mentor for the project idea you are interested in. IRC is usually the quickest way to get an answer.

For general questions about QEMU in GSoC, please contact the following people:

- Stefan Hajnoczi (stefanha on IRC)

Project Ideas

This is the listing of suggested project ideas. Students are free to suggest their own projects, see #How to propose a custom project idea below.

Separate process for UI

Summary: Run QEMU UI in a separate process

Normally, if QEMU is compiled with graphical window support, it displays output such as guest graphics, guest console, and the QEMU monitor in a window.

On the other hand, there are remote display viewers, such as remote-viewer that provide similar functionalities for user interactions and desktop integration. There is a duplication of effort to provide a good experience on the various OS & desktops over time.

With the SPICE protocol, there shouldn't be a performance hit, since the compression is disabled locally and the display process may use shared buffers/textures.

However, some functions are lacking, such as QEMU monitor interactions or console support. They can be provided thanks to SPICE "port" channel for example. Other mechanisms or solutions are possible (for example, quit QEMU when the display process exit by watching the child process). In other cases, virt-viewer provides a superior experience: clipboard sharing, usb hotplugging, kiosk mode, custom key bindings, shared folders, advanced display configuration, etc.

The goal of this summer of code is to provide a new -display backend to run a "QEMU UI" in a seperate process. It should try to provide as much functionality as the existing display backends.

For example, a -display spice backend that would run an enhanced remote-viewer as child process, with additional menu entries and terminal consoles providing a similar experience as -display gtk.

Links:

- https://git.qemu.org/?p=qemu.git;a=blob_plain;f=docs/spice-port-fqdn.txt;hb=HEAD
- https://www.spice-space.org/
- https://virt-manager.org/

Details:

- Skill level: intermediate or advanced
- Language: C
- Mentors: marcandre.lureau@redhat.com, kraxel@redhat.com
- Suggested by: marcandre.lureau@redhat.com

Multiple PCI domains for x86 PCI Express Machine (Q35)

Summary: Implement multiple PCI domains support for x86 machines.

Currently QEMU supports multiple PCIe Host Bridges in Q35 PCI Express chipset using the pxb-pcie device.

However all PCIe Host Bridges are part of the same PCI domain (0). This is a serious limitation since all of them have to share a range of 256 PCI buses, which leads to a hard limit on the number of PCI devices the Q35 machine can use.

The limitation comes from the fact the PCI Express topology limits one PCI Express device per PCI bus.

PCI devices have a set of registers referred to as ‘Configuration Space’ and PCI Express introduces Extended Configuration Space mechanism for devices (ECAM). Basically, the 'Configuration Space' registers are mapped to a memory location which can be accessed by software in order to setup the PCI devices.

Each PCI domain requires a different (ECAM) space, however QEMU implements only one.

The goal of this summer project is to improve Q35's PCIe Host bridges (pxb-pcie) in order to place them on their own PCI domain in order to remove the mentioned limitation.

Links:

- http://qemu-project.org/Features/Q35
- http://wiki.qemu-project.org/images/4/4e/Q35.pdf
- https://git.qemu.org/?p=qemu.git;a=blob_plain;f=docs/pcie_pci_bridge.txt
- http://git.qemu.org/?p=qemu.git;a=blob_plain;f=hw/pci-bridge/pci_expander_bridge.c
- http://wiki.qemu-project.org/images/f/f6/PCIvsPCIe.pdf

Details:

- Skill level: intermediate
- Language: C
- Mentor: marcel@redhat.com, marcel_a on IRC
- Suggested by: Marcel Apfelbaum <marcel@redhat.com>

PCI Express Root Port enhancements

Summary: Add PCI Express advanced features to QEMU's emulated PCI Express Root Ports

QEMU's PCI Express machine (Q35) emulates generic PCI Express Root Ports.

The PCI Express Root Port emulation today supports only a subset of PCI Express advanced capabilities, missing several ones that could add a lot of functionality and make it closer to real hardware.

