Renan da Cunha e Arthur Henrique

Roteiro de gerenciamento de I/O

Sumário

[Parte I 3](#_Toc373234872)

[Princípios de Hardware: 3](#_Toc373234873)

[Tipo de dispositivos de E/S: 3](#_Toc373234874)

[Controlador de dispositivo: 3](#_Toc373234875)

[E/S mapeado em memória: 4](#_Toc373234876)

[Acesso direto à memória(DMA): 5](#_Toc373234877)

[Interrupções e vetor de interrupções: 6](#_Toc373234878)

[Interrupções precisas e imprecisas: 6](#_Toc373234879)

[Parte II 7](#_Toc373234880)

[Conceito Independência de dispositivos: 7](#_Toc373234881)

[Tratamento de erros: 7](#_Toc373234882)

[E/S e interrupções: 7](#_Toc373234883)

[E/S e DMA: 8](#_Toc373234884)

[Interface de software: 8](#_Toc373234885)

[Parte III 9](#_Toc373234886)

[Tratamento de interrupção: 9](#_Toc373234887)

[Driver de dispositivo: 10](#_Toc373234888)

[Software de E/S independente de dispositivos: 12](#_Toc373234889)

[Buffer: 13](#_Toc373234890)

[Software de E/S no nível de usuário: 14](#_Toc373234891)

# Parte I

## Princípios de Hardware:

Um sistema baseado em computador é, na verdade, composto por hardware e software. Hardware é o nome que se dá para a parte física do computador. É tudo que você pode tocar (mouse, teclado, caixas de som, placas, fios, componentes em geral). Software é o nome que se dá a toda parte lógica do computador. Ou seja, são os programas que você vê funcionar na tela do micro e que dão "vida" ao computador.

Sem um software adequado à suas necessidades, o computador, por mais bem equipado e avançado que seja, é completamente inútil.

O computador não é uma máquina com inteligência. Na verdade, é uma máquina com uma grande capacidade para processamento de informações, tanto em volume de dados quanto na velocidade das operações que realiza sobre esses dados. Basicamente, o computador é organizado em três grandes funções ou áreas, as quais são: entrada de dados, processamento de dados e saída de dados.

## Tipo de dispositivos de E/S:

Os dispositivos I/O podem se conectar de forma serial ou paralela. Na interface serial existe apenas uma linha por onde os dados trafegam. Na interface paralela os dados são transmitidos simultaneamente através das várias linhas para dados, a quantidade de linhas é um múltiplo de 1 byte (8 bits).

As informações sobre os endereçamentos de I/O ficam armazenadas, Sistema Operacional Linux, no arquivo /proc/ioports

A transferência pode ser síncrona (bloqueante) - a CPU inicia uma transferência e segue realizando outra atividade até ser sinalizada por um interrupção (o que acontece na maioria das E/S físicas), ou assíncrona (orientada à interrupção) - após um read, o programa é suspenso até que os dados estejam disponíveis no buffer.

## Controlador de dispositivo:

Dispositivos de hardware precisam ser controlados para proporcionar a entrada e saída de dados para o processador. O controle do hardware é realizado por meio de hardware e software apropriados.

A porção de hardware é denominado controlador de hardware e segue padrões determinados pelo barramento (IDE, SCSI, USB, etc), assim, ligados a cada tipo de barramento existem controladores de hardware: controladora de hardware IDE, controladora de hardware SCSI, etc.

Para utilizar um dispositivo de hardware, é necessário conectá-lo a interface física da controladora de hardware. Por exemplo, um disco rígido IDE deve ser conectado a uma das quatro interfaces disponíveis pela controladora IDE. Em geral, o Sistema Operacional pode ter softwares controladores de dispositivo (driver de dispositivos). Os drivers de dispositivos para a controladora de hardware, geralmente são genéricos, embutidos no próprio Sistema Operacional. E os drivers para dispositivos de hardware são geralmente específicos, uma vez que controlam funcionalidades específicas providas pelos fabricantes.

Periférico é dado como qualquer dispositivo de hardware conectado a um computador de forma a permitir a sua interação com o mundo externo.

