

Arquitetura de Computadores

Aula T25 – 06 Junho de 2023

Dispositivos de entrada/saída: interrupções (conclusão), DMA (Direct Memory Access)

Integração dos periféricos no sistema operativo: periféricos orientados para a transferência por bytes e por blocos. Cache de blocos

Bibliografia:

OSTEP Cap. 36

<https://pages.cs.wisc.edu/~remzi/OSTEP/file-devices.pdf>

Objectivos da gestão de dispositivos de entrada/saída

Objectivos

- Apresentar uma visão mais abstracta dos dispositivos de comunicação e armazenamento, escondendo aos seus utilizadores (processos) detalhes sobre o funcionamento
 - Permitir o uso eficiente dos dois tipos de dispositivos
 - Suportar a partilha dos dispositivos de comunicação e armazenamento
-

Partilha dos dispositivos de entrada/saída

- Dispositivos partilháveis
 - Podem ser usados em concorrência por vários processos; exemplo: discos
 - Dispositivos dedicados
 - Atribuído em exclusivo a um processo por um (longo) período de tempo
 - Acesso aos dispositivos não é só uma questão de protecção, mas também é preciso gerir os acessos de acordo com as características do dispositivo
-

Acesso aos dispositivos de entrada/saída

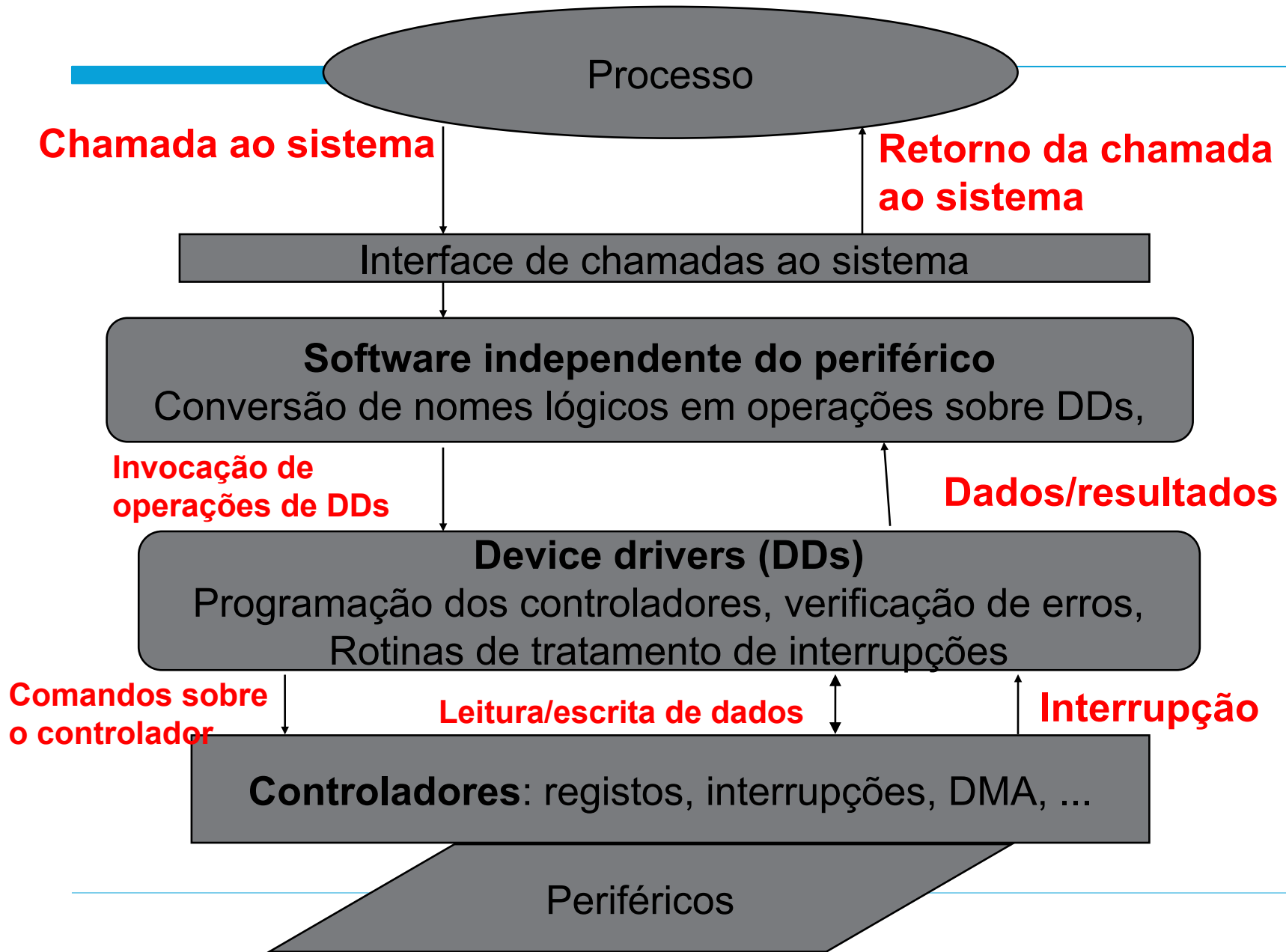
Apenas permitido ao sistema operativo

- Manipulação dos periféricos por **instruções privilegiadas**
 - Código de acesso direto aos periféricos concentrado nos **device drivers** (*supervisores de periféricos*)
 - Utilizadores fazem acesso indireto através de **chamadas ao sistema**
-

Organização do sistema na parte relacionada com E/S

- Dispositivos de E/S integrados no sistema de ficheiros:
 - Designação (“/dev/ttys0”, “COM1:”, ...)
 - Chamadas ao sistema com modelo semelhante ao dos ficheiros (open/read/write/close), com algumas extensões para acomodar certas classes de periféírcos
-

Organização do software de E/S



Conversão de nome lógico em índices na tabela de periféricos (UNIX)

- Um periférico é um ficheiro especial (special file): *character device* ou *block device*
 - A cada periférico corresponde uma entrada numa directoria com um nome e informação associado
 - Os periféricos têm nomes tipo “/dev/XXX”
 - A informação associada ao “ficheiro” é uma tabela que permite o acesso às operações do “device driver”
-

