UNICEUB
CIÊNCIAS DA COMPUTAÇÃO

ALBERT PEREIRA, DAVI COSTA, IGOR LIMA, MATHEUS AMORIM, PEDRO SARMENTO, VINÍCIUS CARNEIRO

SISTEMAS OPERACIONAIS MODERNOS

ENTRADA / SAÍDA

BRASÍLIA / DF
2025

ALBERT PEREIRA, DAVI COSTA, IGOR LIMA, MATHEUS AMORIM, PEDRO SARMENTO, VINÍCIUS CARNEIRO

SISTEMAS OPERACIONAIS MODERNOS

ENTRADA / SAÍDA

Trabalho apresentado a Uniceub a fim de elucidar as propriedades e aspectos do gerenciamento de dispositivos E/S (entrada/saída) pelo Sistema Operacional.

Orientadora: Profa. Alessandra Cristina Gomes Magno

BRASÍLIA / DF

2025

INTRODUÇÃO:

O sistema operacional controla todos os dispositivos de E/S (entrada/ saída) de um computador. Ele deve emitir comandos para os dispositivos, interceptar interrupções e tratar os erros; deve também fornecer uma interface entre os dispositivos e o restante do sistema que seja simples e fácil de usar. Na medida do possível, ela deveria ser a mesma para todos os dispositivos (independentemente do dispositivo). O código de E/S representa uma fração significativa de todo o sistema operacional. O modo como o sistema operacional gerencia E/S é o assunto deste trabalho.

Este trabalho é organizado da seguinte forma: primeiro será estudado alguns dos princípios do hardware de E/S e depois o software de E/S em geral. O software de E/S pode ser estruturado em camadas; cada camada apresenta uma tarefa bem definida para executar. Neste trabalho, será visto o que essas camadas fazem e como se relacionam entre si. Posteriormente, será estudado vários dispositivos de E/S em detalhes: discos, relógios, teclados e vídeos. Para cada dispositivo investigar-se-á seu hardware e seu software. Por fim, será tratado do gerenciamento de energia.

1. PRINCÍPIOS DO HARDWARE DE E/S:

1.1 Dispositivos de E/S

Os dispositivos de E/S podem ser divididos de forma genérica em **dispositivos de blocos**, **dispositivos de caractere** e **que não se enquadram.**

Dispositivos de blocos: Armazenam as informações em blocos de tamanho fixo, cada bloco tendo seu próprio endereço. Os tamanhos de blocos podem ir de 512 bytes a 65.536 bytes. Todas as transferências dos dados estão em unidades de um ou mais blocos inteiros. Sua prioridade principal é de que cada bloco pode ser lido ou escrito de maneira independente de outro dispositivos

Exemplos de dispositivos de bloco: discos rígidos(HDDs), CD-ROMs, pen drives, cartões de memória e SSDs...

O disco é um dispositivo endereçável por bloco, pois não importa onde cabeçote de leitura/gravação esteja no momento, é sempre possível posicionar o cabeçote para outro cilindro/disco e, então esperar o bloco requisitado passar sobre o cabeçote para que seja feita a transferência de dados.

Dispositivos de caractere: Enviam ou recebem um fluxo de caracteres, sem considerar qualquer estrutura de blocos. Não é endereçável, não tem operação de busca e não dispõe de qualquer operação de posicionamento.

Exemplos de dispositivos de caractere: mouses, impressoras, interfaces de redes e a maioria dos outros dispositivos que são diferentes do disco, podem ser considerados dispositivos de caractere

Que não se enquadram : Alguns dispositivos não se enquadram como dispositivos de blocos ou de caractere. Os relógios, por exemplo, não são endereçáveis por blocos nem enviam ou recebem fluxos de caracteres. Tudo que eles fazem é causar interrupções em intervalos bem definidos e temporizados para o funcionamento ideal.

1.2 Controladores de dispositivos

Os dispositivos de O/I consistem normalmente em, um componente **mecânico** e um componente **eletrônico**.

O componente eletrônico é conhecido como **controlador do dispositivo** ou **adaptador.** É comum que nos computadores ele se apresente na forma de uma placa controladora de circuito que pode ser inserida em um barramento PCI.

A placa controladora tem em geral, um conector no qual pode ser plugado um cabo que conecta ao dispositivo propriamente dito, a maioria dos controladores são capazes de lidar com dois e até mesmo oito dispositivos idênticos, caso a interface entre o controlador e o dispositivo seja uma interface-padrão. Muitas empresas desenvolvem controladores de disco compatíveis com as interfaces IDE, SATA, SCSI, USB ou *FireWire*.

A interface entre o controlador do dispositivo e o dispositivo é com uma frequência de nível muito baixo, porém o que realmente é entregue pela unidade de disco é um fluxo serial de bits, começando com um **preâmbulo** (Sequência de bits transmitida no início de uma comunicação para preparar o receptor para os dados que virão, permitindo a sincronização entre o transmissor e o receptor), depois 4.096 bits em um setor e por fim, a soma de verificação, também chamada de **código de correção de erro- ECC** (*error-correcting-code*).

O Controlador de um monitor de vídeo de Tubo de Raios Catódicos (CRT) funciona como um dispositivo serial bit a bit de baixo nível. Ele lê bytes da memória contendo caracteres para exibição e gera os sinais para modular o feixe do tubo de raios catódicos (CRT), escrevendo os caracteres na tela. Além disso, ele gera sinais para o retraço horizontal e a varredura de cada linha, bem como para o retraço vertical após a varredura completa da tela. Se o controlador não existisse, o sistema operacional teria que programar explicitamente a varredura analógica. Com o controlador, o sistema operacional apenas o inicializa com parâmetros como o número de caracteres ou pixels por linha e o número de linhas por tela, deixando-o responsável pelo direcionamento do feixe.

Controlador LCD (*liquid crystal display*): Comum nos TFT (*thin film transistor*), ele funciona como um dispositivo serial bit em baixo nível. Ele lê bytes contendo os caracteres a serem exibidos da memória e gera os sinais para modificar a polarização da luz de fundo para os pixels correspondentes, a fim de escrevê-los na tela.

A presença do controlador significa que o programador do SO não precisa programar explicitamente o campo elétrico da tela!

De forma resumida as tarefas do controlador são: Converter o fluxo serial de bits em blocos de bytes, que são montados bit a bit em um buffer dentro do controlador, fazer toda correção necessária com o ECC, tornar os blocos disponíveis para serem copiados na memória principal.

1.3 E/S Mapeada na 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, ligar ou desligar, ou executar alguma tarefa. Muitos dispositivos também possuem um buffer de dados, como a RAM de vídeo, que o sistema operacional pode usar para leitura e escrita. Isso permite, por exemplo, exibir pixels na tela.

A CPU se comunica com os registradores dos controladores e os buffers de dados dos dispositivos de duas maneiras:

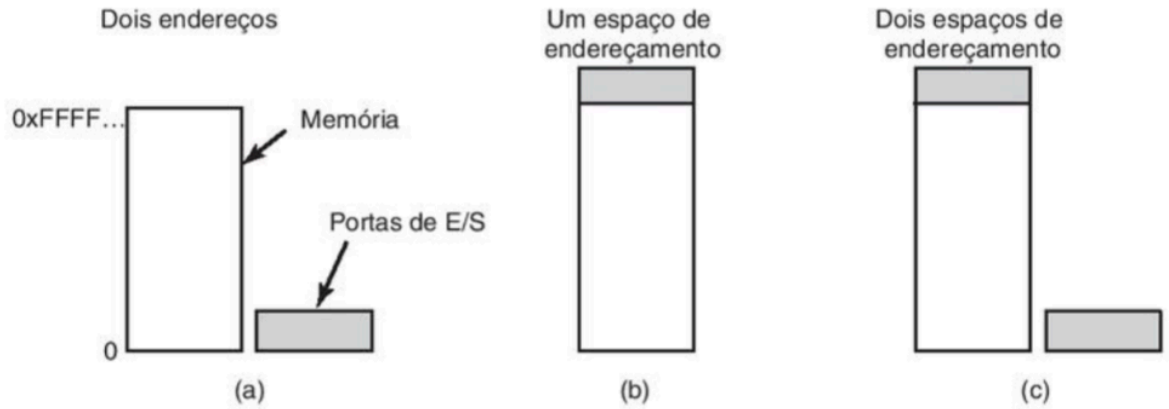
1 - Portas de E/S: Cada registrador de controle é associado a um número de porta de E/S (8 a 64 bits).O conjunto de todas as portas de E/S formam **o espaço de portas E/S** e somente o sistema operacional usa instruções especiais de E/S para acessar essas portas, que são protegidas contra programas de usuário. Usando uma instrução especial de E/S como: OUT PORT, REG a CPU pode escrever os conteúdos de REG para o registrador de controle PORT. A maioria dos computadores de grande porte como IBM 360 e todos os seus sucessores funcionavam assim de acordo com a Figura 1(a).

2 - O segundo método visa mapear todos os registradores de controle no espaço de endereçamento da memória: **E/S Mapeada na Memória**; Figura 1 (b). Essa técnica, exemplificada pelo computador PDP-11, consiste em tratar os registradores de controle dos dispositivos como se fossem posições de memória RAM. Cada registrador recebe um endereço de memória único, que não é usado pela memória RAM propriamente dita. Isso permite que a CPU utilize as mesmas instruções que usa para acessar a memória para interagir com os dispositivos, simplificando o processo. Geralmente, esses endereços de memória reservados para E/S mapeada estão localizados nas posições mais altas do espaço de endereçamento da memória.

Esquema Híbrido; figura 1(c). Essa abordagem combina a E/S mapeada na memória com as portas de E/S tradicionais. Os buffers de dados dos dispositivos são mapeados na memória, enquanto os registradores de controle são acessados por meio de portas de E/S separadas. O processador Pentium utiliza essa arquitetura híbrida. Nos PCs compatíveis com IBM, os endereços de 640 KB a 1 MB

- 1 são reservados para os buffers de dados dos dispositivos, e as portas de E/S de 0 a 64 KB - 1 são usadas para os registradores de controle.

Figura 1:



(a) Espaços de memória e E/S separados. (b) E/S mapeada na memória. (c) Híbrido.

Esses esquemas funcionam em todos os casos quando a CPU quer ler uma palavra - ou da memória ou de uma porta de E/S -, ela coloca o endereço de que precisa nas linhas do barramento e então emite um sinal READ sobre uma linha de controle de barramento.

O texto descreve como a CPU se comunica com dispositivos usando E/S (Entrada/Saída). Ele explica que existe uma linha de sinal para diferenciar entre acesso à memória e acesso a dispositivos de E/S. Quando a CPU precisa acessar um dispositivo, ela usa um endereço específico para se comunicar com ele. A E/S mapeada na memória é uma técnica que permite que a CPU trate os registradores de controle dos dispositivos como se fossem posições de memória, facilitando a comunicação.

Pontos Fortes da E/S Mapeada na Memória:

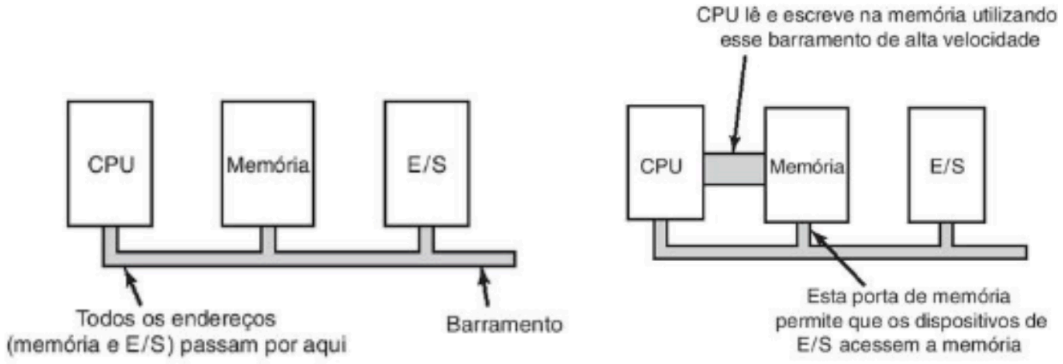
- Simplificação da programação:
  - Permite que os drivers de dispositivos sejam escritos em linguagens de alto nível como C, sem a necessidade de código assembly complexo.
  - Os registradores de controle são tratados como variáveis de memória, facilitando o acesso e a manipulação.

- Eficiência:
  - Elimina a necessidade de instruções especiais de E/S (IN/OUT), que podem adicionar overhead ao processo de comunicação.
  - A CPU pode usar as mesmas instruções que usa para acessar a memória, tornando a comunicação mais rápida.

Pontos Fracos da E/S Mapeada na Memória:

- Conflitos de endereçamento:
  - É crucial garantir que os endereços de memória usados para E/S não entrem em conflito com os endereços usados pela memória RAM.
  - Uma má configuração pode levar a erros e instabilidade do sistema.
- Complexidade do hardware:
  - O hardware para a lógica de endereçamento do módulo de E/S é mais complexo, devido ao tamanho do endereço maior.

Figura 2:



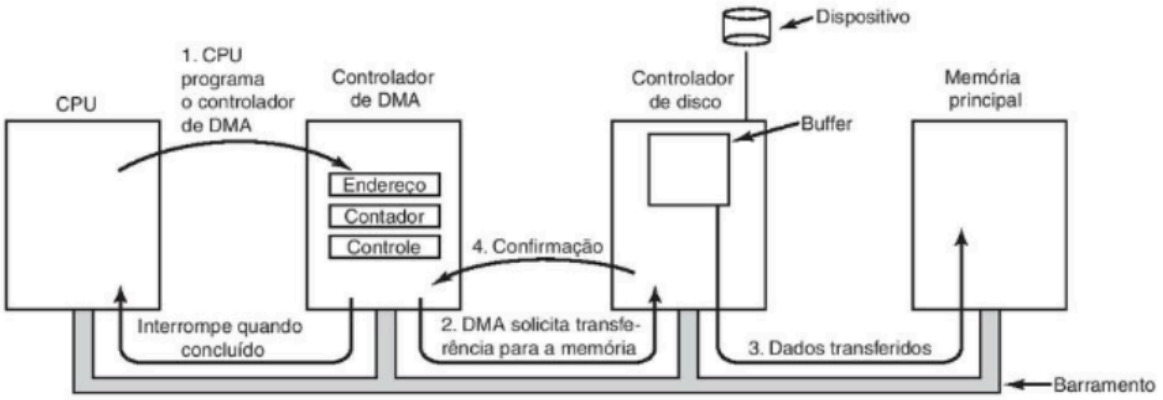
a) Arquitetura com barramento único b) Arquitetura de memória com barramento duplo.

1.4 Acesso direto à memória (DMA)

Não importa se a CPU tem ou não E/S mapeada na memória, ela precisa endereçar os controladores dos dispositivos para poder trocar dados com eles. Em vez de a CPU solicitar dados byte por byte, o que consome muito tempo, o DMA permite que um **controlador de DMA** transfira dados diretamente entre a memória e os dispositivos, sem a intervenção constante da CPU. A maioria dos sistemas possui um controlador de DMA, que pode ser integrado a outros controladores ou ser um

componente separado na placa-mãe. Esse controlador tem acesso independente ao barramento do sistema e possui registradores que a CPU pode configurar para especificar detalhes da transferência, como o endereço de memória, a quantidade de bytes, a direção da transferência e a porta de E/S envolvida. O DMA otimiza o uso da CPU, permitindo que ela realize outras tarefas enquanto as transferências de dados ocorrem em segundo plano. Não importa onde ele está fisicamente localizado, o controlador de DMA tem acesso ao barramento do sistema independente da CPU como mostrado na figura abaixo (Figura 3).

