Entrada e Saída

O SO deve emitir comandos para os dispositivos, interceptar interrupções e lidar com erros. Também deve fornecer uma interface entre os dispositivos e o resto do sistema que seja simples e fácil de usar. Na medida do possível, a interface deve ser a mesma para todos os dispositivos (independentemente do dispositivo). O código de E/S representa uma fração significativa do sistema operacional total. Como o sistema operacional gerencia a E/S é o assunto deste capítulo.

Princípios do hardware de E/S

Nosso interesse é saber como o hardware é programado, não como ele funciona por dentro. Não obstante isso, a programação de muitos dispositivos de E/S está muitas vezes intimamente ligada à sua operação interna.

Dispositivos de E/S

Dispositivos de E/S podem ser divididos de modo geral em duas categorias: dispositivos de blocos e dispositivos de caractere. O primeiro armazena informações em blocos de tamanho fixo, cada um com seu próprio endereço. Tamanhos de blocos comuns variam de 512 a 65.536 bytes. Todas as transferências são em unidades de um ou mais blocos inteiros (consecutivos). A propriedade essencial de um dispositivo de bloco é que cada bloco pode ser lido ou escrito independentemente de todos os outros. Discos rígidos, discos Blu-ray e pendrives são dispositivos de bloco comuns.

O outro dispositivo de E/S é o de caractere. Um dispositivo de caractere envia ou aceita um fluxo de caracteres, desconsiderando qualquer estrutura de bloco. Ele não é endereçável e não tem qualquer operação de busca. Impressoras, interfaces de rede, mouses (para apontar), ratos (para experimentos de psicologia em laboratórios) e a maioria dos outros dispositivos que não são parecidos com discos podem ser vistos como dispositivos de caracteres.

Esse esquema de classificação não é perfeito. Alguns dispositivos não se enquadram nele. Relógios, por exemplo, não são endereçáveis por blocos. Tampouco geram ou aceitam fluxos de caracteres. Tudo o que fazem é causar interrupções em intervalos bem definidos.

Controlador de dispositivos

Unidades de E/S consistem, em geral, de um componente mecânico e um componente eletrônico. É possível separar as duas porções para permitir um projeto mais modular e geral. O componente eletrônico é chamado de controlador do dispositivo ou adaptador. Em computadores pessoais, ele muitas vezes assume a forma de um chip na placa-mãe ou um cartão de circuito impresso que pode ser inserido em um slot de expansão (PCIe). O componente mecânico é o dispositivo em si.

A interface entre o controlador e o dispositivo muitas vezes é de nível muito baixo. Um disco, por exemplo, pode ser formatado com 2 milhões de setores de 512 bytes por trilha. No entanto, o que realmente sai da unidade é um fluxo serial de bits, começando com um preâmbulo, então os 4096 bits em um setor, e por fim uma soma de verificação (checksum), ou código de correção de erro (ECC — Error Correcting Code). O preâmbulo é escrito quando o disco é formatado e contém o cilindro e número de setor, e dados similares, assim como informações de sincronização.

O trabalho do controlador é converter o fluxo serial de bits em um bloco de bytes, assim como realizar qualquer correção de erros necessária.

Comunicação com CPU

Realizada através de registradores de controladores
● fornecer e aceitar dados
● ligar e desligar
● descobrir o estado do dispositivo

Como a comunicação é realizada?
● Portas de E/S
● E/S mapeada em memória

E/S mapeada na memória

Cada controlador tem alguns registradores que são usados para comunicar-se com a CPU. Ao escrever nesses registradores, o sistema operacional pode comandar o dispositivo a fornecer e aceitar dados, ligar-se e desligar-se, ou de outra maneira realizar alguma ação. Ao ler a partir desses registradores, o sistema operacional pode descobrir qual é o estado do dispositivo, se ele está preparado para aceitar um novo comando e assim por diante.

A questão que surge então é como a CPU se comunica com os registradores de controle e também com os buffers de dados do dispositivo. Existem duas alternativas. Na primeira abordagem, para cada registrador de controle é designado um número de porta de E/S, um inteiro de 8 ou 16 bits. O conjunto de todas as portas de E/S formam o espaço de E/S, que é protegido de maneira que programas de usuário comuns não consigam acessá-lo (apenas o sistema operacional). Usando uma instrução de E/S especial como:

● IN REG, PORT //lê o valor da porta para a mem reg
● OUT PORT, REG //escreve para a porta o valor em reg
– Exemplo

IN R0, 4 //lê a porta 4 para o R0 na CPU
Diferente de
MOV R0, 4 //lê a palavra 4 na RAM para o R0 na CPU

A segunda abordagem, introduzida com o PDP-11, é mapear todos os registradores de controle no espaço da memória. Para cada registrador de controle é designado um endereço de memória único para o qual nenhuma memória é designada. Esse sistema é chamado de E/S mapeada na memória. Na maioria dos sistemas, os endereços designados estão no — ou próximos do — topo do espaço de endereçamento.

Em todos os casos, quando a CPU quer ler uma palavra, seja da memória ou de uma porta de E/S, ela coloca o endereço de que precisa nas linhas de endereçamento do barramento e então emite um sinal READ sobre uma linha de controle do barramento. Uma segunda linha de sinal é usada para dizer se o espaço de E/S ou o espaço de memória é necessário. Se for o espaço de memória, a memória responde ao pedido. Se for o espaço de E/S, o dispositivo de E/S responde ao pedido.

Com a E/S mapeada na memória, um driver do dispositivo de E/S pode ser escrito inteiramente em C. Sem a E/S mapeada na memória, é necessário algum código em linguagem de montagem.

Segundo, com a E/S mapeada na memória, nenhum mecanismo de proteção especial é necessário para evitar que processos do usuário realizem E/S. Tudo o que o sistema operacional precisa fazer é deixar de colocar aquela porção do espaço de endereçamento contendo os registros de controle no espaço de endereçamento virtual de qualquer usuário. Melhor ainda, se cada dispositivo tem os seus registradores de controle em uma página diferente do espaço de endereçamento, o sistema operacional pode dar a um usuário controle sobre dispositivos específicos, mas não dar a outros, ao simplesmente incluir as páginas desejadas em sua tabela de páginas. Esse esquema pode permitir que diferentes drivers de dispositivos sejam colocados em diferentes espaços de endereçamento, não apenas reduzindo o tamanho do núcleo, mas também impedindo que um driver interfira nos outros.

Terceiro, com a E/S mapeada na memória, cada instrução capaz de referenciar a memória também referencia os registradores de controle.

A E/S mapeada na memória também tem suas desvantagens. Primeiro, a maioria dos computadores hoje tem alguma forma de cache para as palavras de memória. O uso de cache para um registrador de controle do dispositivo seria desastroso.

Segundo, se houver apenas um espaço de endereçamento, então todos os módulos de memória e todos os dispositivos de E/S terão de examinar todas as referências de memória para ver quais devem ser respondidas por cada um. Se o computador tiver um único barramento, cada componente poderá olhar para cada endereço diretamente.

No entanto, a tendência nos computadores pessoais modernos é ter um barramento de memória de alta velocidade dedicado. O barramento é feito sob medida para otimizar o desempenho da memória, sem concessões para o bem de dispositivos de E/S lentos.

O problema de ter um barramento de memória separado em máquinas mapeadas na memória é que os dispositivos de E/S não têm como enxergar os endereços de memória

quando estes são lançados no barramento da memória, de maneira que eles não têm como responder.

Acesso direto à memória (DMA)

Não importa se uma CPU tem ou não E/S mapeada na memória, ela precisa endereçar os controladores dos dispositivos para poder trocar dados com eles. A CPU pode requisitar dados de um controlador de E/S um byte de cada vez, mas fazê-lo desperdiça o tempo da CPU, de maneira que um esquema diferente, chamado de acesso direto à memória (Direct Memory Access — DMA) é usado muitas vezes. Para simplificar a explicação, presumimos que a CPU acessa todos os dispositivos e memória mediante um único barramento de sistema que conecta a CPU, a memória e os dispositivos de E/S.

Quando o DMA é usado, o procedimento é diferente. Primeiro a CPU programa o controlador de DMA configurando seus registradores para que ele saiba o que transferir para onde. Ela também emite um comando para o controlador de disco dizendo para ele ler os dados do disco para o seu buffer interno e verificar a soma de verificação. Quando os dados que estão no buffer do controlador de disco são válidos, o DMA pode começar.

O controlador de DMA inicia a transferência emitindo uma solicitação de leitura via barramento para o controlador de disco.

A escrita na memória é outro ciclo de barramento-padrão. Quando a escrita está completa, o controlador de disco envia um sinal de confirmação para o controlador de DMA, também via barramento.

Os mais complexos podem ser programados para lidar com múltiplas transferências ao mesmo tempo. Esses controladores têm múltiplos conjuntos de registradores internamente, um para cada canal.

Cada transferência deve usar um controlador de dispositivos diferente. Após cada palavra ser transferida, o controlador de DMA decide qual dispositivo servir em seguida. Ele pode ser configurado para usar um algoritmo de alternância circular (round-robin), ou ter um esquema de prioridade projetado para favorecer alguns dispositivos em detrimento de outros.

Muitos barramentos podem operar em dois modos: modo uma palavra de cada vez (word-at-a-time mode) e modo bloco.

O controlador de DMA solicita a transferência de uma palavra e consegue. Se a CPU também quiser o barramento, ela tem de esperar. O mecanismo é chamado de roubo de ciclo, pois o controlador do dispositivo entra furtivamente e rouba um ciclo de barramento ocasional da CPU de vez em quando, atrasando-a ligeiramente.

No modo bloco, o controlador de DMA diz para o dispositivo para adquirir o barramento, emitir uma série de transferências, então libera o barramento. Essa forma de operação é chamada de modo de surto (burst). Ela é mais eficiente do que o roubo de ciclo, pois

adquirir o barramento leva tempo e múltiplas palavras podem ser transferidas pelo preço de uma aquisição de barramento. A desvantagem do modo de surto é que ele pode bloquear a CPU e outros dispositivos por um período substancial caso um surto longo esteja sendo transferido.

No modelo que estivemos discutindo, também chamado de modo direto (fly-by mode), o controlador do DMA diz para o controlador do dispositivo para transferir os dados diretamente para a memória principal. Um modo alternativo que alguns controladores de DMA usam estabelece que o controlador do dispositivo deve enviar a palavra para o controlador de DMA, que então emite uma segunda solicitação de barramento para escrever a palavra para qualquer que seja o seu destino.

A maioria dos controladores de DMA usa endereços físicos de memória para suas transferências. O uso de endereços físicos exige que o sistema operacional converta o endereço virtual do buffer de memória pretendido em um endereço físico e escreva esse endereço físico no registrador de endereço do controlador de DMA. Um esquema alternativo usado em alguns controladores de DMA é em vez disso escrever o próprio endereço virtual no controlador de DMA. Então o controlador de DMA deve usar a unidade de gerenciamento de memória (Memory Management Unit — MMU) para fazer a tradução de endereço virtual para físico.

