1.Objetivos do software de E/S

Um conceito importante no projeto de software de E/S é a ***independência de dispositivo***, isso significa que deve ser possível escrever programas que possam acessar qualquer dispositivo de E/S sem a necessidade de especificar o dispositivo antecipadamente.

Intimamente relacionado à independência de dispositivo é o objetivo da ***atribuição uniforme de nomes***. O nome de um arquivo ou de um dispositivo deve ser simplesmente uma *string* ou um inteiro e de modo algum deve depender do dispositivo. No UNIX e no MINIX 3, todos os discos podem ser integrados na hierarquia do sistema de arquivos de maneiras arbitrárias, de modo que o usuário não precisa saber qual o nome corresponde a qual dispositivo. Por exemplo, um disquete pode ser ***montado*** na raiz do diretório /usr/ast/backup, de modo que copiar um arquivo para esse diretório significa copiá-lo no disquete. Desse modo, todos os arquivos e dispositivos são endereçados da mesma maneira: por um nome de caminho.

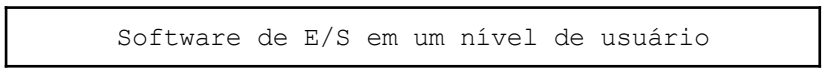
Outra questão importante para o software de E/S é o ***tratamento de erros***. Em geral, os erros devem ser tratados o mais próximo do hardware possível. Se a controladora descobre um erro de leitura, ela mesma deve tentar corrigi-lo, se puder. Se não puder, então o *driver* de dispositivo deverá tratar dele, talvez apenas tentando ler o bloco novamente. Muitos erros são passageiros, como os erros de leitura causados por partículas de poeira no cabeçote de leitura, e desaparecerão se a operação for repetida. Somente se as camadas mais baixas não forem capazes de lidar com o problema é que as camadas mais altas devem ser informadas.

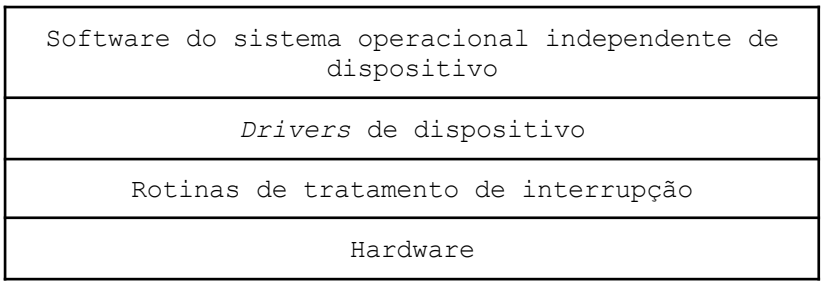
Outra questão importante são as transferências ***síncronas*** versus ***assíncronas***. A maior parte da E/S física é assíncrona (CPU). Outros programas de usuário são muito mais fáceis e vai fazer outra coisa, até a chegada da interrupção. Os programas de usuário são muito mais fáceis de escrever se as operações de E/S causam bloqueio. Cabe ao sistema operacional fazer com que as operações que, na verdade, são baseadas em interrupções, se pareçam com bloqueios para os programas de usuário.

Uma outra questão para o software de E/S é o uso de ***buffers***. Frequentemente, os dados provenientes de um dispositivo não podem ser armazenados diretamente em seu destino final. Além disso, alguns dispositivos têm restrições de tempo real severas de modo que os dados devem ser colocados antecipadamente em um *buffer* de saída para desvincular a velocidade com que o *buffer* é preenchido da velocidade com que ele é esvaziado, para evitar a falta de dados.

Alguns dispositivos que podem compartilhar versus dispositivos dedicados. Alguns dispositivos de E/S, como os discos, podem ser empregados por muitos usuários ao mesmo tempo. Nenhum problema é causado pelo fato de vários usuários terem arquivos abertos, no mesmo disco, ao mesmo tempo. Outros dispositivos, como as unidades de fita, precisam ser dedicados a um único usuário até que ele tenha terminado de usá-la.