## E/S mapeado em memória:

Cada controlador tem alguns registradores usados para a comunicação com a CPU. Por meio da escrita nesses registradores, o SO pode comandar o dispositivo para entregar ou aceitar dados, alternar em ligar/desligar, ou ainda executar alguma outra tarefa. A partir da leitura desses registradores o SO pode descobrir o estado do dispositivo, se ele está preparado para aceitar um novo comando, etc. Além dos registradores, usualmente os dispositivos também possuem um buffer.

Existe duas formas para a CPU ter acesso a esses registradores e buffers:

1º → Cada registrador de controle é associado a um número de porta de E/S, usando uma instrução especial de E/S (IN REG, PORT OUT REG, PORT) que seria um acesso direto.

2º → Mapear todos os registradores de controle no espaço de endereçamento da memória. Cada registrador de controle é associado a um endereço de memória único ao qual nenhuma memória é associada, que seria o mapeamento da E/S na memória.

Essa segunda forma, nos traz duas grandes vantagens, a primeira é que o programador pode tratar esses registradores como se trata a memória normal, isso é, como uma variável, isso faz com que um driver possa ser completamente escrito em C/C++, caso seja feita a primeira abordagem, necessariamente o driver terá algum código em ASSEMBLY. A outra vantagem é que não é necessário qualquer mecanismo de proteção especial para impedir que os processos do usuário executem E/S, tudo o que o SO tem de fazer é deixar de mapear aquela porção do espaço de endereçamento associada aos registradores de controle no espaço de endereçamento virtual do usuário.

## Acesso direto à memória(DMA):

O Acesso Direto à Memória (DMA) é uma das técnicas utilizadas para otimizar o uso de memória por dispositivos. O DMA é um componente de hardware que permite a transferência direta de dados entre dispositivos periféricos e a memória principal, tornando assim dispensável a participação da CPU neste processo.

O controlador de DMA é um hardware desenvolvido para desempenhar toda a sequência de transferência de dados acessando diretamente a memória. Ele gerencia vários canais que podem ser programados para realizar a transferência de dados, quer seja de um dispositivo para a memória ou vice-versa.

O SO somente pode usar o DMA se o hardware tem o controlador de DMA. O DMA melhora o desempenho do sistema, pois poupo tempo ocioso da CPU, que poderia muito bem executar a tarefa do DMA, porém como o tempo de E/S é grande principalmente para grandes quantidades de dados pode fazer com que a CPU fique muito tempo ociosa. Quando a quantidade é pequena as vezes é até mais viável fazer a transferência direto pela CPU que é um hardware mais rápido que o DMA, isso pode causar concorrência no barramento, pois o barramento utilizado pelo DMA para acessar a memória é o mesmo utilizado pela CPU. Utilizando o DMA, a CPU requisita ao DMA de onde começar a ler os bytes, quantos bytes devem ser lidos/escritos e fica livre para executar outras tarefas que sejam CPU Bound, então quando o DMA termina de realizar sua tarefa, ele transmite um sinal de interrupção para a CPU que simplesmente usa os bytes.

Note que a CPU pode fazer exatamente o que o DMA faz, isso fica a cargo de projeto. Coloca uma requisição de leitura no barramento e fica esperando até receber os bytes e assim poder usá-lo, a diferença é que usando a CPU para transferência de uma quantidade maior de dados, poderá ocasionar em CPU ociosa.

Um computador tem dois gerenciadores de DMA, divididos em canais. Os canais 0 a 3 são gerenciados por um gerenciador, enquanto os canais 4 a 7 vão para o outro.

Para visualizar quais canais estão em uso em sistemas Linux basta digitar no shell o comando:

$ cat /proc/dma

Os oito canais e seus usos são descritos a seguir: 0 = usado pelo refresh da memória Ram dinâmica. 1 = placas de som de 8 bits, adaptadores SCSI, placas de rede. 2 = controladora de disquete 3 = porta paralela ECP, placas de som de 8 bits, adaptadores SCSI, placas de rede, controladores de scanner antigos. 4 = ponte para a controladora de DMA 0~3 5 a 7 = placas de som, adaptadores SCSI, placas de rede.

Os canais 1 a 3 operam sob 8 ou 16 bits, enquanto os canais 5 a 7 operam apenas sob 16 bits.

Dispositivos PCI (e outros de alta velocidade) possuem um controlador de DMA embutido, muito mais rápido que o DMA simples. Por exemplo, esse controlador é usado em discos rígidos atuais e pode atingir velocidades de 66MB/s.