Mencionamos anteriormente que o disco primeiro lê dados para seu buffer interno antes que o DMA possa ser inicializado. Por que ele precisa de um buffer interno? Há duas razões.

Primeiro, ao realizar armazenamento interno, o controlador de disco pode conferir a soma de verificação antes de começar uma transferência. Se a soma de verificação estiver incorreta, um erro é sinalizado e nenhuma transferência é feita.

A segunda razão é que uma vez inicializada uma transferência de disco os bits continuam chegando do disco a uma taxa constante, não importa se o controlador estiver pronto para eles ou não. Se o controlador tentasse escrever dados diretamente na memória, ele teria de acessar o barramento do sistema para cada palavra transferida. Se o barramento estivesse ocupado por algum outro dispositivo usando-o (por exemplo, no modo surto), o controlador teria de esperar. Se a próxima palavra de disco chegasse antes que a anterior tivesse sido armazenada, o controlador teria de armazená-la em outro lugar. Se o barramento estivesse muito ocupado, o controlador poderia terminar armazenando um número considerável de palavras e tendo bastante gerenciamento para fazer também. Quando o bloco é armazenado internamente, o barramento não se faz necessário até que o DMA comece.

Nem todos os computadores usam DMA. O argumento contra ele é que a CPU principal muitas vezes é muito mais rápida do que o controlador de DMA e pode fazer o trabalho muito mais rápido (quando o fator limitante não é a velocidade do dispositivo de E/S).

Interrupções revisitadas

Quando um dispositivo de E/S termina o trabalho dado a ele, gera uma interrupção (presumindo que as interrupções tenham sido habilitadas pelo sistema operacional). Ele faz isso enviando um sinal pela linha de barramento à qual está associado. Esse sinal é detectado pelo chip controlador de interrupções na placa-mãe, que então decide o que fazer.

Se nenhuma outra interrupção estiver pendente, o controlador de interrupção processa a interrupção imediatamente. No entanto, se outra interrupção estiver em andamento, ou outro dispositivo tiver feito uma solicitação simultânea em uma linha de requisição de interrupção de maior prioridade no barramento, o dispositivo é simplesmente ignorado naquele momento.

O sinal de interrupção faz a CPU parar aquilo que ela está fazendo e começar outra atividade. O número nas linhas de endereço é usado como um índice em uma tabela chamada de vetor de interrupções para buscar um novo contador de programa. Esse contador de programa aponta para o início da rotina de tratamento da interrupção correspondente. Em geral, interrupções de software (traps ou armadilhas) e de hardware usam o mesmo mecanismo desse ponto em diante, muitas vezes compartilhando o mesmo vetor de interrupções. A localização do vetor de interrupções pode ser estabelecida fisicamente na máquina ou estar em qualquer lugar na memória, com um registrador da CPU (carregado pelo sistema operacional) apontando para sua origem.

No mínimo, o contador do programa deve ser salvo, de maneira que o processo interrompido possa ser reiniciado.

Interrupções precisas e imprecisas

Considere o modelo de pipeline. O que acontece se uma interrupção ocorrer enquanto o pipeline está cheio (o caso usual)? Muitas instruções estão em vários estágios de execução. Quando ocorre a interrupção, o valor do contador do programa pode não refletir o limite correto entre as instruções executadas e as não executadas. Na realidade, muitas instruções talvez tenham sido parcialmente executadas, com diferentes instruções estando mais ou menos completas.

Em uma máquina superescalar, as coisas são ainda piores. As instruções podem ser decompostas em micro-operações e estas podem ser executadas fora de ordem, dependendo da disponibilidade dos recursos internos, como unidades funcionais e registradores. No momento de uma interrupção, algumas instruções enviadas há muito tempo talvez não tenham sido iniciadas e outras mais recentes talvez estejam quase concluídas. No momento em que uma interrupção é sinalizada, pode haver muitas instruções em vários estados de completude, com uma relação menor entre elas e o contador do programa.

Uma interrupção que deixa a máquina em um estado bem definido é chamada de uma interrupção precisa. Uma interrupção assim possui quatro propriedades:

- PC é salvo em um lugar conhecido

● Todas as instruções anteriores ao PC foram completadas
● Nenhuma instrução posterior ao PC foi concluída
● Estado da execução da instrução apontada pelo PC é conhecido

Uma interrupção que não atende a essas exigências é chamada de interrupção imprecisa e dificulta bastante a vida do projetista do sistema operacional.

Isso gera a situação irônica de termos CPUs superescalares muito rápidas sendo, às vezes, inadequadas para o trabalho em tempo real por causa das interrupções lentas.

Alguns computadores são projetados de maneira que alguns tipos de interrupções de software e de hardware são precisos e outros não. Algumas máquinas têm um bit que pode ser configurado para forçar que todas as interrupções sejam precisas. A desvantagem de se configurar esse bit é que ele força a CPU a registrar cuidadosamente tudo o que ela está fazendo e manter cópias de proteção dos registradores a fim de poder gerar uma interrupção precisa a qualquer instante. Toda essa sobrecarga tem um impacto importante sobre o desempenho.

Se interrupções precisas não fossem necessárias para fins de compatibilidade com versões antigas, essa área de chip seria disponível para caches maiores dentro do chip, tornando a CPU mais rápida. Por outro lado, interrupções imprecisas tornam o sistema operacional muito mais complicado e lento, então é difícil dizer qual abordagem é realmente melhor.

Princípios do software de E/S

Objetivos do software de E/S

Independência do dispositivo: Um conceito fundamental no projeto de software de E/S é conhecido como independência de dispositivo. O que isso significa é que devemos ser capazes de escrever programas que podem acessar qualquer dispositivo de E/S sem ter de especificá-lo antecipadamente. Por exemplo, um programa que lê um arquivo como entrada deve ser capaz de ler um arquivo em um disco rígido, um DVD ou em um pen-drive sem ter de ser modificado para cada dispositivo diferente. Fica a cargo do sistema operacional cuidar dos problemas causados pelo fato de que esses dispositivos são realmente diferentes e exigem sequências de comando muito diferentes para ler ou escrever.

Nomeação uniforme: Um objetivo muito relacionado com a independência do dispositivo é a nomeação uniforme. O nome de um arquivo ou um dispositivo deve simplesmente ser uma cadeia de caracteres ou um número inteiro e não depender do dispositivo de maneira alguma.

Tratamento de erros: Em geral, erros devem ser tratados o mais próximo possível do hardware. Se o controlador descobre um erro de leitura, ele deve tentar corrigi-lo se puder. Se ele não puder, então o driver do dispositivo deverá lidar com ele, talvez simplesmente tentando ler o bloco novamente. Muitos erros são transitórios, como erros de leitura

causados por grãos de poeira no cabeçote de leitura, e muitas vezes desaparecerão se a operação for repetida.

Transferências síncronas vs. assíncronas: Ainda outra questão importante é a das transferências síncronas (bloqueantes) versus assíncronas (orientadas à interrupção). A maioria das E/S físicas são assíncronas — a CPU inicializa a transferência e vai fazer outra coisa até a chegada da interrupção. Programas do usuário são muito mais fáceis de escrever se as operações de E/S forem bloqueantes — após uma chamada de sistema read, o programa é automaticamente suspenso até que os dados estejam disponíveis no buffer. Fica a cargo do sistema operacional fazer operações que são realmente orientadas à interrupção parecerem bloqueantes para os programas do usuário. No entanto, algumas aplicações de muito alto desempenho precisam controlar todos os detalhes da E/S, então alguns sistemas operacionais disponibilizam a E/S assíncrona para si.

Utilização do buffer: Muitas vezes, dados provenientes de um dispositivo não podem ser armazenados diretamente em seu destino final. A utilização do buffer envolve consideráveis operações de cópia e muitas vezes tem um impacto importante sobre o desempenho de E/S.

Dispositivos compartilháveis vs. dedicados: Alguns dispositivos de E/S, como discos, podem ser usados por muitos usuários ao mesmo tempo. Nenhum problema é causado por múltiplos usuários terem arquivos abertos no mesmo disco ao mesmo tempo. Outros dispositivos, como impressoras, têm de ser dedicados a um único usuário até ele ter concluído sua operação. Então outro usuário pode ter a impressora. Ter dois ou mais usuários escrevendo caracteres de maneira aleatória e intercalada na mesma página definitivamente não funcionará. Introduzir dispositivos dedicados (não compartilhados) também introduz uma série de problemas, como os impasses. Novamente, o sistema operacional deve ser capaz de lidar com ambos os dispositivos — compartilhados e dedicados — de uma maneira que evite problemas.

E/S programada

A forma mais simples de E/S é ter a CPU realizando todo o trabalho. Esse método é chamado de E/S programada.

Considere um processo de usuário que quer imprimir a cadeia de oito caracteres “ABCDEFGH” na impressora por meio de uma interface serial. O software primeiro monta a cadeia de caracteres em um buffer no espaço do usuário. O processo do usuário requisita então a impressora para escrita fazendo uma chamada de sistema para abri-la. Tão logo a impressora esteja disponível, o sistema operacional copia o primeiro caractere para o registrador de dados da impressora. Tão logo copiado o primeiro caractere para a impressora, o sistema operacional verifica se ela está pronta para aceitar outro. Geralmente, a impressora tem um segundo registrador, que contém seu estado. O ato de escrever para o registrador de dados faz que o estado torne-se “indisponível”. Quando o controlador da impressora tiver processado o caractere atual, ele indica a sua disponibilidade marcando algum bit em seu registrador de status ou colocando algum valor nele.

Após a saída de um caractere, a CPU continuamente verifica o dispositivo para ver se ele está pronto para aceitar outro. Esse comportamento é muitas vezes chamado de espera ocupada (busy waiting) ou polling.

E/S orientada a interrupções

Agora vamos considerar o caso da impressão em uma impressora que não armazena caracteres, mas imprime cada um à medida que ele chega. Se a impressora puder imprimir, digamos, 100 caracteres/segundo, cada caractere levará 10 ms para imprimir. Isso significa que após cada caractere ter sido escrito no registrador de dados da impressora, a CPU vai permanecer em um laço ocioso por 10 ms esperando a permissão para a saída do próximo caractere. Isso é mais tempo do que o necessário para realizar um chaveamento de contexto e executar algum outro processo durante os 10 ms que de outra maneira seriam desperdiçados.

A maneira de permitir que a CPU faça outra coisa enquanto espera que a impressora fique pronta é usar interrupções. Quando a chamada de sistema para imprimir a cadeia é feita, o buffer é copiado para o espaço do núcleo, como já mostramos, e o primeiro caractere é copiado para a impressora tão logo ela esteja disposta a aceitar um caractere. Nesse ponto, a CPU chama o escalonador e algum outro processo é executado. O processo que solicitou que a cadeia seja impressa é bloqueado até que a cadeia inteira seja impressa.