E/S no LINUX

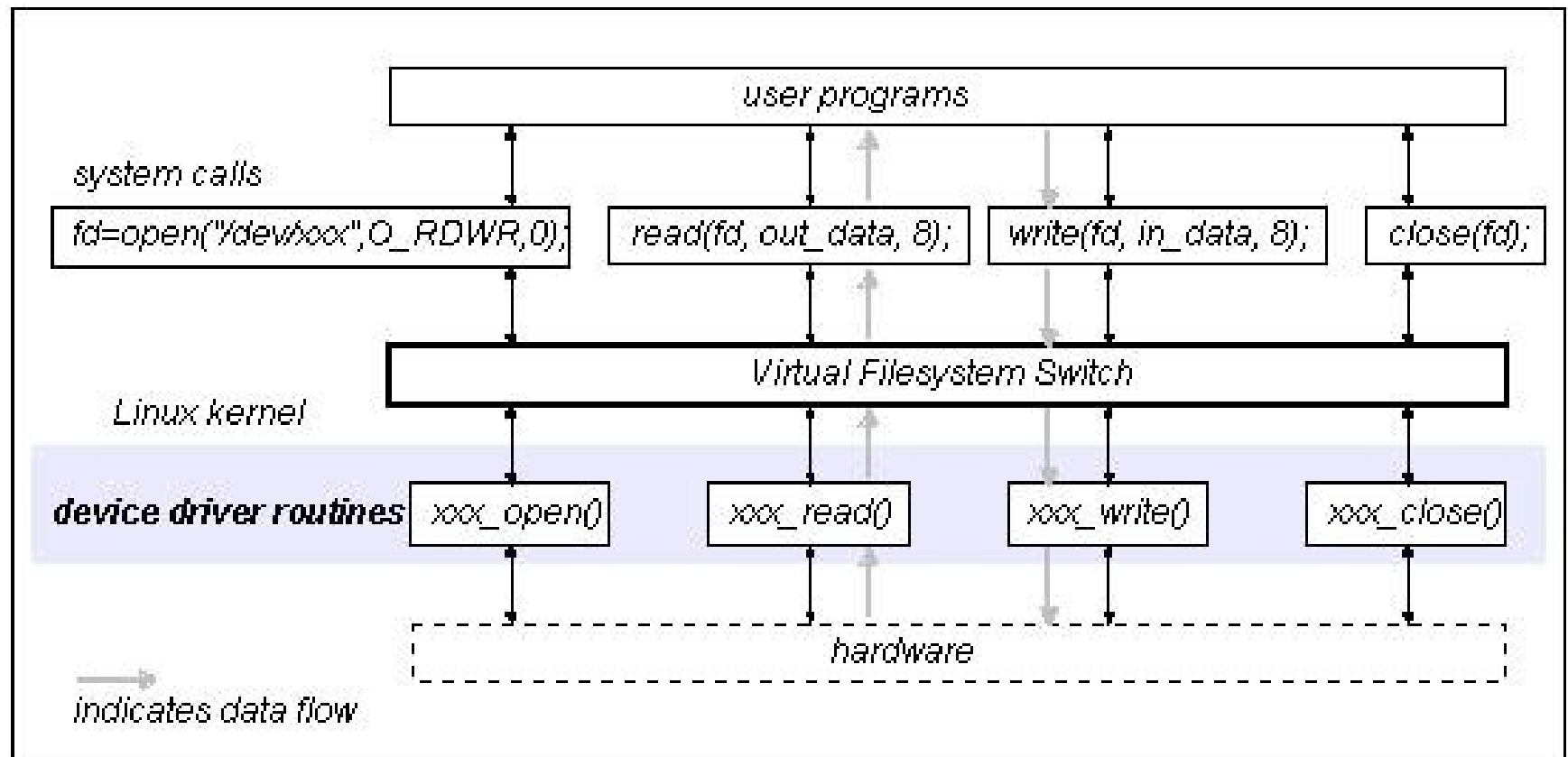
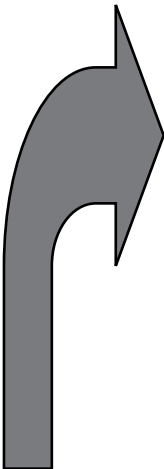


Tabela de operações



Device	Open	Close	Read	Write	ioctl	Other
Null	null	null	null	null	null	...
Memory	null	null	mem_read	mem_write	null	...
Keyboard	k_open	k_close	k_read	error	k_ioctl	...
Tty	tty_open	tty_close	tty_read	tty_write	tty_ioctl	...
Printer	lp_open	lp_close	error	lp_write	lp_ioctl	...

Quando se faz acesso a um periférico, a tabela de canais abertos identifica qual o periférico .

Essa identificação permite invocar as funções do device driver que fazem a transferência efetiva de dados entre a RAM e o periférico

Princípios gerais de E/S no LINUX

- O Linux divide os periféricos em duas classes:
 - *Periféricos de bloco* (*block devices*) permitem acesso directo a blocos de dados de tamanho fixo completamente independentes
 - O acesso a estes blocos usa uma cache
 - Normalmente usados para armazenar um sistema de ficheiros
 - *Periféricos de byte* (*character devices*) incluem quase todos os periféricos restantes: não podem suportar um sistema de ficheiros
 - Há dois tipos de periféricos que não encaixam
 - *Periféricos de rede* (*network devices*)
 - *Graphical User Interface* (*teclado + rato + ecrã bit mapped*)
-

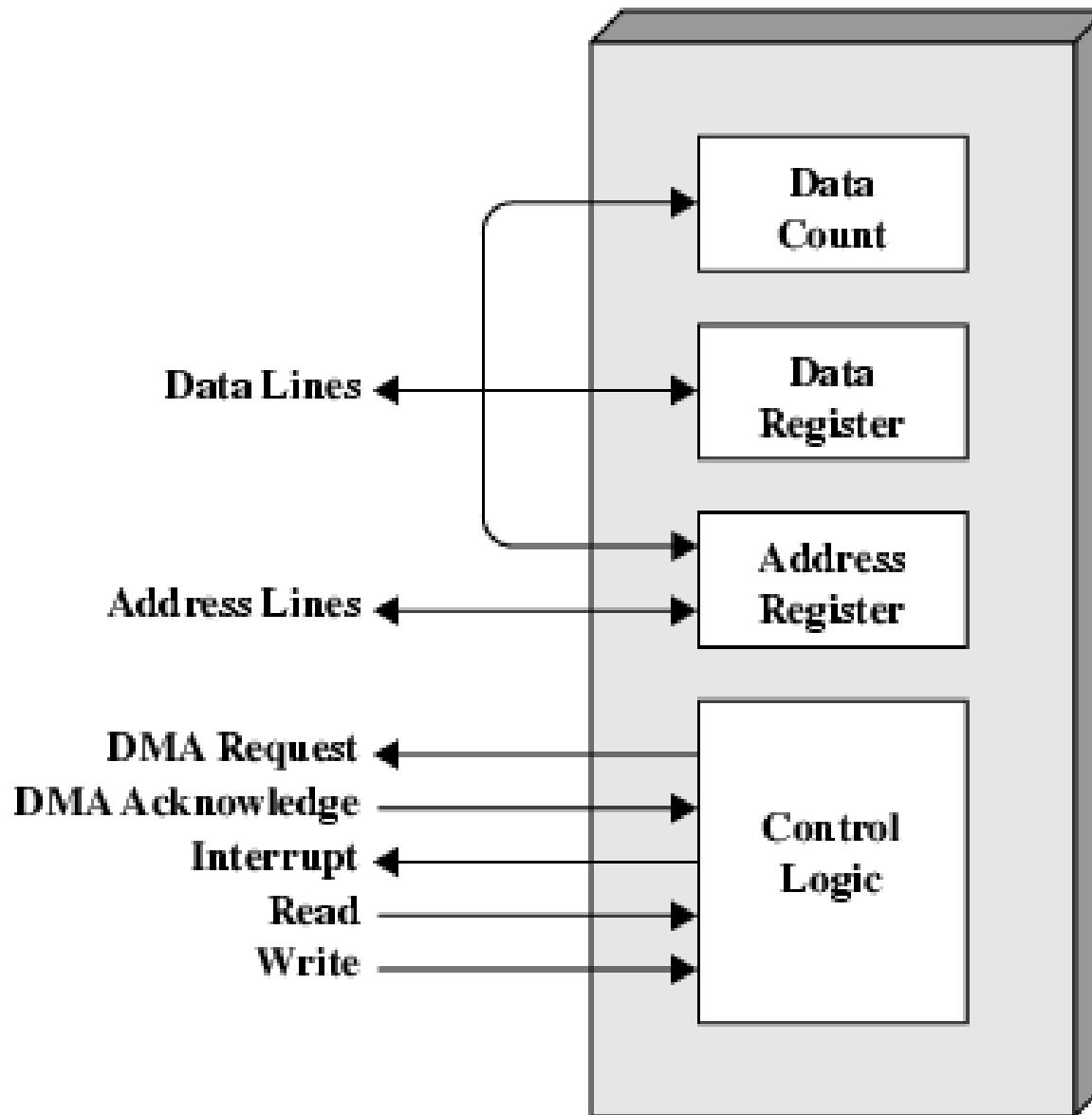
Dispositivos orientados para a transferência de blocos