Frequentemente, o software de E/S é organizado em quatro camadas, como se vê abaixo





2.Rotinas de tratamento de interrupção

As interrupções são uma realidade. Elas vem ser bem ocultadas no interior do sistema operacional, para que o mínimo possível do sistema saiba a seu respeito. A melhor maneira de ocultá-las é fazer com que o *driver* que inicia uma operação de E/S seja bloqueado até que E/S tenha terminado e a interrupção associada ocorra. O *driver* se bloquear sozinho, usando-se por exemplo, uma instrução *DOWN* em um semáforo, uma instrução *WAIT* em uma variável de condição, uma instrução *RECEIVE* em uma mensagem ou algo semelhante.

Quando a interrupção acontece, a função de tratamento faz o que for necessário para atendê-la. Então, ela pode desbloquear o *driver* que a iniciou. Em alguns casos, ela apenas completará uma instrução *UP* em um semáforo. Em outros, executará uma instrução *SIGNAL* em um a variável de condição em um monitor. Ainda, em outros casos, ela enviará uma mensagem para o *driver* bloqueado. Em todos os casos, o efeito geral da interrupção será que um *driver* que anteriormente estava bloqueado agora poderá executar. Esse modelo funciona melhor se os *drivers* forem estruturados como processos independentes, com seus próprios estados, pilhas e contadores de programa.

3.Drivers de dispositivos

Cada dispositivo de E/S ligado a um computador precisa de algum código específico do dispositivo pera controlá-lo. Esse código, chamado de ***driver de dispositivo***, geralmente é escrito pelo fabricante do dispositivo e distribuído junto com ele em um CD-ROM. Como cada sistema operacional precisa de seus próprios *drivers*, os fabricantes de dispositivos normalmente fornecem *drivers* para vários sistemas operacionais populares.

Normalmente, cada *driver* de dispositivo manipula um tipo ou uma classe de dispositivos intimamente relacionados. Por exemplo, seria uma boa ideia ter um único *driver* de mouse, mesmo que o sistema suporte várias marcas diferentes de mouse.

Para acessar o hardware do dispositivo, tradicionalmente, o *driver* de dispositivo faz parte do núcleo do sistema. Essa estratégia oferece o melhor desempenho e a pior confiabilidade, pois um erro em qualquer *driver* de dispositivo pode derrubar o sistema inteiro. O MINIX 3 diverge desse modelo para melhorar a confiabilidade.

A maioria dos sistemas operacionais define uma interface padrão que todos os *drivers* de bloco devem suportar e uma segunda interface padrão que todos os *drivers* de caractere devem suportar. Essas interfaces consistem em várias funções que o restante do sistema operacional pode chamar para fazer o *driver* trabalhar. Em termos gerais, a tarefa de um *driver* de dispositivo é aceitar requisições abstratas do software independente de dispositivo e cuidar para que as requisições sejam executadas. Uma requisição típica para *driver* de disco é

ler o bloco *n*. Se o *driver* estiver ocioso no momento da chegada de uma requisição, ele começará a executá-la imediatamente. Entretanto, se ele já estiver ocupado, normalmente colocará a nova requisição em uma fila de requisições pendentes para serem tratadas assim que forem possível.

O primeiro passo na execução de uma requisição de E/S é verificar se os parâmetros de entrada são válidos, e caso não sejam, retornar um erro. Se a requisição for válida, o próximo passo será transformá-la dos termos abstratos para concretos. Para um *driver* de disco, isso significa descobrir onde o bloco solicitado está realmente no disco, verificar se o motor da unidade de disco está funcionando, determinar se o braço está posicionado no cilindro correto e etc. Em resumo, o *driver* deve decidir quais operações da controladora são exigidas e em que sequência.