Quando a impressora imprimiu o caractere e está preparada para aceitar o próximo, ela gera uma interrupção. Essa interrupção para o processo atual e salva seu estado. Então a rotina de tratamento de interrupção da impressora é executada.

E/S usando DMA

Uma desvantagem óbvia do mecanismo de E/S orientado a interrupções é que uma interrupção ocorre em cada caractere. Interrupções levam tempo; portanto, esse esquema desperdiça certa quantidade de tempo da CPU. Uma solução é usar o acesso direto à memória (DMA). Aqui a ideia é deixar que o controlador de DMA alimente os caracteres para a impressora um de cada vez, sem que a CPU seja incomodada. Na essência, o DMA executa E/S programada, apenas com o controlador do DMA realizando todo o trabalho, em vez da CPU principal. Essa estratégia exige um hardware especial (o controlador de DMA), mas libera a CPU durante a E/S para fazer outros trabalhos.

A grande vantagem do DMA é reduzir o número de interrupções de uma por caractere para uma por buffer impresso. Se houver muitos caracteres e as interrupções forem lentas, esse sistema poderá representar uma melhoria importante. Por outro lado, o controlador de DMA normalmente é 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 normalmente não tem nada para fazer de qualquer forma enquanto esperando pela interrupção do DMA, então a E/S orientada à interrupção ou mesmo a E/S programada podem ser melhores. Na maioria das vezes, no entanto, o DMA vale a pena.

Camadas do software de E/S

Tratadores de interrupção

As interrupções devem ser escondidas longe, nas profundezas do sistema operacional, de maneira que a menor parcela possível do sistema operacional saiba delas. A melhor maneira de escondê-las é bloquear o driver que inicializou uma operação de E/S até que ela se complete e a interrupção ocorra. O driver pode bloquear a si mesmo, por exemplo, realizando um down em um semáforo.

Quando a interrupção acontece, a rotina de interrupção faz o que for necessário a fim de lidar com ela. Então ela pode desbloquear o driver que a chamou.

Passos realizados no software após interrupção
1)Salvar quaisquer registros não salvos pelo hardware
2)Estabelecer um contexto (incluindo TLB, MMU...)
3)Estabelecer uma pilha
4)Sinalizar o controlador de interrupções
5)Copiar os registradores
6)Executar a rotina (lendo registradores do dispositivo)
7)Escolher próximo processo a ser executado
8)Escolher o contexto de MMU para próximo processo
9)Carregar registradores
10)Começar a execução do novo processo

Drivers dos dispositivos

Cada dispositivo de E/S ligado a um computador precisa de algum código específico do dispositivo para controlá-lo. Esse código, chamado driver do dispositivo, geralmente é escrito pelo fabricante do dispositivo e fornecido junto com ele. Tendo em vista que cada sistema operacional precisa dos seus próprios drivers, os fabricantes de dispositivos comumente fornecem drivers para vários sistemas operacionais populares.

Drivers de USB são tipicamente empilhados, como uma pilha de TCP/IP em redes. Na parte de baixo, em geral no hardware, encontramos a camada do link do USB (E/S serial) que lida com questões de hardware como sinalização e decodificação de um fluxo de sinais para os pacotes do USB. Ele é usado por camadas mais altas que lidam com pacotes de dados e a funcionalidade comum para USB que é compartilhada pela maioria dos dispositivos.

Em cima disso, por fim, encontramos as APIs de camadas superiores, como as interfaces para armazenamento em massa, câmeras etc. Desse modo, ainda temos drivers de dispositivos em separado, embora eles compartilhem de parte da pilha de protocolo.

Para acessar o hardware do dispositivo, isto é, os registradores do controlador, o driver do dispositivo deve fazer parte do núcleo do sistema operacional, pelo menos com as arquiteturas atuais.

Drivers de dispositivos são normalmente posicionados abaixo do resto do sistema operacional.

Sistemas operacionais normalmente classificam os drivers entre um número pequeno de categorias. As categorias mais comuns são os dispositivos de blocos — como discos, que contêm múltiplos blocos de dados que podem ser endereçados independentemente — e dispositivos de caracteres, como teclados e impressoras, que geram ou aceitam um fluxo de caracteres.

Em alguns sistemas, o sistema operacional é um programa binário único que contém compilado em si todos os drivers de que ele precisará. Esse esquema era a norma por anos com sistemas UNIX, pois eles eram executados por centros computacionais e os dispositivos de E/S raramente mudavam. Se um novo dispositivo era acrescentado, o administrador do sistema simplesmente recompilava o núcleo com o driver novo para construir o binário novo.

Com o advento dos computadores pessoais, com sua miríade de dispositivos de E/S, esse modelo não funcionava mais. Poucos usuários são capazes de recompilar ou religar o núcleo, mesmo que eles tenham o código- -fonte ou módulos-objeto, o que nem sempre é o caso. Em vez disso, sistemas operacionais, começando com o MS-DOS, se converteram em um modelo no qual os drivers eram dinamicamente carregados no sistema durante a execução. Sistemas diferentes lidam com o carregamento de drivers de maneiras diferentes.

Um driver de dispositivo apresenta diversas funções. A mais óbvia é aceitar solicitações abstratas de leitura e escrita de um software independente de dispositivo localizado na camada acima dele e verificar que elas sejam executadas. Mas há também algumas outras funções que ele deve realizar. Por exemplo, o driver deve inicializar o dispositivo, se necessário. Ele também pode precisar gerenciar suas necessidades de energia e registrar seus eventos.

Muitos drivers de dispositivos têm uma estrutura geral similar. Um driver típico inicia verificando os parâmetros de entrada para ver se eles são válidos. Se não, um erro é retornado. Se eles forem válidos, uma tradução dos termos abstratos para os termos concretos pode ser necessária. Para um driver de disco, isso pode significar converter um número de bloco linear em números de cabeçote, trilha, setor e cilindro para a geometria do disco.

Em seguida, o driver pode conferir se o dispositivo está em uso no momento. Se ele estiver, a solicitação entrará em uma fila para processamento posterior. Se o dispositivo estiver ocioso, o estado do hardware será examinado para ver se a solicitação pode ser cuidada agora. Talvez seja necessário ligar o dispositivo ou um motor antes que as transferências possam começar. Uma vez que o dispositivo esteja ligado e pronto para trabalhar, o controle de verdade pode começar.

Controlar o dispositivo significa emitir uma sequência de comandos para ele. O driver é o local onde a sequência de comandos é determinada. Após o driver saber qual comando ele vai emitir, ele começa escrevendo-os nos registradores do controlador do dispositivo. Após

cada comando ter sido escrito para o controlador, talvez seja necessário conferir para ver se este aceitou o comando e está preparado para aceitar o seguinte. Essa sequência continua até que todos os comandos tenham sido emitidos. Alguns controladores podem receber uma lista encadeada de comandos (na memória), tendo de ler e processar todos eles sem mais ajuda alguma do sistema operacional.

Após os comandos terem sido emitidos, ocorrerá uma de duas situações. Na maioria dos casos, o driver do dispositivo deve esperar até que o controlador realize algum trabalho por ele, de maneira que ele bloqueia a si mesmo até que a interrupção chegue para desbloqueá-lo. Em outros casos, no entanto, a operação termina sem atraso, então o driver não precisa bloquear.

De qualquer maneira, após a operação ter sido completada, o driver deverá conferir a ocorrência de erros. Se tudo estiver bem, o driver poderá ter alguns dados para passar para o software independente do dispositivo (por exemplo, um bloco recém-lido). Por fim, ele retorna ao seu chamador alguma informação de estado para o relatório de erros. Se quaisquer outras solicitações estiverem na fila, uma delas poderá então ser selecionada e inicializada. Se nada estiver na fila, o driver bloqueará a espera para a próxima solicitação.

Enquanto o driver da rede está processando um pacote que chega, outro pacote pode chegar. Em consequência, os drivers têm de ser reentrantes, significando que um driver em execução tem de estar preparado para ser chamado uma segunda vez antes que a primeira chamada tenha sido concluída.

Drivers não têm permissão para fazer chamadas de sistema, mas eles muitas vezes precisam interagir com o resto do núcleo. Em geral, são permitidas chamadas para determinadas rotinas de núcleo.

Software de E/S independente de dispositivo
A função básica do software independente do dispositivo é realizar as funções de E/S que são comuns a todos os dispositivos e fornecer uma interface uniforme para o software no nível do usuário. Examinaremos agora essas questões mais detalhadamente.

Funções:
- Uniformizar interfaces para os drivers de dispositivos
- Armazenar no buffer
- Reportar erros
- Alocar e liberar dispositivos dedicados
- Providenciar um tamanho de bloco independente de dispositivo

1) Interface uniforme para os drivers dos dispositivos
   Uma questão importante em um sistema operacional é como fazer todos os dispositivos de E/S e drivers parecerem mais ou menos o mesmo. Se discos, impressoras, teclados e assim por diante possuem todos interfaces diferentes, sempre que um dispositivo novo aparece, o sistema operacional tem de ser modificado para o novo dispositivo. Ter de modificar o sistema operacional para cada dispositivo novo não é uma boa ideia.

A maneira como isso funciona é a seguinte: para cada classe de dispositivos, como discos ou impressoras, o sistema operacional define um conjunto de funções que o driver deve fornecer. Para um disco, essas naturalmente incluiriam a leitura e a escrita, além de ligar e desligar a energia, formatação e outras coisas típicas de discos. Muitas vezes o driver contém uma tabela com ponteiros para si mesmo para essas funções. Quando o driver está carregado, o sistema operacional registra o endereço dessa tabela de ponteiros de funções, de maneira que, quando ela precisa chamar uma das funções, pode fazer uma chamada indireta via essa tabela. A tabela de ponteiros de funções define a interface entre o driver e o resto do sistema operacional. Todos os dispositivos de uma determinada classe (discos, impressoras etc.) devem obedecer a ela.

O software independente do dispositivo cuida do mapeamento de nomes de dispositivos simbólicos para o driver apropriado. No UNIX, por exemplo, o nome de um dispositivo como /dev/disk0 especifica unicamente o i-node para um arquivo especial, e esse i-node contém o número do dispositivo especial, que é usado para localizar o driver apropriado. O i-node também contém o número do dispositivo secundário, que é passado como um parâmetro para o driver a fim de especificar a unidade a ser lida ou escrita. Todos os dispositivos têm números principal e secundário, e todos os drivers são acessados usando o número de dispositivo especial para selecionar o driver

O administrador do sistema pode então estabelecer as permissões adequadas para cada dispositivo.

2) Armazenar no buffer
   A utilização de buffer também é uma questão, tanto para dispositivos de bloco quanto de caracteres, por uma série de razões. Uma estratégia possível para lidar com os caracteres que chegam é fazer o processo do usuário realizar uma chamada de sistema read e bloquear a espera por um caractere. Cada caractere que chega causa uma interrupção. A rotina de tratamento da interrupção passa o caractere para o processo do usuário e o desbloqueia. Após colocar o caractere em algum lugar, o processo lê outro caractere e bloqueia novamente.