## Interrupções e vetor de interrupções:

Virtualmente cada processador reserva uma área de memória especifica para tratar cada uma das interrupções. Estas localizações são chamadas vetores de interrupção.

Este tipo exige a identificação do dispositivo periférico que solicita interrupção. Essa identificação é utilizada para a localização do endereço da sub-rotina de tratamento da interrupção, em uma tabela localizada em uma região determinada da memória do **microprocessador (vetor de interrupções**). Exemplo: interrupções da família 68000.

## Interrupções precisas e imprecisas:

A informação fundamental que deve ser salva sobre um processo, visando restaurar sua execução em um momento posterior, é o valor do contador de programa. Na prática, outras informações também devem ser salvas, mas o contador de programa é essencial pois somente ele pode definir quais instruções realmente foram executadas antes da interrupção. Dependendo de quais instruções foram completadas ou não, quando do tratamento da interrupção, pode-se classificar as interrupções como precisas ou imprecisas.

Interrupções precisas são uma forma de garantir que um sistema com processamento de interrupções possa continuar a execução correta das instruções, e para serem classificadas assim devem obedecer às seguintes exigências, conhecidas como Condições de Interrupções Precisas de Smith e Pleszkun [SP88]:

Todas instruções despachadas antes da instrução indicada pelo contador de programa salvo já terminaram sua execução, e modificaram corretamente o estado do processo;

Todas instruções despachadas após a instrução indicada pelo contador de programa não foram executadas e não modificaram o estado do processo;

Se uma interrupção foi causada por uma instrução, o contador de programa aponta para aquela instrução. Esta instrução de interrupção deve estar ou completamente executada ou completamente não executada.

Com uma interrupção precisa, o estado do processo que ocorreu logo antes do processamento da interrupção é descrito como serialmente correto. Isto significa que o estado do processo está como se o programa tivesse sido executado serialmente, uma instrução de cada vez.

Quando o tratamento de uma interrupção não consegue atender a essas três regras, denominamos como sendo uma interrupção imprecisa, que pode levar a uma situação onde a recuperação do estado do processo seja impossível. Um caso especial refere-se ao desenvolvimento de sistemas para um processador superescalar, onde as instruções de um programa podem ser despachadas e executadas fora da ordem original, e por isso não pode-se contar com o valor armazenado no contador de programa para identificar quais os comandos dentro da janela de instruções já foram executados.

# Parte II

## Conceito Independência de dispositivos:

Esse conceito trabalho sobre a possibilidade de escrever programas capazes de acessar um dispositivo E/S sem que seja necessário um conhecimento prévio sobre qual é o dispositivo. Ou seja, um programa deverá ser capaz de ler/escrever um arquivo da mesma forma para qualquer dispositivo. Para o caso de se obter a entrada de um dispositivo para saída em outro - como em "/dispositivo1/arquivo > /dispositivo2", o Sistema Operacional fica incumbido de tratar dos problemas causados pelo fato de os dispositivos serem desiguais e necessitarem de sequências de comandos muito diferentes para leitura e escrita.

## Tratamento de erros:

De maneira geral, espera-se que os erros, como de leitura por exemplo, sejam tratados em níveis mais baixos, o mais próximo do hardware.

## E/S e interrupções:

Diferentemente da E/S programada, em que a CPU fica verificando o dispositivo para saber se ele está pronto para mais operações ou se terminou, na E/S orientada à interrupção, a CPU passa a realizar outras tarefas escalonadas até que seja informada pelo dispositivo, por meio de uma interrupção, que este está pronto para mais caracteres ou concluiu sua tarefa. Quando isso acontecer, o processador para o que está fazendo para executar o **tratador de interrupção**, quando sua execução estiver terminada, o processador volta à rotina que foi interrompida inicialmente.

Para que seja empregada essa política de interrupções existem detalhes de software e de hardware que devem ser atendidos, e para que esses detalhes sejam executados, a maioria dos computadores possuem um hardware denominado **controlador de interrupções**. As principais funções do controlador de interrupções são:

1. Identificar a fonte da interrupção;
2. Priorizar uma interrupção em relação a outra;
3. Selecionar quais interrupções serão atendidas.

O emprego de interrupções libera o computador para realizar cálculos, então o processador fica responsável apenas por iniciar a operação de entrada-saída, e quando esta for concluída, executar o tratador de interrupção.

