Sistemas Operacionais

Consiste de um conjunto de programas que compõe o software básico do computador e cuja finalidade é de executar os programas aplicativos e de servir de interface entre os computadores e seus usuários. Um sistema operacional deve atender a três objetivos principais:

- **Conveniência**: Tornar o uso do computador mais fácil;
- **Eficiência**: Tornar eficiente (seguro e justo) o uso do compartilhamento dos recursos existentes;
- **Evolução**: Possibilitar o constante debug e o desenvolvimento de novas funcionalidades;

Sistema operacional é um conjunto de programas.
Controlar o hardware e o software.
Compartilhamento de recursos (hardware, software)

USUÁRIO → SISTEMA OPERACIONAL → HARDWARE

Em uma abordagem mais macro o Sistema Operacional pode ser visto como a primeira camada de software acima do hardware do computador que se encarrega de suportar e servir de interface entre este e os programas aplicativos e utilitários.

É o software que mais está em contato com o hardware.

Hardware ← Sistema operacional → Aplicativos → Usuário

O Sistema Operacional é composto por módulos que se encarregam da comunicação, alocação e gerenciamento de recursos específicos tais como:
Quais são os recursos que o sistema operacional gerencia?

- Processo = Programa em execução
- Memória
- Arquivos
- Entrada e Saída = Comunicação com periféricos de entrada e saída
- Interconexão = Rede, conectividade
- Alocação da UCP = (**Escalonamento = troca entre os processos**)
- Segurança = Dados / Malwares
- Interface com o usuário = Comunicação com o usuário (telas)

A interface entre os programas aplicativos (processos) e o sistema operacional é realizada através de comandos (instruções) de chamadas do sistema (as system calls)

Usuário → Comandos → Sistema operacional

Sistema operacional → System Calls → Programas

Chamadas de sistema = System callls

Usuário envia pedido para imprimir = Comando
O **Sistema operacional** pede para o driver exibir uma imagem no monitor = System calls

As System calls podem ser genericamente agrupadas em cinco categorias:

1. Controle de **processo** (Programas em execução)
2. Manipulação de **arquivos**
3. Manipulação de **dispositivos**
4. Informações de **manutenção** (Comunicação interna)
5. **Comunicações** (Redes)

Resumindo:

Função do Sistema Operacional: é gerenciar os **componentes** e fornecer aos **programas** do usuário uma interface com **hardware** mais **simples**.

Área de atuação do sistema operacional

Sistema Bancário	Reserva de Passagens áreas	Visualizador web	Programas de **Aplicação**
Compiladores	Editores	Interpretador de comandos	Programas do Sistema
Sistema Operacional			
Linguagem de máquina **(ISA)**			Hardware
Micro arquitetura **(ULA/UC/Registradores)**			
Dispositivos físicos			

Programas de aplicação = Sistema bancário, Navegador, programas do usuário
Programas de sistema = Compiladores, Editores, Interpretadores

Dispositivos físicos = Monitor, Hd, ssd, RAM, impressora

→ Nível de micro arquitetura → nível no quais os dispositivos físicos são agrupados em unidades funcionais. Normalmente, esse nível contém alguns registradores internos à CPU (unidade central de processamento) e um caminho de dados (data path) contendo uma unidade lógico-aritmética.

Nível de hardware que agrupa as funções do **processador**
→ **ULA**
→ **UC**
→ **Registradores**
→ **Caminho de dados (data patch)**

*** Função do Caminho de dados:** em algumas máquinas, a operação do caminho de dados é controlada por um software denominado microprogramas, em outras é controlada diretamente pelo hardware e sua função é executar um determinado conjunto de instruções.

Pré-Fetch → Pré busca
Fetch → Busca
Decode → Decodificação
Data fetch → Busca dos dados
Execute → Execução
Write back → Resposta

→ Sequência em que as instruções são executadas:

PréFetch Fetch Decode DataFetch Execute WriteBack
06 05 04 03 02 01

→ **Pipeline** = Quando as unidades internas do processador não ficam ociosas
→ **Superescalar** = Quando o processador possui mais de uma unidade de cada

2 unidades = Superescalar de dupla canalização.

PréFetch Fetch Decode DataFetch Execute WriteBack
06 05 04 03 02 01

PréFetch Fetch Decode DataFetch Execute WriteBack
12 11 10 09 08 07

→ **Nível ISA:** Muitas vezes denominado Linguagem de máquina **(arquitetura do conjunto de instruções):** é formado pelo hardware mais as instruções visíveis a um programador de linguagem de montagem. Esse nível muitas vezes é denominado linguagem de máquina. Nesse nível, os dispositivos de entrada e saída são controlados carregando-se valores em registradores de dispositivo.

→ Nível da linguagem de máquina (Assembly)
→ **Hardware (E/S)**
→ **Instruções visíveis** ao **programador**
→ Dispositivos de **entrada** e **saída**

Sistema Operacional: Consiste em uma camada de software que oculta (parcialmente) o hardware e fornece ao programador um conjunto de instruções mais conveniente. É normalmente executada em modo supervisor ou modo núcleo (no caso, a parte mais interna de um sistema operacional). Ele é protegido dos usuários pelo hardware. Realizam basicamente duas funções não relacionadas: estender a máquina e gerenciar recursos.

Ocultar o hardware = Não mostrar exatamente o que o hardware está executando para facilitar o uso do usuário.

A função do sistema operacional como uma máquina estendida

É apresentar ao usuário o equivalente a uma máquina estendida ou máquina virtual mais fácil de programar do que o hardware, fornecendo uma variedade de serviços que os programas podem obter usando instruções especiais conhecidas como chamadas ao sistema.

Cria uma máquina "virtual" = Computador de mentira

A função do sistema operacional como um gerenciador de recursos

Fornecer uma alocação ordenada e controlada de processadores, memórias e dispositivos de E/S entre vários programas que competem por eles, ou seja, manter o controle sobre quem está usando qual recurso, garantindo suas requisições de recursos, controlando as contas e mediando conflitos de requisições entre diferentes programas e usuários. O gerenciamento de recursos realiza o compartilhamento (ou multiplexação) no tempo, diferentes programas ou usuários aguardam sua vez de usá-lo, e de espaço em que cada um ocupa uma parte do recurso.

Compartilhamento = Multiplexação

Compartilhamento de tempo = O processador fica um tempo determinado em cada recurso.

Programa 1	Programa 2	Programa 3
1/3 tempo	1/3 tempo	1/3 tempo

Compartilhamento de espaço = O processador se divide para todos os processos

Programa 1	Programa 2	Programa 3
1/3 Processamento	1/3 Processamento	1/3 Processamento

EXPLICADORES.NET
(55) 21 99461-8818
EXPLICADORES.NET WWW.EXPLICADORES.NET.BR

Modos de execução do sistema operacional

Kernel → Núcleo do Sistema operacional

Modo Usuário
→ Modo limitado com relação aos recursos;
→ Somente uma parte do Kernel pode ser acessada (Parte superficial)
→ Para executar operações simples

Modo Núcleo / Kernel / Supervisor
→ Funções mais específicas podem ser executadas;

Compiladores e Editores: executados em modo usuário.

Função do Sistema Operacional: Controla recursos usando a **multiplexação/compartilhamento de tempo e espaço**.

Histórico dos sistemas Operacionais

1º Geração – Válvulas e painéis de programação: Toda a programação era feita em código absoluto e muitas vezes conectando **plugs** e **painéis** para controlar as funções básicas da máquina. Não havia linguagens de programação (nem mesmo a linguagem de montagem existia).
Foi introduzido as perfuradoras de cartões.

2º Geração – Transistores e sistemas em lote (batch): Surgimento da programação em bath (lote). Os grandes computadores de segunda geração foram usados, em sua maioria, para cálculos científicos, como equações diferenciais, muito freqüentes na física e na engenharia. Eles eram preponderantemente programados em **FORTRAN** e em linguagem de montagem. Os sistemas operacionais típicos eram o **FMS** (Fortran Monitor System) e o **IBSYS**, sistema operacional da IBM para o 7094.

Batch = Lote de processos
Sistema em batch juntávamos vários processos para serem executados de uma vez “máquina de lavar”.
Não interativo = Não temos como prever quando o processamento irá acontecer.
Monoprogramáveis = Somente um programa(lote) é executado por vez.

3º Geração – CIs e Multiprogramação: Popularização de várias técnicas fundamentais nos sistemas operacionais de segunda geração, a mais importante: Multiprogramação. Outro aspecto importante nos sistemas operacionais de terceira geração era a capacidade de transferir jobs de cartões perfurados para disco magnéticos, assim que um job fosse completado, o sistema operacional poderia carregar um novo job a partir do disco apenas nessa partição que acabou de ser liberada e então processá-lo. Essa técnica é denominada spooling (simultaneous peripheral operation online).

Circuitos integrados
Multiprogramação = Onde várias tarefas podem ser executadas ao mesmo tempo.
Jobs = Trabalhos = Processos

Cartão perfurado → Hd (Hard Disk) → UCP
(Spooling)

A série 360 foi a primeira linha de computadores a usar circuitos integrados em pequena escala, propiciando assim uma melhor relação custo e beneficio em comparação à segunda geração de máquinas, construídas com transistores individuais.

- Surgiu o conceito de Compartilhamento de tempo – **TimeSharing**
- O primeiro sistema importante de timesharing o CTSS (capatible time sharing system – sistema de compartilhamento compatível) foi desenvolvido pelo **MIT** em um 7094 modificado.

4º Geração – Computadores Pessoais: Surgiram os **PC´s**.
Desenvolvimento de circuitos integrados em larga escala (LSI)
Em termos de arquitetura, os computadores pessoais (inicialmente denominados microcomputadores) não eram muito diferentes dos minicomputadores da classe PDP-11.
8080 – Lançado em 1974 pela Intel foi o primeiro processador a ter oito bits de propósito gerais e o primeiro microcomputador a ter unidade de disco flexível.

Obs.: a) **Primeira versão do Windows** que é totalmente independente do DOS: **Windows 95**.
b) **Windows 95 / 98** ainda contém uma grande quantidade de código em 16 bits.
c) Windows NT: 32 bits.

Diversos tipos de sistemas Operacionais podem se identificados: **monoprogramáveis**, **multiprogramáveis**, **multiprocessáveis**, **sistemas de rede**, **sistemas distribuídos**, **sistemas em lote** (batch), **sistemas de tempo compartilhado** (time sharing) e **de tempo real**.