   O problema com essa maneira de fazer negócios é que o processo do usuário precisa ser inicializado para cada caractere que chega. Permitir que um processo execute muitas vezes durante curtos intervalos de tempo é algo ineficiente; portanto, esse projeto não é uma boa opção.

   Uma outra abordagem é que o processo do usuário fornece um buffer de n caracteres no espaço do usuário e faz uma leitura de n caracteres. A rotina de tratamento da interrupção coloca os caracteres que chegam nesse buffer até que ele esteja completamente cheio. Apenas então ele desperta o processo do usuário. Esse esquema é muito mais eficiente do que o anterior, mas tem uma desvantagem: o que acontece se o buffer for paginado para o disco quando um caractere chegar? O buffer poderia ser trancado na memória, mas se muitos processos começarem a

trancar páginas na memória descuidadamente, o conjunto de páginas disponíveis diminuirá e o desempenho cairá.

Outra abordagem ainda é criar um buffer dentro do núcleo e fazer o tratador de interrupção colocar os caracteres ali. Quando esse buffer estiver cheio, a página com o buffer do usuário é trazida. No entanto, mesmo esse esquema melhorado sofre de um problema: o que acontece com os caracteres que chegam enquanto a página com o buffer do usuário está sendo trazida do disco? Já que o buffer está cheio, não há um lugar para colocá-los. Uma solução é ter um segundo buffer de núcleo. Quando o primeiro buffer está cheio, mas antes que ele seja esvaziado, o segundo buffer é usado. Dessa maneira, os dois buffers se revezam: enquanto um está sendo copiado para o espaço do usuário, o outro está acumulando novas entradas. Esse tipo de esquema é chamado de utilização de buffer duplo (double buffering).

Outra forma comum de utilização de buffer é o buffer circular. Ele consiste em uma região da memória e dois ponteiros. Um ponteiro aponta para a próxima palavra livre, onde novos dados podem ser colocados. O outro ponteiro aponta para a primeira palavra de dados no buffer que ainda não foi removida.

A utilização de buffer é uma técnica amplamente utilizada, mas ele tem uma desvantagem também: se os dados forem armazenados em buffer vezes demais, o desempenho sofre. Uma vez inicializada uma transmissão do pacote, ela deve continuar a uma velocidade uniforme O driver não pode garantir essa velocidade uniforme, pois canais de DMA e outros dispositivos de E/S podem estar roubando muitos ciclos. Uma falha na obtenção de uma palavra a tempo arruinaria o pacote. A utilização de buffer para o pacote dentro do controlador pode contornar esse problema.

3) Reportar erros
   Muitos erros são específicos de dispositivos e devem ser tratados pelo driver apropriado, mas o modelo para o tratamento de erros é independente do dispositivo.

   Uma classe de erros de E/S é a dos erros de programação. Eles ocorrem quando um processo pede por algo impossível, como escrever em um dispositivo de entrada (teclado, scanner, mouse etc.) ou ler de um dispositivo de saída (impressora, plotter etc.).

   Outra classe de erros é a que engloba erros de E/S reais, por exemplo, tentar escrever em um bloco de disco que foi danificado ou tentar ler de uma câmera de vídeo que foi desligada. Se o driver não sabe o que fazer, ele pode passar o problema de volta para o software independente do dispositivo.

4) Alocar e liberar dispositivos dedicados
   Alguns dispositivos, como impressoras, podem ser usados somente por um único processo a qualquer dado momento. Cabe ao sistema operacional examinar solicitações para o uso de dispositivos e aceitá-los ou rejeitá-los, dependendo de o dispositivo solicitado estar disponível ou não. Uma maneira simples de lidar com

essas solicitações é exigir que os processos executem chamadas de sistema open diretamente nos arquivos especiais para os dispositivos. Se o dispositivo estiver indisponível, a chamada open falha. O fechamento desse dispositivo dedicado então o libera.

Uma abordagem alternativa é ter mecanismos especiais para solicitação e liberação de serviços dedicados. Uma tentativa de adquirir um dispositivo que não está disponível bloqueia o processo chamador em vez de causar uma falha. Processos bloqueados são colocados em uma fila.

5) Providenciar um tamanho de bloco independente de dispositivo
   Discos diferentes podem ter tamanhos de setores diferentes. Cabe ao software independente do dispositivo esconder esse fato e fornecer um tamanho de bloco uniforme para as camadas superiores, por exemplo, tratando vários setores como um único bloco lógico. Dessa maneira, as camadas superiores lidam apenas com dispositivos abstratos, que usam o mesmo tamanho de bloco lógico, independentemente do tamanho do setor físico.

Software de E/S do espaço do usuário

Embora a maior parte do software de E/S esteja dentro do sistema operacional, uma pequena porção dele consiste em bibliotecas ligadas aos programas do usuário e mesmo programas inteiros sendo executados fora do núcleo. Chamadas de sistema, incluindo chamadas de sistema de E/S, são normalmente feitas por rotinas de biblioteca.

Em particular, a formatação de entrada e saída é feita pelas rotinas de biblioteca. Um exemplo de C é printf, que recebe uma cadeia de caracteres e possivelmente algumas variáveis como entrada, constrói uma cadeia ASCII, e então chama o write para colocá-la na saída.

A biblioteca de E/S padrão contém um número de rotinas que envolvem E/S e todas executam como parte dos programas do usuário.

Nem todo software de E/S no nível do usuário consiste em rotinas de biblioteca. Outra categoria importante é o sistema de spooling. O uso de spool é uma maneira de lidar com dispositivos de E/S dedicados em um sistema de multiprogramação.

Um processo especial, chamado daemon, é um diretório especial, chamado de diretório de spooling. Para imprimir um arquivo, um processo primeiro gera o arquivo inteiro a ser impresso e o coloca no diretório de spooling. Cabe ao daemon, que é o único processo com permissão de usar o arquivo especial da impressora, imprimir os arquivos no diretório. Ao proteger o arquivo especial contra o uso direto pelos usuários, o problema de ter alguém mantendo-o aberto por um tempo desnecessariamente longo é eliminado.

Resume o sistema de E/S, mostrando todas as camadas e as principais funções de cada uma. Começando na parte inferior, as camadas são o hardware, tratadores de interrupção,

drivers do dispositivo, software independente de dispositivos e, por fim, os processos do usuário.

Quando um programa do usuário tenta ler um bloco de um arquivo, por exemplo, o sistema operacional é invocado para executar a chamada. O software independente de dispositivos o procura, digamos, na cache do buffer. Se o bloco necessário não está lá, ele chama o driver do dispositivo para emitir a solicitação para o hardware ir buscá-lo do disco. O processo é então bloqueado até que a operação do disco tenha sido completada e os dados estejam seguramente disponíveis no buffer do chamador.

Quando o disco termina, o hardware gera uma interrupção. O tratador de interrupção é executado para descobrir o que aconteceu, isto é, qual dispositivo quer atenção agora. Ele então extrai o estado do dispositivo e desperta o processo dormindo para finalizar a solicitação de E/S e deixar que o processo do usuário continue.

Discos

Hardware do disco

Existe uma série de tipos de discos. Os mais comuns são os discos rígidos magnéticos. Eles se caracterizam pelo fato de que leituras e escritas são igualmente rápidas, o que os torna adequados como memória secundária (paginação, sistemas de arquivos etc.). Arranjos desses discos são usados às vezes para fornecer um armazenamento altamente confiável. Para distribuição de programas, dados e filmes, discos ópticos (DVDs e Blu-ray) também são importantes. Por fim, discos de estado sólido são cada dia mais populares à medida que eles são rápidos e não contêm partes móveis.

Discos magnéticos
Discos magnéticos são organizados em cilindros, cada um contendo tantas trilhas quanto for o número de cabeçotes dispostos verticalmente. As trilhas são divididas em setores, com o número de setores em torno da circunferência sendo tipicamente 8 a 32 nos discos flexíveis e até várias centenas nos discos rígidos. O número de cabeçotes varia de 1 a cerca de 16.

Discos IDE (Integrated Drive Electronics — eletrônica integrada ao disco) e SATA (Serial ATA — ATA serial), a própria unidade contém um microcontrolador que realiza um trabalho considerável e permite que o controlador real emita um conjunto de comandos de nível mais elevado. O controlador muitas vezes controla a cache, faz o remapeamento de blocos defeituosos e muito mais.

Uma característica do dispositivo que tem implicações importantes para o driver do disco é a possibilidade de um controlador realizar buscas em duas ou mais unidades ao mesmo tempo. Elas são conhecidas como buscas sobrepostas (overlapped seeks). Enquanto o controlador e o software estão esperando que uma busca seja concluída em uma unidade, o controlador pode iniciar uma busca em outra. Muitos controladores também podem ler ou escrever em uma unidade enquanto realizam uma busca em uma ou mais unidades, mas

um controlador de disco flexível não pode ler ou escrever em duas unidades ao mesmo tempo.

A situação é diferente para discos rígidos com controladores integrados e, em um sistema com mais de um desses discos rígidos, eles podem operar simultaneamente, pelo menos até o ponto de transferência entre o disco e o buffer de memória do controlador. No entanto, apenas uma transferência entre o controlador e a memória principal é possível ao mesmo tempo. A capacidade de desempenhar duas ou mais operações ao mesmo tempo pode reduzir o tempo de acesso médio consideravelmente.

Um detalhe para o qual precisamos atentar ao examinarmos as especificações dos discos rígidos modernos é que a geometria especificada, e usada pelo driver, é quase sempre diferente do formato físico. Em discos antigos, o número de setores por trilha era o mesmo para todos os cilindros. Discos modernos são divididos em zonas com mais setores nas zonas externas do que nas internas. O software é instruído a agir como se houvesse x cilindros, y cabeçotes e z setores por trilha.. Em ambos os casos o disco tem 192 setores, apenas o arranjo publicado é diferente do real.

Todos os discos modernos aceitam um sistema chamado endereçamento lógico de bloco (logical block addressing), no qual os setores do disco são numerados consecutivamente começando em 0, sem levar em consideração a geometria do disco.

RAID

O desempenho da CPU tem aumentado exponencialmente na última década, dobrando mais ou menos a cada 18 meses, mas nem tanto o desempenho de disco. Em seu estudo de 1988, Patterson et al. sugeriram seis organizações de disco específicas que poderiam ser usadas para melhorar o desempenho do disco e levaram a uma nova classe de dispositivo de E/S chamada RAID. Patterson et al. definiram RAID como arranjo redundante de discos baratos.

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. Em outras palavras, um RAID deve parecer com um SLED para o sistema operacional, mas ter um desempenho melhor e mais confiável.

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. Vários esquemas diferentes para fazer isso foram definidos por Patterson et al. Hoje, a maioria dos fabricantes refere-se às sete configurações padrão com RAID nível 0 a RAID nível 6.