- Dispositivos tipo bloco (ex: discos)
 - Comandos: ler_bloco, escrever_bloco
 - “Raw I/O” – acesso directo aos blocos – normalmente só para programas com privilégios especiais
 - Através do sistema de ficheiros
- Controladores de disco
 - Comandos: ler_bloco, escrever_bloco
 - Têm autonomia
 - Transferem o conteúdo dos blocos directamente para a RAM (DMA Direct Memory Access)
 - Assinalam o fim da operação com o envio de uma interrupção

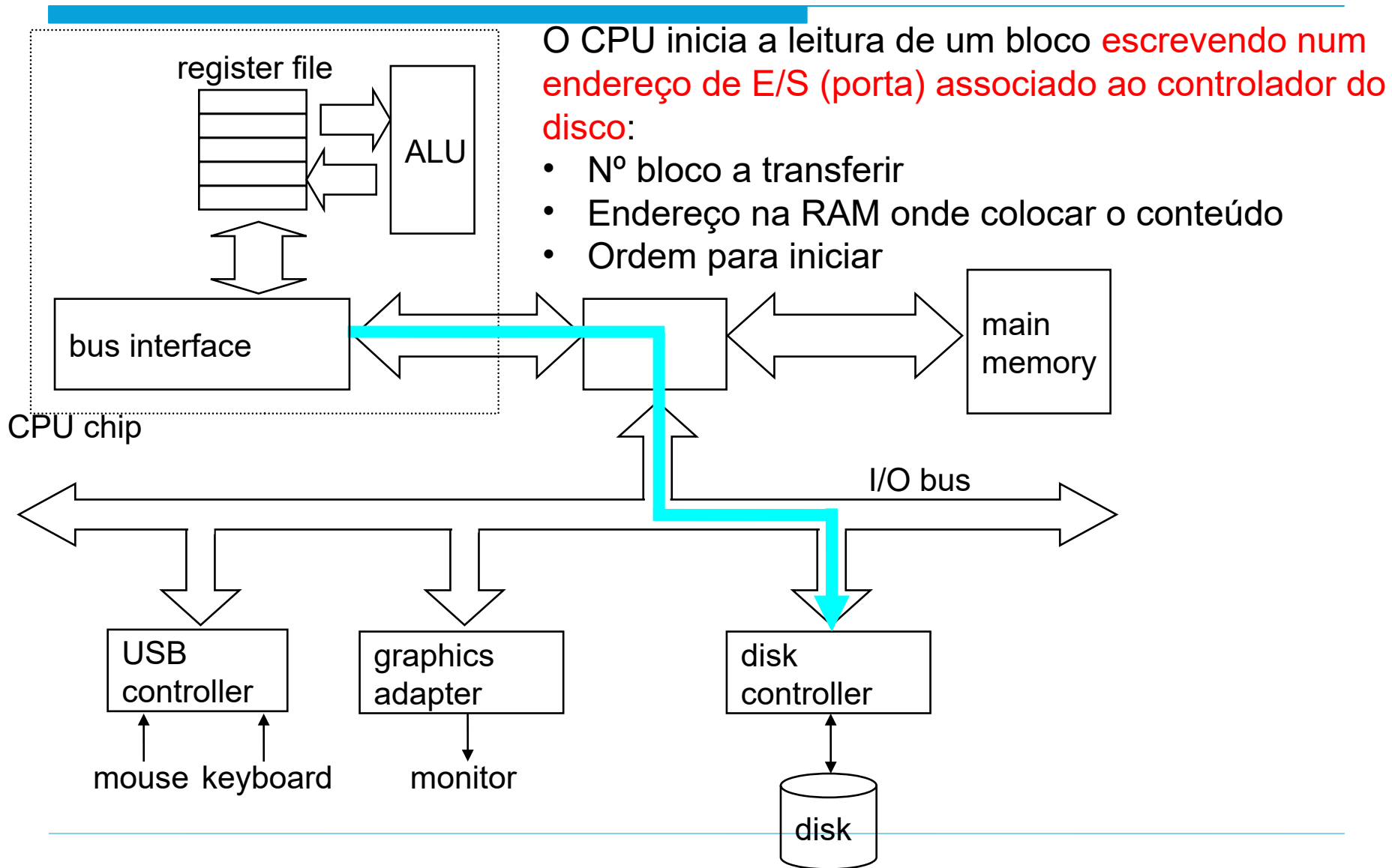
Operação do DMA

- O CPU informa o controlador de DMA sobre a transferência:
 - Leitura ou escrita
 - Endereço do dispositivo
 - Endereço do bloco de memória central onde estão / para onde vão os dados a transferir
 - Número de bytes a transferir
 - O CPU continua com o processamento
 - O controlador de DMA trata da transferência
 - O controlador de DMA envia uma interrupção quando termina
-