Figura 3:



Funcionamento da DMA:

- Primeiro, a CPU programa o controlador DMA configurando seus registradores para que ele saiba o que transferir para e para onde transferir.
- Paralelamente, o DMAC emite um comando para o controlador de disco dizendo para ele ler dados do disco em seu buffer interno e verificar a soma de verificação. Quando dados válidos estão no buffer do controlador de disco, o DMA pode começar.
- O controlador DMA inicia a transferência emitindo uma solicitação de leitura pelo barramento para o controlador de disco
- A gravação na memória é outro ciclo de barramento padrão
- Quando a gravação é concluída, o controlador de disco envia um sinal de confirmação ao controlador DMA, também pelo barramento
- O controlador DMA então incrementa o endereço de memória a ser usado e diminui a contagem de bytes. Se a contagem de bytes ainda for maior que

0, as etapas 2 a 4 são repetidas até que a contagem chegue a 0. Nesse momento, o controlador DMA interrompe a CPU para informá-la de que a transferência está completa.

Os controladores DMA podem variar bastante em complexidade. Os mais básicos realizam uma transferência por vez, enquanto os mais avançados (DMACs) possuem múltiplos conjuntos de registradores para gerenciar diversos canais simultaneamente. A transferência de dados em controladores DMA sofisticados pode ser organizada de diferentes maneiras: usando um sistema de rodízio (round-robin) ou um esquema de prioridades, onde alguns dispositivos têm preferência sobre outros.

Existem dois modos principais de operação para muitos barramentos e alguns controladores DMA:

- **Modo palavra por vez (roubo de ciclo):** O controlador DMA solicita e obtém a transferência de uma palavra de cada vez. Se a CPU precisar usar o barramento, ela precisa esperar. Isso é chamado de "roubo de ciclo" porque o controlador DMA "rouba" pequenos períodos de uso do barramento da CPU, causando um pequeno atraso.

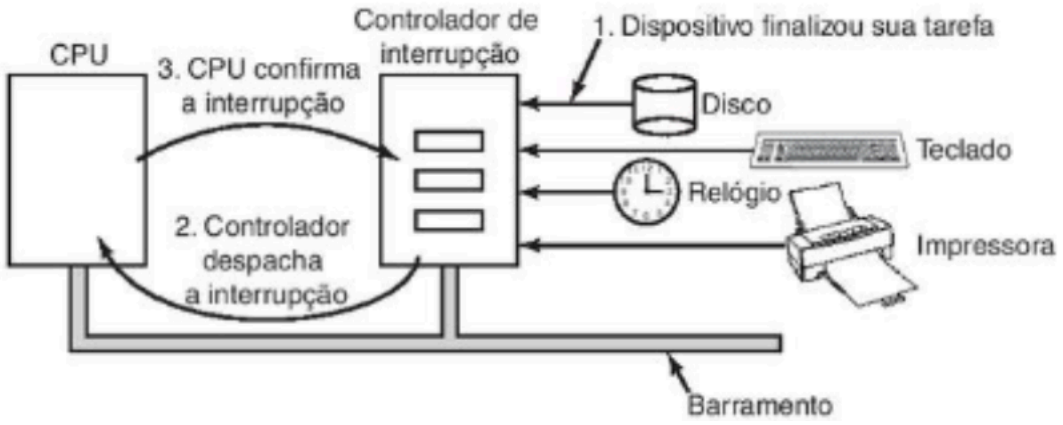
- **Modo bloco (burst mode):** O controlador DMA instrui o dispositivo a assumir o controle do barramento, realizar uma sequência de transferências e, em seguida, liberar o barramento. Esse modo é mais eficiente porque o tempo gasto para obter o controle do barramento é aproveitado para transferir várias palavras de uma vez. A desvantagem do modo bloco é que ele pode impedir que a CPU e outros dispositivos usem o barramento por um período considerável se uma grande quantidade de dados estiver sendo transferida de forma contínua.

1.5 Interrupções revisitadas

Quando um dispositivo de E/S termina o trabalho que lhe foi dado, ele causa uma interrupção ao afirmar um sinal em uma linha de barramento que lhe foi atribuída, sinais que são detectados pelo chip controlador de interrupção. Se nenhuma outra interrupção estiver pendente, o controlador de interrupção processa

a interrupção imediatamente. Se outra interrupção estiver em andamento ou houver uma solicitação simultânea em uma linha de solicitação de interrupção de prioridade mais alta que ele continua a afirmar até ser atendido pela CPU. O controlador coloca um número nas linhas de endereço e afirma um sinal que interrompe a CPU. Este número é usado como um índice em uma tabela chamada **vetor de interrupção** para iniciar um procedimento de serviço de interrupção correspondente. O procedimento de serviço em determinado momento reconhece a interrupção enviando algum valor para alguma porta do controlador que permite que o controlador emita outras interrupções.

Figura 4:



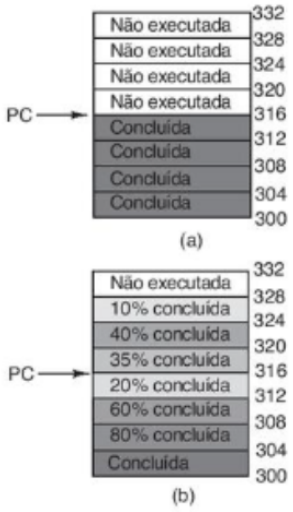
Existem dois tipos de interrupção: a **interrupção precisa** e a **interrupção imprecisa.**

Interrupção precisa : Uma interrupção que deixa a máquina em um estado bem definido é chamada de interrupção precisa (Walker e Cragon, 1995). Tal interrupção tem quatro propriedades:

- O PC (Contador de Programa) é salvo em um local conhecido;
- Todas as instruções anteriores àquela apontada pelo PC foram concluídas;
- Nenhuma instrução além daquela apontada pelo PC foi concluída;
- O estado de execução da instrução apontada pelo PC é conhecido.

Interrupção imprecisa : Uma interrupção que não atende a esses requisitos é chamada de interrupção imprecisa e torna a vida muito desagradável para o criador do sistema operacional, que agora precisa descobrir o que aconteceu e o que ainda precisa acontecer. Máquinas com interrupções imprecisas geralmente vomitam uma grande quantidade de estado interno na pilha para dar ao sistema operacional uma chance de descobrir o que está acontecendo. Interrupções imprecisas permitem que mais espaço da CPU seja usado para cache, etc., mas tornam o sistema operacional mais complexo. Na imagem a seguir (Figura 5) é possível comparar os dois tipos de interrupções;

Figura 5:



2. PRINCÍPIOS DO SOFTWARE DE E/S:

2.1 Objetivos do software de E/S

Os princípios do software de Entrada e Saída (E/S) são fundamentais para garantir que os sistemas computacionais interajam de maneira eficiente e segura com dispositivos de hardware, como teclados, mouses, discos rígidos e redes. Aqui estão alguns dos principais conceitos:

. Independência do dispositivo: Um conceito primordial no projeto de software de E/S é conhecido como independência do dispositivo. Esse conceito

propõe que deveria ser possível escrever programas aptos a acessar qualquer dispositivo de E/S sem a necessidade de especificar antecipadamente o dispositivo. Por exemplo, um programa que lê um arquivo como entrada deveria ser capaz de ler um arquivo de um disco rígido, deu um CD-ROM, de um DVD ou de um dispositivo USB sem ter de modificar o programa para cada dispositivo diferente

. **Nomeação uniforme:** Estreitamente relacionado à independência do dispositivo está o objetivo da nomeação uniforme. O dispositivo tem o objetivo da nomeação uniforme. O nome de um arquivo ou de um dispositivo deveria simplesmente ser uma cadeia de caracteres ou um número inteiro totalmente independente do dispositivo. Por exemplo, um dispositivo USB pode ser montado no topo do diretório /usr/ast/backup, de maneira que copiar um arquivo para o subdiretório /usr/ast/backup/monday significa copiar o arquivo para o dispositivo USB. Assim, todos os arquivos e dispositivos são endereçados do mesmo modo: pelo nome do caminho. Outra questão importante para os programas de E/S é o tratamento de erros. Em geral, os erros deveriam ser tratados o mais próximo possível do hardware. Se o controlador descobre um erro de leitura, ele deveria tentar corrigi-lo por si próprio. Se ele não tem condições de fazê-lo, então o driver do dispositivo deveria tratar disso, talvez simplesmente tentando ler de novo o bloco. Muitos erros são transitórios, como erros de leitura causados por partículas de pó sobre o cabeçote de leitura e que frequentemente não ocorrem nas leituras seguintes. Somente se as camadas nas faixas não fossem capazes de tratar o problema é que as camadas superiores deveriam ser informadas sobre ele. Em muitos casos, a recuperação de um erro pode ser feita com transparência em um baixo nível sem que os níveis superiores tomem conhecimento desse erro.

. **Tino de transferência:** Uma outra questão ainda primordial é o tino de transferência, que pode ser síncrona (bloqueante) ou assíncrona (orientada à interrupção). A maioria das E/S físicas é assíncrona - a CPU inicializa a transferência e segue fazendo outra atividade até receber uma interrupção. Os programas do usuário são muito mais fáceis de escrever quando as operações de E/S são bloqueadas após uma chamada de sistema read o programa é automaticamente suspenso até que os dados estejam disponíveis no buffer. Outra questão para os programas de E/S é a utilização de buffer para armazenamento temporário. Muitas vezes, os dados provenientes de um dispositivo não podem ser armazenados diretamente em seu destino. Alguns dispositivos apresentam restrições severas de

tempo real (por exemplo, dispositivos de áudio digital), de modo que os dados devem ser antecipadamente colocados em um buffer de saída para separar a taxa com a qual o buffer é preenchido da taxa com a qual ele é esvaziado, a fim de evitar seu completo esvaziamento. A utilização de buffer para armazenamento temporário envolve muitas operações de cópia e com frequência gera um impacto significativo no desempenho da E/S.

. Dispositivos compartilhados versus dedicados: O conceito final aqui é o de dispositivos compartilhados versus dedicados. Alguns dispositivos de E/S, como discos, podem ser usados por muitos usuários ao mesmo tempo. O fato de vários usuários terem arquivos abertos simultaneamente no mesmo disco não acarreta qualquer problema. Outros dispositivos, como unidades de fita, devem ser dedicados a um único usuário até que este finalize suas operações. Então, outro usuário pode usar a unidade de fita. Definitivamente, não é viável ter dois ou mais usuários escrevendo blocos aleatoriamente e de maneira intercalada na mesma fita. O uso de dispositivos dedicados (não compartilhados) também gera uma série de problemas, como os impasses. Novamente, o sistema operacional deve ser capaz de tratar tanto os dispositivos compartilhados quanto os dedicados de uma maneira que evite problemas.

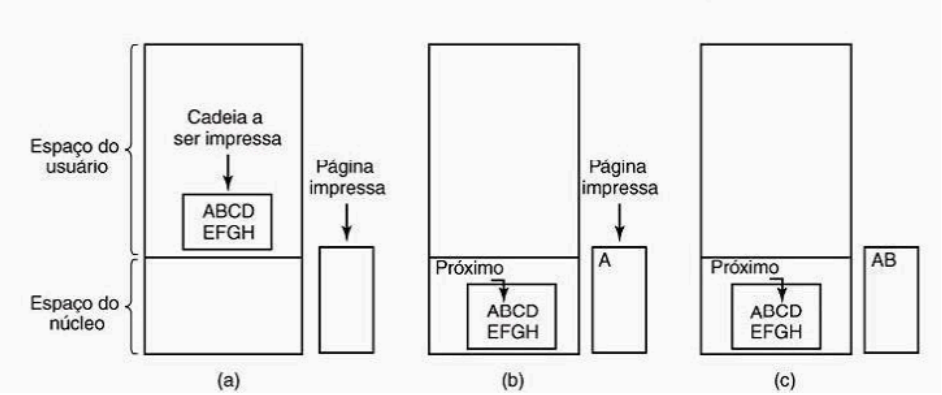
2.2 E/S programada

Existem três diferentes maneiras fundamentais de realizar E/S. Nesta seção será examinado a primeira delas, a **E/S programada**. Nas próximas duas seções serão examinadas as outras **E/S orientadas à interrupção** e a **E/S que usa DMA**. A forma mais simples de E/S é ter a CPU fazendo todo o trabalho. Esse método é chamado de E/S programada.

A E/S programada é simples mas tem a desvantagem de segurar a CPU o tempo todo até que a E/S seja feita. Se o tempo para 'imprimir' um caractere for muito curto (pois tudo o que a impressora está fazendo é copiar o novo caractere para um buffer interno), então a espera ociosa será conveniente Além disso, em um sistema embarcado no qual a CPU não tem nada mais para fazer, a espera ociosa é razoável. Contudo, em sistemas mais complexos, em que a CPU tem outro trabalho a realizar, a espera ociosa é ineficiente. Faz-se necessário um melhor método de E/S.

Esse método é muito simples de ilustrar com um exemplo. Considere um processo de usuário que quer imprimir a cadeia de caracteres de oito caracteres 'ABCDEFGH' na impressora. Primeiro ele monta a cadeia de caracteres em um buffer no espaço do usuário, como mostrado na imagem abaixo (Figura 6), o usuário requisita então a impressora para escrita por meio de uma chamada de sistema para abri-la. Se a impressora está sendo usada por outro processo, essa chamada falha e retorna um código de erro ou um bloqueio até que a impressora esteja disponível, dependendo do sistema operacional e dos parâmetros da chamada. Uma vez que tenha a impressora, o processo do usuário efetuará uma chamada de sistema para imprimir a cadeia de caracteres na impressora.

Figura 6:



Normalmente, o sistema operacional então copia o buffer com a cadeia de caracteres para um vetor- considere, no espaço do núcleo, onde ele é mais facilmente acessado (pois o núcleo pode precisar trocar o mapa da memória para acessar o espaço do usuário). Ele então verifica se a impressora está disponível no momento; em caso negativo, ele espera até que ela esteja. Logo que a impressora fica disponível o sistema operacional copia o primeiro caractere para o registrador de dados da impressora - no exemplo dado, isso é feito mediante o uso da E/S mapeada na memória. Essa ação ativa a impressora. O caractere pode não aparecer ainda porque algumas impressoras armazenam uma linha ou uma página antes de imprimir qualquer coisa.

2.3 E/S usando interrupção

A **E/S por interrupção** representa um avanço significativo na comunicação entre a CPU e os dispositivos de entrada e saída, tornando o sistema mais eficiente

e permitindo um melhor aproveitamento dos recursos do computador. Diferente da **E/S programada**, onde a CPU precisa verificar constantemente se o dispositivo está pronto para realizar uma operação, a interrupção permite que a CPU execute outras tarefas enquanto aguarda um sinal do dispositivo. Esse mecanismo melhora o desempenho geral do sistema, pois evita desperdício de tempo de processamento em verificações desnecessárias. Quando um dispositivo precisa de atenção, ele envia uma **interrupção** à CPU, que pausa temporariamente sua execução atual para processar a solicitação do dispositivo. Depois de atender à interrupção, a CPU retorna exatamente ao ponto onde havia parado, garantindo que a execução dos programas não seja prejudicada. Isso é essencial para sistemas modernos, onde múltiplos dispositivos podem estar operando simultaneamente e exigindo a atenção do processador em momentos diferentes.

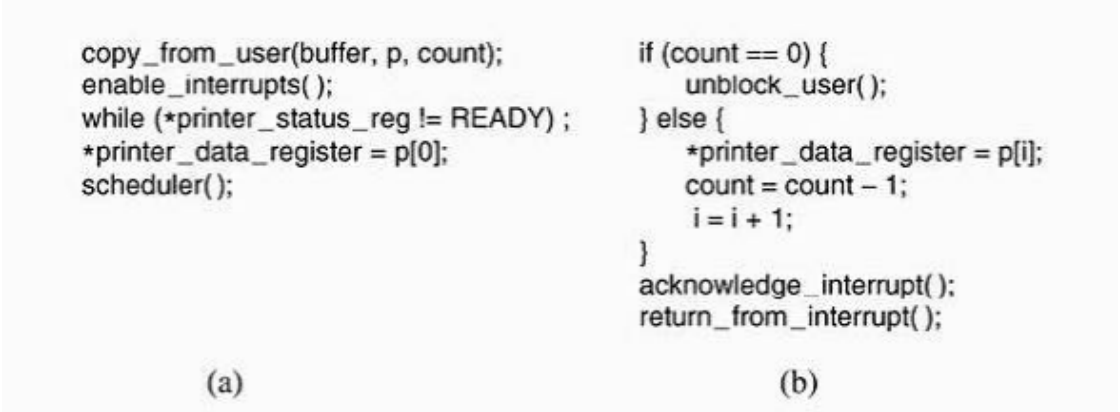
Além disso, esse método de E/S é amplamente utilizado em dispositivos como teclados, discos rígidos, placas de rede e outros periféricos, garantindo uma resposta mais rápida e eficiente às solicitações de hardware. No entanto, embora a E/S por interrupção seja mais vantajosa que a E/S programada, ela também pode gerar desafios, como a necessidade de um **mecanismo eficiente de gerenciamento de interrupções** para evitar sobrecarga da CPU quando muitas interrupções ocorrem ao mesmo tempo.