O RAID nível 0

O RAID nível 0 consiste em ver o disco único virtual simulado pelo RAID como dividido em faixas de k setores cada, com os setores 0 a k − 1 sendo a faixa 0,setores k a 2k − 1 faixa 1, e assim por diante. Para k = 1, cada faixa é um setor, para k = 2 uma faixa são dois setores etc. A organização do RAID nível 0 grava faixas consecutivas nos discos em um estilo de alternância circular (round-robin).

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 operarem em paralelo. Desse modo, temos E/S paralela sem que o software saiba a respeito.

O RAID nível 0 funciona melhor com grandes solicitações, quanto maiores melhor. Se uma solicitação for maior que o produto do número de discos pelo tamanho da faixa, alguns discos receberão múltiplas solicitações, assim quando terminam a primeira, eles iniciam a segunda. Cabe ao controlador dividir a solicitação e alimentar os comandos apropriados para os discos apropriados na sequência certa e então montar os resultados na memória corretamente. O desempenho é excelente e a implementação, direta.

O RAID nível 1
O RAID nível 1 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 quebra, a cópia é simplesmente usada em seu lugar. A recuperação consiste em apenas instalar um disco novo e copiar o backup inteiro para ele.

O RAID nível 2
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 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. 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
O RAID nível 3 é uma versão simplificada do RAID nível 2. 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.

O RAID nível 4
O RAID nível 4 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.

O RAID nível 5
Mesma ideia do RAID nível 4, porém em consequência da carga pesada sobre o disco de paridade, o RAID nível 4 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). No entanto, caso ocorra uma quebra do disco, a reconstrução do disco falhado é um processo complexo.

O RAID nível 6
O RAID nível 6 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).

Formatação de disco

Antes que o 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.

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.

Vale a pena observar que o chaveamento entre cabeçotes também leva um tempo finito; portanto, existe também um deslocamento de cabeçote assim como um de cilindro, mas o

deslocamento de cabeçote não é muito grande, normalmente muito menor do que um tempo de setor.

Como resultado da formatação de baixo nível, a capacidade do disco é 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.

Quando uma cópia para a memória for concluída, o controlador terá de esperar quase o tempo de uma rotação inteira para que o segundo setor dê a volta novamente.

Esse problema pode ser eliminado numerando os setores de maneira entrelaçada quando se formata o disco.Temos o padrão de numeração usual (ignorando o deslocamento de cilindro aqui). Temos também um entrelaçamento simples (single interleaving), que dá ao controlador algum descanso entre os setores consecutivos a fim de copiar o buffer para a memória principal.

Se o processo de cópia for muito lento, o entrelaçamento duplo poderá ser necessário. Se o controlador tem um buffer de apenas um setor, não importa se a cópia do buffer para a memória principal é feita pelo controlador, a CPU principal ou um chip DMA; ela ainda leva algum tempo. Para evitar a necessidade do entrelaçamento, o controlador deve ser capaz de armazenar uma trilha inteira. A maioria dos controladores modernos consegue armazenar trilhas inteiras.

Após a formatação de baixo nível ter sido concluída, o disco é dividido em partições. Logicamente, cada partição é como um disco separado. Partições são necessárias para permitir que múltiplos sistemas operacionais coexistam. Também, em alguns casos, uma partição pode ser usada como área de troca (swapping).

No x86 e na maioria dos outros computadores, o setor 0 contém o registro mestre de inicialização (Master Boot Record — MBR), que contém um código de inicialização mais a tabela de partição no fim.

A maioria dos sistemas operacionais desde então também suporta o novo GPT (GUID Partition Table), que suporta discos de até 9,4 ZB.

A tabela de partição dá o setor de inicialização e o tamanho de cada partição.

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.

Algoritmos de escalonamento de braço de disco

Quanto tempo é necessário para ler ou escrever um bloco de disco?O tempo exigido é determinado por três fatores.

- Tempo de busca
- Atraso de rotação
- Transferência real do dado

Para a maioria dos discos, o tempo de busca domina os outros dois, então a redução do tempo de busca médio pode melhorar substancialmente o desempenho do sistema.

FCFS (First-Come, First-Served)
Se o driver do disco aceita as solicitações uma de cada vez e as atende nessa ordem, isto é, “primeiro a chegar, primeiro a ser servido” (FCFS — First-Come, First-Served), pouco pode ser feito para otimizar o tempo de busca. No entanto, outra estratégia é possível quando o disco está totalmente carregado. É provável que enquanto o braço está se posicionando para uma solicitação, outras solicitações de disco sejam geradas por outros processos. Muitos drivers de disco mantêm uma tabela, indexada pelo número do cilindro, com todas as solicitações pendentes para cada cilindro encadeadas juntas em uma lista ligada encabeçada pelas entradas da tabela.

Exemplo: Considere um disco imaginário com 40 cilindros. Uma solicitação chega para ler um bloco no cilindro 11. Enquanto a busca para o cilindro 11 está em andamento, chegam novos pedidos para os cilindros 1, 36, 16, 34, 9 e 12, nessa ordem. Elas são colocadas na tabela de solicitações pendentes, com uma lista encadeada separada para cada cilindro.

Quando a solicitação atual (para o cilindro 11) tiver sido concluída, o driver do disco tem uma escolha de qual solicitação atender em seguida. Usando FCFS, ele iria em seguida para o cilindro 1, então para o 36, e assim por diante. Esse algoritmo exigiria movimentos de braço de 10, 35, 20, 18, 25 e 3, respectivamente, para um total de 111 cilindros.

Shortest Seek First (SSF)
Como alternativa, ele sempre poderia tratar com a solicitação mais próxima em seguida, a fim de minimizar o tempo de busca. Dadas as solicitações da Figura 5.24, a sequência é 12, 9, 16, 1, 34 e 36, como mostra a linha com quebras irregulares na base da Figura 5.24. Com essa sequência, os movimentos de braço são 1, 3, 7, 15, 33 e 2, para um total de 61 cilindros. Esse algoritmo, chamado de busca mais curta primeiro (SSF — Shortest Seek First), corta o movimento de braço total quase pela metade em comparação com o FCFS.

Infelizmente, o SSF tem um problema. Com um disco totalmente carregado, o braço tenderá a ficar no meio do disco na maior parte do tempo, de maneira que as solicitações em

qualquer um dos extremos terão de esperar até que uma flutuação estatística na carga faça que não existam solicitações próximas do meio. As solicitações longe do meio talvez sejam mal servidas. As metas de tempo de resposta mínimo e justiça estão em conflito aqui.

Algoritmo do elevador

No entanto, a maioria dos elevadores usa um algoritmo diferente a fim de reconciliar as metas mutuamente conflitantes da eficiência e da justiça. Eles continuam se movimentando na mesma direção até que não existam mais solicitações pendentes naquela direção, então eles trocam de direção. Esse algoritmo, conhecido tanto no mundo dos discos quanto no dos elevadores como o algoritmo do elevador, exige que o software mantenha 1 bit: o bit de direção atual, SOBE ou DESCE. Quando uma solicitação é concluída, o driver do disco ou elevador verifica o bit. Se ele for SOBE, o braço ou a cabine se move para a próxima solicitação pendente mais alta. Se nenhuma solicitação estiver pendente em posições mais altas, o bit de direção é invertido. Quando o bit contém DESCE, o movimento será para a posição solicitada seguinte mais baixa, se houver alguma. Se nenhuma solicitação estiver pendente, ele simplesmente para e espera.

Uma ligeira modificação desse algoritmo que tem uma variação menor nos tempos de resposta (TEORY, 1972) consiste em sempre varrer as solicitações na mesma direção. Quando o cilindro com o número mais alto com uma solicitação pendente tiver sido atendido, o braço vai para o cilindro com o número mais baixo com uma solução pendente e então continua movendo-se para cima. Na realidade, o cilindro com o número mais baixo é visto como logo acima do cilindro com o número mais alto.

Alguns controladores de disco fornecem uma maneira para o software inspecionar o número de setor atual sob o cabeçote. Com esse controlador, outra otimização é possível. Se duas ou mais solicitações para o mesmo cilindro estiverem pendentes, o driver poderá emitir uma solicitação para o setor que passará sob o cabeçote em seguida. Observe que, quando múltiplas trilhas estão presentes em um cilindro, solicitações consecutivas podem ocorrer para diferentes trilhas sem uma penalidade. O controlador pode selecionar qualquer um dos seus cabeçotes quase instantaneamente.

Com um disco rígido moderno, a busca e os atrasos rotacionais dominam de tal maneira o desempenho que a leitura de um ou dois setores de cada vez é algo ineficiente demais. Por essa razão, muitos controladores de disco sempre leem e armazenam múltiplos setores, mesmo quando apenas um é solicitado.

Vale a pena observar que a cache do controlador de disco é completamente independente da cache do sistema operacional. A cache do controlador em geral contém blocos que não foram realmente solicitados, mas cuja leitura era conveniente porque eles apenas estavam passando sob o cabeçote como um efeito colateral de alguma outra leitura. Em comparação, qualquer cache mantida pelo sistema operacional consistirá de blocos que foram explicitamente lidos e que o sistema operacional acredita que possam ser necessários novamente em um futuro próximo.

Quando vários dispositivos estão presentes no mesmo controlador, o sistema operacional deve manter uma tabela de solicitações pendentes para cada dispositivo separadamente.

Sempre que um dispositivo estiver ocioso, um comando de busca deve ser emitido para mover o seu braço para o cilindro onde ele será necessário em seguida (presumindo que o controlador permita buscas simultâneas). Quando a transferência atual é concluída, uma verificação deve ser feita para ver se algum dispositivo está posicionado no cilindro correto. Se um ou mais estiverem, a próxima transferência poderá ser inicializada em um drive que já esteja no cilindro correto. Se nenhum dos braços estiver no local correto, o driver deverá emitir um novo comando de busca sobre o dispositivo que acabou de completar uma transferência e esperar até a próxima interrupção para ver qual braço chega ao seu destino primeiro.

É importante perceber que todos os algoritmos de escalonamento de disco acima tacitamente presumem que a geometria de disco real é a mesma que a geometria virtual.

Tratamento de erros

Os defeitos de fabricação introduzem setores defeituosos, isto é, setores que não leem corretamente de volta o valor recém-escrito neles. Se o defeito for muito pequeno, digamos, 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. Suponha 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. A outra saída é deslocar todos os setores de uma posição. 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 5.26(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.

Erros 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.

Se o controlador não tiver a capacidade de remapear setores transparentemente como discutimos, 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.

No entanto, há ainda outro problema: backups. 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.

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. A maioria dos controladores de disco rígido corrige erros de busca automaticamente.

Recalibragem pode ser feita automaticamente pelo controlador. Recalibrar um disco faz um ruído esquisito, mas de outra forma, não chega a incomodar. No entanto, há uma situação em que a recalibragem é um problema: sistemas com restrições de tempo real.

