Universidade Federal de Santa Catarina – UFSC
INE5424 - Sistemas Operacionais II
Prof. Antônio Augusto Fröhlich

Seminário H:
I/O Bus Operation

Guilherme Antônio Ferreira da Silva
João Fellipe Uller

Florianópolis, 14 de Outubro de 2020

RISC-V I/O

Para lidar com dispositivos de entrada/saída, a ISA do RISC-V não prevê nenhum tipo de instrução especial para controle ou leitura/escrita de um barramento. Ao invés disso, é previsto um esquema de entrada e saída mapeado em memória (Memory Mapped I/O). Assim, cada um dos dispositivos de entrada e saída é mapeado para um ou mais endereços no espaço de endereçamento dos processos, que passam a ser monitorados por esses dispositivos a fim de interceptar requisições de I/O para os mesmos.

Desse modo, operações de entrada e saída são realizadas utilizando as instruções de load/store comuns que lidam com manipulação de dados da memória, permitindo a utilização de todos os modos de acesso previstos na especificação. Isso simplifica a programação, quando comparado com acessos diretos aos dispositivos, além de forçar proteção de acessos através do espaço de endereçamento, evitando que *threads* do usuário acessem esses endereços diretamente. Por outro lado, esse tipo de solução incorre na necessidade de circuitos de decodificação em *hardware* mais complexos, uma vez que precisa-se de uma decodificação total dos endereços virtuais para físicos para sua utilização.

Além disso, a utilização de entrada e saída mapeada em memória permite a configuração e a utilização de uma Direct Memory Access (DMA) para realizar as operações de transferência de dados entre os registradores do dispositivo e o buffer de memória. Reduzindo, dessa forma, o tempo perdido pela CPU bloqueando à espera de uma leitura ou uma escrita na memória e aumentando a performance do sistema como um todo.

Memory Mapped I/O (MMIO)

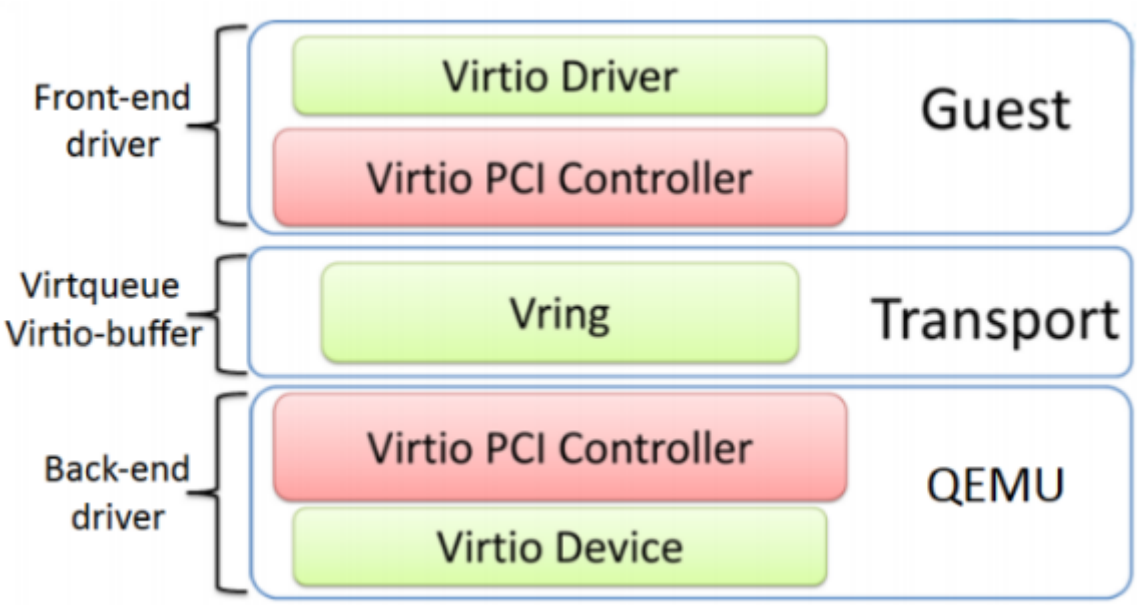
I/O Ordering

VirtIO

O VirtIO é um protocolo de comunicação entre dispositivos virtuais, criado com o objetivo de definir uma API para unificar drivers e facilitar o uso e configuração desses dispositivos, visto que existem diversos sistemas de virtualização diferentes e cada um desses realiza a sua própria implementação. Para alcançar tais objetivos, o virtIO define uma série de abstrações e estruturas, mas de maneira geral seus principais componentes de funcionamento são 3 (três): *device status field*, *feature bits* e uma interface chamada *Virtqueue*.

