Описание аппаратных конфигураций гостевых систем в эмуляторе QEMU в виде отдельных текстовых файлов

Горемыкин О.В.
ИСП РАН, Москва
goremykin@ispras.ru

Аннотация. QEMU – свободное программное обеспечение с открытым исходным кодом, предназначенное для эмуляции различных платформ. В данной статье описывается разработанный способ описания аппаратных конфигураций гостевых систем в эмуляторе QEMU с помощью отдельных текстовых файлов, использующих удобный для чтения и написания текстовый формат JSON. Внешние конфигурационные файлы позволят избежать необходимости перекомпилировать QEMU при любых изменениях аппаратных конфигураций гостевых систем, а также избавят от необходимости разбираться во всех тонкостях внутренней реализации эмулятора.

Ключевые слова: QEMU; эмуляция; аппаратная конфигурация

1. Введение

Эмулятор в вычислительной технике – это комплекс программных или аппаратных средств, а также их сочетание, предназначенный для выполнения функций одной вычислительной системы (гостевой) на другой (целевой), таким образом, чтобы эмулируемое поведение как можно ближе соответствовало поведению оригинальной системы [1].

В настоящее время существует множество систем эмулирования, одной из них является QEMU [2]. Данный эмулятор имеет два режима работы: режим полносистемной эмуляции и режим эмуляции приложения. В первом режиме эмулируется все аппаратное обеспечение, включая периферийные устройства, и в полученной виртуальной машине загружается своя операционная система. Второй режим позволяет запускать отдельные пользовательские приложения, написанные для одной архитектуры, на другой [3].

Все аппаратные конфигурации гостевых систем в QEMU представлены в виде файлов, написанных на языке Си. Данные файлы являются громоздкими и сложными для восприятия и редактирования. Добавление нового устройства в реализованную систему представляет собой множество вызовов различных

функций с большим числом параметров. Также, после добавления устройства или любых других изменений в гостевой системе, необходимо заново собирать QEMU. Поэтому, с целью упрощения добавления новых гостевых систем или внесения в них изменений, разрабатывается средство описания аппаратных конфигураций гостевых систем с использованием текстовых файлов.

Подобный способ уже реализован в эмуляторе Simics[4], который имеет свой внутренний язык для моделирования устройств – Device Modeling Language (DML). Для описания конфигураций гостевых систем в Simics используются отдельные скриптовые файлы, которые полностью описывают аппаратные и программные особенности эмулируемых систем. Но данный эмулятор является платаным программным обеспечением и его код отсутствует в свободном доступе.

2. Реализация гостевых систем в QEMU

В QEMU гостевые системы представлены в виде структур, одним из полей которых является функция инициализации. В функции инициализации происходит создание центрального процессора, памяти, таймеров, контроллеров прерываний и различных периферийных устройств. Параметры данной функции соответствуют аргументам запуска QEMU, которые указывает пользователь.

Весь процесс инициализации гостевой системы можно условно поделить на три части:

1) инициализация памяти;
2) инициализация процессора;
3) инициализация периферийных устройств.

2.1. Модель памяти гостевой системы

Память в QEMU моделируется с помощью ациклического графа объектов MemoryRegion, листья которого являются RAM и MMIO(memory-mapped I/O).
В QEMU у гостевых систем существует четыре основных типа памяти [3]:

- RAM: диапазон RAM памяти целевой системы, которая доступная гостевой системе;
- MMIO: адреса гостевой системы, которым соответствуют указатели на функции чтения и записи, обеспечивающие пересылку данных между CPU и устройствами;
- Alias: диапазон памяти гостевой системы, позволяющий через свои адреса получать доступ к ячейкам памяти другого диапазона памяти;
- Контейнер: единая область памяти фиксированного размера, содержащая подобласти с разными смещениями относительно своего начала.

В QEMU возможна ситуация, когда алиасы или контейнеры перекрывают адресное пространство друг друга. Для разрешения коллизий используются приоритеты. Приоритеты локальны и сравниваются между подобластями одного уровня внутри контейнера. Перекрытие областей памяти служит исключительно для удобства моделирования памяти гостевой системы.