The goal of this summer project is to improve the PCI Express Root Port implementation by:

1. *Exposing the Root Port as PCIe Gen3*. This requires tweaking the PCIe Root Port configuration registers to report link capabilities advertising Gen3 speeds/widths.
2. *Supporting up to 256 PCI Devices*. This requires implementing the Alternative Routing-ID Interpretation (ARI) capability for the PCI Express Root Port. A PCI function can be located in Configuration Space by a <bus,dev,func> tuple. The 'func' part is 3 bits wide, meaning a multi-function device can have up to 8 functions. Since PCI Express architecture dictates only one device per PCIe bus, ARI uses a <bus,func> address, this time 'func' using 8 bits resulting in 256 functions per PCI device.
3. *Securing PCI Express devices via the Access Control Services (ACS) mechanism*. ACS can force Peer-to-Peer PCIe transactions to go up through the PCIe Root Complex. ACS can be thought of as a kind of gate-keeper - preventing unauthorized transactions from occurring. For example without ACS, several multi-function PCIe Root Ports belonging to the same slot cannot be used with the vIOMMU for nested virtualization since they can "talk" to each other bypassing it.

Links:

- http://qemu-project.org/Features/Q35
- http://wiki.qemu-project.org/images/4/4e/Q35.pdf
- http://git.qemu.org/?p=qemu.git;a=blob_plain;f=hw/pci-bridge/pcie_root_port.c
- http://wiki.qemu-project.org/images/f/f6/PCIvsPCIe.pdf

Details:

- Skill level: intermediate
- Language: C
- Mentor: marcel@redhat.com, marcel_a on IRC
- Suggested by: Marcel Apfelbaum <marcel@redhat.com>

micro:bit machine type

Summary: Add emulation support for the micro:bit board

The micro:bit (http://microbit.org/) is a small computer for educational use that is also suitable for embedded and Internet of Things (IoT) projects. It uses a 16 MHz ARM Cortex-M0 processor with 256 KB flash and 16 KB RAM. It features many I/O capabilities including a 5x5 LED display, 2 buttons, Bluetooth and Nordic Gazell radio communications, an accelerometer, a compass, temperature and light sensing, UART, and GPIO pins for external devices.

This project will add micro:bit emulation support to QEMU, allowing code written with the Python and Javascript Blocks editors (http://microbit.org/code/) to run on your computer. Baremetal code not written with the official editors, such as C/C++ code using the microbit-dal runtime, will also work since QEMU emulates a full system including CPU and hardware devices.

This project involves:

1. Implementing ARM Cortex-M0 CPU support based on existing Cortex-M3 support in QEMU
2. Implementing a micro:bit .hex ROM loader
3. Implementing a "microbit" machine type
4. Implementing at least the 5x5 LED display, buttons, and UART.
5. Stubbing out other devices as needed for the runtime to start successfully.

The goal is to run "Hello World!" programs created with the Python and Javascript Blocks editors. If you have time it may be possible to implement additional hardware devices or a graphical user interface.

Previous experience with low-level systems software (firmware, bootloaders, kernels) and hardware (Arduino, PIC, microcontrollers) is recommended. Start by looking at the reference design schematic (https://github.com/microbit-foundation/microbit-reference-design/blob/master/PDF/Schematic%20Print/Schematic%20Prints.PDF) to understand how the micro:bit works.

Links:

- Hardware specifications (http://tech.microbit.org/hardware/)
- Reference design hardware schematic (https://github.com/microbit-foundation/microbit-reference-design/blob/master/PDF/Schematic%20Print/Schematic%20Prints.PDF)
- microbit-dal runtime (https://github.com/lancaster-university/microbit-dal/)

Details:

- Skill level: advanced
- Language: C
- Mentors: Stefan Hajnoczi <stefanha@gmail.com> ('stefanha' on IRC), Jim Mussared <jim@groklearning.com>, Joel Stanley <joel@jms.id.au>

virgl on Windows host

Summary: make virgl rendering work on Windows host

virgl enables accelerated 3d rendering in a VM. It requires Desktop GL on the host.

In theory, virgl should work on Windows with a capable host driver. This project aim at making virgl work well with various GPU on Windows. Since many Windows OpenGL drivers have bad behaviours, it would be worth to support ANGLE/opengles instead. This would require various modifications in virgl library. Additionally, it would be a good opportunity to ease the cross-compilation and packaging of qemu/virgl with msitools.