Armazenamento estável

RAIDs não protegem contra erros de gravação. Quando uma escrita falha, os dados originais devem permanecer intactos. Erro pode ser detectado através do campo ECC. Recuperação feita por um par de discos idênticos.

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. A meta é manter o disco consistente a todo custo.

Antes de descrever o algoritmo, é importante termos um modelo claro dos erros possíveis. O modelo presume que, quando um disco escreve um bloco (um ou mais setores), ou a escrita está correta ou ela está incorreta e esse erro pode ser detectado em uma leitura subsequente examinando os valores dos campos ECC.

O modelo também presume que um setor escrito corretamente pode ficar defeituoso espontaneamente e tornar- -se ilegível. No entanto, a suposição é que esses eventos são tão raros que a probabilidade de ter o mesmo setor danificado em um segundo dispositivo (independente) durante um intervalo de tempo razoável (por exemplo, 1 dia) é pequena o suficiente para ser ignorada.

O modelo também presume que a CPU pode falhar, caso em que ela simplesmente para. Qualquer escrita no disco em andamento no momento da falha também para, levando a dados incorretos em um setor e um ECC incorreto que pode ser detectado mais tarde. Com todas essas considerações, o armazenamento estável pode se tornar 100% confiável, no sentido de que as escritas funcionem corretamente ou deixem os dados antigos no lugar.

O armazenamento estável usa um par de discos idênticos com blocos correspondentes trabalhando juntos para formar um bloco livre de erros. Na ausência de erros, os blocos correspondentes em ambos os dispositivos são os mesmos. Qualquer um pode ser lido para conseguir o mesmo resultado. Para alcançar essa meta, as três operações a seguir são definidas:

1. Escritas estáveis. Uma escrita estável consiste em primeiro escrever um bloco na unidade 1, então lê-lo de volta para verificar que ele foi escrito corretamente.

2. Leituras estáveis. Uma leitura estável primeiro lê o bloco da unidade 1. Se isso produzir um ECC incorreto, a leitura é tentada novamente, até n vezes. Se todas elas derem ECCs defeituosos, o bloco correspondente é lido da unidade 2. Levando-se em consideração o fato de que uma escrita estável bem-sucedida deixa duas boas cópias de um bloco para trás, e nossa suposição de que a probabilidade de o mesmo bloco espontaneamente apresentar um defeito em ambas as unidades em um intervalo de tempo razoável é desprezível, uma leitura estável sempre é bem-sucedida.

3. Recuperação de falhas. Após uma queda do sistema, um programa de recuperação varre ambos os discos comparando blocos correspondentes. Se um par de blocos está bem e ambos são iguais, nada é feito. Se um deles tiver um erro de ECC, o bloco defeituoso é sobrescrito com o bloco bom correspondente. Se um par de blocos está bem mas eles são diferentes, o bloco da unidade 1 é escrito sobre o da unidade 2.

Na ausência de falhas de CPU, esse esquema sempre funciona, pois escritas estáveis sempre escrevem duas cópias válidas de cada bloco e erros espontâneos são presumidos que jamais ocorram em ambos os blocos correspondentes ao mesmo tempo. E na presença de falhas na CPU durante escritas estáveis? Depende precisamente de quando ocorre a falha. Há cinco possibilidades.

1: a falha da CPU ocorre antes que qualquer uma das cópias do bloco seja escrita;
2: a CPU falha durante a escrita na unidade 1, destruindo o conteúdo do bloco.
3: a falha da CPU acontece após a unidade 1 ter sido escrita, mas antes de a unidade 2 ter sido escrita;
4: a CPU falha durante a escrita na unidade 2, destruindo o conteúdo do bloco;
5: o programa de recuperação vê que ambos os blocos são os mesmos, portanto nenhum deles é modificado e a escrita é bem-sucedida aqui também.

Um avanço enorme é controlar qual bloco estava sendo escrito durante a escrita estável, de maneira que apenas um bloco precisa ser conferido durante a recuperação.

Uma vez por dia uma varredura completa de ambos os discos deve ser feita, reparando qualquer defeito. Dessa maneira, todas as manhãs ambos os discos estarão sempre idênticos. Mesmo que ambos os blocos em um par apresentarem defeitos dentro de um período de alguns dias, todos os erros serão reparados corretamente.

Relógios

Relógios (também chamados de temporizadores — timers) são essenciais para a operação de qualquer sistema multiprogramado por uma série de razões. Eles mantêm a hora do dia e evitam que um processo monopolize a CPU, entre outras coisas. O software do relógio pode assumir a forma de um driver de dispositivo, embora um relógio não seja nem um dispositivo de bloco, como um disco, tampouco um dispositivo de caractere, como um mouse.

Hardware de relógios

Dois tipos de relógios são usados em computadores. Os relógios mais simples são ligados à rede elétrica de 110 ou 220 volts e causam uma interrupção a cada ciclo de voltagem, em 50 ou 60 Hz. Esses relógios costumavam dominar o mercado, mas são raros hoje.

O outro tipo de relógio é construído de três componentes: um oscilador de cristal, um contador e um registrador de apoio. Quando um fragmento de cristal é cortado adequadamente e montado sob tensão, ele pode ser usado para gerar um sinal periódico de altíssima precisão, em geral na faixa de várias centenas de mega-hertz até alguns giga-hertz, dependendo do cristal escolhido. Esse sinal é colocado em um contador para fazê-lo contar regressivamente até zero. Quando o contador chega a zero, ele provoca uma interrupção na CPU.

Relógios programáveis tipicamente têm vários modos de operação. No modo disparo único (one-shot mode), quando o relógio é inicializado, ele copia o valor do registrador de apoio no contador e então decrementa o contador em cada pulso do cristal. Quando o contador chega a zero, ele provoca uma interrupção e para até que ele é explicitamente inicializado novamente pelo software. No modo onda quadrada, após atingir o zero e causar a interrupção, o registrador de apoio é automaticamente copiado para o contador, e todo o processo é repetido de novo indefinidamente. Essas interrupções periódicas são chamadas de tiques do relógio.

A vantagem do relógio programável é que a sua frequência de interrupção pode ser controlada pelo software.

Para evitar que a hora atual seja perdida quando a energia do computador é desligada, a maioria dos computadores tem um relógio de back-up mantido por uma bateria, implementado com o tipo de circuito de baixo consumo usado em relógios digitais. O relógio de bateria pode ser lido na inicialização. Se o relógio de backup não estiver presente, o software pode pedir ao usuário a data e o horário atuais. Normalmente programas utilitários são fornecidos para ajustar manualmente o relógio do sistema e o relógio de backup e para sincronizar os dois.

Software de relógio

Tudo o que o hardware de relógios faz é gerar interrupções a intervalos conhecidos. Todo o resto envolvendo tempo deve ser feito pelo software, o driver do relógio. As tarefas exatas

do driver do relógio variam entre os sistemas operacionais, mas em geral incluem a maioria das ações seguintes:

1. Manter o horário do dia.
2. Evitar que processos sejam executados por mais tempo do que o permitido.
3. Contabilizar o uso da CPU.
4. Tratar a chamada de sistema alarm 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 difícil. Ela apenas exige incrementar um contador a cada tique do relógio, como mencionado anteriormente. A única coisa a ser observada é o número de bits no contador da hora do dia. Com uma frequência de relógio de 60 Hz, um contador de 32 bits ultrapassaria sua capacidade em apenas um pouco mais de dois anos. Claramente, o sistema não consegue armazenar o tempo real como o número de tiques desde 1o de janeiro de 1970 em 32 bits.

Há três maneiras de resolver esse problema. A primeira maneira é usar um contador de 64 bits. A segunda maneira é manter a hora do dia em segundos, em vez de em tiques, usando um contador subsidiário para contar tiques até que um segundo inteiro tenha sido acumulado. A terceira abordagem é contar os tiques, mas fazê-lo em relação ao momento em que o sistema foi inicializado, em vez de em relação a um momento externo fixo.

A segunda função do relógio é evitar que os processos sejam executados por um tempo longo demais. Sempre que um processo é iniciado, o escalonador inicializa um contador com o valor do quantum do processo em tiques do relógio. A cada interrupção do relógio, o driver decrementa o contador de quantum em 1. Quando chega a 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 precisa de fazer isso é inicializar um segundo temporizador, distinto do temporizador principal do sistema, sempre que um processo é iniciado. Quando um processo é parado, o temporizador pode ser lido para dizer quanto tempo ele esteve em execução. Para fazer as coisas direito, o segundo temporizador deve ser salvo quando ocorre uma interrupção e restaurado mais tarde.

Se o driver do relógio tivesse relógios o bastante, ele poderia separar um relógio para cada solicitação. Não sendo o caso, ele deve simular múltiplos relógios virtuais com um único relógio físico. Uma maneira é manter uma tabela na qual o tempo do sinal para todos os temporizadores pendentes é mantido, assim como uma variável dando o tempo para o próximo. Sempre que a hora do dia for atualizada, o driver confere para ver se o sinal mais próximo ocorreu. Se ele tiver ocorrido, a tabela é pesquisada para encontrar o próximo sinal a ocorrer.

Se muitos sinais são esperados, é mais eficiente simular múltiplos relógios encadeando juntas todas as solicitações de relógio pendentes, ordenadas no tempo, em uma lista

encadeada. Cada entrada na lista diz quantos tiques do relógio desde o sinal anterior deve-se esperar antes de gerar um novo sinal.

Partes do sistema operacional também precisam estabelecer temporizadores. Eles são chamados de temporizadores watchdog (cão de guarda) e são frequentemente usados (especialmente em dispositivos embarcados) para detectar problemas como travamentos do sistema. Por exemplo, um temporizador watchdog pode reinicializar um sistema que para de executar.

O mecanismo usado pelo driver do relógio para lidar com temporizadores watchdog é o mesmo que para os sinais do usuário. A única diferença é que, quando um temporizador é desligado, em vez de causar um sinal, o driver do relógio chama uma rotina fornecida pelo chamador. A rotina faz parte do código do chamador. A rotina chamada pode fazer o que for necessário, mesmo causar uma interrupção, embora dentro do núcleo as interrupções sejam muitas vezes inconvenientes e sinais não existem. Essa é a razão pela qual o mecanismo do watchdog é fornecido.

A última questão em nossa lista é o perfil de execução. Alguns sistemas operacionais fornecem um mecanismo pelo qual um programa do usuário pode obter do sistema um histograma do seu contador do programa, então ele pode ver onde está gastando seu tempo. Esse mecanismo também pode ser usado para extrair o perfil de execução do próprio sistema.

Temporizadores por software

A maioria dos computadores tem um segundo relógio programável que pode ser ajustado para provocar interrupções no temporizador a qualquer frequência de que um programa precisar. Esse temporizador é um acréscimo ao temporizador do sistema principal cujas funções foram descritas antes. Enquanto a frequência de interrupção for baixa, não há problema em se usar esse segundo temporizador para fins específicos da aplicação. O problema ocorre quando a frequência do temporizador específico da aplicação for muito alta.