Agora, tomando como exemplo ainda a Figura 6 que foi vista acima, considere o caso da impressão em uma impressora que não armazena os caracteres, mas imprime-os um a um à medida que chegam. Se a impressora pode imprimir cem caracteres por segundo, cada caractere leva 10 ms para ser impresso. Isso significa que, após cada caractere ser escrito no registrador de dados da impressora, a CPU permanece em um laço ocioso durante 10 ms esperando a permissão para a saída do próximo caractere. Isso é mais do que o tempo necessário para fazer um chaveamento de contexto e executar algum outro processo durante os 10 ms que de outra maneira seriam perdidos.

Outro modo de permitir que a CPU faça outra coisa enquanto espera a impressão tornar-se pronta é usar interrupções. Quando é feita uma chamada de sistema para a impressão de uma cadeia de caracteres, o buffer é copiado para o espaço do núcleo do sistema operacional, como mostrado anteriormente, e o primeiro caractere é copiado para a impressora tão logo ela concorde em aceitá-lo.

Nesse ponto, a CPU chama o escalonador e outro processo é executado. O processo que solicitou a impressão da cadeia de caracteres é bloqueado até que toda a cadeia seja impressa. O trabalho feito durante a chamada de sistema é mostrado na Figura 7 abaixo.

Figura 7:



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

(a)

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

(b)

2.4 E/S usando DMA

Uma desvantagem óbvia do mecanismo de E/S orientada à interrupção é a ocorrência de uma interrupção para cada caractere. Interrupções levam tempo, de modo que esse esquema desperdiça uma certa quantidade de tempo de CPU. Uma solução é usar o acesso direto à memória (DMA). A ideia é fazer o controlador de DMA alimentar os caracteres para a impressora um por vez, sem que a CPU seja perturbada. Na verdade, o DMA executa E/S programada, em que somente o controlador de DMA faz todo o trabalho, em vez da CPU principal. Essa estratégia requer hardware especial (o controlador de DMA).

A grande vantagem do DMA é reduzir o número de interrupções de uma por caractere para uma por buffer impresso. Se existem muitos caracteres e as interrupções são lentas, esse sistema pode significar uma melhoria substancial. Por outro lado, o controlador de DMA é, em geral, muito mais lento do que a CPU principal. Se o controlador de DMA não é capaz de dirigir o dispositivo em velocidade máxima ou a CPU não tem nada para fazer enquanto espera pela interrupção do DMA, então a E/S orientada à interrupção E/S programada podem ser mais vantajosas. Na maior parte das vezes, o DMA vale a pena.

3. CAMADAS DO SOFTWARE DE E/S:

3.1 Tratadores de interrupção

. E/S programada: A CPU fica esperando o dispositivo terminar a operação, tornando isso ineficiente.

. Interrupções: O dispositivo avisa a CPU quando está pronto, liberando-a para outras tarefas. O que acaba induzindo complexidade no sistema operacional (SO).

O SO tem como objetivo esconder a complexidade das interrupções dos programas tratando elas no núcleo (*kernel*):

Primeiro bloqueando o driver que iniciou a E/S até a interrupção sinalizar a conclusão. Desbloquear o driver quando a interrupção ocorrer (permitindo que ele execute a operação novamente), usando: Semáforos (*up/down*), Variáveis de condição (*signal/wait*), Troca de mensagens.

Ex: **Driver de disco** - Ele inicia uma leitura e se bloqueia (*down* (semáforo)), assim que terminar gera uma interrupção, o tratador de interrupção processa os dados do disco e libera o driver (*up*(semáforo)) e o driver, desbloqueado, continua a execução.

. Passos do Tratamento de Interrupções:

1. **Salva o estado atual:** Registradores (incluindo a PSW) que não foram salvos pelo hardware de interrupção
2. **Prepara o ambiente para o tratador:** Configura a pilha, memória MMU/TLB, e a tabela de páginas
3. **Sinaliza o controlador de interrupções:** Avisa o hardware que a interrupção já está sendo tratada para evitar *retriggering* (reativação do processo).
4. **Executa o tratado específico:** Lê os dados dos registradores do controlador do dispositivo que está sendo interrompido.
5. **Escolhe o próximo processo:** Caso a interrupção tenha deixado um processo de alta prioridade anteriormente bloqueado, este pode ser escolhido para executar agora.
6. **Restaura o contexto:** Recarrega registradores, MMU e TLB para o próximo processo.
7. **Retoma a execução:** Continua o processo escolhido.

. Desvantagens:

- **Overhead:** Salvar/restaurar registradores consome ciclos da CPU
- **Concorrência:** O sistema deve garantir a atomicidade nas interrupções (desabilitar interrupções temporariamente).
- **Hardwares específicos:** Alguns hardwares exigem passos adicionais (como limpar caches).

. Importância:

- A CPU não fica ociosa esperando E/S.
- Múltiplos dispositivos operam em paralelo sem interferência.
- Programas de usuário não precisam lidar diretamente com interrupções.

. Comparação prática:

1. Garçom anota seu pedido (E/S iniciada);
2. Bloqueia sua mesa (semáforo);
3. Volta quando a comida está pronta (interrupção);
4. Libera você para comer (continuação do processo).

3.2 Drivers dos dispositivos

. Drivers: Programas específicos que permitem o Sistema Operacional se comunicar com hardware (dispositivos como impressora, mouse, etc.).

Normalmente são rodados no modo Kernel (Núcleo do SO) o que dá a eles acesso direto ao hardware, pois assim desempenham em alta potência, mas em contrapartida, caso falhem podem travar o sistema. Alguns drivers rodam no modo Usuário (como no MINIX 3) como processos comuns isolados do núcleo, garantindo mais segurança, porém menor desempenho.

Existem 2 tipos de drivers:

1. Para dispositivos de bloco, que são dispositivos feitos para acessarem dados em blocos como os HDs.
2. Dispositivos de caracteres, que trabalham com fluxos de caracteres, como os teclados.

. Funcionamento do Driver: Inicia verificando se o dispositivo está pronto, converte soluções abstratas (ler arquivo) em comandos físicos para o hardware (ler setor X do disco), envia comandos para os registradores do dispositivo e espera a

conclusão do processo (pode bloqueá-lo ou pode ser interrompido) e depois verifica se a operação foi bem-sucedida e informa o status aos sistema.

. Desafios:

- **Reentrância:** Um driver pode ser chamado antes de terminar a operação atual (chegada de novos pacotes de rede durante o processamento);
- **Hot Plug:** Caso um dispositivo seja removido durante uma operação, o driver deve abortá-la sem corromper o sistema.
- **Alocar Recursos:** Alguns dispositivos novos exigem ajustes dinâmicos (como as linhas de interrupção).
- **Evolução:** Antigamente os drivers eram compilados diretamente no núcleo do Sistema Operacional, já atualmente eles são carregados dinamicamente o que permite adicionar ou remover dispositivos sem recompilar o Sistema Operacional.

3.3 Software de E/S independente de dispositivo

É parte do SO que padroniza a comunicação entre os dispositivos (como discos, teclados, etc.) e o resto do sistema, sem precisar de detalhes específicos dos dispositivos, agindo como uma ponte entre os drivers de dispositivos (específicos para cada hardware) e o software de nível do usuário, o que garante eficiência e simplicidade.

. Funções:

- **Interface uniforme para drivers:** Um ponto importante para o Sistema Operacional (SO) é fazer os dispositivos E/S e os drivers de dispositivos parecerem os mesmos, se cada dispositivo possuir uma interface diferente toda vez que um novo dispositivo aparece o SO deve ser modificado para esse novo dispositivo, o que não é uma boa estratégia. Então a Interface uniforme para drivers serve para padronizar como os drivers se comunicam com o SO, evitando que o dispositivo exija modificações no SO, assim como todos os drivers de disco usam a mesma estrutura de funções (leitura, escrita e formatação).
- **Utilização de buffer:** É uma questão importante tanto para dispositivos de bloco quanto para dispositivos de caracteres. Por exemplo,

considere um processo que quer ler dados de um modem, sem uso de buffer cada caractere enviado causa uma interrupção para esperar um próximo chegar e mandá-lo ao usuário, o que faz com que o processo do usuário seja iniciado a cada caractere recebido sendo ineficiente. Um modo de melhorar é quando o processo do usuário fornece um buffer que por meio do envio de caracteres é preenchido e só depois disso ele acorda o processo do usuário para recebê-los. Mas ainda sim é possível melhorar com o uso de um buffer duplicado, pois caso um deles fique cheio o outro armazena dados enquanto o primeiro envia. Ainda existem o buffer circular, que é formado por dois ponteiros, um aponta para a próxima palavra livre e o outro para a primeira que ainda não foi removida, assim criando um ciclo que é bem mais eficiente evitando que os espaços fiquem cheios e atraem o envio de caracteres.

- **Tratamento de erros:** Erros são bem comuns durante uma E/S. Quando eles ocorrem o SO deve lidar com eles da melhor forma possível. Os erros específicos do dispositivo (como um disco danificado) serão tratados pelo driver, os erros genéricos (processo tenta escrever em um dispositivo de entrada) serão reportados pelo usuário e em casos graves (como corrupção de estruturas importantes) podem precisar do desligamento do sistema.
- **Alocação de dispositivos dedicados:** Dispositivos como os gravadores CD-ROM são alocados para um único processo por vez e o SO deve ser capaz de verificar as requisições do dispositivo. São usadas as chamadas (*open/close*) para o controle de acesso e a fila de processos bloqueados esperam pela liberação do dispositivo.
- **Tamanho de bloco independente:** Serve para uniformizar o tamanho de blocos lógicos, mesmo que os dispositivos físicos usem setores diferentes, por exemplo um disco com setores de 512 bytes pode ser acessado pelo SO como blocos de 4KB.

3.4 Software de E/S do espaço do usuário

É a parte do sistema de E/S que opera fora do núcleo (kernel) do SO, executando diretamente nos programas de usuário. Incluindo bibliotecas de E/S e

sistemas de spooling, simplificando a interação com dispositivos e melhorando a eficiência.

. Bibliotecas de E/S: Fornecem rotinas que programas de usuário podem chamar para realizar operações de E/S como o *printf* (saída) e o *scanf* (entrada) em C. Elas são vantajosas pois escondem complexidades e permitem que os programas funcionem em diferentes sistemas.

. Spooling: É uma técnica para gerenciar dispositivos dedicados (como uma impressora) em sistemas multitarefa, evitando a monopolização de um dispositivo. Um processo especial (*daemon*) controla o acesso ao dispositivo, os arquivos a serem impressos são colocados em um diretório de *spool* (fila) e o *daemon* imprime um por um, liberando o dispositivo para outros usuários, por exemplo, pode evitar que arquivos grandes demais travem a impressora. É vantajoso pois os dispositivos que são lentos não bloqueiam os processos e não permite que usuários monopolízem o dispositivo.

. Fluxo completo de E/S: O programa do usuário chama uma rotina de biblioteca, essa biblioteca formata os dados e chama um *syscall*, o SO verifica os buffers em memória, caso os dados não estejam disponíveis chama o driver do dispositivo, o dispositivo executa o operação e gera uma interrupção ao concluir e o tratador de interrupção acorda o processo que foi bloqueado e retorna os dados.

4. DISCOS

4.1 Hardware do disco

Existem uma grande variedade de discos, como o **magnético** (disco rígido e flexível), que é caracterizado devido à sua velocidade tanto para leitura quanto para escrita, tornando - os ideias para memórias não - voláteis (paginação, sistemas de arquivos e etc.) arranjo desses discos servem para fornecer armazenamento altamente confiável. E também há o **óptico,** que é mais utilizado para distribuição de programas, dados e filmes, vários tipos de discos ópticos (CD - ROMs, CDs graváveis e DVDs) também são importantes.

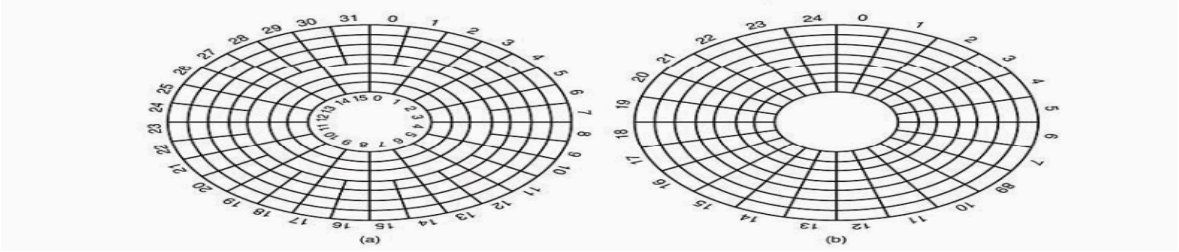
4.2 Discos magnéticos

Os discos magnéticos são constituídos por: **Pratos** (Discos metálicos giratórios), **Cabeçote** (“Agulha” que lê e escreve dados no prato), **Trilha** (Círculo concêntrico no prato), **Cilindro** (Conjunto de trilhas alinhadas verticalmente em

todos os pratos), **Setores** (Pedaços menores de uma trilha). Nos discos flexíveis o setor vai de 8 a 32 e nos discos rígidos até várias centenas de milhares. O controlador é aquele que comanda o disco, leitura e gravação. Geralmente é responsável por controlar o cache, remapear os blocos defeituosos e outros.

Uma característica do dispositivo é a possibilidade do controlador fazer o *overlapped seeks* que trata-se da possibilidade de o controlador fazer posicionamentos simultâneos em duas ou mais unidades de discos, enquanto o controlador e o software estão esperando pela finalização de um posicionamento em um disco, o controlador pode começar um posicionamento em outro disco. Pode-se observar que as especificações dos discos rígidos modernos usadas pelo driver, são diferentes do formato físico. Os discos modernos são divididos em zonas, das quais há mais setores nas zonas externas do que nas internas, devido ao formato circular. Na figura 8 (a), mostra como seria a geometria do disco fisicamente, já a figura 8 (b) trata-se da geometria virtual criada pelo driver, em ambos os discos há 192 setores, mas as organizações são diferentes. O software age como se existisse x cilindros, y cabeçotes e z setores, assim fazendo os cálculos e criando uma geometria virtual, que mais se assemelha com a figura “b”.

Figura 8:



Usando esses parâmetros e considerando 512 bytes por setor, o maior disco possível é 31,5 GB. Para superar esse limite muitos discos suportam um sistema chamado de endereçamento lógico, na qual os setores do disco são simplesmente numerados a partir do 0, sem considerar a geometria do disco.

4.3 RAID

RAID foi definido, por Patterson, como arranjo redundante de discos baratos (*Redundant Array of Inexpensive Disks*), depois a indústria foi e trocou o *Inexpensive* por *Independent.* Foi criado na necessidade de melhorar o desempenho do disco. A ideia fundamental por trás de um RAID é instalar uma caixa cheia de discos junto ao computador, em geral um grande servidor, substituir a placa controladora de disco

com um controlador RAID, copiar os dados para o RAID e então continuar a operação normal. Hoje, muitos fabricantes também oferecem RAIDs (mais baratos) baseados no SATA. Dessa maneira, nenhuma mudança no software é necessária para usar o RAID, um grande atrativo para muitos administradores de sistemas.