Links:

- https://virgil3d.github.io/
- http://angleproject.org
- https://wiki.gnome.org/msitools

Details:

- Skill level: intermediate or advanced

- Language: C
- Mentors: marcandre.lureau@redhat.com, airlied@redhat.com
- Suggested by: marcandre.lureau@redhat.com

Vulkan-ize virgl

Summary: accelerated rendering of Vulkan APIs

virgl enables accelerated 3d rendering in a VM. It uses Desktop GL on host, and provides OpenGL/GLES in guest.

This project would aim at implementing Vulkan accelerated rendering. There are multiple ways of interpreting this idea. One interesting approach would be to support Vulkan in VM on a Vulkan-capable host, doing more passthrough.

Links:

- https://virgil3d.github.io/
- https://www.khronos.org/vulkan/

Details:

- Skill level: advanced
- Language: C
- Mentors: airlied@redhat.com, marcandre.lureau@redhat.com
- Suggested by: marcandre.lureau@redhat.com

Multi-CPU cluster support for GDB server in QEMU

There are many examples in modern computing where multiple CPU clusters are grouped together in a single SoC. This is common in the ARM world especially. There are numerous examples such as ARM's big.LITTLE implementations and Xilinx's 4xA53s and 2xR5s on the ZynqMP SoC. The goal of this task is to add support to the GDB server to allow users to debug across these clusters.

Xilinx has an out of tree implementation that can be used as a starting point. Work will need to be done on top of this to prepare it for upstream submission and to ensure the implementation is more generic.

This will mostly involve extending GDB server to tell GDB about different architectures and then allow the user to swap between them.

The Xilinx implementation can be seen here: https://github.com/Xilinx/qemu/blob/master/gdbstub.c There has been some steps in preparing the work to go upstream, which can be seen here: https://github.com/Xilinx/qemu/tree/mainline/alistair/gdb

Details:

- Skill level: advanced
- Language: C
- Mentor: Alistair Francis <alistair23@gmail.com>, Philippe Mathieu-Daudé <f4bug@amsat.org>
- Suggested by: Alistair Francis

ARM SMMU Support for Jailhouse Hypervisor

Summary: Implement basic SMMU support for ARM in the Jailhouse hypervisor

The Jailhouse hypervisor already supports IOMMUs for x86 targets but lacks a corresponding feature for ARM. Specifically modern ARM64 targets like the Xilinx Zynq UltraScale+ MPSoC come with ARM SMMU-based IOMMU support, and Jailhouse already suppors exactly that target. Even better, there is an emulation of that target available in QEMU. So the task is to develop the ARM-specific parts in Jailhouse to IOMMU support and fill them with SMMU driver code that runs on the QEMU model (real hardware would be available for occasional remote tests).

The task requires building up a test environment using the QEMU machine model of the Zynq MPSoC, reading the related hardware specifications, and then designing and implementing the extension to Jailhouse. While the target hardware is the (virtual) Zynq MPSoC, the design should allow later extensions to other SMMU-compatible targets like the AMD Opteron 1100.

Links:

- Jailhouse project, including setup in QEMU/KVM (https://github.com/siemens/jailhouse)
- Jailhouse tutorial (https://events.linuxfoundation.org/sites/events/files/slides/ELCE2016-Jailhouse-Tutorial.pdf)
- SMMU specification (http://infocenter.arm.com/help/topic/com.arm.doc.ihi0062d.c/IHI0062D_c_system_mmu_architecture_specification.pdf)
- Xilinx QEMU version for Zynq MPSoC with SMMU support (https://github.com/Xilinx/qemu)

Details:

- Skill level: advanced
- Language: C
- Mentor: Jan Kiszka <jan.kiszka@web.de>

Python module for Jailhouse

Summary: Factor out a Python module for the Jailhouse command & control interface

The partitioning hypervisor Jailhouse is controlled from a Linux guest system (the root cell) via a command line tool (`jailhouse`). This tool consists of a small core written in C, providing the minimally required services to operate the hypervisor. For more advanced features like configuration file generation or the loading of additional Linux guests, a set of extension scripts have been written in Python. The scripts are now at a point where they would benefit from using a "pyjailhouse" module that should implements common functionality only once, e.g. configuration format or system information parsing but also interaction with the low-level hypervisor control interface.

The goal of this project is to create such Python module, transfer existing code from the scripts into it, and make sure that the module is both properly installed and also usable from the source directory. As a reuse case and benchmark for a reasonable split-up, `jailhouse hardware check` shall be enabled to parse the system configuration itself, instead of relying on a config file previously generated by `jailhouse config create`.