Uma vez que o *driver* tiver determinado quais comandos deve enviar para a controladora, ele começará a executá-los, escrevendo nos registradores de dispositivo da controladora. As controladoras simples podem manipular apenas um comando por vez. As controladoras mais sofisticadas aceitam uma lista encadeada de comandos, os quais serão executados sem a intervenção do sistema operacional.

Após o(s) comando(s) ter(em) sido executado(s), ocorre uma de duas situações. Em muitos casos, o *driver* de dispositivo deve esperar até que a controladora realize algum trabalho para ele, logo, ele bloqueia a si mesmo até a interrupção então para desbloqueá-lo. Em outros casos, entretanto, a operação é executada rapidamente, de modo que o *driver* não precisa ser bloqueado.

No primeiro caso, o *driver* será desbloqueado pela ocorrência da interrupção. No segundo caso, ele nunca será bloqueado. De qualquer modo, após a operação ter terminado, ele deve verificar a existência de erros. Se tudo estiver correto, o *driver* poderá ter dados para passar para o software independente do dispositivo. Finalmente, ele retorna algumas informações de status para informar situações de erros, ou não, para quem o chamou. Se houver outras requisições enfileiradas, agora uma delas poderá ser selecionada e iniciada. Se nada estiver enfileirado, o *driver* será bloqueado e ficará aguardando a próxima requisição.

4.Software de E/S independete de dispositivo

Embora um trecho do software de E/S seja específico a um dispositivo, uma grande parte dele é independente deste. O limite exato entre os *drivers* e o software independente de dispositivo depende do sistema, pois algumas funções que poderiam ser executadas de maneira independente de dispositivos podem, na verdade, por eficiência ou outros motivos, serem executadas nos *drivers*.

No MINIX 3, a maioria do software independente de dispositivo faz parte do sistema de arquivos. A função básica do software independente de dispositivo é executar as funções de E/S comuns a todos os dispositivos e fornecer uma interface uniforme para o software em nível de usuário.

4.1 Interface uniforme para *drivers* de dispositivo

Um problema importante em um sistema operacional é como fazer todos os dispositivos e *drivers* de E/S parecerem mais ou menos iguais. Se discos, impressoras, monitores, teclados, etc., tiverem todos interfaces diferentes, sempre que aparecer um novo dispositivo periférico, o sistema operacional deverá ser modificado para esse novo dispositivo.

Com uma interface padrão é muito mais fácil de instalar um novo *driver*, desde que ele seja compatível com a interface existente. Isso também significa que os desenvolvedores de *drivers* sabem o que é esperado deles. Na prática, nem todos os dispositivos são absolutamente idênticos, mas normalmente existe apenas um pequeno número de tipos de dispositivos e mesmo esses geralmente são quase idênticos.

Outro aspecto do fato de ter uma interface uniforme é o modo como os dispositivos de E/S são nomeados. O software independente de dispositivos cuida do mapeamento de nomes de dispositivos simbólicos para o *driver* correto.

Por exemplo, no UNIX e no MINIX 3, um nome de dispositivo, como ***/dev/disk0***, especifica exclusivamente o *i-node* de um dispositivo especial e esse *i-node* contém o ***número principal do dispositivo***, que é usada para localizar o *driver* apropriado. O *i-node* também contém o ***número secundário do dispositivo***, que é passado como parâmetro para o *driver*, para especificar a unidade a ser lida ou escrita. Todos os dispositivos possuem números principais e secundários, e todos os *drivers* são acessados usando-se o número principal do dispositivo para selecionar o *driver*.

Intimamente relacionada com a atribuição de nomes está a proteção. Como o sistema impede que os usuários acessem dispositivos que não podem acessar? No UNIX, no MINIX 3 e também nas versões do Windows XP e para frente, os dispositivos aparecem no sistema de arquivos como objetos nomeados, o que significa que as regras de proteção normais para arquivos também se aplicam aos dispositivos de E/S. O administrador do sistema pode então configurar as permissões corretas para cada dispositivo.