Além de se parecer como um disco único para o software, todos os RAIDs têm a propriedade de que os dados são distribuídos pelos dispositivos, a fim de permitir a operação em paralelo. Hoje, a maioria dos fabricantes refere-se às sete configurações padrão com RAID nível 0 a RAID nível 6. Além deles, existem alguns outros níveis secundários que não serão discutidos. O termo “nível” é de certa maneira equivocado, pois nenhuma hierarquia está envolvida; simplesmente há sete organizações diferentes possíveis.

- A organização do RAID nível 0 grava faixas consecutivas nos discos em um estilo de alternância circular (*round-robin*), como descrito na Figura 9 (a) para um RAID com quatro discos. Essa distribuição de dados por meio de múltiplos discos é chamada de ***striping***. Por exemplo, se o software emitir um comando para ler um bloco de dados consistindo em quatro faixas consecutivas começando no limite de uma faixa, o controlador de RAID dividirá esse comando em quatro comandos separados, um para cada um dos quatro discos, e os fará operar em paralelo. Desse modo, tem-se E/S paralela sem que o software saiba a respeito.
- RAID nível 1, mostrada na Figura 9 (b), duplica todos os discos, portanto há quatro discos primários e quatro de backup. Em uma escrita, cada faixa é escrita duas vezes. Em uma leitura, cada cópia pode ser usada, distribuindo a carga através de mais discos. Em consequência, o desempenho de escrita não é melhor do que para um disco único, mas o desempenho de leitura pode ser duas vezes melhor. A tolerância a falhas é excelente: se um disco quebrar, a cópia é simplesmente usada em seu lugar. A recuperação consiste em apenas instalar um disco novo e copiar o backup inteiro para ele.
- Diferentemente dos níveis 0 e 1, que trabalham com faixas de setores, o RAID nível 2 trabalha com palavras, talvez mesmo com bytes. Imagine dividir cada byte de um único disco virtual em um par de pedaços de 4 bits cada, então acrescentar um código *Hamming* para cada um para formar uma palavra de 7 bits, das quais os bits 1, 2 e 4 são de paridade. Imagine ainda que os sete discos da Figura 9 (c), foram sincronizados em termos de posição do braço e posição rotacional. Então seria possível escrever a palavra de 7 bits codificada com

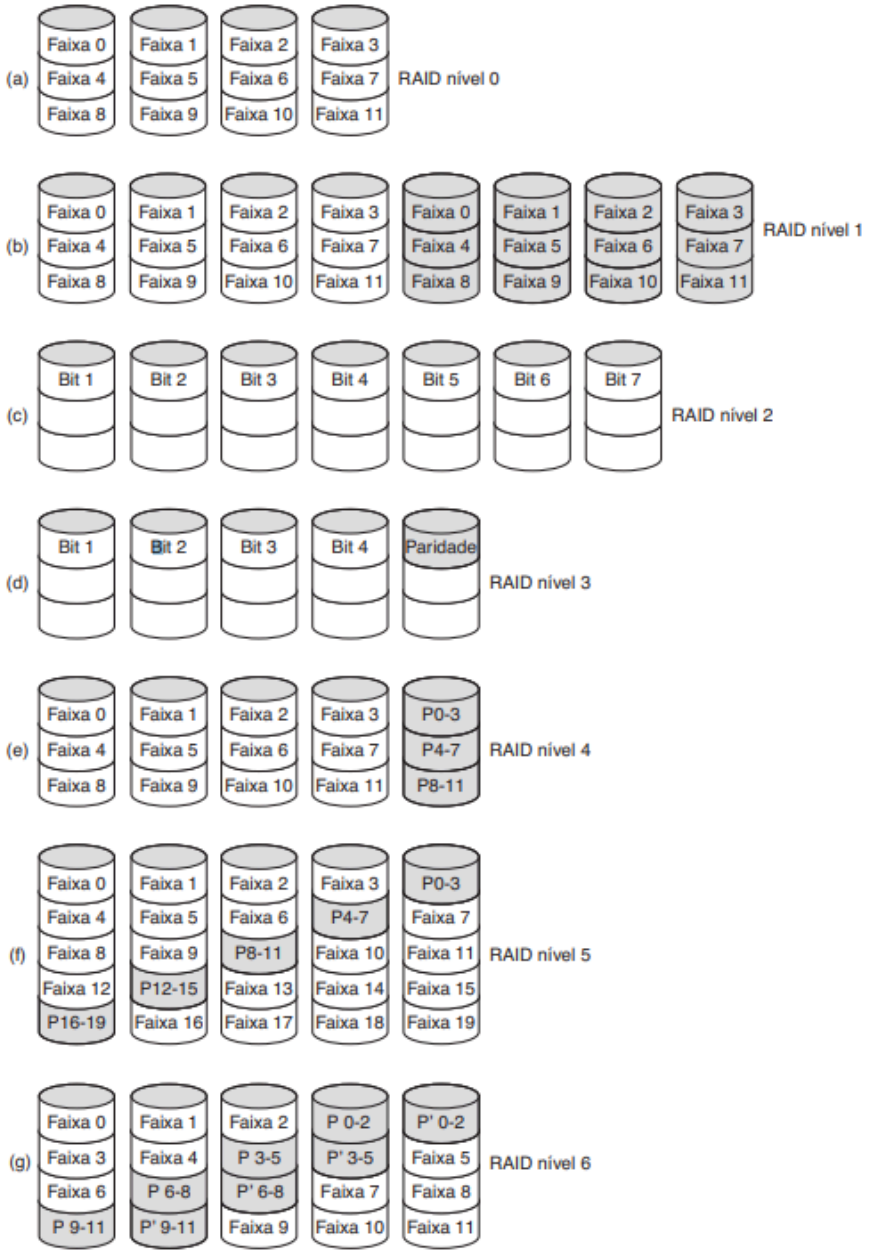
Hamming nos sete discos, um bit por disco. A desvantagem é que esse esquema exige que todos os discos tenham suas rotações sincronizadas, e isso só faz sentido com um número substancial de discos (mesmo com 32 discos de dados e 6 discos de paridade, a sobrecarga é 19%). Ele também exige muito do controlador, visto que é necessário fazer a verificação de erro do código *Hamming* a cada chegada de bit.

- O RAID nível 3 é uma versão simplificada do RAID nível 2. Ele está ilustrado na Figura 9 (d). Aqui um único bit de paridade é calculado para cada palavra de dados e escrito para o disco de paridade. Como no RAID nível 2, os discos devem estar exatamente sincronizados, tendo em vista que palavras de dados individuais estão espalhadas por múltiplos discos. Em um primeiro momento, poderia parecer que um único bit de paridade fornece apenas a detecção de erros, não a correção deles. Para o caso de erros não detectados aleatórios, essa observação é verdadeira. No entanto, para o caso de uma quebra de disco, ele fornece a correção completa do erro de 1 bit, pois a posição do bit defeituoso é conhecida. Caso um disco quebre, o controlador apenas finge que todos os bits são 0s. Se uma palavra tem um erro de paridade, o bit do disco quebrado deve ter sido um 1, então ele é corrigido. Embora ambos os níveis RAID 2 e 3 ofereçam taxas de dados muito altas, o número de solicitações de E/S separadas por segundo que eles podem tratar não é melhor do que para um único disco.
- O RAID nível 4, Figura 9 (e), é como o RAID nível 0, com uma paridade faixa-por-faixa escrita em um disco extra. Por exemplo, se cada faixa tiver k bytes de comprimento, todas as faixas são processadas juntas por meio de um OU EXCLUSIVO, resultando em uma faixa de paridade de k bytes de comprimento. Se um disco quebra, os bytes perdidos podem ser recalculados do disco de paridade por uma leitura do conjunto inteiro de discos. Esse projeto protege contra a perda de um disco, mas tem um desempenho ruim para atualizações pequenas. Se um setor for modificado, é necessário ler todos os arquivos a fim de recalcular a paridade, que então deve ser reescrita. Como alternativa, ele pode ler os dados antigos do usuário, assim como os dados antigos de paridade, e recalcular a nova paridade a partir deles. Mesmo com essa otimização, uma pequena atualização exige duas leituras e duas escritas.
- Em consequência da carga pesada sobre o disco de paridade, ele pode tornar-se um gargalo. Esse gargalo é eliminado no RAID nível 5 distribuindo os bits de paridade uniformemente por todos os discos, de modo circular (round-robin),

como mostrado na Figura 9 (f). No entanto, caso ocorra uma quebra do disco, a reconstrução do disco falhado é um processo complexo.

- O RAID nível 6, Figura 9 (g), é similar ao RAID nível 5, exceto que um bloco de paridade adicional é usado. Em outras palavras, os dados são divididos pelos discos com dois blocos de paridade em vez de um. Em consequência, as escritas são um pouco mais caras por causa dos cálculos de paridade, mas as leituras não incorrem em nenhuma penalidade de desempenho. Ele oferece mais confiabilidade (imagine o que acontece se um RAID nível 5 encontra um bloco defeituoso bem no momento em que ele está reconstruindo seu conjunto).

Figura 9:



4.4 Formatação de disco

Antes que um disco possa ser usado, cada prato deve passar por uma formatação de baixo nível feita por software. A formatação consiste em uma série de trilhas concêntricas, cada uma contendo uma série de setores, com pequenos intervalos entre eles. O preâmbulo começa com um determinado padrão de bits que permite que o hardware reconheça o começo do setor. Ele também contém os números do cilindro e setor, assim como outras informações. O tamanho da porção de dados é determinado pelo programa de formatação de baixo nível. A maioria dos discos usa setores de 512 bytes.

O campo ECC contém informações redundantes que podem ser usadas para a recuperação de erros de leitura. O tamanho e o conteúdo desse campo variam de fabricante para fabricante, dependendo de quanto espaço de disco o projetista está disposto a abrir mão em prol de uma maior confiabilidade, assim como o grau de complexidade do código de ECC que o controlador é capaz de manejar. Um campo de ECC de 16 bytes não é incomum. Além disso, todos os discos rígidos têm algum número de setores sobressalentes alocados para serem usados para substituir setores com defeito de fabricação.

A posição do setor 0 em cada trilha é deslocada com relação à trilha anterior quando a formatação de baixo nível é realizada. Esse deslocamento, chamado de deslocamento de cilindro (*cylinder skew*), é feito para melhorar o desempenho. A ideia é permitir que o disco leia múltiplas trilhas em uma operação contínua sem perder dados.

O resultado da formatação de baixo nível faz com que a capacidade do disco seja reduzida, dependendo dos tamanhos do preâmbulo, do intervalo entre setores e do ECC, assim como o número de setores sobressalentes reservado. Muitas vezes a capacidade formatada é 20% mais baixa do que a não formatada. Os setores sobressalentes não contam para a capacidade formatada; então, todos os discos de um determinado tipo têm exatamente a mesma capacidade quando enviados, independentemente de quantos setores defeituosos eles de fato têm (se o número de setores defeituosos exceder o de sobressalentes, o disco será rejeitado e não enviado). Há uma confusão considerável a respeito da capacidade de disco porque alguns fabricantes anunciavam a capacidade não formatada para fazer seus discos parecerem maiores do que eram na realidade. Por exemplo, considere um disco cuja

capacidade não formatada é de 200 × 109 bytes. Ele poderia ser vendido como um disco de 200 GB. No entanto, após a formatação, possivelmente apenas 170 × 109 bytes estavam disponíveis para dados.

O passo final na preparação de um disco para ser usado é realizar uma formatação de alto nível de cada partição (separadamente). Essa operação insere um bloco de inicialização, a estrutura de gerenciamento de armazenamento livre (lista de blocos livres ou mapa de bits), diretório-raiz e um sistema de arquivo vazio. Ela também coloca um código na entrada da tabela de partições dizendo qual sistema de arquivos é usado na partição, pois muitos sistemas operacionais suportam múltiplos sistemas de arquivos incompatíveis (por razões históricas). Nesse ponto, o sistema pode ser inicializado. Quando a energia é ligada, o BIOS entra em execução inicialmente e então carrega o registro mestre de inicialização e salta para ele. Esse programa então confere para ver qual partição está ativa. Então ele carrega o setor de inicialização específico daquela partição e o executa. O setor de inicialização contém um programa pequeno que geralmente carrega um carregador de inicialização maior que busca no sistema de arquivos para encontrar o núcleo do sistema operacional. Esse programa é carregado na memória e executado.

4.5 Algoritmos de escalonamento de braço de disco

O tempo necessário para ler ou escrever um bloco de disco é determinado por três fatores principais: o tempo de busca (movimento do braço para o cilindro correto), o atraso de rotação (espera pelo setor desejado posicionar-se sob o cabeçote) e o tempo efetivo de transferência de dados. Dentre esses, o tempo de busca é o mais significativo, tornando sua otimização essencial para melhorar o desempenho do sistema.

Quando um driver de disco utiliza uma política ***First-Come, First-Served*** **(FCFS)**, as solicitações são atendidas na ordem de chegada, o que pode resultar em movimentos ineficientes do braço, aumentando o tempo total de acesso. Uma alternativa mais eficiente é o algoritmo ***Shortest Seek First*** **(SSF)**, que prioriza a solicitação mais próxima da posição atual do braço, reduzindo significativamente o movimento total. No entanto, esse método pode causar inanição para requisições em cilindros distantes se novas solicitações chegarem continuamente.

Para equilibrar eficiência e justiça, o **algoritmo do elevador (SCAN)** é frequentemente empregado. Ele mantém o braço em movimento em uma direção (subida ou descida) até que não haja mais solicitações pendentes nesse sentido, invertendo então o sentido. Esse método evita inanição e limita a distância máxima percorrida pelo braço a no máximo duas vezes o número total de cilindros. Uma variação desse algoritmo, conhecida como **C-SCAN**, realiza varreduras unidirecionais completas, retornando ao cilindro inicial após atingir o final, o que reduz a variação nos tempos de resposta.

Otimizações adicionais incluem a seleção do setor mais próximo quando o controlador permite identificar a posição atual do cabeçote, priorizando requisições que estão prestes a passar sob ele. Além disso, caches no controlador armazenam setores lidos recentemente ou adjacentes, acelerando acessos subsequentes. Em sistemas com múltiplos dispositivos, o sistema operacional gerencia tabelas de solicitações separadas para cada unidade, buscando otimizar buscas simultâneas e reduzir tempos ociosos.

É importante destacar que os algoritmos de escalonamento assumem que a geometria virtual do disco, vista pelo sistema operacional, corresponde à física. Caso contrário, o escalonamento pode ser ineficaz. No entanto, controladores avançados podem implementar esses algoritmos internamente, mantendo a eficiência mesmo com abstrações de geometria.

Em resumo, a escolha do algoritmo de escalonamento deve considerar tanto a eficiência quanto a equidade no atendimento das solicitações. O **SCAN** e suas variantes são amplamente adotados por oferecerem um equilíbrio adequado entre desempenho e justiça, adaptando-se a diferentes cargas de trabalho e configurações de hardware.

4.6 Tratamento de erros

Os defeitos de fabricação de um disco introduzem setores defeituosos, isto é, setores que não leem corretamente de volta o valor recém-escrito neles. Se o defeito for muito pequeno, imagine, apenas alguns bits, será possível usar o setor defeituoso e deixar que o ECC corrija os erros todas as vezes. Se o defeito for maior, o erro não poderá ser mascarado.

