UFPR
UNIVERSIDADE FEDERAL DO PARANÁ

Sistemas Operacionais

Gestão de entrada/saída - software

Prof. Carlos Maziero

DInf UFPR, Curitiba PR

Abril de 2019

Conteúdo

1 Software de Entrada/saída

2 Drivers

3 Estratégias de interação
- Entrada/saída por programa
- Entrada/saída por eventos
- Entrada/saída por DMA

4 Tratamento de interrupções

UFPR
UNIVERSIDADE FEDERAL DO PARANÁ

Software de entrada/saída

Software que interage com os dispositivos:

- drivers
- abstrações de baixo nível (sockets, blocos, etc)

Grande diversidade de dispositivos

- muitos drivers distintos
- 70% do núcleo Linux

Estrutura geral

Drivers:

- Interagem com os dispositivos
- portas de E/S, interrupções

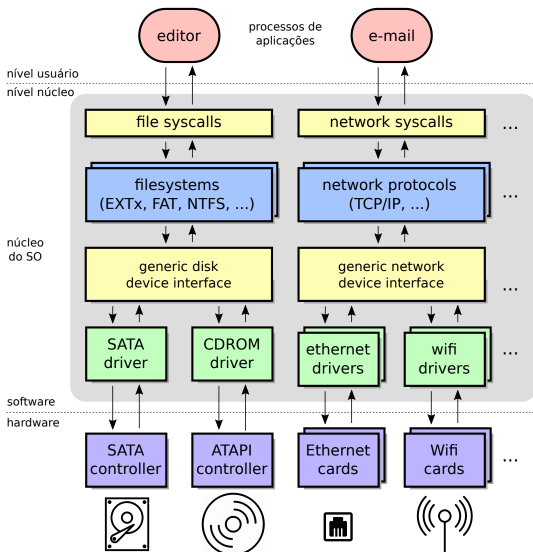
Dispositivos genéricos:

- Visão genérica de dispositivos similares
- NDIS - *Network Device Interface Specification*

Abstrações: Sistemas de arquivos, protocolos de rede

Chamadas de sistema: interface oferecida aos processos

Estrutura geral



Classes de dispositivos

- **Dispositivos orientados a caracteres**
  - transferências de dados byte a byte, sequencial
  - USB, *mouse*, teclado, modems, etc.
- **Dispositivos orientados a blocos**
  - transferências feitas em blocos de bytes com endereços
  - Discos, fitas e dispositivos de armazenamento
- **Dispositivos de rede**
  - mensagens são blocos de dados de tamanho variável
  - envios de blocos de forma sequencial
- **Dispositivos gráficos**
  - renderização de texto e gráficos em terminais
  - funções de configuração e *framebuffer* compartilhado

Driver

Componente de baixo nível do SO:

- Executa geralmente em modo núcleo
- Interage com o hardware do dispositivo
- Um driver para cada modelo ou família de modelos

Funcionalidades:

- Funções de entrada/saída
- Funções de gerência
- Funções de tratamento de eventos

Estrutura de um driver

Funções de entrada/saída:

- transferência de dados entre dispositivo e o SO
- caracteres, blocos, pacotes

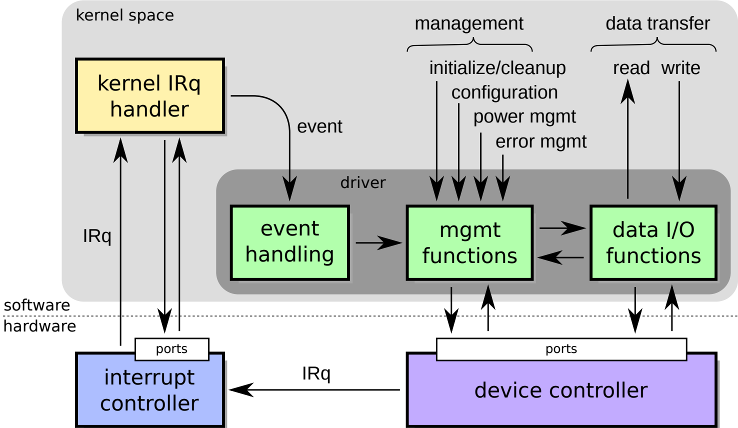
Funções de gerência:

- inicialização e configuração do dispositivo
- inicialização e configuração do *driver*
- Syscalls: `ioctl()`, `DeviceIoControl()`

Funções de tratamento de eventos:

- Tratamento de interrupções geradas pelo dispositivo

Estrutura de um driver



Estratégias de interação

Como o núcleo interage com os dispositivos?

Através dos *drivers*!

Os *drivers* implementam **estratégias de interação**:

- Por programa
- Por eventos
- Acesso direto à memória

Entrada/saída por programa

Estratégia de entrada/saída mais simples

- Também chamada **varredura** ou **polling**

O *driver* pede a operação e aguarda sua conclusão:

1. Espera o dispositivo estar pronto (*status*)
2. Escreve dado na porta de saída (*data out*)
3. Escreve comando (*control*)
4. Espera dispositivo concluir a operação (*status*)

Exemplo: porta paralela simples

P_0 (*data port*): porta de saída (e de entrada), 8 bits

P_1 (*status port*), 8 bits

0. reservado
1. reservado
2. $\overline{\texttt{nIRQ}}$: se 0, gerou uma interrupção
3. `error`: há um erro interno na impressora
4. `select`: a impressora está pronta (*online*)
5. `paper_out`: falta papel na impressora
6. $\overline{\texttt{ack}}$: se 0, dado foi recebido
7. `busy`: controlador está ocupado

Exemplo: porta paralela simples

P_2 (*control port*):

0. $\overline{\texttt{strobe}}$: há um dado em P_0
1. `auto_lf`: *line feed* a cada *carriage return*
2. `reset`: a impressora deve ser reiniciada
3. `select`: a impressora está selecionada para uso
4. `enable_IRQ`: permite gerar interrupções
5. `bidirectional`: ativa modo bidirecional
6. reservado
7. reservado

P_3 a P_7: usadas nos modos estendidos (EPP e ECP)

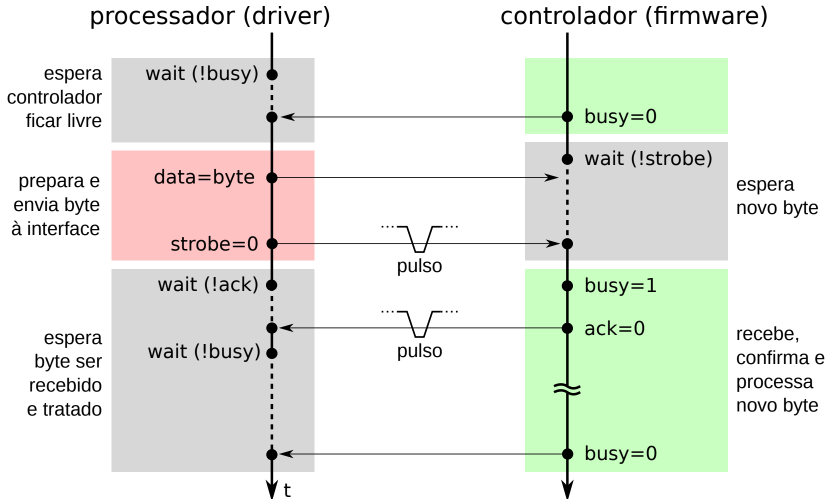
Entrada/saída por programa

```
1  // portas da interface paralela LPT1 (endereço inicial em 0378h)
2  #define P0    0x0378        # porta de dados
3  #define P1    0x0379        # porta de status
4  #define P2    0x037A        # porta de controle
5
6  // bits de controle e status das portas
7  #define BUSY  7             # bit 7 da porta de status
8  #define ACK   6             # bit 6 da porta de status
9  #define STROBE 0            # bit 0 da porta de controle
10
11 // operações em bits individuais de bytes
12 #define BIT_SET(a,n) (a |= 1 << n) // n-esimo bit de a = 1
13 #define BIT_CLR(a,n) (a &= 1 << n) // n-esimo bit de a = 0
14 #define BIT_TEST(a,n) (a & 1 << n) // testa n-esimo bit de a
```