O *device status field* é uma sequência de bits utilizada pelo dispositivo em conjunto com o sistema operacional para realizar a inicialização do mesmo. Os bits que podem ser setados nesse campo são: **ACKNOWLEDGE**, indicando que o dispositivo foi reconhecido, **DRIVER** para indicar que a inicialização começou, **DRIVER_OK** e **FEATURES_OK** para sinalizar que a comunicação está pronta para começar. Por fim, **DEVICE_NEEDS_RESET** para indicar uma falha fatal do lado do dispositivo e **FAILED** pelo lado do sistema operacional. Já o *feature bits field* é utilizado para comunicar sobre quais features são suportadas pelo dispositivo e entrar em acordo com o sistema operacional sobre quais dessas serão utilizadas. Por exemplo, um dispositivo de interface de rede poderia escolher entre checksumming ou scatter-gather realizar o offload.

Por fim, temos a Virtqueue, uma interface alocada na memória do guest (dispositivo) e que realiza a comunicação entre dois grupos: front-end, onde os pedidos de I/O do processo usuário são realizados e então transferidos para o back-end, que os recebe e então executa as operações utilizando os dispositivos físicos.



Estrutura da Virtqueue

A Virtqueue é implementada usando uma estrutura chamada *Vring*, que consiste basicamente de 3 (três) partes: um array de descriptors, onde são inseridos os dados, um *available ring*, usado para a comunicação do sistema operacional com o dispositivo e um *used ring*, utilizado para a comunicação no sentido do dispositivo para o sistema operacional.

Os descriptors contém um endereço físico de 64 bits de buffer junto com o seu tamanho, além de um campo para flags, que podem ser duas: uma para indicar se existe um encadeamento de buffers, permitindo o uso de memória não contígua. A outra flag indica o tipo do buffer, se é *read-only* ou *write-only*. Por fim, o campo next aponta para o próximo descriptor, se o mesmo existir. Os outros dois componentes da Virtqueue são chamados de available ring e used ring. O motivo de se utilizar essa estrutura em anéis é permitir assincronicidade, fazendo com que diversas requisições possam ser feitas sem ter que esperar por uma específica finalizar primeiro.

```
struct vring_desc
{
  __u64 addr;
  __u32 len;
  __u16 flags;
  __u16 next;
};
```

Quando queremos realizar uma ação com o dispositivo, devemos preencher os valores desse descriptor e então adicionar o seu índice dentro de uma lista no available ring. Dessa forma, ao notificar o sistema operacional, o mesmo irá verificar qual descriptor deve ser lido. A estrutura de um available ring possui seu índice e uma flag, onde é possível desabilitar interrupções, enquanto a constante NUM é negociada entre o dispositivo e o sistema operacional quando o mesmo é inicializado.

```
struct vring_avail
{
  __u16 ring[NUM];
  __u16 flags;
  __u16 idx;
};
```

Ainda, temos os used rings, onde o dispositivo pode enviar algumas informações ao sistema operacional, como por exemplo indicar que uma operação foi finalizada. Sua estrutura é muito semelhante a do available ring, com a diferença que aqui o SO tem que procurar qual descriptor realizou a notificação.

```
struct vring_sed
{
  __u16 flags;
  __u16 idx;
  __u16 avail_event;
  Elem ring[NUM];
};

struct Elem {
  u32 id;
  u32 len;
}
```

Outra diferença entre o available e o used ring é que aqui, no used ring, o campo *idx* indica o primeiro elemento não usado do anel. Isso é utilizado pois dentro da estrutura 'Elem', também temos o índice do descriptor. Assim, a cada leitura realizada, o dispositivo incrementa o índice em used ring. Dessa forma, se os índices não são iguais, significa que existe ainda dados a serem lidos. Tanto o sistema operacional quanto o dispositivo acompanham o estado desses anéis, garantindo que estejam falando sobre a mesma requisição.

Por fim, para indicar onde procurar na memória pelos descriptors, os dispositivos do virtIO tem um registrador chamado *QueuePFN (Page Field Number)* , que contém o endereço de memória física para o qual foi mapeado. Com esse componentes, conseguimos então realizar todo o ciclo de uma requisição. Iniciamos apontando para o endereço de memória existente em QueuePFN e então, quando um descriptor for preenchido com dados, notificamos que uma requisição foi realizada através de um outro registrador, chamado *QueueNotify*, escrevendo um valor qualquer no mesmo. O processo se inicia e quando o mesmo finalizar, uma interrupção externa será enviada pelo dispositivo, de forma que o sistema operacional precisa apenas verificar o used ring e obter os dados.

Exemplo de configuração e uso de um dispositivo

Referências

[1] RISC-V Foundation. *The RISC-V Instruction Set Manual Volume I: Unprivileged ISA*, mar 2019. Disponivel em: <https://riscv.org//wp-content/uploads/2019/06/riscv-spec.pdf>. Acessado em: 29 set. 2020.