2.5 Packed Virtqueues

Packed virtqueues is an alternative compact virtqueue layout using read-write memory, that is memory that is both read and written by both host and guest.

This layout is enabled by negotiating[1] a VIRTIO_F_PACKED_VIRTQUEUE[1] feature.

Packed virtqueues support up to 2^{14} queues, with up to 2^{15} entries each.

Each packed virtqueue consists of three parts:

- Descriptor Ring
- Device Event Suppression
- Driver Event Suppression

Where Descriptor Ring in turn consists of descriptors, and where each descriptor can contain the following parts:

- Buffer ID
- Buffer Address
- Buffer Length
- Flags

A buffer consists of zero or more device-readable physically-contiguous elements followed by zero or more physically-contiguous device-writable elements (each buffer has at least one element).

When the driver wants to send such a buffer to the device, it writes at least one available descriptor describing elements of the buffer into the Descriptor Ring. The descriptor(s) are associated with a buffer by means of a Buffer ID stored within the descriptor.

Driver then notifies the device. When the device has finished processing the buffer, it writes a used device descriptor including the Buffer ID into the Descriptor Ring (overwriting a driver descriptor previously made available), and sends an interrupt.

Descriptor Ring is used in a circular manner: driver writes descriptors into the ring in order. After reaching end of ring, the next descriptor is placed at head of the ring. Once ring is full of driver descriptors, driver stops sending new requests and waits for device to start processing descriptors and to write out some used descriptors before making new driver descriptors available.

Similarly, device reads descriptors from the ring in order and detects that a driver descriptor has been made available. As processing of descriptors is completed used descriptors are written by the device back into the ring.

Note: after reading driver descriptors and starting their processing in order, device might complete their processing out of order. Used device descriptors are written in the order in which their processing is complete.

Device Event suppression data structure is read-only by the device. It includes information for reducing the number of device interrupts to driver.

Driver Event suppression data structure is write-only by the device. It includes information for reducing the number of driver notifications to device.

2.5.1 Available and Used Ring Full Counters

Each of the driver and the device are expected to maintain, internally, a single-bit ring wrap counter initialized to 1.

The counter maintained by the driver is called the Available Ring Full Counter. Driver changes its value each time it makes available the last descriptor in the ring (after making the last descriptor available).

The counter maintained by the device is called the Used Ring Wrap Counter. Device changes its value each time it uses the last descriptor in the ring (after marking the last descriptor used).

It is easy to see that the Availablering Wrap Counter in the driver matches the Used Ring Wrap Counter in the device when both are processing the same descriptor, or when all available descriptors have been used.

To mark a descriptor as available and used, both driver and device use the following two flags:

```
#define VIRTQ_DESC_F_AVAIL     7
#define VIRTQ_DESC_F_USED      15
```

To mark a descriptor as available, driver sets the VIRTQ_DESC_F_AVAIL bit in Flags to match the internal Available Ring Wrap Counter. It also sets the VIRTQ_DESC_F_USED bit to match the *inverse* value.

To mark a descriptor as used, device sets the VIRTQ_DESC_F_USED bit in Flags to match the internal Used Ring Wrap Counter. It also sets the VIRTQ_DESC_F_AVAIL bit to match the *same* value.

Thus VIRTQ_DESC_F_AVAIL and VIRTQ_DESC_F_USED bits are different for an available descriptor and equal for a used descriptor.

2.5.2 Polling of available and used descriptors

Writes of device and driver descriptors can generally be reordered, but each side (driver and device) are only required to poll a single location in memory: next device descriptor after the one they processed previously, in circular order.

Sometimes device needs to only write out a single used descriptor after processing a batch of multiple available descriptors. As described in more detail below, this can happen when using descriptor chaining or with in-order use of descriptors. In this case, device writes out a used descriptor with buffer id of the last descriptor in the group. After processing the used descriptor, both device and driver then skip forward in the ring the number of the remaining descriptors in the group until processing (reading for the driver and writing for the device) the next used descriptor.

2.5.3 Write Flag

In an available descriptor, VIRTQ_DESC_F_WRITE bit within Flags is used to mark a descriptor as corresponding to a write-only or read-only element of a buffer.

```
/* This marks a buffer as device write-only (otherwise device read-only). */
#define VIRTQ_DESC_F_WRITE     2
```

In a used descriptor, this bit it used to specify whether any data has been written by the device into any parts of the buffer.

2.5.4 Buffer Address and Length

In an available descriptor, Buffer Address corresponds to the physical address of the buffer. The length of the buffer assumed to be physically contigious is stored in Buffer Length.

In a used descriptor, Buffer Address is unused. Buffer Length specifies the length of the buffer that has been initialized (written to) by the device.

Buffer length is reserved for used descriptors without the VIRTQ_DESC_F_WRITE flag, and is ignored by drivers.

2.5.5 Scatter-Gather Support

Some drivers need an ability to supply a list of multiple buffer elements (also known as a scatter/gather list) with a request. Two optional features support this: descriptor chaining and indirect descriptors.

If neither feature has been negotiated, each buffer is physically-contigious, either read-only or write-only and is described completely by a single descriptor.

While unusual (most implementations either create all lists solely using non-indirect descriptors, or always use a single indirect element), if both features have been negotiated, mixing direct and direct descriptors in a ring is valid, as long as each list only contains descriptors of a given type.

Scatter/gather lists only apply to available descriptors. A single used descriptor corresponds to the whole list.

The device limits the number of descriptors in a list through a bus-specific and/or device-specific value. If not limited, the maximum number of descriptors in a list is the virt queue size.

2.5.6 Next Flag: Descriptor Chaining

The VIRTIO_F_LIST_DESC feature allows driver to do this by using multiple descriptors, and setting the VIRTQ_DESC_F_NEXT in Flags for all but the last available descriptor.

```
/* This marks a buffer as continuing. */
#define VIRTQ_DESC_F_NEXT   1
```

Buffer ID is included in the last descriptor in the list.

The driver always makes the the first descriptor in the list available after the rest of the list has been written out into the ring. This guarantees that the device will never observe a partial scatter/gather list in the ring.

Device only writes out a single used descriptor for the whole list. It then skips forward according to the number of descriptors in the list. Driver needs to keep track of the size of the list corresponding to each buffer ID, to be able to skip to where the next used descriptor is written by the device.

For example, if descriptors are used in the same order in which they are made available, this will result in the used descriptor overwriting the first available descriptor in the list, the used descriptor for the next list overwriting the first available descriptor in the next list, etc.

VIRTQ_DESC_F_NEXT is reserved in used descriptors, and should be ignored by drivers.

2.5.7 Indirect Flag: Scatter-Gather Support

Some devices benefit by concurrently dispatching a large number of large requests. The VIRTIO_F_INDIRECT_DESC feature allows this. To increase ring capacity the driver can store (read-only by the device) table of indirect descriptors anywhere in memory, and insert a descriptor in main virtqueue (with *Flags*&VIRTQ_DESC_F_INDIRECT on) that refers to a memory buffer containing this indirect descriptor table; *addr* and *len* refer to the indirect table address and length in bytes, respectively.

```
/* This means the buffer contains a table of buffer descriptors. */
#define VIRTQ_DESC_F_INDIRECT   4
```

The indirect table layout structure looks like this (*len* is the Buffer Length of the descriptor that refers to this table, which is a variable, so this code won't compile):

```
struct indirect_descriptor_table {
        /* The actual descriptor structures (struct Desc each) */
        struct Desc desc[len / sizeof(struct Desc)];
};
```

The first descriptor is located at start of the indirect descriptor table, additional indirect descriptors come immediately afterwards. *Flags* &VIRTQ_DESC_F_WRITE is the only valid flag for descriptors in the indirect table. Others are reserved are ignored by the device. Buffer ID is also reserved and is ignored by the device.