Bonus (with advanced Python experience): Further refactoring of the system configuration parser and config file generator to enable the reuse on non-x86 architectures or for generating and validating non-root cell configurations.

Links:

- Jailhouse project, including setup in QEMU/KVM (https://github.com/siemens/jailhouse)
- Jailhouse tutorial (https://events.linuxfoundation.org/sites/events/files/slides/ELCE2016-Jailhouse-Tutorial.pdf)

Details:

- Skill level: intermediate to advanced
- Language: Python (basic C knowledge helpful)
- Mentors: Jan Kiszka <jan.kiszka@web.de>, Valentine Sinitsyn <valentine.sinitsyn@gmail.com>

AMD Interrupt Remapping Support for Jailhouse hypervisor

Summary: Supplement existing AMD IOMMU code with interrupt remapping support.

Jailhouse currently supports IOMMU on AMD-based x86 systems, however it does so for memory transfers only. In order for the isolation to be complete, the analogous translation should be applied to interrupt messages as well. This technique is commonly known as interrupt remapping and it is provided by AMD IOMMU.

The goal of this project is to implement the missing bits in AMD IOMMU interrupt remapping support to bring it on par with Intel interrupt remapping support Jailhouse already has. This involves writing code for the hypervisor core and arch-specific parts as well as adding relevant options to the config file generator.

In order to succeed with this project, you must have a sufficiently recent (2015 or newer) AMD computer (PC or laptop) which you can use as a testbed. You should also check that:

- The APU is Kaveri-based or newer
- There is an IOMMU option in the UEFI setup tool and it's enabled
- A recent Linux distribution reports IOMMU support in dmesg
- And you can pass-through a PCI device such as a network or sound card to a guest running in the virt-manager.

You should also check that the mainboard has a serial port header as this helps debugging a lot.

Links:

- Jailhouse project, including setup in QEMU/KVM: https://github.com/siemens/jailhouse
- Jailhouse tutorial: https://events.linuxfoundation.org/sites/events/files/slides/ELCE2016-Jailhouse-Tutorial.pdf
- A strawman implementation on top of the old Jailhouse tree (can be used as a starting point): https://github.com/vsinitsyn/jailhouse/tree/amd-vi-ir
- AMD IOMMU specification: https://support.amd.com/TechDocs/48882_IOMMU.pdf

Details:

- Skill level: Advanced
- Language: C, Python
- Mentor: Valentine Sinitsyn <valentine.sinitsyn@gmail.com>, Jan Kiszka <jan.kiszka@web.de>

Patchew REST API improvements

Summary: Improve the REST API to retrieve patch metadata, testing results, etc. from patchew.org (http://patchew.org)

Patchew is an open source CI project to automate testing of patches submitted as emails on mailing lists. Currently Patchew has a simple API, but it is complicated to use it because it is not REST-like and it exposes low-level details of the patchew database schema. The project aims at replacing it with a new REST API. A first implementation of the API is available in a pull request (https://github.com/patchew-project/patchew/pull/45); other improvements on top include implementing missing features of the old API (importing from mailing lists, tracking git repositories), OAuth authentication, and a webhooks plugin.

Details:

- Skill level: beginner/intermediate
- Language: Python
- Mentor: Paolo Bonzini <pbonzini@redhat.com>, Fam Zheng <famz@redhat.com>
- Suggested by: Paolo Bonzini

QEMU NVMe Performance Optimization

Summary: QEMU's NVMe implementation uses traditional trap-and-emulation method to emulate I/Os, thus the performance suffers due to frequent VM-exits. NVMe Specification 1.3 defines a new feature to update doorbell registers using a Shadow Doorbell Buffer. This can be utilized to enhance performance of emulated controllers like QEMU NVMe. The goal of this summer of code is to add such support in QEMU and apply polling techniques to achieve comparable performance as virtio-blk dataplane. Specifically, this project includes the following parts: (1) add shadow doorbell buffer and ioeventfd support into QEMU NVMe emulation, which will reduce the number of VM-exits and make them less expensive (reducing VCPU latency); (2) add iothread support to QEMU NVMe emulation to reduce or eliminate VM-exits; (3) add a RAM disk back-end for debugging; (4) implement an interrupt coalescing scheme for efficient host-to-guest communication (at least one of (3) and (4)).

