[PATCH v9] introduce vfio-user protocol specification

Thanos Makatos posted 1 patch **6 months, 1 week ago**
⇄ Diff against v4 (/QEMU/1600180157-74760-1-git-send-email-thanos.makatos@nutanix.com/diff/20210614104608.212276-1-thanos.makatos@nutanix.com/) v5 (/QEMU/20201028161005.115810-1-thanos.makatos@nutanix.com/diff/20210614104608.212276-1-thanos.makatos@nutanix.com/) v6 (/QEMU/20201109180001.177665-1-thanos.makatos@nutanix.com/diff/20210614104608.212276-1-thanos.makatos@nutanix.com/) v7 (/QEMU/20201130161229.23164-1-thanos.makatos@nutanix.com/diff/20210614104608.212276-1-thanos.makatos@nutanix.com/) v8 (/QEMU/20210414114122.236193-1-thanos.makatos@nutanix.com/diff/20210614104608.212276-1-thanos.makatos@nutanix.com/)
Download mbox (/QEMU/20210614104608.212276-1-thanos.makatos@nutanix.com/mbox)

✔ Test **checkpatch** passed (/logs/20210614104608.212276-1-thanos.makatos@nutanix.com/testing.checkpatch/?type=message)

✔ Patches applied successfully (tree (https://github.com/patchew-project/qemu/tree/patchew/20210614104608.212276-1-thanos.makatos@nutanix.com), apply log (/logs/20210614104608.212276-1-thanos.makatos@nutanix.com/git/))

```
git fetch https://github.com/patchew-project/qemu tags/patchew/20210614104608.212276-1-thanos.makatos@nutanix.com
```

```
MAINTAINERS              |    6 +
docs/devel/index.rst     |    1 +
docs/devel/vfio-user.rst | 1810 ++++++++++++++++++++++++++++++++++++++++
3 files changed, 1817 insertions(+)
create mode 100644 docs/devel/vfio-user.rst
```

Expand all

Fold all

[PATCH v9] introduce vfio-user protocol specification

Posted by **Thanos Makatos** 6 months, 1 week ago

```
This patch introduces the vfio-user protocol specification (formerly
known as VFIO-over-socket), which is designed to allow devices to be
emulated outside QEMU, in a separate process. vfio-user reuses the
existing VFIO defines, structs and concepts.

It has been earlier discussed as an RFC in:
"RFC: use VFIO over a UNIX domain socket to implement device offloading"

Signed-off-by: John G Johnson <john.g.johnson@oracle.com>
Signed-off-by: Thanos Makatos <thanos.makatos@nutanix.com>
Signed-off-by: John Levon <john.levon@nutanix.com>

---

Changed since v1:
  * fix coding style issues
  * update MAINTAINERS for VFIO-over-socket
  * add vfio-over-socket to ToC

Changed since v2:
  * fix whitespace

Changed since v3:
  * rename protocol to vfio-user
  * add table of contents
  * fix Unicode problems
  * fix typos and various reStructuredText issues
  * various stylistic improvements
  * add backend program conventions
  * rewrite part of intro, drop QEMU-specific stuff
  * drop QEMU-specific paragraph about implementation
  * explain that passing of FDs isn't necessary
  * minor improvements in the VFIO section
  * various text substitutions for the sake of consistency
  * drop paragraph about client and server, already explained in
  * intro
  * drop device ID
  * drop type from version
  * elaborate on request concurrency
  * convert some inessential paragraphs into notes
  * explain why some existing VFIO defines cannot be reused
  * explain how to make changes to the protocol
  * improve text of DMA map
  * reword comment about existing VFIO commands
  * add reference to Version section
  * reset device on disconnection
  * reword live migration section
  * replace sys/vfio.h with linux/vfio.h
  * drop reference to iovec
  * use argz the same way it is used in VFIO
  * add type field in header for clarity

Changed since v4:
  * introduce support for live migration as defined in
  * include/uapi/linux/vfio.h
  * introduce 'max_fds' and 'migration' capabilities:
  * remove 'index' from VFIO_USER_DEVICE_GET_IRQ_INFO
  * fix minor typos and reworded some text for clarity
```

```
Changed since v5:
  * fix minor typos
  * separate VFIO_USER_DMA_MAP and VFIO_USER_DMA_UNMAP
  * clarify meaning of VFIO bitmap size field
  * move version major/minor outside JSON
  * client proposes version first
  * make Errno optional in message header
  * clarification about message ID uniqueness
  * clarify that server->client request can appear in between
    client->server request/reply

Changed since v6:
  * put JSON strings in double quotes
  * clarify reply behavior on error
  * introduce max message size capability
  * clarify semantics when failing to map multiple DMA regions in a
    single command

Changed since v7:
  * client proposes version instead of server
  * support ioeventfd and ioregionfd for unmapped regions
  * reword struct vfio_bitmap for clarity
  * clarify use of argsz in VFIO device info
  * allow individual IRQs to be disabled

Changed since v8:
 * Improvements
   - various spelling/typo corrections and stylistic fixes
   - document specific reply/response for each command
   - add support for ioeventfd and ioregionfd
   - rewrite dirty page tracking text
   - explain latency implications of device reset
   - expand on device disconnection behavior
   - expand on trust between client/server
   - specify version encoding
 * Major API changes
   - use argsz consistently and the same way VFIO does
   - max_msg_size requirement removed; new max_data_xfer_size parameter for
     socket DMA read/writes
   - removed VM IRQ message
   - DMA:
     * use VFIO's format for DMA map/unmap
     * only allow one region per map/unmap command
     * don't abuse flags
     * implicitly regard DMA region as mappable if FD is passed
     * allow only one dirty bitmap per unmap
     * unmap: require region to match previous mapping
---
 MAINTAINERS              |    6 +
 docs/devel/index.rst     |    1 +
 docs/devel/vfio-user.rst | 1810 ++++++++++++++++++++++++++++++++++++++++++++++
 3 files changed, 1817 insertions(+)
 create mode 100644 docs/devel/vfio-user.rst

diff --git a/MAINTAINERS b/MAINTAINERS
index 7d9cd29042..20dd886e5b 100644
--- a/MAINTAINERS
+++ b/MAINTAINERS
@@ -1843,6 +1843,12 @@ F: hw/vfio/ap.c
 F: docs/system/s390x/vfio-ap.rst
```

```
 L: qemu-s390x@nongnu.org

+vfio-user
+M: John G Johnson <john.g.johnson@oracle.com>
+M: Thanos Makatos <thanos.makatos@nutanix.com>
+S: Supported
+F: docs/devel/vfio-user.rst
+
 vhost
 M: Michael S. Tsirkin <mst@redhat.com>
 S: Supported
diff --git a/docs/devel/index.rst b/docs/devel/index.rst
index 791925dcda..190d0e170d 100644
--- a/docs/devel/index.rst
+++ b/docs/devel/index.rst
@@ -44,3 +44,4 @@ Contents:
    block-coroutine-wrapper
    multi-process
    ebpf_rss
+   vfio-user
diff --git a/docs/devel/vfio-user.rst b/docs/devel/vfio-user.rst
new file mode 100644
index 0000000000..1f6b3d41a2
--- /dev/null
+++ b/docs/devel/vfio-user.rst
@@ -0,0 +1,1810 @@
+.. include:: <isonum.txt>
+
+*******************************
+vfio-user Protocol Specification
+*******************************
+
+--------------
+Version_ 0.9.1
+--------------
+
+.. contents:: Table of Contents
+
+Introduction
+============
+vfio-user is a protocol that allows a device to be emulated in a separate
+process outside of a Virtual Machine Monitor (VMM). vfio-user devices consist
+of a generic VFIO device type, living inside the VMM, which we call the client,
+and the core device implementation, living outside the VMM, which we call the
+server.
+
+The vfio-user specification is partly based on the
+`Linux VFIO ioctl interface <https://www.kernel.org/doc/html/latest/driver-api/vfio.h
tml>`_.
+
+VFIO is a mature and stable API, backed by an extensively used framework. The
+existing VFIO client implementation in QEMU (``qemu/hw/vfio/``) can be largely
+re-used, though there is nothing in this specification that requires that
+particular implementation. None of the VFIO kernel modules are required for
+supporting the protocol, on either the client or server side. Some source
+definitions in VFIO are re-used for vfio-user.
+
+The main idea is to allow a virtual device to function in a separate process in
+the same host over a UNIX domain socket. A UNIX domain socket (``AF_UNIX``) is
+chosen because file descriptors can be trivially sent over it, which in turn
```

