システムプログラミング最終課題

慶應義塾大学 環境情報学部 1 年
72375699 二又康輔

2024 年 7 月 21 日

1 外部仕様書

1.1 目的

xv6 にはネットワーク機能が実装されていない。本プロジェクトでは xv6 でパケットの送受信を行うことを目標とした。

1.2 利用ガイド

ネットワーク機能はファイルディスクリプタによって抽象化されている。

1.2.1 デバイスファイルのオープン

root ディレクトリ中に net というデバイスファイルが存在している。システムコール open などを使って net ファイルをオープンする。

1.2.2 パケットの送信

net ファイルオープン時に O_WRONLY などの適切なフラグを付与してやる。パケットの送信は write システムコールを用いて、net ファイルに書き込みを行うことによって実現される。

Listing 1: パケットの送信

```
int fd = open("net", O_WRONLY);
if (fd < 0) {
    fprintf(2, "failed␣to␣open␣net");
    exit(-1);
}
char buf[512];
int n;
while ((n = read(0, buf, 512)) > 0) {
    write(fd, buf, n);
}
close(fd);
```

1.2.3 パケットの受信

net ファイルオープン時に O_RDONLY などの適切なフラグを付与してやる。パケットの受信は read システムコールを用いて、net ファイルから読み込みを行うことによって実現される。1 回の読み込みで 1 つのパケットを得ることができ、読み込んでいないパケットが存在しない場合は、新たなパケットを受信するまでブロッキングされる。

Listing 2: パケットの受信

```
int fd = open("net", O_RDONLY);
if (fd < 0) {
    fprintf(2, "failed to open net");
    exit(-1);
}
char buf[512];
int n = read(fd, buf, 512);
if (n > 0) {
    write(0, buf, n);
}
close(fd);
```

1.3 テスト

パケットの受信は以下のコマンドで ARP パケットを飛ばすことにより確認できる。

```
curl localhost:1234
```

また、パケットの送信は wireshark などで pcap ファイルを解析することによって確かめることができる。

```
wireshark dump.pcap
```

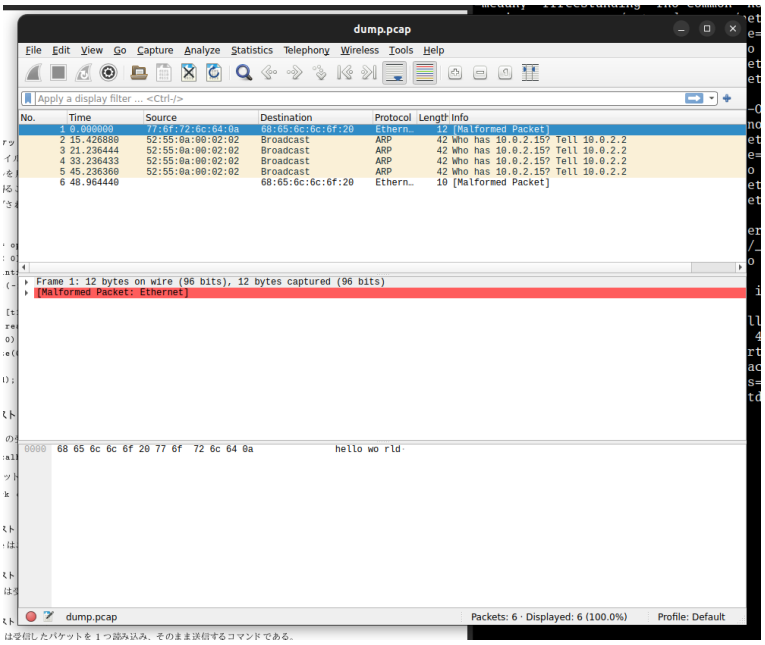

図 1.3.1: wireshark による解析結果の例

また、いくつかのテストプログラムを用意している。

1.3.1 **テストプログラム**: netwrite

netwrite はユーザーが入力した内容をそのまま送信するコマンドである。$cat > net$ と同等の処理を行う。

1.3.2 **テストプログラム**: netread

netread は受信したパケットを 1 つ読み込み出力するコマンドである。

1.3.3 **テストプログラム**: netecho

netecho は受信したパケットを 1 つ読み込み、そのまま送信するコマンドである。

2 内部仕様書

2.1 パケットの送受信

パケットの送受信は VirtIO(MMIO) を用いて実現した。プログラムは kernel/virtio_net.c に記述されている。また、VirtIO 関連の処理の一部については liumOS[2] のソースコードを参考にした。