4.2 Uso de *buffers*

O uso de *buffers* também é um problema tanto para dispositivos de bloco como para dispositivos de caractere. Para dispositivos de bloco, o hardware geralmente insiste em ler e escrever blocos inteiros simultaneamente, mas os processos de usuário estão livres para ler e escrever em unidades arbitrárias. Se um processo de usuário escrever meio bloco, o sistema operacional normalmente manterá esses dados em memória até que sejam escritos os dados restantes, momento este em que o bloco irá para o disco.

4.3 Informe de erros

Os erros são muito mais comuns no contexto da E/S do que em qualquer outro. Quando eles ocorrerem, o sistema operacional precisa tratar deles da melhor forma possível. Muitos erros são específicos do dispositivo, logo, apenas o *driver* sabe o que fazer. Um erro típico é causado por um bloco de disco que foi danificado e não pode mais ser lido. Após o *driver* ter tentado ler o bloco certo número de vezes, ele desiste e informa o software independente do dispositivo. O modo como o erro é tratado a partir daí é independente do dispositivo.

4.4 Alocando e liberando dispositivos dedicados

Alguns dispositivos, como gravadores de CD-ROM, só podem ser usados por um único processo em dado momento. Cabe ao sistema operacional examinar as requisições de utilização do dispositivo e aceitá-los ou rejeitá-los, dependendo do dispositivo solicitado estar disponível ou não. Uma maneira simples de tratar essas requisições é exigir que os processos executem operações *OPEN* diretamente nos arquivos especiais dos dispositivos. Se o dispositivo não estiver disponível, a operação *OPEN* falhará, fechando esse dispositivo dedicado, e então liberando-o.

4.5 Tamanho de bloco independente de dispositivo

Nem todos os discos têm o mesmo tamanho de setor. Cabe ao software independente de dispositivos ocultar esse fato e fornecer um tamanho de bloco uniforme para as camadas superiores. Desse modo, as camadas superiores só tratam com dispositivos abstratos, todos os quais utilizam o mesmo tamanho de bloco lógico, independente do tamanho do setor físico.

5.Software de E/S em espaço de usuário

Embora a maior parte do software de E/S esteja dentro do sistema operacional, uma pequena parte dele consiste em bibliotecas ligadas aos programas do usuário e até de programas inteiros executados fora do espaço de endereçamento do núcleo. As chamadas de sistema, incluindo as de E/S, normalmente são feitas por funções de biblioteca. Quando um programa em C contiver a chamada: ***count = write(fd, buffer, nbytes)***, a função de biblioteca *write* será ligada ao código-objeto do usuário e contida no programa binário presente na memória no momento da execução. O conjunto de todas essas funções de biblioteca claramente faz parte de E/S.

