Aula 11 - Princípios de Hardware e Software de E/S

Sistemas Operacionais
Ciência da Computação
IFB - Campus Taguatinga

Professor João Victor de A. Oliveira

Hoje

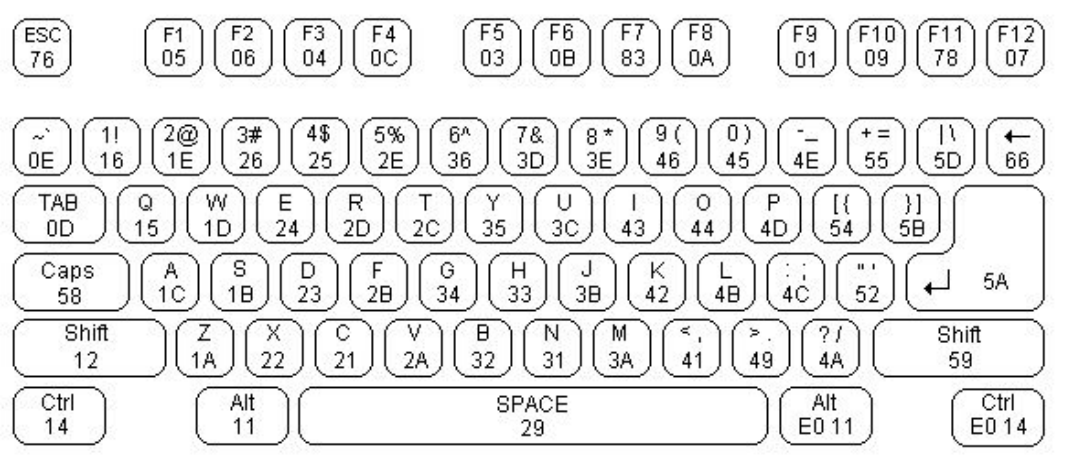
- Princípios do Hardware de E/S
  - Dispositivos de E/S
  - Controladores de Dispositivos
  - E/S mapeada na memória
  - Acesso direto à memória (DMA)
  - Interrupções

- **Princípios do Software de E/S**
  - Objetivos
  - E/S Programada
  - E/S orientada a interrupções
  - E/S usando DMA
- **Camadas do Software de E/S**
    - Tratadores de interrupção
    - Drivers dos dispositivos
    - Softwares de E/S independente de dispositivo
    - Software de E/S do espaço de usuário

Hardwares de Entrada e Saída

Entrada e Saída

- Sistema Operacional é responsável por controlar todos os dispositivos de E/S do computador
  - Deve **emitir comandos para os dispositivos**, **interceptar interrupções** e **lidar com erros**
  - Deve também fornecer uma **interface** entre os dispositivos e o resto de sistema que seja **fácil e simples de usar**

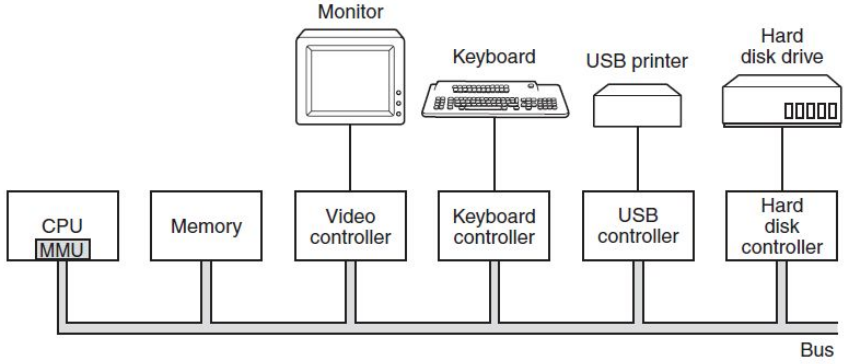


Dispositivos de E/S

- Podem ser divididos em duas categorias
  - **Dispositivos de blocos**
    - Armazena informações em blocos de tamanho fixo, cada um com seu próprio endereço
    - Transferências são em unidades de um ou mais blocos inteiros (consecutivos)
    - **Propriedade essencial:** Cada bloco pode ser lido ou escrito independentemente de todos os outros
    - **Ex.:** Discos rígidos, discos de Blu-ray e pendrives
  - **Dispositivos de caracteres**
    - Envia ou aceita um fluxo de caracteres, desconsiderando qualquer estrutura de bloco
    - Não é endereçável e não tem nenhuma operação de busca
    - **Ex.:** Impressoras, interfaces de rede, mouses, teclados...àqueles que não se parecem com discos...

Controladores de dispositivos

- Unidades de E/S consistem, em geral de um **componente mecânico** e um **componente eletrônico**
  - Componente eletrônico é chamado de **controlador do dispositivo (ou adaptador)**
    - Assume a forma de um **chip na placa mãe** ou um **cartão de circuito impresso** que pode ser inserido em um slot de expansão **(PCIe)**
  - Componente mecânico é o dispositivo em si
  - **Interface entre controlador e dispositivo** seguem padronizações oficiais
    - ANSI, IEEE ou ISO
    - Controladores de disco normalmente seguem os padrões de interface SATA, SCSI, USB, Thunderbolt ou FireWire...



Controladores de dispositivos

- Interface entre controlador e dispositivos muitas vezes é de nível muito baixo
  - Ex.: Disco pode ser formatado com 2 milhões de setores de 512 bytes por trilha
    - No entanto o que sai da unidade é um fluxo serial de bits (um por um) consistindo de:
      - **Preâmbulo** (Escrito quando o disco é formatado, contendo o cilindro e número de setor, além de dados similares, assim como informações de sincronização
      - **4096 bytes em um setor** e,
      - **Uma soma de verificação (checksum)** ou código de correção de erro (**ECC - *Error Correcting Code)***
- **Trabalho do controlador:** Converter o fluxo serial de bits em um bloco de bytes, assim como realizar qualquer correção de erros necessária
  - Bloco de bytes normalmente é montado em um *buffer* primeiro dentro do controlador (bit a bit)
  - Depois a sua soma de verificação é verificada e o bloco é (ou não) declarado livre de erros
  - Caso esteja livre de erros ele pode, então, ser copiada para a memória principal

Controladores de dispositivos

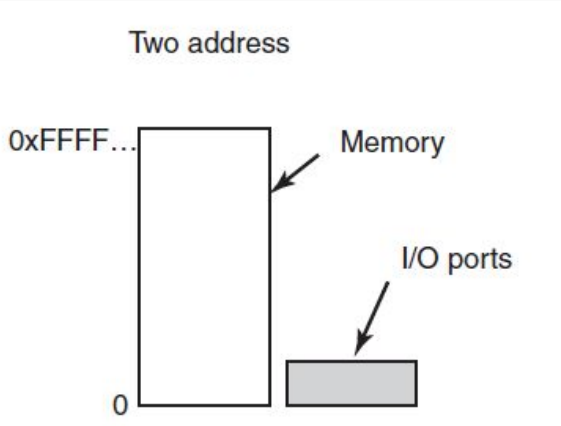
- Outro Exemplo: Controlador para um monitor LCD
  - Lê os bytes contendo os caracteres a serem exibidos da memória
  - Gera, então, os sinais para modificar a polarização da retroiluminação para os pixels correspondentes a fim de escrevê-los na tela
  - **Se não fosse pelo controlador de tela, o programador do SO teria de programar explicitamente os campos elétricos de todos os pixels!!!**

Comunicação com Dispositivos de Entrada e Saída

- Controladores tem alguns **registradores** que são usados para comunicar-se com a CPU
  - **Ao escrever nesses registradores**, o SO pode comandar o dispositivo a fornecer e aceitar dados, ligar-se e desligar-se, ou de outra maneira, realizar alguma ação
  - **Ao ler a partir desses registradores,** o SO pode descobrir qual é o estado do dispositivo, se ele está ou não preparado para aceitar um novo comando e assim por diante
- Além dos registradores de controle, muitos dispositivos têm um *buffer* de dados
  - Ex.: RAM de vídeo
    - Buffer de dados disponível para ser escrita pelos programas ou sistema operacional

Comunicação com Dispositivos de Entrada e Saída

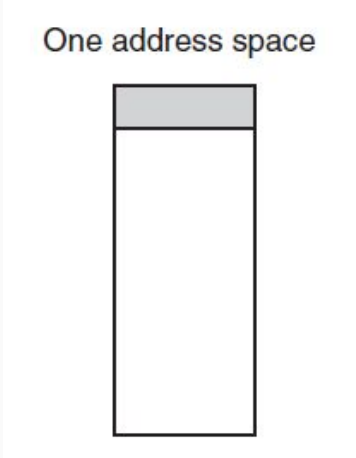
- **Como a CPU se comunica com os registradores de controle e também com os buffers de dados do dispositivo?**
  - **Duas alternativas**
    1. Para cada registrador de controle é designado um **número de porta de E/S,** um inteiro de 8 ou 16 bits...
       - Conjunto de todas as portas de E/S formam o **espaço de E/S,** protegido de maneira que programas de usuários comuns não consigam acessá-lo
       - Ex. de instruções de E/S especial
         - **IN** REG, PORT
         - **OUT** PORT, REG
         - IN R1,4 **é diferente de** LD R1,#4? **por quê?**
       - Maioria dos primeiros computadores funcionava dessa maneira



E/S mapeada na memória

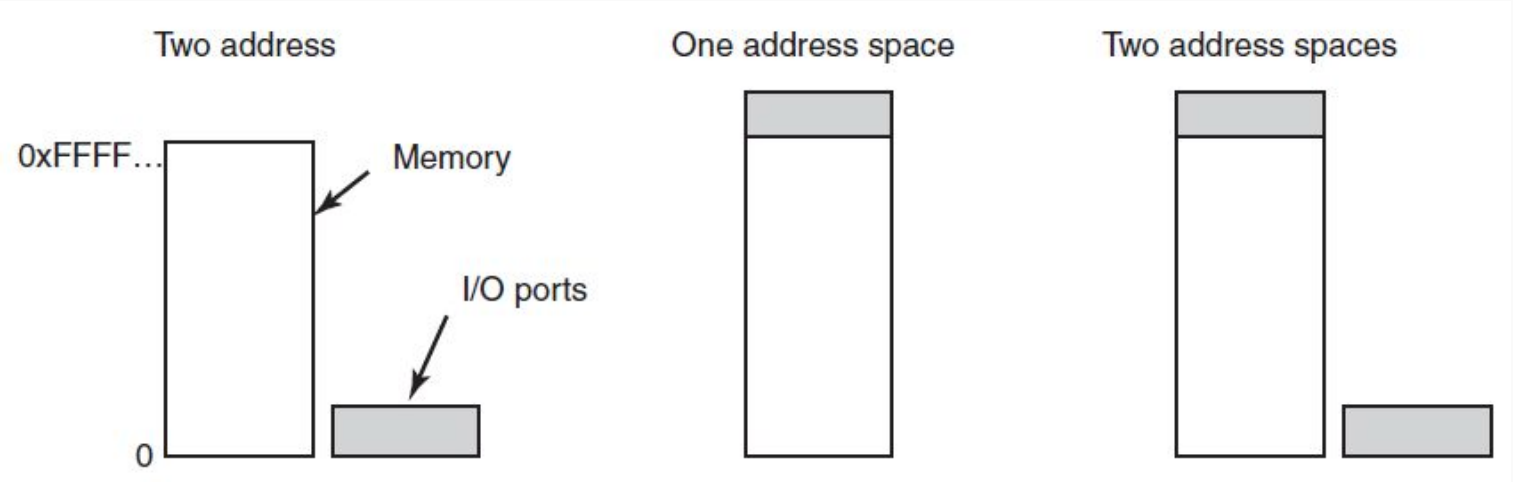
2ª Alternativa: Mapear todos os registradores de controle no espaço da memória

- Para cada registrador de controle é designado um e**ndereço de memória único,** para o qual nenhuma memória é designada
- Esse sistema é chamado de **E/S mapeada na memória**



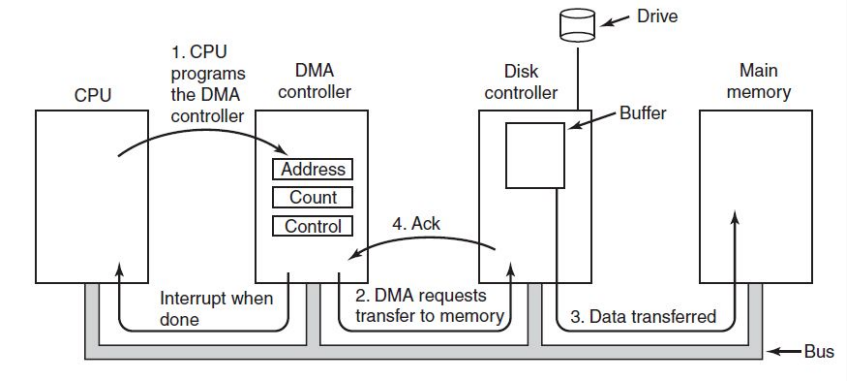
E/S mapeada na memória

- **Terceira abordagem:** esquema híbrido, com buffers de dados de E/S mapeados na memória e portas de E/S separadas para os registradores de controle
  - x86 usa essa abordagem, com endereços de 640K a 1M - 1, portas de E/S de 0 a 64K -1

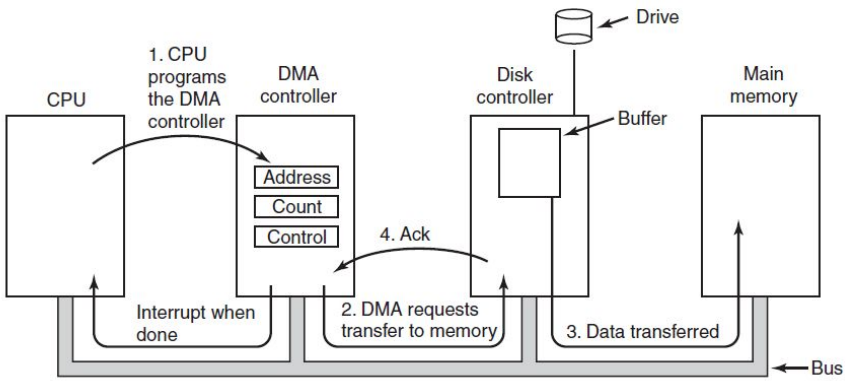


Acesso direto à memória (DMA)

- **Não importa se uma CPU tem ou não E/S mapeada na memória**
  - Ela precisa endereçar os controladores dos dispositivos para poder trocar dados com eles
- CPU pode requisitar dados de um controlador de E/S um byte de cada vez
  - Fazê-lo desperdiça o tempo da CPU, de maneira que um esquema diferente chamado **Acesso Direto à Memória (*Direct Memory Access - DMA*)** é usado muitas vezes
- ***Controlador DMA***
  - Imagine que a CPU acessa todos os dispositivos e memória mediante um único barramento no sistema

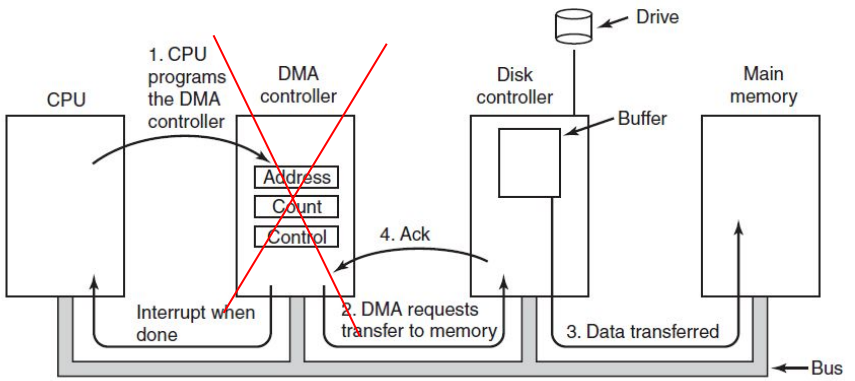


Acesso direto à memória (DMA)



- ***Controlador DMA***
  - Tem acesso ao barramento do sistema independente da CPU
  - Contém vários registradores que podem ser lidos e escritos pela CPU
    - Inclui um registrador de endereço de memória, registrador de contador de bytes e um ou mais **registradores de controle**
    - Registradores de controle especificam
      - Uma porta de E/S a ser usada
      - Direção da transferência, a quantidade de bytes ou palavras por vez....
  - Para entendermos como o DMA funciona, vejamos como ocorre a leitura de disco quando o DMA não é usado

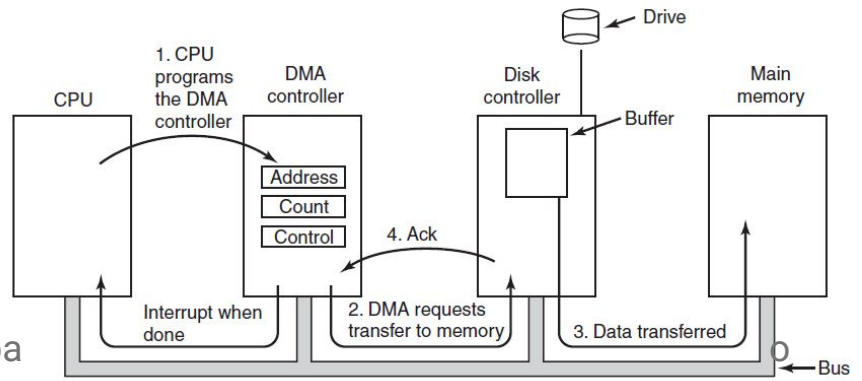
Acesso direto à memória (DMA)



- ***Leitura de disco sem DMA***

1. Controlador de disco lê o bloco do dispositivo serialmente, bit por bit, até encher o bloco no seu buffer interno
2. Em seguida, calcula a soma de verificação para conferir se nenhum erro de leitura tenha ocorrido
3. Após isso, o controlador causa uma interrupção
    - Quando o SO começa a ser executado, ele pode ler o bloco de disco do buffer iterativamente (bit a bit ou palavra por palavra) e armazená-lo na memória principal

Acesso direto à memória (DMA)



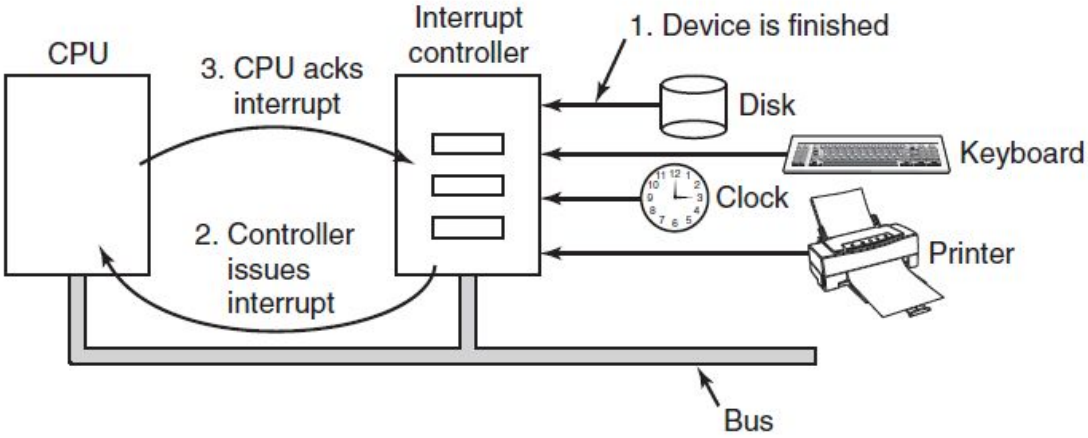
- ***Leitura de disco com DMA***
  1. CPU programa o controlador de DMA configurando seus registradores para que ele saiba que transferir e para onde
     - Também emite um comando ao controlador de disco dizendo para ele ler os dados do disco para o seu buffer interno e verificar a soma de verificação
  2. Controlador DMA inicia a transferência emitindo uma solicitação de leitura via barramento para o controlador de disco
  3. Escrita na memória principal dos blocos requisitados
  4. Quando a escrita é completada, o controlador de disco envia um sinal de confirmação para o controlador de DMA, também via barramento
  5. Se o contador de bytes for igual a zero, o controlador de DMA interrompe a CPU para deixá-lo ciente de que a transferência está completa agora.

Acesso direto à memória (DMA)

- **Barramentos operam em dois modos**
  - **Uma palavra de cada vez (*word at-a-time mode*)**
    - Controlador DMA solicita a transferência de de uma palavra e a consegue
    - Se a CPU também quiser o barramento, ela tem de esperar
    - Esse mecanismo é chamado de **roubo de ciclo**, pois o controlador de dispositivo rouba um ciclo de barramento ocasional da CPU, de vez em quando, atrasando-a ligeiramente
  - **Modo bloco**
    - Controlador DMA diz para o dispositivo adquirir o barramento, emitir uma série de transferências e então libera o barramento
    - Essa operação é chamada de **modo de surto (*burst*)**
    - É mais eficiente, pois adquirir o barramento leva tempo e múltiplas palavras podem ser transferidas pelo preço de uma aquisição de barramento
    - **Desvantagem**: pode bloquear a CPU e outros dispositivos por um período mais longo

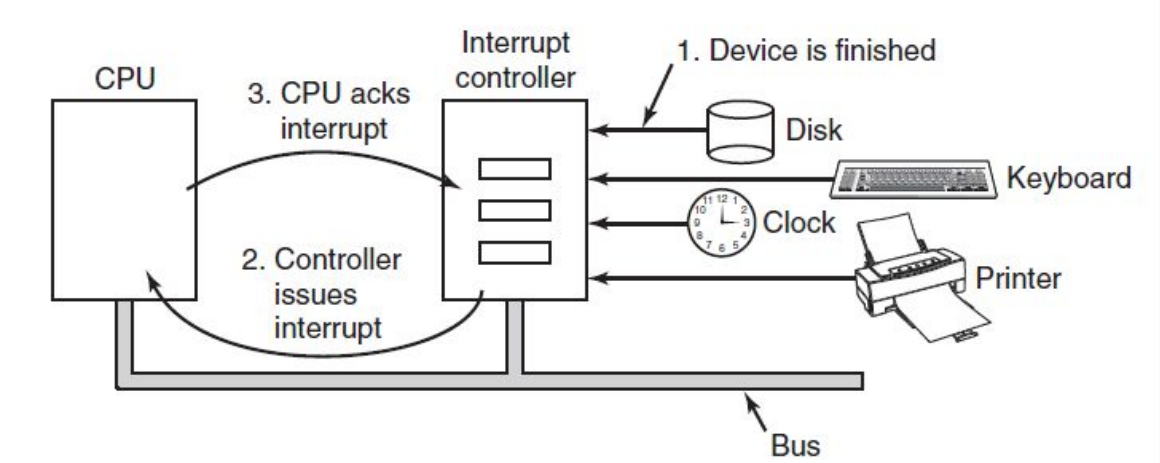
Interrupções

1. Quando um dispositivo de E/S termina o trabalho ele gera uma **interrupção**
   - Faz isso enviando um sinal pela linha de barramento à qual está associado
   - Esse sinal é detectado pelo chip controlador de interrupções da placa mãe, que então decide o que fazer
   - Caso não haja interrupções pendentes, o controlador de interrupção processa a interrupção imediatamente



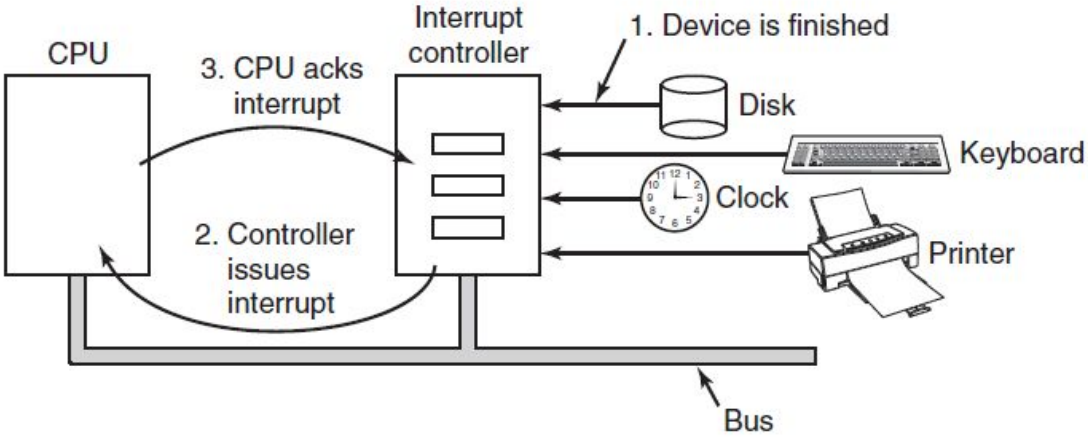
Interrupções

2. Controlador coloca um número sobre as linhas de endereço especificando qual dispositivo requer atenção e repassa um sinal para interromper a CPU
    - O número nas linhas de endereço é usado como um índice em uma tabela chamada **vetor de interrupções** para buscar um **novo contador de programa**
    - Este contador de programa aponta para o início da rotina de tratamento de interrupção correspondente



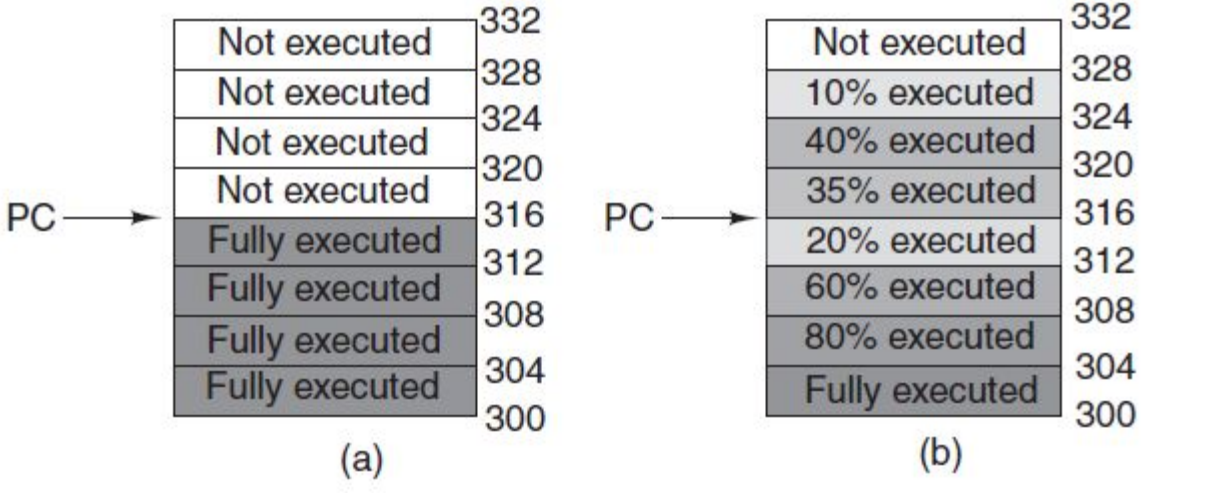
Interrupções

3. Após o início da execução, a rotina de tratamento de execução **reconhece** a interrupção escrevendo um determinado valor para uma das portas de E/S do controlador de de interrupção
    - Esse reconhecimento diz ao controlador que ele está livre para gerar outra interrupção



Interrupções precisas x imprecisas (ideia)

- **Interrupção precisa**
  1. Contador do programa é salvo em um lugar conhecido
  2. Todas as instruções anteriores àquela apontada pelo PC foram completadas
  3. Nenhuma instrução posterior à apontada pelo PC foi concluída
  4. O estado de execução da instrução apontada pelo PC é conhecido
- Problema: pipelines e superescalares (paralelismo interno)



Softwares de Entrada e Saída

Objetivos do software de E/S

- **Independência de dispositivo**
  - Devemos ser capazes de escrever programas que podem acessar qualquer dispositivo de E/S sem ter de especificá-lo antecipadamente
  - Ex.:
    - Um programa que lê um arquivo como entrada deve ser capaz de ler um arquivo em um disco rígido, um DVD ou um pen-drive sem ter de ser modificado
    - Deve ser possível digitar o comando **sort < input > output** que trabalhe com uma entrada vinda de qualquer tipo de disco ou teclado e a saída indo para qualquer tipo de disco ou tela
  - Fica a cargo do SO cuidar dos problemas causados pelo fato desses dispositivos serem diferentes e exigirem sequências de comandos muito diferentes para ler ou escrever

Objetivos do software de E/S

- **Nomeação uniforme**
  - O nome de um arquivo ou um dispositivo deve simplesmente ser uma cadeia de caracteres ou um número inteiro e não depender do dispositivo de maneira alguma
  - No UNIX, todos os discos podem estar integrados na hierarquia do sistema de arquivos de maneira arbitrária
    - Usuário não precisa estar ciente de qual nome corresponde a qual dispositivo
    - Ex.: Pen drive montado em cima do diretório */usr/ast/backup*
- **Tratamento de Erros**
  - Erros devem ser tratados o mais próximo possível do hardware
  - Se o controlador descobre um erro de leitura, ele deve tentar corrigi-lo se puder
    - Se não puder, então o driver do dispositivo deverá lidar com ele, talvez simplesmente tentando ler o bloco novamente

Objetivos do software de E/S

- **Transferências síncronas (bloqueantes) vs assíncronas (orientadas à interrupção)**
  - Maioria das E/S físicas são assíncronas - A CPU inicializa a transferência e vai fazer outra coisa até a chegada da interrupção
  - Após uma chamada de sistema ***read***, o programa de usuário é automaticamente suspenso até que os dados estejam disponíveis no buffer
    - Fica a cargo do SO fazer operações que são realmente orientadas à interrupção parecerem bloqueantes para os programas de usuário
    - Entretanto, algumas aplicações de alto desempenho precisam controlar todos os detalhes da E/S, então alguns SOs disponibilizam a E/S síncrona para si

Objetivos do software de E/S

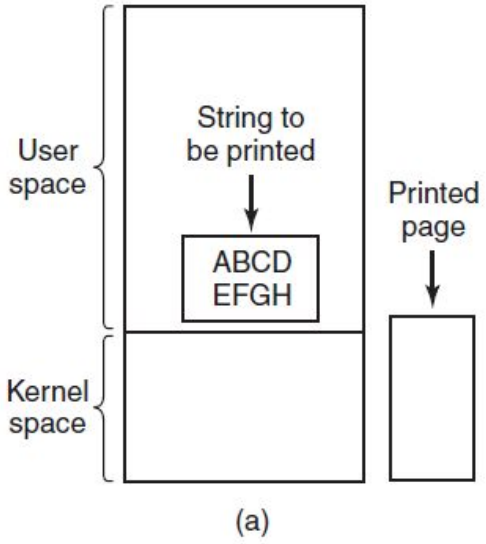
- **Utilização de Buffer**
  - Dado provenientes de um disco, em alguns casos, não podem ser armazenados diretamente em seu destino final
  - Ex.: pacote que chega da Internet
    - SO não sabe onde armazená-lo até que o tenha colocado em algum lugar para examiná-lo
  - Alguns dispositivos têm severas restrições de tempo real
    - Ex.: áudio digital
    - Dados devem ser colocados antecipadamente em um buffer de saída para separar a taxa na qual o buffer é preenchido da taxa na qual ele é esvaziado a fim de evitar seu completo esvaziamento

Objetivos do software de E/S

- **Dispositivos compartilhados vs dedicados**
  - Alguns dispositivos, como os discos, podem ser usados por muitos usuários ao mesmo tempo
  - Outros, como as impressoras, têm de ser dedicados a um único usuário até ele ter concluído sua operação
  - Dispositivos dedicados introduzem uma série de problemas em SOs, tais como os **impasses**
  - **SO deve ser capaz de lidar com ambos tipos de dispositivos de maneira que evite problemas**

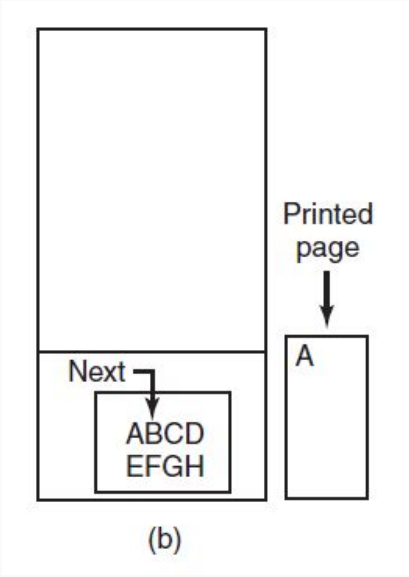
E/S programada

- **3 maneiras diferentes de realizar E/S**
  - **E/S programada**
  - **E/S orientada à interrupções**
  - **E/S usando DMA**
- Forma mais simples de E/S é ter a CPU realizando todo o trabalho
  - Esse método é chamado de **E/S programada**
- **E/S programada**
  - Considere um processo de usuário que quer imprimir a cadeia de 8 caracteres “ABCDEFGH” na impressora por meio de uma interface serial

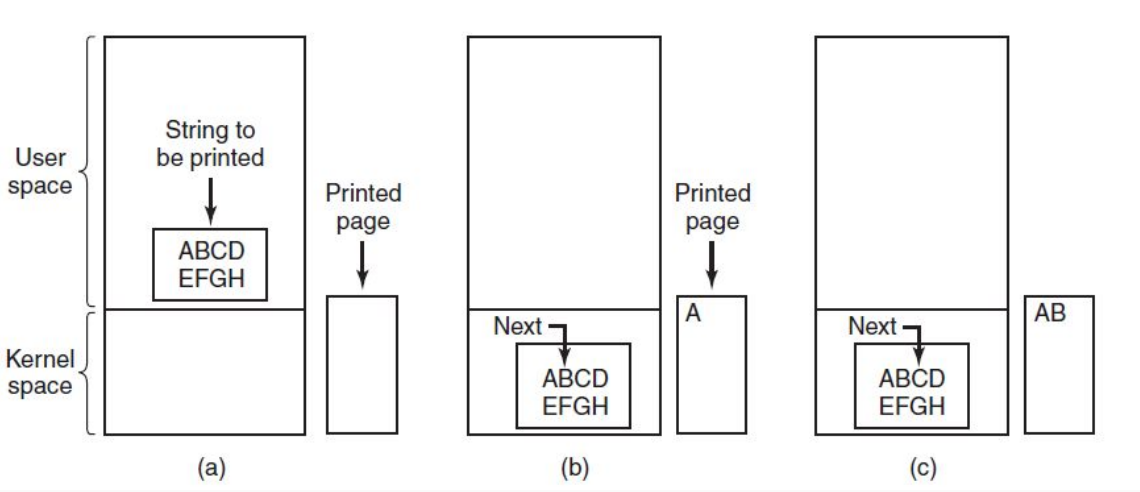


E/S programada

- **Tarefa: Imprimir a cadeia de 8 caracteres “ABCDEFGH” na impressora por meio de uma interface serial**
  1. Software primeiro monta a cadeia de caracteres em um buffer no espaço de usuário
  2. Processo do usuário requisita, então, a impressora para escrita fazendo uma chamada de sistema para abrí-la
  3. Se a impressora estiver em uso, a chamada fracassará e retornará um erro ou bloqueio até que a impressora esteja disponível
     - Caso contrário, o SO copia o buffer para um vetor no espaço do núcleo
  4. Após isso, o SO confere para ver se a impressora está disponível
  5. Tão logo esteja disponível, o SO copia o primeiro caractere para o registrador impressora, nesse exemplo usando **a E/S mapeada na memória**
     - Essa ação ativa a impressora que imprime “A” e o SO indica que “B” é o próximo elemento a ser impresso



E/S programada



Espera ocupada

```
copy_from_user(buffer, p, count);                /* p is the kernel buffer */
for (i = 0; i < count; i++) {                    /* loop on every character */
        while (*printer_status_reg != READY) ;   /* loop until ready */
        *printer_data_register = p[i];           /* output one character */
}
return_to_user( );
```

E/S programada

- Desvantagens
  - Segura a CPU o tempo todo até que toda a E/S tenha sido feita (espera ocupada)
  - Se o tempo para imprimir um caractere for muito curto, então tudo bem...
    - Em sistemas mais complexos, que a CPU tem outros trabalhos a fazer, a espera ocupada é ineficiente e é necessário um método de E/S melhor

E/S orientada a interrupções

- Precisamos de um modo de permitir que a CPU faça outra coisa enquanto espera que a impressora fique pronta para receber caracteres
  - Precisamos, então, usar **interrupções**
  - Quando a chamada de sistema para imprimir a cadeia é feita, o buffer é copiado para o espaço do núcleo e o primeiro caractere é copiado para a impressora, tão logo ela esteja disposta a aceitá-lo
  - **Nesse ponto a CPU chama o escalonador e algum outro processo é executado**
    - Ou seja o processo que solicitou a impressão é bloqueado até que a cadeia inteira seja impressa

```
copy_from_user(buffer, p, count);
enable_interrupts( );
while (*printer_status_reg != READY) ;
*printer_data_register = p[0];
scheduler( );
```

E/S orientada a interrupções

- Quando a impressora imprimiu o caractere e está preparada para aceitar o próximo, ela gera uma interrupção
  - Essa interrupção para o processo atual e salva seu estado
  - Então a rotina de tratamento de interrupção da impressora é executada
  - Se não houver mais caracteres a serem impressos, o tratador de interrupção executa alguma ação para desbloquear o usuário
  - Caso contrário, ele sai com o caractere seguinte, reconhece a interrupção e retorna o processo que estava sendo executado um momento antes da interrupção

```
if (count == 0) {
    unblock_user( );
} else {
    *printer_data_register = p[i];
    count = count – 1;
    i = i + 1;
}
acknowledge_interrupt( );
return_from_interrupt( );
```

E/S usando DMA

- Desvantagem óbvia do mecanismo de E/S orientado à interrupção:
  - Temos uma interrupção em cada caractere
  - Interrupções levam tempo, logo desperdiça certa quantidade de tempo de CPU
- Solução: usar **acesso direto à memória (DMA)**
  - **Ideia:** deixar que o controlador de DMA alimente os caracteres para a impressora um de cada vez, sem que a CPU seja incomodada

```
copy_from_user(buffer, p, count);
set_up_DMA_controller( );
scheduler( );
```

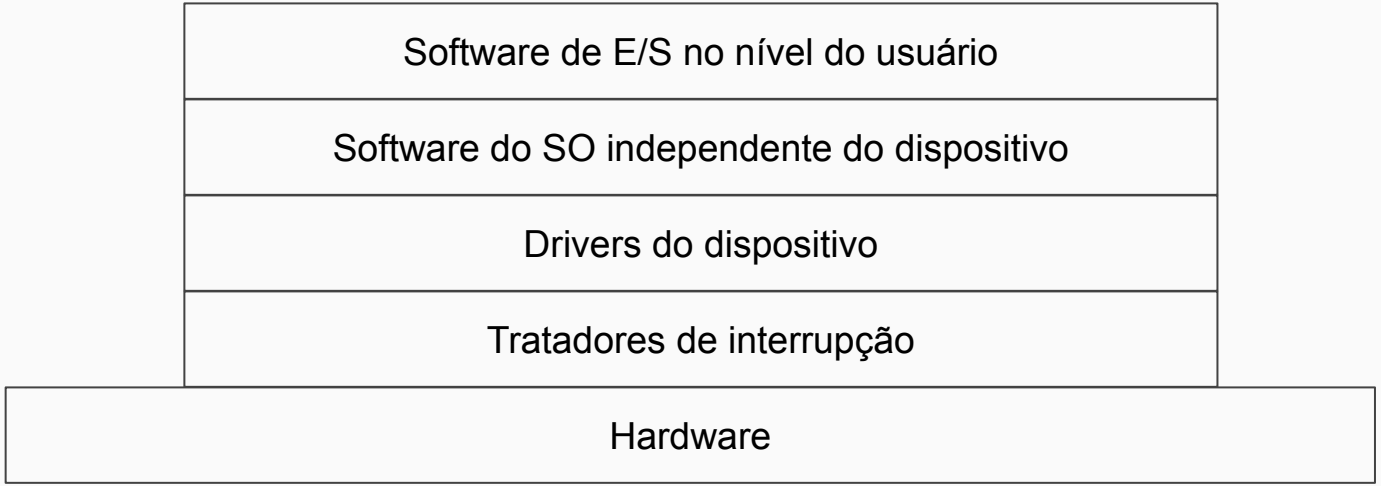
```
acknowledge_interrupt( );
unblock_user( );
return_from_interrupt( );
```

  - **Vantagem**: reduz o número de interrupções de uma por caractere para uma por buffer impresso

Camadas do Software de E/S

Camadas do software de E/S

- Software de E/S costuma estar organizado em quatro camadas
  - Cada camada tem uma função bem definida a desempenhar e uma interface bem definida para as camadas adjacentes



Tratadores de Interrupção

- Interrupções são um fato desagradável da vida e não podem ser evitadas :-(
  - Deve-se escondê-las de forma que a menor parcela do SO saiba da ocorrência delas
  - **Melhor forma de escondê-la:**
    - Bloquear o driver que inicializou uma operação de E/S até que ela se complete e a interrupção ocorra
    - O driver pode bloquear-se a si mesmo realizando, por exemplo uma chamada **down** em um semáforo, ou algo similar

Tratadores de Interrupção

- Quando a interrupção acontece:
  - **Rotina de interrupção:** deve lidar com a interrupção, de forma a desbloquear o driver bloqueado
    - Em alguns casos deverá emitir um comando **up** em um semáforo ou emitir um sinal sobre uma variável de condição
  - Esse modelo funciona melhor se os drivers estiverem estruturados em processos do núcleo
- **Infelizmente, na realidade, o tratamento de interrupção não é bem simples**

Tratadores de Interrupção (Exemplo)

- Exemplos de passos que devem ser realizados no software após a interrupção de hardware ter sido completada
  1. Salvar quaisquer registros que ainda não foram salvos pelo hardware de interrupção
  2. Estabelecer um contexto para a rotina de tratamento de interrupção
     - Isso pode envolver a configuração de TLB, MMU e uma tabela de páginas
  3. Estabelecer uma pilha para a rotina de tratamento de interrupção
  4. Sinalizar o controlador de interrupções
  5. Copiar os registradores de onde eles foram salvos para a tabela de processos

Tratadores de Interrupção (Exemplo)

- Exemplos de passos que devem ser realizados no software após a interrupção de hardware ter sido completada
    6. Executar uma rotina de tratamento de interrupção
    7. Escolher qual processo executar em seguida
    8. Escolher o contexto de MMU para o próximo processo e executar, além de alguns ajustes na TLB
    9. Carregar os registradores do novo processo
    10. Começar a execução do novo processo.

Drivers dos dispositivos

- Números de registradores do dispositivo e a natureza dos comandos variam de dispositivo para dispositivo
  - Ex.: Driver de mouse x Driver de disco
- Cada dispositivo de E/S precisa de algum código específico do dispositivo para controlá-lo
  - Esse código é chamado **driver do dispositivo**

Drivers dos dispositivos

- **Driver do dispositivo**
  - Geralmente escritos e fornecidos pelo fabricante do dispositivo para diferentes SOs
  - Normalmente lida com um tipo, ou no máximo, uma classe de dispositivos
    - Em alguns casos, dispositivos completamente diferentes são baseados na mesma tecnologia subjacente, por exemplo o USB

Drivers dos dispositivos

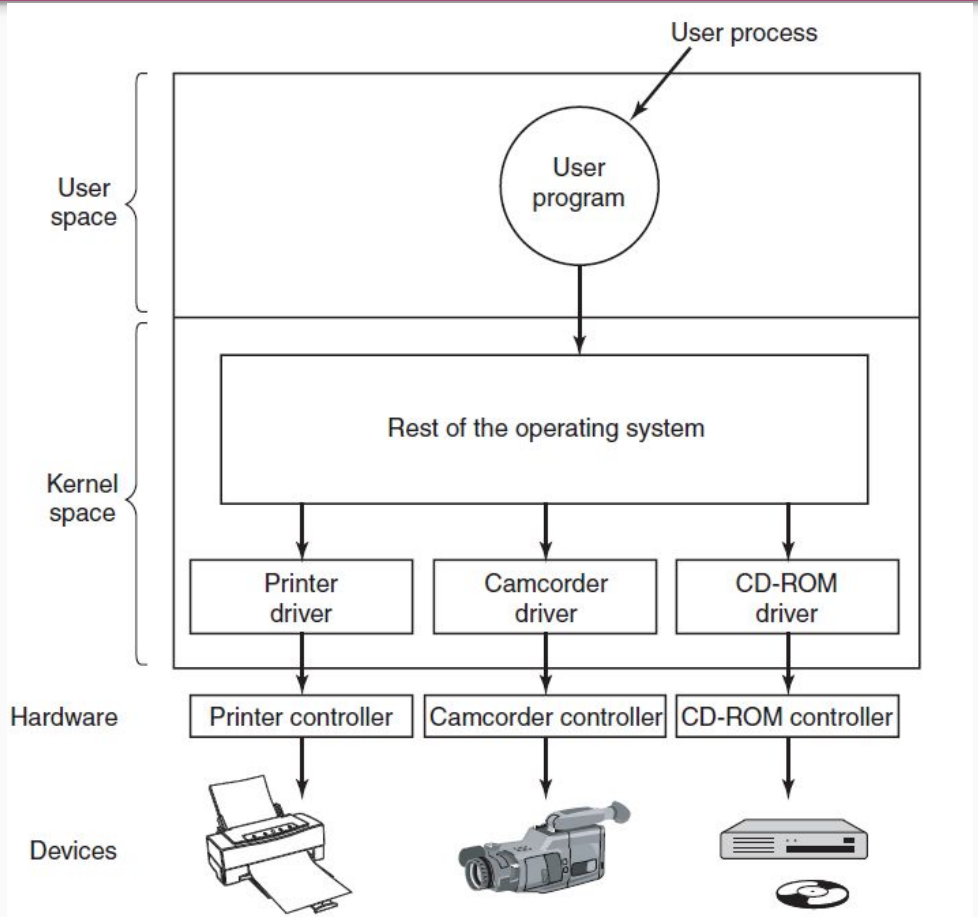
- Dispositivos USB
  - Drivers USB são tipicamente empilhados, como uma pilha de TCP/IP em redes
  - **Nas camadas mais baixas,** em geral no hardware encontramos a **camada do link do usb** (E/S serial)
    - Lida com questões de hardware, como sinalização e decodificação de um fluxo de sinais para os pacotes USB
  - **Acima dessas camadas,** que usam a saída das camadas mais baixas, lidando com p**acotes de dados e a funcionalidade comum para USB** que é compartilhada pela maioria dos dispositivos.
  - **No topo,** encontramos as APIs de camadas superiores, como as interfaces para armazenamento em massa, câmeras, dentre outros...
    - Deste modo, temos drivers de dispositivos em separado, embora compartilhem de parte da pilha do protocolo

Drivers dos dispositivos

- Para acessar o hardware (ou seja, os registradores do controlador), o driver deve fazer parte do **núcleo do SO**
  - Pode-se, também, construí-los n**o espaço de usuário**, com chamadas de sistemas para leitura e escrita nos registradores de dispositivos
    - Qual vantagem teríamos nesse caso?
  - Embora vantajoso construí-lo no espaço de usuário, a maioria dos SOs implementa os drivers no núcleo do SO.

Drivers dos dispositivos

- Drivers de dispositivos costumam ser posicionados abaixo do resto do sistema operacional
- **Drivers de bloco x Drivers de caracteres**
- SOs definem uma interface padrão a que todos os drivers de blocos devem dar suporte e uma interface padrão a que todos os drivers de caracteres devem dar suporte
    - Série de rotinas que o resto do SO pode utilizar para fazer o driver trabalhar para ele



Driver dos dispositivos

- Drivers de dispositivos apresentam diversas funções
  - **Principal:** Aceitar solicitações abstratas de leitura e escrita de um software independente de dispositivo localizado na camada dele e verificar que elas sejam executadas
  - **Outras funções:**
    - Inicializar o dispositivo
    - Gerenciar suas necessidades de energia
    - Gerenciar seus eventos
    - Dentre outros

Driver dos dispositivos

- No geral, possuem estrutura similar
  - Um driver típico inicia verificando os parâmetros de entrada para ver se são válidos
  - Se forem válidos, uma tradução dos termos abstratos para os termos concretos é realizada
    - Ex.: Driver de disco
      - Converter um bloco linear em números de cabeçote, trilha, setor e cilindro para a geometria do disco

Driver dos dispositivos

- Em seguida, o driver pode conferir se o dispositivo está em uso no momento
  - Se estiver, a solicitação entra em uma fila para processamento posterior
- Se estiver ocioso, o estado do hardware é examinado para ver se a solicitação pode ser cuidada de imediato
  - Ex.: talvez seja necessário ligar o dispositivo ou motor antes da transferência iniciar
- Uma vez que esteja ligado e pronto para trabalhar o **controle** de verdade pode começar

Driver dos dispositivos

- Controlar um dispositivo significa emitir uma sequência de comandos para ele
  - O driver é o local onde a sequência de comandos é determinada
  - Após o drive saber o que vai emitir, ele começa escrevendo-os nos registradores do controlador do dispositivo
- Após os comandos terem sido emitidos, ocorrerá uma de duas situações
  - O driver deve esperar até que o controlador realize algum trabalho por ele, de modo que ele o **bloqueie a si mesmo** até que a interrupção chegue para desbloqueá-lo; Ou
  - A operação termina sem atraso, então **o drive não precisa bloquear**
    - **Ex.:** Rolagem da tela exige que sejam escritos apenas alguns bytes nos registradores do controlador (a operação é completada em nanossegundos)
- Por fim, ele retorna ao seu chamador alguma informação de estado para conferir o relatório de erros

Software de E/S independente de dispositivo

- **Função principal:** realizar as funções de E/S que são comuns a todos os dispositivos e fornecer uma interface uniforme para o software no nível de usuário

Uniformizar interfaces para os drivers de dispositivos
Armazenar no buffer
Reportar Erros
Alocar e Liberar dispositivos dedicados
Providenciar um tamanho de bloco independente de dispositivo

Exemplos de funções do software de E/S independente de dispositivo

Software de E/S do espaço de usuário

- Uma pequena parte dos software de E/S operam em espaço de usuário
  - Bibliotecas ligadas aos programas de usuário
    - Realizam chamadas de sistemas de E/S
    - Ex. de rotina que realiza E/S: *Printf*
      - Recebe uma cadeia de caracteres e algumas variáveis de entrada
      - Constrói uma cadeia ASCII, e então chama a rotina *write* para colocá-la na saída padrão

Software de E/S do espaço de usuário

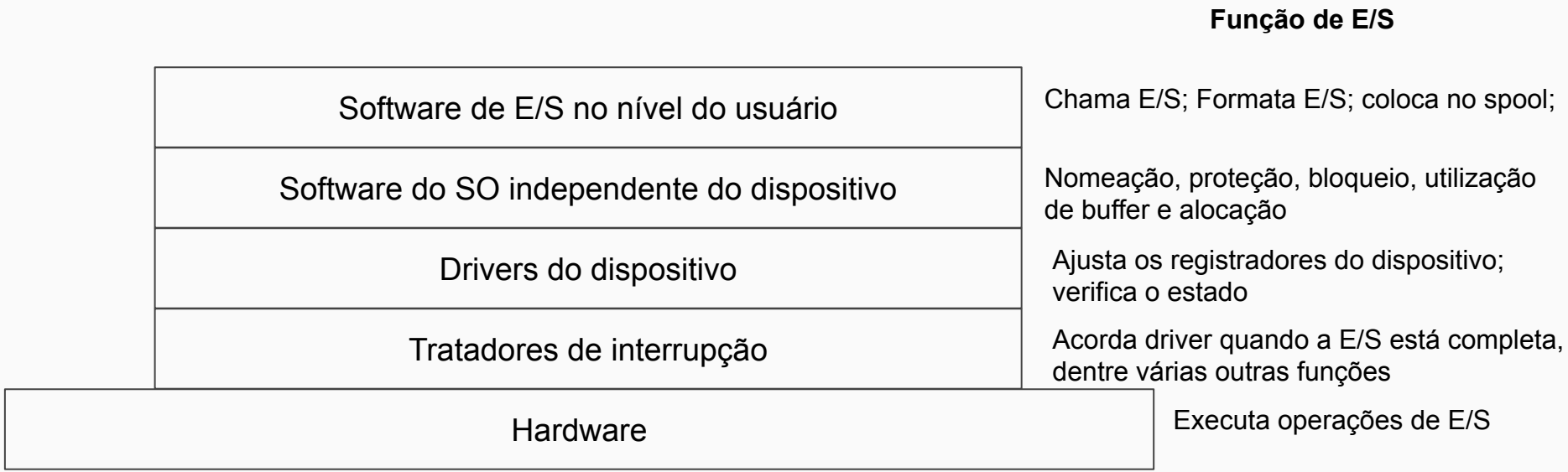
- Uma pequena parte dos software de E/S operam em espaço de usuário
  - Sistema Spooling
    - Lida com dispositivos de E/S dedicados em um sistema de multiprogramação
    - Dispositivo *spooled* típico: **impressora**
      - Embora seja fácil deixar qualquer processo de usuário abrir o arquivo especial de caractere para a impressora, suponha que um processo abriu o arquivo e não fez nada por horas
      - **Nenhum processo poderia imprimir nada neste período!**

Software de E/S do espaço de usuário

- Solução: Criar um processo especial chamada ***Daemon*** e um diretório especial chamado **diretório de spooling**
  - Para imprimir um arquivo, um processo primeiro gera o arquivo inteiro a ser impresso e coloca no diretório de spooling
  - Cabe ao ***daemon,*** que é o único processo com permissão de usar o arquivo especial da impressora, imprimir os arquivos no diretório

Camadas do software de E/S

- Software de E/S costuma estar organizado em quatro camadas
  - Cada camada tem uma função bem definida a desempenhar e uma interface bem definida para as camadas adjacentes



Referências

Tanenbaum, A. S. e Bos, H.. Sistemas Operacionais Modernos. 4.ed. Pearson/Prentice-Hall. 2016.