2.1.1 Virtqueue について

- VirtIO では Virtqueue と呼ばれる構造を用いてデータの送受信を行う。
- Virtqueue は Descriptor Table、Available Ring、Used Ring の 3 つの領域から構成されている。
  - Descriptor Table: やり取りするデータを格納する領域。

  - Available Ring: Driver から Device への送信内容を記述する Queue、Descriptor へのインデックスを保持。
  - Used Ring: Device から Driver への受信内容を記述する Queue、Descriptor へのインデックスを保持。
- virtio-net では送信用と受信用の 2 つの Virtqueue を用いる。

2.1.2 VirtIO の初期化

- 初期化の手順は VirtIO の仕様書 [1] の内容に基づいている。
- 受信用 Virtqueue の Descriptor は事前に Available Ring に登録されている必要がある。
- Device への通知の前では、実行順序が乱れることを防ぐためにメモリバリアが差し込まれている。

Listing 3: VirtIO の初期化

```
static void setup_virtq(uint8 sel, struct virtq *vq) {
    *R(VIRTIO_MMIO_QUEUE_SEL) = sel;

    if (*R(VIRTIO_MMIO_QUEUE_READY)) {
        panic("virtio net should not be ready");
    }

    uint32 max = *R(VIRTIO_MMIO_QUEUE_NUM_MAX);
    if (max == 0) {
        panic("virtio net has no queue 0");
    }
    if (max < NUM) {
        panic("virtio net max queue too short");
    }

    vq->desc = kalloc();
    vq->avail = kalloc();
    vq->used = kalloc();

    if (!vq->desc || !vq->avail || !vq->used) {
        panic("virtio net kalloc");
    }
    memset(vq->desc, 0, PGSIZE);
    memset(vq->avail, 0, PGSIZE);
    memset(vq->used, 0, PGSIZE);

    *R(VIRTIO_MMIO_QUEUE_NUM) = NUM;

    *R(VIRTIO_MMIO_QUEUE_DESC_LOW)      = (uint64)vq->desc;
    *R(VIRTIO_MMIO_QUEUE_DESC_HIGH)     = (uint64)vq->desc >> 32;
    *R(VIRTIO_MMIO_DRIVER_DESC_LOW)     = (uint64)vq->avail;
```

```
    *R(VIRTIO_MMIO_DRIVER_DESC_HIGH)    = (uint64)vq->avail >> 32;
    *R(VIRTIO_MMIO_DEVICE_DESC_LOW)     = (uint64)vq->used;
    *R(VIRTIO_MMIO_DEVICE_DESC_HIGH)    = (uint64)vq->used >> 32;

    *R(VIRTIO_MMIO_QUEUE_READY) = 0x1;
}

void virtio_net_init(void) {
    uint32 status = 0;

    initlock(&net.vnet_lock, "virtio_net");

    if (*R(VIRTIO_MMIO_MAGIC_VALUE) != 0x74726976 || // "virt"(little endian)
        *R(VIRTIO_MMIO_VERSION) != 0x2 || // modern device
        *R(VIRTIO_MMIO_DEVICE_ID) != 1 || // network card
        *R(VIRTIO_MMIO_VENDOR_ID) != 0x554d4551) {
        panic("could␣not␣find␣virtio␣nic");
    }

    // initialize
    *R(VIRTIO_MMIO_STATUS) = status;

    status |= VIRTIO_CONFIG_S_ACKNOWLEDGE;
    *R(VIRTIO_MMIO_STATUS) = status;

    status |= VIRTIO_CONFIG_S_DRIVER;
    *R(VIRTIO_MMIO_STATUS) = status;

    // initialize features bits
    uint64 features = *R(VIRTIO_MMIO_DEVICE_FEATURES);
    features &= ~(1 << VIRTIO_F_ANY_LAYOUT);
    features &= ~(1 << VIRTIO_RING_F_EVENT_IDX);
    features &= ~(1 << VIRTIO_RING_F_INDIRECT_DESC);
    features &= ~(1 << VIRTIO_NET_F_MQ);
    *R(VIRTIO_MMIO_DRIVER_FEATURES) = features;

    status |= VIRTIO_CONFIG_S_FEATURES_OK;
    *R(VIRTIO_MMIO_STATUS) = status;

    setup_virtq(0, &net.rx_vq);
    setup_virtq(1, &net.tx_vq);

    status |= VIRTIO_CONFIG_S_DRIVER_OK;
    *R(VIRTIO_MMIO_STATUS) = status;
```

```
    // initialize rx_vq
    // memory barrier(for NOTIFY)
    __sync_synchronize();

    for (int i = 0; NUM > i; i++) {
        net.rx_vq.desc[i].addr = (uint64)kalloc();
        free_desc(&net.rx_vq, i, VRING_DESC_F_WRITE);
        set_avail(&net.rx_vq, i);
    }

    __sync_synchronize();

    *R(VIRTIO_MMIO_QUEUE_NOTIFY) = 0;

    // initialize tx_vq
    __sync_synchronize();

    for (int i = 0; NUM > i; i++) {
        net.tx_vq.desc[i].addr = (uint64)kalloc();
        free_desc(&net.tx_vq, i, 0);
    }

    __sync_synchronize();

    *R(VIRTIO_MMIO_QUEUE_NOTIFY) = 1;

    // print mac address
    struct virtio_net_config *cfg
        = (struct virtio_net_config *)R(VIRTIO_MMIO_CONFIG);
    printf("mac: %x:%x:%x:%x:%x:%x\n", cfg->mac[0], cfg->mac[1], cfg->mac[2],
        cfg->mac[3], cfg->mac[4], cfg->mac[5]);
}
```