### 

## E/S e DMA:

No método de E/S orientada à interrupção, a cada caractere processado, é gerado uma nova interrupção à CPU. Para diminuir o peso de processamento sobre a CPU, que perderia muito tempo por conta das contínuas interrupções, passa-se tal tarefa para o DMA, que passará a administrar as interrupções por buffer (não mais por caractere). O DMA executa então a E/S programada (neste caso, não é a CPU que faz o trabalho, mas sim o controlador do DMA).

Quando existe uma quantidade de dados significante para ser transferida a técnica de Interrupções se torna ineficaz, sendo melhor a utilização de um hardware especial (**Controlador de DMA**), que transfere os dados diretamente de um dispositivo de E/S para a memória, ou vice e versa.

A transferência por DMA acontece quando o processador inicializa o controlador DMA, fornecendo todas as informações necessárias sobre os dados a serem transferidos (quantidade de dados, origem e destino dos blocos e ainda o sentido da transferência), depois ele dispara a execução do DMA e enquanto a transferência estiver ocorrendo o processador pode se dedicar a outra tarefa. Ao final da transferência o DMA sinaliza ao processador por meio de uma interrupção de hardware.

Interface de software:

Em programação, a utilização de interfaces permite a composição de componentes de um software sem que a sua implementação seja conhecida. Um exemplo clássico de utilização de interfaces é o do sistema operacional que, através de uma interfaces de programação de aplicativos, permite que os programas utilizem os recursos do sistema (memória, CPU e etc) sem que os seus detalhes de implementação sejam conhecidos do programador. Este esquema isola e protege o sistema operacional de eventuais erros cometidos pela aplicação.

Os componentes de software utilizam interfaces padronizadas para criar uma camada de abstração que facilite a reutilização e a manutenção do software. Neste cenário, a interface de um módulo de software A deve ser mantida em separado da sua implementação e qualquer outro módulo B, que interaja com A (cliente de A), deve ser forçado a fazê-lo apenas através da interface. Este mecanismo permite que no caso de uma alteração em A, o módulo B continue funcionando; desde que a utilização do módulo A pelo módulo B satisfaça as especificações da interface. (Ver também o princípio da substituição de Liskov).

Uma interface disponibiliza tipos variados de acesso entre componentes, como por exemplo: constantes, tipos de dado, procedimentos, especificação de exceções e assinaturas de métodos. Em alguns casos é mais apropriado definir as variáveis como parte das interfaces. As interfaces também especificam a funcionalidade disponibilizada através de comentários ou através de declarações lógicas formais (assertivos).

Parte III

## Tratamento de interrupção:

Tratadores de interrupção são parte do SO, usualmente bem

Escondidos.

Tendo o driver iniciado uma operação de E/S o mesmo é bloqueado até

Que a operação se complete e uma interrupção ocorra (down, wait,

Receive)

Ocorrendo a Interrupção a rotina de interrupção faz sua tarefa

Em seguida libera o driver que iniciou a operação de E/S (up, signal,

Send)

Processar uma interrupção não é apenas interceptar uma interrupção,

Sinalizar o driver e executar um IRET

Existe trabalho adicional realizado pelo SO

INE5412– 2011.2

27

Passos que devem ser realizados em software depois da interrupção ser

Concluída:

Salvar regs ainda não salvos pelo hardware da interrupção.

Estabelece um contexto para rotina de tratamento da interrupção (configuração

de TLB, MMU, tab. Páginas).

Estabelece uma pilha para rotina de tratamento da interrupção.

Sinaliza o controlador de interrupções. Senão existe um, reabilita interrupções.

Copiar registradores de onde foram salvos (pilha) para a tabela de processos.

Executa rotina de tratamento de interrupção. Ela extrai informações dos

Registradores do controlador do dispositivo que está interrompendo.

Escolhe próximo processo a executar. Se a interrupção deixou pronto algum

Processo de alta prioridade, este pode ser escolhido.

Estabelece contexto da MMU para próximo processo a executar. Algum ajuste

Na TLB também pode ser necessário.

Carrega registradores do novo processo, incluindo PSW.

Inicializa a execução do novo processo.

## Driver de dispositivo:

Um driver é uma camada de software que faz a comunicação do sistema operacional com o controlador do hardware que por sua vez faz a interface com o hardware. Drivers escondem as diferenças entre os diversos dispositivos, através de uma interface de programação única.