Entrada/saída por programa (cont.)

```
1   void polling_output (char c)
2   {
3     // espera o controlador ficar livre, testando a porta de status (P1)
4     while (BIT_TEST (in(P1), BUSY)) ;
5
6     // escreve o byte "c" a enviar na porta de dados (P0)
7     out (P0, c) ;
8
9     // gera pulso em 0 no bit STROBE da porta de controle (P2),
10    // para indicar ao controlador que há um novo dado em P0
11    out (P2, BIT_CLR (in(P2), STROBE)) ;    // bit STROBE de P2 = 0
12    usleep (1) ;                            // aguarda 1 us
13    out (P2, BIT_SET (in(P2), STROBE)) ;    // bit STROBE de P2 = 1
14
15    // espera controlador receber o dado (pulso em 0 no bit ACK de P1)
16    while (BIT_TEST (in(P1), ACK)) ;
17
18    // espera o controlador concluir a operação solicitada
19    while (BIT_TEST (in(P1), BUSY)) ;
20  }
```

Entrada/saída por programa



Entrada/saída por eventos

Estratégia básica:

1. requisitar a operação desejada
2. suspender o fluxo de execução atual
3. ao concluir a operação, o dispositivo gera uma IRq
4. o *driver* retoma a operação de E/S

A operação de E/S pelo *driver* é dividida em duas etapas:

1. uma função de E/S inicia a operação
2. uma função de tratamento de interrupção a continua

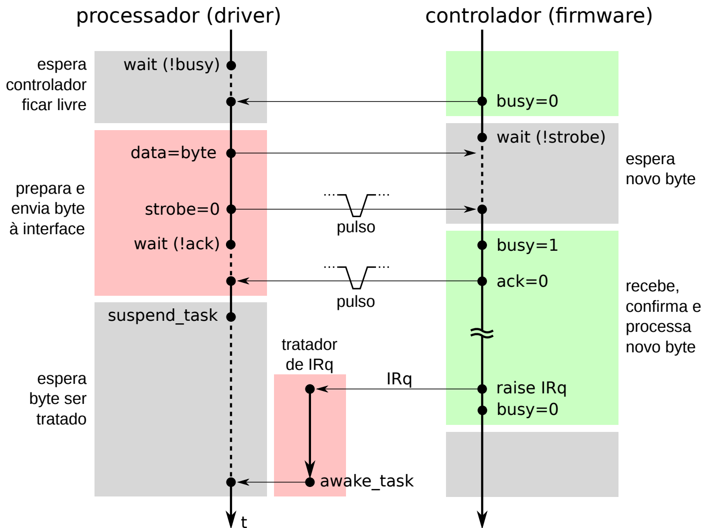
Entrada/saída por eventos

```
1   // saída de dados por evento: solicitação de operação
2   void event_output (char c)
3   {
4     // espera o controlador ficar livre, testando a porta de status (P1)
5     while (BIT_TEST (in(P1), BUSY)) ;
6
7     // escreve o byte "c" a enviar na porta de dados (P0)
8     out (P0, c) ;
9
10    // gera pulso em 0 no bit STROBE da porta de controle (P2),
11    // para indicar ao controlador que há um novo dado em P0
12    out (P2, BIT_CLR (in(P2), STROBE)) ;    // bit STROBE de P2 = 0
13    usleep (1) ;                            // aguarda 1 us
14    out (P2, BIT_SET (in(P2), STROBE)) ;    // bit STROBE de P2 = 1
15
16    // espera controlador receber o dado (pulso em 0 no bit ACK de P1)
17    while (BIT_TEST (in(P1), ACK)) ;
```