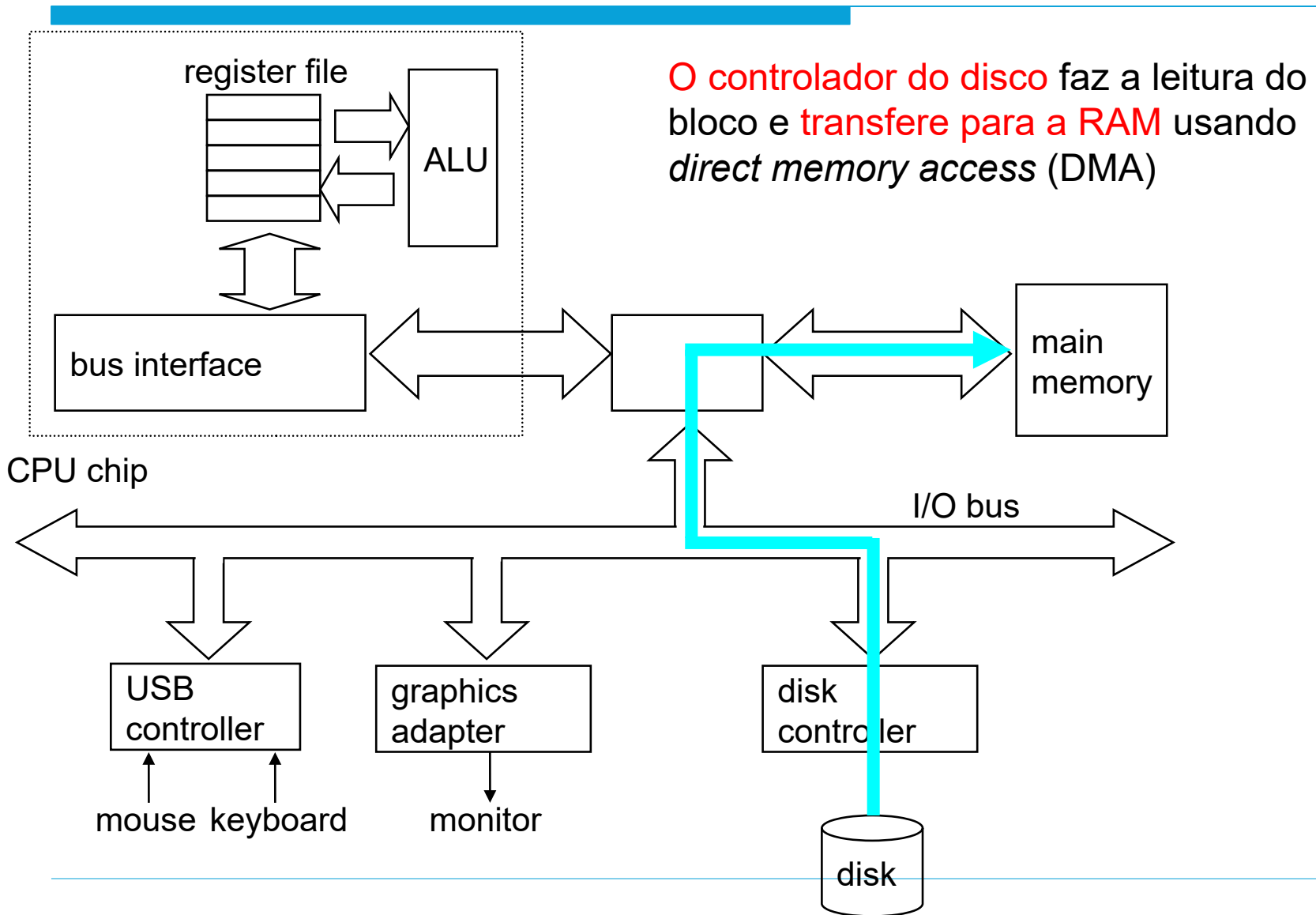
Módulo para realização de DMA



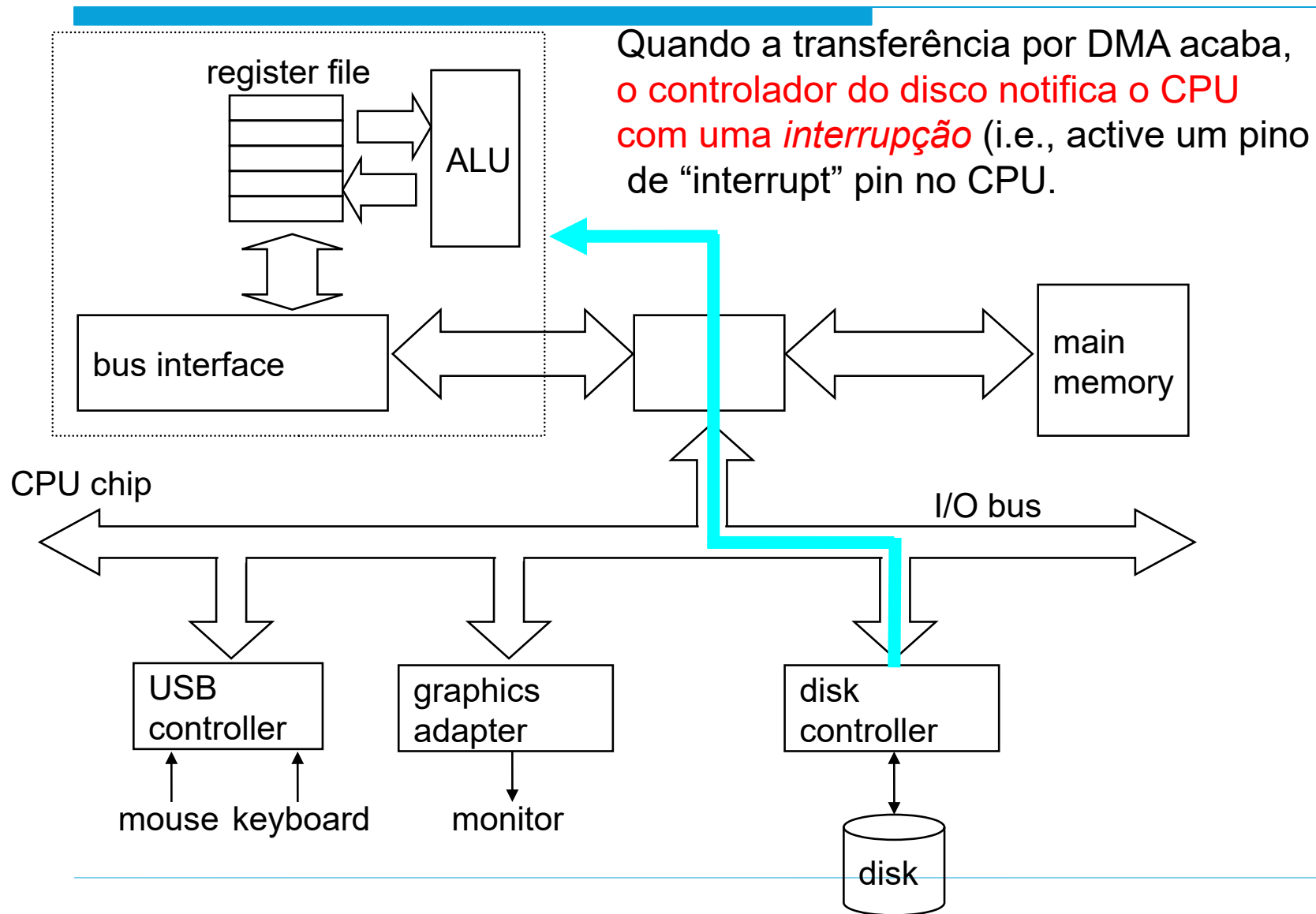
Leitura de um bloco do disco: Passo 1



Leitura de um bloco do disco: Passo 2



Leitura de um bloco do disco: Passo 3

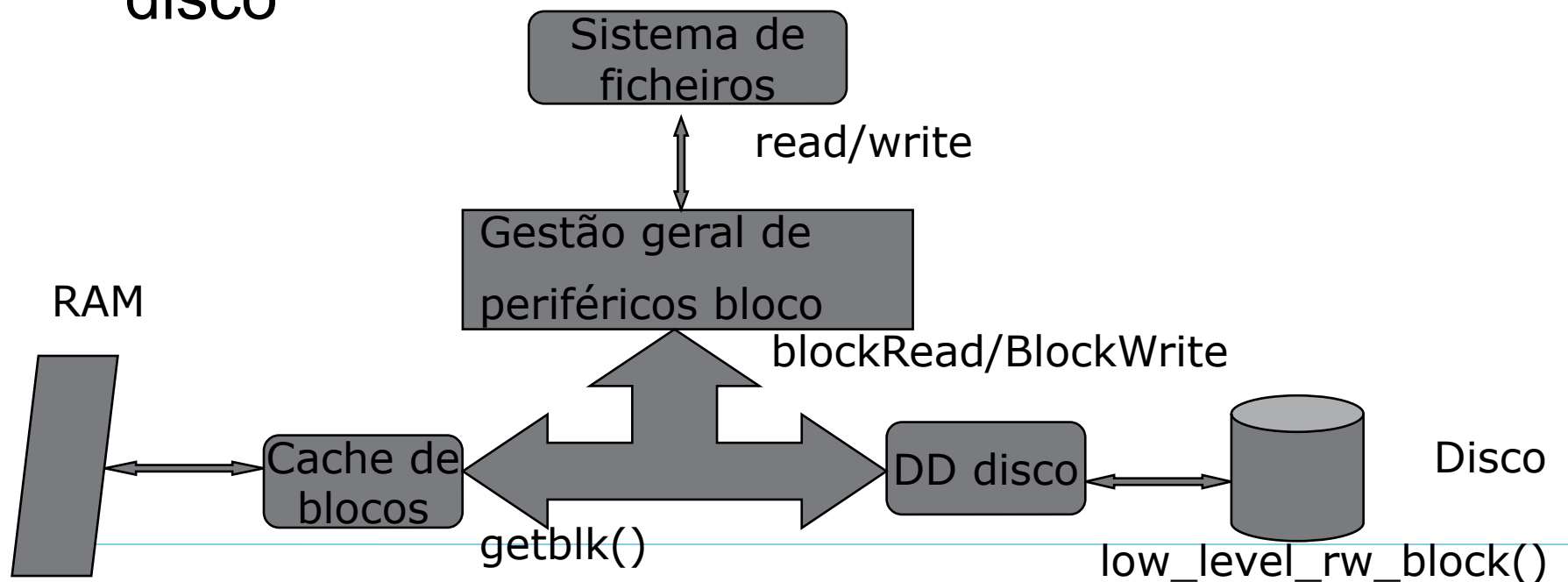


Cache de blocos

- Tipicamente discos – latência elevada, taxa de transferência elevada
- Mas há localidade nos acessos: várias leituras e escritas no mesmo bloco, acesso sequencial
- O sistema operativo usa a RAM que estiver livre para guardar blocos de disco a que se fez acesso recentemente: **Cache de blocos**
 - Em leitura, reutiliza-se a cópia em disco