Во время инициализации гостевой памяти происходит создание объектов, каждый из которых характеризует одну из приведенных выше типов памяти.

2.2. Модель процессора и периферийных устройств гостевой системы

Для моделирования различных аппаратных компонентов гостевых систем в QEMU используется объектная модель (QOM – QEMU Object Model). QOM обеспечивает создание и регистрацию произвольных типов, а также создание экземпляров этих типов. Каждый QOM объект состоит из двух структур: Class и Object. Class содержит перечень указателей на функции и общие для всех классов поля. Object содержит поля, значения которых индивидуальны для каждой копии объекта [3].

Каждый объект может иметь свойства. Свойства описываются структурой «ObjectProperty». Данная структура содержит информацию об имени свойства и его типе, указатели на функции доступа к свойству, указатель на внутренние данные свойства, а также деструктор. Все свойства объекта образуют единый список.

На основе QOM в QEMU создаются процессор, различные устройства, а также шины. При создании новых устройств создаются и инициализируются новые объекты QOM.

У каждого устройства в QEMU имеется указатель на шину, к которой оно подключено, а также список шин, которыми оно управляет. Аналогично с шинами: каждая шина имеет указатель на одно родительское устройство и список указателей на дочерние устройства. Таким способом в QEMU задается дерево, где на каждом уровне находятся либо устройства, либо шины. Корнем данного дерева является системная шина.

3. Реализация гостевых систем с использование конфигурационного файла

3.1. Язык описания аппаратных конфигураций

Для описания аппаратных конфигураций в виде отдельных текстовых файлов был выбран текстовый формат JSON (JavaScript Object Notation)[5].

Текст в формате JSON представляет собой одну из двух структур:

- коллекция пар ключ/значение. В разных языках, эта концепция реализована как объект, запись, структура, словарь, хэш, именованный список или ассоциативный массив;

- упорядоченный список значений. В большинстве языков это реализовано как массив, вектор, список или последовательность.

Коллекция пар ключ/значение в JSON называется объектом, а упорядоченный список значений называется массивом. Значение может быть строкой в двойных кавычках, числом, true, false, null, объектом или массивом. Объекты и массивы могут быть вложенными друг в друга.

Данный формат был выбран по следующим причинам:

- большие расходы на разработку собственного текстового формата, а также разработка, внедрение и сопровождение инструмента, позволяющего переводить данные из текстового документа во внутреннее представление QEMU;
- текстовый формат JSON используется «QEMU Monitor Protocol»; он необходим для взаимодействия QEMU с другими программами, поэтому в QEMU уже встроен транслятор формата JSON, который осуществляет однозначное преобразование поступивших данных во внутренне представление QEMU(QOM объекты);
- данный текстовый формат является простым и в тоже время способен полностью покрыть все возможные аппаратные конфигурации гостевой системы.

Далее рассмотрим способ инициализации памяти, процессора, периферийных устройств и шин с помощью конфигурационного файла.

3.2. Описание аппаратных конфигураций гостевых систем с использованием формата JSON.

Для инициализации гостевой системы используется текстовый файл аппаратных конфигураций. Аппаратные конфигурации в текстовом файле представлены в виде объекта, ключи которого характеризуют различные аппаратные свойства гостевой системы. Для указания файла был добавлен новый параметр запуска QEMU – config_file «file_name».

3.2.1 Описание памяти гостевой системы

Конфигурации памяти гостевой системы представлены в виде массива. Элементами массива являются объекты. Каждый объект массива описывает определенную область памяти. Описания полей объекта приведены в табл. №1.