Entrada/saída por eventos

```
1    // suspende a execução, liberando o processador para outras tarefas
2    // enquanto o controlador está ocupado processando o dado recebido.
3    suspend_task () ;
4  }
5
6
7  // saída de dados por evento: tratamento da interrupção
8  void event_handle ()
9  {
10   // o controlador concluiu sua operação, acordar a tarefa solicitante.
11   awake_task () ;
12 }
```

Entrada/saída por eventos



Acesso direto à memória

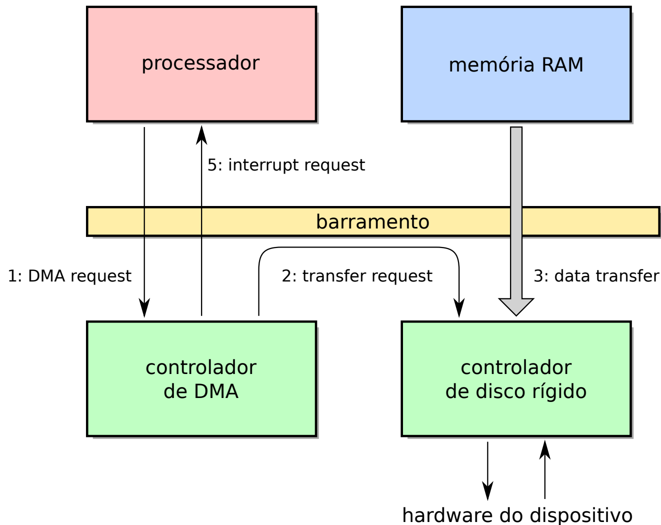
Transferência direta entre dispositivo e RAM, sem a CPU

Muito usado para transferir grandes volumes de dados

Passos:

1. a CPU programa o controlador DMA com os parâmetros (endereços e tamanho da transferência)
2. o controlador de DMA interage com o controlador do disco para transferir os dados da RAM
3. o controlador do disco recebe os dados da RAM
4. essa operação pode ser repetida caso necessário
5. no final, o controlador de DMA interrompe a CPU

Acesso direto à memória



Acesso direto à memória

```
// requisição da operação de saída através de DMA
void dma_output ()
{
  // solicita uma operação DMA, informando os dados da transferência
  // através da estrutura "dma_data"
  request_dma (dma_data) ;

  // suspende o processo solicitante, liberando o processador
  suspend_task () ;
}

// rotina de tratamento da interrupção do controlador de DMA
void interrupt_handle ()
{
  // informa o controlador de interrupções que a IRq foi tratada
  acknowledge_irq () ;

  // saída terminou, acordar o processo solicitante
  awake_task (...) ;
}
```

Tratamento de interrupções

Interrupções são tratadas por *handlers* (tratadores):

- compõem a estrutura dos *drivers*
- operam usualmente com interrupções inibidas
- devem ser muito rápidos, portanto simples

A maioria dos SOs trata interrupções em **dois níveis**:

- FLIH - *First-Level Interrupt Handler*
- SLIH - *Second-Level Interrupt Handler*

Tratamento de interrupções

- FLIH - *First-Level Interrupt Handler*
  - *hard, fast, top-half interrupt handler*
  - tratador primário (rápido)
  - recebe a IRq e a reconhece junto ao PIC
  - registra dados da IRq em uma fila de eventos
- SLIH - *Second-Level Interrupt Handler*
  - *soft, slow, bottom-half interrupt handler*
  - tratador secundário (lento)
  - trata a fila de eventos
  - escalonado com os demais processos e threads