 - Em escrita, escreve-se na cópia que está em RAM
 - Há diferenças entre o bloco que está em RAM e o que está em disco. Só se atualiza o disco quando se fecha o ficheiro ou explicitamente

Periféricos tipo bloco

- A RAM é usada como cache do disco
 - Se o bloco está na cache em RAM o acesso é mais rápido (hit na cache)
 - Se o bloco não está na cache faz-se acesso ao disco

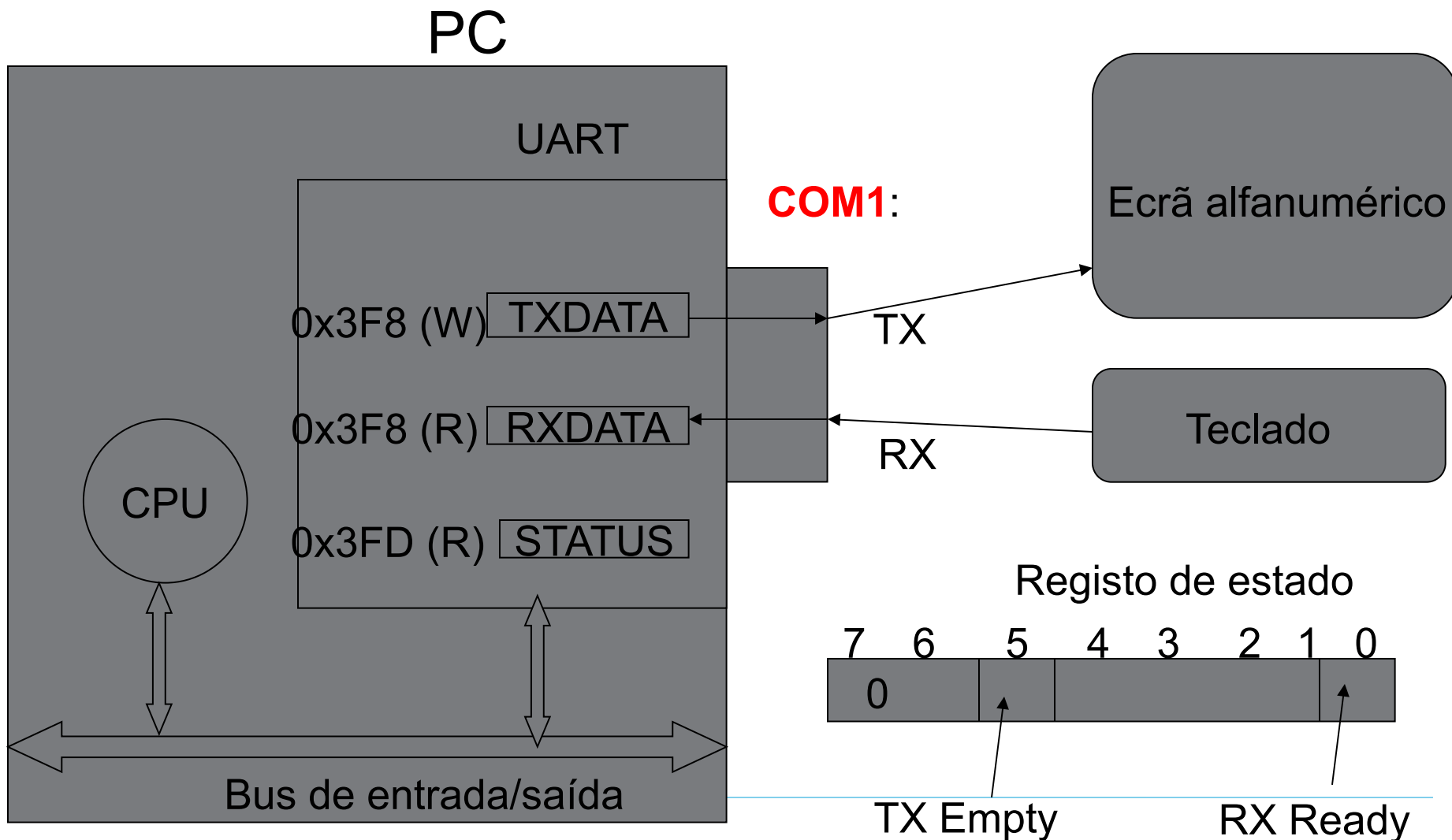


Dispositivos orientados para a transferência de bytes isolados

- Dispositivos tipo carácter (exemplos: teclados, ratos, portas série)
 - Modelo “stream” de bytes
 - Comandos: `get_char`, `put_char`