Em geral, há duas maneiras de gerenciar E/S: interrupções e polling. Interrupções têm baixa latência, isto é, elas acontecem imediatamente após o evento em si com pouco ou nenhum atraso. Por outro lado, com as CPUs modernas, as interrupções têm uma sobrecarga substancial pela necessidade de chaveamento de contexto e sua influência no pipeline, TLB e cache.

A alternativa às interrupções é permitir que a própria aplicação verifique por meio de polling (verificações ativas) o evento esperado em si. Isso evita interrupções, mas pode haver uma latência substancial, pois um evento pode acontecer imediatamente após uma verificação, caso em que ele esperará quase um intervalo inteiro de verificação. Na média, a latência representa metade do intervalo de verificação.

Os temporizadores por software (soft timers) evitam interrupções. Em vez disso, sempre que o núcleo está executando por alguma outra razão, imediatamente antes de retornar para o modo do usuário ele verifica o relógio e tempo real para ver se um temporizador por software expirou. Se ele expirou, o evento escalonado é realizado sem a necessidade de chavear para o modo núcleo, dado que o sistema já está ali. Após o trabalho ter sido realizado, o temporizador por software é reinicializado novamente. Tudo o que precisa ser feito é copiar o valor do relógio atual para o temporizador e acrescentar o intervalo de tempo a ele.

Interfaces com o usuário: teclado, mouse, monitor

Em computadores de grande porte, frequentemente há muitos usuários remotos, cada um com um dispositivo contendo um teclado e um monitor conectados. Esses dispositivos foram chamados historicamente de terminais.

Software de entrada

As informações do usuário vêm fundamentalmente do teclado e do mouse. Uma interrupção é gerada sempre que uma tecla é pressionada e uma segunda é gerada sempre que uma tecla é liberada. Em cada uma dessas interrupções, o driver do teclado extrai as informações sobre o que acontece na porta de E/S associada com o teclado. Todo o restante acontece no software e é bastante independente do hardware.

Software de teclado

O número no registrador de E/S é o número da tecla, chamado de código de varredura, não o código de ASCII. Teclados normais têm menos de 128 teclas, então apenas 7 bits são necessários para representar o 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). Assim, tudo o que o hardware faz é gerar interrupções de pressão e liberação. O software faz o resto.

Duas filosofias possíveis podem ser adotadas para o driver. Na primeira, seu trabalho é apenas aceitar a entrada e passá-la adiante sem modificá-la. Um programa lendo a partir do teclado recebe uma sequência bruta de códigos ASCII.

A segunda filosofia: o driver lida com toda a edição interna da linha e entrega apenas as linhas corrigidas para os programas do usuário. A primeira filosofia é baseada em caracteres; a segunda em linhas. Na origem elas eram referidas como modo cru (raw mode) e modo cozido (cooked mode), respectivamente. O padrão POSIX usa o termo menos pitoresco modo canônico para descrever o modo baseado em linhas. O modo não canônico é o equivalente ao modo cru.

Se o teclado estiver em modo canônico (cozido), os caracteres devem ser armazenados até que uma linha inteira tenha sido acumulada, pois o usuário pode subsequentemente decidir apagar parte dela. Mesmo que o teclado esteja em modo cru, o programa talvez ainda não tenha solicitado uma entrada, então os caracteres precisam ser armazenados para viabilizar

a digitação antecipada. Um buffer dedicado pode ser usado ou buffers podem ser alocados de um reservatório (pool). No primeiro tipo, a digitação antecipada tem um limite; no segundo, não.

Embora o teclado e o monitor sejam dispositivos logicamente separados, muitos usuários se acostumaram a ver somente os caracteres que eles recém-digitaram aparecer na tela. Esse processo é chamado de eco.

Logicamente, ao final de uma linha de texto, você quer um CR a fim de mover o cursor de volta para a coluna 1, e um caractere de alimentação de linha para avançar para a próxima linha. Exigir que os usuários digitassem ambos ao final de cada linha não venderia bem. Cabe ao driver do dispositivo converter o que entrar para o formato usado pelo sistema operacional. No UNIX, a tecla Enter é convertida para uma alimentação de linha para armazenamento interno; no Windows ela é convertida para um CR seguido de uma alimentação de linha.

Não importa qual seja a convenção interna, o monitor pode exigir que tanto uma alimentação de linha quanto um CR sejam ecoados a fim de obter uma atualização adequada da tela.

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 do mouse

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.

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.

Software de saída

Primeiro examinaremos uma saída simples para uma janela de texto, que é o que os programadores em geral preferem usar. Então consideraremos interfaces do usuário gráficas, que outros usuários muitas vezes preferem.

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.

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.

O 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.

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.

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.

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.

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.

Também 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, controla a criação, remoção e movimento das janelas na tela.

Um conceito fundamental em X é o recurso (resource). 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.

Interfaces gráficas do usuário
A maioria dos computadores pessoais oferece uma interface gráfica do usuário (Graphical User Interface — GUI).

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.

O software GUI pode ser implementado como código no nível do usuário, como nos sistemas UNIX, ou no próprio sistema operacional, como no caso do Windows.

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.

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.

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.

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.

WndProc, que lida com várias mensagens que podem ser enviadas para a janela. Há duas maneiras de o Windows conseguir que um programa faça algo. Uma é postar uma mensagem para sua fila de mensagens. Esse método é usado para a entrada do teclado, entrada do mouse e temporizadores que expiraram. A outra maneira, enviar uma mensagem para a janela, consiste no Windows chamar diretamente o próprio WndProc. Esse método é usado para todos os outros eventos. Tendo em vista que o Windows é

notificado quando uma mensagem é totalmente processada, ele pode abster-se de fazer uma nova chamada até que a anterior tenha sido concluída. Desse modo as condições de corrida são evitadas.

Resumindo, um programa para o Windows normalmente cria uma ou mais janelas com um objeto-classe para cada uma. Associado com cada programa há uma fila de mensagens e um conjunto de rotinas tratadoras. Em última análise, o comportamento do programa é impelido por eventos que chegam, que são processados pelas rotinas tratadoras. Esse é um modelo do mundo muito diferente da visão mais procedimental adotada pelo UNIX.

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 (GDI — Graphics Device Interface). 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.

Existem várias maneiras para adquirir o contexto do dispositivo. Um exemplo simples da sua aquisição e uso é

```
hdc = GetDC(hwnd);
TextOut(hdc, x, y, psText, iLength);
ReleaseDC(hwnd, hdc);
```

A primeira instrução obtém um nome para o contexto do dispositivo, hdc. A segunda usa o contexto do dispositivo para escrever uma linha de texto na tela, especificando as coordenadas (x, y) de onde começa a cadeia, um ponteiro para a cadeia em si e seu comprimento. A terceira chamada libera o contexto do dispositivo para indicar que o programa terminou de desenhar por ora.

Outra nota interessante é que, quando hdc é adquirido dessa maneira, o programa pode escrever apenas na área do cliente da janela, não na barra de títulos e outras partes dela. Internamente, na estrutura de dados do contexto do dispositivo, uma região de pintura é mantida. Qualquer desenho fora dessa região é ignorado.

As rotinas de desenho caem em quatro categorias: traçado de linhas e curvas, desenho de áreas preenchidas, gerenciamento de mapas de bits e apresentação de texto.

Mapas de bits (bitmaps)

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.

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.

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). Existem muitos recursos para manipular os bitmaps no Windows.

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).

BitBlt pode fazer mais que somente copiar mapas de bits. O último parâmetro proporciona a possibilidade de realizar operações Booleanas a fim de combinar o mapa de bits fonte com o mapa de bits destino. Por exemplo, a operação lógica OU pode ser aplicada entre a origem e o destino para se fundir com ele. Também pode ser usada a operação OU EXCLUSIVO, que mantém as características tanto da origem quanto do destino.

Um problema com mapas de bits é que eles não são extensíveis. Por essa razão, o Windows também suporta uma estrutura de dados chamada mapa de bits independente de dispositivo (Device Independent Bitmap — DIB).

Fontes
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, podemos nos certificar de que os caracteres exibidos na tela ou impressos na impressora serão o mais próximos quanto possível, diferenciando-se somente na precisão do erro.

Clientes magros (thin clients)

Uso de sistema centralizado
– Poucas máquinas precisam ser atualizadas
– Backups feitos de forma mais fácil

– Compartilhamento de recursos e melhor utilização

Dizer que a maioria dos usuários quer uma computação interativa de alto desempenho, mas não quer realmente administrar um computador, provavelmente seja uma conclusão justa. Isso levou os pesquisadores a reexaminar os sistemas de tempo compartilhado usando terminais burros (agora educadamente chamados de clientes magros) que atendem às expectativas de terminais modernos.

Um dos clientes magros mais conhecidos é o Chromebook. Ele é ativamente promovido pelo Google, mas com uma ampla variedade de fabricantes fornecendo uma ampla variedade de modelos. O notebook executa ChromeOS que é baseado no Linux e no navegador Chrome Web. Presume-se que esteja on-line o tempo inteiro. A maior parte dos outros softwares está hospedada na web na forma de Web Apps, fazendo a pilha de software no próprio Chromebook ser consideravelmente menor do que na maioria dos notebooks tradicionais.

Gerenciamento de energia

Vamos começar com os PCs de mesa. Um PC de mesa muitas vezes tem um suprimento de energia de 200 watts. A outra situação em que a energia é uma questão importante é nos computadores movidos a bateria, incluindo notebooks, laptops, palmtops e Webpads, entre outros. O cerne do problema é que as baterias não conseguem conter carga suficiente para durar muito tempo, algumas horas no máximo.

No nível mais baixo os vendedores de hardware estão tentando tornar os seus componentes eletrônicos mais eficientes em termos de energia. As técnicas usadas incluem a redução do tamanho de transistores, o escalonamento dinâmico de tensão, o uso de barramentos adiabáticos e low-swing, e técnicas similares.

Existem duas abordagens gerais para reduzir o consumo de energia. A primeira é o sistema operacional desligar partes do computador (na maior parte dispositivos de E/S) quando elas não estão sendo usadas, pois um dispositivo que está desligado usa pouca ou nenhuma energia. A segunda abordagem é o programa aplicativo usar menos energia, possivelmente degradando a qualidade da experiência do usuário, a fim de estender o tempo da bateria.

Questões de hardware

As baterias vêm em dois tipos gerais: descartáveis e recarregáveis. As baterias descartáveis (mais comumente as do tipo AAA, AA e D) podem ser usadas para executar dispositivos de mão, mas não têm energia suficiente para suprir notebooks com telas grandes e reluzentes. Uma bateria recarregável, por sua vez, pode armazenar energia suficiente para suprir um notebook por algumas horas.