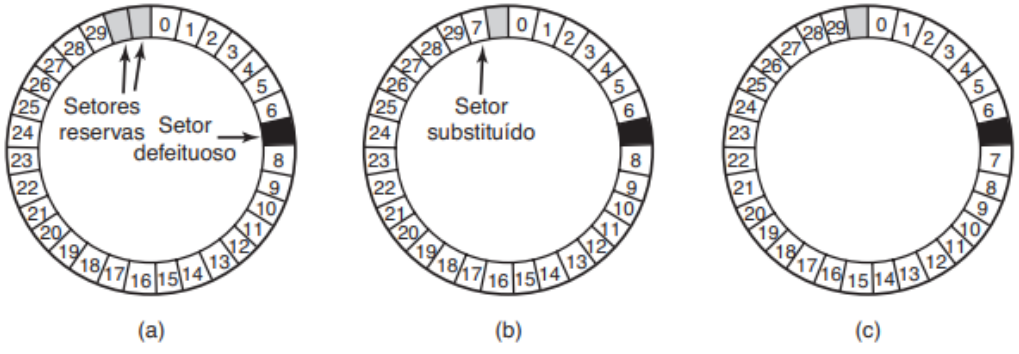
Existem duas abordagens gerais para blocos defeituosos: lidar com eles no controlador ou no sistema operacional. Na primeira abordagem, antes que o disco

seja enviado da fábrica, ele é testado e uma lista de setores defeituosos é escrita no disco. Para cada setor defeituoso, um dos reservas o substitui. Há duas maneiras de realizar essa substituição. Na Figura 10 (a), vê-se uma única trilha de disco com 30 setores de dados e dois reservas. O setor 7 é defeituoso. O que o controlador pode fazer é remapear um dos reservas como setor 7, como mostrado na Figura 10 (b). A outra saída é deslocar todos os setores de uma posição, como mostrado na Figura 10 (c). Em ambos os casos, o controlador precisa saber qual setor é qual. Ele pode controlar essa informação por meio de tabelas internas (uma por trilha) ou reescrevendo os preâmbulos para dar o novo número dos setores remapeados. Se os preâmbulos forem reescritos, o método da Figura 10 (c) dará mais trabalho (pois 23 preâmbulos precisam ser reescritos), mas em última análise ele proporcionará um desempenho melhor, pois uma trilha inteira ainda poderá ser lida em uma rotação. rros também podem ser desenvolvidos durante a operação normal após o dispositivo ter sido instalado.

A primeira linha de defesa para tratar um erro com que o ECC não consegue lidar é apenas tentar a leitura novamente. Alguns erros de leitura são passageiros, isto é, são causados por grãos de poeira sob o cabeçote e desaparecerão em uma segunda tentativa. Se o controlador notar que ele está encontrando erros repetidos em um determinado setor, pode trocar para um reserva antes que o setor morra completamente. Dessa maneira, nenhum dado é perdido e o sistema operacional e o usuário nem notam o problema. Em geral, o método da Figura 10 (b) tem de ser usado, pois os outros setores podem conter dados agora. A utilização do método da Figura 10 (c) exigiria não apenas reescrever os preâmbulos, como copiar todos os dados também.

Figura 10:

(a) Uma trilha de disco com um setor defeituoso. (b) Substituindo um setor defeituoso com um reserva. (c) Deslocando todos os setores para pular o setor defeituoso.



Se o controlador não tiver a capacidade de remapear setores transparentemente, o sistema operacional precisará fazer a mesma coisa no software. Isso significa que ele precisará primeiro adquirir uma lista de setores defeituosos, seja lendo a partir do disco, ou simplesmente testando o próprio disco inteiro. Uma vez que ele saiba quais setores são defeituosos, poderá construir as tabelas de remapeamento. Se o sistema operacional quiser usar a abordagem da Figura 10 (c), deverá deslocar os dados dos setores 7 a 29 um setor para cima.

Se for feito um backup do disco, arquivo por arquivo, é importante que o utilitário usado para realizar o backup não tente copiar o arquivo com o bloco defeituoso. Para evitar isso, o sistema operacional precisa esconder o arquivo com o bloco defeituoso tão bem que mesmo esse utilitário para backup não consiga encontrá-lo. Se o disco for copiado setor por setor em vez de arquivo por arquivo, será difícil, senão impossível, evitar erros de leitura durante o backup. A única esperança é que o programa de backup seja esperto o suficiente para desistir depois de 10 leituras fracassadas e continuar com o próximo setor.

Setores defeituosos não são a única fonte de erros. Erros de busca causados por problemas mecânicos no braço também ocorrem. O controlador controla a posição do braço internamente. Para realizar uma busca, ele emite um comando para o motor do braço para movê-lo para o novo cilindro. Quando o braço chega ao seu destino, o controlador lê o número do cilindro real do preâmbulo do setor seguinte. Se o braço estiver no lugar errado, ocorreu um erro de busca. O controlador é de fato um pequeno computador especializado, completo com software, variáveis, buffers e logo, defeitos.

4.7 Armazenamento estável

Apesar de existirem diversas maneiras de impedir erros, esses sistemas listados acima não protegem contra erros de gravação que inserem dados corrompidos. Eles também não protegem contra quedas do sistema durante a gravação corrompendo os dados originais sem substituí-los por dados novos.

Para algumas aplicações, é essencial que os dados nunca sejam perdidos ou corrompidos, mesmo diante de erros de disco e CPU. O ideal é que um disco deve funcionar simplesmente o tempo inteiro sem erros. Infelizmente, isso não é possível. O que é possível é um subsistema de disco que tenha a seguinte propriedade:

quando uma escrita é lançada para ele, o disco ou escreve corretamente os dados ou não faz nada, deixando os dados existentes intactos. Esse sistema é chamado de armazenamento estável e é implementado no software.

O armazenamento estável utiliza um par de discos idênticos com blocos correspondentes para garantir a integridade dos dados. O modelo assume que erros de escrita podem ser detectados através da verificação do ECC (Código de Correção de Erros) durante operações subsequentes de leitura.

Embora a detecção não seja absolutamente infalível devido ao limite matemático das combinações possíveis entre dados e ECC, a probabilidade de um erro não ser detectado é extremamente baixa, tornando-a praticamente irrelevante. Além disso, o modelo considera que falhas espontâneas em blocos de disco são raras e que a probabilidade de ocorrência simultânea em ambos os discos é insignificante. Falhas na CPU também são levadas em conta, assumindo-se que podem interromper operações de escrita, resultando em dados inconsistentes, mas detectáveis.

Para assegurar a confiabilidade, três operações principais são definidas:

1. Escrita Estável: Esta operação começa escrevendo um bloco no Disco 1, seguido de uma releitura para verificar sua integridade. Caso o ECC indique um erro, o processo é repetido até que a escrita seja bem-sucedida ou até que um número máximo de tentativas seja alcançado, momento em que o bloco é remapeado para uma área reserva.

Após a confirmação da escrita correta no Disco 1, o mesmo bloco é escrito e verificado no Disco 2. Essa abordagem garante que, na ausência de falhas na CPU, ambas as cópias do bloco sejam idênticas e válidas.

2. Leitura Estável: Durante uma leitura, o bloco é primeiro lido do Disco 1. Se o ECC indicar um erro, a operação é repetida por um número pré-definido de vezes. Se todas as tentativas falharem, o bloco correspondente no Disco 2 é lido. Como uma escrita estável bem-sucedida garante duas cópias válidas do bloco, e considerando a baixa probabilidade de falhas simultâneas em ambos os discos, a leitura estável é considerada sempre bem-sucedida.

3. Recuperação de falhas: Após uma queda do sistema, um programa de recuperação compara os blocos correspondentes nos dois discos. Se um bloco estiver com erro de ECC, ele é substituído pelo bloco válido do outro disco. Se ambos os blocos estiverem íntegros mas forem diferentes, o bloco do Disco 1

prevalece e é copiado para o Disco 2. Esse processo assegura a consistência dos dados mesmo após falhas inesperadas.

Em caso de falha da CPU durante uma escrita estável, o sistema é capaz de lidar com diferentes cenários:

- Se a falha ocorrer antes de qualquer escrita, os dados antigos permanecem intactos.
- Se a falha acontecer durante a escrita no Disco 1, o bloco danificado é restaurado a partir do Disco 2 durante a recuperação.
- Se a falha ocorrer após a escrita no Disco 1, mas antes da escrita no Disco 2, o programa de recuperação copia o bloco do Disco 1 para o Disco 2, completando a operação.

Para otimizar o processo de recuperação, é possível utilizar memória RAM não volátil para registrar o bloco sendo escrito, reduzindo a necessidade de verificar todos os blocos após uma falha. Caso a RAM não volátil não esteja disponível, uma simulação pode ser feita usando blocos reservas nos discos, embora isso aumente a sobrecarga de operações.

Por fim, para evitar a degradação acumulativa dos dados, recomenda-se uma varredura diária em ambos os discos para corrigir quaisquer erros espontâneos, garantindo que os discos permaneçam consistentes e íntegros ao longo do tempo. Essa abordagem assegura que o armazenamento estável seja altamente confiável, mesmo em cenários de falhas parciais.

5. RELÓGIOS

Relógios - também chamados de temporizadores (*timers*) são essenciais para o funcionamento de qualquer sistema multiprogramado por uma variedade de razões. Entre outras coisas, eles mantêm a hora do dia e evitam que um processo monopolize a CPU. O software do relógio pode tomar a forma de um driver de dispositivo, muito embora um relógio não seja nem um dispositivo de bloco, como um disco, nem um dispositivo de caractere, como um mouse.

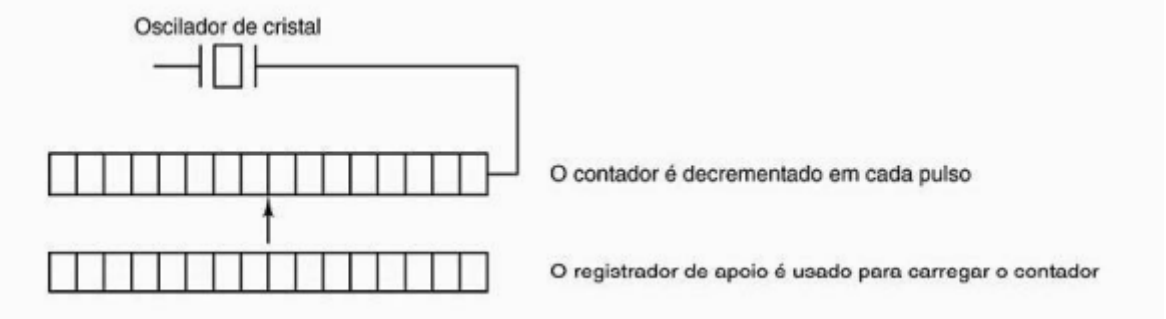
5.1 Hardware do relógio

Dois tipos de relógios são usualmente utilizados nos computadores e ambos são bem diferentes dos relógios comuns de pulso ou de parede. Os relógios mais simples são ligados a uma linha de voltagem de 110 ou 220 volts e causam uma

interrupção em cada ciclo de voltagem, a uma frequência de 50 ou 60 hertz. Esses relógios já foram os mais empregados, mas são raros hoje em dia.

O outro tipo de relógio é construído a partir de três componentes: um **oscilador de cristal**, um **contador** e um **registrador de apoio**, como mostra na imagem abaixo (Figura 11).

Figura 11:



Quando um fragmento de cristal de quartzo é cortado corretamente e montado sob tensão, ele pode ser usado para gerar um sinal periódico de altíssima precisão, geralmente na faixa de várias centenas de megahertz, dependendo do cristal escolhido. Usando a eletrônica, esse sinal básico pode ser multiplicado por um pequeno inteiro para obter frequências de até 1.000 MHz ou ainda mais. Pelo menos um desses circuitos pode ser encontrado em qualquer computador, fornecendo um sinal de sincronização para os vários circuitos do computador. Esse sinal é colocado em um contador para fazê-lo contar regressivamente até zero. Quando o contador atinge o zero ele gera uma interrupção na CPU.

. Relógios programáveis: Apresentam vários modos de operação. No modo disparo único (*one-shot mode*), ao ser inicializado, o relógio copia o valor do registrador de apoio para dentro do contador e, então, decrementa o contador a cada pulso do cristal. Quando o contador chega a zero, ele causa uma interrupção e para até que seja explicitamente reinicializado pelo software. No modo onda quadrada, após atingir o zero e causar a interrupção, o registrador de apoio é automaticamente copiado para dentro do contador e o processo todo é repetido interminavelmente. Essas interrupções periódicas são chamadas de tiques do relógio.

A vantagem do relógio programável é que sua frequência de interrupção pode ser controlada pelo software. Se um cristal de 500 MHz é usado, então o contador é pulsado a cada 2ns. Com registradores de 32 Bits (sem sinal) as interrupções podem ser programadas para ocorrer em intervalos de 2 ns a 8,6 s. Os chips dos

relógios programáveis contêm, em geral, dois ou mais relógios programáveis independentes e apresentam ainda muitas outras opções, como contar com incremento em vez de decremento, desabilitar interrupções etc.

Para evitar a perda do horário atual quando a energia do computador é desligada, a maioria dos computadores tem um relógio de segurança mantido por uma bateria, implementado com o tipo de circuito de baixo consumo usado nos relógios digitais de pulso. O relógio da bateria pode ser lido na inicialização do sistema. Se o relógio de segurança não está presente, o software pode perguntar para o usuário a data e a hora atualizadas. Existe também uma forma padrão para um sistema de rede obter o horário atual de um computador remoto.

5.2 Software do relógio

Tudo o mais que envolva tempo deve ser realizado pelo software, o driver do relógio. As obrigações exatas do driver do relógio variam de acordo com o sistema operacional, mas em sua maioria incluem as seguintes ações:

1. Manter a hora do dia;
2. Evitar que algum processo execute durante um tempo maior do que o permitido;
3. Contabilizar o uso da CPU;
4. Tratar a chamada de sistema alarme feita pelos processos do usuário;
5. Fornecer temporizadores *watch-dog* para partes do próprio sistema;
6. Gerar perfis de execução, realizar monitoramentos e coletar estatísticas.

A primeira função do relógio - a manutenção da hora do dia (também chamada de tempo real) - não é muito complexa. Ela requer apenas o incremento do contador em cada tick do relógio, conforme mencionado anteriormente. O único cuidado a ser tomado é com relação ao número de bits no contador. Com uma frequência de relógio de 60 Hz, um contador de 32 bits estouraria sua capacidade em apenas dois anos.

A segunda função do relógio é evitar que os processos executem por muito tempo. Sempre que um processo é inicializado, o escalonador inicializa o contador com o valor do quantum do processo em tiques de relógio. Em cada interrupção do relógio, o driver do relógio decrementa o contador do *quantum* de 1. Quando o contador atinge o zero, o driver do relógio chama o escalonador para selecionar outro processo.

A terceira função do relógio é contabilizar o uso da CPU. A maneira mais correta de fazer isso é inicializar um segundo temporizador, diferente do relógio principal do sistema, sempre que um processo é inicializado. Quando referido processo é interrompido, o temporizador pode ser lido, permitindo saber por quanto tempo ele esteve em execução. Para isso, o segundo temporizador deve ser salvo na ocorrência de uma interrupção e restaurado posteriormente. Uma maneira menos precisa, mas muito mais simples para contabilizar o uso da CPU é manter um ponteiro para a entrada da tabela de processos relativa ao processo em execução em uma variável global. A cada tique do relógio, um campo na entrada do processo atual sofre um incremento. Desse modo, cada tique de relógio é contabilizado para o processo em execução no momento do tique. Um problema não muito sério com essa estratégia é que, se muitas interrupções ocorrerem durante a execução de um processo, ainda assim ele será contabilizado com um tick completo embora não tenha trabalhado muito. A contabilidade correta para a CPU durante as interrupções é bastante dispendiosa e raramente é feita.

Em muitos sistemas, um processo pode solicitar que o sistema operacional lhe dê um aviso após um certo intervalo. O aviso consiste, em geral, em um sinal, uma interrupção, uma mensagem ou algo similar. Uma aplicação que requer o uso desses avisos e a comunicação em rede, na qual um pacote não confirmado dentro de um certo intervalo de tempo deve ser retransmitido. Uma outra aplicação é o ensino auxiliado por computador, em que um estudante que não forneça a resposta dentro de um certo tempo recebe a resposta do computador

Se o driver do relógio gerencia relógios suficientes, ele pode ajustar um relógio separado para cada requisição. Se não é esse o caso, ele deve simular vários relógios virtuais com um único relógio físico. Uma maneira de fazer isso é ter uma tabela na qual são mantidos os tempos dos sinais para todos os temporizadores pendentes, bem como uma variável que fornece o tempo do sinal seguinte. Sempre que a hora do dia é atualizada, o driver verifica se o tempo do sinal mais próximo já decorreu. Em caso afirmativo, a tabela é pesquisada para encontrar o próximo sinal a ocorrer.