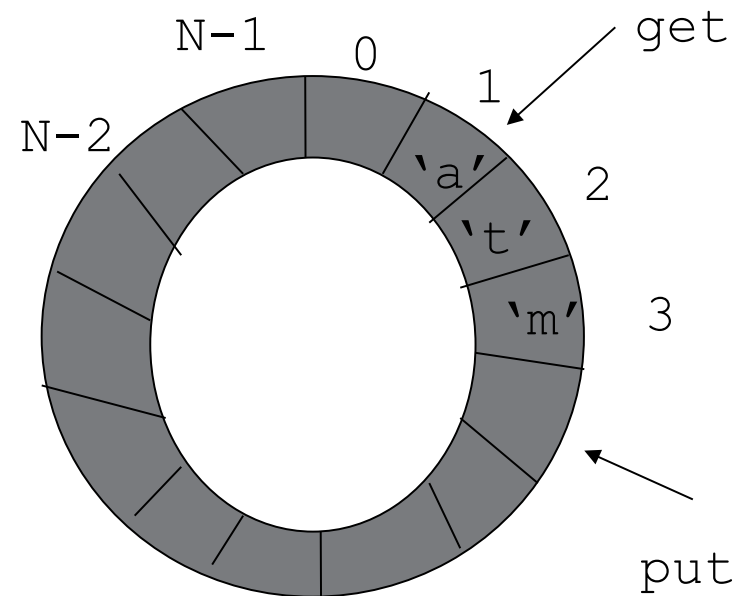
Exemplo: Porta série do PC

Terminal série



Uso de interrupções em periféricos orientados para transferência por bytes

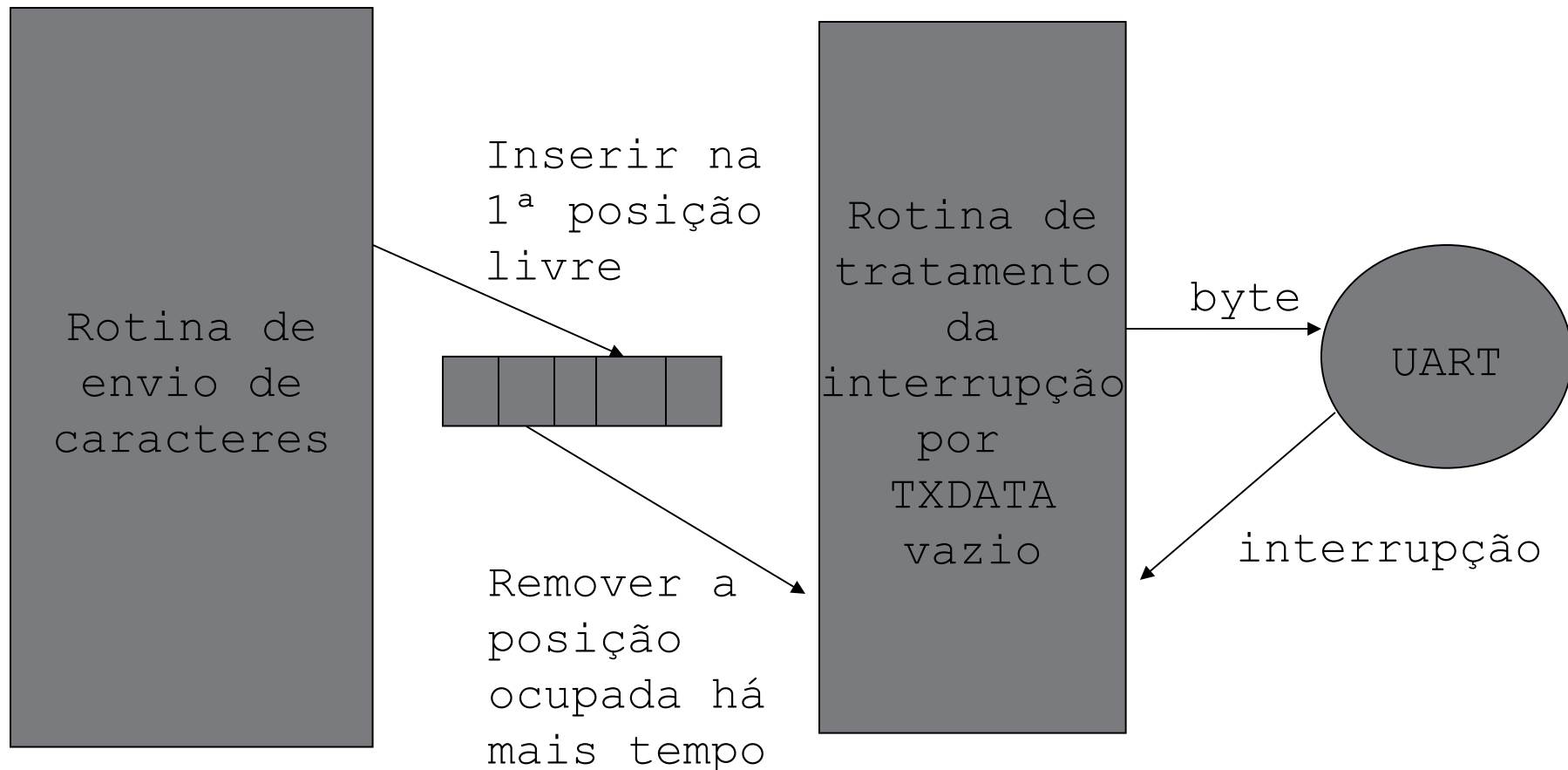
- Buffer circular permite tornar independente a velocidade a que os programas lêem e escrevem nos periféricos e o ritmo a que os periféricos transferem
- **Produtor:** coloca items no buffer. Só se bloqueia se o buffer estiver cheio
- **Consumidor:** retira items do buffer. Só se bloqueia se o buffer estiver vazio