2.1.3 パケットの送信

- 空の header を Descriptor に書き込む。
- 入力 buffer の内容を Descriptor にチェイン構造で書き込む。
- Available Ring に先頭の Descriptor を登録する。
- Device に Virtqueue の更新を通知する。
- Device の処理が完了するまで待機する。
- Descriptor を解放する。

Listing 4: パケットの送信

```
int virtio_net_send(void *buf, int buf_size) {
```

```
acquire(&net.vnet_lock);

struct virtq *vq = &net.tx_vq;

struct virtio_net_hdr hdr = {0};

int idx = 0;
int offset = 0;
memmove((void *)vq->desc[idx].addr, &hdr, sizeof(struct virtio_net_hdr));
vq->desc[idx].len = sizeof(struct virtio_net_hdr);
vq->desc[idx].flags = VRING_DESC_F_NEXT;
vq->desc[idx].next = idx + 1;
idx++;

for (; buf_size > 0 && NUM > idx; idx++) {
    int size = MIN(PGSIZE, buf_size);
    memmove((void *)vq->desc[idx].addr, buf + offset, size);
    buf_size -= size;
    offset += size;

    vq->desc[idx].len = size;
    if (buf_size > 0 && idx != NUM - 1) {
        vq->desc[idx].flags = VRING_DESC_F_NEXT;
        vq->desc[idx].next = idx + 1;
    }
    else {
        vq->desc[idx].flags = 0;
        vq->desc[idx].next = 0;
    }
}

vq->avail->ring[vq->avail->idx % NUM] = 0;

__sync_synchronize();

vq->avail->idx++;

__sync_synchronize();

*R(VIRTIO_MMIO_QUEUE_NOTIFY) = 1;

vq->used_idx++;
while (vq->used_idx != vq->used->idx) {
    __asm__("nop");
}
```

```
    for (int i = 0; idx > i; i++) {
        free_desc(vq, i, 0);
    }

    release(&net.vnet_lock);
    return offset;
}
```