Partes do sistema operacional também precisam ajustar temporizadores. Estes, por sua vez, são chamados de temporizadores *watch-dog*. Por exemplo, os discos flexíveis não sofrem rotação quando não estão em uso, para evitar desgaste na mídia e no cabeçote do disco. Quando os dados de um disco flexível são

necessários, o motor deve primeiro ser inicializado. Somente após o disco flexível estar em rotação na velocidade ideal é que as operações de E/S podem ser inicializadas. Quando um processo tenta ler de um disco flexível ocioso, o driver do disco flexível inicializa o motor e, então, ajusta um temporizador *watch-dog* para que gere uma interrupção após um intervalo de tempo suficientemente longo.

O mecanismo usado pelo driver do relógio para tratar temporizadores *watch-dog* é o mesmo empregado para tratar os sinais do usuário. A única diferença é que, quando um temporizador se esgota, o driver do relógio, em vez de causar um sinal, chama um procedimento fornecido pelo chamador. O procedimento é parte do código do chamador. O procedimento chamado pode fazer o que for necessário, até mesmo causar uma interrupção, embora dentro do núcleo as interrupções muitas vezes sejam inconvenientes e não haja sinais. Essa é a razão de ser do mecanismo de watch-dog. É importante notar que esse mecanismo funciona somente quando o driver do relógio e o procedimento a ser chamado estão no mesmo espaço de endereçamento. A última questão da lista é o perfil de execução. Alguns sistemas operacionais fornecem um mecanismo pelo qual um programa do usuário pode obter do sistema um histograma de seu contador de programa, de modo que possa ver onde seu tempo está sendo gasto. Quando existe a possibilidade de extrair o perfil de execução, a cada tick de relógio o driver verifica se o perfil de execução do processo atual está sendo obtido e, em caso afirmativo, calcula o intervalo (uma faixa de endereços) correspondente ao contador de programa atual. Ele, então, incrementa esse intervalo de 1. Esse mecanismo também pode ser usado para extrair o perfil de execução do próprio sistema.

5.3 Temporizadores por software

A maioria dos computadores tem um segundo relógio programável, que pode ser ajustado para causar interrupções a qualquer taxa que um programa precisar. Esse relógio é outro temporizador além do principal, cujas funções foram descritas anteriormente. Quando a frequência de interrupção é baixa, não há por que não usar um segundo temporizador para os propósitos da aplicação. O problema surge quando a frequência do temporizador da aplicação é muito alta.

Em geral, existem duas maneiras de gerenciar E/S: **interrupções** e ***polling***. As interrupções têm baixas latências, isto é, elas ocorrem imediatamente após o evento propriamente dito, com pouco ou nenhum atraso. Por outro lado, nas CPUs modernas, as interrupções causam uma sobrecarga substancial em virtude da

necessidade dos chaveamentos de contextos e da influência delas no *pipeline*, na TLB e na cache.

Uma alternativa às interrupções é permitir que a própria aplicação verifique por meio de ***polling***, em certos intervalos de tempo, a ocorrência do evento esperado. Agindo assim, as interrupções são evitadas, mas em compensação talvez ocorra um atraso substancial de tempo pois o evento pode ocorrer imediatamente após uma verificação que acabou de ser realizada, fazendo com que a aplicação tenha de esperar mais um intervalo de *polling* quase completo para perceber a ocorrência do evento. Na média, o atraso é de metade do intervalo de *polling*.

Os temporizadores por software evitam interrupções. Em vez disso, sempre que o núcleo está executando por alguma outra razão, imediatamente antes de retornar para o modo usuário ele verifica o relógio de tempo real para averiguar se um temporizador por software expirou. Se o temporizador expirar, o evento escalonado (por exemplo, a transmissão de um pacote ou a verificação da chegada de um pacote) é executado, sem a necessidade de chavear para o modo núcleo, visto que o sistema já está em execução. Após o trabalho ter sido executado, o temporizador por software é de novo reinicializado. Tudo o que se tem a fazer é copiar o valor atual do relógio para o temporizador e adicionar-lhe um intervalo de tempo.

Os temporizadores por software podem ou não ser sustentados pela frequência na qual as entradas no núcleo são feitas por outras razões. Entre essas razões estão:

1. Chamadas de sistema.
2. Faltas na TLB.
3. Faltas de páginas.
4. Interrupções de E/S.
5. CPU ociosa.

Para saber a frequência de ocorrência desses eventos, Aron e Druschel realizaram medições com várias cargas na CPU, incluindo um servidor da Web totalmente carregado com um processo em segundo plano, executando áudios em tempo real da Internet e recompilando o núcleo do UNIX. A frequência média de entrada no núcleo variou de 2 us a 18 μs, com aproximadamente metade dessas entradas sendo chamadas de sistema. Assim, para uma aproximação de primeira ordem, a existência de um temporizador por software operando a cada 12 us é

possível, embora o tempo limite ocasionalmente expire. Para aplicações que enviam pacotes ou verificam a chegada de pacotes, um atraso de 10 us de tempos em tempos é melhor do que ter interrupções consumindo até 35 por cento do tempo da CPU.

Obviamente, existirão períodos em que não haverá chamadas de sistema, faltas na TLB ou faltas de página, nos quais nenhum temporizador por software será reinicializado. Para colocar um limite superior nesses intervalos, o segundo relógio de hardware pode ser ajustado para reinicializar, suponha, a cada 1 ms. Se a aplicação pode viver com somente mil pacotes por segundo em intervalos ocasionais, então a combinação dos temporizadores por software com um temporizador de hardware de baixa frequência pode ser melhor do que a E/S orientada somente à interrupção ou controlada apenas por *polling*.

6. INTERFACES COM O USUÁRIO: TECLADO, MOUSE, MONITOR

Todo computador de propósito geral tem um teclado e um monitor de vídeo (e, frequentemente, um mouse) que permitem que as pessoas interajam com ele.

. Terminais: Dispositivos contendo um monitor e um teclado conectados, utilizados por usuários remotos em computadores de grande porte.

6.1 Software de entrada

As informações do usuário vêm, primeiro, do teclado e do mouse. Em um computador pessoal, o teclado contém um processador embutido que costuma se comunicar com um chip controlador na placa-mãe através de uma porta serial, embora seja cada vez mais comum que os teclados sejam conectados a uma porta USB. Uma interrupção é gerada sempre que uma tecla é pressionada e outra sempre que a tecla é liberada. Em cada uma dessas interrupções, o driver do teclado extrai a informação sobre o que acontece na porta de E/S associada ao periférico. Todo o restante acontece no nível do software e é bastante independente do hardware.

- **Software de teclado**

O número na porta de E/S é o da tecla - denominado código de varredura - e não o código ASCII. Os teclados têm menos de 128 teclas e, portanto, somente 7 bits são necessários na representação do número da tecla. O oitavo bit é definido

como 0 quando a tecla é pressionada e 1 quando ela é liberada. Cabe ao driver controlar o estado de cada tecla (pressionada ou liberada).

Quando a tecla A é pressionada, o código da tecla (30) é colocado no registrador de E/S. O driver é capaz de determinar se ela é minúscula, maiúscula, CTRL-A, ALT-A, CTRL-ALT-A ou alguma outra combinação. Visto que o driver pode dizer quais teclas foram pressionadas mas ainda não liberadas (por exemplo, usando SHIFT), ele tem informação suficiente para fazer o trabalho. Por exemplo, a sequência de teclas

DEPRESS SHIFT, DEPRESS A, RELEASE A, RELEASE SHIFT

indica um caractere A maiúsculo. Contudo, a sequência

DEPRESS SHIFT, DEPRESS A, RELEASE SHIFT, RELEASE A

também indica um caractere A maiúsculo.

Há duas filosofias possíveis para o driver:

1. Na primeira delas, o trabalho do driver consiste apenas em aceitar a entrada e passá-la adiante inalterada. Um programa que está lendo de um terminal obtém uma sequência grosseira de códigos ASCII. Essa filosofia é bastante adequada para as necessidades de editores de terminais sofisticados como o *emacs*, que permite que o usuário associe uma ação arbitrária para qualquer caractere ou sequência de caracteres. Contudo, ela implica que, se o usuário digitar *dste* em vez de *date* e depois corrigir o erro digitando três caracteres de retrocesso (*backspace*) mais *ate*, seguidos de um caractere de retorno de carro (*carriage return - CR*), o programa do usuário receberá todos os 11 caracteres digitados em código ASCII, como segue:

d s t e □ □ □ a t e C R

Nem todos os programas querem esse nível de detalhe. Muitas vezes eles simplesmente desejam a entrada corrigida e não a sequência exata em que foi produzida. Essa observação leva à segunda filosofia.

2. O driver trata toda a edição interna da linha e somente entrega as linhas corrigidas para os programas do usuário. A primeira filosofia é baseada em caracteres; a segunda, em linhas. Originalmente, eles eram chamados de modo bruto ou natural (*raw mode*) e modo preparado (*cooked mode*), respectivamente. O padrão POSIX usa a expressão menos pitoresca modo canônico para descrever o modo com base em linha. O modo não canônico é equivalente ao modo bruto, embora muitos detalhes do comportamento do terminal possam ser trocados. Os

sistemas compatíveis com o POSIX fornecem várias funções de biblioteca que suportam a escolha do modo e a troca de muitos parâmetros.

Se o teclado estiver no modo canônico (preparado), os caracteres devem ser armazenados até que uma linha inteira seja montada, já que o usuário pode decidir apagar parte dela mais tarde. Mesmo quando o teclado está no modo bruto (*raw*), o programa pode ainda não ter solicitado nenhuma entrada e os caracteres podem estar armazenados para viabilizar a digitação antecipada. Pode ser usado um buffer dedicado ou diversos buffers podem ser alocados em um pool. No primeiro tipo, a digitação antecipada tem um limite; no segundo, não. Essa limitação fica clara quando o usuário está digitando em uma janela de comandos (como no Windows) e inicializa um comando (como uma compilação) que ainda não foi concluído. Os próximos caracteres a serem digitados precisam ser bufferizados, pois o shell não está preparado para lê-los.

. **Eco:** Aparição dos caracteres digitados pelos usuários na tela de vídeo.

. **Problemas em relação ao eco:** Em relação ao eco, o driver do teclado deve ser capaz de superar algumas complicações. O eco é complicado pelo fato de que um programa pode estar escrevendo no monitor enquanto o usuário digita (pense na digitação em uma janela do shell.) No mínimo, o driver do teclado tem de calcular onde colocar a nova entrada sem que ela seja sobrescrita pela saída do programa. O eco também se torna complicado quando, por exemplo, mais de 80 caracteres são mostrados em um monitor com linhas de 80 caracteres (ou algum outro número). Dependendo da aplicação, pode ser apropriado mostrar na linha seguinte os caracteres excedentes. Alguns drivers simplesmente truncam as linhas em 80 caracteres descartando todos os caracteres além da octogésima coluna. Outro problema é o tratamento da tabulação. Em geral, o driver é capaz de calcular onde o cursor está atualmente localizado, levando em consideração tanto a saída dos programas quanto a saída do eco, e assim calcular também o número correto de espaços a ser ecoado.

Também há o problema de equivalência de dispositivo. Logicamente, no final de uma linha de texto existem um caractere CR, para mover o cursor de volta à coluna 1, e um caractere de alimentação de linha (*linefeed - LF*), para avançar o cursor para a linha seguinte. Obrigar os usuários a digitar ambos os caracteres ao final de cada linha não seria uma boa ideia. Fica a cargo do driver converter qualquer coisa que chega para o formato-padrão interno usado pelo sistema

operacional. No UNIX, a tecla ENTER é convertida para uma alimentação de linha (LF-line feed) para fins de armazenamento interno. No Windows, a mesma tecla é convertida para um retorno do carro (CR-carriage return), seguido de uma alimentação de linha. Não importa qual seja a convenção interna: o terminal pode requerer que ambos, LF e CR, sejam ecoados para que a tela de vídeo seja atualizada corretamente.

. Caracteres especiais: Ao executar em modo canônico, vários caracteres de entrada têm significados especiais. A tabela (Figura 12) mostra todos os caracteres especiais necessários para o POSIX. Os caracteres padrão são todos de controle e não deveriam conflitar com a entrada de texto ou com os códigos usados pelos programas; mas todos, com exceção dos dois últimos, podem ser trocados de acordo com o controle do programa.

Figura 12:

Caractere	Nome POSIX	Comentário
CTRL-H	ERASE	Apagar um caractere à esquerda
CTRL-U	KILL	Apagar toda a linha em edição
CTRL-V	LNEXT	Interpretar literalmente o próximo caractere
CTRL-S	STOP	Parar a saída
CTRL-Q	START	Iniciar a saída
DEL	INTR	Interromper processo (SIGINT)
CTRL-\	QUIT	Forçar gravação da imagem da memória (SIGQUIT)
CTRL-D	EOF	Final de arquivo
CTRL-M	CR	Retorno do carro (não modificável)
CTRL-J	NL	Alimentação de linha (não modificável)

- **Software de mouse**

A maioria dos PCs tem um mouse, ou às vezes um *trackball* (que é apenas um mouse deitado de costas). Um tipo comum de mouse tem uma bola de borracha dentro que sai parcialmente através de um buraco na parte de baixo e gira à medida que o mouse é movido sobre uma superfície áspera. À medida que a bola gira, ela desliza contra rolos de borracha colocados em bastões ortogonais. O movimento na direção leste-oeste faz girar o bastão paralelo ao eixo y; o movimento na direção norte-sul faz girar o bastão paralelo ao eixo x.

. Mouse óptico: Outro tipo popular é o mouse óptico, que é equipado com um ou mais diodos emissores de luz e fotodetectores na parte de baixo. Os primeiros tinham de operar em um mouse pad especial com uma grade retangular traçada nele de maneira que o mouse pudesse contar as linhas cruzadas. Os mouses ópticos modernos contêm um chip de processamento de imagens e tiram fotos de baixa resolução contínuas da superfície debaixo deles, procurando por mudanças de uma imagem para outra.

Sempre que um mouse se move uma determinada distância mínima em qualquer direção ou que um botão é acionado ou liberado, uma mensagem é enviada para o computador. A distância mínima é de mais ou menos 0,1 mm (embora ela possa ser configurada no software). Algumas pessoas chamam essa unidade de *mickey*. Mouses podem ter um, dois, ou três botões, dependendo da estimativa dos projetistas sobre a capacidade intelectual dos usuários de controlar mais do que um botão. Alguns mouses têm rodas que podem enviar dados adicionais de volta para o computador. Os mouses *wireless* são iguais aos conectados, exceto que em vez de enviar seus dados de volta para o computador através de um cabo, eles usam rádios de baixa potência, por exemplo, usando o padrão *Bluetooth*.

A mensagem para o computador contém três itens: Δx, Δy, botões. O primeiro item é a mudança na posição x desde a última mensagem. Então vem a mudança na posição y desde a última mensagem. Por fim, o estado dos botões é incluído. O formato da mensagem depende do sistema e do número de botões que o mouse tem. Normalmente, são necessários 3 bytes. A maioria dos mouses responde em um máximo de 40 vezes/s, de maneira que o mouse pode ter percorrido múltiplos mickeys desde a última resposta. Muitas interfaces gráficas fazem distinções entre cliques simples e duplos de um botão de mouse. Se dois cliques estiverem próximos o suficiente no espaço (*mickeys*) e também próximos o suficiente no tempo (milissegundos), um clique duplo é sinalizado. Cabe ao software definir “próximo o suficiente”, com ambos os parâmetros normalmente sendo estabelecidos pelo usuário.

6.2 Software de saída

- **Janelas de texto**

A saída é mais simples do que a entrada quando a saída está sequencialmente em uma única fonte, tamanho e cor. Na maioria das vezes, o programa envia caracteres para a janela atual e eles são exibidos ali. Normalmente, um bloco de caracteres, por exemplo, uma linha, é escrito em uma chamada de sistema.