Transmissão com interrupções

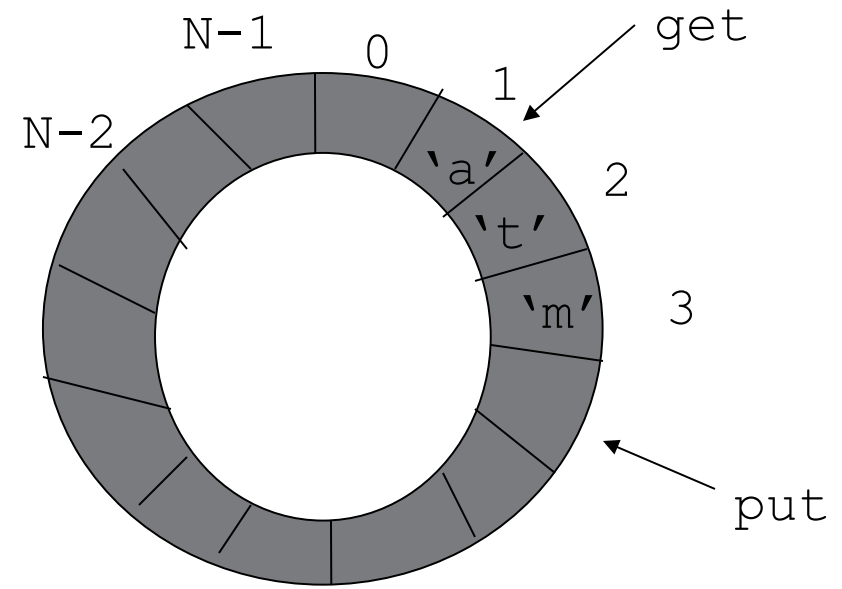
Produtor

Consumidor



Inicialização do “buffer” circular

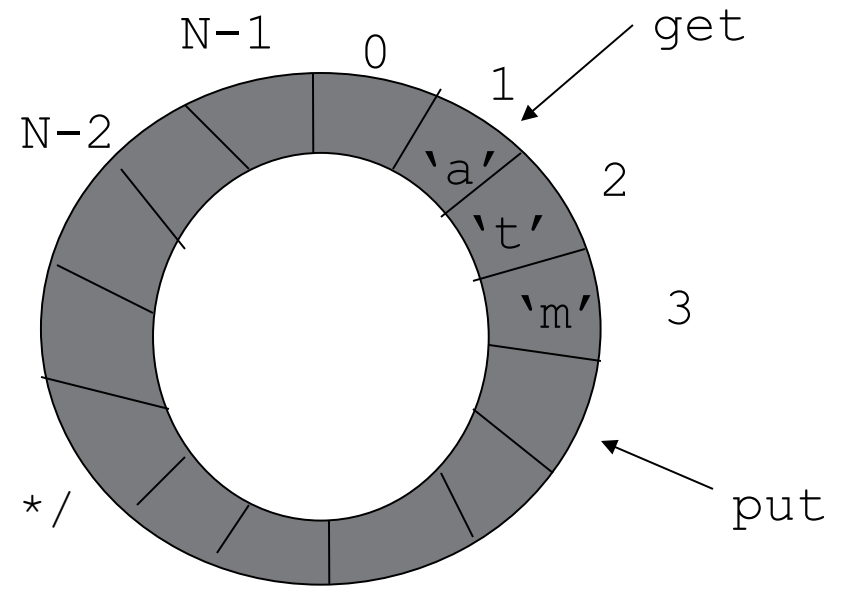
```
#define BUFSIZ 128
unsigned char buf[BUFSIZ];
int put;  // 1ª casa livre
int get;  // casa ocupada há mais
          // tempo
int nc;   // nº de bytes no buffer
put = get = nc = 0; // inicial
```



Colocar um byte no buffer circular

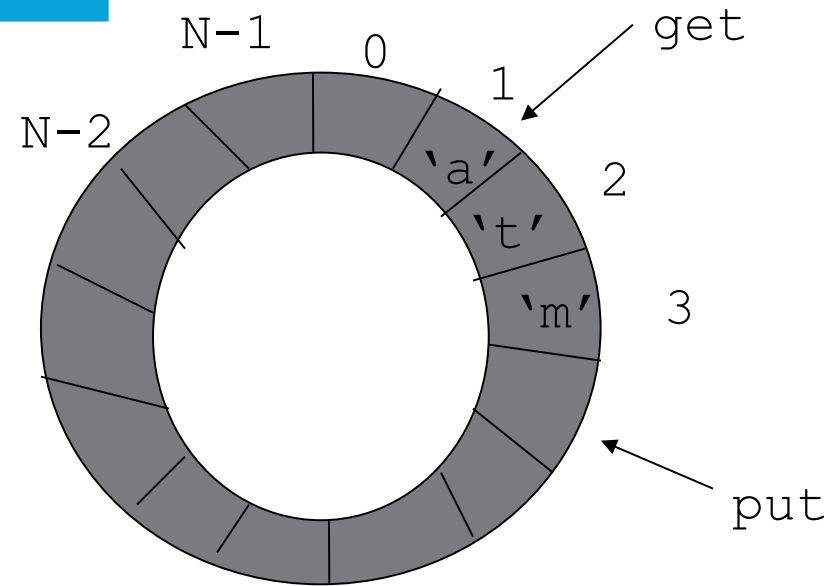
```
void bufPut( unsigned char c){  
/* assume que buf não está cheio */  
    buf[put] = c;  
    put = (put + 1) % BUFSIZ;  
    nc ++;  
}
```

```
int bufFull(){  
    return (nc == BUFSIZ);  
}
```



Obter um byte do “buffer” circular

```
unsigned char bufGet(void) {  
    /* assume que buf não está  
    vazio */  
    unsigned char x = buf[get];  
    get = (get + 1) % BUFSIZ;  
    nc --;  
    return x;  
}  
  
int bufEmpty() {  
    return (nc == 0);  
}
```



Transmissão com interrupções

```
Enviar_serie(ch) {  
    if bufFull()  
        assinala erro  
    if bufEmpty()  
        bufPut(ch)  
        ligar interrupç. TXEMPTY  
    else  
        bufPut(ch)  
}
```

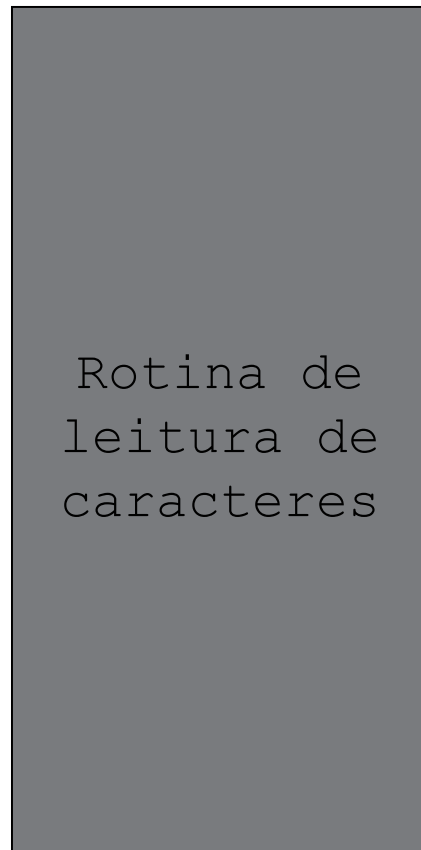
```
Rotina de tratamento de  
interrupções TX_EMPTY{  
    ch = bufGet();  
    outportb( TXDATA , ch );  
    if bufEmpty()  
        desligar as interrupções  
        TXEMPTY  
  
    return_from_interrupt  
}
```

Notas importantes:

- Quando o hardware é inicializado as interrupções por TXEMPTY estão desativadas
- A rotina de interrupções desliga as interrupções da UART por TXEMPTY se o “buffer” fica vazio
- A rotina enviar_série verifica se o “buffer” está vazio; se tal acontecer deposita o carácter no “buffer” e liga as interrupções na UART por TXEMPTY

Recepção com interrupções

Consumidor



Remover a posição ocupada há mais tempo

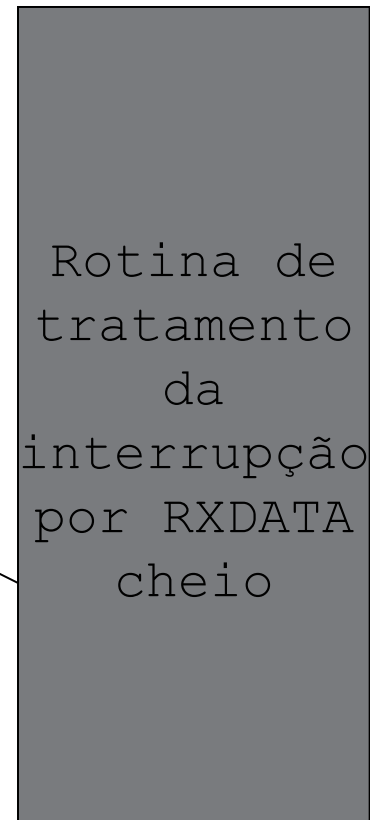
Text describing the consumer's action: removing the oldest occupied position from the buffer.