A abordagem geral que a maioria dos vendedores de computadores escolhe para a conservação da bateria é projetar a CPU, memória e dispositivos de E/S para terem múltiplos estados: ligados, dormindo, hibernando e desligados. Para usar o dispositivo, ele

precisa estar ligado. Quando o dispositivo não for necessário por um curto período de tempo, ele pode ser colocado para dormir, o que reduz o consumo de energia. Quando se espera que ele não seja necessário por um intervalo de tempo mais longo, ele pode ser colocado para hibernar, o que reduz o consumo da energia ainda mais. A escolha que precisa ser feita aqui é que tirar um dispositivo da hibernação muitas vezes leva mais tempo e dispende mais energia do que tirá-lo do estado de adormecimento. Por fim, quando um dispositivo está desligado, ele não faz nada e não consome energia.

Pesquisas apontaram que os três maiores consumidores de energia eram a tela, o disco rígido e a CPU, nessa ordem.

Questões do sistema operacional

O sistema operacional tem um papel fundamental no gerenciamento da energia. Ele controla todos os dispositivos, por isso tem de decidir o que desligar e quando fazê-lo. Se desligar um dispositivo e esse dispositivo for necessário imediatamente de novo, pode haver um atraso incômodo enquanto ele é reiniciado. Por outro lado, se ele esperar tempo demais para desligar um dispositivo, a energia é desperdiçada por nada.

Monitor
Para obter uma imagem clara e nítida, a tela deve ser iluminada por trás e isso consome uma energia substancial. Muitos sistemas operacionais tentam poupar energia desligando o monitor quando não há atividade alguma por um número determinado de minutos.

Uma melhoria possível foi proposta por Flinn e Satyanarayanan (2004). Eles sugeriram que o monitor consista em um determinado número de zonas que podem ser ligadas ou desligadas independentemente.

Disco rígido
Ele consome uma energia substancial para manter-se girando em alta velocidade, mesmo que não haja acessos. Muitos computadores, em especial notebooks, param de girar o disco após determinado número de minutos com ele ocioso.

Outra maneira de poupar energia é ter uma cache de disco substancial na RAM. Se um bloco necessário está na cache, um disco ocioso não precisa ser ativado para satisfazer a leitura. Similarmente, se uma escrita para o disco pode ser armazenada na cache, um disco parado não precisa ser ativado apenas para lidar com a escrita.

Outra maneira de evitar que um disco seja ativado desnecessariamente é fazer que o sistema operacional mantenha os programas em execução informados a respeito do estado do disco enviando-lhes mensagens ou sinais. Alguns programas têm escritas programadas que podem ser “puladas” ou postergadas. Por exemplo, um editor de texto pode ser ajustado para escrever o arquivo que está sendo editado para o disco de tempos em tempos. Se o editor de texto sabe que o disco está desligado naquele momento em que ele normalmente escreveria o arquivo, pode postergar essa escrita até que o disco esteja ligado.

A CPU
O software pode colocar a CPU de um notebook para dormir, reduzindo o uso de energia a quase zero. A única coisa que ela pode fazer nesse estado é despertar quando ocorre uma interrupção. Portanto, sempre que a CPU fica ociosa, seja esperando por E/S ou porque não há trabalho para fazer, ela vai dormir.

Em muitos computadores, há uma relação entre a voltagem da CPU, ciclo do relógio e consumo de energia. A voltagem da CPU muitas vezes pode ser reduzida no software, o que poupa energia, mas também reduz o ciclo do relógio (mais ou menos linearmente). Tendo em vista que a energia consumida é proporcional ao quadrado da voltagem, cortar a voltagem pela metade torna a CPU cerca de 50% mais lenta, mas a ¼ da energia.

Essa propriedade pode ser explorada para programas com prazos bem definidos, como programas de visualização multimídia que devem descomprimir e mostrar no vídeo um quadro a cada 40 ms e ficam ociosos se o fazem mais rapidamente.

De maneira similar, se um usuário está digitando 1 caractere/s, mas o trabalho necessário para processar o caractere leva 100 ms, é melhor para o sistema operacional detectar os longos períodos ociosos e desacelerar a CPU por um fator de 10. Resumindo, executar lentamente é mais eficiente em termos de consumo de energia do que executar rapidamente.

De maneira interessante, reduzir a velocidade de núcleos da CPU nem sempre implica uma redução no desempenho. Por exemplo, imagine uma CPU com diversos núcleos rápidos, onde um núcleo é responsável pela transmissão dos pacotes de rede em prol de um produtor executando em outro núcleo. O produtor e a pilha de rede comunicam-se diretamente via uma memória compartilhada e ambos executam em núcleos dedicados. O produtor realiza uma quantidade considerável de computação e não consegue acompanhar o núcleo da pilha de rede. Uma solução mais atraente é executar a pilha de rede em um núcleo mais lento, de maneira que ele esteja constantemente ocupado (e desse modo, nunca durma), enquanto ainda reduz o consumo de energia.

A memória
Existem duas opções possíveis para poupar energia com a memória. Primeiro, a cache pode ser esvaziada e então desligada. Uma opção mais drástica é escrever os conteúdos da memória principal para o disco, então desligar a própria memória.

Comunicação sem fio
O cerne da sua solução explora o fato de que os computadores móveis comunicam-se com estações-base fixas que têm grandes memórias e discos e nenhuma restrição de energia. O que eles propõem é que o computador móvel envie uma mensagem para a estação-base quando estiver prestes a desligar o rádio. Daquele momento em diante, a estação-base armazena em seu disco as mensagens que chegarem. O computador móvel pode indicar explicitamente quanto tempo ele está planejando dormir, ou simplesmente informar a estação-base quando ele ligar o rádio novamente. A essa altura, quaisquer mensagens acumuladas podem ser enviadas para ele.

Mensagens de saída que são geradas enquanto o rádio está desligado são armazenadas no computador móvel. Se o buffer ameaça saturar, o rádio é ligado e a fila, transmitida para a estação-base.

Quando o rádio deve ser desligado? Uma possibilidade é deixar o usuário ou o programa de aplicação decidir. Outra é desligá-lo após alguns segundos de tempo ocioso. Quando ele deve ser ligado novamente? Mais uma vez, o usuário ou o programa poderiam decidir, ou ele poderia ser ligado periodicamente para conferir o tráfego que chega e transmitir quaisquer mensagens na fila. É claro, ele também deve ser ligado quando o buffer de saída está quase cheio. Várias outras heurísticas são possíveis.

Gerenciamento térmico

As CPUs modernas ficam extremamente quentes por causa de sua alta velocidade. Computadores de mesa em geral têm um ventilador elétrico interno para soprar o ar quente para fora do gabinete. Dado que a redução do consumo de energia normalmente não é uma questão fundamental com os computadores de mesa, o ventilador em geral está ligado o tempo inteiro.

Com os notebooks a situação é diferente. O sistema operacional tem de monitorar a temperatura continuamente. Quando chega próximo da temperatura máxima permitida, o sistema operacional tem uma escolha. Ele pode ligar o ventilador, o que faz barulho e consome energia. Alternativamente, ele pode reduzir o consumo de energia reduzindo a iluminação da tela, desacelerando a CPU, sendo mais agressivo no desligamento do disco, e assim por diante.

Gerenciamento de bateria

Os dispositivos móveis usam baterias inteligentes agora, que podem comunicar-se com o sistema operacional. Mediante uma solicitação do sistema operacional, elas podem dar retorno sobre coisas como a sua voltagem máxima, voltagem atual, carga máxima, carga atual, taxa de descarga máxima e mais. A maioria dos dispositivos móveis tem programas que podem ser executados para obter e exibir todos esses parâmetros.

Alguns notebooks têm múltiplas baterias. Quando o sistema operacional detecta que uma bateria está prestes a se esgotar, ele deve desativá-la e ativar outra graciosamente, sem causar nenhuma falha durante a transição. Quando a bateria final está nas suas últimas forças, cabe ao sistema operacional avisar o usuário e então provocar um desligamento ordeiro, por exemplo, certificando-se de que o sistema de arquivos não seja corrompido.

Interface do driver

Vários sistemas operacionais têm um mecanismo elaborado para realizar o gerenciamento de energia chamado de interface avançada de configuração e energia (Advanced Configuration and Power Interface — ACPI). O sistema operacional pode enviar quaisquer comandos para o driver requisitando informações sobre as capacidades dos seus dispositivos e seus estados atuais. Essa característica é especialmente importante quando combinada com a característica plug and play, pois logo após a inicialização, o sistema operacional não faz nem ideia de quais dispositivos estão presentes, muito menos suas propriedades em relação ao consumo ou modo de gerenciamento de energia.

Ele também pode enviar comandos para os drivers instruindo-os para cortar seus níveis de energia (com base nas capacidades a respeito das quais ele ficou sabendo antes, é claro). Há também algum tráfego no outro sentido. Em particular, quando um dispositivo como um teclado ou um mouse detecta atividade após um período de ociosidade, esse é um sinal para o sistema voltar à operação (quase) normal.

Questões dos programas aplicativos

Dizer aos programas para usarem menos energia, mesmo que isso signifique fornecer uma experiência do usuário de pior qualidade. Cabe aos programas então decidirem entre degradar o desempenho para estender a vida da bateria, ou manter o desempenho e arriscar ficar sem energia. Uma questão que surge é como um programa pode degradar o seu desempenho para poupar energia.

Exemplos:

O primeiro programa mensurado foi um reprodutor de vídeo. No modo sem degradação, ele reproduz 30 quadros/s em uma resolução total e em cores. Uma forma de degradação é abandonar a informação das cores e exibir o vídeo em preto e branco. Outra forma de degradação é reduzir a frequência de reprodução, o que faz a imagem piscar (flicker) e deixa o filme com uma qualidade irregular. Ainda outra forma de degradação é reduzir o número de pixels em ambas as direções, seja reduzindo a resolução espacial ou tornando a imagem exibida menor. Medidas desse tipo pouparam em torno de 30% da energia.

O segundo programa foi um reconhecedor de voz. Ele coletava amostras do microfone e construía um modelo de onda. Esse modelo de onda podia ser analisado no notebook ou ser enviado por um canal de rádio para análise em um computador fixo. Fazer isso poupa energia da CPU, mas usa energia para o rádio. A degradação foi conseguida usando um vocabulário menor e um modelo acústico mais simples. O ganho aqui foi de aproximadamente 35%.

O exemplo seguinte foi um programa visualizador de mapas que buscava o mapa por um canal de rádio. A degradação consistia em cortar o mapa para dimensões menores ou dizer ao servidor remoto para omitir estradas menores, desse modo exigindo menos bits para serem transmitidos. Mais uma vez aqui foi conseguido um ganho de aproximadamente 35%.

O quarto experimento foi com a transmissão de imagens JPEG para um navegador na internet. O padrão JPEG permite vários algoritmos, negociando entre a qualidade da imagem e o tamanho do arquivo. Aqui o ganho foi em média 9%. Ainda assim, como um todo, os experimentos mostraram que ao aceitar alguma degradação de qualidade, o usuário pode dispor de uma determinada bateria por mais tempo.