Editores de tela e muitos outros programas sofisticados precisam ser capazes de atualizar a tela de maneiras complexas, como substituindo uma linha no meio da tela. Para acomodar essa necessidade, a maioria dos drivers de saída suporta uma série de comandos para mover o cursor, inserir e deletar caracteres ou linhas no cursor, e assim por diante. Esses comandos são muitas vezes chamados de sequências de escape. No auge do terminal burro ASCII 25 × 80, havia centenas de tipos de terminais, cada um com suas próprias sequências de escape. Como consequência, era difícil de escrever um software que funcionasse em mais de um tipo de terminal.

. Termcap: Uma solução, introduzida no UNIX de Berkeley, foi um banco de dados terminal chamado de termcap. Esse pacote de software definiu uma série de ações básicas, como mover o cursor para uma coordenada (linha, coluna). Para mover o cursor para um ponto em particular, o software — considere, um editor — usava uma sequência de escape genérica que era então convertida para a sequência de escape real para o terminal que estava sendo escrito. Dessa maneira, o editor trabalhava em qualquer terminal que tivesse uma entrada no banco de dados termcap. Grande parte do software UNIX ainda funciona desse jeito, mesmo em computadores pessoais.

Por fim, a indústria viu a necessidade de padronizar a sequência de escape, então foi desenvolvido o padrão ANSI. Alguns dos valores são mostrados na figura 13.

Figura 13:

Sequência de escape	Significado
ESC [*n*A	Mover *n* linhas para cima
ESC [*n*B	Mover *n* linhas para baixo
ESC [*n*C	Mover *n* espaços para a direita
ESC [*n*D	Mover *n* espaços para a esquerda
ESC [*m* ; *n*H	Mover o cursor para (*m*,*n*)
ESC [*s*J	Limpar a tela a partir do cursor (0 até o final, 1 a partir do início, 2 para ambos)
ESC [*s*K	Limpar a linha a partir do cursor (0 até o final, 1 a partir do início, 2 para ambos)
ESC [*n*L	Inserir *n* linhas a partir do cursor
ESC [*n*M	Excluir *n* linhas a partir do cursor
ESC [*n*P	Excluir *n* caracteres a partir do cursor
ESC [*n*@	Inserir *n* caracteres a partir do cursor
ESC [*n*m	Habilitar efeito *n* (0 = normal, 4 = negrito, 5 = piscante, 7 = reverso)
ESC M	Rolar a tela para cima se o cursor estiver na primeira linha

Considere como essas sequências de escape poderiam ser usadas por um editor de texto. Suponha que o usuário digite um comando dizendo ao editor para deletar toda a linha 3 e então reduzir o espaço entre as linhas 2 e 4. O editor pode enviar a sequência de escape a seguir através da linha serial para o terminal:

ESC [3 ; 1 H ESC [0 K ESC [1 M

(onde os espaços são usados acima somente para separar os símbolos; eles não são transmitidos). Essa sequência move o cursor para o início da linha 3, apaga a linha inteira, e então deleta a linha agora vazia, fazendo que todas as linhas começando em 5 sejam movidas uma linha acima. Então o que era a linha 4 torna-se a linha 3; o que era a linha 5 torna-se a linha 4, e assim por diante. Sequências de escape análogas podem ser usadas para acrescentar texto ao meio da tela. Palavras podem ser acrescentadas ou removidas de uma maneira similar.

- **Sistema X-Window**

Quase todos os sistemas UNIX baseiam sua interface de usuário no Sistema X Window (muitas vezes chamado simplesmente X), desenvolvido no MIT, como parte do projeto Athena na década de 1980. Ele é bastante portátil e executa totalmente no espaço do usuário. Na origem, a ideia era que ele conectasse um grande número de terminais de usuários remotos com um servidor de computadores central, de maneira que ele é logicamente dividido em software cliente e software hospedeiro, que pode executar em diferentes computadores. Nos computadores

pessoais modernos, ambas as partes podem executar na mesma máquina. Nos sistemas Linux, os populares ambientes Gnome e KDE executam sobre o X.

. Servidor X: Quando o X está executando em uma máquina, o software que coleta a entrada do teclado e mouse e escreve a saída para a tela é chamado de servidor X. Ele tem de controlar qual janela está atualmente ativa (onde está o ponteiro do mouse), de maneira que ele sabe para qual cliente enviar qualquer nova entrada do teclado. Ele se comunica com programas em execução (possível sobre uma rede) chamados clientes X. Ele lhes envia entradas do teclado e mouse e aceita comandos da tela deles. Resumindo, o servidor é o responsável por controlar o hardware gráfico e gerenciar a exibição das janelas na tela. Ele lida com as interações com o monitor, o teclado, o mouse e outros dispositivos de entrada. O servidor X "entende" os comandos gráficos que são emitidos pelos aplicativos (clientes), e exibe as janelas no monitor.

. Cliente X: Os clientes são os programas que solicitam ao servidor a exibição de gráficos e interações com o usuário. Cada aplicação gráfica (como um navegador, editor de texto, etc.) é um cliente X. O cliente X comunica-se com o servidor para desenhar janelas, gerenciar entradas do teclado e mouse, etc.

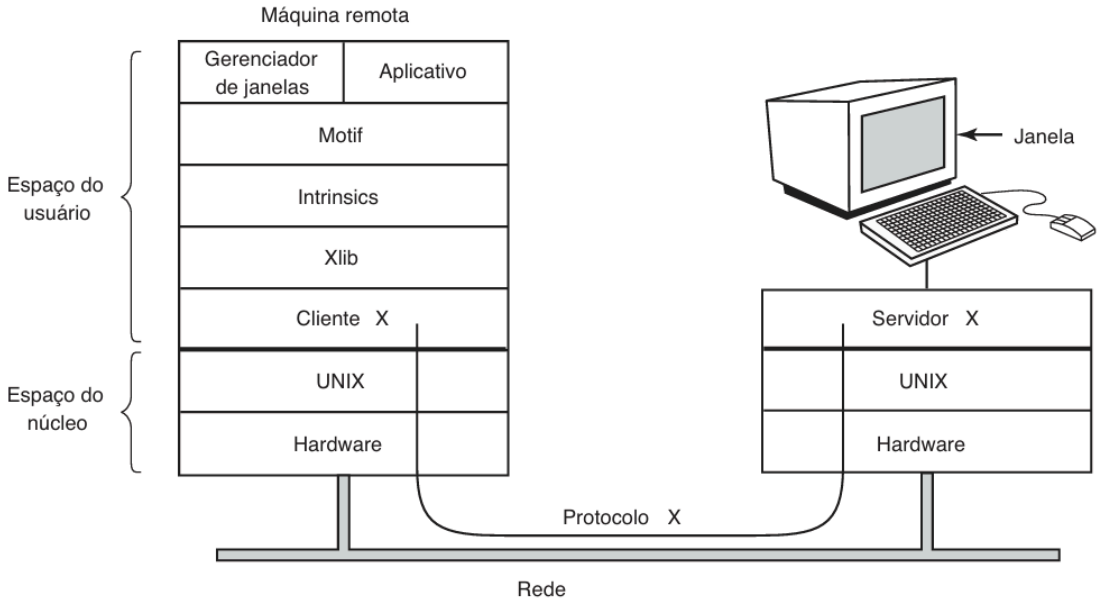
O arranjo do cliente e servidor é mostrado na Figura 14 para o caso em que o cliente X e o servidor X estão em máquinas diferentes. Mas quando executa Gnome ou KDE em uma única máquina, o cliente é apenas algum programa de aplicação usando a biblioteca X falando com o servidor X na mesma máquina (mas usando uma conexão TCP através de soquetes, a mesma que ele usaria no caso remoto).

. Protocolo X: A razão de ser possível executar o sistema X Window no UNIX (ou outro sistema operacional) em uma única máquina ou através de uma rede é o fato de que o que o X realmente define é o protocolo X entre o cliente X e o servidor X, como mostrado na Figura 14. O protocolo X permite a comunicação entre o servidor e os clientes. Ele pode transmitir comandos como "desenhar um botão" ou "mover uma janela", e também pode capturar eventos de entrada, como "clique do mouse" ou "pressionamento de tecla". Não importa se o cliente e o servidor estão na mesma máquina, separados por 100 metros sobre uma rede de área local, ou distantes milhares de quilômetros um do outro e conectados pela internet. O protocolo e a operação do sistema são idênticos em todos os casos.

. Biblioteca: O X é apenas um sistema de gerenciamento de janelas. Ele não é uma GUI completa. Para obter uma GUI completa, outras camadas de software são executadas sobre ele. Uma camada é a Xlib, um conjunto de rotinas de biblioteca para acessar a funcionalidade do X. Essas rotinas formam a base do Sistema X Window e serão examinadas a seguir, embora sejam primitivas demais para a maioria dos programas de usuário acessar diretamente. Por exemplo, cada clique de mouse é relatado em separado, então determinar que dois cliques realmente formem um clique duplo deve ser algo tratado acima da Xlib.

. Toolkits: Para tornar a programação com X mais fácil, um toolkit consistindo em Intrinsics é fornecido como parte do X. Essa camada gerencia botões, barras de rolagem e outros elementos da GUI chamados widgets. Para produzir uma verdadeira interface GUI, com aparência e sensação uniformes, outra camada é necessária (ou várias delas). Um exemplo é o Motif, mostrado na Figura 14 abaixo, que é a base do *Common Desktop Environment* (Ambiente de Desktop Comum) usado no Solaris e outros sistemas UNIX comerciais. A maioria das aplicações faz uso de chamadas para o Motif em vez da Xlib. Gnome e KDE têm uma estrutura similar à da Figura 14, apenas com bibliotecas diferentes. Gnome usa a biblioteca GTK+ e KDE usa a biblioteca Qt.

Figura 14:



. Gerenciador de janelas: vale a pena observar que o gerenciamento de janela não é parte do X em si. A decisão de deixá-lo de fora foi absolutamente intencional. Em vez disso, um processo de cliente X separado, chamado de

gerenciador de janela (como Metacity, i3, Openbox, KDE KWin), controla a criação, remoção e movimento das janelas na tela. Para gerenciar as janelas, ele envia comandos para o servidor X dizendo-lhe o que fazer. Ele muitas vezes executa na mesma máquina que o cliente X, mas na teoria pode executar em qualquer lugar.

. Estações de Trabalho (*Workstations*): Agora será examinado brevemente o X como visto do nível Xlib. Quando um programa X inicializa, ele abre uma conexão para um ou mais servidores X — estações de trabalho (*workstations*) — , embora possam ser colocados na mesma máquina que o próprio programa X. Este considera a conexão confiável no sentido de que mensagens perdidas e duplicadas são tratadas pelo software de rede e ele não tem de se preocupar com erros de comunicação.

Quatro tipos de mensagens trafegam pela conexão:

1. Comandos gráficos do programa para a estação de trabalho;
2. Respostas da estação de trabalho a perguntas do programa;
3. Teclado, mouse e outros avisos de eventos;
4. Mensagens de erro.

A maioria dos comandos gráficos é enviada do programa para a estação de trabalho como mensagens de uma só via. Não é esperada uma resposta. A razão para esse design é que, quando os processos do cliente e do servidor estão em máquinas diferentes, pode ser necessário um período de tempo substancial para o comando chegar ao servidor e ser executado. Bloquear o programa de aplicação durante esse tempo o atrasaria desnecessariamente. Por outro lado, quando o programa precisa de informações da estação de trabalho, basta-lhe esperar até que a resposta retorne.

X é altamente impelido por eventos. Eventos fluem da estação de trabalho para o programa, em geral como resposta a alguma ação humana como teclas digitadas, movimentos do mouse, ou uma janela sendo aberta. Cada mensagem de evento tem 32 bytes, com o primeiro byte fornecendo o tipo do evento e os 31 bytes seguintes fornecendo informações adicionais. Várias dúzias de tipos de eventos existem, mas a um programa é enviado somente aqueles eventos que ele disse que está disposto a manejar. Por exemplo, se um programa não quer ouvir falar de liberação de teclas, ele não recebe quaisquer eventos relacionados ao assunto. Como no Windows, os eventos entram em filas, e os programas leem os eventos da fila de entrada. No entanto, diferentemente do Windows, o sistema operacional

jamais chama sozinho rotinas dentro do programa de aplicação por si próprio. Ele não faz ideia de qual rotina lida com qual evento.

. Recurso(Resource): Refere-se a objetos gerenciados pelo servidor X que são criados e manipulados pelos aplicativos clientes .Um recurso é uma estrutura de dados que contém determinadas informações. Programas de aplicação criam recursos em estações de trabalho. Recursos podem ser compartilhados entre múltiplos processos na estação de trabalho. Eles tendem a ter vidas curtas e não sobrevivem às reinicializações das estações de trabalho. Recursos típicos incluem janelas, fontes, mapas de cores (paletas de cores), pixmaps (mapas de bits), cursores e contextos gráficos. Cada recurso tem um identificador único, permitindo que os programas interajam com eles de maneira eficiente.

- **Interfaces gráficas do usuário**

A maioria dos computadores pessoais oferece uma interface gráfica de usuário (*Graphical User Interface* — GUI).

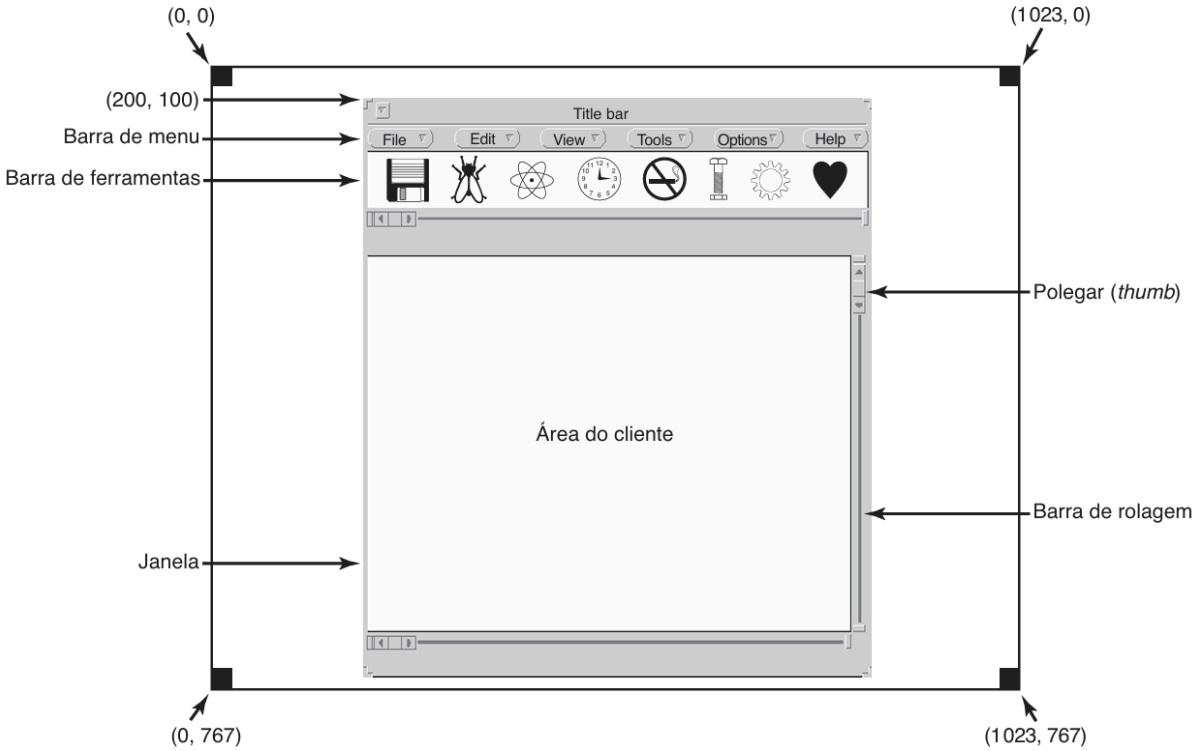
. WIMP: A GUI tem quatro elementos essenciais, denotados pelos caracteres WIMP. Essas letras representam *Windows, Icons, Menus* e *Pointing device* (Janelas, Ícones, Menus e Dispositivo apontador, respectivamente). As janelas são blocos retangulares de área de tela usados para executar programas. Os ícones são pequenos símbolos que podem ser clicados para fazer que determinada ação aconteça. Os menus são listas de ações das quais uma pode ser escolhida. Por fim, um dispositivo apontador é um mouse, trackball, ou outro dispositivo de hardware usado para mover um cursor pela tela para selecionar itens.

