Desenvolvimento de Driver para GNU/Linux

Adalberto Ribeiro Sampaio Junior
Everton Lima Aleixo
Pedro Henrique Pires de Castro

Sumário

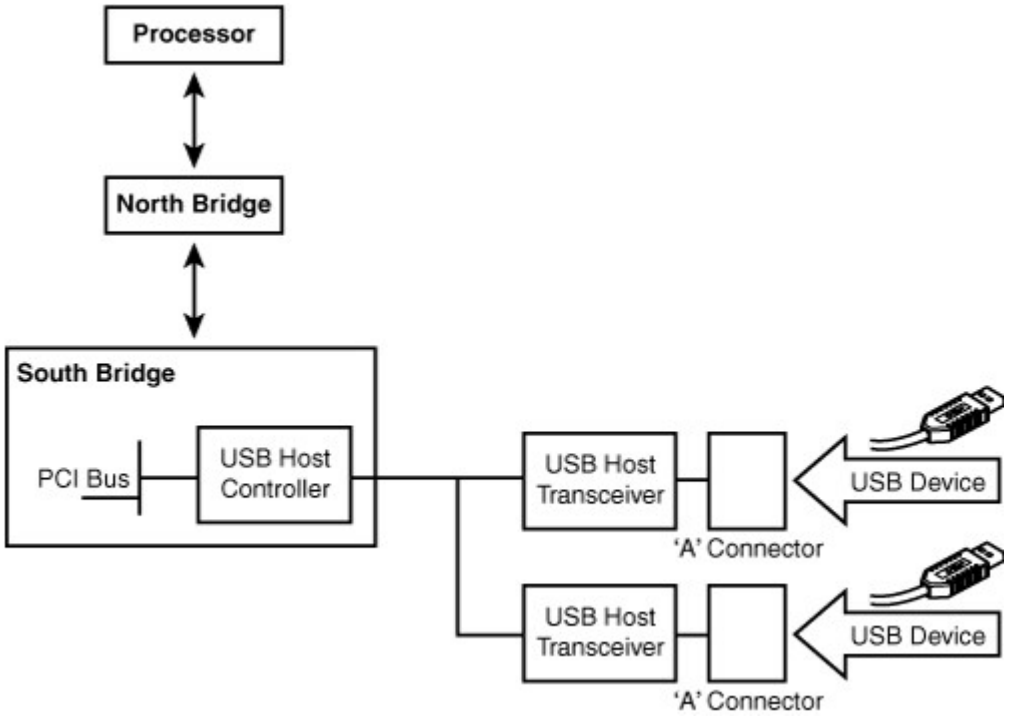
Conceitos

- Arquitetura USB
- Estruturas de Dados
- Enumeração
- Inicialização e *Probing*
- Acessando registradores
- Transferência de dados

Demonstração

Conceitos

Arquitetura USB



Arquitetura USB

Transferência

- Control → Configuração e controle
- Bulk → Grande quantidade de dados
- Interrupt → Pequena quantidade de dados
- Isochronous → Dados em tempo-real

Endereçamento

- *Endpoints*
  - Possui endereço único
  - *endpoint 0* → Configuração

Estruturas de Dados

Representação do dispositivo

```
<include/linux/usb/usb.h>

struct usb_device{
            /* ... */
            enum usb_device_state state;                /* Configurado, Desligado, etc */
            enum usb_device_speed speed;                /* Alta/Baixa (ou 'error') */
            /* ... */
            struct usb_device_descriptor descriptor;    /* Descritor */
            struct usb_host_config *config;             /* Todos os configs */
            struct usb_host_config *actconfig;          /* Config ativo */
            /* ... */
};
```

Estruturas de Dados

URB

```
<include/linux/usb/usb.h>

struct urb
{
  struct kref kref;             /* Referencia da URB */
  /* ... */
  struct usb_device *dev;       /* Ponteiro associado ao dispositivo */
  unsigned int pipe;            /* Informação do Pipe */
  int status;                   /* Status não-ISO */
  unsigned int transfer_flags; /* URB_SHORT_NOT_OK | ...*/
  void *transfer_buffer;        /* Associado ao buffer de dados */
  dma_addr_t transfer_dma;      /* Endereço do DMA para transferência para o buffer */
  int transfer_buffer_length;  /* Tamanho do buffer */
  /* ... */
  unsigned char *setup_packet; /* Pacote de configuração */
  /* ... */
  int interval;                 /* Intervalo de transferência (INT/ISO) */
  /* ... */
  usb_complete_t complete;      /* Rotina de conclusão */
  /* ... */
};
```

Estruturas de Dados

URB

- Criação
- Preenchimento
- Submissão
- Finalização

Estruturas de Dados

URB

- Criação

```
usb_alloc_urb()
```

- Preenchimento

```
void usb_fill_[control|int|bulk]_urb(
      struct urb *urb,              /* Ponteiro para URB */
      struct usb_device *usb_dev,   /* Estrutura do dispositivo USB */
      unsigned int pipe,            /* Codificação do Pipe */
      unsigned char *setup_packet,  /* Apenas para URB do tipo Control */
      void *transfer_buffer,        /* Buffer para I/O */
      int buffer_length,            /* Tamanho do buffer de I/O */
      usb_complete_t completion_fn, /* Rotina de callback */
      void *context,                /* Utilizado pela rotina de conclusão */
      int interval);                /* Apenas para URB do tipo Interrupt */
```

Estruturas de Dados

URB

- Preenchimento
  - Mascarar conexão síncrona

```
int usb_[control|interrupt|bulk]_msg(
      struct usb_device *usb_dev,
      unsigned int pipe,
      void *data,
      int len,
      int *actual_length,
      int timeout);
```