Driver de dispositivo é responsável por implementar as rotinas necessárias ao acesso e à gerência de um dispositivo específico.

A camada de drivers de dispositivo representa uma parte significativa do sistema de entrada e saída em relação às funcionalidades. Ela é responsável por implementar as rotinas necessárias ao acesso e à gerencia de um dispositivo específico. É necessário que o software de E/S realiza a programação de registradores internos de controladores que compõem a interface física dos dispositivos e implementa os respectivos tratadores de interrupção. Assim, cada tipo de dispositivo requer um driver apropriado. Essa camada fornece uma abstração a mais genérica possível para a camada superior, a de E/S independente do dispositivo.

Cada dispositivo de E/S ligado ao computador precisa de algum código específico do dispositivo para controlá-lo. Esse código, chamado de driver do dispositivo.

Para acessar o hardware do dispositivo, o driver normalmente deve ser parte do núcleo do SO.

Os sistemas operacionais geralmente classificam os drivers dentre algumas poucas categorias. As categorias mais comuns são dispositivos de bloco - os quais contêm vários blocos de dados que podem ser endereçados independentemente - e os dispositivos de caractere, os quais geram ou aceitam uma sequência de caracteres.

A maioria dos SOs define uma interface-padrão para todos os drivers de blocos e uma segunda interface-padrão para todos os drivers de caracteres. Essas interfaces consistem em um número de procedimentos que o restante do só pode utilizar para fazer o driver trabalhar para ele.

Um driver de dispositivo apresenta várias funções. A mais óbvia é aceitar e executar requisições abstratas, de leitura ou gravação, de um software independente de um dispositivo localizado na camada acima da camada de drivers dos dispositivos. Mas existem também algumas poucas outras funções que ele tem de executar.

Drivers no Linux X Drivers no Windows

Enquanto no Windows, os drivers são desenvolvidos pelos próprios fabricantes do dispositivo, precisando ser instalados manualmente e seguindo de um processo de reboot do sistema. Em ambientes GNU/Linux, a instalação dos "drivers" são incorporados diretamente ao Kernel e vêm pré-instalados no sistema.

Drivers Linux

Em sistemas GNU/Linux os "drivers" são chamados de módulos. O kernel desse sistema é dito monolítico com vantagens de micro-kernel já que os sistemas GNU/Linux são LKM (Loadable Kernel Modules), já que os "drivers" ou módulos são carregáveis ao sistema sem a necessidade de um novo processo de bootstrap após a instalação. Para maior facilidade com o usuário, esse carregamento é feito no processo de bootstrap. Na verdade na maioria dos sistemas operacionais que conta com o desenvolvimento da comunidade do software livre, os módulos são desenvolvidos pela comunidade de desenvolvimento. Se esse módulo tem uma boa aceitação pela própria comunidade e passou por todos os requisitos propostos pela equipe de desenvolvimento do kernel do sistema, esse pode passar a fazer parte do próprio kernel do sistema. Dessa forma, o usuário não precisa correr atrás da instalação uma vez que o driver está incorporado e instalado no sistema.