+allows:
+
+* Sharing of client memory for DMA with the server.
+* Sharing of server memory with the client for fast MMIO.
+* Efficient sharing of eventfd's for triggering interrupts.
+
+Other socket types could be used which allow the server to run in a separate
+guest in the same host (``AF_VSOCK``) or remotely (``AF_INET``). Theoretically
+the underlying transport does not necessarily have to be a socket, however we do
+not examine such alternatives. In this protocol version we focus on using a UNIX
+domain socket and introduce basic support for the other two types of sockets
+without considering performance implications.
+
+While passing of file descriptors is desirable for performance reasons, support
+is not necessary for either the client or the server in order to implement the
+protocol. There is always an in-band, message-passing fall back mechanism.
+
+Overview
+========
+
+VFIO is a framework that allows a physical device to be securely passed through
+to a user space process; the device-specific kernel driver does not drive the
+device at all. Typically, the user space process is a VMM and the device is
+passed through to it in order to achieve high performance. VFIO provides an API
+and the required functionality in the kernel. QEMU has adopted VFIO to allow a
+guest to directly access physical devices, instead of emulating them in
+software.
+
+vfio-user reuses the core VFIO concepts defined in its API, but implements them
+as messages to be sent over a socket. It does not change the kernel-based VFIO
+in any way, in fact none of the VFIO kernel modules need to be loaded to use
+vfio-user. It is also possible for the client to concurrently use the current
+kernel-based VFIO for one device, and vfio-user for another device.
+
+VFIO Device Model
+-----------------
+
+A device under VFIO presents a standard interface to the user process. Many of
+the VFIO operations in the existing interface use the ``ioctl()`` system call, and
+references to the existing interface are called the ``ioctl()`` implementation in
+this document.
+
+The following sections describe the set of messages that implement the vfio-user
+interface over a socket. In many cases, the messages are analogous to data
+structures used in the ``ioctl()`` implementation. Messages derived from the
+``ioctl()`` will have a name derived from the ``ioctl()`` command name. E.g., the
+``VFIO_DEVICE_GET_INFO`` ``ioctl()`` command becomes a
+``VFIO_USER_DEVICE_GET_INFO`` message. The purpose of this reuse is to share as
+much code as feasible with the ``ioctl()`` implementation``.
+
+Connection Initiation
+^^^^^^^^^^^^^^^^^^^^^
+
+After the client connects to the server, the initial client message is
+``VFIO_USER_VERSION`` to propose a protocol version and set of capabilities to
+apply to the session. The server replies with a compatible version and set of
+capabilities it supports, or closes the connection if it cannot support the
+advertised version.
+
+Device Information

+^^^^^^^^^^^^^^^^^^
+
+The client uses a ``VFIO_USER_DEVICE_GET_INFO`` message to query the server for
+information about the device. This information includes:
+
+* The device type and whether it supports reset (``VFIO_DEVICE_FLAGS_``),
+* the number of device regions, and
+* the device presents to the client the number of interrupt types the device
+ supports.
+
+Region Information
+^^^^^^^^^^^^^^^^^^
+
+The client uses ``VFIO_USER_DEVICE_GET_REGION_INFO`` messages to query the
+server for information about the device's regions. This information describes:
+
+* Read and write permissions, whether it can be memory mapped, and whether it
+ supports additional capabilities (``VFIO_REGION_INFO_CAP_``).
+* Region index, size, and offset.
+
+When a device region can be mapped by the client, the server provides a file
+descriptor which the client can ``mmap()``. The server is responsible for
+polling for client updates to memory mapped regions.
+
+Region Capabilities
+"""""""""""""""""""
+
+Some regions have additional capabilities that cannot be described adequately
+by the region info data structure. These capabilities are returned in the
+region info reply in a list similar to PCI capabilities in a PCI device's
+configuration space.
+
+Sparse Regions
+""""""""""""""
+A region can be memory-mappable in whole or in part. When only a subset of a
+region can be mapped by the client, a ``VFIO_REGION_INFO_CAP_SPARSE_MMAP``
+capability is included in the region info reply. This capability describes
+which portions can be mapped by the client.
+
+.. Note::
+   For example, in a virtual NVMe controller, sparse regions can be used so
+   that accesses to the NVMe registers (found in the beginning of BAR0) are
+   trapped (an infrequent event), while allowing direct access to the doorbells
+   (an extremely frequent event as every I/O submission requires a write to
+   BAR0), found in the next page after the NVMe registers in BAR0.
+
+Device-Specific Regions
+"""""""""""""""""""""""
+
+A device can define regions additional to the standard ones (e.g. PCI indexes
+0-8). This is achieved by including a ``VFIO_REGION_INFO_CAP_TYPE`` capability
+in the region info reply of a device-specific region. Such regions are reflected
+in ``struct vfio_user_device_info.num_regions``. Thus, for PCI devices this
+value can be equal to, or higher than, ``VFIO_PCI_NUM_REGIONS``.
+
+Region I/O via file descriptors
+-------------------------------
+
+For unmapped regions, region I/O from the client is done via
+``VFIO_USER_REGION_READ/WRITE``. As an optimization, ioeventfds or ioregionfds

+may be configured for sub-regions of some regions. A client may request
+information on these sub-regions via ``VFIO_USER_DEVICE_GET_REGION_IO_FDS``; by
+configuring the returned file descriptors as ioeventfds or ioregionfds, the
+server can be directly notified of I/O (for example, by KVM) without taking a
+trip through the client.
+
+Interrupts
+^^^^^^^^^^
+
+The client uses ``VFIO_USER_DEVICE_GET_IRQ_INFO`` messages to query the server
+for the device's interrupt types. The interrupt types are specific to the bus
+the device is attached to, and the client is expected to know the capabilities
+of each interrupt type. The server can signal an interrupt by directly injecting
+interrupts into the guest via an event file descriptor. The client configures
+how the server signals an interrupt with ``VFIO_USER_SET_IRQS`` messages.
+
+Device Read and Write
+^^^^^^^^^^^^^^^^^^^^^
+
+When the guest executes load or store operations to an unmapped device region,
+the client forwards these operations to the server with
+``VFIO_USER_REGION_READ`` or ``VFIO_USER_REGION_WRITE`` messages. The server
+will reply with data from the device on read operations or an acknowledgement on
+write operations. See `Read and Write Operations`_.
+
+Client memory access
+--------------------
+
+The client uses ``VFIO_USER_DMA_MAP`` and ``VFIO_USER_DMA_UNMAP`` messages to
+inform the server of the valid DMA ranges that the server can access on behalf
+of a device (typically, VM guest memory). DMA memory may be accessed by the
+server via ``VFIO_USER_DMA_READ`` and ``VFIO_USER_DMA_WRITE`` messages over the
+socket. In this case, the "DMA" part of the naming is a misnomer.
+
+Actual direct memory access of client memory from the server is possible if the
+client provides file descriptors the server can ``mmap()``. Note that ``mmap()``
+privileges cannot be revoked by the client, therefore file descriptors should
+only be exported in environments where the client trusts the server not to
+corrupt guest memory.
+
+See `Read and Write Operations`_.
+
+Client/server interactions
+==========================
+
+Socket
+------
+
+A server can serve:
+
+1) one or more clients, and/or
+2) one or more virtual devices, belonging to one or more clients.
+
+The current protocol specification requires a dedicated socket per
+client/server connection. It is a server-side implementation detail whether a
+single server handles multiple virtual devices from the same or multiple
+clients. The location of the socket is implementation-specific. Multiplexing
+clients, devices, and servers over the same socket is not supported in this
+version of the protocol.
+

+Authentication
+--------------
+
+For ``AF_UNIX``, we rely on OS mandatory access controls on the socket files,
+therefore it is up to the management layer to set up the socket as required.
+Socket types than span guests or hosts will require a proper authentication
+mechanism. Defining that mechanism is deferred to a future version of the
+protocol.
+
+Command Concurrency
+-------------------
+
+A client may pipeline multiple commands without waiting for previous command
+replies. The server will process commands in the order they are received. A
+consequence of this is if a client issues a command with the *No_reply* bit,
+then subsequently issues a command without *No_reply*, the older command will
+have been processed before the reply to the younger command is sent by the
+server. The client must be aware of the device's capability to process
+concurrent commands if pipelining is used. For example, pipelining allows
+multiple client threads to concurrently access device regions; the client must
+ensure these accesses obey device semantics.
+
+An example is a frame buffer device, where the device may allow concurrent
+access to different areas of video memory, but may have indeterminate behavior
+if concurrent accesses are performed to command or status registers.
+
+Note that unrelated messages sent from the server to the client can appear in
+between a client to server request/reply and vice versa.
+
+Implementers should be prepared for certain commands to exhibit potentially
+unbounded latencies. For example, ``VFIO_USER_DEVICE_RESET`` may take an
+arbitrarily long time to complete; clients should take care not to block
+unnecessarily.
+
+Socket Disconnection Behavior
+-----------------------------
+The server and the client can disconnect from each other, either intentionally
+or unexpectedly. Both the client and the server need to know how to handle such
+events.
+
+Server Disconnection
+^^^^^^^^^^^^^^^^^^^^
+A server disconnecting from the client may indicate that:
+
+1) A virtual device has been restarted, either intentionally (e.g. because of a
+   device update) or unintentionally (e.g. because of a crash).
+2) A virtual device has been shut down with no intention to be restarted.
+
+It is impossible for the client to know whether or not a failure is
+intermittent or innocuous and should be retried, therefore the client should
+reset the VFIO device when it detects the socket has been disconnected.
+Error recovery will be driven by the guest's device error handling
+behavior.
+
+Client Disconnection
+^^^^^^^^^^^^^^^^^^^^
+The client disconnecting from the server primarily means that the client
+has exited. Currently, this means that the guest is shut down so the device is
+no longer needed therefore the server can automatically exit. However, there
+can be cases where a client disconnection should not result in a server exit:

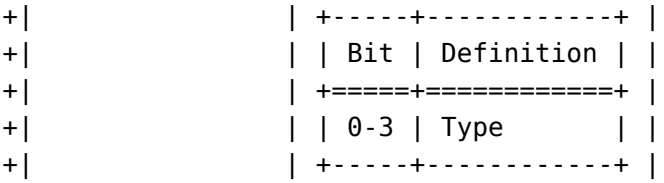
+
+1) A single server serving multiple clients.
+2) A multi-process QEMU upgrading itself step by step, which is not yet
+   implemented.
+
+Therefore in order for the protocol to be forward compatible, the server should
+respond to a client disconnection as follows:
+
+ - all client memory regions are unmapped and cleaned up (including closing any
+   passed file descriptors)
+ - all IRQ file descriptors passed from the old client are closed
+ - the device state should otherwise be retained
+
+The expectation is that when a client reconnects, it will re-establish IRQ and
+client memory mappings.
+
+If anything happens to the client (such as qemu really did exit), the control
+stack will know about it and can clean up resources accordingly.
+
+Security Considerations
+-----------------------
+
+Speaking generally, vfio-user clients should not trust servers, and vice versa.
+Standard tools and mechanisms should be used on both sides to validate input and
+prevent against denial of service scenarios, buffer overflow, etc.
+
+Request Retry and Response Timeout
+----------------------------------
+A failed command is a command that has been successfully sent and has been
+responded to with an error code. Failure to send the command in the first place
+(e.g. because the socket is disconnected) is a different type of error examined
+earlier in the disconnect section.
+
+.. Note::
+   QEMU's VFIO retries certain operations if they fail. While this makes sense
+   for real HW, we don't know for sure whether it makes sense for virtual
+   devices.
+
+Defining a retry and timeout scheme is deferred to a future version of the
+protocol.
+
+Message sizes
+-------------
+
+Some requests have an ``argsz`` field. In a request, it defines the maximum
+expected reply payload size, which should be at least the size of the fixed
+reply payload headers defined here. The *request* payload size is defined by the
+usual ``msg_size`` field in the header, not the ``argsz`` field.
+
+In a reply, the server sets ``argsz`` field to the size needed for a full
+payload size. This may be less than the requested maximum size. This may be
+larger than the requested maximum size: in that case, the full payload is not
+included in the reply, but the ``argsz`` field in the reply indicates the needed
+size, allowing a client to allocate a larger buffer for holding the reply before
+trying again.
+
+In addition, during negotiation (see `Version`_), the client and server may
+each specify a ``max_data_xfer_size`` value; this defines the maximum data that
+may be read or written via one of the ``VFIO_USER_DMA/REGION_READ/WRITE``
+messages; see `Read and Write Operations`_.

+
+Protocol Specification
+======================
+
+To distinguish from the base VFIO symbols, all vfio-user symbols are prefixed
+with ``vfio_user`` or ``VFIO_USER``. In this revision, all data is in the
+little-endian format, although this may be relaxed in future revisions in cases
+where the client and server are both big-endian.
+
+Unless otherwise specified, all sizes should be presumed to be in bytes.
+
+.. _Commands:
+
+Commands
+--------
+The following table lists the VFIO message command IDs, and whether the
+message command is sent from the client or the server.
+

Name	Command	Request Direction
``VFIO_USER_VERSION``	1	client -> server
``VFIO_USER_DMA_MAP``	2	client -> server
``VFIO_USER_DMA_UNMAP``	3	client -> server
``VFIO_USER_DEVICE_GET_INFO``	4	client -> server
``VFIO_USER_DEVICE_GET_REGION_INFO``	5	client -> server
``VFIO_USER_DEVICE_GET_REGION_IO_FDS``	6	client -> server
``VFIO_USER_DEVICE_GET_IRQ_INFO``	7	client -> server
``VFIO_USER_DEVICE_SET_IRQS``	8	client -> server
``VFIO_USER_REGION_READ``	9	client -> server
``VFIO_USER_REGION_WRITE``	10	client -> server
``VFIO_USER_DMA_READ``	11	server -> client
``VFIO_USER_DMA_WRITE``	12	server -> client
``VFIO_USER_DEVICE_RESET``	13	client -> server
``VFIO_USER_DIRTY_PAGES``	14	client -> server

+
+Header
+------
+
+All messages, both command messages and reply messages, are preceded by a
+16-byte header that contains basic information about the message. The header is
+followed by message-specific data described in the sections below.
+

Name	Offset	Size
Message ID	0	2
Command	2	2
Message size	4	4
Flags	8	4



```
+|                | | 4   | No_reply   | |
+|                | +-----+------------+ |
+|                | | 5   | Error      | |
+|                | +-----+------------+ |
++-----------------+--------+-------------+
+| Error          | 12     | 4           |
++-----------------+--------+-------------+
+| <message data> | 16     | variable    |
++-----------------+--------+-------------+
+
+* *Message ID* identifies the message, and is echoed in the command's reply
+  message. Message IDs belong entirely to the sender, can be re-used (even
+  concurrently) and the receiver must not make any assumptions about their
+  uniqueness.
+* *Command* specifies the command to be executed, listed in Commands_. It is
+  also set in the reply header.
+* *Message size* contains the size of the entire message, including the header.
+* *Flags* contains attributes of the message:
+
+  * The *Type* bits indicate the message type.
+
+    *  *Command* (value 0x0) indicates a command message.
+    *  *Reply* (value 0x1) indicates a reply message acknowledging a previous
+       command with the same message ID.
+  * *No Reply* in a command message indicates that no reply is needed for this
+    command.  This is commonly used when multiple commands are sent, and only
+    the last needs acknowledgement.
+  * *Error* in a reply message indicates the command being acknowledged had
+    an error. In this case, the *Error* field will be valid.
+
+* *Error* in a reply message is an optional UNIX errno value. It may be zero
+  even if the Error bit is set in Flags. It is reserved in a command message.
+
+Each command message in Commands_ must be replied to with a reply message,
+unless the message sets the *No Reply* bit.  The reply consists of the header
+with the *Reply* bit set, plus any additional data.
+
+If an error occurs, the reply message must only include the reply header.
+
+As the header is standard in both requests and replies, it is not included in
+the command-specific specifications below; each message definition should be
+appended to the standard header, and the offsets are given from the end of the
+standard header.
+
+``VFIO_USER_VERSION``
+---------------------
+
+.. _Version:
+
+This is the initial message sent by the client after the socket connection is
+established; the same format is used for the server's reply.
+
+Upon establishing a connection, the client must send a ``VFIO_USER_VERSION``
+message proposing a protocol version and a set of capabilities. The server
+compares these with the versions and capabilities it supports and sends a
+``VFIO_USER_VERSION`` reply according to the following rules.
+
+* The major version in the reply must be the same as proposed. If the client
+  does not support the proposed major, it closes the connection.
+* The minor version in the reply must be equal to or less than the minor
```

+ version proposed.
+* The capability list must be a subset of those proposed. If the server
+ requires a capability the client did not include, it closes the connection.
+
+The protocol major version will only change when incompatible protocol changes
+are made, such as changing the message format. The minor version may change
+when compatible changes are made, such as adding new messages or capabilities,
+Both the client and server must support all minor versions less than the
+maximum minor version it supports. E.g., an implementation that supports
+version 1.3 must also support 1.0 through 1.2.
+
+When making a change to this specification, the protocol version number must
+be included in the form "added in version X.Y"
+
+Request
+^^^^^^^
+

Name	Offset	Size
version major	0	2
version minor	2	2
version data	4	variable (including terminating NUL). Optional.

+
+The version data is an optional UTF-8 encoded JSON byte array with the following
+format:
+

Name	Type	Description
capabilities	object	Contains common capabilities that the sender supports. Optional.

+
+Capabilities:
+

Name	Type	Description
max_msg_fds	number	Maximum number of file descriptors that can be received by the sender in one message. Optional. If not specified then the receiver must assume a value of ``1``.
max_data_xfer_size	number	Maximum ``count`` for data transfer messages; see `Read and Write Operations`_. Optional, with a default value of 1048576 bytes.
migration	object	Migration capability parameters. If missing then migration is not supported by the sender.

+
+The migration capability contains the following name/value pairs:
+

Name	Type	Description
pgsize	number	Page size of dirty pages bitmap. The smallest between the client and the server is used.

```
++--------+--------+-----------------------------------------------+
+
+Reply
+^^^^^
+
+The same message format is used in the server's reply with the semantics
+described above.
+
+``VFIO_USER_DMA_MAP``
+---------------------
+
+This command message is sent by the client to the server to inform it of the
+memory regions the server can access. It must be sent before the server can
+perform any DMA to the client. It is normally sent directly after the version
+handshake is completed, but may also occur when memory is added to the client,
+or if the client uses a vIOMMU.
+
+Request
+^^^^^^^
+
+The request payload for this message is a structure of the following format:
+
++-------------+--------+-------------+
+| Name        | Offset | Size        |
++=============+========+=============+
+| argsz       | 0      | 4           |
++-------------+--------+-------------+
+| flags       | 4      | 4           |
++-------------+--------+-------------+
+|             | +-----+------------+ |
+|             | | Bit | Definition | |
+|             | +=====+============+ |
+|             | | 0   | readable   | |
+|             | +-----+------------+ |
+|             | | 1   | writeable  | |
+|             | +-----+------------+ |
++-------------+--------+-------------+
+| offset      | 8      | 8           |
++-------------+--------+-------------+
+| address     | 16     | 8           |
++-------------+--------+-------------+
+| size        | 24     | 8           |
++-------------+--------+-------------+
+
+* *argsz* is the size of the above structure. Note there is no reply payload,
+  so this field differs from other message types.
+* *flags* contains the following region attributes:
+
+  * *readable* indicates that the region can be read from.
+
+  * *writeable* indicates that the region can be written to.
+
+* *offset* is the file offset of the region with respect to the associated file
+  descriptor, or zero if the region is not mappable
+* *address* is the base DMA address of the region.
+* *size* is the size of the region.
+
+This structure is 32 bytes in size, so the message size is 16 + 32 bytes.
+
+If the DMA region being added can be directly mapped by the server, a file
```