2.1.4 パケットの受信

- Device によって Used Ring に Descriptor が登録されるまで待機する。
- 受信内容を buffer に書き込む。
- Descriptor を解放して、Available Ring に登録する。

Listing 5: パケットの受信

```
int virtio_net_recv(void *buf, int buf_size) {
    acquire(&net.vnet_lock);

    struct virtq *vq = &net.rx_vq;

    while (vq->used_idx == vq->used->idx) {
        __asm__("nop");
    }

    int idx = vq->used->ring[vq->used_idx % NUM].id;
    int offset = 0;
    while (buf_size > 0) {
        int size = MIN(vq->used->ring[vq->used_idx % NUM].len, buf_size);
        memmove(buf, (void *)(vq->desc[idx].addr + offset), size);
        buf_size -= size;
        offset += size;

        if (vq->desc[idx].flags & VRING_DESC_F_NEXT) {
            idx = vq->desc[idx].next;
        }
        else {
            break;
        }
    }

    idx = vq->used->ring[vq->used_idx % NUM].id;
    while (1) {
        int flag = vq->desc[idx].flags;
        int nxt = vq->desc[idx].next;
```

```
            free_desc(vq, idx, VRING_DESC_F_WRITE);
            set_avail(vq, idx);

            if (flag & VRING_DESC_F_NEXT) {
                idx = nxt;
            }
            else {
                break;
            }
        }

        vq->used->ring[vq->used_idx % NUM].len = 0;
        vq->used_idx++;

        release(&net.vnet_lock);
        return offset;
}
```

2.2 ネットワーク機能の抽象化

ネットワーク機能はファイルディスクリプタで抽象化されている。プログラムは kernel/net.c に記述されている。

- netwrite はユーザー空間にある送信データを buffer にコピーして、virtio_net_send で送信する。
- netread は virtio_net_recv で受信したデータをユーザー空間にコピーする。
- netread と netwrite はそれぞれデバイスファイルに対する read、write の処理として登録される。

Listing 6: ネットワーク機能の抽象化

```
int netwrite(int user_src, uint64 src, int n) {
    acquire(&net.lock);

    int retval;
    int size = MIN(n, PGSIZE);
    if (either_copyin(&net.buf, user_src, src, size) == -1) {
        retval = -1;
        goto end;
    }
    int real_size = virtio_net_send(&net.buf, size);
    retval = real_size;

end:
    release(&net.lock);
    return retval;
```

```
}

int netread(int user_dst, uint64 dst, int n) {
    acquire(&net.lock);

    int retval;
    int size = MIN(n, PGSIZE);
    int real_size = virtio_net_recv(&net.buf, size);
    if (either_copyout(user_dst, dst, &net.buf, real_size) == -1) {
        retval = -1;
        goto end;
    }
    retval = real_size;

end:
    release(&net.lock);
    return retval;
}

void netinit(void) {
    initlock(&net.lock, "net");

    devsw[NET].read = netread;
    devsw[NET].write = netwrite;
}
```

参考文献

[1] Virtual I/O Device (VIRTIO) Version 1.1 https://docs.oasis-open.org/virtio/virtio/v1.1/csprd01/virtio-v1.1-csprd01.html

[2] liumOS https://github.com/hikalium/liumos/tree/main

[3] Virtual I/O Device (VritIO) について e-trees.Japan 開発ブログ https://e-trees.jp/wp/?p=1268

[4] virtio-net の概要とアーキテクチャ bobuhiro11's blog https://blog.bobuhiro11.net/2020/08-21-virtio.html

[5] MikanOS に NIC ドライバを実装する - 送受信編 https://qiita.com/Saza-ku/items/6da0f2fb804620719a71