In Descriptors with VIRTQ_DESC_F_INDIRECT set VIRTQ_DESC_F_WRITE is reserved and is ignored by the device.

2.5.8 In-order use of descriptors

Some devices always use descriptors in the same order in which they have been made available. These devices can offer the VIRTIO_F_IN_ORDER feature. If negotiated, this knowledge allows devices to notify the use of a batch of buffers to the driver by only writing out a single used descriptor with the Buffer ID corresponding to the last descriptor in the batch.

Device then skips forward in the ring according to the size of the the batch. Driver needs to look up the used Buffer ID and calculate the batch size to be able to advance to where the next used descriptor will be written by the device.

This will result in the used descriptor overwriting the first available descriptor in the batch, the used descriptor for the next batch overwriting the first available descriptor in the next batch, etc.

The skipped buffers (for which no used descriptor was written) are assumed to have been used (read or written) by the device completely.

2.5.9 Multi-buffer requests

Some devices combine multiple buffers as part of processing a single request. These devices always makes the the first descriptor in the request available after the rest of the request has been written out request the ring. This guarantees that the driver will never observe a partial request in the ring.

2.5.10 Driver and Device Event Suppression

In many systems driver and device notifications involve significant overhead. To mitigate this overhead, each virtqueue includes two identical structures used for controlling notifications between device and driver.

Driver Event Suppression structure is read-only by the device and controls the events sent by the device (e.g. interrupts).

Device Event Suppression structure is read-only by the driver and controls the events sent by the driver (e.g. IO).

Each of these structures includes the following fields:

Descriptor Event Flags Takes values:

- 00b reserved
- 01b enable events
- 11b disable events
- 10b enable events for a specific descriptor (as specified by Descriptor Event Offset/Wrap Counter).

Descriptor Event Offset If Event Flags set to descriptor specific event: offset within the ring (in units of descriptor size). Event will only trigger when this descriptor is made available/used respectively.

Descriptor Event Wrap Counter If Event Flags set to descriptor specific event: offset within the ring (in units of descriptor size). Event will only trigger when Ring Wrap Counter matches this value and a descriptor is made available/used respectively.

After writing out some descriptors, both device and driver are expected to consult the relevant structure to find out whether interrupt should be sent. As this access to shared memory involves overhead for some transports, the following additional field is present:

Structure Change Event Flags Enable/disable sending an event notification when the other side changes its own Event Suppression structure.

when enabled through this field, device and driver send an event notification whenever they change the driver and device event suppression structure respectively.

2.5.10.1 Driver notifications

Some devices benefit from ability to find out the number of available descriptors in the ring, and whether to send interrupts to drivers without accessing ring memory: for efficiency or as a debugging aid.

To help with these optimizations, driver notifications to the device include the following information:

- VQ number
- Flags - set to 00b
- Offset (in units of descriptor size) within the ring where the next available descriptor will be written

- Available Ring Wrap Counter

Whenever driver notifies device about a Device Event Suppression Structure change (if enabled through Structure Change Event Flags in Driver Event Suppression Structure), it sends a copy of the up-to-date Event Suppression Structure:

- VQ number
- Descriptor Event Flags
- Descriptor Event Offset
- Descriptor Event Wrap Counter

2.5.10.2 Structure Size and Alignment

Each part of the virtqueue is physically-contiguous in guest memory, and has different alignment requirements.

The memory aligment and size requirements, in bytes, of each part of the virtqueue are summarized in the following table:

Virtqueue Part	Alignment	Size
Descriptor Ring	16	$16*$(Queue Size)
Device Event Suppression	4	4
Driver Event Suppression	4	4

The Alignment column gives the minimum alignment for each part of the virtqueue.

The Size column gives the total number of bytes for each part of the virtqueue.

Queue Size corresponds to the maximum number of descriptors in the virtqueue[3]. Queue Size value does not have to be a power of 2 unless enforced by the transport.

2.5.11 Driver Requirements: Virtqueues

The driver MUST ensure that the physical address of the first byte of each virtqueue part is a multiple of the specified alignment value in the above table.

2.5.12 Device Requirements: Virtqueues

The device MUST start processing driver descriptors in the order in which they appear in the ring. The device MUST start writing device descriptors into the ring in the order in which they complete. Device MAY reorder descriptor writes once they are started.

2.5.13 The Virtqueue Descriptor Format

The available descriptor refers to the buffers the driver is sending to the device. *addr* is a physical address, and the descriptor is identified with a buffer using the *id* field.

```
struct virtq_desc {
        /* Buffer Address. */
        le64 addr;
        /* Buffer Length. */
        le32 len;
        /* Buffer ID. */
        le16 id;
        /* The flags depending on descriptor type. */
```

[3]For example, if Queue Size is 4 then at most 4 buffers can be queued at any given time.

```
        le16 flags;
};
```

The descriptor ring is zero-initialized.

2.5.14 Event Suppression Structure Format

The following structure is used to reduce the number of notifications sent between driver and device.

```
__le16 desc_event_off : 15; /* Descriptor Event Offset */
int    desc_event_wrap : 1; /* Descriptor Event Wrap Counter */
__le16 desc_event_flags : 2; /* Descriptor Event Flags */
__le16 structure_change_flags : 1; /* Structure Change Event Flags */
```

2.5.15 Driver Notification Format

The following structure is used to notify device of available descriptors and of event suppression structure changes:

```
__le16 vqn : 14;
__le16 desc_event_flags : 2;
__le16 desc_event_off : 15;
int    desc_event_wrap : 1;
```

2.5.15.1 Device Requirements: The Virtqueue Descriptor Table

A device MUST NOT write to a device-readable buffer, and a device SHOULD NOT read a device-writable buffer. A device MUST NOT use a descriptor unless it observes VIRTQ_DESC_F_AVAIL bit in its *flags* being changed. A device MUST NOT change a descriptor after changing it's VIRTQ_DESC_F_USED bit in its *flags*.

2.5.15.2 Driver Requirements: The Virtqueue Descriptor Table

A driver MUST NOT change a descriptor unless it observes VIRTQ_DESC_F_USED bit in its *flags* being changed. A driver MUST NOT change a descriptor after changing VIRTQ_DESC_F_USED bit in its *flags*.

2.5.15.2.1 Driver Requirements: Scatter-Gather Support

The driver MUST NOT set the DESC_F_LIST_NEXT flag unless the VIRTIO_F_LIST_DESC feature was negotiated.

A driver MUST NOT create a descriptor list longer than allowed by the device.

A driver MUST NOT create a descriptor list longer than the Queue Size.

This implies that loops in the descriptor list are forbidden!

The driver MUST place any device-writable descriptor elements after any device-readable descriptor elements.

A driver MUST NOT depend on the device to use more descriptors to be able to write out all descriptors in a list. A driver MUST make sure there's enough space in the ring for the whole list before making any of the descriptors available to the device.

A driver MUST NOT make the first descriptor in the list available before initializing the rest of the descriptors.

2.5.15.2.2 Device Requirements: Scatter-Gather Support

The device MUST use descriptors in a list chained by the VIRTQ_DESC_F_NEXT flag in the same order that they were made available by the driver.

The device MAY limit the number of buffers it will allow in a list.

2.5.15.2.3 Driver Requirements: Indirect Descriptors

The driver MUST NOT set the DESC_F_INDIRECT flag unless the VIRTIO_F_INDIRECT_DESC feature was negotiated. The driver MUST NOT set any flags except DESC_F_WRITE within an indirect descriptor.

A driver MUST NOT create a descriptor chain longer than allowed by the device.

A driver MUST NOT write direct descriptors with DESC_F_INDIRECT set in a scatter-gather list linked by VIRTQ_DESC_F_NEXT. *flags*.