+descriptor must be sent as part of the message meta-data. The region can be
+mapped via the mmap() system call. On ``AF_UNIX`` sockets, the file descriptor
+must be passed as ``SCM_RIGHTS`` type ancillary data. Otherwise, if the DMA
+region cannot be directly mapped by the server, no file descriptor must be sent
+as part of the message meta-data and the DMA region can be accessed by the
+server using ``VFIO_USER_DMA_READ`` and ``VFIO_USER_DMA_WRITE`` messages,
+explained in `Read and Write Operations`_. A command to map over an existing
+region must be failed by the server with ``EEXIST`` set in error field in the
+reply.
+
+Reply
+^^^^^
+
+There is no payload in the reply message.
+
+``VFIO_USER_DMA_UNMAP``
+-----------------------
+
+This command message is sent by the client to the server to inform it that a
+DMA region, previously made available via a ``VFIO_USER_DMA_MAP`` command
+message, is no longer available for DMA. It typically occurs when memory is
+subtracted from the client or if the client uses a vIOMMU. The DMA region is
+described by the following structure:
+
+Request
+^^^^^^^
+
+The request payload for this message is a structure of the following format:
+

```
++--------------+--------+-------------------------+
+| Name         | Offset | Size                    |
++==============+========+=========================+
+| argsz        | 0      | 4                       |
++--------------+--------+-------------------------+
+| flags        | 4      | 4                       |
++--------------+--------+-------------------------+
+|              | +-----+----------------------+ |
+|              | | Bit | Definition           | |
+|              | +=====+======================+ |
+|              | | 0   | get dirty page bitmap | |
+|              | +-----+----------------------+ |
++--------------+--------+-------------------------+
+| address      | 8      | 8                       |
++--------------+--------+-------------------------+
+| size         | 16     | 8                       |
++--------------+--------+-------------------------+
```

+
+* *argsz* is the maximum size of the reply payload.
+* *flags* contains the following DMA region attributes:
+
+ * *get dirty page bitmap* indicates that a dirty page bitmap must be
+   populated before unmapping the DMA region. The client must provide a
+   `VFIO Bitmap`_ structure, explained below, immediately following this
+   entry.
+
+* *address* is the base DMA address of the DMA region.
+* *size* is the size of the DMA region.
+
+The address and size of the DMA region being unmapped must match exactly a
+previous mapping. The size of request message depends on whether or not the

+*get dirty page bitmap* bit is set in Flags:
+
+* If not set, the size of the total request message is: 16 + 24.
+
+* If set, the size of the total request message is: 16 + 24 + 16.
+
+.. _VFIO Bitmap:
+
+VFIO Bitmap Format
+""""""""""""""""""
+
++--------+--------+------+
+| Name | Offset | Size |
++========+========+======+
+| pgsize | 0 | 8 |
++--------+--------+------+
+| size | 8 | 8 |
++--------+--------+------+
+
+* *pgsize* is the page size for the bitmap, in bytes.
+* *size* is the size for the bitmap, in bytes, excluding the VFIO bitmap header.
+
+Reply
+^^^^^
+
+Upon receiving a ``VFIO_USER_DMA_UNMAP`` command, if the file descriptor is
+mapped then the server must release all references to that DMA region before
+replying, which potentially includes in-flight DMA transactions.
+
+The server responds with the original DMA entry in the request. If the
+*get dirty page bitmap* bit is set in flags in the request, then
+the server also includes the `VFIO Bitmap`_ structure sent in the request,
+followed by the corresponding dirty page bitmap, where each bit represents
+one page of size *pgsize* in `VFIO Bitmap`_ .
+
+The total size of the total reply message is:
+16 + 24 + (16 + *size* in `VFIO Bitmap`_ if *get dirty page bitmap* is set).
+
+``VFIO_USER_DEVICE_GET_INFO``
+-----------------------------
+
+This command message is sent by the client to the server to query for basic
+information about the device.
+
+Request
+^^^^^^^
+
++-------------+--------+--------------------------+
+| Name | Offset | Size |
++=============+========+==========================+
+| argsz | 0 | 4 |
++-------------+--------+--------------------------+
+| flags | 4 | 4 |
++-------------+--------+--------------------------+
+| | +-----+-------------------------+ |
+| | | Bit | Definition | |
+| | +=====+=========================+ |
+| | | 0 | VFIO_DEVICE_FLAGS_RESET | |
+| | +-----+-------------------------+ |
+| | | 1 | VFIO_DEVICE_FLAGS_PCI | |

```
+|             | +-----+-------------------------+ |
++-------------+--------+--------------------------+
+| num_regions | 8      | 4                        |
++-------------+--------+--------------------------+
+| num_irqs    | 12     | 4                        |
++-------------+--------+--------------------------+
+
+* *argsz* is the maximum size of the reply payload
+* all other fields must be zero.
+
+Reply
+^^^^^
+
++-------------+--------+--------------------------+
+| Name        | Offset | Size                     |
++=============+========+==========================+
+| argsz       | 0      | 4                        |
++-------------+--------+--------------------------+
+| flags       | 4      | 4                        |
++-------------+--------+--------------------------+
+|             | +-----+-------------------------+ |
+|             | | Bit | Definition              | |
+|             | +=====+=========================+ |
+|             | | 0   | VFIO_DEVICE_FLAGS_RESET | |
+|             | +-----+-------------------------+ |
+|             | | 1   | VFIO_DEVICE_FLAGS_PCI   | |
+|             | +-----+-------------------------+ |
++-------------+--------+--------------------------+
+| num_regions | 8      | 4                        |
++-------------+--------+--------------------------+
+| num_irqs    | 12     | 4                        |
++-------------+--------+--------------------------+
+
+* *argsz* is the size required for the full reply payload (16 bytes today)
+* *flags* contains the following device attributes.
+
+  * ``VFIO_DEVICE_FLAGS_RESET`` indicates that the device supports the
+    ``VFIO_USER_DEVICE_RESET`` message.
+  * ``VFIO_DEVICE_FLAGS_PCI`` indicates that the device is a PCI device.
+
+* *num_regions* is the number of memory regions that the device exposes.
+* *num_irqs* is the number of distinct interrupt types that the device supports.
+
+This version of the protocol only supports PCI devices. Additional devices may
+be supported in future versions.
+
+``VFIO_USER_DEVICE_GET_REGION_INFO``
+------------------------------------
+
+This command message is sent by the client to the server to query for
+information about device regions. The VFIO region info structure is defined in
+``<linux/vfio.h>`` (``struct vfio_region_info``).
+
+Request
+^^^^^^^
+
++------------+--------+-----------------------------+
+| Name       | Offset | Size                        |
++============+========+=============================+
+| argsz      | 0      | 4                           |
```

```
++------------+--------+------------------------------+
+| flags      | 4      | 4                            |
++------------+--------+------------------------------+
+| index      | 8      | 4                            |
++------------+--------+------------------------------+
+| cap_offset | 12     | 4                            |
++------------+--------+------------------------------+
+| size       | 16     | 8                            |
++------------+--------+------------------------------+
+| offset     | 24     | 8                            |
++------------+--------+------------------------------+
+
+* *argsz* the maximum size of the reply payload
+* *index* is the index of memory region being queried, it is the only field
+  that is required to be set in the command message.
+* all other fields must be zero.
+
+Reply
+^^^^^
+
++------------+--------+------------------------------+
+| Name       | Offset | Size                         |
++============+========+==============================+
+| argsz      | 0      | 4                            |
++------------+--------+------------------------------+
+| flags      | 4      | 4                            |
++------------+--------+------------------------------+
+|            | +-----+----------------------------+ |
+|            | | Bit | Definition                 | |
+|            | +=====+============================+ |
+|            | | 0   | VFIO_REGION_INFO_FLAG_READ | |
+|            | +-----+----------------------------+ |
+|            | | 1   | VFIO_REGION_INFO_FLAG_WRITE| |
+|            | +-----+----------------------------+ |
+|            | | 2   | VFIO_REGION_INFO_FLAG_MMAP | |
+|            | +-----+----------------------------+ |
+|            | | 3   | VFIO_REGION_INFO_FLAG_CAPS | |
+|            | +-----+----------------------------+ |
++------------+--------+------------------------------+
++------------+--------+------------------------------+
+| index      | 8      | 4                            |
++------------+--------+------------------------------+
+| cap_offset | 12     | 4                            |
++------------+--------+------------------------------+
+| size       | 16     | 8                            |
++------------+--------+------------------------------+
+| offset     | 24     | 8                            |
++------------+--------+------------------------------+
+
+* *argsz* is the size required for the full reply payload (region info structure
+  plus the size of any region capabilities)
+* *flags* are attributes of the region:
+
+  * ``VFIO_REGION_INFO_FLAG_READ`` allows client read access to the region.
+  * ``VFIO_REGION_INFO_FLAG_WRITE`` allows client write access to the region.
+  * ``VFIO_REGION_INFO_FLAG_MMAP`` specifies the client can mmap() the region.
+    When this flag is set, the reply will include a file descriptor in its
+    meta-data. On ``AF_UNIX`` sockets, the file descriptors will be passed as
+    ``SCM_RIGHTS`` type ancillary data.
+  * ``VFIO_REGION_INFO_FLAG_CAPS`` indicates additional capabilities found in the
```

```
+    reply.
+
+* *index* is the index of memory region being queried, it is the only field
+  that is required to be set in the command message.
+* *cap_offset* describes where additional region capabilities can be found.
+  cap_offset is relative to the beginning of the VFIO region info structure.
+  The data structure it points is a VFIO cap header defined in
+  ``<linux/vfio.h>``.
+* *size* is the size of the region.
+* *offset* is the offset that should be given to the mmap() system call for
+  regions with the MMAP attribute. It is also used as the base offset when
+  mapping a VFIO sparse mmap area, described below.
+
+VFIO region capabilities
+""""""""""""""""""""""""
+
+The VFIO region information can also include a capabilities list. This list is
+similar to a PCI capability list - each entry has a common header that
+identifies a capability and where the next capability in the list can be found.
+The VFIO capability header format is defined in ``<linux/vfio.h>`` (``struct
+vfio_info_cap_header``).
+
+VFIO cap header format
+""""""""""""""""""""""
+
++---------+--------+------+
+| Name    | Offset | Size |
++=========+========+======+
+| id      | 0      | 2    |
++---------+--------+------+
+| version | 2      | 2    |
++---------+--------+------+
+| next    | 4      | 4    |
++---------+--------+------+
+
+* *id* is the capability identity.
+* *version* is a capability-specific version number.
+* *next* specifies the offset of the next capability in the capability list. It
+  is relative to the beginning of the VFIO region info structure.
+
+VFIO sparse mmap cap header
+"""""""""""""""""""""""""""
+
++------------------+----------------------------------+
+| Name             | Value                            |
++==================+==================================+
+| id               | VFIO_REGION_INFO_CAP_SPARSE_MMAP |
++------------------+----------------------------------+
+| version          | 0x1                              |
++------------------+----------------------------------+
+| next             | <next>                           |
++------------------+----------------------------------+
+| sparse mmap info | VFIO region info sparse mmap     |
++------------------+----------------------------------+
+
+This capability is defined when only a subrange of the region supports
+direct access by the client via mmap(). The VFIO sparse mmap area is defined in
+``<linux/vfio.h>`` (``struct vfio_region_sparse_mmap_area`` and ``struct
+vfio_region_info_cap_sparse_mmap``).
+
```

+VFIO region info cap sparse mmap
+""""""""""""""""""""""""""""""""
+
++----------+--------+------+
+| Name | Offset | Size |
++==========+========+======+
+| nr_areas | 0 | 4 |
++----------+--------+------+
+| reserved | 4 | 4 |
++----------+--------+------+
+| offset | 8 | 8 |
++----------+--------+------+
+| size | 16 | 9 |
++----------+--------+------+
+| ... | | |
++----------+--------+------+
+
+* *nr_areas* is the number of sparse mmap areas in the region.
+* *offset* and size describe a single area that can be mapped by the client.
+ There will be *nr_areas* pairs of offset and size. The offset will be added to
+ the base offset given in the ``VFIO_USER_DEVICE_GET_REGION_INFO`` to form the
+ offset argument of the subsequent mmap() call.
+
+The VFIO sparse mmap area is defined in ``<linux/vfio.h>`` (``struct
+vfio_region_info_cap_sparse_mmap``).
+
+VFIO region type cap header
+"""""""""""""""""""""""""""
+
++------------------+---------------------------+
+| Name | Value |
++==================+===========================+
+| id | VFIO_REGION_INFO_CAP_TYPE |
++------------------+---------------------------+
+| version | 0x1 |
++------------------+---------------------------+
+| next | <next> |
++------------------+---------------------------+
+| region info type | VFIO region info type |
++------------------+---------------------------+
+
+This capability is defined when a region is specific to the device.
+
+VFIO region info type cap
+"""""""""""""""""""""""""
+
+The VFIO region info type is defined in ``<linux/vfio.h>``
+(``struct vfio_region_info_cap_type``).
+
++---------+--------+------+
+| Name | Offset | Size |
++=========+========+======+
+| type | 0 | 4 |
++---------+--------+------+
+| subtype | 4 | 4 |
++---------+--------+------+
+
+The only device-specific region type and subtype supported by vfio-user is
+``VFIO_REGION_TYPE_MIGRATION`` (3) and ``VFIO_REGION_SUBTYPE_MIGRATION`` (1).
+

+``VFIO_USER_DEVICE_GET_REGION_IO_FDS``
+--------------------------------------
+
+Clients can access regions via ``VFIO_USER_REGION_READ/WRITE`` or, if provided, by
+``mmap()`` of a file descriptor provided by the server.
+
+``VFIO_USER_DEVICE_GET_REGION_IO_FDS`` provides an alternative access mechanism via
+file descriptors. This is an optional feature intended for performance
+improvements where an underlying sub-system (such as KVM) supports communication
+across such file descriptors to the vfio-user server, without needing to
+round-trip through the client.
+
+The server returns an array of sub-regions for the requested region. Each
+sub-region describes a span (offset and size) of a region, along with the
+requested file descriptor notification mechanism to use. Each sub-region in the
+response message may choose to use a different method, as defined below. The
+two mechanisms supported in this specification are ioeventfds and ioregionfds.
+
+The server in addition returns a file descriptor in the ancillary data; clients
+are expected to configure each sub-region's file descriptor with the requested
+notification method. For example, a client could configure KVM with the
+requested ioeventfd via a ``KVM_IOEVENTFD`` ``ioctl()``.
+
+Request
+^^^^^^^
+
++-------------+--------+------+
+| Name | Offset | Size |
++=============+========+======+
+| argsz | 0 | 4 |
++-------------+--------+------+
+| flags | 4 | 4 |
++-------------+--------+------+
+| index | 8 | 4 |
++-------------+--------+------+
+| count | 12 | 4 |
++-------------+--------+------+
+
+* *argsz* the maximum size of the reply payload
+* *index* is the index of memory region being queried
+* all other fields must be zero
+
+The client must set ``flags`` to zero and specify the region being queried in
+the ``index``.
+
+Reply
+^^^^^
+
+-------------+--------+------+
+| Name | Offset | Size |
++=============+========+======+
+| argsz | 0 | 4 |
++-------------+--------+------+
+| flags | 4 | 4 |
++-------------+--------+------+
+| index | 8 | 4 |
++-------------+--------+------+
+| count | 12 | 4 |
++-------------+--------+------+
+| sub-regions | 16 | ... |

```
++------------+--------+------+
+
+* *argsz* is the size of the region IO FD info structure plus the
+  total size of the sub-region array. Thus, each array entry "i" is at offset
+  i * ((argsz - 32) / count). Note that currently this is 40 bytes for both IO
+  FD types, but this is not to be relied on. As elsewhere, this indicates the
+  full reply payload size needed.
+* *flags* must be zero
+* *index* is the index of memory region being queried
+* *count* is the number of sub-regions in the array
+* *sub-regions* is the array of Sub-Region IO FD info structures
+
+The reply message will additionally include at least one file descriptor in the
+ancillary data. Note that more than one sub-region may share the same file
+descriptor.
+
+Note that it is the client's responsibility to verify the requested values (for
+example, that the requested offset does not exceed the region's bounds).
+
+Each sub-region given in the response has one of two possible structures,
+depending whether *type* is ``VFIO_USER_IO_FD_TYPE_IOEVENTFD`` or
+``VFIO_USER_IO_FD_TYPE_IOREGIONFD``:
+
+Sub-Region IO FD info format (ioeventfd)
+""""""""""""""""""""""""""""""""""""""""
+
++-----------+--------+------+
+| Name      | Offset | Size |
++===========+========+======+
+| offset    | 0      | 8    |
++-----------+--------+------+
+| size      | 8      | 8    |
++-----------+--------+------+
+| fd_index  | 16     | 4    |
++-----------+--------+------+
+| type      | 20     | 4    |
++-----------+--------+------+
+| flags     | 24     | 4    |
++-----------+--------+------+
+| padding   | 28     | 4    |
++-----------+--------+------+
+| datamatch | 32     | 8    |
++-----------+--------+------+
+
+* *offset* is the offset of the start of the sub-region within the region
+  requested ("physical address offset" for the region)
+* *size* is the length of the sub-region. This may be zero if the access size is
+  not relevant, which may allow for optimizations
+* *fd_index* is the index in the ancillary data of the FD to use for ioeventfd
+  notification; it may be shared.
+* *type* is ``VFIO_USER_IO_FD_TYPE_IOEVENTFD``
+* *flags* is any of:
+
+  * ``KVM_IOEVENTFD_FLAG_DATAMATCH``
+  * ``KVM_IOEVENTFD_FLAG_PIO``
+  * ``KVM_IOEVENTFD_FLAG_VIRTIO_CCW_NOTIFY`` (FIXME: makes sense?)
+
+* *datamatch* is the datamatch value if needed
+
+See https://www.kernel.org/doc/Documentation/virtual/kvm/api.txt, *4.59
```

+KVM_IOEVENTFD* for further context on the ioeventfd-specific fields.
+
+Sub-Region IO FD info format (ioregionfd)
+""
+
++-----------+--------+------+
+| Name | Offset | Size |
++===========+========+======+
+| offset | 0 | 8 |
++-----------+--------+------+
+| size | 8 | 8 |
++-----------+--------+------+
+| fd_index | 16 | 4 |
++-----------+--------+------+
+| type | 20 | 4 |
++-----------+--------+------+
+| flags | 24 | 4 |
++-----------+--------+------+
+| padding | 28 | 4 |
++-----------+--------+------+
+| user_data | 32 | 8 |
++-----------+--------+------+
+
+* *offset* is the offset of the start of the sub-region within the region
+ requested ("physical address offset" for the region)
+* *size* is the length of the sub-region. This may be zero if the access size is
+ not relevant, which may allow for optimizations; ``KVM_IOREGION_POSTED_WRITES``
+ must be set in *flags* in this case
+* *fd_index* is the index in the ancillary data of the FD to use for ioregionfd
+ messages; it may be shared
+* *type* is ``VFIO_USER_IO_FD_TYPE_IOREGIONFD``
+* *flags* is any of:
+
+ * ``KVM_IOREGION_PIO``
+ * ``KVM_IOREGION_POSTED_WRITES``
+
+* *user_data* is an opaque value passed back to the server via a message on the
+ file descriptor
+
+For further information on the ioregionfd-specific fields, see:
+https://lore.kernel.org/kvm/cover.1613828726.git.eafanasova@gmail.com/
+
+(FIXME: update with final API docs.)
+
+``VFIO_USER_DEVICE_GET_IRQ_INFO``
+---------------------------------
+
+This command message is sent by the client to the server to query for
+information about device interrupt types. The VFIO IRQ info structure is
+defined in ``<linux/vfio.h>`` (``struct vfio_irq_info``).
+
+Request
+^^^^^^^
+
++-------+--------+--------------------------+
+| Name | Offset | Size |
++=======+========+==========================+
+| argsz | 0 | 4 |
++-------+--------+--------------------------+
+| flags | 4 | 4 |

```
++-------+--------+---------------------------+
+|       | +-----+--------------------------+ |
+|       | | Bit | Definition               | |
+|       | +=====+==========================+ |
+|       | | 0   | VFIO_IRQ_INFO_EVENTFD    | |
+|       | +-----+--------------------------+ |
+|       | | 1   | VFIO_IRQ_INFO_MASKABLE   | |
+|       | +-----+--------------------------+ |
+|       | | 2   | VFIO_IRQ_INFO_AUTOMASKED | |
+|       | +-----+--------------------------+ |
+|       | | 3   | VFIO_IRQ_INFO_NORESIZE   | |
+|       | +-----+--------------------------+ |
++-------+--------+---------------------------+
+| index | 8      | 4                         |
++-------+--------+---------------------------+
+| count | 12     | 4                         |
++-------+--------+---------------------------+
+
+* *argsz* is the maximum size of the reply payload (16 bytes today)
+* index is the index of IRQ type being queried (e.g. ``VFIO_PCI_MSIX_IRQ_INDEX``)
+* all other fields must be zero
+
+Reply
+^^^^^
+
++-------+--------+---------------------------+
+| Name  | Offset | Size                      |
++=======+========+===========================+
+| argsz | 0      | 4                         |
++-------+--------+---------------------------+
+| flags | 4      | 4                         |
++-------+--------+---------------------------+
+|       | +-----+--------------------------+ |
+|       | | Bit | Definition               | |
+|       | +=====+==========================+ |
+|       | | 0   | VFIO_IRQ_INFO_EVENTFD    | |
+|       | +-----+--------------------------+ |
+|       | | 1   | VFIO_IRQ_INFO_MASKABLE   | |
+|       | +-----+--------------------------+ |
+|       | | 2   | VFIO_IRQ_INFO_AUTOMASKED | |
+|       | +-----+--------------------------+ |
+|       | | 3   | VFIO_IRQ_INFO_NORESIZE   | |
+|       | +-----+--------------------------+ |
++-------+--------+---------------------------+
+| index | 8      | 4                         |
++-------+--------+---------------------------+
+| count | 12     | 4                         |
++-------+--------+---------------------------+
+
+* *argsz* is the size required for the full reply payload (16 bytes today)
+* *flags* defines IRQ attributes:
+
+  * ``VFIO_IRQ_INFO_EVENTFD`` indicates the IRQ type can support server eventfd
+    signalling.
+  * ``VFIO_IRQ_INFO_MASKABLE`` indicates that the IRQ type supports the ``MASK``
+    and ``UNMASK`` actions in a ``VFIO_USER_DEVICE_SET_IRQS`` message.
+  * ``VFIO_IRQ_INFO_AUTOMASKED`` indicates the IRQ type masks itself after being
+    triggered, and the client must send an ``UNMASK`` action to receive new
+    interrupts.
+  * ``VFIO_IRQ_INFO_NORESIZE`` indicates ``VFIO_USER_SET_IRQS`` operations setup
```

+   interrupts as a set, and new sub-indexes cannot be enabled without disabling
+   the entire type.
+* index is the index of IRQ type being queried
+* count describes the number of interrupts of the queried type.
+
+``VFIO_USER_DEVICE_SET_IRQS``
+-----------------------------
+
+This command message is sent by the client to the server to set actions for
+device interrupt types. The VFIO IRQ set structure is defined in
+``<linux/vfio.h>`` (``struct vfio_irq_set``).
+
+Request
+^^^^^^^
+

Name	Offset	Size	
argsz	0	4	
flags	4	4	
		Bit	Definition
		0	VFIO_IRQ_SET_DATA_NONE
		1	VFIO_IRQ_SET_DATA_BOOL
		2	VFIO_IRQ_SET_DATA_EVENTFD
		3	VFIO_IRQ_SET_ACTION_MASK
		4	VFIO_IRQ_SET_ACTION_UNMASK
		5	VFIO_IRQ_SET_ACTION_TRIGGER
index	8	4	
start	12	4	
count	16	4	
data	20	variable	

+
+* *argsz* is the size of the VFIO IRQ set request payload, including any *data*
+ field. Note there is no reply payload, so this field differs from other
+ message types.
+* *flags* defines the action performed on the interrupt range. The ``DATA``
+ flags describe the data field sent in the message; the ``ACTION`` flags
+ describe the action to be performed. The flags are mutually exclusive for
+ both sets.
+
+ * ``VFIO_IRQ_SET_DATA_NONE`` indicates there is no data field in the command.
+   The action is performed unconditionally.
+ * ``VFIO_IRQ_SET_DATA_BOOL`` indicates the data field is an array of boolean
+   bytes. The action is performed if the corresponding boolean is true.
+ * ``VFIO_IRQ_SET_DATA_EVENTFD`` indicates an array of event file descriptors

```
+    was sent in the message meta-data. These descriptors will be signalled when
+    the action defined by the action flags occurs. In ``AF_UNIX`` sockets, the
+    descriptors are sent as ``SCM_RIGHTS`` type ancillary data.
+    If no file descriptors are provided, this de-assigns the specified
+    previously configured interrupts.
+  * ``VFIO_IRQ_SET_ACTION_MASK`` indicates a masking event. It can be used with
+    ``VFIO_IRQ_SET_DATA_BOOL`` or ``VFIO_IRQ_SET_DATA_NONE`` to mask an interrupt,
+    or with ``VFIO_IRQ_SET_DATA_EVENTFD`` to generate an event when the guest masks
+    the interrupt.
+  * ``VFIO_IRQ_SET_ACTION_UNMASK`` indicates an unmasking event. It can be used
+    with ``VFIO_IRQ_SET_DATA_BOOL`` or ``VFIO_IRQ_SET_DATA_NONE`` to unmask an
+    interrupt, or with ``VFIO_IRQ_SET_DATA_EVENTFD`` to generate an event when the
+    guest unmasks the interrupt.
+  * ``VFIO_IRQ_SET_ACTION_TRIGGER`` indicates a triggering event. It can be used
+    with ``VFIO_IRQ_SET_DATA_BOOL`` or ``VFIO_IRQ_SET_DATA_NONE`` to trigger an
+    interrupt, or with ``VFIO_IRQ_SET_DATA_EVENTFD`` to generate an event when the
+    server triggers the interrupt.
+
+* *index* is the index of IRQ type being setup.
+* *start* is the start of the sub-index being set.
+* *count* describes the number of sub-indexes being set. As a special case, a
+  count (and start) of 0, with data flags of ``VFIO_IRQ_SET_DATA_NONE`` disables
+  all interrupts of the index.
+* *data* is an optional field included when the
+  ``VFIO_IRQ_SET_DATA_BOOL`` flag is present. It contains an array of booleans
+  that specify whether the action is to be performed on the corresponding
+  index. It's used when the action is only performed on a subset of the range
+  specified.
+
+Not all interrupt types support every combination of data and action flags.
+The client must know the capabilities of the device and IRQ index before it
+sends a ``VFIO_USER_DEVICE_SET_IRQ`` message.
+
+In typical operation, a specific IRQ may operate as follows:
+
+1. The client sends a ``VFIO_USER_DEVICE_SET_IRQ`` message with
+   ``flags=(VFIO_IRQ_SET_DATA_EVENTFD|VFIO_IRQ_SET_ACTION_TRIGGER)`` along
+   with an eventfd. This associates the IRQ with a particular eventfd on the
+   server side.
+
+#. The client may send a ``VFIO_USER_DEVICE_SET_IRQ`` message with
+   ``flags=(VFIO_IRQ_SET_DATA_EVENTFD|VFIO_IRQ_SET_ACTION_MASK/UNMASK)`` along
+   with another eventfd. This associates the given eventfd with the
+   mask/unmask state on the server side.
+
+#. The server may trigger the IRQ by writing 1 to the eventfd.
+
+#. The server may mask/unmask an IRQ which will write 1 to the corresponding
+   mask/unmask eventfd, if there is one.
+
+5. A client may trigger a device IRQ itself, by sending a
+   ``VFIO_USER_DEVICE_SET_IRQ`` message with
+   ``flags=(VFIO_IRQ_SET_DATA_NONE/BOOL|VFIO_IRQ_SET_ACTION_TRIGGER)``.
+
+6. A client may mask or unmask the IRQ, by sending a
+   ``VFIO_USER_DEVICE_SET_IRQ`` message with
+   ``flags=(VFIO_IRQ_SET_DATA_NONE/BOOL|VFIO_IRQ_SET_ACTION_MASK/UNMASK)``.
+
+Reply
+^^^^^
```

+
+There is no payload in the reply.
+
+.. _Read and Write Operations:
+
+Note that all of these operations must be supported by the client and/or server,
+even if the corresponding memory or device region has been shared as mappable.
+
+The ``count`` field must not exceed the value of ``max_data_xfer_size`` of the
+peer, for both reads and writes.
+
+``VFIO_USER_REGION_READ``
+-------------------------
+
+If a device region is not mappable, it's not directly accessible by the client
+via ``mmap()`` of the underlying file descriptor. In this case, a client can
+read from a device region with this message.
+
+Request
+^^^^^^^
+
++--------+--------+----------+
+| Name | Offset | Size |
++========+========+==========+
+| offset | 0 | 8 |
++--------+--------+----------+
+| region | 8 | 4 |
++--------+--------+----------+
+| count | 12 | 4 |
++--------+--------+----------+
+
+* *offset* into the region being accessed.
+* *region* is the index of the region being accessed.
+* *count* is the size of the data to be transferred.
+
+Reply
+^^^^^
+
++--------+--------+----------+
+| Name | Offset | Size |
++========+========+==========+
+| offset | 0 | 8 |
++--------+--------+----------+
+| region | 8 | 4 |
++--------+--------+----------+
+| count | 12 | 4 |
++--------+--------+----------+
+| data | 16 | variable |
++--------+--------+----------+
+
+* *offset* into the region accessed.
+* *region* is the index of the region accessed.
+* *count* is the size of the data transferred.
+* *data* is the data that was read from the device region.
+
+``VFIO_USER_REGION_WRITE``
+--------------------------
+
+If a device region is not mappable, it's not directly accessible by the client
+via mmap() of the underlying fd. In this case, a client can write to a device

+region with this message.
+
+Request
+^^^^^^^
+
++--------+--------+----------+
+| Name | Offset | Size |
++========+========+==========+
+| offset | 0 | 8 |
++--------+--------+----------+
+| region | 8 | 4 |
++--------+--------+----------+
+| count | 12 | 4 |
++--------+--------+----------+
+| data | 16 | variable |
++--------+--------+----------+
+
+* *offset* into the region being accessed.
+* *region* is the index of the region being accessed.
+* *count* is the size of the data to be transferred.
+* *data* is the data to write
+
+Reply
+^^^^^
+
++--------+--------+----------+
+| Name | Offset | Size |
++========+========+==========+
+| offset | 0 | 8 |
++--------+--------+----------+
+| region | 8 | 4 |
++--------+--------+----------+
+| count | 12 | 4 |
++--------+--------+----------+
+
+* *offset* into the region accessed.
+* *region* is the index of the region accessed.
+* *count* is the size of the data transferred.
+
+``VFIO_USER_DMA_READ``
+----------------------
+
+If the client has not shared mappable memory, the server can use this message to
+read from guest memory.
+
+Request
+^^^^^^^
+
++---------+--------+----------+
+| Name | Offset | Size |
++=========+========+==========+
+| address | 0 | 8 |
++---------+--------+----------+
+| count | 8 | 8 |
++---------+--------+----------+
+
+* *address* is the client DMA memory address being accessed. This address must have
+ been previously exported to the server with a ``VFIO_USER_DMA_MAP`` message.
+* *count* is the size of the data to be transferred.
+

+Reply
+^^^^^
+
++---------+--------+----------+
+| Name | Offset | Size |
++=========+========+==========+
+| address | 0 | 8 |
++---------+--------+----------+
+| count | 8 | 8 |
++---------+--------+----------+
+| data | 16 | variable |
++---------+--------+----------+
+
+* *address* is the client DMA memory address being accessed.
+* *count* is the size of the data transferred.
+* *data* is the data read.
+
+``VFIO_USER_DMA_WRITE``
+-----------------------
+
+If the client has not shared mappable memory, the server can use this message to
+write to guest memory.
+
+Request
+^^^^^^^
+
++---------+--------+----------+
+| Name | Offset | Size |
++=========+========+==========+
+| address | 0 | 8 |
++---------+--------+----------+
+| count | 8 | 8 |
++---------+--------+----------+
+| data | 16 | variable |
++---------+--------+----------+
+
+* *address* is the client DMA memory address being accessed. This address must have
+ been previously exported to the server with a ``VFIO_USER_DMA_MAP`` message.
+* *count* is the size of the data to be transferred.
+* *data* is the data to write
+
+Reply
+^^^^^
+
++---------+--------+----------+
+| Name | Offset | Size |
++=========+========+==========+
+| address | 0 | 8 |
++---------+--------+----------+
+| count | 8 | 4 |
++---------+--------+----------+
+
+* *address* is the client DMA memory address being accessed.
+* *count* is the size of the data transferred.
+
+``VFIO_USER_DEVICE_RESET``
+--------------------------
+
+This command message is sent from the client to the server to reset the device.
+Neither the request or reply have a payload.

+
+``VFIO_USER_DIRTY_PAGES``
+------------------------
+
+This command is analogous to ``VFIO_IOMMU_DIRTY_PAGES``. It is sent by the client
+to the server in order to control logging of dirty pages, usually during a live
+migration.
+
+Dirty page tracking is optional for server implementation; clients should not
+rely on it.
+
+Request
+^^^^^^^
+

Name	Offset	Size	
argsz	0	4	
flags	4	4	
		Bit	Definition
		0	VFIO_IOMMU_DIRTY_PAGES_FLAG_START
		1	VFIO_IOMMU_DIRTY_PAGES_FLAG_STOP
		2	VFIO_IOMMU_DIRTY_PAGES_FLAG_GET_BITMAP

+
+* *argsz* is the size of the VFIO dirty bitmap info structure for
+ ``START/STOP``; and for ``GET_BITMAP``, the maximum size of the reply payload
+
+* *flags* defines the action to be performed by the server:
+
+ * ``VFIO_IOMMU_DIRTY_PAGES_FLAG_START`` instructs the server to start logging
+   pages it dirties. Logging continues until explicitly disabled by
+   ``VFIO_IOMMU_DIRTY_PAGES_FLAG_STOP``.
+
+ * ``VFIO_IOMMU_DIRTY_PAGES_FLAG_STOP`` instructs the server to stop logging
+   dirty pages.
+
+ * ``VFIO_IOMMU_DIRTY_PAGES_FLAG_GET_BITMAP`` requests the server to return
+   the dirty bitmap for a specific IOVA range. The IOVA range is specified by
+   a "VFIO Bitmap Range" structure, which must immediately follow this
+   "VFIO Dirty Pages" structure. See `VFIO Bitmap Range Format`_.
+   This operation is only valid if logging of dirty pages has been previously
+   started.
+
+ These flags are mutually exclusive with each other.
+
+This part of the request is analogous to VFIO's ``struct
+vfio_iommu_type1_dirty_bitmap``.
+
+.. _VFIO Bitmap Range Format:
+
+VFIO Bitmap Range Format
+""""""""""""""""""""""""

```
+
++--------+--------+------+
+| Name   | Offset | Size |
++========+========+======+
+| iova   | 0      | 8    |
++--------+--------+------+
+| size   | 8      | 8    |
++--------+--------+------+
+| bitmap | 16     | 24   |
++--------+--------+------+
+
+* *iova* is the IOVA offset
+
+* *size* is the size of the IOVA region
+
+* *bitmap* is the VFIO Bitmap explained in `VFIO Bitmap`_.
+
+This part of the request is analogous to VFIO's ``struct
+vfio_iommu_type1_dirty_bitmap_get``.
+
+Reply
+^^^^^
+
+For ``VFIO_IOMMU_DIRTY_PAGES_FLAG_START`` or
+``VFIO_IOMMU_DIRTY_PAGES_FLAG_STOP``, there is no reply payload.
+
+For ``VFIO_IOMMU_DIRTY_PAGES_FLAG_GET_BITMAP``, the reply payload is as follows:
+
++--------------+--------+-----------------------------------------+
+| Name         | Offset | Size                                    |
++==============+========+=========================================+
+| argsz        | 0      | 4                                       |
++--------------+--------+-----------------------------------------+
+| flags        | 4      | 4                                       |
++--------------+--------+-----------------------------------------+
+|              | +-----+-----------------------------------------+ |
+|              | | Bit | Definition                              | |
+|              | +=====+=========================================+ |
+|              | | 2   | VFIO_IOMMU_DIRTY_PAGES_FLAG_GET_BITMAP | |
+|              | +-----+-----------------------------------------+ |
++--------------+--------+-----------------------------------------+
+| bitmap range | 8      | 40                                      |
++--------------+--------+-----------------------------------------+
+| bitmap       | 48     | variable                                |
++--------------+--------+-----------------------------------------+
+
+* *argsz* is the size required for the full reply payload (dirty pages structure
+  + bitmap range structure + actual bitmap)
+* *flags* is ``VFIO_IOMMU_DIRTY_PAGES_FLAG_GET_BITMAP``
+* *bitmap range* is the same bitmap range struct provided in the request, as
+  defined in `VFIO Bitmap Range Format`_.
+* *bitmap* is the actual dirty pages bitmap corresponding to the range request
+
+VFIO Device Migration Info
+--------------------------
+
+A device may contain a migration region (of type
+``VFIO_REGION_TYPE_MIGRATION``).  The beginning of the region must contain
+``struct vfio_device_migration_info``, defined in ``<linux/vfio.h>``. This
+subregion is accessed like any other part of a standard vfio-user region
```

+using ``VFIO_USER_REGION_READ``/``VFIO_USER_REGION_WRITE``.
+
++--------------+--------+-----------------------------+
+| Name | Offset | Size |
++==============+========+=============================+
+| device_state | 0 | 4 |
++--------------+--------+-----------------------------+
+| | +-----+---------------------------+ |
+| | | Bit | Definition | |
+| | +=====+===========================+ |
+| | | 0 | VFIO_DEVICE_STATE_RUNNING | |
+| | +-----+---------------------------+ |
+| | | 1 | VFIO_DEVICE_STATE_SAVING | |
+| | +-----+---------------------------+ |
+| | | 2 | VFIO_DEVICE_STATE_RESUMING | |
+| | +-----+---------------------------+ |
++--------------+--------+-----------------------------+
+| reserved | 4 | 4 |
++--------------+--------+-----------------------------+
+| pending_bytes | 8 | 8 |
++--------------+--------+-----------------------------+
+| data_offset | 16 | 8 |
++--------------+--------+-----------------------------+
+| data_size | 24 | 8 |
++--------------+--------+-----------------------------+
+
+* *device_state* defines the state of the device:
+
+ The client initiates device state transition by writing the intended state.
+ The server must respond only after it has successfully transitioned to the new
+ state. If an error occurs then the server must respond to the
+ ``VFIO_USER_REGION_WRITE`` operation with the Error field set accordingly and
+ must remain at the previous state, or in case of internal error it must
+ transition to the error state, defined as
+ ``VFIO_DEVICE_STATE_RESUMING | VFIO_DEVICE_STATE_SAVING``. The client must
+ re-read the device state in order to determine it afresh.
+
+ The following device states are defined:
+
+ +-----------+---------+----------+-----------------------------------+
+ | _RESUMING | _SAVING | _RUNNING | Description |
+ +===========+=========+==========+===================================+
+ | 0 | 0 | 0 | Device is stopped. |
+ +-----------+---------+----------+-----------------------------------+
+ | 0 | 0 | 1 | Device is running, default state. |
+ +-----------+---------+----------+-----------------------------------+
+ | 0 | 1 | 0 | Stop-and-copy state |
+ +-----------+---------+----------+-----------------------------------+
+ | 0 | 1 | 1 | Pre-copy state |
+ +-----------+---------+----------+-----------------------------------+
+ | 1 | 0 | 0 | Resuming |
+ +-----------+---------+----------+-----------------------------------+
+ | 1 | 0 | 1 | Invalid state |
+ +-----------+---------+----------+-----------------------------------+
+ | 1 | 1 | 0 | Error state |
+ +-----------+---------+----------+-----------------------------------+
+ | 1 | 1 | 1 | Invalid state |
+ +-----------+---------+----------+-----------------------------------+
+
+ Valid state transitions are shown in the following table:

+
+
+-------------------------+---------+---------+---------------+----------+----------+
+ | |darr| From / To |rarr| | Stopped | Running | Stop-and-copy | Pre-copy | Resuming
|
+ +=========================+=========+=========+===============+==========+=========
=+
+ | Stopped | \- | 1 | 0 | 0 | 0
|
+
+-------------------------+---------+---------+---------------+----------+----------+
+ | Running | 1 | \- | 1 | 1 | 1
|
+
+-------------------------+---------+---------+---------------+----------+----------+
+ | Stop-and-copy | 1 | 1 | \- | 0 | 0
|
+
+-------------------------+---------+---------+---------------+----------+----------+
+ | Pre-copy | 0 | 0 | 1 | \- | 0
|
+
+-------------------------+---------+---------+---------------+----------+----------+
+ | Resuming | 0 | 1 | 0 | 0 | \-
|
+
+-------------------------+---------+---------+---------------+----------+----------+
+
+ A device is migrated to the destination as follows:
+
+ * The source client transitions the device state from the running state to
+   the pre-copy state. This transition is optional for the client but must be
+   supported by the server. The source server starts sending device state data
+   to the source client through the migration region while the device is
+   running.
+
+ * The source client transitions the device state from the running state or the
+   pre-copy state to the stop-and-copy state. The source server stops the
+   device, saves device state and sends it to the source client through the
+   migration region.
+
+ The source client is responsible for sending the migration data to the
+ destination client.
+
+ A device is resumed on the destination as follows:
+
+ * The destination client transitions the device state from the running state
+   to the resuming state. The destination server uses the device state data
+   received through the migration region to resume the device.
+
+ * The destination client provides saved device state to the destination
+   server and then transitions the device to back to the running state.
+
+* *reserved* This field is reserved and any access to it must be ignored by the
+ server.
+
+* *pending_bytes* Remaining bytes to be migrated by the server. This field is
+ read only.
+
+* *data_offset* Offset in the migration region where the client must:

+
+ * read from, during the pre-copy or stop-and-copy state, or
+
+ * write to, during the resuming state.
+
+ This field is read only.
+
+* *data_size* Contains the size, in bytes, of the amount of data copied to:
+
+ * the source migration region by the source server during the pre-copy or
+   stop-and copy state, or
+
+ * the destination migration region by the destination client during the
+   resuming state.
+
+Device-specific data must be stored at any position after
+``struct vfio_device_migration_info``. Note that the migration region can be
+memory mappable, even partially. In practise, only the migration data portion
+can be memory mapped.
+
+The client processes device state data during the pre-copy and the
+stop-and-copy state in the following iterative manner:
+
+ 1. The client reads ``pending_bytes`` to mark a new iteration. Repeated reads
+   of this field is an idempotent operation. If there are no migration data
+   to be consumed then the next step depends on the current device state:
+
+   * pre-copy: the client must try again.
+
+   * stop-and-copy: this procedure can end and the device can now start
+   resuming on the destination.
+
+ 2. The client reads ``data_offset``; at this point the server must make
+   available a portion of migration data at this offset to be read by the
+   client, which must happen *before* completing the read operation. The
+   amount of data to be read must be stored in the ``data_size`` field, which
+   the client reads next.
+
+ 3. The client reads ``data_size`` to determine the amount of migration data
+   available.
+
+ 4. The client reads and processes the migration data.
+
+ 5. Go to step 1.
+
+Note that the client can transition the device from the pre-copy state to the
+stop-and-copy state at any time; ``pending_bytes`` does not need to become zero.
+
+The client initializes the device state on the destination by setting the
+device state in the resuming state and writing the migration data to the
+destination migration region at ``data_offset`` offset. The client can write the
+source migration data in an iterative manner and the server must consume this
+data before completing each write operation, updating the ``data_offset`` field.
+The server must apply the source migration data on the device resume state. The
+client must write data on the same order and transaction size as read.
+
+If an error occurs then the server must fail the read or write operation. It is
+an implementation detail of the client how to handle errors.
+
+Appendices

```
+==========
+
+Unused VFIO ``ioctl()`` commands
+--------------------------------
+
+The following VFIO commands do not have an equivalent vfio-user command:
+
+* ``VFIO_GET_API_VERSION``
+* ``VFIO_CHECK_EXTENSION``
+* ``VFIO_SET_IOMMU``
+* ``VFIO_GROUP_GET_STATUS``
+* ``VFIO_GROUP_SET_CONTAINER``
+* ``VFIO_GROUP_UNSET_CONTAINER``
+* ``VFIO_GROUP_GET_DEVICE_FD``
+* ``VFIO_IOMMU_GET_INFO``
+
+However, once support for live migration for VFIO devices is finalized some
+of the above commands may have to be handled by the client in their
+corresponding vfio-user form. This will be addressed in a future protocol
+version.
+
+VFIO groups and containers
+^^^^^^^^^^^^^^^^^^^^^^^^^^
+
+The current VFIO implementation includes group and container idioms that
+describe how a device relates to the host IOMMU. In the vfio-user
+implementation, the IOMMU is implemented in SW by the client, and is not
+visible to the server. The simplest idea would be that the client put each
+device into its own group and container.
+
+Backend Program Conventions
+---------------------------
+
+vfio-user backend program conventions are based on the vhost-user ones.
+
+* The backend program must not daemonize itself.
+* No assumptions must be made as to what access the backend program has on the
+  system.
+* File descriptors 0, 1 and 2 must exist, must have regular
+  stdin/stdout/stderr semantics, and can be redirected.
+* The backend program must honor the SIGTERM signal.
+* The backend program must accept the following commands line options:
+
+  * ``--socket-path=PATH``: path to UNIX domain socket,
+  * ``--fd=FDNUM``: file descriptor for UNIX domain socket, incompatible with
+    ``--socket-path``
+* The backend program must be accompanied with a JSON file stored under
+  ``/usr/share/vfio-user``.
+
+TODO add schema similar to docs/interop/vhost-user.json.
--
2.22.3
```