Inserir na 1ª posição livre

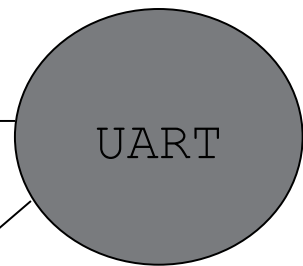
Text describing the producer's action: inserting data into the first free position of the buffer.

Produtor



byte

Text indicating the data being transferred from the UART to the producer routine.



interrupção

Text indicating the interrupt signal being sent from the UART to the producer routine.

Recepção com interrupções

```
unsigned char Receber_serie() {  
    while (bufEmpty())  
        retorna com erro  
    disable() ;  
    ch = bufGet() ;  
    enable() ;  
    return ch ;  
}  
  
Rotina de tratamento de  
interrupções RXREADY {  
    ch = inportb( RXDATA) ;  
    if bufFull()  
        # erro, não há espaço no  
        buffer mas não se pode  
        esperar  
    else bufPut(ch) ;  
  
    return_from_interrupt  
}
```

Notas importantes:

- As interrupções por RXREADY estão sempre ligadas
 - A rotina de interrupções pode encontrar o “buffer” cheio; se tal acontece não pode ficar em espera activa
 - Ver a razão da necessidade `disable()` – CLI e `enable()` – STI a seguir
-

Atualização do nº de bytes no “buffer” (1)

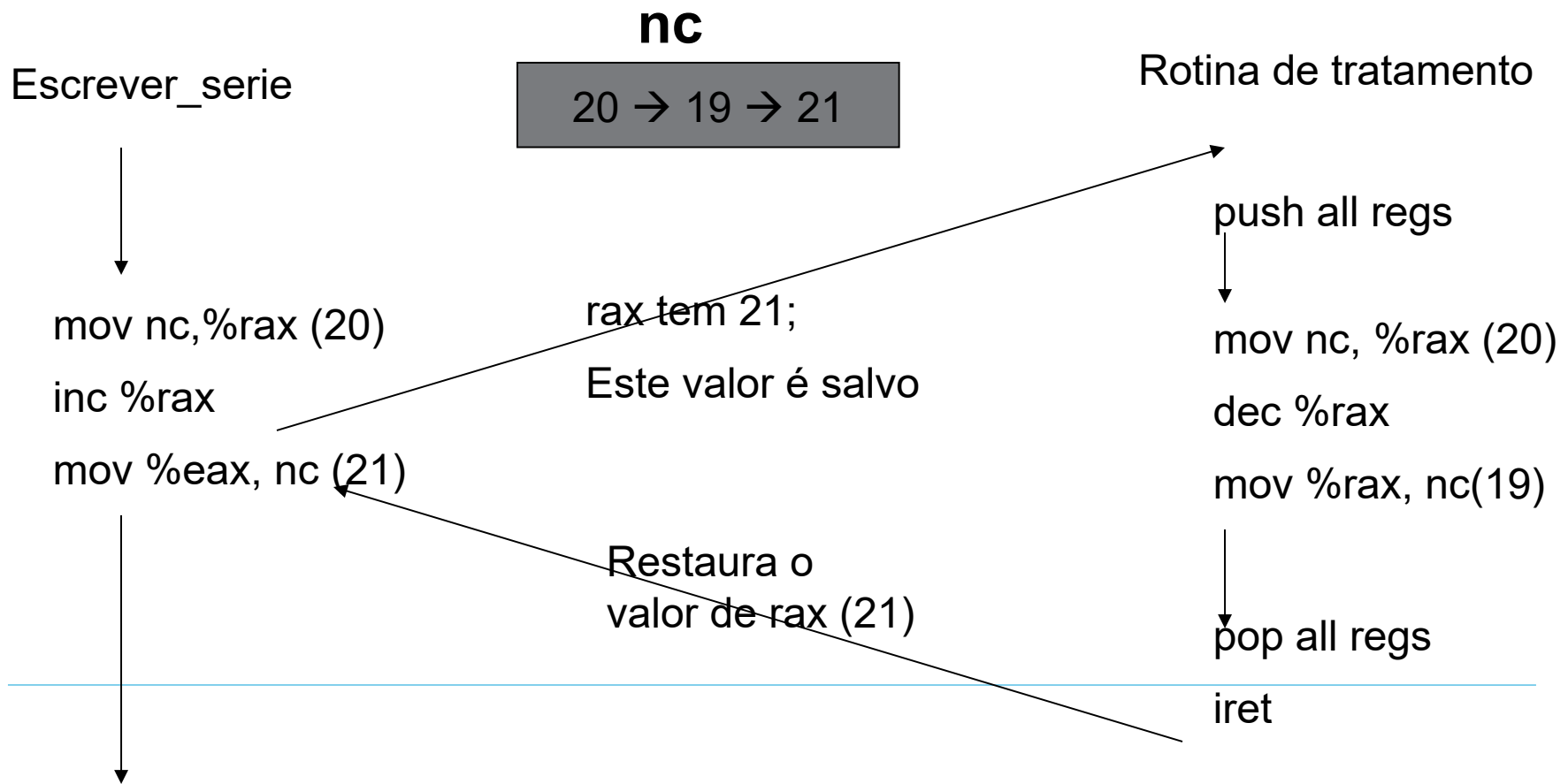
- A rotina **escrever_série** incrementa o número de bytes no buffer
 - *nc* ++
 - O compilador traduz para

```
mov nc, %rax
inc %rax
mov %rax, nc
```
 - A rotina de **atendimento de interrupções de transmissão** decrementa o número de bytes no buffer
 - *nc* - -
 - O compilador traduz para

```
mov nc, %rax
dec %rax
mov %rax, nc
```
-

Atualização do nº de bytes no “buffer” (2)

- Suponhamos que nc é 20 e que enquanto a rotina escrever série deposita um carácter, ocorre uma interrupção motivada pelo facto do registo THR ter ficado vazio



Actualização do nº de bytes no “buffer” (3)

- Após inserir um carácter e remover outro *nc* deveria ter ficado com 20
- Ficou com 21 o que é um erro que vai provocar problemas para o futuro nas chamadas de *bufEmpty()* e *bufFull()*
- Isto aconteceu porque a acção de actualização da variável foi interrompida a meio; antes de voltar à actualização foi chamada uma rotina que actualiza a mesma variável
- A variável *nc* é partilhada pela rotina *enviar_série* e pela *rotina de tratamento de interrupções*. A sua actualização não pode ser interrompida – constitui o que se chama uma secção crítica

Actualização do nº de bytes no “buffer” (4)

- Para resolver isto é preciso alterar a rotina *escrever_serie*
`disable();` // execução da instrução máquina **CLI**:
colocar a Int Flag do CPU a 0
`bufPut();`
`enable();` // execução da instrução máquina **STI**:
colocar a Int Flag do CPU a 1
 - Assim garante-se que nc é actualizado correctamente
 - A mesma alteração também tem de ser feita na função *ler_serie()*
-