Links:

- https://nvmexpress.org/wp-content/uploads/NVM_Express_Revision_1.3.pdf
- http://ucare.cs.uchicago.edu/pdf/fast18-femu.pdf
- https://github.com/ucare-uchicago/femu

- https://vmsplice.net/~stefan/stefanha-kvm-forum-2017.pdf

Details:

- Skill level: intermediate-advanced
- Language: C
- Mentor: Paolo Bonzini <pbonzini@redhat.com>, bonzini on IRC
- Suggested by: Huaicheng Li <huaicheng@cs.uchicago.edu>, Paolo Bonzini

Driver framework for libqos

QEMU uses "qtest" as a mechanism for tests to interact with devices. qtest unit tests are currently written in C, using the GTest framework from glib and glue libraries called "libqtest" and "libqos". libqtest implements the qtest socket protocol, while libqos provides utility functions to deal with e.g. guest memory allocation and PCI devices. However, the functionality of libqos is limited. The purpose of the project is to build a driver framework in libqos to enable combinatorial testing. For more information, see Features/qtest_driver_framework.

Details:

- Skill level: medium
- Language: C
- Mentor: Paolo Bonzini <pbonzini@redhat.com> (bonzini on IRC), Laurent Vivier <lvivier@redhat.com> (lvivier on IRC)

Project idea template

```
=== TITLE ===

'''Summary:''' Short description of the project

Detailed description of the project.

'''Links:'''
* Wiki links to relevant material
* External links to mailing lists or web sites

'''Details:'''
* Skill level: beginner or intermediate or advanced
* Language: C
* Mentor: Email address and IRC nick
* Suggested by: Person who suggested the idea
```

How to propose a custom project idea

Applicants are welcome to propose their own project ideas. The process is as follows:

1. Email your project idea to qemu-devel@nongnu.org. CC Stefan Hajnoczi <stefanha@gmail.com> and regular QEMU contributors who you think might be interested in mentoring.
2. If a mentor is willing to take on the project idea, work with them to fill out the "Project idea template" above and email Stefan Hajnoczi <stefanha@gmail.com>.
3. Stefan will add the project idea to the wiki.

Note that other candidates can apply for newly added project ideas. This ensures that custom project ideas are fair and open.

How to get familiar with our software

See what people are developing and talking about on the mailing lists:

- qemu-devel (http://thread.gmane.org/gmane.comp.emulators.qemu/)
- libvir-list (https://www.redhat.com/archives/libvir-list/)
- kvm (http://www.spinics.net/lists/kvm/)

Grab the source code or browse it:

- qemu.git (http://git.qemu-project.org/?p=qemu.git;a=summary)
- libvirt.git (http://libvirt.org/git/?p=libvirt.git;a=summary)
- kvm.git (https://git.kernel.org/cgit/virt/kvm/kvm.git/)

Build QEMU and run it: QEMU on Linux Hosts (http://wiki.qemu-project.org/Hosts/Linux)

Important links

- Student Manual (http://write.flossmanuals.net/gsocstudentguide/what-is-google-summer-of-code/)
- FAQ (https://developers.google.com/open-source/gsoc/faq)
- Timeline (https://developers.google.com/open-source/gsoc/timeline)
- Advice for students applying (from 2011 but still relevant!) (http://blog.vmsplice.net/2011/03/advice-for-students-applying-to-google.html)

Information for mentors

Mentors are responsible for keeping in touch with their student and assessing the student's progress. GSoC has a mid-term evaluation and a final evaluation where both the mentor and student assess each other.

The mentor typically gives advice, reviews the student's code, and has regular communication with the student to ensure progress is being made.

Being a mentor is a significant time commitment, plan for 5 hours per week. Make sure you can make this commitment because backing out during the summer will affect the student's experience.

The mentor chooses their student by reviewing student application forms and conducting IRC interviews with candidates. Depending on the number of candidates, this can be time-consuming in itself. Choosing the right student is critical so that both the mentor and the student can have a successful experience.

Retrieved from "https://wiki.qemu.org/index.php?title=Google_Summer_of_Code_2018&oldid=7504"
Category: GSoC