Наименование	Тип JSON	Свойства
name	строка	название области памяти, используется исключительно для отладки
id	строка	уникальный идентификатор области памяти; используется для доступа к указанной области

parent	строка	строкой является *id* области памяти, в которую необходимо добавить подобласть; если не указывать данный параметр, то по умолчанию считается, что *parent* – выделенная для гостевой системы память
size	строка	размер создаваемой области памяти; размер задается в десятеричной или шестнадцатеричной системе счисления в виде строки, например, «0xff9a»; возможно в качестве параметра указать «*ram size*», что будет соответствовать размеру выделенной для гостевой системы памяти
offset	строка	смещение создаваемой подобласти относительно *parent*; смещение задается в таком же виде, как и поле *size*; если данное поле не указано, то будет создан контейнер
alias_to	строка	созданная область памяти будет являться алиасом к области памяти с *id*, указанным в строке
readonly	булев тип	по умолчанию данное поле имеет значение false; если данное поле имеет значение true, то созданная область будет доступна только для чтения
overlap_priority	число	по умолчанию созданные области памяти имеют приоритет 0, но для разрешения коллизий при перекрытии областей памяти необходимо использовать приоритеты

Табл. 1. Атрибуты объекта JSON, описывающего область памяти.

Все поля, за исключением size, не обязательны, так как при создании области памяти достаточно знать только её размер. Однако, объект с единственным полем size будет бесполезен, так как будет задавать пустой контейнер, который из-за отсутствия id нельзя будет использовать.

Данная структура объектов, описывающих области памяти, благодаря наличию уникальных идентификаторов, позволяет полностью моделировать древовидную структуру памяти гостевой системы аналогично тому, как описано в разделе 2.1.

Ниже, в качестве примера, приведен объект, отвечающий за инициализацию памяти в гостевой системе «kzm» архитектуры ARM с помощью конфигурационного файла.

«memory»: [


```
  {«name»:  «kzm.ram»,  «size»:  «ram_size»,  «offset»:
«0x80000000», «id»: «ram»},
  {«name»:  «ram.alias»,  «size»:  «ram_size»,  «offset»:
«0x88000000», «alias_to»: «ram»},
  {«name»:  «kzm.sram»,  «size»:  «0x4000»,  «offset»:
«0x1FFFC000»}
].
```

В данном примере создаются три разных области памяти, одна из которых является алиасом. Так как поле «parent» явно не указано, то все смещения «offset» создаваемых областей памяти берутся относительно начала выделенной для гостевой системы памяти.

3.2.2. Описание устройств гостевой системы

Устройства (за исключением процессора) в конфигурационном файле представлены в виде массива объектов. Из-за наличия зависимостей между устройствами их необходимо инициализировать в определенном порядке, например, перед инициализацией контроллера прерываний необходимо создать процессор. Каждое устройство характеризуется атрибутами, которые задаются с помощью полей объекта. В табл. №2 приведены основные атрибуты устройств.

Наименование	Тип JSON	Свойства
type	строка	данное поле однозначно определяет тип устройства во время создания QOM объекта
id	строка	уникальный идентификатор устройства
addr	строка или массив строк	MMIO: адреса гостевой системы, которым соответствуют указатели на функции чтения и записи, обеспечивающие пересылку данных между CPU и устройствами
irq	объект	объект состоит из строки *device id* и массива *num*; *device_id* – уникальный идентификатор устройства, к которому адресовано прерывание, *num* – массив номеров прерываний
gpio_out	объект	объект состоит из строки *device id* и массива *num*; *device id* – уникальный идентификатор устройства, от которого необходимо обрабатывать прерывания, *num* – массив номеров прерываний

prop_set	объект	объект состоит из строк *name* и *value*, где *name* – название свойства, а *value* – его значение; объект *prop set* необходим для инициализации имеющихся в устройстве дополнительных атрибутов; все добавляемые свойства должны быть явно закреплены за устройствами с определенными наименованиями
bus	объект	объект состоит из строк *device id* и *property*; данным объектом является дочерняя шина со свойством *property* у устройства с *id* равным *device id;* если объект *bus* явно не указан, то устройство будет подключено к системной шине

Табл. 2. Атрибуты объекта JSON, описывающего периферийные устройства.

Процессор в конфигурационном файле представлен в виде отдельного объекта. У него есть два атрибута:

1) «model_default»: наименование модели процессора, которая будет использоваться при инициализации, если другая модель не будет указана во время запуска QEMU;
2) «id»: уникальный идентификатор, который необходим для последующего использования процессора при инициализации устройств.

