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２０回 bhyve における仮想ディスクの実装

はじめに

前回の記事では、bhyve における仮想 NIC の実装について TAP デバイスを用いたホストドライバの実現方法を例に挙げ解説しました。

今回の記事では、bhyve における仮想ディスクの実装について解説していきます。

bhyve における仮想ディスクの実装

bhyve がゲストマシンに提供する仮想 IO デバイスは、全てユーザプロセスである/usr/sbin/bhyve 上に実装されています（図１）。

bhyve は実機上のディスクコントローラと異なり、ホスト OS のファイルシステム上のディスクイメージファイルに対してディスク IO を行います。

これを実現するために、/usr/sbin/bhyve 上の仮想ディスクコントローラは、ゲスト OS からの IO リクエストをディスクイメージファイルへのファイル IO へ変換します。

以下に、ディスク読み込み手順と全体図（図１）を示します。

1. ゲスト OS は virtio-blk ドライバを用いて、共有メモリ上のリングバッファに IO リクエストを書き込みます。そして、IO ポートアクセスによってハイパーバイザにリクエスト送出を通知する。IO ポートアクセスによって VMExit が発生し、CPU の制御がホスト OS の vmm.ko のコードに戻る。bhyve の仮想ディスクコントローラのエミュレーションは、ユーザランドで行われています。vmm.ko はこの VMExit を受けて ioctl を return し/usr/sbin/bhyve へ制御を移す。
2. ioctl の return を受け取った/usr/sbin/bhyve は、仮想ディスクコントローラの共有メモリ上のリングバッファからリクエストを取り出します。
3. ２で取り出したリクエストをパースし、ディスクイメージファイルに read() を行います。
4. 読み出したデータを共有メモリ上のリングバッファに乗せ、ゲスト OS に割り込みを送ります。
5. ゲスト OS は割り込みを受け、リングバッファからデータを読み出します。

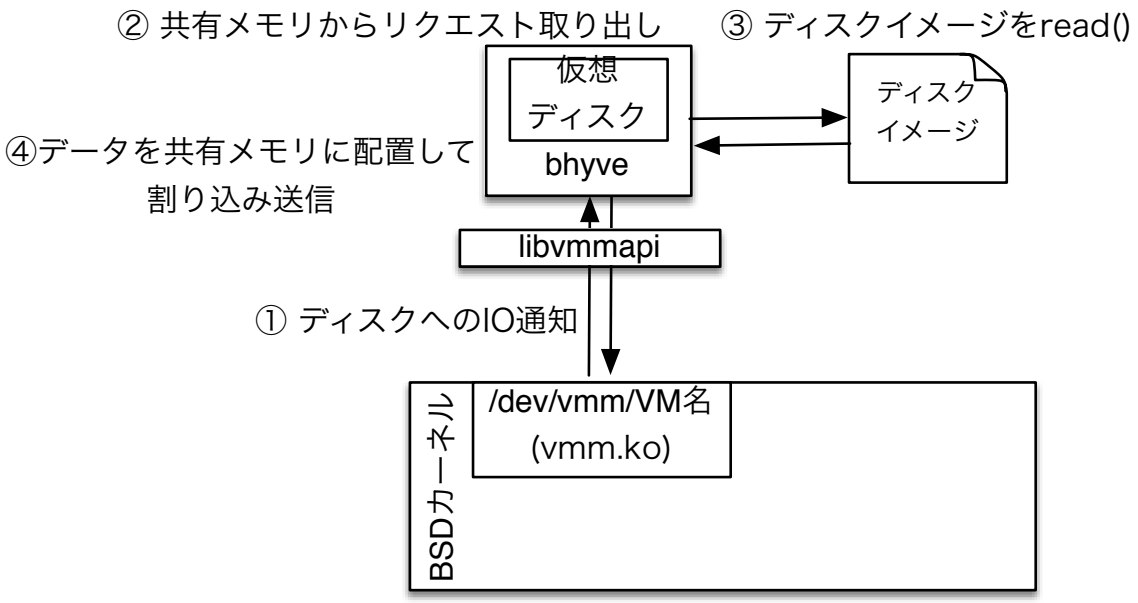

図１　ディスク読み込み手順

書き込み処理では、リクエストと共にデータをリングバッファを用いて送りますが、それ以外は読み込みと同様です。

virtio-blk のしくみ

これまでに、準仮想化 I/O の仕組みとして、virtio と Virtqueue、virtio-net について解説してきました。ここでは、ブロックデバイスを準仮想化する、virtio-blk について解説を行います。

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 からなっていましたが、virtio-blk では単一の Virtqueue を用います。これは、ディスクコントローラの挙動が NIC とは異なり、必ず OS からコマンド送信を行った後にデバイスからレスポンスが返るという順序になるためです[*1]。

ブロック IO のリクエストは連載第１２回の「ゲスト→ホスト方向のデータ転送方法」で解説した手順で送信されます。

virtio-blk では１つのブロック IO リクエストに対して、以下のように Descriptor[*2]群を使用します。１個目の Descriptor は struct virtio_blk_outhdr（表１）を指します。この構造体にはリクエストの種類、リクエスト優先度、アクセス先オフセットを指定します。２～（ｎ－１）個目以降の Descriptor はリクエストに使用するバッファを指します。リクエストが read な場合は読み込み結果を入れる空きバッファを、write な場合は書き込むデータを含むバッファを指定します。