Estruturas de Dados

URB

- Submissão

```
usb_submit_urb()
```

- Finalização

```
usb_free_urb          /* limpa a referência para uma URB que foi completada */
usb_unlink_urb        /* cancela uma operação de URB pendente */
```

Estruturas de Dados

Pipe

- Canal de comunicação entre host e endpoint do dispositivo
- É um inteiro codificado da combinação de:
  - Endereço do endpoint
  - Direção da transferência de dados
  - Tipo de transferência de dados

```
<include/linux/usb/usb.h>

usb_[rcv|snd][ctrl|int|bulk|isoc]pipe(struct usb_device *usb_dev, __u8 endpointAddress);
```

Estruturas de Dados

Descritores

```
<include/linux/usb/ch9.h>


usb_device_descriptor        /* Contém informações gerais do dispositivo, como Vendor ID e Product ID */
usb_config_descriptor        /* É utilizado para descrever diferentes modos de configuração,
                                como "bus-powered" e "self-powered" */
usb_interface_descriptor     /* Permite o dispositivo USB suportar múltiplas funções */
usb_endpoint_descriptor      /* Carrega informações referentes ao dispositivo */
```

Enumeração

- Reconhecimento de que o dispositivo foi plugado
- Após o reconhecimento, o método probe() associado ao Driver do cliente é invocado.

Inicialização e *Probing*

- Identificação do dispositivo

```
struct usb_device_id {
        /* ... */
        __u16 idVendor;         /* Vendor ID */
        __u16 idProduct;        /* Product ID */
        /* ... */
        __u8 bDeviceClass;      /* Classe do Dispositivo */
        __u8 bDeviceSubClass;   /* SubClasse do Dispositivo */
        __u8 bDeviceProtocol;   /* Protocolo do Dispositivo */
        /* ... */
};
```

- usb_device_init()
    - Chama usb_register() para registrar a estrutura usb_driver no USB core

Inicialização e *Probing*

- Estrutura usb_driver

```
static struct usb_driver device_driver
{
    .name       = "device_name",
    .probe      = device_probe,
    .disconnect = device_disconnect,
    .id_table   = device_ids,
};
```

Inicialização e *Probing*

- Estrutura do dispositivo

```
typedef struct {
          struct usb_device      *usbdev;        /* Representação do dispositivo */
          struct usb_interface   *interface;     /* Representação da interface*/
          struct urb             *ctrl_urb;      /* URB do tipo Control para acessa registros */
          struct usb_ctrlrequest  ctrl_req;      /* Requisição Control */
          unsigned char          *bulk_in_buf;   /* Buffer de dados recebidos */
          size_t                  bulk_in_len;   /* Tamanho do buffer de recebimento */
          __u8                    bulk_in_addr;  /* Endereço do endpoint IN */
          __u8                    bulk_out_addr; /* Endereço do endpoint OUT */
          /* ... */                              /* Travas, filas de espera, estatísticas.. */
} device_t;
```

Inicialização e *Probing*

- probe()
  - Aloca memória para a estrutura do dispositivo
  - Inicializa os campos, em device_t, referentes aos *endpoints*
  - Exporta o dispositivo de caractere /dev/“device_name” para o espaço do usuário

Inicialização e *Probing*

- probe()
  - Aplicações rodam sobre /dev/“device_name” para trocar dados com o dispositivo
  - device_probe() invoca usb_register_dev() e fornece as file_operations na interface por meio da estrutura usb_class_driver

Inicialização e *Probing*

- probe()

```
static struct file_operations device_fops =
{
  .owner    =  THIS_MODULE,    /* Dono do módulo */
  .read     =  device_read,    /* Método de leitura */
  .write    =  device_write,   /* Método de escrita */
  .ioctl    =  device_ioctl,   /* Método de controle de I/O */
  .open     =  device_open,    /* Método de abertura */
  .release =  device_release, /* Método de fechamento */
};

static struct usb_class_driver device_class = {
  .name       = "device_name",
  .fops       = &tele_fops,      /* Conecta com /dev/"device_name" */
  .minor_base = TELE_MINOR_BASE, /* Minor number inicial */
};
```

Inicialização e *Probing*

- Guardar referência do endereço da estrutura do dispositivo, entre as invocações do probe() e do open():
  - usb_set_intfdata()
  - usb_get_intfdata()

Acessando registradores

• O método open() inicializa a configuração dos registradores quando uma aplicação acessa /dev/"device_name"

• Para configurar o conteúdo do registrador, device_open() submete uma URB do tipo Control para o *endpoint* 0

Acessando registradores

```
struct usb_ctrlrequest {
        __u8 bRequestType;    /* qualifica a requisição pela codificação da direção
                                 da transferência dos dados, categoria da requisição
                                 e se o receptor é um dispositivo, interface, endpoint ou outra coisa */
        __u8 bRequest;        /* identifica a requisição do tipo control */
        __le16 wValue;        /* dados a serem escritos nos registradores */
        __le16 wIndex;        /* intervalo desejado no espaço do registrador */
        __le16 wLength;       /* número de dados a serem transmitidos */
} __attribute__ ((packed));
```

Acessando registradores

- device_open()
  - Aloca uma URB do tipo Control
  - Cria uma usb_ctrlrequest e configura com identificação, tipo, intervalo e valores a serem programados
  - Cria um pipe do tipo Control para o *endpoint* 0
  - Inicializa campos na URB usando usb_fill_control_urb()
  - Submete URB ao USB core usando usb_submite_urb()
  - Espera até que a função callback sinalize que o registro está completo
  - Retorna status

Transferência de Dados

- device_read()
- device_write()

Demonstração


SOS