Drivers Windows

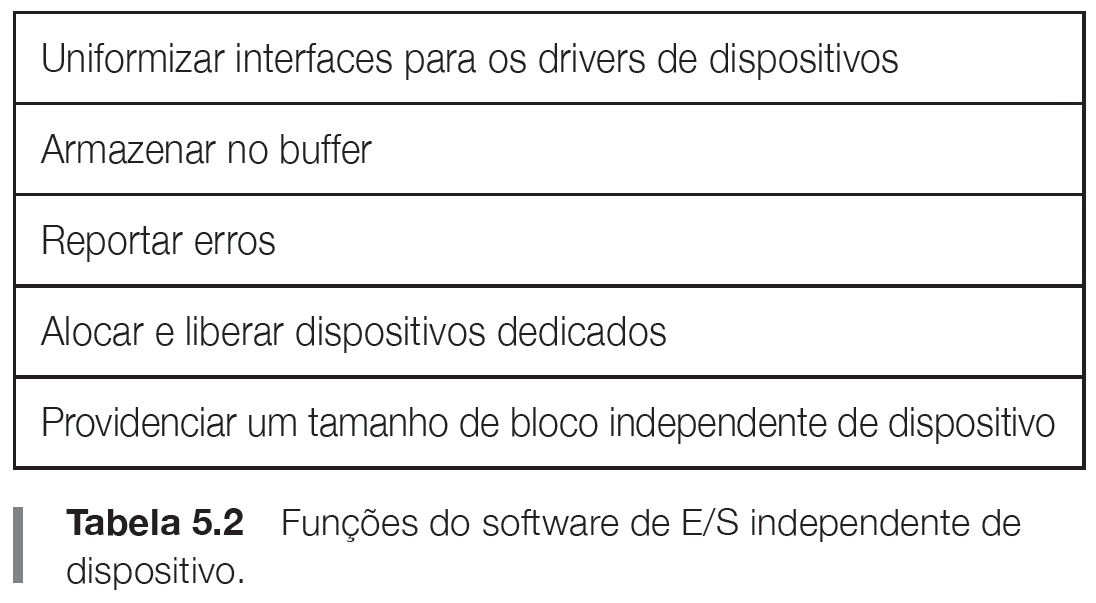
Nem sempre a companhia que desenvolveu certo hardware, também tem que desenvolver o driver para o mesmo. Há casos em que o hardware foi desenvolvido sobre um certo padrão de hardware. Nesses casos, um driver genérico é desenvolvido pela Microsoft para esse dado padrão. Nem todos os drivers se comunicam diretamente com o dispositivo. Pode haver uma pilha de drivers para determinado dispositivo, em que parte deles age como um filtro, transformando os dados de um formato para o outro, enquanto apenas a base da pilha se comunica diretamente com o dispositivo. Isso pode ser melhor visualizado por uma imagem disponibilizada pela própria Microsoft em seu site:

Também, nem sempre os drivers estarão associados a um dispositivo. Também existe os “Software Drivers”. Em determinados momentos, uma aplicação precisa de acessar recursos que ela só poderia acessar em modo kernel, porém ela está em modo usuário. Então, divide-se essa aplicação em dois componentes: a aplicação que rodará em modo usuário, que fará a interface com o usuário e um driver que rodará em modo kernel, dando acesso aos recursos necessários. Esse driver que roda em modo kernel é chamado de Software Driver. No site da Microsoft também está disponível uma imagem que ilustra isso:

Em casos em que determinado hardware desenvolvido não segue um padrão pré-existente, que tenha driver desenvolvido pela Microsoft, a companhia também precisa desenvolver o driver. No site da Microsoft, existem explicações sobre como construir um driver para Windows, dicas de performance e outros

Para desenvolver esses drivers, o desenvolvedor deve saber conceitos do funcionamento interno do Windows, como o gerenciamento de memória, fluxo de entrada e saída, entre outros. Para ajudar em seu desenvolvimento, a Microsoft disponibiliza o Windows Driver Kit (WDK), com uma série de utilitários para criar um driver, como bibliotecas, ambiente de desenvolvimento, exemplos, etc. Como parte do WDK, também é disponibilizado o Windows Driver Foundation (WDF). O WDF define um único modelo de driver que pode ser usado para criar drivers orientado a objetos, tanto para o modo kernel quanto para o modo usuário. Ele também inclui frameworks para drivers de modo kernel e modo usuário, além de uma série de utilitários de verificação para o driver.