Рассмотрим на примере платы Versatile инициализацию процессора и контроллера прерываний PL190 (в данном примере пропущена инициализация памяти и устройств, отличных от PL190):

```
{
    «cpu»:  { «model_default»:  «arm926»,  «id»: «cpu»},
    «device»:
    [
          { «type»: «pl190»,   «addr»: «0x10140000»,   «id»:
«vectored interrupt controller»,
       «irq»: { «device_id»:  «cpu»,  «num»:  [0, 1] }}
    ]
}
```

Из примера видно, что для создания контроллера прерываний необходимо указать его название, адрес MMIO и номера прерываний, которые будут поступать на процессор. Прерывания с номерами 0 и 1 являются

прерываниями типа IRQ(Interruppt Request) и FIQ (Fast Interrapt Request) соответственно. После инициализации контроллера прерываний и процессора появляется возможность создания и других устройств.

4. Заключение

Разработанный способ описания аппаратных конфигураций позволяет инициализировать память гостевой системы, процессор и устройства вне исходного кода QEMU. Благодаря этому, появляется возможность без перекомпиляции эмулятора вносить изменения в аппаратные конфигурации гостевых систем, а также описывать новые системы.

Данный способ описания был протестирован на гостевых системах архитектуры ARM. Так как инициализация гостевых систем происходит один раз при запуске QEMU, то использование конфигурационного файла никак не сказывается на скорости работы эмулятора.

Описанный подход универсальный, поэтому позволяет легко включать в свою реализацию новые архитектуры гостевых систем.

Список литературы

[1] James Smith, Ravi Nair. Virtual Machines: Versatile Platforms for Systems and Processes. Morgan Kaufmann Publishers is an imprint by Elsevier, 500 Sansome Street, Suite 400, San Francisco, CA 94111, 2005.

[2] Fabrice Bellard, QEMU, a fast and portable dynamic translator, In Proc. of the USENIX Annual Technical Conference, pages 41–46, April 2005.

[3] QEMU – Open Source Processor Emulator. http://wiki.qemu.org . Дата обращения: 19.03.2014.

[4] Wind River Simics. http://www.windriver.com/simics. Date of visit: 19.03.2014.

[5] The JSON Data Interchange Format, Standart ECMA-404, Geneva, 2013.

Description of hardware configurations of guest systems in QEMU emulator as separate text files

O.V.Goremykin
ISP RAS, Moscow, Russia
goremykin@ispras.ru

Abstract. QEMU is an open-source full system emulator based on the dynamic binary translation approach. QEMU emulates a complete hardware environment including CPUs, peripheral devices (VGA cards, network interfaces, IDE controllers, sound and USB components, and other). Guest system hardware configuration initialized at QEMU startup. During initialization phase QEMU creates virtual CPUs, peripheral devices and memory hierarchy. Hardware configuration is a part of QEMU source code. Being written in C language hardware configuration can be difficult to edit. Also, QEMU must be recompiled after any change in hardware configuration like addition of a new device. A method of the guest system hardware configuration description as an external file is discussed in this paper. JSON format was selected for configuration files. QEMU has parser for this format readily available because JSON is used by QEMU Monitor Protocol. Each file describes hardware configuration of one board. It consists of sequence of JSON objects. Each object describes properties one peripheral device, bus, CPU or memory segment. Each object has a unique ID associated with it and can reference other objects by their IDs. An arbitrary board configuration can be described with this format. The proposed method was implemented in QEMU and tested on ARM guest systems. The method is generic and can be applied to any board with any CPU architecture.

Keywords: QEMU; emulation; hardware configuration.

References

[1] James Smith, Ravi Nair. Virtual Machines: Versatile Platforms for Systems and Processes. Morgan Kaufmann Publishers is an imprint by Elsevier, 500 Sansome Street, Suite 400, San Francisco, CA 94111, 2005.
[2] Fabrice Bellard, QEMU, a fast and portable dynamic translator, In Proc. of the USENIX Annual Technical Conference, pages 41–46, April 2005.
[3] QEMU – Open Source Processor Emulator. http://wiki.qemu.org. Date of visit: 19.03.2014.
[4] Wind River Simics. http://www.windriver.com/simics. Date of visit: 19.03.2014.
[5] The JSON Data Interchange Format, Standart ECMA-404, Geneva, 2013.