ハイパーバイザの作り方〜ちゃんと理解する仮想化技術〜 第１２回 virtio による準仮想化デバイス その２「Virtqueue と virtio-net の実現」

はじめに

前回は、ゲスト OS の I/O パフォーマンスを大きく改善する「virtio」準仮想化ドライバの概要と、virtio のコンポーネントの 1 つである「Virtio PCI」について解説しました。今回は Virtqueue とこれを用いた NIC(virtio-net) の実現方法について見ていきます。

virtio のおさらい

virtio は、大きく分けて Virtio PCI と Virtqueue の 2 つのコンポーネントからなります。Virtio PCI はゲストマシンに対して PCI デバイスとして振る舞い、次のような機能を提供します。

- デバイス初期化時のホスト<->ゲスト間ネゴシエーションや設定情報通知に使うコンフィギュレーションレジスタ

これを利用してキュー長やキュー数、キューのアドレスなどを通知する、

- 割り込み (ホスト->ゲスト)、I/O ポートアクセス (ゲスト->ホスト) によるホスト<->ゲスト間イベント通知機構
- 標準的な PCI デバイスの DMA 機構を用いたデータ転送機能

があります。

Virtqueue はデータ転送に使われるゲストメモリ空間上のキュー構造です。デバイスごとに 1 つまたは複数のキューを持つことができます。たとえば、virtio-net は送信用キュー, 受信用キュー, コントロール用キューの 3 つを必要とします。ゲスト OS は、PCI デバイスとして virtio デバイスを検出して初期化し、Virtqueue をデータの入出力に、割り込みと I/O ポートアクセスをイベント通知に用いてホストに対して I/O を依頼します。本稿では、Virtqueue についてより詳しく見ていきましょう。

Virtqueue

Virtqueue は送受信するデータをキューイング先の Descriptor が並ぶ Descriptor Table、ゲストからホストへ受け渡す descriptor を指定する Available Ring、ホストからゲストへ受け渡す descriptor を指定する Used Ring の 3 つからなります (図 1)。

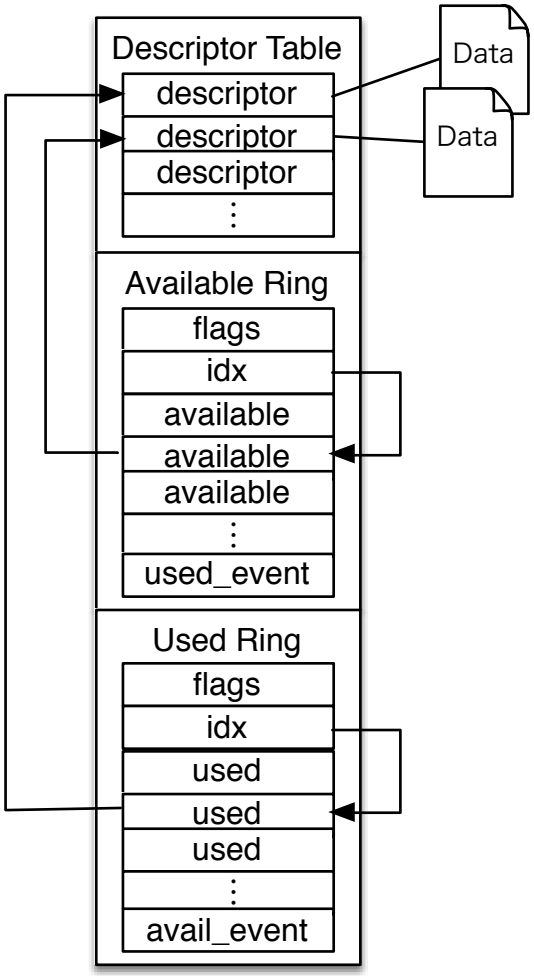

図 1　Virtqueue の構造

Descriptor Table, Available Ring, Used Ring のエントリ数は Virtio PCI デバイスの初期化時に Virtio header の QUEUE_NUM へ設定した値で決められます。

また、Virtqueue の領域はページサイズ[*1]へアラインされている必要があります。1 つの Virtqueue は片方向の通信に用いられます。このため、双方向通信をサポートするには 2 つの Virtqueue を使用する必要があり

[*1] ページサイズ = 4KB

ます。通信方向によって、Available Ring と Used Ring の使われ方が異なります。

Descriptor Table

Descriptor Table は Descriptor が QUEUE_NUM 個[*2]並んでいる配列です。Descriptor はデータ転送を行う都度動的にアロケートされるのではなく、Descriptor Table 内の空きエントリを探して使用します。空きエントリを管理する構造は Virtqueue 上にないため、ゲストドライバは空き Descriptor を記憶しておく必要があります (後述)。

Descriptor は転送するデータ 1 つに対して 1 つ使われ、データのアドレス、データ長などが含まれます (表 1)。

データのアドレスはゲスト上の物理アドレスが用いられるため、仮想アドレス上で連続する領域でも物理ページがばらばらな場合、物理ページごとに Descriptor が 1 つ必要です。このように複数の Descriptor を連続して転送したい場合には、next で次の Descriptor の番号を指定して flags に 0x1 をビットセットします。

type	member	description
u64	addr	データのアドレス（ゲスト物理アドレス）
u32	len	データ長
u16	flags	フラグ
		（0x1: 次の Descriptor があるかどうか
		0x2: ホストから見て Write Only の Descriptor かどうか
		0x4: Indirect Descriptor かどうか）
u16	next	次の Descriptor 番号

表 1: Descriptor の構造

Indirect Descriptor

ある種の virtio デバイスは多数の descriptor を消費するリクエストを大量に並列に発行することにより、性能を向上させることができます。

これを可能にするのが Indirect Descriptor です。Descriptor の flags に 0x4 が指定された場合、addr は Indirect Descriptor Table のアドレスを、len は Indirect Descriptor Table の長さ (バイト数) を示すようになります。

[*2] Virtio Header の QUEUE_NUM で指定する。

Indirect Descriptor Table は Descriptor Table と同様、Descriptor の配列になっています。Indirect Descriptor Table に含まれる Descriptor の数は len/16 個になります (*3)。

それぞれのデータは Indirect Descriptor Table 上の Descriptor へリンクされます。

Available Ring

Available Ring はゲストからホストへ渡したい Descriptor を指定するのに使用します (表 2)。ゲストはリング上の空きエントリへ Descriptor 番号を書き込んで idx をインクリメントします。idx は単純にインクリメントし続ける使い方が想定されているため、リング長を超える idx 値が指定された時は idx をリング長で割った余りをインデックス値として使用します。

ホストは最後に処理したリング上のエントリの番号を記憶しておき (後述)、idx と比較して新しいエントリが指している Descriptor を処理します。

type	member	description
u16	flags	フラグ（0x1: 割り込みの一時的な抑制）
u16	idx	リング上で一番新しいエントリの番号
u16[QUEUE_NUM]	ring	Descriptor 番号を書き込むリングの本体
u16	used_event	ここで指定した番号の Descriptor が処理されるまで割り込みを抑制

表 2: Available Ring の構造

Used Ring

Used Ring はホストからゲストへ渡したい Descriptor を指定するのに使用されます (表 3)。

構造と使用方法は基本的に Available Ring と同じですが、リング上のエントリの構造が Available Ring と異なり、連続する Descriptor を先頭番号 (id) と長さ (len) で範囲指定するようになっています (表 4)。

type	member	description
u16	flags	フラグ（0x1: ゲストからの通知の一時的な抑制）
u16	idx	リング上で一番新しいエントリの番号
UsedRingEntry[QUEUE_NUM]	ring	Descriptor 番号を書き込むリングの本体
u16	avail_event	ここで指定された番号の Descriptor が処理されるまで割り込みを抑制

*3 1 つの Descriptor の長さが 16bytes であるため。

type	member	description

表 3: Used Ring の構造

type	member	description
u32	id	先頭の Descriptor 番号
u32	len	Descriptor チェーンの長さ

表 4: Used Ring エントリの構造

Virtqueue に含まれない変数

Virtqueue を用いてデータ転送を行うために、Virtqueue に含まれない次の変数が必要です。

- ゲストドライバ
  - free_head. . . . . 空き Descriptor を管理するため、空き Descriptor の先頭番号を保持
  - last_used_idx. . . . . 最後に処理した Used Ring 上のエントリの番号
- ホストドライバ
  - last_avail_idx. . . . . 最後に処理した Available Ring 上のエントリの番号

ゲスト->ホスト方向のデータ転送方法

ゲストからホストへデータを転送するために、Descriptor Table, Available Ring, Used Ring をどのように使うかを次に示します (図 2)。

この方向のデータ転送では、Available Ring は転送データを含む Descriptor の通知に使われ、Used Ring は処理済み Descriptor の回収に使われます。

ゲストドライバ

図 2 の番号にそって解説します。

1. ドライバの初期化時にあらかじめすべての Descriptor の next の値を隣り合った Descriptor のエントリ番号に設定し空き Descriptor のチェーンを作成、チェーンの先頭 Descriptor の番号を free_head に

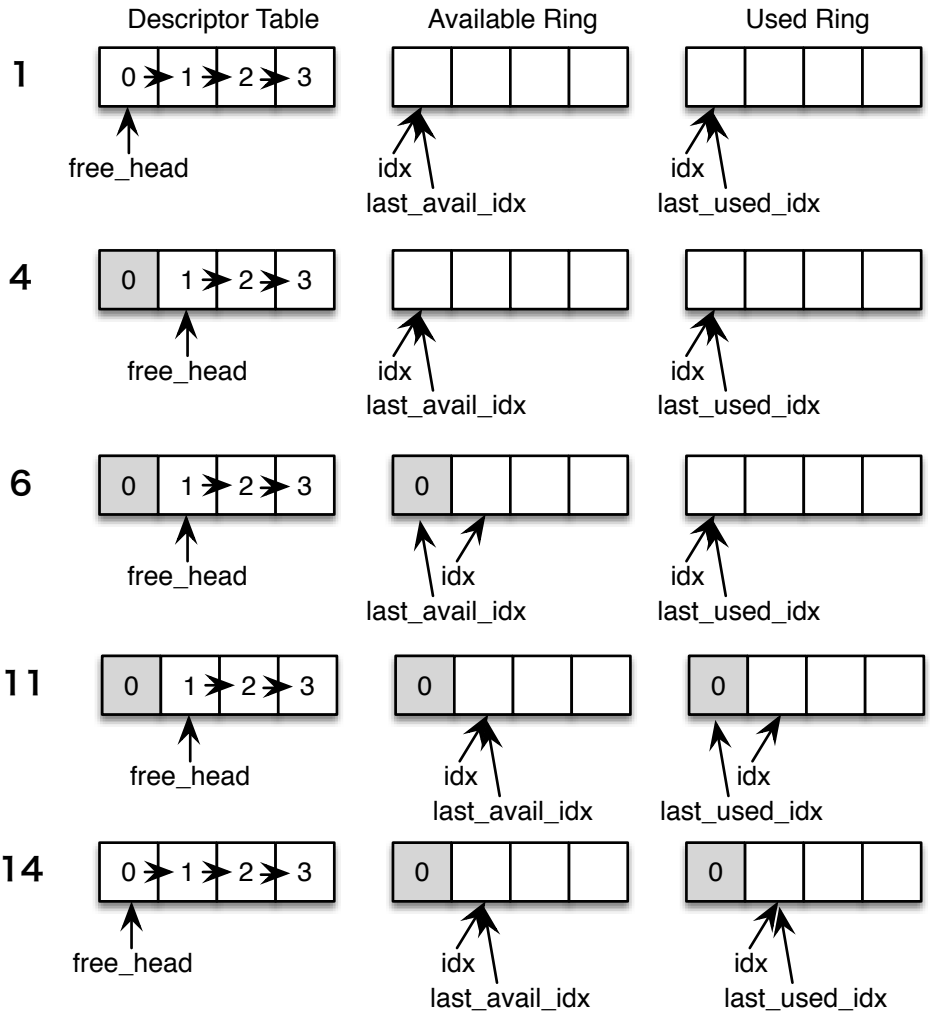

図 2　ゲスト->ホスト方向データ転送のイメージ

代入しておく

2. free_head の値から空き Descriptor 番号を取得
3. Descriptor の addr にデータのアドレス、len にデータ長を代入
4. Descriptor の next が指す次の空き Descriptor の番号を free_head へ代入
5. Available Ring の idx が指す空きエントリに Descriptor の番号を代入
6. Available Ring の idx をインクリメント (新しい空きエントリ)
7. Virtio Header の QUEUE_SEL にキュー番号を書き込み
8. 未処理データがあることをホストへ通知するため Virtio Header の QUEUE_NOTIFY へ書き込み[*4]

[*4] QUEUE_NOTIFY へ書き込むことにより VMExit が発生し、ホスト側へ制御が移ることを意図している。

ホストドライバ

図 2 の番号にそって解説します。

9. ゲストからの通知を受けて last_avail_idx と Available Ring の idx を比較、新しいエントリが指している Descriptor を順に処理、last_avail_idx をインクリメント
10. Used Flags の idx が指す次の空きエントリに処理済み Descriptor の番号を代入
11. Used Flags の idx をインクリメント
12. 処理が終わったことを通知するためゲストへ割り込み

ゲストドライバ

図 2 の番号にそって解説します。

13. ホストからの割り込みを受けて last_used_idx と Used Ring の idx を比較、新しいエントリが指している処理済み Descriptor を順に回収、last_used_idx をインクリメント
14. 回収対象の Descriptor を空き Descriptor のチェーンへ戻し、free_head を更新

ホスト->ゲスト方向のデータ転送方法

ホストからゲストへデータを転送するために、Descriptor Table, Available Ring, Used Ring をどのように使うかを次に示します (図 3)。

この方向のデータ転送では、Available Ring は空き Descriptor の受け渡しに使われ、Used Ring は転送データを含む Descriptor の通知に使われます。

ゲストドライバ

図 3 の番号にそって解説します。

1. ドライバの初期化時にあらかじめすべての Descriptor の next の値を隣り合った Descriptor のエントリ番号に設定し空き Descriptor のチェーンを作成、チェーンの先頭 Descriptor の番号を free_head に代入しておく
2. Available Ring の idx が指す次の空きエントリに空き Descriptor チェーンの先頭番号を代入
3. Available Ring の idx をインクリメント
4. Virtio Header の QUEUE_SEL にキュー番号を書き込み
5. 未処理データがあることをホストへ通知するため Virtio Header の QUEUE_NOTIFY へ書き込み

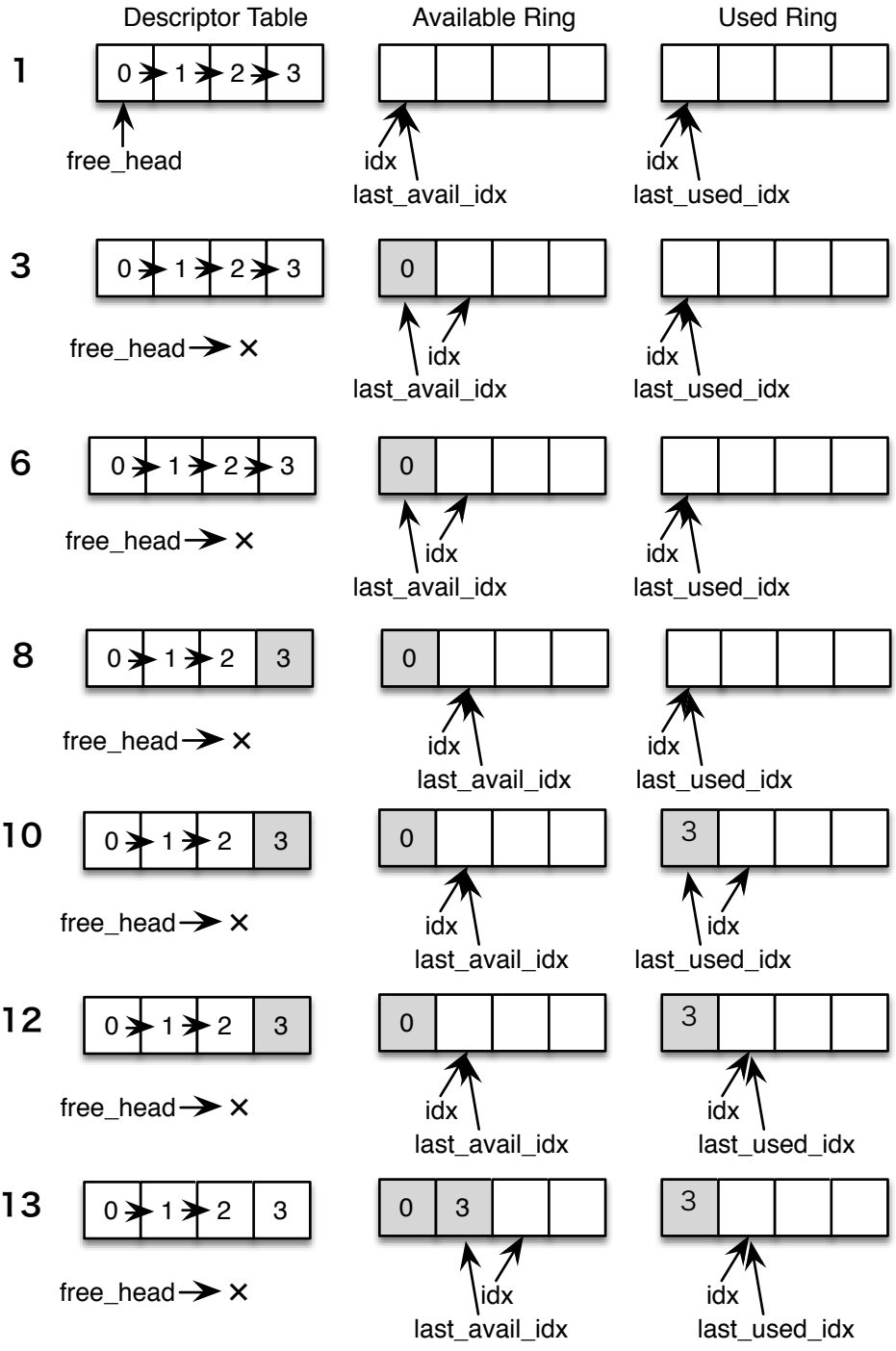

図 3　ホスト->ゲスト方向データ転送のイメージ

ホストドライバ

図 3 の番号にそって解説します。

6. データ送信要求を受けて Available Ring を参照、必要な数の Descriptor を取り出す
7. Descriptor を Available Ring 上の、Descriptor チェーンから切り離す
8. Descriptor の addr にデータのアドレス、len にデータ長を代入
9. Used Ring の idx が指す次の空きエントリに Descriptor の番号を代入
10. Used Ring の idx をインクリメント
11. 未処理データがあることを通知するためゲストへ割り込み

ゲストドライバ

図 3 の番号にそって解説します。

12. ホストからの割り込みを受けて last_used_idx と Used Ring の idx を比較、新しいエントリが指している処理済み Descriptor を順に処理、last_used_idx をインクリメント
13. 処理済み Descriptor を空き Descriptor のチェーンへ戻し、Available Ring を更新

virtio-net の実現方法

virtio-net は受信キュー、送信キュー、コントロールキューの 3 つの Virtqueue からなります。送信キューとコントロールキューはゲスト->ホスト方向のデータ転送方法で解説した手順でデータを転送します。受信キューはホスト->ゲスト方向のデータ転送方法で解説した手順でデータを転送します。受信キュー, 送信キューでは、パケットごとに 1 つの Descriptor を使用します。

Descriptor の addr には直接パケットのアドレスを指定しますが、ホストドライバからゲストドライバへいくつかの情報を通知するため、パケットの手前に専用の構造体を追加しています (表 5、図 4)。

type	member	description
u8	flags	フラグ（Checksum offload）
u8	gso_type	GSO によるパケットタイプ情報
u16	hdr_len	Ethernet + IP + TCP/UDP ヘッダの長さ
u16	gso_size	データ長
u16	csum_start	チェックサムフィールドの位置

type	member	description
u16	csum_offset	チェックサムの計算開始位置

表 5: struct virtio_net_hdr

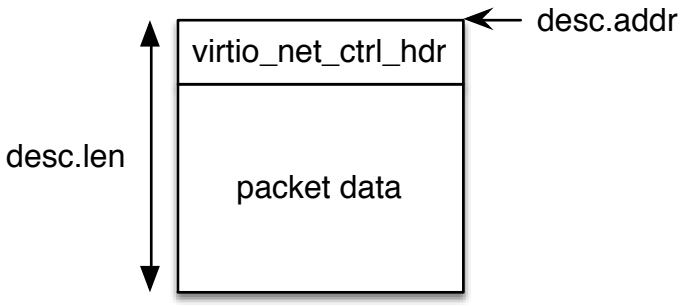

図 4　送受信キューのデータ構造

コントロールキューでは、コマンド用構造体 (表 6、図 5) にコマンド名を設定してゲストからホストへメッセージ送出します。コマンドに付属データが必要な場合は、コマンド用構造体の直後に続いてデータを配置します。コマンドはクラス (大項目) とコマンド (小項目) で整理されており、次のような種類があります。

type	member	description
u8	class	クラス（大項目）
u8	cmd	コマンド（小項目）

表 6: struct virtio_net_ctrl_hdr

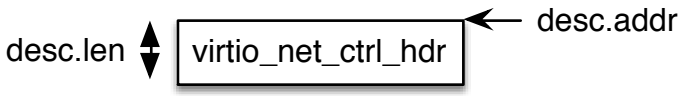

図 5　コントロールキューのデータ構造

VIRTIO_NET_CTRL_RX クラスは次のようなコマンドを持ち、NIC のプロミスキャスモード、ブロードキャスト受信、マルチキャスト受信などの有効/無効化を行います。

- VIRTIO_NET_CTRL_RX_PROMISC

- VIRTIO_NET_CTRL_RX_ALLMULTI
- VIRTIO_NET_CTRL_RX_ALLUNI
- VIRTIO_NET_CTRL_RX_NOMULTI
- VIRTIO_NET_CTRL_RX_NOUNI
- VIRTIO_NET_CTRL_RX_NOBCAST

VIRTIO_NET_CTRL_MAC クラスは次のようなコマンドを持ち、MAC フィルタテーブルの設定に使用します。

- VIRTIO_NET_CTRL_MAC_TABLE_SET
- VIRTIO_NET_CTRL_MAC_ADDR_SET

VIRTIO_NET_CTRL_VLAN クラスは次のようなコマンドを持ち、VLAN の設定に使用します。

- VIRTIO_NET_CTRL_VLAN_ADD
- VIRTIO_NET_CTRL_VLAN_DEL

VIRTIO_NET_CTRL_ANNOUNCE クラスは次のようなコマンドを持ち、リンクステータス通知に対して ack を返すのに使用します。

- VIRTIO_NET_CTRL_ANNOUNCE
- VIRTIO_NET_CTRL_ANNOUNCE_ACK

VIRTIO_NET_CTRL_MQ クラスクラスは次のようなコマンドを持ち、マルチキューのコンフィギュレーションに使用します。

- VIRTIO_NET_CTRL_MQ_VQ_PAIRS_SET
- VIRTIO_NET_CTRL_MQ_VQ_PAIRS_MIN
- VIRTIO_NET_CTRL_MQ_VQ_PAIRS_MAX

まとめ

Virtqueue と、これを用いた NIC(virtio-net) の実現方法について解説しました。次号では、これまでの総集編で、仮想化システムの全体像を振り返ります。

ライセンス