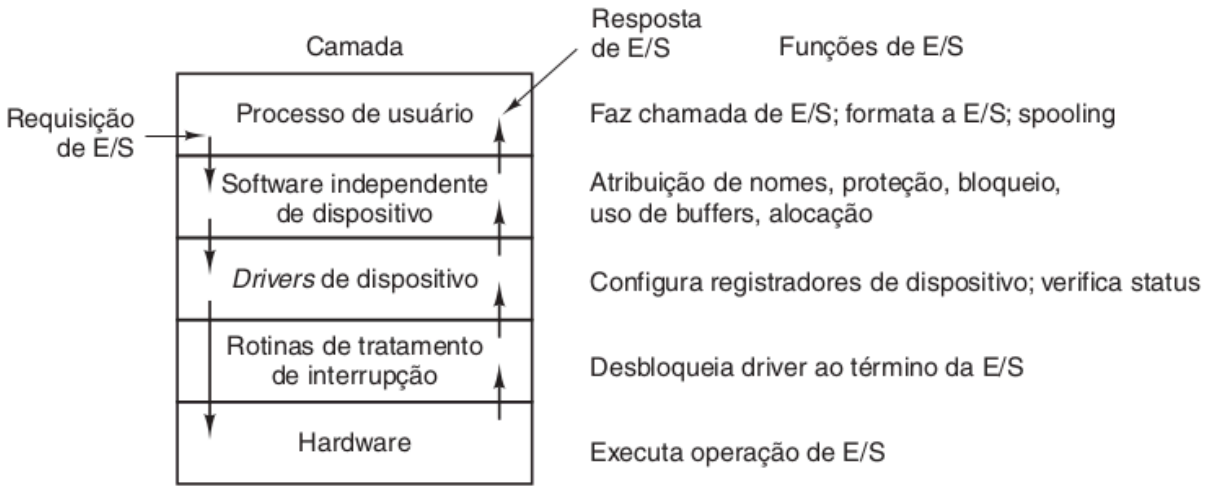
Embora essas funções faça pouco mais do que colocar seus parâmetros no lugar apropriado da chamada de sistema, existem outras funções de E/S que executam um trabalho real adicional. Em particular, a formatação da entrada e saída é feita por funções de biblioteca. Um exemplo da linguagem C é ***printf***, que recebe como entrada uma *string* de formato e possivelmente algumas variáveis, constrói uma *string* em ASCII e, então, chama *WRITE* para enviar a *string* para a saída.

Nem todo software de E/S em nível de usuário consiste em funções de biblioteca. Outra categoria importante é o sistema de ***spooling***. O *spool* é uma maneira de tratar com dispositivos de E/S dedicados em um sistema de multiprogramação. Considere um dispositivo típico com ***spool***, uma impressora. Embora seja simples permitir que qualquer processo de usuário abra o arquivo de caracteres especial que corresponde a impressora, suponha que ele o abrisse e depois não fizesse mais nada por várias horas. Nenhum outro processo poderia imprimir nada.

Em vez disso, é criado um processo especial, chamado ***daemon***, em um diretório especial, o diretório ***spool***. Para imprimir um arquivo, um processo primeiro gera o arquivo inteiro a ser impresso e o coloca no diretório de ***spool***. Cabe ao ***daemon***, que é o único processo a ter permissão para usar o arquivo especial associado a impressora, imprimir os arquivos no diretório. Protegendo-se o arquivo especial contra o uso direto por parte dos

usuários, o problema de alguém deixá-lo aberto desnecessariamente por muito tempo é eliminado.

O ***spool*** não é utilizado apenas por impressoras, mas também em várias outras situações. Por exemplo, o correio eletrônico normalmente usa um ***daemon***. Quando uma mensagem é enviada, ela é na verdade colocada em um diretório de ***spool*** de correio eletrônico. Posteriormente, o ***daemon*** de correio tentará enviá-la realmente. Eventualmente, em um dado momento, pode não ser possível contratar o destinatário, nesse caso, o ***daemon*** deixa a mensagem de alguns instantes. O ***daemon*** também pode enviar um aviso para o remetente dizendo que o envio da mensagem foi adiado, ou, após um atraso de algumas horas, ou alguns dias, que a mensagem não pôde ser entregue. Tudo isso se dá fora do sistema operacional.



Se o bloco necessário não estiver lá, ele chama o *driver* de dispositivo para enviar a requisição para o hardware, para obtê-lo do disco. Então, o processo é bloqueado até que a operação de disco tenha terminado. Quando a operação de disco tiver terminado, o hardware gerará uma interrupção. A rotina de tratamento de interrupção será executada para descobrir o que aconteceu, isto é, qual dispositivo quer ser atendido imediatamente. Então, ela extrai o status do dispositivo e desbloqueia o processo que estava esperando a conclusão da operação de E/S para permitir que o processo de usuário continue sua execução.