[*1] NIC では OS から何もリクエストを送らなくてもネットワーク上の他のノードからパケットが届くのでデータが送られてきます。このため、必ず OS からリクエストを送ってから届くというような処理にはなりません。
[*2] Virtqueue 上でデータ転送をおこなうための構造体。第１２回の Virtqueue の項目を参照。

バッファのアドレスは物理アドレス指定になるため、仮想アドレスで連続した領域でも物理的配置がバラバラな状態な場合があります。これをサポートするためにバッファ用 Descriptor を複数に別けて確保出来るようになっています。

struct virtio_blk_outhdr にはバッファ長のフィールドがありませんが、これは Descriptor の len フィールドを用いてホストへ通知されます。n 個目の Descriptor は 1byte のステータスコード（表 2）のアドレスを指します。このフィールドはホスト側がリクエストの実行結果を返すために使われます。

type	member	description
u32	type	リクエストの種類（read=0x0, write=0x1, ident=0x8)
u32	ioprio	リクエスト優先度
u64	sector	セクタ番号（オフセット値）

表 1: struct virtio_blk_outhdr

type	member	description
0	OK	正常終了
1	IOERR	IO エラー
2	UNSUPP	サポートされないリクエスト

表 2: ステータスコード

ディスクイメージへの IO

/usr/sbin/bhyve は virtio-blk を通じてゲスト OS からディスク IO リクエストを受け取り、ディスクイメージへ読み書きを行います。bhyve が対応するディスクイメージは RAW 形式のみなので、ディスクイメージへの読み書きはとても単純です。ゲスト OS から指定されたオフセット値とバッファ長をそのまま用いてディスクイメージへ読み書きを行えばよいだけです[*3]。

それでは、このディスクイメージへの IO の部分について bhyve のコードを実際に確認してみましょう。/usr/sbin/bhyve の仮想ディスク IO 処理のコードをコードリスト 1 に示します。

[*3] QCOW2 形式などのより複雑なフォーマットでは未使用領域を圧縮するため、ゲスト・ホスト間でオフセット値が一致しなくなり、またメタデータを持つ必要が出てくるので RAW イメージと比較して複雑な実装になります。

コードリスト１，/usr/sbin/bhyve **の仮想ディスク** IO **処理**

```
/* ゲスト OS から IO 要求があった時に呼ばれる */
static void
pci_vtblk_proc(struct pci_vtblk_softc *sc, struct vqueue_info *vq)
{
    struct virtio_blk_hdr *vbh;
    uint8_t *status;
    int i, n;
    int err;
    int iolen;
    int writeop, type;
    off_t offset;
    struct iovec iov[VTBLK_MAXSEGS + 2];
    uint16_t flags[VTBLK_MAXSEGS + 2];
    /* iov に１リクエスト分の Descriptor を取り出し */
    n = vq_getchain(vq, iov, VTBLK_MAXSEGS + 2, flags);
                    〜 略 〜
        /* 一つ目の Descriptor は struct virtio_blk_outhdr */
    vbh = iov[0].iov_base;
        /* 最後の Descriptor はステータスコード */
    status = iov[--n].iov_base;
                    〜 略 〜
    /* リクエストの種類 */
        type = vbh->vbh_type;
    writeop = (type == VBH_OP_WRITE);
    /* オフセットを sector から byte に変換 */
    offset = vbh->vbh_sector * DEV_BSIZE;
    /* バッファの合計長 */
    iolen = 0;
    for (i = 1; i < n; i++) {
                    〜 略 〜
        iolen += iov[i].iov_len;
    }
                    〜 略 〜
    switch (type) {
    /* WRITE なら pwritev() で iov の配列で表されるバッファリストからディスクイメージへ書き込
み */
```

```
    case VBH_OP_WRITE:
        err = pwritev(sc->vbsc_fd, iov + 1, i - 1, offset);
        break;
    /* READ なら preadv() でディスクイメージから iov の配列で表されるバッファリストへ読み込み */
    case VBH_OP_READ:
        err = preadv(sc->vbsc_fd, iov + 1, i - 1, offset);
        break;
    /* IDENT なら仮想ディスクの identify を返す */
    case VBH_OP_IDENT:
        /* Assume a single buffer */
        strlcpy(iov[1].iov_base, sc->vbsc_ident,
            min(iov[1].iov_len, sizeof(sc->vbsc_ident)));
        err = 0;
        break;
    default:
        err = -ENOSYS;
        break;
    }
                    ～ 略 ～
    /* ステータスコードのアドレスに IO の結果を書き込む */
    if (err < 0) {
        if (err == -ENOSYS)
            *status = VTBLK_S_UNSUPP;
        else
            *status = VTBLK_S_IOERR;
    } else
        *status = VTBLK_S_OK;

                    ～ 略 ～
    /* ステータスコードを書き込んだ事を通知 */
    vq_relchain(vq, 1);
}
```

まとめ

今回は仮想マシンのストレージデバイスについて解説しました。次回は、仮想マシンのコンソールデバイスについて解説します。