Sistemas Monoprogramáveis:
Só executam uma tarefa por vez.
Se caracterizam pela execução de uma única tarefa (processo) por vez, sendo que todos os recursos (processador, memória e periféricos) ficam exclusivamente a eles dedicados. Nesses sistemas, enquanto o programa aguarda a ocorrência de um evento qualquer, o processador ficará ocioso (idle); a memória ficará subtilizada, caso o programa não ocupe totalmente e os periféricos também ficarão ociosos se não utilizados.

Sistemas Multiprogramáveis:
Executam "várias tarefas" por vez
Através das trocas rápidas parace que está sendo tudo executado ao mesmo tempo;
Se caracterizam por permitir que vários programas (tarefas) **residam** **"simultaneamente"** na memória e "**concorram" pelo uso dos recursos disponíveis** (apenas um programa detém, num determinado instante o controle da UCP). O sistema operacional se encarrega de gerenciar o acesso concorrente das diversas tarefas aos diversos recursos, de forma ordenada e protegida. O throughput do sistema melhora, isto é, o número de processos concluídos por unidade de tempo aumenta. Os sistemas multiprogramáveis oferecem condições de maior eficiência computacional que um sistema monoprogramável.

Executar várias tarefas ao mesmo tempo, várias tarefas ficam na memória ao mesmo tempo, onde **concorrem** entre si para utilizar o processador.

Compartilhamento de tempo ou de espaço.

Throughput = Esse tipo de sistema precisa encontrar maneiras de aumentar a vazão (throughput)
É melhor que o throughput seja maior...

Um sistema multiprogramável pode ser dos tipos: lote (batch), tempo compartilhado (time sharing) ou tempo real (real time), sendo que o único sistema pode suportar um ou mais desses tipos de processamento.

EXPLICADORES.NET
(55) 21 99461-8818
EXPLICADORES.NET WWW.EXPLICADORES.NET.BR

- **Sistemas Batch:** Não há iteração com o usuário e os programas armazenados vão sendo executados na medida que haja disponibilidade de recursos.
  - **Não há interação = Não interativo = Não podemos saber quando um job será executado.**
  - **Não sabemos quando será executado exatamente;**

Acumula um lote e envia o lote inteiro para ser executado de uma única vez.

- Sistema em lote;
- Semelhante a máquina de lavar;

JOB → LOTE → SISTEMA OPERACIONAL → CPU

- **Sistemas Time Sharing:** Surgiram com o aparecimento dos terminais de vídeo que permitiam ao usuário compartilhar a distância o uso de recursos do computador. Dessa forma o usuário acessa e interage com o computador tanto na fase de desenvolvimento quanto na fase de execução e análise dos resultados. Não só o processador é compartilhado neste sistema, mas também a memória e os periféricos, como discos e impressoras. O sistema cria para o usuário um ambiente de trabalho próprio, dando a impressão de que todo o sistema está dedicado, exclusivamente a ele.
  - **Sistema interativos = Podemos prever quando o processo será executado, quando executamos o programa será atendido imediatamente pelo processador;**

Tempo compartilhado

Processador fica um tempo em cada tarefa

Tempo para cada tarefa = Fatia de tempo = Time slice

Programa 1	Programa 2	Programa 3
1ms	1ms	1ms

Time slice = 1ms

- **Tempo real** São caracterizados por terem o tempo como um parâmetro fundamental. Se as ações precisam necessariamente ocorrer em determinados instantes ou em determinados intervalos de tempo tem-se então um sistema de tempo real, mas se o descumprimento ocasional de um prazo é aceitável temos um sistema de tempo real não crítico;

Real – Time = Sistemas que o **tempo** é o fator determinante.

Exemplos :

- Lançamento de mísseis
- Multimedia
- Compra de passagens
- Compra de ingressos

Sistemas de tempo real crítico = Sistema de tempo real onde se o tempo falhar, coisas graves acontecem. (Lançamento de mísseis, compra de ingressos e passagens)

Sistemas de tempo real não crítico = Sistema de tempo real onde se o tempo falhar, não acontecem coisas graves (Sistemas multimedia)

- **Sistemas Operacionais de rede:** Em um sistema operacional de redes os usuários sabem da existência de múltiplos computadores e podem se conectar-se a máquinas remotas e copiar arquivos de uma máquina para outra. Cada máquina executa seu próprio sistema operacional local e tem seu próprio usuário local. Precisa de um controlador de interface de rede e de software de baixo nível para controlá-la, bem como programas para conseguir sessões remotas e também ter acesso remoto a arquivos, as esses sem acréscimos não alteram a estrutura essencial do sistema operacional.

Sistemas operacionais que se conectam com as redes.

Outros tipos de SO:

- **Grande porte**
  - Normalmente oferece três tipos de serviços: batch (lote – processa Jobs sem a presença interativa do usuário), tempo compartilhado (permitem que múltiplos usuários remotos executem seus Jobs simultaneamente no computador, como na realização de consultas a um grande banco de dados), processamento de transações (administram uma grande quantidade de pequenas requisições);

Mainframes = Servidor que armazena todo o poder de processamento e memória além dos arquivos da rede.
Terminais Burros = Terminal que não possui memória e processador

- **Servidores**: Servem múltiplos usuários de uma vez em uma rede e permitem-lhes compartilhar recursos de hardware e de software. Servidores podem fornecer serviços de impressão, serviços de arquivos ou serviços Web. Provedores de acesso a internet utilizam várias máquinas servidores para dar suporte a seus clientes.

Sistemas operacionais para servidores
→ Windows NT Server
→ Windows 2000 Server
→ Linux Server

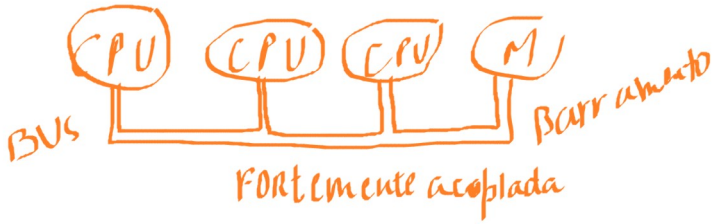
Sistemas Multiprocessados
Caracterizam-se por permitir a execução simultânea de duas ou mais instruções, o que requer a existência de mais de um processador. O multiprocessamento mantém todos os conceitos da multiprogramação agora aplicados a vários processadores ao mesmo tempo.

Sistemas que aceitam vários **processadores** ao mesmo tempo.
Processadores com **vários núcleos(multi core)** são considerados **multiprocessados**.

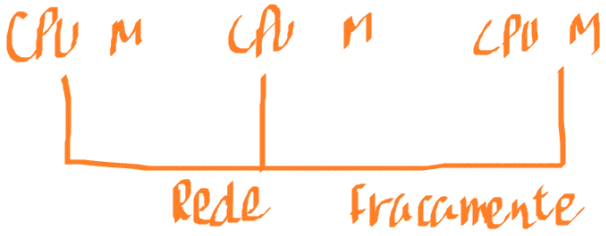
O multiprocessamento pode ser obtido pela configuração de múltiplos processadores que compartilham de uma mesma memória primária (fortemente acoplada) ou de múltiplos computadores independentes do tipo de sistemas em rede e sistemas distribuídos (fracamente acoplado), onde cada um tem seus próprios recursos.

Tipos de multiprocessados:

Fortemente acoplados
Os processadores e as memórias compartilham o mesmo barramento.
Mesma placa mãe
Multiprocessadores



Fracamente acoplados
Os processadores e as memórias compartilham a mesma rede.
Mesma rede LAN ou WAN
Multicomputador

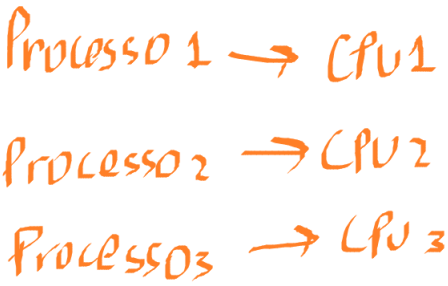


Os sistemas multiprocessados permitem que vários programas sejam executados em paralelo (granularidade grossa) , ou que um programa tenha duas ou mais de suas instruções executadas em paralelo (granularidade fina)

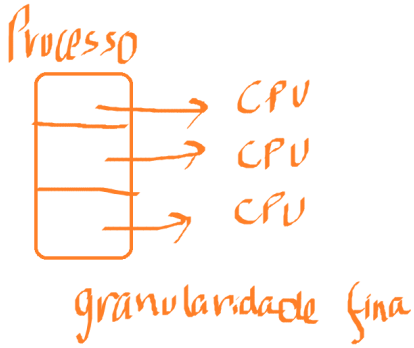
Também conhecidos como servidores paralelos, multicomputadores ou multiprocessadores; Precisam de sistemas operacionais especiais, mas muitos deles são variações dos sistemas operacionais de servidores com aspectos especiais de comunicação e conectividade;

Executar vários programas ao mesmo tempo.

Cada processador com um programa
Granularidade grossa



Cada processador com uma fatia(instruções) de um programa único
Granularidade fina



Sistemas Distribuídos

É, conceitualmente, um sistema em rede que possibilita a integração e a cooperação transparente dos diversos nós que compõem a rede. Dessa forma, sob o enfoque dos usuários e das tarefas, o sistema é único e se comporta como uma arquitetura multiprocessada possibilitando tanto paralelismo de granularidade fina como grossa.

- Sistema muliprocessadores
- Multicomputador (Vários computadores funcionando como 1)
- Granularidade grossa;
- Granularidade fina;

- **Computadores pessoais;**

Sistemas operacionais voltados para atender um único usuário

- **Embarcados** São executados em computadores que controlam dispositivos que geralmente não são considerados computadores, como aparelho de **TV**, **fornos de microondas** e **telefones móveis**. Eles têm muitas vezes, características de sistemas de **tempo real**, mas também apresentam restrições de tamanho, memória e de consumo de energia o que os fazem especiais.

Instalado em componentes que não são computadores.

- TV;
- Microondas;
- Geladeiras;
- Tv box;
- Pequenos;
- Com economia de energia;
- Celulares;

- **Cartão inteligente: os menores sistemas operacionais** são executados em cartões inteligentes – dispositivo de tamanho de cartões de crédito que contem um chip de CPU. Possuem restrições severas de **consumo de energia e de memória**. Alguns deles podem realizar apenas uma única função, como pagamento eletrônico, mas outros podem tratar múltiplas funções no mesmo cartão inteligente. São comumente sistemas **proprietários**.

- Menores sistemas;
- Muito rigorosos com a economia de energia;
- Orientados a JAVA;
- Só executam uma tarefas;

- **Alguns cartões inteligentes são orientados a Java. Isso significa que a ROM do cartão inteligente contém um interpretador para a máquina virtual Java. As pequenas aplicações Java (applets) são carregadas no cartão e interpretadas pelo JVM. Alguns desses cartões podem tratar múltiplas applets Java ao mesmo tempo, acarretando Multiprogramação e a conseqüente necessidade de escalonamento. O gerenciamento de recursos e proteção são também um problema quando duas ou mais applets estão presentes simultaneamente. Esses problemas devem ser tratados pelo sistema operacional (normalmente muito primitivo) contido no cartão.**

Processos e Threads

Estamos prestes a embarcar agora em um estudo detalhado de como os sistemas operacionais são projetados e construídos. O conceito mais central em qualquer sistema operacional é o **processo**.

Processo é um programa em execução
Um programa não é um processo
Um arquivo não é um processo

Processos : **É uma abstração de um programa em execução**. Tudo o mais depende desse conceito, e o projetista (e estudante) do sistema operacional deve ter uma compreensão profunda do que é um processo o mais cedo possível. Processos são uma das mais antigas e importantes abstrações que os sistemas operacionais proporcionam. Eles dão suporte à possibilidade de haver operações (pseudo) concorrentes mesmo quando há apenas uma CPU disponível, transformando uma única CPU em múltiplas CPUs virtuais. Sem a abstração de processo, a computação moderna não poderia existir.

Processos

Todos os computadores modernos frequentemente realizam várias tarefas ao mesmo tempo. As pessoas acostumadas a trabalhar com computadores talvez não estejam totalmente cientes desse fato, então alguns exemplos podem esclarecer este ponto. Primeiro, considere um servidor da web, em que solicitações de páginas da web chegam de toda parte. Quando uma solicitação chega, o servidor confere para ver se a página requisitada está em cache. Se estiver, ela é enviada de volta; se não, uma solicitação de acesso ao disco é iniciada para buscá-la. No entanto, do ponto de vista da CPU, as solicitações de acesso ao disco levam uma eternidade. Enquanto espera que uma solicitação de acesso ao disco seja concluída, muitas outras solicitações podem chegar. Se há múltiplos discos presentes, algumas ou todas as solicitações mais recentes podem ser enviadas para os outros discos muito antes de a primeira solicitação ter sido concluída. Está claro que algum método é necessário para modelar e controlar essa concorrência. Processos (e especialmente threads) podem ajudar nisso.

(55) 21 99461-8818
EXPLICADORES.NET
WWW.EXPLICADORES.NET.BR

Agora considere um PC de usuário. Quando o sistema é inicializado, muitos processos são secretamente iniciados, quase sempre desconhecidos para o usuário. Por exemplo, um processo pode ser inicializado para esperar pela chegada de e-mails. Outro pode ser executado em prol do programa antivírus para conferir periodicamente se há novas definições de vírus disponíveis. Além disso, processos explícitos de usuários podem ser executados, imprimindo arquivos e salvando as fotos do usuário em um pen-drive, tudo isso enquanto o usuário está navegando na Web. Toda essa atividade tem de ser gerenciada, e um sistema de multiprogramação que dê suporte a múltiplos processos é muito útil nesse caso.

Em qualquer sistema de multiprogramação, a CPU muda de um processo para outro rapidamente, executando cada um por dezenas ou centenas de milissegundos. Enquanto, estritamente falando, em qualquer dado instante a CPU está executando apenas um processo, no curso de 1s ela pode trabalhar em vários deles, dando a ilusão do paralelismo. Às vezes, as pessoas falam em pseudoparalelismo neste contexto, para diferenciar do verdadeiro paralelismo de hardware dos sistemas multiprocessadores (que têm duas ou mais CPUs compartilhando a mesma memória física). Ter controle sobre múltiplas atividades em paralelo é algo difícil para as pessoas realizarem. Portanto, projetistas de sistemas operacionais através dos anos desenvolveram um modelo conceitual (processos sequenciais) que torna o paralelismo algo mais fácil de lidar.

Pseudoparalelismo = Simular a Execução de várias tarefas ao mesmo tempo. O processador fica somente uma fatia de tempo em cada processo (time slice)
Paralelismo = Só existe quando temos mais de um processador ou mais de um núcleo.

O modelo de processo

Nesse modelo, todos os softwares executáveis no computador, às vezes incluindo o sistema operacional, são organizados em uma série de processos sequenciais, ou, simplesmente, processos. Um processo é apenas uma instância de um programa em execução, incluindo os valores atuais do contador do programa, registradores e variáveis. Conceitualmente, cada processo tem sua própria CPU virtual. Na verdade, a CPU real troca a todo momento de processo em processo, mas, para compreender o sistema, é muito mais fácil pensar a respeito de uma coleção de processos sendo executados em (pseudo) paralelo do que tentar acompanhar como a CPU troca de um programa para o outro.

Multiprogramação

Procedimento de trocas rápidas entre os processos que acessam a CPU.

Com o chaveamento rápido da CPU (time slice) entre os processos, a taxa pela qual um processo realiza a sua computação não será uniforme e provavelmente nem reproduzível se os mesmos processos forem executados outra vez. Desse modo, processos não devem ser programados com suposições predefinidas sobre a temporização.

As trocas rápidas acontecem pelo compartilhamento de tempo.
Cada processo leva um tempo específico para ser executado.

A ideia fundamental aqui é que um processo é uma atividade de algum tipo.
Ela tem:
* Um programa (Processo em execução)
* Uma entrada (Informações de pedidos)
* Uma saída (Informações de resposta)
* Um estado. (Em execução, Pronto , Bloqueado)

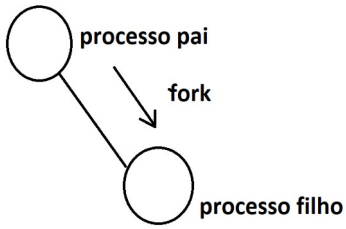
Um único processador pode ser compartilhado entre vários processos, com algum algoritmo de escalonamento sendo usado para determinar quando parar o trabalho em um processo e servir outro. Em comparação, um programa é algo que pode ser armazenado em disco sem fazer nada.

Vale a pena observar que se um programa está sendo executado duas vezes, é contado como dois processos. Por exemplo, muitas vezes é possível iniciar um processador de texto duas vezes ou imprimir dois arquivos ao mesmo tempo, se duas impressoras estiverem disponíveis.

Criação de processos

Sistemas operacionais precisam de alguma maneira para criar processos. Em sistemas muito simples, ou em sistemas projetados para executar apenas uma única aplicação (por exemplo, o controlador em um forno micro-ondas), pode ser possível ter todos os processos que serão em algum momento necessários quando o sistema for ligado. Em sistemas para fins gerais, no entanto, alguma maneira é necessária para criar e terminar processos, na medida do necessário, durante a operação. Vamos examinar agora algumas das questões. Quatro eventos principais fazem com que os processos sejam criados:
Um processo pode ser criado em 4 situações:
1. Inicialização do sistema.
2. Execução de uma chamada de sistema de criação de processo por um processo em execução.
3. Solicitação de um usuário para criar um novo processo.
4. Início de uma tarefa em lote.

Obs.: No **UNIX**, há apenas uma **chamada de sistema** para criar um novo processo: **fork.** Essa chamada cria um clone exato do processo que a chamou. Após a fork, os dois processos, o pai e o filho, têm a mesma imagem de memória, as mesmas variáveis de ambiente e os mesmos arquivos abertos.



Daemons

Processos que ficam em **segundo plano** para lidar com algumas atividades, como e-mail, páginas da web, notícias, impressão e assim por diante.

Processos que são executados porém não conseguimos visualizar.

Anti-vírus

Término dos processos

Após um processo ter sido criado, ele começa a ser executado e realiza qualquer que seja o seu trabalho. No entanto, nada dura para sempre, nem mesmo os processos. Cedo ou tarde, o novo processo terminará, normalmente devido a uma das condições a seguir:

Quatro situações que um processo pode ser terminado:

1. Saída normal (voluntária).
2. Erro fatal (involuntário).
3. Saída por erro (voluntária).
4. Morto por outro processo (involuntário).

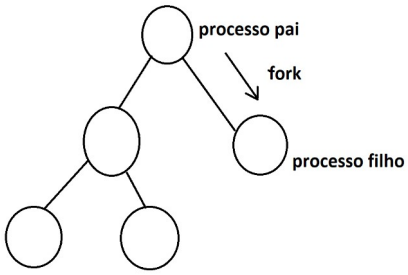
A maioria dos processos termina por terem realizado o seu trabalho. Quando um compilador termina de traduzir o programa dado a ele, o compilador executa uma chamada para dizer ao sistema operacional que ele terminou. Essa chamada é

Chamadas de sistemas são os comando do sistema operacional para os programas:

exit = Chamada de sistema de **término** de processo em **UNIX**.
ExitProcess = Chamada de sistema de **término** de processo no **Windows**.

Hierarquias de processos

Em alguns sistemas, quando um processo cria outro, o processo pai e o processo filho continuam a ser associados de certas maneiras. O processo filho pode em si criar mais processos, formando uma hierarquia de processos. Observe que, diferentemente das plantas e dos animais que usam a reprodução sexual, um processo tem apenas um pai (mas zero, um, dois ou mais filhos). Então um processo lembra mais uma hidra do que, digamos, uma vaca.



Grupo de processos

Todos os processos e seus descendentes

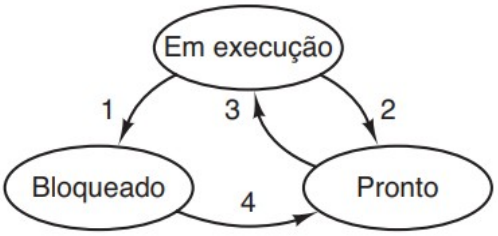
O Windows não tem conceito de uma hierarquia de processos. Todos os processos são iguais. O único indício de uma hierarquia ocorre quando um processo é criado e o pai recebe um identificador especial (chamado de handle) que ele pode usar para controlar o filho. No entanto, ele é livre para passar esse identificador para algum outro processo, desse modo invalidando a hierarquia. Processos em UNIX não podem deserdar seus filhos.

Estados de processos

Embora cada processo seja uma entidade independente, com seu próprio contador de programa e estado interno, processos muitas vezes precisam interagir entre si. Um processo pode gerar alguma saída que outro processo usa como entrada.

Os três estados nos quais um processo pode se encontrar são:

1. **Em execução** (realmente usando a CPU naquele instante).
2. **Pronto** (executável, temporariamente parado para deixar outro processo ser executado).
3. **Bloqueado** (incapaz de ser executado até que algum evento externo aconteça).



Transição entre processos:
Execução → Bloqueado
Execução → Pronto
Pronto → Execução
Bloqueado → Pronto

Execução → Bloqueado = Quando ele acabou.

Execução → Pronto = Quando ele está executando **entrada e saída;**
= Acabou o time slice;

Pronto → Execução = O time slice do processo anterior acabou.
= Quando terminou a entrada e saída.

Bloqueado → Pronto = Quando o processo tem o pedido de execução e vai para a fila de pronto.

Cpu-Bound = Quando o processo passa a maior parte do tempo em processamento (Em execução)
Mais prejudicial para o desempenho.

Io-Bound = Quando o processo passa a maior parte do tempo executando entrada e saída (Em pronto)
Menos prejudicial para o desempenho

Como apresentado na Figura, quatro transições são possíveis entre esses três estados.

A transição 1 ocorre quando o sistema operacional descobre que um processo não pode continuar agora. Em alguns sistemas o processo pode executar uma chamada de sistema, como em pause, para entrar em um estado bloqueado. Em outros, incluindo UNIX, quando um processo lê de um pipe ou de um arquivo especial (por exemplo, um terminal) e não há uma entrada disponível, o processo é automaticamente bloqueado.

As transições 2 e 3 são causadas pelo escalonador de processos, uma parte do sistema operacional, sem o processo nem saber a respeito delas. A transição 2 ocorre quando o escalonador decide que o processo em andamento foi executado por tempo suficiente, e é o momento de deixar outro processo ter algum tempo de CPU. A transição 3 ocorre quando todos os outros processos tiveram sua parcela justa e está na hora de o primeiro processo chegar à CPU para ser executado novamente. O escalonamento, isto é, decidir qual processo deve ser executado, quando e por quanto tempo, é um assunto importante; nós o examinaremos mais adiante neste capítulo. Muitos algoritmos foram desenvolvidos para tentar equilibrar as demandas concorrentes de eficiência para o sistema como um todo e justiça para os processos individuais. Estudaremos algumas delas ainda neste capítulo.

A transição 4 se verifica quando o evento externo pelo qual um processo estava esperando (como a chegada de alguma entrada) acontece. Se nenhum outro processo estiver sendo executado naquele instante, a transição 3 será desencadeada e o processo começará a ser executado. Caso contrário, ele talvez tenha de esperar no estado de pronto por um intervalo curto até que a CPU esteja disponível e chegue sua vez.

Portanto :

Um processo pode ir de:

Execução → Bloqueado
Execução → Pronto
Pronto → Execução
Bloqueado → Pronto

Implementação de processos

Para implementar o modelo de processos, o sistema operacional mantém uma tabela (um arranjo de estruturas) chamada de tabela de processos, com uma entrada para cada um deles. (Alguns autores chamam essas entradas de blocos de controle de processo.) Essas entradas contêm informações importantes sobre o estado do processo, incluindo o seu contador de programa, ponteiro de pilha, alocação de memória, estado dos arquivos abertos, informação sobre sua contabilidade e escalonamento e tudo o mais que deva ser salvo quando o processo é trocado do estado em execução para pronto ou bloqueado, de maneira que ele possa ser reiniciado mais tarde como se nunca tivesse sido parado.

Tabela de processos

Arranjo de estruturas que contém informações sobre todos os processos do computador.
- Estado do processos
Pronto, Em execução, Bloqueado
- Contador de programa
Program counter = Registrador que armazena o próximo processo a ser executado.
- Ponteiro de pilha
Registrador que armazena o topo da pilha.

PILHA
O primeiro a chegar é o último a sair
O último a chegar é o primeiro a sair
LIFO = Last in First Out
UEPS = Último a entra primeiro a sair

FILA
O primeiro a chegar é o primeira a ser atendido
FIFO = First in First out
PEPS = Primeiro a chegar, primeiro a sair.

- Alocação de memória
Informações sobre onde o programa está **armazenado em memória**.

- Estado dos arquivos abertos
Informações sobre os arquivos que o processo está utilizando

- Informações de escalonamento
Multiplexação = Escalonamento
Quantas vezes o processo passou pelo time-slice
Quantos time slice faltam para o processo terminar

Alguns campos de uma entrada típica de uma tabela de processos:

Gerenciamento de Processo	Gerenciamento de Memória	Gerenciamento de arquivo
Registros Contador de programa Palavra de estado do programa Ponteiro da pilha Estado do processo Prioridade Parâmetros de escalonamento ID do processo Processo pai Grupo de processo Sinais Momento em que um processo foi iniciado Tempo de CPU usado Tempo de CPU do processo filho Tempo do alarme seguinte	Ponteiro para informações sobre o segmento de texto. Ponteiro para informações sobre o segmento de dados. Ponteiro para informações sobre o segmento de pilha.	Diretório-raiz Diretório de trabalho Descritores de arquivo ID do usuário ID do grupo

Threads

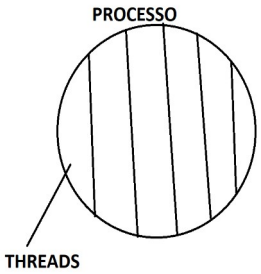
Em sistemas operacionais tradicionais, cada processo tem um espaço de endereçamento e um único thread de controle. Na realidade, essa é quase a definição de um processo. Não obstante isso, em muitas situações, é desejável ter múltiplos threads de controle no mesmo espaço de endereçamento executando em quase paralelo, como se eles fossem (quase) processos separados (exceto pelo espaço de endereçamento compartilhado).

Ou seja Threads são divisões dos processos que podem ser executadas em paralelo.

Processador
8 Núcleos (Físicos)
16 Threads (Simula 16 núcleos)

THREADS

Divisões do processo que possibilitam escalonar partes do processo, ou ainda executar ao mesmo tempo.



Sistema operacional Multithread

Sistema operacional capaz de dividir seus processos em threads.

Utilização de threads

Por que alguém iria querer ter um tipo de processo dentro de um processo? Na realidade, há várias razões para a existência desses miniprocessos, chamados threads. Vamos examinar agora algumas delas:

* A principal razão para se ter threads é que em muitas aplicações múltiplas atividades estão ocorrendo simultaneamente e algumas delas podem bloquear de tempos em tempos. Ao decompormos uma aplicação dessas em múltiplos threads sequenciais que são executados em quase paralelo, o modelo de programação **torna-se mais simples.**

* Um segundo argumento para a existência dos threads é que como eles são mais leves do que os processos, eles são mais fáceis (isto é, mais rápidos) para criar e destruir do que os processos. Em muitos sistemas, criar um thread é algo de 10 a 100 vezes mais rápido do que criar um processo. Quando o número necessário de threads muda dinâmica e rapidamente, é útil se contar com essa propriedade.

* Uma terceira razão para a existência de threads também é o argumento do desempenho. O uso de threads não resulta em um ganho de desempenho quando todos eles são limitados pela CPU, mas quando há uma computação substancial e também E/S substancial, contar com threads permite que essas atividades se sobreponham, acelerando desse modo a aplicação.

* Por fim, threads são úteis em sistemas com múltiplas CPUs, onde o paralelismo real é possível.

Máquina de estados finitos

Modelo que não é monothread e nem multithread, modelo de computação que tem um estado salvo e algum conjunto de eventos podem modificar esse estado.

Modelo	**Características**
Threads	Paralelismo, chamadas de sistema bloqueantes
Processo monothread	Não paralelismo, chamadas de sistema bloqueantes
Máquina de estados finitos	Paralelismo, chamadas não bloqueantes, interrupções

Tipos de Threads:

Threads de execução ou simplesmente Threads

O thread tem um contador de programa que controla qual instrução deve ser executada em seguida. Ele tem registradores, que armazenam suas variáveis de trabalho atuais. Tem uma pilha, que contém o histórico de execução, com uma estrutura para cada rotina chamada, mas ainda não retornada. Embora um thread deva executar em algum processo, o thread e seu processo são conceitos diferentes e podem ser tratados separadamente. Processos são usados para agrupar recursos; threads são as entidades escalonadas para execução na CPU.

Thread despachante

Lê as requisições de trabalho que chegam da rede. Depois de examinar a requisição, ele escolhe um thread operário ocioso (bloqueado) e entrega-lhe a requisição, possivelmente colocando um ponteiro para a mensagem em uma palavra especial associada a cada thread. O despachante então acorda o operário que está descansando, tirando-o do estado de bloqueado e colocando-o como pronto.

Obs.:
Multithread também é usado para descrever a situação de permitir múltiplos threads no mesmo processo.

Threads POSIX

Para possibilitar que se escrevam programas com threads portáteis, o IEEE definiu um padrão para threads no padrão IEEE 1003.1c. O pacote de threads que ele define é chamado Pthreads. A maioria dos sistemas UNIX dá suporte a ele. O padrão define mais de 60 chamadas de função.

Threads Pop-Up

Normalmente empregrada em sistemas distribuídos, a chegada de uma mensagem faz o sistema criar um novo thread para lidar com a mensagem. Esse thread é chamado de thread pop-up.

Thread – Vield

Thread desiste voluntariamente da CPU para deixar outro thread executar.
Vantagens → possibilitam que múltiplas atividades ocorram no mesmo tempo, é mais fácies de criar e destruir que o processo, tem ganhado em desempenho na execução baseada Entrada/Saída.

Jacket ou wrapper

Código que envolve a chamada de sistema para evitar eventuais bloqueios futuros, bloqueios que podem parecer para o sistema corretos, mas podem prejudicar o desempenho de execução das Threads.

Comunicação entre processos (IPC – Inter Processor Communications)

Processos quase sempre precisam comunicar-se com outros processos. Por exemplo, em um pipeline do interpretador de comandos, a saída do primeiro processo tem de ser passada para o segundo, e assim por diante até o fim da linha. Então, há uma necessidade por comunicação entre os processos, de preferência de uma maneira bem estruturada sem usar interrupções.

Condições de disputa/Corrida

Em alguns sistemas operacionais, processos que estão trabalhando juntos podem compartilhar de alguma memória comum que cada um pode ler e escrever. A memória compartilhada pode encontrar-se na memória principal (possivelmente em uma estrutura de dados de núcleo) ou ser um arquivo compartilhado; o local da memória compartilhada não muda a natureza da comunicação ou os problemas que surgem.

Dois processos, um processo lê e outro escreve em um arquivo compartilhado.

Regiões críticas

Parte do programa que tem acesso à memória compartilhada.

Exclusão Mútua

Como evitar as condições de corrida? A chave para evitar problemas aqui e em muitas outras situações envolvendo memória compartilhada, arquivos compartilhados e tudo o mais compartilhado é encontrar alguma maneira de proibir mais de um processo de ler e escrever os dados compartilhados ao mesmo tempo. Colocando a questão em outras palavras, o que precisamos é de exclusão mútua, isto é, alguma maneira de se certificar de que se um processo está usando um arquivo ou variável compartilhados, os outros serão impedidos de realizar a mesma coisa.

Algum modo de impedir acesso a um recurso compartilhado que esteja em uso. Boa solução:

- Nunca dois processos podem estar simultaneamente em suas regiões críticas.
- Nada pode ser afirmado sobre velocidade ou números de CPUs.
- Nenhum processo executando fora da região crítica pode bloquear outro.
- Nenhum processo deve esperar eternamente para entrar em sua região crítica.

Gerenciamento de Memória

A memória principal (RAM) é um recurso importante que deve ser cuidadosamente gerenciado. Apesar de o computador pessoal médio hoje em dia ter 10.000 vezes mais memória do que o IBM 7094, o maior computador no mundo no início da década de 1960, os programas estão ficando maiores mais rápido do que as memórias. Parafraseando a Lei de Parkinson, “programas tendem a expandir-se a fim de preencher a memória disponível para contê-los”. Neste capítulo, estudaremos como os sistemas operacionais criam abstrações a partir da memória e como eles as gerenciam.

O que todo programador gostaria é de uma memória privada, infinitamente grande e rápida, que fosse não volátil também, isto é, não perdesse seus conteúdos quando faltasse energia elétrica. Aproveitando o ensejo, por que não torná-la barata, também? Infelizmente, a tecnologia ainda não produz essas memórias no momento. Talvez você descubra como fazê-lo. Qual é a segunda escolha? Ao longo dos anos, as pessoas descobriram o conceito de hierarquia de memórias, em que os computadores têm alguns megabytes de memória cache volátil, cara e muito rápida, alguns gigabytes de memória principal volátil de velocidade e custo médios, e alguns terabytes de armazenamento em disco em estado sólido ou magnético não volátil, barato e lento, sem mencionar o armazenamento removível, com DVDs e dispositivos USB. É função do sistema operacional abstrair essa hierarquia em um modelo útil e então gerenciar a abstração.

Sem abstração de memória

A abstração de memória mais simples é não ter abstração alguma. Os primeiros computadores de grande porte (antes de 1960), os primeiros minicomputadores (antes de 1970) e os primeiros computadores pessoais (antes de 1980) não tinham abstração de memória. Cada programa apenas via a memória física.

Como um todo, expor a memória física a processos tem várias desvantagens importantes.

Primeiro: Se os programas do usuário podem endereçar cada byte de memória, eles podem facilmente derrubar o sistema operacional, intencionalmente ou por acidente, provocando uma parada total no sistema (a não ser que exista um hardware especial como o esquema de bloqueio e chave do IBM 360). Esse problema existe mesmo que só um programa do usuário (aplicação) esteja executando.

Segundo: Com esse modelo, é difícil ter múltiplos programas executando ao mesmo tempo (revezando-se, se houver apenas uma CPU). Em computadores pessoais, é comum haver vários programas abertos ao mesmo tempo (um processador de texto, um programa de e-mail, um navegador da web), um deles tendo o foco atual, mas os outros sendo reativados ao clique de um mouse.

Executando múltiplos programas sem uma abstração de memória

No entanto, mesmo sem uma abstração de memória, é possível executar múltiplos programas ao mesmo tempo. O que um sistema operacional precisa fazer é salvar o conteúdo inteiro da memória em um arquivo de disco, então introduzir e executar o programa seguinte. Desde que exista apenas um programa de cada vez na memória, não há conflitos. Esse conceito (swapping — troca de processos).

Espaço de endereçamento

Para solucionarmos o problema de múltiplas aplicações estejam em memória ao mesmo tempo sem interferir uma nas outras, temos que resolver dois problemas:

Proteção : Não permitir que um programa invada a área do outro na memória.
Relocação : Quando um programa necessita mudar de lugar na memória.

Para isso necessitamos criar um espaço de endereçamento.
Da mesma forma que o conceito de processo cria uma espécie de CPU abstrata para executar os programas, o espaço de endereçamento cria uma espécie de memória abstrata para abrigá-los.

Espaço de endereçamento

É o conjunto de endereços que um processo pode usar para endereçar a memória. Cada processo tem seu próprio espaço de endereçamento, independente daqueles pertencentes a outros processos (exceto em algumas circunstâncias especiais onde os processos querem compartilhar seus espaços de endereçamento).

Registradores base e registradores limite

Essa solução simples usa uma versão particularmente simples da realocação dinâmica. O que ela faz é mapear o espaço de endereçamento de cada processo em uma parte diferente da memória física de uma maneira simples. A solução clássica, que foi usada em máquinas desde o CDC 6600 (o primeiro supercomputador do mundo) ao Intel 8088 (o coração do PC IBM original), é equipar cada CPU com dois registradores de hardware especiais, normalmente chamados de registradores base e registradores limite.

Registrador base **:** Aponta para o início do programa.
Registrador limite **:** Aponta para o final do código do programa.

Troca de processos (Swapping)

Se a memória física do computador for grande o suficiente para armazenar todos os processos, os esquemas descritos até aqui bastarão de certa forma. Mas na prática, o montante total de RAM demandado por todos os processos é muitas vezes bem maior do que pode ser colocado na memória. Em sistemas típicos Windows, OS X ou Linux, algo como 50-100 processos ou mais podem ser iniciados tão logo o computador for ligado. Por exemplo, quando uma aplicação do Windows é instalada, ela muitas vezes emite comandos de tal forma que em inicializações subsequentes do sistema, um processo será iniciado somente para conferir se existem atualizações para as aplicações. Um processo desses pode facilmente ocupar 5-10 MB de memória. Outros processos de segundo plano conferem se há e-mails, conexões de rede chegando e muitas outras coisas. E tudo isso antes de o primeiro programa do usuário ter sido iniciado. Programas sérios de aplicação do usuário, como o Photoshop, podem facilmente exigir 500 MB apenas para serem inicializados e muitos gigabytes assim que começam a processar dados. Em consequência, manter todos os processos na memória o tempo inteiro exige um montante enorme de memória e é algo que não pode ser feito se ela for insuficiente.

Duas abordagens gerais para lidar com a sobrecarga de memória foram desenvolvidas ao longo dos anos.

Swapping (troca de processos), estratégia mais simples, consiste em trazer cada processo em sua totalidade, executá-lo por um tempo e então colocá-lo de volta no disco. Processos ociosos estão armazenados em disco em sua maior parte, portanto não ocupam qualquer memória quando não estão sendo executados (embora alguns “despertem” periodicamente para fazer seu trabalho, e então voltam a “dormir”).

Memória virtual, permite que os programas possam ser executados mesmo quando estão apenas parcialmente na memória principal.

TLB (Translation Lookaside Buffer)

Ele normalmente está dentro da MMU e consiste em um pequeno número de entradas, oito neste exemplo, mas raramente mais do que 256. Cada entrada contém informações sobre uma página, incluindo o número da página virtual, um bit que é configurado quando a página é modificada, o código de proteção (ler/escrever/permissões de execução) e o quadro de página física na qual a página está localizada. Esses campos têm uma correspondência de um para um com os campos na tabela de páginas, exceto pelo número da página virtual, que não é necessário na tabela de páginas. Outro bit indica se a entrada é válida (isto é, em uso) ou não.

Algoritmos de substituição de páginas

Quando ocorre uma falta de página, o sistema operacional tem de escolher uma página para remover da memória a fim de abrir espaço para a que está chegando. Se a página a ser removida foi modificada enquanto estava na memória, ela precisa ser reescrita para o disco a fim de atualizar a cópia em disco. Se, no entanto, ela não tiver sido modificada (por exemplo, ela contém uma página de código), a cópia em disco já está atualizada, portanto não é preciso reescrevê-la. A página a ser lida simplesmente sobrescreve a página que está sendo removida.

O algoritmo ótimo de substituição de página

O algoritmo de substituição de página melhor possível é fácil de descrever, mas impossível de implementar de fato. Ele funciona deste modo: no momento em que ocorre uma falta de página, há um determinado conjunto de páginas na memória. Uma dessas páginas será referenciada na próxima instrução (a página contendo essa instrução). Outras páginas talvez não sejam referenciadas até 10, 100 ou talvez 1.000 instruções mais tarde. Cada página pode ser rotulada com o número de instruções que serão executadas antes de aquela página ser referenciada pela primeira vez.

O algoritmo ótimo diz que a página com o maior rótulo deve ser removida. Se uma página não vai ser usada para 8 milhões de instruções e outra página não vai ser usada para 6 milhões de instruções, remover a primeira adia ao máximo a próxima falta de página. Computadores, como as pessoas, tentam adiar ao máximo a ocorrência de eventos desagradáveis. O único problema com esse algoritmo é que ele é irrealizável. No momento da falta de página, o sistema operacional não tem como saber quando cada uma das páginas será referenciada em seguida.

O algoritmo de substituição de páginas não usadas recentemente (NRU)

A fim de permitir que o sistema operacional colete estatísticas de uso de páginas úteis, a maioria dos computadores com memória virtual tem dois bits de status, R e M, associados com cada página. R é colocado sempre que a página é referenciada (lida ou escrita). M é colocado quando a página é escrita (isto é, modificada).

Os bits estão contidos em cada entrada de tabela de página, como mostrado na Figura 3.11. É importante perceber que esses bits precisam ser atualizados em cada referência de memória, então é essencial que eles sejam atualizados pelo hardware. Assim que um bit tenha sido modificado para 1, ele fica em 1 até o sistema operacional reinicializá-lo em 0.

O algoritmo de substituição de páginas primeiro a entrar, primeiro a sair

Outro algoritmo de paginação de baixo custo é o primeiro a entrar, primeiro a sair (first in, first out — FIFO). Para ilustrar como isso funciona, considere um supermercado que tem prateleiras suficientes para exibir exatamente k produtos diferentes. Um dia, uma empresa introduz um novo alimento de conveniência — um iogurte orgânico, seco e congelado, de reconstituição instantânea em um forno de micro-ondas. É um sucesso imediato, então nosso supermercado finito tem de se livrar do produto antigo para estocá-lo.

Uma possibilidade é descobrir qual produto o supermercado tem estocado há mais tempo (isto é, algo que ele começou a vender 120 anos atrás) e se livrar dele supondo que ninguém mais se interessa. Na realidade, o supermercado mantém uma lista encadeada de todos os produtos que ele vende atualmente na ordem em que foram introduzidos. O produto novo vai para o fim da lista; o que está em primeiro na lista é removido.

O algoritmo de substituição de páginas segunda chance

Uma modificação simples para o FIFO que evita o problema de jogar fora uma página intensamente usada é inspecionar o bit R da página mais antiga. Se ele for 0, a página é velha e pouco utilizada, portanto é substituída imediatamente. Se o bit R for 1, o bit é limpo, e a página é colocada no fim da lista de páginas, e seu tempo de carregamento é atualizado como se ela tivesse recém-chegado na memória. Então a pesquisa continua

O algoritmo de substituição de páginas do relógio

Embora segunda chance seja um algoritmo razoável, ele é desnecessariamente ineficiente, pois ele está sempre movendo páginas em torno de sua lista. Uma abordagem melhor é manter todos os quadros de páginas em uma lista circular na forma de um relógio. Um ponteiro aponta para a página mais antiga.

Algoritmo de substituição de páginas usadas menos recentemente (LRU)

Uma boa aproximação para o algoritmo ótimo é baseada na observação de que as páginas que foram usadas intensamente nas últimas instruções provavelmente o serão em seguida de novo. De maneira contrária, páginas que não foram usadas há eras provavelmente seguirão sem ser utilizadas por um longo tempo. Essa ideia sugere um algoritmo realizável: quando ocorre uma falta de página, jogue fora aquela que não tem sido usada há mais tempo. Essa estratégia é chamada de paginação LRU (Least Recently Used — usada menos recentemente).

O algoritmo de substituição de páginas do conjunto de trabalho

Na forma mais pura de paginação, os processos são inicializados sem nenhuma de suas páginas na memória. Tão logo a CPU tenta buscar a primeira instrução, ela detecta uma falta de página, fazendo que o sistema operacional traga a página contendo a primeira instrução. Outras faltas de páginas para variáveis globais e a pilha geralmente ocorrem logo em seguida. Após um tempo, o processo tem a maior parte das páginas que ele precisa para ser executado com relativamente poucas faltas de páginas. Essa estratégia é chamada de paginação por demanda, pois as páginas são carregadas apenas sob demanda, não antecipadamente.

Resumo de algoritmos de substituição de páginas

Algoritmo	Comentário
Ótimo	Não implementável, mas útil como um padrão de desempenho
NRU (não usado recentemente)	Aproximação muito rudimentar do LRU
FIFO (primeiro a entrar, primeiro a sair)	Pode descartar páginas importantes
Segunda chance	Algoritmo FIFO bastante melhorado
Relógio	Realista
LRU (usada menos recentemente)	Excelente algoritmo, porém difícil de ser implementado de maneira exata
NFU (não frequentemente usado)	Aproximação bastante rudimentar do LRU
Envelhecimento (aging)	Algoritmo eficiente que aproxima bem o LRU
Conjunto de trabalho	Implementação um tanto cara
WSClock	Algoritmo bom e eficiente

Segmentação

A memória virtual discutida até aqui é unidimensional, pois os endereços virtuais vão de 0 a algum endereço máximo, um endereço depois do outro. Para muitos problemas, ter dois ou mais espaços de endereços virtuais pode ser muito melhor do que ter apenas um. Por exemplo, um compilador tem muitas tabelas construídas em tempo de compilação, possivelmente incluindo:

1. O código-fonte sendo salvo para impressão (em sistemas de lote).
2. A tabela de símbolos, contendo os nomes e atributos das variáveis.
3. A tabela contendo todas as constantes usadas, inteiras e em ponto flutuante.
4. A árvore sintática, contendo a análise sintática do programa.
5. A pilha usada pelas chamadas de rotina dentro do compilador.

Cada uma das quatro primeiras tabelas cresce continuamente à medida que a compilação prossegue. A última cresce e diminui de maneiras imprevisíveis durante a compilação. Em uma memória unidimensional, essas cinco tabelas teriam de ser alocadas em regiões contíguas do espaço de endereçamento virtual.

Implementação da segmentação pura

A implementação da segmentação difere da paginação de uma maneira essencial: as páginas são de um tamanho fixo e os segmentos, não.

Segmentação com paginação: MULTICS

Se os segmentos forem grandes, talvez seja inconveniente, ou mesmo impossível, mantê-los na memória principal em sua totalidade. Isso leva à ideia de realizar a paginação dos segmentos, de maneira que apenas aquelas páginas de um segmento que são realmente necessárias tenham de estar na memória. Vários sistemas significativos têm dado suporte a segmentos paginados. Nesta seção, descreveremos o primeiro deles: MULTICS. Na próxima discutiremos um mais recente: o Intel x86 até o x86-64.

Consideração	Paginação	Segmentação
O programador precisa saber que essa técnica está sendo usada?	Não	Sim
Há quantos espaços de endereçamento linear?	1	Muitos
O espaço de endereçamento total pode superar o tamanho da memória física?	Sim	Sim
Rotinas e dados podem ser distinguidos e protegidos separadamente?	Não	Sim
As tabelas cujo tamanho flutua podem ser facilmente acomodadas?	Não	Sim
O compartilhamento de rotinas entre os usuários é facilitado?	Não	Sim
Por que essa técnica foi inventada?	Para obter um grande espaço de endereçamento linear sem a necessidade de comprar mais memória física	Para permitir que programas e dados sejam divididos em espaços de endereçamento logicamente independentes e para auxiliar o compartilhamento e a proteção

Sistemas de Arquivos

Todas as aplicações de computadores precisam armazenar e recuperar informações. Enquanto um processo está sendo executado, ele pode armazenar uma quantidade limitada de informações dentro do seu próprio espaço de endereçamento.

No entanto, a capacidade de armazenamento está restrita ao tamanho do espaço do endereçamento virtual. Para algumas aplicações esse tamanho é adequado, mas, para outras, como reservas de passagens aéreas, bancos ou sistemas corporativos, ele é pequeno demais.

Um segundo problema em manter informações dentro do espaço de endereçamento de um processo é que, quando o processo é concluído, as informações são perdidas.

Para muitas aplicações (por exemplo, bancos de dados), as informações precisam ser retidas por semanas, meses, ou mesmo para sempre. Perdê- -las quando o processo que as está utilizando é concluído é algo inaceitável.

Além disso, elas não devem desaparecer quando uma falha no computador mata um processo. Um terceiro problema é que frequentemente é necessário que múltiplos processos acessem (partes de) uma informação ao mesmo tempo. Se temos um diretório telefônico on-line armazenado dentro do espaço de um único processo, apenas aquele processo pode acessá-lo. A maneira para solucionar esse problema é tornar a informação em si independente de qualquer processo. Assim, temos três requisitos essenciais para o armazenamento de informações por um longo prazo:

1. Deve ser possível armazenar uma quantidade muito grande de informações.
2. As informações devem sobreviver ao término do processo que as está utilizando.
3. Múltiplos processos têm de ser capazes de acessá-las ao mesmo tempo.

Arquivos

Arquivos são unidades lógicas de informação criadas por processos. Um disco normalmente conterá milhares ou mesmo milhões deles, cada um independente dos outros. Na realidade, se pensar em cada arquivo como uma espécie de espaço de endereçamento, você não estará muito longe da verdade, exceto que eles são usados para modelar o disco em vez de modelar a RAM.

Sistemas de arquivos

Arquivos são gerenciados pelo sistema operacional. Como são estruturados, nomeados, acessados, usados, protegidos, implementados e gerenciados são tópicos importantes no projeto de um sistema operacional. Como um todo, aquela parte do sistema operacional lidando com arquivos é conhecida como sistema de arquivos e é o assunto deste capítulo.

Nomeação de arquivos

Um arquivo é um mecanismo de abstração. Ele fornece uma maneira para armazenar informações sobre o disco e lê-las depois. Isso deve ser feito de tal modo que isole o usuário dos detalhes de como e onde as informações estão armazenadas, e como os discos realmente funcionam.

É provável que a característica mais importante de qualquer mecanismo de abstração seja a maneira como os objetos que estão sendo gerenciados são nomeados; portanto, começaremos nosso exame dos sistemas de arquivos com o assunto da nomeação de arquivos. Quando um processo cria um arquivo, ele lhe dá um nome. Quando o processo é concluído, o arquivo continua a existir e pode ser acessado por outros processos usando o seu nome.

Alguns sistemas de arquivos distinguem entre letras maiúsculas e minúsculas, enquanto outros, não. O UNIX pertence à primeira categoria; o velho MS-DOS cai na segunda. (Como nota, embora antigo, o MS-DOS ainda é amplamente usado em sistemas embarcados, portanto ele não é obsoleto de maneira alguma.) Assim, um sistema UNIX pode ter todos os arquivos a seguir como três arquivos distintos: maria, Maria e MARIA. No MS- -DOS, todos esses nomes referem-se ao mesmo arquivo. Talvez seja um bom momento para fazer um comentário aqui sobre os sistemas operacionais.

O Windows 95 e o Windows 98 usavam o mesmo sistema de arquivos do MS-DOS, chamado FAT-16, e portanto herdaram muitas de suas propriedades, como a maneira de se formarem os nomes dos arquivos. O Windows 98 introduziu algumas extensões ao FAT-16, levando ao FAT32, mas esses dois são bastante parecidos. Além disso, o Windows NT, Windows 2000, Windows XP, Windows Vista, Windows 7 e Windows 8 ainda dão suporte a ambos os sistemas de arquivos FAT, que estão realmente obsoletos agora. No entanto, esses sistemas operacionais novos também têm um sistema de arquivos nativo muito mais avançado (NTFS — native file system) que tem propriedades diferentes (como nomes de arquivos em Unicode). Na realidade, há um segundo sistema de arquivos para o Windows 8, conhecido como ReFS (ou Resilient File System — sistema de arquivos resiliente), mas ele é voltado para a versão de servidor do Windows 8. Neste capítulo, quando nos referimos ao MS-DOS ou sistemas de arquivos FAT, estaremos falando do FAT-16 e FAT-32 como usados no Windows, a não ser que especificado de outra forma. Discutiremos o sistema de arquivos FAT mais tarde neste capítulo e NTFS no Capítulo 12, onde examinaremos o Windows 8 com detalhes. Incidentalmente, existe também um novo sistema de arquivos semelhante ao FAT, conhecido como sistema de arquivos exFAT, uma extensão da Microsoft para o FAT-32 que é otimizado para flash drives e sistemas de arquivos grandes. ExFAT é o único sistema de arquivos moderno da Microsoft que o OS X pode ler e escrever.

Muitos sistemas operacionais aceitam nomes de arquivos de duas partes, com as partes separadas por um ponto, como em prog.c. A parte que vem em seguida ao ponto é chamada de extensão do arquivo e costuma indicar algo seu a respeito. No MS-DOS, por exemplo, nomes de arquivos têm de 1 a 8 caracteres, mais uma extensão opcional de 1 a 3 caracteres. No UNIX, o tamanho da extensão, se houver, cabe ao usuário decidir, e um arquivo pode ter até duas ou mais extensões, como em homepage.html.zip, onde .html indica uma página da web em HTML e .zip indica que o arquivo (homepage.html) foi compactado usando o programa zip. Algumas das extensões de arquivos mais comuns e seus significados.

Extensão	Significado
.bak	Cópia de segurança
.c	Código-fonte de programa em C
.gif	Imagem no formato Graphical Interchange Format
.hlp	Arquivo de ajuda
.html	Documento em HTML
.jpg	Imagem codificada segundo padrões JPEG
.mp3	Música codificada no formato MPEG (camada 3)
.mpg	Filme codificado no padrão MPEG
.o	Arquivo objeto (gerado por compilador, ainda não ligado)
.pdf	Arquivo no formato PDF (Portable Document File)
.ps	Arquivo PostScript
.tex	Entrada para o programa de formatação TEX
.txt	Arquivo de texto
.zip	Arquivo compactado

Tipos de arquivos

Muitos sistemas operacionais aceitam vários tipos de arquivos. O UNIX (novamente, incluindo OS X) e o Windows, por exemplo, apresentam arquivos regulares e diretórios. O UNIX também tem arquivos especiais de caracteres e blocos. Arquivos regulares são aqueles que contêm informações do usuário. Todos os arquivos da Figura 4.2 são arquivos regulares. Diretórios são arquivos do sistema para manter a estrutura do sistema de arquivos. Estudaremos diretórios a seguir. Arquivos especiais de caracteres são relacionados com entrada/ saída e usados para modelar dispositivos de E/S seriais como terminais, impressoras e redes. Arquivos especiais de blocos são usados para modelar discos. Neste capítulo, estaremos interessados fundamentalmente em arquivos regulares.

Acesso aos arquivos

Acesso Sequencial

Os primeiros sistemas operacionais forneciam apenas um tipo de acesso aos arquivos: acesso sequencial. Nesses sistemas, um processo podia ler todos os bytes ou registros em um arquivo em ordem, começando do princípio, mas não podia pular nenhum ou lê-los fora de ordem. No entanto, arquivos sequenciais podiam ser trazidos de volta para o ponto de partida, então eles podiam ser lidos tantas vezes quanto necessário. Arquivos sequenciais eram convenientes quando o meio de armazenamento era uma fita magnética, em vez de um disco.

Acesso aleatório

Quando os discos passaram a ser usados para armazenar arquivos, tornou-se possível ler os bytes ou registros de um arquivo fora de ordem, ou acessar os registros pela chave em vez de pela posição. Arquivos ou registros que podem ser lidos em qualquer ordem são chamados de arquivos de acesso aleatório. Eles são necessários para muitas aplicações.

Atributos de arquivos

Todo arquivo possui um nome e sua data. Além disso, todos os sistemas operacionais associam outras informações com cada arquivo, por exemplo, a data e o horário em que foi modificado pela última vez, assim como o tamanho do arquivo. Chamaremos esses itens extras de atributos do arquivo. Algumas pessoas os chamam de metadados. A lista de atributos varia bastante de um sistema para outro. A tabela da Figura 4.4 mostra algumas das possibilidades, mas existem outras. Nenhum sistema existente tem todos esses atributos, mas cada um está presente em algum sistema.

Atributo	Significado
Proteção	Quem tem acesso ao arquivo e de que modo
Senha	Necessidade de senha para acesso ao arquivo
Criador	ID do criador do arquivo
Proprietário	Proprietário atual
Flag de somente leitura	0 para leitura/escrita; 1 para somente leitura
Flag de oculto	0 para normal; 1 para não exibir o arquivo
Flag de sistema	0 para arquivos normais; 1 para arquivos de sistema
Flag de arquivamento	0 para arquivos com backup; 1 para arquivos sem backup
Flag de ASCII/binário	0 para arquivos ASCII; 1 para arquivos binários
Flag de acesso aleatório	0 para acesso somente sequencial; 1 para acesso aleatório
Flag de temporário	0 para normal; 1 para apagar o arquivo ao sair do processo
Flag de travamento	0 para destravados; diferente de 0 para travados
Tamanho do registro	Número de bytes em um registro
Posição da chave	Posição da chave em cada registro
Tamanho da chave	Número de bytes na chave
Momento de criação	Data e hora de criação do arquivo
Momento do último acesso	Data e hora do último acesso do arquivo
Momento da última alteração	Data e hora da última modificação do arquivo
Tamanho atual	Número de bytes no arquivo
Tamanho máximo	Número máximo de bytes no arquivo

Operações com arquivos

Arquivos existem para armazenar informações e permitir que elas sejam recuperadas depois. Sistemas diferentes proporcionam operações diferentes para permitir armazenamento e recuperação. A seguir uma discussão das chamadas de sistema mais comuns relativas a arquivos.

1. Create. O arquivo é criado sem dados. A finalidade dessa chamada é anunciar que o arquivo está vindo e estabelecer alguns dos atributos.

2. Delete. Quando o arquivo não é mais necessário, ele tem de ser removido para liberar espaço para o disco. Há sempre uma chamada de sistema para essa finalidade.

3. Open. Antes de usar um arquivo, um processo precisa abri-lo. A finalidade da chamada open é permitir que o sistema busque os atributos e lista de endereços do disco para a memória principal a fim de tornar mais rápido o acesso em chamadas posteriores.
4. Close. Quando todos os acessos são concluídos, os atributos e endereços de disco não são mais necessários, então o arquivo deve ser fechado para liberar espaço da tabela interna. Muitos sistemas encorajam isso impondo um número máximo de arquivos abertos em processos. Um disco é escrito em blocos, e o fechamento de um arquivo força a escrita do último bloco dele, mesmo que não esteja inteiramente cheio ainda.
5. Read. Dados são lidos do arquivo. Em geral, os bytes vêm da posição atual. Quem fez a chamada deve especificar a quantidade de dados necessária e também fornecer um buffer para colocá-los.
6. Write. Dados são escritos para o arquivo de novo, normalmente na posição atual. Se a posição atual for o final do arquivo, seu tamanho aumentará. Se estiver no meio do arquivo, os dados existentes serão sobrescritos e perdidos para sempre.
7. Append. Essa chamada é uma forma restrita de write. Ela pode acrescentar dados somente para o final do arquivo. Sistemas que fornecem um conjunto mínimo de chamadas do sistema raramente têm append, mas muitos sistemas fornecem múltiplas maneiras de fazer a mesma coisa, e esses às vezes têm append.
8. Seek. Para arquivos de acesso aleatório, é necessário um método para especificar de onde tirar os dados. Uma abordagem comum é uma chamada de sistema, seek, que reposiciona o ponteiro de arquivo para um local específico dele. Após essa chamada ter sido completa, os dados podem ser lidos da, ou escritos para, aquela posição.
9. Get attributes. Processos muitas vezes precisam ler atributos de arquivos para realizar seu trabalho. Por exemplo, o programa make da UNIX costuma ser usado para gerenciar projetos de desenvolvimento de software consistindo de muitos arquivos-fonte. Quando make é chamado, ele examina os momentos de alteração de todos os arquivos-fonte e objetos e organiza o número mínimo de compilações necessárias para atualizar tudo. Para realizar o trabalho, o make deve examinar os atributos, a saber, os momentos de alteração.
10.Set attributes. Alguns dos atributos podem ser alterados pelo usuário e modificados após o arquivo ter sido criado. Essa chamada de sistema torna isso possível. A informação sobre o modo de proteção é um exemplo óbvio. A maioria das sinalizações também cai nessa categoria.
11.Rename. Acontece com frequência de um usuário precisar mudar o nome de um arquivo. Essa chamada de sistema torna isso possível. Ela nem sempre é estritamente necessária, porque o arquivo em geral pode ser copiado para um outro com um nome novo, e o arquivo antigo é então deletado.

Diretórios

Para controlar os arquivos, sistemas de arquivos normalmente têm diretórios ou pastas, que são em si arquivos. Nesta seção discutiremos diretórios, sua organização, suas propriedades e as operações que podem ser realizadas por eles.

Sistemas de diretório em nível único

A forma mais simples de um sistema de diretório é ter um diretório contendo todos os arquivos. Às vezes ele é chamado de diretório-raiz, mas como ele é o único, o nome não importa muito. Nos primeiros computadores pessoais, esse sistema era comum, em parte porque havia apenas um usuário. Curiosamente, o primeiro supercomputador do mundo, o CDC 6600, também tinha apenas um único diretório para todos os arquivos, embora fosse usado por muitos usuários ao mesmo tempo. Essa decisão foi tomada sem dúvida para manter simples o design do software.

Sistemas de diretórios hierárquicos

O nível único é adequado para aplicações dedicadas muito simples (e chegou a ser usado nos primeiros computadores pessoais), mas para os usuários modernos com milhares de arquivos seria impossível encontrar qualquer coisa se todos os arquivos estivessem em um único diretório. Em consequência, é necessária uma maneira para agrupar arquivos relacionados em um mesmo local. Um professor, por exemplo, pode ter uma coleção de arquivos que juntos formam um livro que ele está escrevendo, uma segunda coleção contendo programas apresentados por estudantes para outro curso, um terceiro grupo contendo o código de um sistema de escrita de compiladores avançado que ele está desenvolvendo, um quarto grupo contendo propostas de doações, assim como outros arquivos para correio eletrônico, minutas de reuniões, estudos que ele está escrevendo, jogos e assim por diante. Faz-se necessária uma hierarquia (isto é, uma árvore de diretórios). Com essa abordagem, o usuário pode ter tantos diretórios quantos forem necessários para agrupar seus arquivos de maneira natural. Além disso, se múltiplos usuários compartilham um servidor de arquivos comum, como é o caso em muitas redes de empresas, cada usuário pode ter um diretório-raiz privado para sua própria hierarquia.

Nomes de caminhos

Quando o sistema de arquivos é organizado com uma árvore de diretórios, alguma maneira é

Diretório-raiz

Um sistema de diretório em nível único contendo quatro arquivos. necessária para especificar os nomes dos arquivos. Dois métodos diferentes são os mais usados. No primeiro, cada arquivo recebe um **nome de caminho absoluto** consistindo no caminho do diretório-raiz para o arquivo. Como exemplo, o caminho /usr/ast/ caixapostal significa que o diretório-raiz contém um subdiretório usr, que por sua vez contém um subdiretório ast, que contém o arquivo caixapostal. Nomes de caminhos absolutos sempre começam no diretório-raiz e são únicos. No UNIX, os componentes do caminho são separados por /.

No Windows o separador é \. No MULTICS era >. Desse modo, o mesmo nome de caminho seria escrito como a seguir nesses três sistemas:

Windows	\usr\ast\caixapostal
UNIX	/usr/ast/caixapostal
MULTICS	>usr>ast>caixapostal

Não importa qual caractere é usado, se o primeiro caractere do nome do caminho for o separador, então o caminho será absoluto.

O outro tipo é o **nome de caminho relativo**. Esse é usado em conjunção com o conceito do **diretório de trabalho** (também chamado de **diretório atual**). Um usuário pode designar um diretório como o de trabalho atual, caso em que todos os nomes de caminho não começando no diretório-raiz são presumidos como relativos ao diretório de trabalho. Por exemplo, se o diretório de trabalho atual é /usr/ast, então o arquivo cujo caminho absoluto é /usr/ast/caixapostal pode ser referenciado somente como caixa postal. Em outras palavras, o comando UNIX

Operações com diretórios

As chamadas de sistema que podem gerenciar diretórios exibem mais variação de sistema para sistema do que as chamadas para gerenciar arquivos. Para dar uma impressão do que elas são e como funcionam, daremos uma amostra (tirada do UNIX).

1. Create. Um diretório é criado. Ele está vazio exceto por ponto e pontoponto, que são colocados ali automaticamente pelo sistema (ou em alguns poucos casos, pelo programa mkdir).

2. Delete. Um diretório é removido. Apenas um diretório vazio pode ser removido. Um diretório contendo apenas ponto e pontoponto é considerado vazio à medida que eles não podem ser removidos.

3. Opendir. Diretórios podem ser lidos. Por exemplo, para listar todos os arquivos em um diretório, um programa de listagem abre o diretório para ler os nomes de todos os arquivos que ele contém. Antes que um diretório possa ser lido, ele deve ser aberto, de maneira análoga a abrir e ler um arquivo.

4. Closedir. Quando um diretório tiver sido lido, ele será fechado para liberar espaço de tabela interno.

5. Readdir. Essa chamada retorna a próxima entrada em um diretório aberto. Antes, era possível ler diretórios usando a chamada de sistema read usual, mas essa abordagem tem a desvantagem de forçar o programador a saber e lidar com a estrutura interna de diretórios. Por outro lado, readdir sempre retorna uma entrada em um formato padrão, não importa qual das estruturas de diretório possíveis está sendo usada.

6. Rename. Em muitos aspectos, diretórios são como arquivos e podem ser renomeados da mesma maneira que eles.

7. Link. A ligação (linking) é uma técnica que permite que um arquivo apareça em mais de um diretório. Essa chamada de sistema especifica um arquivo existente e um nome de caminho, e cria uma ligação do arquivo existente para o nome especificado pelo caminho. Dessa maneira, o mesmo arquivo pode aparecer em múltiplos diretórios. Uma ligação desse tipo, que incrementa o contador no i-node do arquivo (para monitorar o número de entradas de diretório contendo o arquivo), às vezes é chamada de ligação estrita (hard link).
8. Unlink. Uma entrada de diretório é removida. Se o arquivo sendo removido estiver presente somente em um diretório (o caso normal), ele é removido do sistema de arquivos. Se ele estiver presente em múltiplos diretórios, apenas o nome do caminho especificado é removido. Os outros continuam. Em UNIX, a chamada de sistema para remover arquivos (discutida anteriormente) é, na realidade, unlink.

Alocação contígua

O esquema de alocação mais simples é armazenar cada arquivo como uma execução contígua de blocos de disco. Assim, em um disco com blocos de 1 KB, um arquivo de 50 KB seria alocado em 50 blocos consecutivos. Com blocos de 2 KB, ele seria alocado em 25 blocos consecutivos.

Alocação por lista encadeada

O segundo método para armazenar arquivos é manter cada um como uma lista encadeada de blocos de disco, como mostrado na Figura 4.11. A primeira palavra de cada bloco é usada como um ponteiro para a próxima. O resto do bloco é reservado para dados. Diferentemente da alocação contígua, todos os blocos do disco podem ser usados nesse método. Nenhum espaço é perdido para a fragmentação de disco (exceto para a fragmentação interna no último bloco). Também, para a entrada de diretório é suficiente armazenar meramente o endereço em disco do primeiro bloco. O resto pode ser encontrado a partir daí.

Alocação por lista encadeada usando uma tabela na memória

Ambas as desvantagens da alocação por lista encadeada podem ser eliminadas colocando-se as palavras do ponteiro de cada bloco de disco em uma tabela na memória. A Figura 4.12 mostra como são as tabelas para o exemplo da Figura 4.11. Em ambas, temos dois arquivos. O arquivo A usa os blocos de disco 4, 7, 2, 10 e 12, nessa ordem, e o arquivo B usa os blocos de disco 6, 3, 11 e 14, nessa ordem. Usando a tabela da Figura 4.12, podemos começar com o bloco 4 e seguir a cadeia até o fim. O mesmo pode ser feito começando com o bloco 6. Ambos os encadeamentos são concluídos com uma marca especial (por exemplo, −1) que corresponde a um número de bloco inválido. Essa tabela na memória principal é chamada de FAT (File Allocation Table — tabela de alocação de arquivos).

I-nodes

Nosso último método para monitorar quais blocos pertencem a quais arquivos é associar cada arquivo a uma estrutura de dados chamada de i-node (index-node — nó-índice), que lista os atributos e os endereços de disco dos blocos do disco.

Se cada i-node ocupa n bytes e um máximo de k arquivos puderem estar abertos simultaneamente, a memória total ocupada pelo arranjo contendo os i- -nodes para os arquivos abertos é de apenas kn bytes. Apenas essa quantidade de espaço precisa ser reservada antecipadamente.

Esse arranjo é em geral muito menor do que o espaço ocupado pela tabela de arquivos descrita na seção anterior. A razão é simples. A tabela para conter a lista encadeada de todos os blocos de disco é proporcional em tamanho ao disco em si. Se o disco tem n blocos, a tabela precisa de n entradas. À medida que os discos ficam maiores, essa tabela cresce linearmente com eles. Por outro lado, o esquema i-node exige um conjunto na memória cujo tamanho seja proporcional ao número máximo de arquivos que podem ser abertos ao mesmo tempo. Não importa que o disco tenha 100 GB, 1.000 GB ou 10.000 GB.

Cotas de disco

Para evitar que as pessoas exagerem no uso do espaço de disco, sistemas operacionais de múltiplos usuários muitas vezes fornecem um mecanismo para impor cotas de disco. A ideia é que o administrador do sistema designe a cada usuário uma cota máxima de arquivos e blocos, e o sistema operacional se certifique de que os usuários não excedam essa cota. Um mecanismo típico é descrito a seguir. Quando um usuário abre um arquivo, os atributos e endereços de disco são localizados e colocados em uma tabela de arquivos aberta na memória principal. Entre os atributos há uma entrada dizendo quem é o proprietário. Quaisquer aumentos no tamanho do arquivo serão cobrados da cota do proprietário.

Desfragmentação de disco

Quando o sistema operacional é inicialmente instalado, os programas e os arquivos que ele precisa são instalados de modo consecutivo começando no início do disco, cada um seguindo diretamente o anterior. Todo o espaço de disco livre está em uma única unidade contígua seguindo os arquivos instalados. No entanto, à medida que o tempo passa, arquivos são criados e removidos, e o disco prejudica-se com a fragmentação, com arquivos e espaços vazios por toda parte. Em consequência, quando um novo arquivo é criado, os blocos usados para isso podem estar espalhados por todo o disco, resultando em um desempenho ruim. O desempenho pode ser restaurado movendo os arquivos a fim de deixá-los contíguos e colocando todo (ou pelo menos a maior parte) o espaço livre em uma ou mais regiões contíguas no disco. O Windows tem um programa, defrag, que faz precisamente isso. Os usuários do Windows devem executá-lo com regularidade, exceto em SSDs.

Gerenciamento de entrada e saída

Princípios do hardware de E/S Diferentes pessoas veem o hardware de E/S de maneiras diferentes. Engenheiros elétricos o veem em termos de chips, cabos, motores, suprimento de energia e todos os outros componentes físicos que compreendem o hardware. Programadores olham para a interface apresentada ao software — os comandos que o hardware aceita, as funções que ele realiza e os erros que podem ser reportados de volta. Neste livro, estamos interessados na programação de dispositivos de E/S, e não em seu projeto, construção ou manutenção; portanto, 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. Nas próximas três seções, apresentaremos uma pequena visão geral sobre hardwares de E/S à medida que estes se relacionam com a programação.

Dispositivos de E/S

Dispositivos de E/S podem ser divididos de modo geral em duas categorias:

Dispositivos de blocos

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

Dispositivos 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.

Controladores 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.

Dispositivo	Taxa de dados
Teclado	10 bytes/s
Mouse	100 bytes/s
Modem 56 K	7 KB/s
Scanner em 300 dpi	1 MB/s
Filmadora *camcorder* digital	3,5 MB/s
Disco Blu-ray 4x	18 MB/s
Wireless 802.11n	37,5 MB/s
USB 2.0	60 MB/s
FireWire 800	100 MB/s
Gigabit Ethernet	125 MB/s
Drive de disco SATA 3	600 MB/s
USB 3.0	625 MB/s
Barramento SCSI Ultra 5	640 MB/s
Barramento de faixa única PCIe 3.0	985 MB/s
Barramento Thunderbolt2	2,5 GB/s
Rede SONET OC-768	5 GB/s

E/S programada

O aspecto essencial da E/S programada, claramente ilustrado nessa figura, é que, 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. A E/S programada é simples, mas tem a desvantagem de segurar a CPU o tempo todo até que toda a E/S tenha sido feita. Se o tempo para “imprimir” um caractere for muito curto (pois tudo o que a impressora está fazendo é copiar o novo caractere para um buffer interno), então a espera ocupada estará bem. No entanto, em sistemas mais complexos, em que a CPU tem outros trabalhos a fazer, a espera ocupada será ineficiente, e será necessário um método de E/S melhor.

E/S orientada a interrupções

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. O

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. Uma linha geral do código é dada na Figura 5.10. 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.

Drivers dos dispositivos

Em consequência, 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.

Sistemas com múltiplos processadores