. Adaptador gráfico: A entrada de dados para sistemas GUI ainda usa o teclado e o mouse, mas a saída quase sempre vai para um dispositivo de hardware especial chamado adaptador gráfico. Um adaptador gráfico contém uma memória especial chamada RAM de vídeo que armazena as imagens que aparecem na tela. Adaptadores gráficos muitas vezes têm poderosas CPUs de 32 ou 64 bits e até 4GB de sua própria RAM, separada da memória principal do computador. Cada adaptador gráfico suporta um determinado número de tamanhos de telas, possuindo ou a proporção 4:3 ou a 16:9. A uma resolução de apenas 1920 × 1080 (o tamanho dos vídeos HD inteiros), uma tela colorida com 24 bits por pixel exige em torno de 6,2 MB de RAM apenas para conter a imagem, então com 256 MB ou mais, o adaptador gráfico pode conter muitas imagens ao mesmo tempo. Se a tela inteira for

renovada 75 vezes/s, a RAM de vídeo tem de ser capaz de fornecer dados continuamente a 445 MB/s.

. Janela: O item básico na tela é uma área retangular chamada de janela. A posição e o tamanho de uma janela são determinados exclusivamente por meio das coordenadas (em pixels) de dois vértices diagonalmente opostos. Uma janela pode conter uma barra de título, uma barra de menu, uma barra de ferramentas, uma barra de rolagem vertical e uma barra de rolagem horizontal. Uma janela típica é mostrada na Figura 15 abaixo. Observe que o sistema de coordenadas do Windows coloca a origem no vértice superior esquerdo e estabelece que o y aumenta para baixo, o que é diferente das coordenadas cartesianas usadas na matemática. Quando uma janela é criada, os parâmetros especificam se ela pode ser movida, redimensionada ou rolada (arrastando a trava deslizante da barra de rolagem) pelo usuário. A principal janela produzida pela maioria dos programas pode ser movida, redimensionada e rolada, o que tem enormes consequências para a maneira como os programas do Windows são escritos. Em particular, os programas precisam ser informados a respeito de mudanças no tamanho de suas janelas e devem estar preparados para redesenhar os conteúdos de suas janelas a qualquer momento, mesmo quando menos esperarem por isso. Como consequência, os programas do Windows são orientados a mensagens. As ações do usuário envolvendo o teclado ou o mouse são capturadas pelo Windows e convertidas em mensagens para o programa proprietário da janela sendo endereçada. Cada programa tem uma fila de mensagens para a qual as mensagens relacionadas a todas as suas janelas são enviadas. O principal laço do programa consiste em pescar a próxima mensagem e processá-la chamando uma rotina interna para aquele tipo de mensagem. Em alguns casos, o próprio Windows pode chamar essas rotinas diretamente, ignorando a fila de mensagens. Esse modelo é bastante diferente do código procedimental do modelo UNIX que faz chamadas de sistema para interagir com o sistema operacional. X, no entanto, é orientado a eventos.

Figura 15:



. Interface do dispositivo gráfico (GDI — *Graphics Device Interface*): A ação de desenhar para a tela é manejada por um pacote que consiste em centenas de rotinas que são reunidas para formar a interface do dispositivo gráfico. Ela pode lidar com texto e gráficos e é projetada para ser independente de plataformas e dispositivos. Antes que um programa possa desenhar (isto é, pintar) em uma janela, ele precisa adquirir um contexto do dispositivo, que é uma estrutura de dados interna contendo propriedades da janela, como a fonte, cor do texto, cor do segundo plano, e assim por diante. A maioria das chamadas para a GDI usa o contexto de dispositivo, seja para desenhar ou para obter ou ajustar as propriedades.

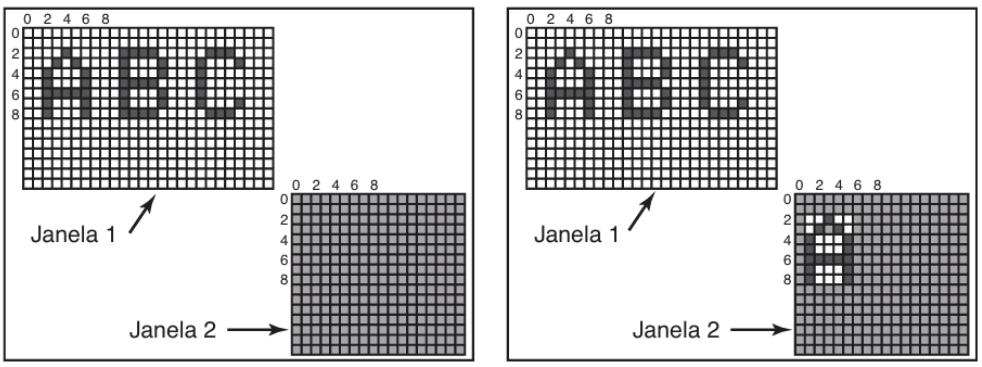
As rotinas GDI são exemplos de gráficos vetoriais. Eles são usados para colocar figuras geométricas e texto na tela. Podem ser escalados facilmente para telas maiores ou menores (desde que o número de pixels na tela seja o mesmo). Eles também são relativamente independentes do dispositivo. Uma coleção de chamadas para rotinas GDI pode ser reunida em um arquivo que consiga descrever um desenho complexo. Esse arquivo é chamado de um meta-arquivo do Windows, e é amplamente usado para transmitir desenhos de um programa do Windows para outro. Esses arquivos têm uma extensão .wmf. Muitos programas do Windows permitem que o usuário copie (parte de) um desenho e o coloque na área de transferência do Windows. O usuário pode então ir a outro programa e colar os

conteúdos da área de transferência em outro documento. Uma maneira de efetuar isso é fazer que o primeiro programa represente o desenho como um meta-arquivo do Windows e o coloque na área de transferência no formato .wmf.

- **Mapas de bits (*bitmaps*)**

Nem todas as imagens que os computadores manipulam podem ser geradas usando gráficos vetoriais. Fotografias e vídeos, por exemplo, não usam gráficos vetoriais. Em vez disso, esses itens são escaneados sobrepondo-se uma grade na imagem. Os valores médios do vermelho, verde e azul de cada quadrante da grade são então amostrados e salvos como o valor de um pixel. Esse arquivo é chamado de mapa de bits (*bitmap*). Outro uso para os bitmaps é para o texto. Uma maneira de representar um caractere em particular em alguma fonte é um pequeno bitmap. Acrescentar texto à tela então torna-se uma questão de movimentar bitmaps. Uma maneira geral de usar os mapas de bits é através de uma rotina chamada BitBlt. Em sua forma mais simples, ela copia um mapa de bits de um retângulo em uma janela para um retângulo em outra janela (ou a mesma). Um problema com mapas de bits é que eles não são extensíveis. Um caractere que está em uma caixa de 8 × 12 em uma tela de 640 × 480 parecerá razoável. No entanto, se esse mapa de bits for copiado para uma página impressa de 1200 pontos/polegada, o que é 10.200 bits × 13200 bits, a largura do caractere (8 pixels) será de 8/1200 polegadas ou 0,17 mm. Além disso, copiar entre dispositivos com propriedades de cores diferentes ou entre monocromático e colorido não funciona bem. Por essa razão, o Windows também suporta uma estrutura de dados chamada mapa de bits independente de dispositivo (*Device Independent Bitmap* — DIB). Arquivos usando esse formato usam a extensão .bmp. Eles têm cabeçalhos de arquivo e informação e uma tabela de cores antes dos pixels. Essa informação facilita a movimentação de mapas de bits entre dispositivos diferentes.

Figura 16:



- **Fontes**

Nas versões do Windows antes do 3.1, os caracteres eram representados como mapas de bits e copiados na tela ou impressora usando BitBlt. O problema com isso é que um mapa de bits que faz sentido na tela é pequeno demais para a impressora. Também, um mapa de bits é necessário para cada caractere em cada tamanho. Em outras palavras, dado o mapa de bits para A no padrão de 10 pontos, não há como calculá-lo para o padrão de 12 pontos. Uma vez que podem ser necessários todos os caracteres de todas as fontes para tamanhos que vão de 4 pontos a 120 pontos, um vasto número de bitmaps era necessário. O sistema inteiro era simplesmente inadequado demais para texto. A solução foi a introdução das fontes TrueType, que não são bitmaps, mas contornos de caracteres. Cada caractere TrueType é definido por uma sequência de pontos em torno do seu perímetro. Todos os pontos são relativos à origem (0, 0). Usando esse sistema, é fácil colocar os caracteres em uma escala crescente ou decrescente. Tudo o que precisa ser feito é multiplicar cada coordenada pelo mesmo fator da escala. Dessa maneira, um caractere TrueType pode ser colocado em uma escala para cima ou para baixo até qualquer tamanho de pontos, mesmo tamanhos de pontos fracionais. Uma vez no tamanho adequado, os pontos podem ser conectados usando o conhecido algoritmo ligue-os-pontos (follow--the-dots). Após o contorno ter sido concluído, o caractere pode ser preenchido. Assim que o caractere preenchido estiver disponível em forma matemática, ele pode ser varrido, isto é, convertido em um mapa de bits para qualquer que seja a resolução desejada. Ao colocar primeiro em escala e então varrer, será garantido de que os caracteres exibidos na tela ou impressos na impressora serão o mais próximos quanto possível, diferenciando-se somente na precisão do erro. Para melhorar ainda mais a qualidade, é possível embutir dicas em cada caractere dizendo como realizar a varredura (*hinting*). Por exemplo, o acabamento das pontas laterais no topo da letra T deve ser idêntico, algo que talvez não fosse o caso devido a um erro de arredondamento. As dicas melhoram a aparência final. É possível visualizar a criação dos contornos na Figura 17.

Figura 17:

7. CLIENTES MAGROS (*THIN CLIENTS*)

Thin clients (ou "clientes magros") são dispositivos de computação leve com configurações mínimas de hardware, projetados para depender quase inteiramente de um servidor central (como *mainframes* ou ambientes em nuvem) para processamento, armazenamento e execução de aplicativos. Eles funcionam essencialmente como terminais de acesso remoto, com capacidade de processamento local limitada ou inexistente.

7.1 Evolução

1. **Origens nos Mainframes e Terminais Burros (1960–1980)**

Na década de 1960, a computação era centralizada em *mainframes*, computadores de grande porte que atendiam múltiplos usuários simultaneamente. Os terminais burros eram dispositivos conectados a esses mainframes, servindo apenas para exibir informações e enviar alguns comandos, sem capacidade de processamento própria.

2. **Era do PC e Descentralização (1980–1990)**

A partir dos anos 1980, os *Personal Computers* (PCs) começaram a substituir os terminais burros (monitor e teclado). Com o lançamento do IBM PC (1981) e do Apple Macintosh, o processamento passou a ser feito localmente, eliminando a dependência de servidores centrais (mainframes). Essa descentralização melhorou

a experiência do usuário, mas tornou a administração por parte das empresas mais difícil e cara.

Figuras 18 e 19:

3. **Retorno da Centralização com Thin Clients (1990–2000)**

Na década de 1990, os desafios da descentralização levaram **à volta do modelo cliente-servidor**, agora com uma abordagem moderna: os **Thin Clients**. Que era a ideia de oferecer terminais com hardware mínimo, deixando o processamento para os servidores centralizados, reduzindo custos e facilitando a manutenção.

4. **Thin Clients Modernos (2000–Presente)**

Com o avanço da computação em nuvem e da virtualização, os Thin Clients evoluíram para dispositivos mais eficientes, sendo amplamente adotados em ambientes empresariais, call centers e instituições de ensino.

8. GERENCIAMENTO DE ENERGIA

Existem duas estratégias gerais para reduzir o consumo de energia. A primeira consiste no sistema operacional desligar partes do computador, principalmente os dispositivos de entrada e saída, que não estejam em uso, pois um dispositivo desligado usa pouca ou nenhuma energia. A segunda é um determinado aplicativo usar menos energia, possivelmente degradando a qualidade da experiência do usuário com objetivo de esticar o tempo de bateria de um notebook, por exemplo.

8.1 Modos

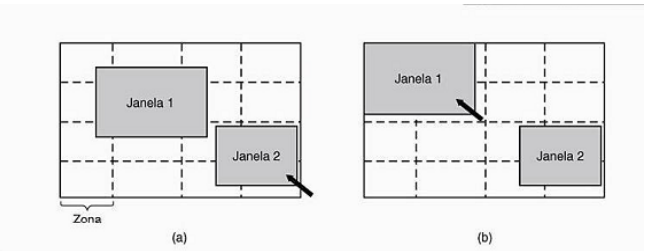
Os modos Suspender, Hibernar e Desligar são diferentes formas de gerenciar a energia de um computador.

- No modo **Suspender**, o sistema entra em um estado de baixa energia, mantendo a memória RAM ativa enquanto o processador, disco rígido e outros componentes são colocados em modo de baixo consumo ou desligados. Isso permite um retorno quase instantâneo ao uso
- Já no modo **Hibernar**, o estado atual do sistema, incluindo os programas e arquivos abertos, é salvo no disco rígido ou SSD, permitindo que o computador seja desligado completamente sem perder o progresso. Dessa forma, ao ligá-lo novamente, o sistema retorna exatamente ao ponto onde estava.
- Por fim, o modo **Desligar** encerra todos os processos e programas, finaliza as tarefas do sistema operacional e desliga completamente o computador. Ao religá-lo, o sistema começa do zero.

8.2 Estratégias

- **Nos monitores**, é possível criar zonas que podem ser ativadas ou desativadas individualmente, economizando energia (solução apresentada em 2004).

Figura 20:



- Nos discos rígidos, a velocidade de rotação pode ser reduzida quando não estão em uso, ou parte da energia gerada pelo movimento do disco pode ser reaproveitada para reiniciá-lo.
- **Na CPU**, o consumo de energia pode ser diminuído por meio da redução do clock.
- **Nas memórias**, como a RAM, é possível entrar em estado de baixo consumo (dormência) quando ociosas.
- **Em notebooks,** para equilibrar temperatura e o consumo energético, o sistema operacional deve decidir entre ativar a ventoinha para resfriamento ou reduzir o desempenho da CPU e o brilho da tela.

- Nos modelos atuais, há uma comunicação entre a bateria e o SO, que monitora a voltagem e a carga restante. Assim, o sistema pode alertar o usuário sobre quando recarregar e estimar o tempo restante de uso antes do desligamento.

Figuras 21 e 22:

Exemplo de Terminal Burro

Figura 23:

Exemplo de *Thin Client* (Moderno)

9. CONCLUSÃO

A entrada/saída é um tópico importante, mas muitas vezes negligenciado. Uma fração substancial de qualquer sistema operacional diz respeito à E/S. Esse aspecto é essencial para o funcionamento eficiente de computadores e dispositivos, pois permite a comunicação entre o hardware e o software, garantindo que os dados sejam transferidos de maneira eficaz entre o processador, a memória e os periféricos. Conclui-se que a entrada/saída desempenha um papel crucial na experiência do usuário e na eficiência geral de um sistema computacional. Um gerenciamento inadequado pode resultar em gargalos de desempenho, tornando o sistema lento e ineficiente. Por isso, compreender e aprimorar as técnicas de E/S é fundamental para desenvolvedores e administradores de sistemas que buscam otimizar o uso dos recursos computacionais.

REFERÊNCIAS:

- TANENBAUM, A. S. Sistemas operacionais modernos. 3. ed. São Paulo, SP: Pearson, 2009. E-book. Disponível em: https://plataforma.bvirtual.com.br. Acesso em: 30 mar. 2025.

- https://gcallah.github.io/OperatingSystems/IOHardware.html.Acesso em: 29 mar. 2025.