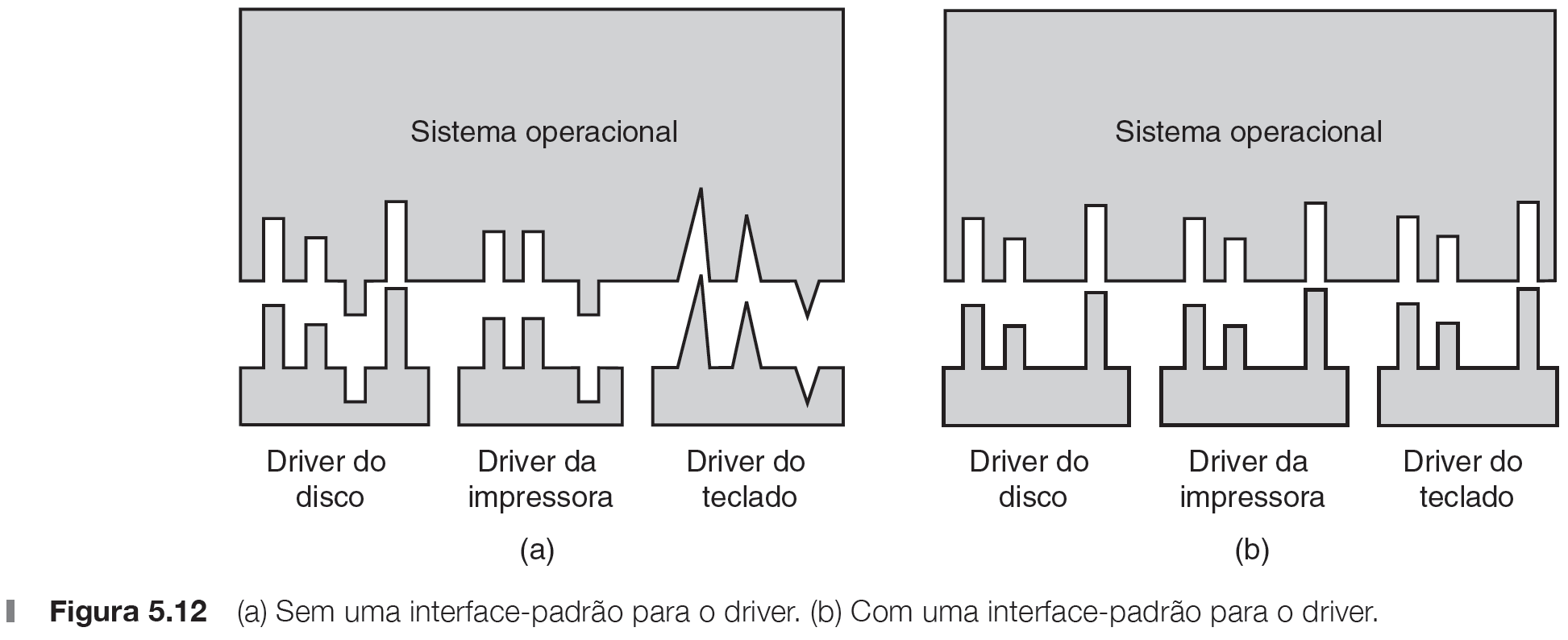
## Software de E/S independente de dispositivos:



Partes do software de E/S são independentes de dispositivo. As funções básicas de um software

Independente de dispositivo são executar as funções de E/S comuns para todos os dispositivos e

Fornecer uma interface uniforme para o software no nível de usuário.



Uma questão importante em um sistema operacional é como fazer todos os dispositivos de E/S e drivers parecerem mais ou menos os mesmos.

(a) Com interfaces diferentes entre drivers e o restante do SO para cada driver.

• Funções diferentes para cada driver

(b) Com uma interface padrão do driver

Cada classe (disco) define um conjunto de funções

• Nomeação dos dispositivos Ex. /dev/disk0

Buffer:  
  
Em ciência da computação, buffer (retentor) é uma região de armazenamento de memória física utilizada para armazenar temporariamente os dados enquanto eles estão sendo movidos de um lugar para outro. Normalmente, os dados são armazenados em um buffer enquanto eles são recuperados de um dispositivo de entrada (como um microfone) ou pouco antes de serem enviados para um dispositivo de saída (como auto-falantes). Contudo, um buffer pode ser utilizado quando movem-se dados entre processos dentro de um computador. Isto é comparável com os buffers de telecomunicações. Os buffers podem ser implementados em um local de memória fixa no hardware - ou utilizando-se um buffer de dados virtual em software, apontando para um local na memória física. Em todos os casos, os dados armazenados em um buffer de dados são armazenados em um meio de armazenamento físico. A maioria dos buffers são implementados em software, que normalmente usa a RAM mais rápida para armazenar dados temporários, devido ao tempo de acesso muito mais rápido comparado com as unidades de disco rígido. Os buffers normalmente são usados quando há uma diferença entre a taxa a qual os dados são recebidos e a taxa a qual eles podem ser processados, ou no caso em que estas taxas são variáveis, por exemplo em um spooler de impressão ou em streaming de vídeos online.

Um buffer frequentemente ajusta o tempo pela implementação de um algoritmo de fila (ou FIFO) na memória, simultaneamente escrevendo dados na fila em uma taxa e lendo-os em outra taxa.

Software de E/S no nível de usuário: