GOVERNO DO ESTADO DO CEARÁ
Secretaria da Educação

ESCOLA ESTADUAL DE EDUCAÇÃO PROFISSIONAL - EEEP

ENSINO MÉDIO INTEGRADO À EDUCAÇÃO PROFISSIONAL

CURSO TÉCNICO EM REDES DE COMPUTADORES

SISTEMAS OPERACIONAIS

GOVERNO DO ESTADO DO CEARÁ

Secretaria da Educação

Governador
Cid Ferreira Gomes

Vice Governador
Francisco José Pinheiro

Secretária da Educação
Maria Izolda Cela de Arruda Coelho

Secretário Adjunto
Maurício Holanda Maia

Secretário Executivo
Antônio Idilvan de Lima Alencar

Assessora Institucional do Gabinete da Seduc
Cristiane Carvalho Holanda

Coordenadora de Desenvolvimento da Escola
Maria da Conceição Ávila de Misquita Vinãs

Coordenadora da Educação Profissional – SEDUC
Thereza Maria de Castro Paes Barreto

Sistemas Operacionais - Carga Horária: 40

EMENTA

Conceitos Básicos. Evolução dos Sistemas Operacionais. Estrutura e Funções dos Sistemas Operacionais. Gerenciamento de Processos. Gerência de Memória. Gerência de Dispositivos. Sistemas de Arquivos. Fundamentos de Sistemas Operacionais Distribuídos.

OBJETIVOS

• Fornecer ao aluno uma visão geral sobre os Sistemas Operacionais e seus módulos de gerenciamento;

• Apresentar as técnicas de implementação e funcionamento dos Sistemas Operacionais.

HABILIDADES E COMPETÊNCIAS

O aluno deverá entender a importância dos Sistemas Operacionais, além de compreender como se dá o gerenciamento dos recursos de hardware e software durante o funcionamento de um computador.

CONTEÚDO PROGRAMÁTICO

1. Introdução aos Sistemas Operacionais

1.1. Conceitos básicos

1.2. Funções de um sistema operacional

1.3. Evolução dos sistemas operacionais

1.4. Componentes de um sistema operacional

2. Processos

2.1. Definição e estrutura de processos

2.2. Estados de um processo

2.3. Tipos de processo

2.4. Comunicação entre processos

2.5. Escalonamento

3. Gerência de Memória

3.1. Endereços lógicos e físicos

3.2. Alocação de Memória

3.3. Compartilhamento de Memória

3.4. Paginação

3.5. Memória Virtual

4. Sistemas de Arquivos

4.1. Arquivos e diretórios

4.2. Alocação de arquivos

4.3. Segurança e mecanismos de proteção da informação

5. Gerência de Dispositivos

5.1. Dispositivos de entrada e saída

5.2. Device drivers

5.3. Controladores

6. Sistemas Operacionais Distribuídos

6.1. Comunicação síncrona e assíncrona

6.2. Modelo Cliente-Servidor e Peer-to-peer

6.3. Sockets

6.4. Chamada remota a procedimento

BIBLIOGRAFIA BÁSICA

• TANEMBAUM, Andrew S. Sistemas Operacionais Modernos. Prentice-Hall

• CÔRTES, Pedro Luis. Sistemas Operacionais – Fundamentos. Ed. Érica

• SILBERSCHATZ. Sistemas Operacionais – Conceitos e Aplicações, Ed. Campus

• MACHADO, Francis. Arquitetura de Sistemas Operacionais, Ed. LTC

1. Introdução aos Sistemas Operacionais

Programas computacionais (ou software) constituem o elo entre o aparato eletrônico (ou hardware) e o ser humano. Tal elo se faz necessário dada a discrepância entre o tipo de informação manipulada pelo homem e pela máquina. A máquina opera com cadeias de códigos binários enquanto o homem opera com estruturas mais abstratas como conjuntos, arquivos, algoritmos, etc.

Programas computacionais podem ser grosseiramente divididos em dois tipos:

- Programas do sistema, que manipulam a operação do computador;
- Programas aplicativos, que resolvem problemas para o usuário.

O mais importante dos programas do sistema é o sistema operacional, que controla todos os recursos do computador e proporciona a base de sustentação para a execução de programas aplicativos.

1.1. O que é um Sistema Operacional?

A maioria de usuários de computador tem alguma experiência com sistemas operacionais, mas é difícil definir precisamente o que é um sistema operacional. Parte do problema decorre do fato do sistema operacional realizar duas funções básicas que, dependendo do ponto de vista abordado, uma se destaca sobre a outra. Estas funções são descritas a seguir.

O Sistema Operacional como uma Máquina Virtual

A arquitetura (conjunto de instruções, organização de memória, E/S e estrutura de barramento) da maioria dos computadores a nível de linguagem de máquina é primitiva e difícil de programar, especificamente para operações de entrada e saída. É preferível para um programador trabalhar com abstrações de mais alto nível onde detalhes de implementação das abstrações não são visíveis. No caso de discos, por exemplo, uma abstração típica é que estes armazenam uma coleção de arquivos identificados por nomes simbólicos.

O programa que esconde os detalhes de implementação das abstrações é o sistema operacional. A abstração apresentada ao usuário pelo sistema operacional é simples e mais fácil de usar que o hardware original.

Nesta visão, a função do sistema operacional é apresentada ao usuário como uma máquina estendida ou máquina virtual que é mais fácil de programar que o hardware que a suporta.

O Sistema Operacional como um Gerenciador de Recursos

Um computador moderno é composto de vários subsistemas tais como processadores, memórias, discos, terminais, fitas magnéticas, interfaces de rede, impressoras, e outros dispositivos de E/S. Neste ponto de vista, o sistema operacional tem a função de gerenciar de forma adequada estes recursos de sorte que as tarefas impostas pelos usuários sejam atendidas da forma mais rápida e confiável possível. Um exemplo típico é o compartilhamento da unidade central de processamento (CPU) entre as várias tarefas (programas) em sistemas multiprogramados. O sistema operacional é o responsável pela distribuição de forma otimizada da CPU entre as tarefas em execução.

1.2. História dos Sistemas Operacionais

Os sistemas operacionais têm evoluído com o passar dos anos. Nas próximas seções vamos apresentar de forma sucinta este desenvolvimento.

A Primeira Geração (1945-1955): Válvulas e Plugs

Após muitos esforços mal sucedidos de se construir computadores digitais antes da 2ª guerra mundial, em torno da metade da década de 1940 alguns sucessos foram obtidos na construção de máquinas de cálculo empregando-se válvulas e relés. Estas máquinas eram enormes, ocupando salas com racks que abrigavam dezenas de milhares de válvulas (e consumiam quantidades imensas de energia).

Naquela época, um pequeno grupo de pessoas projetava, construía, programavam operava e dava manutenção em cada máquina. Toda programação era feita absolutamente em linguagem de máquina, muitas vezes interligando plugs para controlar funções básicas da máquina. Linguagens de programação eram desconhecidas; sistemas operacionais idem. Por volta de 1950 foram introduzidos os cartões perfurados aumentando a facilidade de programação.

A Segunda Geração (1955-1965): Transistores e Processamento em Batch

A introdução do transistor mudou radicalmente o quadro. Computadores tornaram-se confiáveis e difundidos (com a fabricação em série), sendo empregados em atividades múltiplas. Pela primeira vez, houve uma separação clara entre projetistas, construtores, operadores, programadores e pessoal de manutenção. Entretanto, dado seu custo ainda elevado, somente corporações e universidades de porte detinham recursos e infraestrutura para empregar os computadores desta geração.

Estas máquinas eram acondicionadas em salas especiais com pessoal especializado para sua operação. Para rodar um job (programa), o programador produzia um conjunto de cartões perfurados (um cartão por comando do programa), e o entregava ao operador que dava entrada do programa no computador. Quando o computador completava o trabalho, o operador devolvia os cartões com a impressão dos resultados ao programador.

A maioria dos computadores de 2ª geração foram utilizados para cálculos científicos e de engenharia. Estes sistemas eram largamente programados em FORTRAN e ASSEMBLY. Sistemas operacionais típicos eram o FMS (Fortran Monitor Systems) e o IBSYS (IBM's Operating Systems).

A Terceira Geração (1965-1980): Circuitos Integrados e Multiprogramação

No início dos anos 60, a maioria dos fabricantes de computadores tinha duas linhas distintas e incompatíveis de produtos. De um lado, havia os computadores científicos que eram usados para cálculos numéricos na ciência e engenharia. Do outro, haviam os computadores comerciais que executavam tarefas como ordenação de dados e impressão de relatórios, sendo utilizados principalmente por instituições financeiras.

A IBM tentou resolver este problema introduzindo a série System/360. Esta série consistia de máquinas com mesma arquitetura e conjunto de instruções. Desta maneira, programas escritos para uma máquina da série executavam em todas as demais. A série 360 foi projetada para atender tanto aplicações científicas quanto comerciais.

Não foi possível para a IBM escrever um sistema operacional que atendesse a todos os conflitos de requisitos dos usuários. O resultado foi um sistema operacional (OS/360) enorme e complexo comparado com o FMS.

A despeito do tamanho e problemas, o OS/360 atendia relativamente bem às necessidades dos usuários. Ele também popularizou muitas técnicas ausentes nos sistemas operacionais de 2ª geração, como por exemplo a multiprogramação. Outra característica apresentada foi a capacidade de ler jobs dos cartões perfurados para os discos, assim que o programador os entregasse. Dessa maneira, assim que um job terminasse, o computador iniciava a execução do seguinte, que já fôra lido e armazenado em disco. Esta técnica foi chamada spool (simultaneous peripherical operation on line), sendo também utilizada para a saída de dados.

O tempo de espera dos resultados dos programas reduziu-se drasticamente com a 3ª geração de sistemas. O desejo por respostas rápidas abriu caminho para o time-sharing, uma variação da multiprogramação onde cada usuário tem um terminal on-line e todos compartilham uma única CPU.

Após o sucesso do primeiro sistema operacional com capacidade de time-sharing (o CTSS) desenvolvido no MIT, um consórcio envolvendo o MIT, a GE e o Laboratório Bell foi formado com o intuito de desenvolver um projeto ambicioso para a época: um sistema operacional que suportasse centenas de usuários on-line. O MULTICS (MULTiplexed Information and Computing Service) introduziu muitas idéias inovadoras, mas sua implementação mostrou-se impraticável para a década de sessenta. O projeto MULTICS influenciou os pesquisadores da Bell que viriam a desenvolver o UNIX uma década depois.

A Quarta Geração (1980-): Computadores Pessoais e Estações de Trabalho

Com o desenvolvimento de circuitos LSI, chips contendo milhares de transistores em um centímetro quadrado de silício, surgiu a era dos computadores pessoais e estações de trabalho. Em termos de arquitetura, estes não diferem dos minicomputadores da classe do PDP-11, exceto no quesito mais importante: preço. Enquanto os minicomputadores atendiam companhias e universidades, os computadores pessoais e estações de trabalho passaram a atender usuários individualmente.

O aumento do potencial destas máquinas criou um vastíssimo mercado de software a elas dirigido. Como requisito básico, estes produtos (tanto aplicativos quanto o próprio sistema operacional) necessitavam ser “amigáveis”, visando usuários sem conhecimento aprofundado de computadores e sem intenção de estudar muito para utilizá-los. Esta foi certamente a maior mudança em relação ao OS/360 que era tão obscuro que diversos livros foram escritos sobre ele. Dois sistemas operacionais dominaram o mercado: MS-DOS para os computadores pessoais e UNIX para as estações de trabalho.

O próximo desenvolvimento no campo dos sistemas operacionais surgiu com a tecnologia de redes de computadores: os sistemas operacionais de rede e distribuídos. Sistemas operacionais de rede diferem dos sistemas operacionais para um simples processador no tocante à capacidade de manipular recursos distribuídos pelos processadores da rede. Por exemplo, um arquivo pode ser acessado por um usuário num processador, mesmo que fisicamente o arquivo se encontre em outro processador. Sistemas operacionais de rede proveem ao usuário uma interface transparente de acesso a recursos compartilhados (aplicativos, arquivos, impressoras, etc), sejam estes recursos locais ou remotos.

Sistemas operacionais distribuídos são muito mais complexos. Estes sistemas permitem que os processadores cooperem em serviços intrínsecos de sistemas operacionais tais como escalonamento de tarefas e paginação. Por exemplo, num sistema operacional distribuído uma tarefa pode “migrar” durante sua execução de um computador

sobrecarregado para outro que apresente carga mais leve. Contrário aos sistemas operacionais de rede que são largamente disponíveis comercialmente, sistemas operacionais distribuídos têm sua utilização ainda restrita.

1.3 Conceitos Básicos em Sistemas Operacionais

Processos

Um conceito fundamental em sistemas operacionais é o de processo ou tarefa. Um processo (às vezes chamado de processo sequencial) é basicamente um programa em execução. Ele é uma entidade ativa que compete por recursos (principalmente CPU) e interage com outros processos.

Em um instante qualquer, um processo está em um determinado estado. Estes estados podem ser:

- Executando (usando a CPU para executar as instruções do programa);
- Bloqueado (aguardando recursos 2 não disponíveis no momento);
- Ativo (aguardando apenas CPU para executar).

Um processo em execução passa por uma sequência de estados ordenados no tempo. Um processo possui duas importantes propriedades:

- O resultado da execução de um processo independe da velocidade com que é executado;
- Se um processo for executado novamente com os mesmos dados, ele passará precisamente pela mesma sequência de instruções e fornecerá o mesmo resultado.

Estas propriedades enfatizam a natureza sequencial de um processo. O processo sequencial é definido pelo resultado de suas instruções, não pela velocidade com que as instruções são executadas.

Sistemas Multitarefas e Multiusuários

Como já mencionado, um programa em execução é chamado de processo ou tarefa. Um sistema operacional multitarefa se distingue pela sua habilidade de suportar a execução concorrente de processos sobre um processador único, sem necessariamente prover elaborada forma de gerenciamento de recursos (CPU, memória, etc).

Sistemas operacionais multiusuários permitem acessos simultâneos ao computador através de dois ou mais terminais de entrada. Embora frequentemente associada com multiprogramação, multitarefa não implica necessariamente em uma operação multiusuário. Operação multiprocessos sem suporte de multiusuários pode ser encontrado em sistemas operacionais de alguns computadores pessoais avançados e em sistemas de tempo-real.

Multiprogramação

Multiprogramação é um conceito mais geral que multitarefa e denota um sistema operacional que provê gerenciamento da totalidade de recursos tais como CPU, memória, sistema de arquivos, em adição ao suporte da execução concorrente dos processos.

Em uma máquina podemos ter o conjunto de processos sendo executados de forma serial ou de forma concorrente, ou seja, os recursos presentes na máquina podem ser alocados a um único programa até a conclusão de sua execução ou esses recursos podem ser alocados de modo dinâmico entre um número de programas ativos de acordo com o nível de prioridade ou o estágio de execução de cada um dos programas.

No caso de um computador no qual o sistema operacional usado permite apenas a monoprogramação, os programas serão executados instrução-a-instrução, até que seu processamento seja concluído.

Durante a sua execução, o programa passará por diversas fases, alterando momentos em que se encontra executando ou bloqueado aguardando, por exemplo, a conclusão de uma operação de entrada/saída de dados (normalmente lenta, se comparada à velocidade de execução das instruções por parte do processador).

Através do uso da multiprogramação é possível reduzir os períodos de inatividade da CPU e consequentemente aumentar a eficiência do uso do sistema como um todo. O termo multiprogramação denota um sistema operacional o qual em adição ao suporte de múltiplos processos concorrentes, permite que instruções e dados de dois ou mais processos disjuntos estejam residentes na memória principal simultaneamente.

O nível de multiprogramação presente em um sistema pode ser classificado como integral ou serial. A multiprogramação é dita integral caso mais de um processo possa se encontrar em execução em um dado instante, enquanto que no caso da serial apenas um processo se encontra em execução a cada instante, sendo a CPU alocada aos processos de forma intercalada ao longo do tempo. Uma vez que a maioria dos computadores apresenta apenas uma única CPU, a multiprogramação serial é encontrada com mais frequência.

Multiprocessamento

Embora a maioria dos computadores disponha de uma única CPU que executa instruções uma a uma, certos projetos mais avançados incrementaram a velocidade efetiva de computação permitindo que várias instruções fossem executadas ao mesmo tempo. Um computador com múltiplos processadores que compartilhem uma memória principal comum é chamado um multiprocessador. O sistema que suporta tal configuração é um sistema que suporta o multiprocessamento.

Interpretador de Comandos (Shell)

O interpretador de comando é um processo que perfaz a interface do usuário com o sistema operacional. Este processo lê o teclado a espera de comandos, interpreta-os e passa seus parâmetros ao sistema operacional. Serviços como login/logout, manipulação de arquivos, execução de programas, etc, são solicitados através do interpretador de comandos.

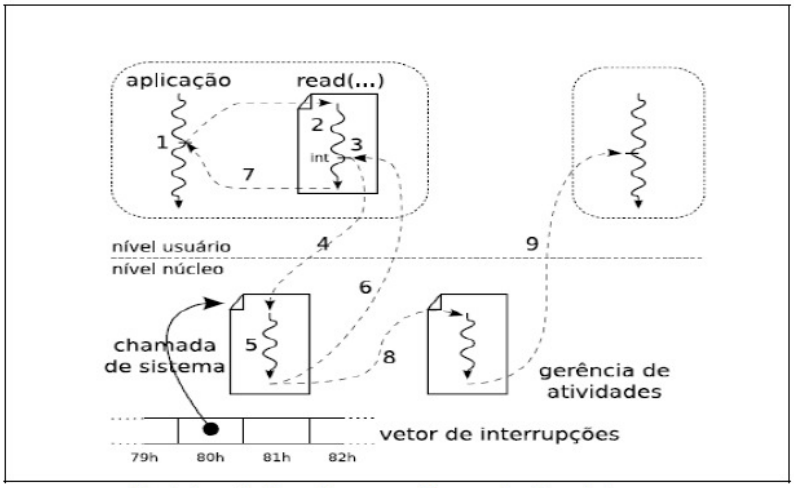
Chamadas de Sistema (System Calls)

Assim como o interpretador de comandos é a interface usuário/sistema operacional, as chamadas do sistema constituem a interface programas aplicativos/sistema operacional. As chamadas do sistema são funções que podem ser ligadas com os aplicativos provendo serviços como: leitura do relógio interno, operações de entrada/saída, comunicação inter-processos, etc.

1.4 Arquiteturas de Sistemas Operacionais

Embora a definição de níveis de privilegio imponha uma estruturação mínima a um sistema operacional, as múltiplas partes que compõem o sistema podem ser organizadas de diversas formas, separando suas funcionalidades e modularizando seu projeto. Nesta seção

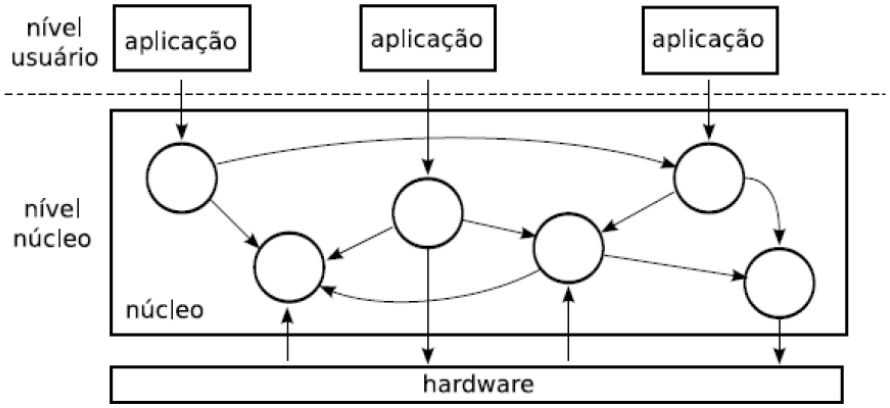
serão apresentadas as arquiteturas mais populares para a organização de sistemas operacionais.



Roteiro típico de uma chamada de sistema

Sistemas monolíticos

Em um sistema monolítico, todos os componentes do núcleo operam em modo núcleo e se inter-relacionam conforme suas necessidades, sem restrições de acesso entre si, pois o código no nível núcleo tem acesso pleno a todos os recursos e áreas de memoria.



Uma arquitetura monolítica

A grande vantagem dessa arquitetura é seu desempenho: qualquer componente do núcleo pode acessar os demais componentes, toda a memória ou mesmo dispositivos periféricos diretamente, pois não há barreiras impedindo esse acesso. A interação direta entre componentes também leva a sistemas mais compactos.

Todavia, a arquitetura monolítica pode pagar um preço elevado por seu desempenho: a robustez e a facilidade de desenvolvimento. Caso um componente do núcleo perca o controle devido a algum erro, esse problema pode se alastrar rapidamente por todo o núcleo, levando o sistema ao colapso (travamento, reinicialização ou funcionamento errado). Além disso, a manutenção e evolução do núcleo se tornam mais complexas, porque as dependências e pontos de interação entre os componentes podem não ser evidentes: pequenas alterações na estrutura de dados de um componente podem

ter um impacto inesperado em outros componentes, caso estes acessem aquela estrutura diretamente.

A arquitetura monolítica foi a primeira forma de organizar os sistemas operacionais; sistemas UNIX antigos e o MS-DOS seguiam esse modelo. Atualmente, apenas sistemas operacionais embutidos usam essa arquitetura, devido às limitações do hardware sobre o qual executam.

O núcleo do Linux nasceu monolítico, mas vem sendo paulatinamente estruturado e modularizado desde a versão 2.0 (embora boa parte de seu código ainda permaneça no nível de núcleo).

Sistemas em camadas

Uma forma mais elegante de estruturar um sistema operacional faz uso da noção de camadas: a camada mais baixa realiza a interface como hardware, enquanto as camadas intermediárias proveem níveis de abstração e gerência cada vez mais sofisticados. Por fim, a camada superior define a interface do núcleo para as aplicações (as chamadas de sistema).

Essa abordagem de estruturação de software fez muito sucesso no domínio das redes de computadores, através do modelo de referência OSI (Open Systems Interconnection), e também seria de se esperar sua adoção no domínio dos sistemas operacionais. No entanto, alguns inconvenientes limitam sua aceitação nesse contexto:

- O empilhamento de várias camadas de software faz com que cada pedido de uma aplicação demore mais tempo para chegar até o dispositivo periférico ou recurso a ser acessado, prejudicando o desempenho do sistema.
- Não é óbvio como dividir as funcionalidades de um núcleo de sistema operacional em camadas horizontais de abstração crescente, pois essas funcionalidades são interdependentes, embora tratem muitas vezes de recursos distintos.

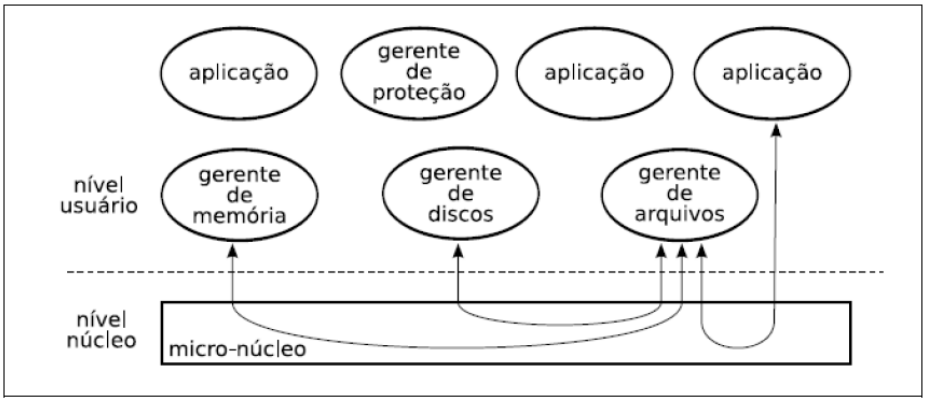
Em decorrência desses inconvenientes, a estruturação em camadas é apenas parcialmente adotada hoje em dia. Muitos sistemas implementam uma camada inferior de abstração do hardware para interagir com os dispositivos (a camada HAL – Hardware Abstraction Layer, implementada no Windows NT e seus sucessores), e também organizam em camadas alguns sub-sistemas como a gerência de arquivos e o suporte de rede (seguindo o modelo OSI). Como exemplos de sistemas fortemente estruturados em camadas podem ser citados o IBM OS/2 e o MULTICS.

Sistemas micro-núcleo

Uma outra possibilidade de estruturação consiste em retirar do núcleo todo o código de alto nível (normalmente associado às políticas de gerência de recursos), deixando no núcleo somente o código de baixo nível necessário para interagir como hardware e criar as abstrações fundamentais (como a noção de atividade). Por exemplo, usando essa abordagem o código de acesso aos blocos de um disco rígido seria mantido no núcleo, enquanto as abstrações de arquivo e diretório seriam criadas e mantidas por um código fora do núcleo, executando da mesma forma que uma aplicação do usuário.

Por fazer os núcleos de sistema ficarem menores, essa abordagem foi denominada micro-núcleo (ou µ-kernel). Um micro-núcleo normalmente implementa somente a noção de atividade, de espaços de memória protegidos e de comunicação entre atividades.

Todos os aspectos de alto nível, como políticas de uso do processador e da memória, o sistema de arquivos e o controle de acesso aos recursos são implementados fora do núcleo, em processos que se comunicam usando as primitivas do núcleo.



Visão geral de uma arquitetura micro-núcleo

Em um sistema micro-núcleo, as interações entre componentes e aplicações são feitas através de trocas de mensagens. Assim, se uma aplicação deseja abrir um arquivo no disco rígido, envia uma mensagem para o gerente de arquivos que, por sua vez, se comunica com o gerente de dispositivos para obter os blocos de dados relativos ao arquivo desejado. Os processos não podem se comunicar diretamente, devido às restrições impostas pelos mecanismos de proteção do hardware. Por isso, todas as mensagens são transmitidas através de serviços do micro-núcleo. Como os processos têm de solicitar "serviços" uns dos outros, para poder realizar suas tarefas, essa abordagem também foi denominada cliente-servidor.

O micro-núcleos foram muito investigados durante os anos 80. Dois exemplos clássicos dessa abordagem são os sistemas Mach e Chorus. As principais vantagens dos sistemas micro-núcleo são sua robustez e flexibilidade: caso um sub-sistema tenha problemas, os mecanismos de proteção de memória e níveis de privilégio irão confiná-lo, impedindo que a instabilidade se alastre ao restante do sistema. Além disso, é possível customizar o sistema operacional, iniciando somente os componentes necessários ou escolhendo os componentes mais adequados às aplicações que serão executadas.

Vários sistemas operacionais atuais adotam parcialmente essa estruturação; por exemplo, o Maços X da Apple tem suas raízes no sistema Mach, ocorrendo o mesmo com o Digital UNIX. Todavia, o custo associado às trocas de mensagens entre componentes pode ser bastante elevado, o que prejudica seu desempenho e diminui a aceitação desta abordagem. O QNX é um dos poucos exemplos de micro-núcleo amplamente utilizado, sobretudo em sistemas embutidos e de tempo-real.

Máquinas virtuais

Em um computador real uma camada de software de baixo nível (por exemplo, a BIOS dos sistemas PC) fornece acesso aos vários recursos do hardware para o sistema operacional que os disponibiliza de forma abstrata às aplicações. Quando o sistema operacional acessa os dispositivos de hardware ele faz uso dos drivers respectivos que interagem diretamente com a memória e os dispositivos da máquina.

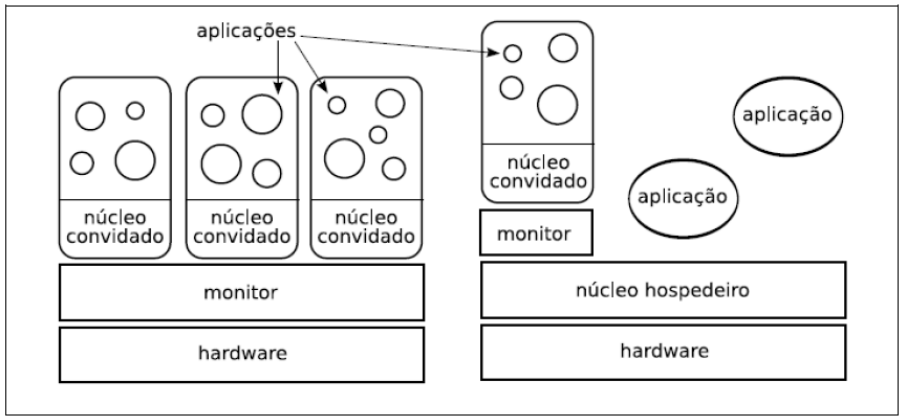
Por outro lado um emulador implementa todas as instruções realizadas pela máquina real em um ambiente abstrato, possibilitando executar um aplicativo de uma plataforma em outra, por exemplo, um aplicativo do Windows executando no Linux. Infelizmente, um emulador perde muito em eficiência ao traduzir as instruções de uma máquina real.

Além disso, emuladores são bastante complexos, pois, geralmente necessitam simular a quase totalidade das instruções do processador e demais características do hardware que os circundam.

Uma máquina virtual é definida como “uma duplicata eficiente e isolada de uma máquina real”. A funcionalidade e o nível de abstração de uma máquina virtual se encontra entre uma máquina real e um emulador, na medida em que abstrai somente os recursos de hardware e de controle usados pelas aplicações. Uma máquina virtual é um ambiente criado por um monitor de máquinas virtuais, também denominado “sistema operacional para sistemas operacionais”. O monitor pode criar uma ou mais máquinas virtuais sobre uma única máquina real. Enquanto um emulador fornece uma camada de abstração completa entre o sistema em execução e o hardware, um monitor abstrai o hardware subjacente e controla uma ou mais máquinas virtuais. Cada máquina virtual fornece facilidades para uma aplicação ou um “sistema convidado” que acredita estar executando sobre um ambiente normal com acesso físico ao hardware.

Existem basicamente duas abordagens para a construção de sistemas de máquinas virtuais: o tipo I, onde o monitor de máquinas virtuais é implementado entre o hardware e os sistemas convidados, e o tipo II, onde o monitor é implementado como um processo de um sistema operacional real subjacente, denominado sistema anfitrião ou sistema hospedeiro.

Ambas as abordagens estão ilustradas na figura abaixo, Como exemplos de sistemas de tipo I podem ser citados o VM Ware ESX Server e o Xen; para o tipo II podem ser citados o VM Ware Workstation, o MS Virtual PC e o User-Mode Linux.



Máquinas virtuais de tipo I (esquerda) e de tipo II (direita)

2. Processos

2.1. Definição e estrutura de processos

Programa é o código fonte escrito em alguma linguagem de programação (por exemplo, C, Pascal, Fortran), já executável é esse código traduzido para a linguagem específica (binário) da máquina real onde ele será executado.

Um processo pode ser definido como um fluxo de controle seqüencial e seu espaço de endereçamento, ou seja, é a execução de um programa junto com os dados usados por ele. Processo, de forma resumida, é então um programa em execução. Como exemplo, o mesmo programa pode ser executado ao mesmo tempo por dois usuários, gerando dois processos distintos. Além disso, na sua execução um programa pode originar vários processos.

O contexto de um processo é o conjunto de dados necessários a sua execução no tempo como, por exemplo, a identificação do processo (PID – Process Identification), contador de programa (PC Program Counter), ponteiros (SP Stack Pointer), as variáveis e dados armazenados em memória, o conteúdo dos registradores da CPU, a lista de arquivos que estão sendo utilizados, tempo de CPU disponível e prioridade de execução, entre outros. Essas informações são fundamentais para que o processo interrompido pelo escalonador possa voltar a executar exatamente a partir do ponto de parada sem perda de dados ou inconsistências. Tais informações são armazenadas em estruturas de dados conhecidas como Tabela de Processos.

O processo, quando em execução na CPU, está no estado executando (run). Ao terminar sua quota de tempo para execução, o escalonador o interrompe e colocao no estado pronto. Ao chegar novamente sua vez de executar, o escalonador invoca (acorda) o processo permitindo sua continuidade.

Um processo pode ser bloqueado se os dados de entrada ainda não estiverem disponíveis e ele não poder continuar sua execução, e isto pode ocorrer pelo sinal enviado por outro processo ou pela própria decisão do escalonador.

O processo é desbloqueado por um evento externo (chegada dos dados, sinalização de outro processo), e então passa ao estado pronto. Para efetuar o compartilhamento da CPU entre processos, o sistema operacional possui duas filas de controle: a de processos prontos (ReadyList) e a de processos bloqueados (BlockedList).

A manipulação dessas filas depende da política de escalonamento adotada pelo sistema, que é um critério para determinar, entre os diversos processos no estado pronto, qual o próximo processo a executar.

Concorrência

Num sistema multitarefa, normalmente vários programas são executados simultaneamente e o número de processos é maior que o de CPUs. Um dos problemas mais difíceis na administração de recursos está relacionado ao fato de muitos processos existirem simultaneamente, ocorrendo em alguns casos uma disputa entre processos pelo uso do mesmo recurso.

No caso de haver disputa dizemos então que esses processos são concorrentes. Por exemplo, processos concorrentes frequentemente acessam o mesmo arquivo, e o sistema operacional deve garantir que um processo não possa alterar os dados que outro processo esteja usando.

Outro exemplo de processos concorrentes ocorre quando um processo gera dados que um outro processo usará, nesse caso o sistema operacional deve garantir que o segundo processo não tente usar os dados antes que o primeiro os tenha gerado. Isso é importante no processamento de entrada e saída, onde vários processos diferentes participam na transferência de dados.

No caso de haver concorrência entre processos, o sistema operacional deve regular a ordem segundo a qual os processos acessam os dados, que é o conceito de sincronização.

Thread ou processo leve

Um processo tem duas partes: a ativa que é o fluxo de controle, e a passiva que é o espaço de endereçamento. Uma thread é um caso de processo leve, pois possui somente o fluxo de controle. A sua vantagem reside no espaço de endereçamento ser compartilhado e por isso consumir menos recursos do sistema.

Muitas vezes os processos usam dados compartilhados, e o uso de várias threads (multthread) no mesmo espaço de endereçamento é mais eficiente e econômico que múltiplos processos. Porém nem tudo são vantagens. Uma desvantagem é justamente o fato do compartilhamento do espaço de endereçamento, que cria um problema de segurança. Outro problema está na robustez dessa construção. Por exemplo, num caso de múltiplos processos, se um deles tiver de ser finalizado os demais não serão afetados, já num multthread a paralisação de uma thread poderá afetar o processo, ou seja, afeta todas as demais threads.

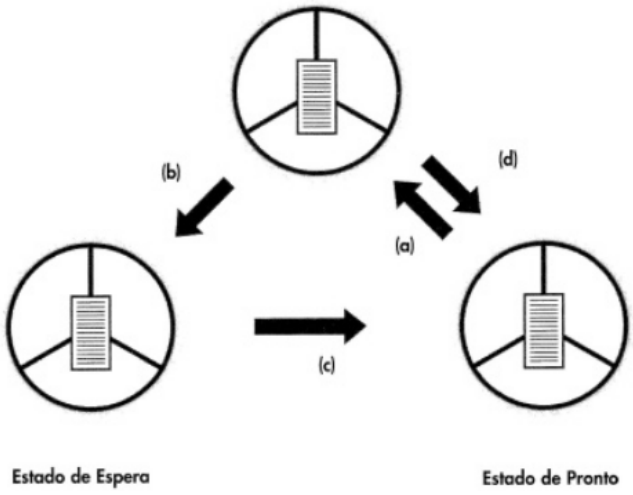
2.2. Estados de um processo

Em um sistema multiprogramável, um processo não deve alocar a CPU com exclusividade, de forma que possa existir um compartilhamento no uso do processador. Os processos passam por diferentes estados ao longo do seu processamento, em função de eventos gerados pelo sistema operacional ou pelo próprio processo. Um processo pode encontrar-se em três estados diferentes:

- Execução (running) Um processo é dito no estado de execução quando está sendo processado pela CPU. Os processos se alternam na utilização do processador seguindo uma política estabelecida pelo sistema operacional.
- Pronto (ready) Um processo está no estado de pronto quando ele tem condições lógicas de executar e apenas aguarda para ser executado. O sistema operacional é responsável por determinar a ordem e os critérios pelos quais os processos em estado de pronto devem fazer uso do processador. Esse mecanismo é conhecido como escalonamento. Em geral, existem vários processos no sistema no estado de pronto organizados em listas encadeadas.
- Espera (wait) Um processo no estado de espera aguarda por algum evento externo ou por algum recurso para prosseguir seu processamento, ou seja, ele não tem condições lógicas de executar. O sistema organiza os vários processos no estado de espera também em listas encadeadas, associadas a cada tipo de evento.

Um processo muda de estado durante seu processamento em função de eventos originados por ele próprio (eventos voluntários) ou pelo sistema operacional (eventos

involuntários). Basicamente, existem quatro mudanças de estado que podem ocorrer a um processo:



Pronto⇒ Execução

Após a criação de um processo, o sistema o coloca em uma lista de processos no estado de pronto, onde aguarda uma oportunidade para ser executado. Cada sistema operacional tem seus próprios critérios e algoritmos para a escolha da ordem em que os processos serão executados.

Execução⇒ Espera

Um processo em execução passa para o estado de espera por eventos gerados pelo próprio processo, como uma operação de E/S, ou por eventos externos (sistema operacional suspende por um período de tempo a execução do processo).

Espera⇒ Pronto

Um processo no estado de espera passa para o estado pronto quando a operação solicitada é atendida ou o recurso esperado é concedido. Um processo em estado de espera sempre terá que passar pelo estado de pronto antes de poder ser novamente selecionado para execução.

Execução⇒ Pronto

Um processo em execução passa para o estado de pronto por eventos gerados pelo sistema, como o término da fatia de tempo que o processador possui para sua execução.

Um processo em estado de pronto ou de espera pode não se encontrar na memória principal. Esta condição ocorre quando não existe espaço suficiente para todos os processos na memória principal e parte do contexto do processo é levada para a memória secundária. Uma técnica conhecida como swapping retira processos da memória principal e os traz de volta seguindo critérios de cada sistema operacional.

2.3. Tipos de processo

Além dos processos do usuário, a CPU também executa processos do sistema. Estes executam sempre, com certa prioridade, concorrendo com os processos do usuário.

Os processos em execução, do usuário, podem assumir dois tipos diferentes, de acordo com suas características de uso de CPU e periféricos:

Processo CPU-bound

• É aquele processo que utiliza muito a CPU.

• Ele ganha uma fatia de tempo e a utiliza por inteiro, sem desperdiçar nenhum tempo.

• É o caso de programas científicos, de cálculo numérico, estatística, matemática, e também na área de simulação.

• Normalmente fazem pouca ou nenhuma entrada de dados, e muito processamento.

Processo I/O-bound

• é o tipo de processo que utiliza muito mais E/S do que CPU.

• Aplicações em Banco de Dados, onde se faz consultas e atualizações constantes em arquivos em disco são um bom exemplo deste tipo de processo.

• De acordo com essas características, podemos dizer que este tipo de processo permanece mais tempo em espera (tratando interrupções) do que propriamente em execução, ocupando a CPU por períodos mínimos de tempo.

2.4. Comunicação entre processos

Frequentemente os processos do sistema precisam se comunicar com outros processos para a execução de determinadas tarefas. Neste contexto, dois paradigmas são mais utilizados: memória compartilhada (shared memory) e troca de mensagem (message passing). No primeiro, os processos tem acesso a uma área comum para leitura e escrita de informações. A segunda abordagem consiste no uso de primitivas para o envio/recebimento (send/receive) de mensagens entre processos.

Quando há compartilhamento de recursos entre dois ou mais processos, diversas situações estranhas podem ocorrer durante o processamento, e isso devido principalmente à interrupção provocada pelo escalonador do sistema operacional (preempção). Por exemplo, a atualização "descontrolada" de uma variável global compartilhada pode gerar resultados inconsistentes. Este tipo de situação é conhecida como situações de corrida (race conditions), porque o resultado final em algumas situações especiais depende da "competição" entre os processos e não do algoritmo implementado.

Uma tentativa de resolver este problema garante que os dados compartilhados só possam ser manipulados por um processo de cada vez, e esta estratégia é chamada de exclusão mútua (mutual exclusion) entre processos. A parte do programa que referencia dados compartilhados que possam sofrer condições de corrida com outros processos é denominada de região ou seção crítica (critical section) do processo. Portanto, uma maneira de evitar situações de corrida é garantir que dois ou mais processos nunca estejam executando simultaneamente na sua seção crítica.

Entretanto, esta restrição não é suficiente e uma boa solução para o problema envolve quatro requisitos básicos:

● Dois processos não podem estar simultaneamente acessando suas regiões críticas;

● A velocidade e quantidade de processadores não deve interferir no algoritmo;

● Um processo que esteja fora de sua seção crítica não deve impedir outro processo de usá-la;

● Um processo não deve esperar indefinidamente para acessar sua região crítica (evitar situação de starvation).

2.5. Escalonamento

A entidade responsável pelo escalonamento é o escalonador (scheduler), que é quem determina qual processo deve sair ou ir para a CPU em determinado momento. É ele o elemento responsável pela alocação de processos a processadores, definindo sua ordem de execução. A política de escalonamento (scheduling) é um problema complexo e depende do tipo de sistema suportado e da natureza das aplicações.

Em sistemas do tipo lote (batch), o escalonamento era feito simplesmente selecionando o próximo processo na fila de espera, já em sistemas multiusuário de tempo repartido geralmente combinados a sistemas em lote, o algoritmo de escalonamento deve ser mais complexo em virtude da existência de diversos usuários interativos solicitando serviços, e da execução de tarefas em segundo plano (background).

Um escalonador poderia funcionar da seguinte forma: quando a CPU tornasse disponível, o primeiro elemento da fila de processos prontos é retirado e inicia sua execução. Caso seja bloqueado, este irá para o final da fila de processos bloqueados. Se a sua quota de execução se esgotar, este será retirado da CPU e colocado no final da lista de processos prontos.

O escalonamento tem como principais objetivos:

● maximizar a utilização do processador;
● maximizar o número de processos completados por unidade de tempo;
● garantir que todos os processos recebam o processador;
● minimizar o tempo de resposta para o usuário.

Os tipos de escalonamento são:

Não-preemptivo: ocorre quando o processo que está executando não pode ser interrompido. Esse tipo de escalonamento estava presente nos primeiros sistemas multiprogramáveis, onde predominava o processamento em batch, e as políticas que implementam escalonamento não-preemptivo não são aplicáveis à sistemas de tempo compartilhado. Os primeiros sistemas de programas em lote (batch) inicialmente liam todas as instruções sequencialmente e depois as executavam uma após a outra.

Preemptivo: o processo pode ser retirado do processador que está executando. Permite atenção imediata aos processos mais prioritários (tempo real), melhores tempos de resposta (tempo compartilhado) e compartilhamento uniforme do processador.

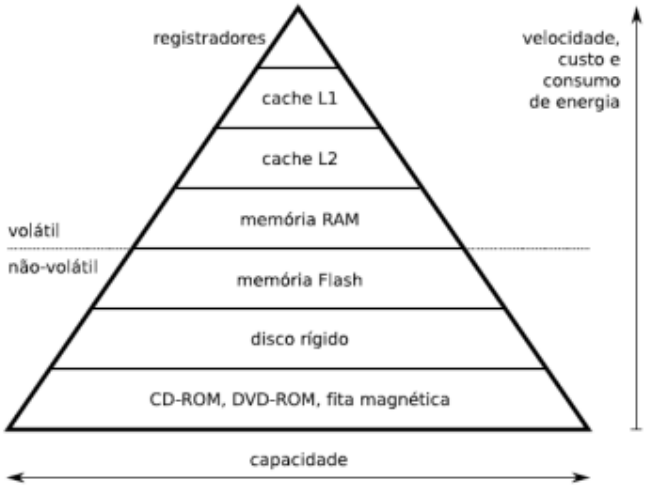
Existe também a multitarefa cooperativa onde não existe a figura do escalonador, pois nesse caso são os aplicativos que cooperativamente se revezam no uso dos recursos de CPU e memória. Porém, nesse caso, se determinado aplicativo apresentar algum problema trava o sistema. Era a multitarefa cooperativa que dava aos aplicativos escritos para Windows 3.X a impressão de ser multitarefa, muito embora todos eles rodassem sobre o DOS, um sistema monotarefa. Era também esse o principal motivo das "travadas"

frequentes, além de estimulador da posterior incorporação do botãozinho reset na parte da frente do gabinete do IBMPC.

3. Gerência de Memória

Memória é um importante recurso que deve ser cuidadosamente gerenciado. Enquanto a capacidade de armazenamento dos computadores vem crescendo continuamente, a complexidade do software cresce talvez à taxas maiores. A parte do sistema operacional que gerencia a memória é chamada de gerenciador de memória, sendo o objeto deste capítulo.

Dentre outras tarefas, o gerenciador de memória monitora quais partes da memória estão em uso e quais estão disponíveis; aloca e libera memória para os processos; gerencia a permuta de processos entre memória principal e secundária (quando a memória principal não é capaz de abrigar todos os processos).

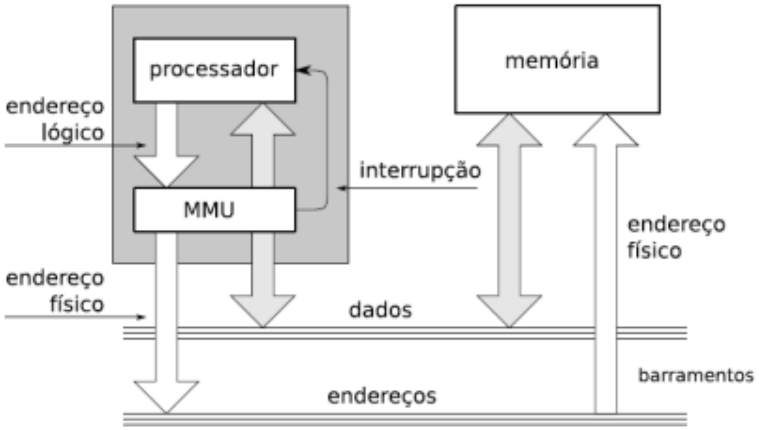


3.1. Endereços lógicos e físicos

Ao executar uma sequência de instruções, o processador escreve endereços no barramento de endereços do computador, que servem para buscar instruções e operandos, mas também para ler e escrever valores em posições de memória e portas de entrada/ saída. Os endereços de memória gerados pelo processador à medida que executa algum código são chamados de endereços lógicos, porque correspondem à lógica do programa, mas não são necessariamente iguais aos endereços reais das instruções e variáveis na memória real do computador, que são chamados de endereços físicos.

Os endereços lógicos emitidos pelo processador são interceptados por um hardware especial denominado Unidade de Gerência de Memória (MMU - Memory Management Unit), que pode fazer parte do próprio processador (como ocorre nos sistemas atuais) ou constituir um dispositivo separado (como ocorria nas máquinas mais antigas). A MMU faz a análise dos endereços lógicos emitidos pelo processador e determina os endereços físicos correspondentes na memória da máquina, permitindo então seu acesso pelo processador. Caso o acesso a um determinado endereço solicitado pelo processador não seja possível, a

MMU gera uma interrupção de hardware para notificar o processador sobre a tentativa de acesso indevido. O funcionamento básico da MMU está ilustrado na figura abaixo.



A proteção de memória entre processos é essencial para a segurança e estabilidade dos sistemas mais complexos, nos quais centenas ou milhares de processos podem estar na memória simultaneamente. A MMU pode ser rapidamente ajustada para mudar a forma de conversão entre endereços lógicos e físicos, o que permite implementar uma área de memória exclusiva para cada processo do sistema. Assim, a cada troca de contexto entre processos, as regras de conversão da MMU devem ser ajustadas para somente permitir o acesso à área de memória definida para cada novo processo corrente.

3.2. Alocação de Memória

A alocação de memória está dividida em:

Alocação Estática: Decisão tomada quando o programa é compilado.

Quando o programa é executado o Sistema operacional lê o mesmo e cria um processo, sendo o programa uma noção estática e o processo o programa em execução, ele é criado em armazenamento primário e após isso recebe um espaço na memória. O espaço de memória é dividido em varias partes, uma das partes se chama segmentos de memória, que armazena dados estáticos, e outro se chama segmento de código que guarda instruções do programa. Quando o programa é executado o registrador PC apontará para determinado endereço do segmento de código do processo, que se chama TEXT. Para que se realize a alocação estática o compilador deve saber o total de memória que está livre, mandar esta informação para o SO para que este crie um segmento de dados.

Alocação Dinâmica: Decisão é adiada até a execução. (Permite Swapping)

Os objetos alocados dinamicamente podem ser criados e liberados a qualquer momento, em qualquer ordem, o que difere dos objetos locais das funções, que são criados e destruídos em uma ordem específica. Dado isto, é preciso organizar a memória para objetos dinâmicos de uma forma que possibilite o gerenciamento do tempo de vida dos objetos por parte do programador. A memória reservada para objetos dinâmica costuma ser chamada de heap, existem várias formas de organizar um heap. Em linguagens sem gerenciamento automático(linguagem C), da memória dinâmica, uma organização usual do heap é uma lista encadeada de blocos livres, porém este tipo de organização pode ter problemas devido à fragmentação dos blocos. Já em linguagens com gerenciamento

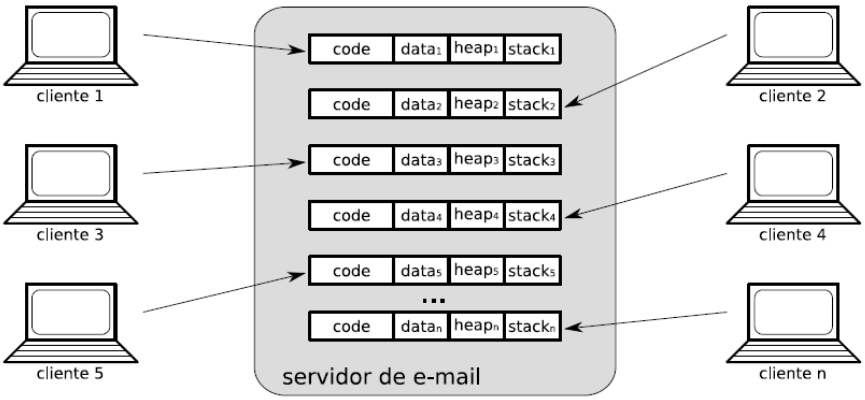
automático de memória dinâmica (Java), a organização do heap depende da parte do sistema de tempo de execução encarregada deste gerenciamento. Este componente é normalmente chamado de coletor de lixo.

Alocação Local:

Este processo de alocação é usado para variáveis que são locais a funções e sub-rotinas. Isso significa que o processo em execução deve manter acessível as variáveis locais da função ou procedimento que está executando no momento. Além disso, pelas propriedades do escopo em blocos, também devem estar acessíveis as variáveis de blocos mais externos. Em linguagens que permitem a definição de funções aninhadas, acessando as variáveis de quaisquer funções definidas externamente à função atualmente em execução. Como uma função pode chamar outras funções, um número arbitrário de funções pode estar no meio de sua execução em um determinado momento, mesmo que apenas uma esteja realmente sendo executada, isso indica que o contexto de várias funções deve ser mantido enquanto as mesmas não concluíram sua execução.

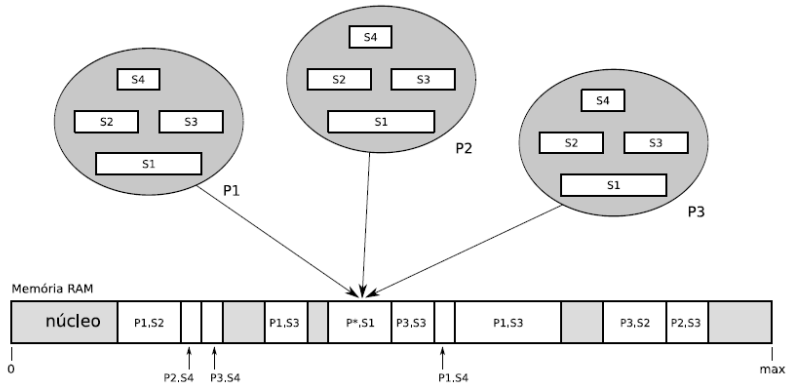
3.3. Compartilhamento de Memória

A memória RAM é um recurso escasso, que deve ser usado de forma eficiente. Nos sistemas atuais, é comum ter várias instâncias do mesmo programa em execução, como várias instâncias de editores de texto, de navegadores, etc. Em servidores, essa situação pode ser ainda mais frequente, com centenas ou milhares de instâncias do mesmo programa carregadas na memória. Por exemplo, em um servidor de e-mail UNIX, cada cliente que se conecta através dos protocolos POP3 ou IMAP terá um processo correspondente no servidor, para atender suas consultas de e-mail.

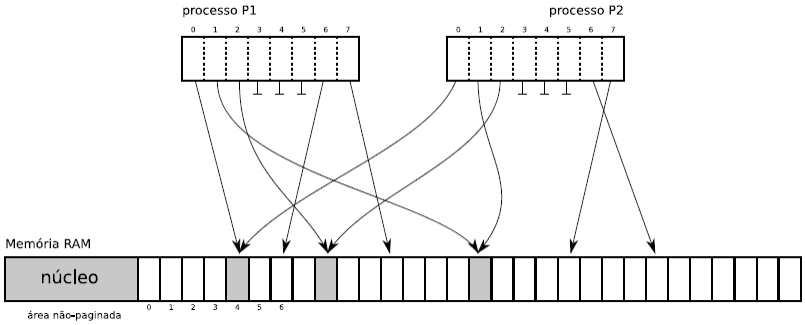


Todos esses processos operam com dados distintos (pois atendem a usuários distintos), mas executam o mesmo código. Assim, centenas ou milhares de cópias do mesmo código executável poderão coexistir na memória do sistema.

O compartilhamento de código entre processos pode ser implementado de forma muito simples e transparente para os processos envolvidos, através dos mecanismos de tradução de endereços oferecidos pela MMU, como segmentação e paginação. No caso da segmentação, bastaria fazer com que todos os segmentos de código dos processos apontem para o mesmo segmento da memória física. É importante observar que o compartilhamento é transparente para os processos: cada processo continua a acessar endereços lógicos em seu próprio segmento de código, buscando suas instruções a executar.



No caso da paginação, a unidade básica de compartilhamento é a página. Assim, as entradas das tabelas de páginas dos processos envolvidos são ajustadas para referenciar os mesmos quadros de memória física. É importante observar que, embora referenciem os mesmos endereços físicos, as páginas compartilhadas podem ter endereços lógicos distintos. A figura abaixo ilustra o compartilhamento de páginas entre processos.



3.4. Paginação

A técnica de partições fixas gera muita perda de memória e não é mais utilizada na prática. Embora partições variáveis seja um mecanismo mais flexível, o desperdício de memória em função da fragmentação externa é um grande problema. A origem da fragmentação externa está no fato de cada programa necessitar ocupar uma única área contígua de memória. Se essa necessidade for eliminada, ou seja, se cada programa puder ser espalhado por áreas não contíguas de memória, a fragmentação externa é eliminada. Esse efeito é obtido com a paginação.

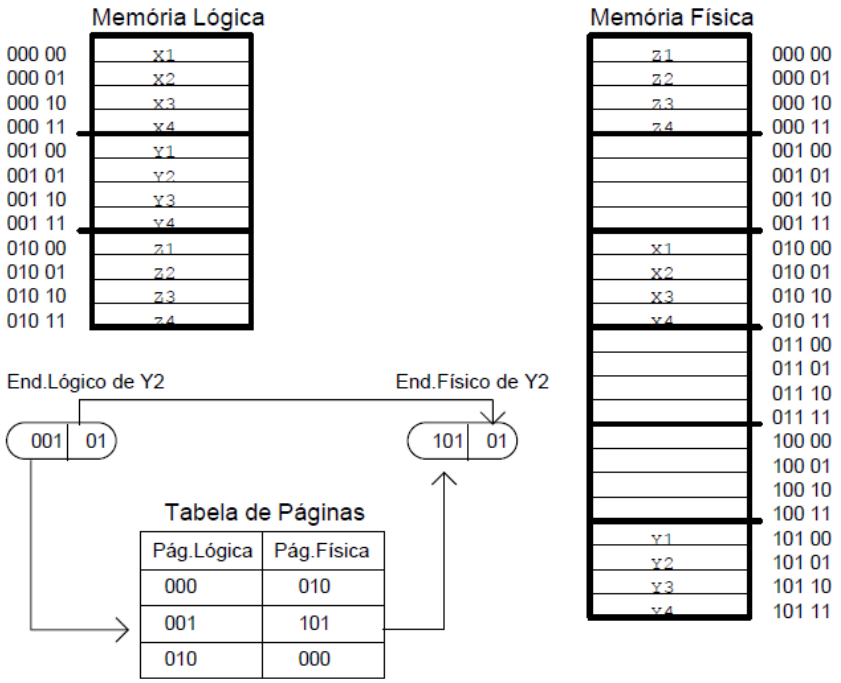
A figura a seguir ilustra o funcionamento da técnica de paginação. O exemplo da figura utiliza um tamanho de memória exageradamente pequeno para tornar a figura mais clara e menor. O espaço de endereçamento lógico de um processo é dividido em páginas lógicas de tamanho fixo. No exemplo, todos os números mostrados são valores binários. A memória lógica é composta por 12 bytes. Ela foi dividida em 3 páginas lógicas de 4 bytes cada uma. O endereço lógico também é dividido em duas partes: um número de página lógica e um deslocamento dentro dessa página. No exemplo, endereços lógicos possuem 5 bits.

Considere o byte Y2. Ele possui o endereço lógico 00101. Podemos ver esse endereço composto por duas partes. Os primeiros 3 bits indicam o número da página, isto é, 001. Os últimos 2 bits indicam a posição de Y2 dentro da página, isto é, 01. Observe que

todos os bytes pertencentes a uma mesma página lógica apresentam o mesmo número de página. De forma semelhante, todas as páginas possuem bytes com deslocamento entre 00 e 11.

A memória física também é dividida em páginas físicas com tamanho fixo, idêntico ao tamanho da página lógica. No exemplo, a memória física é composta por 20 bytes. A página física tem o mesmo tamanho que a página lógica, ou seja, 4 bytes. A memória física foi dividida em 5 páginas físicas de 4 bytes cada uma.

Os endereços de memória física também podem ser vistos como compostos por duas partes. Os 3 primeiros bits indicam um número de página física. Os 2 últimos bits indicam um deslocamento dentro dessa página física.



Um programa é carregado página a página. Cada página lógica do processo ocupa exatamente uma página física da memória física. Entretanto, a área ocupada pelo processo na memória física não precisa ser contígua. Mais do que isso, a ordem em que as páginas lógicas aparecem na memória física pode ser qualquer, não precisa ser a mesma da memória lógica. Observe que, no exemplo, o conteúdo das páginas lógicas foi espalhado pela memória física, mantendo o critério de que cada página lógica é carregada em exatamente uma página física. A página lógica 000 (X1, X2, X3 e X4) foi carregada na página física 010.

A página lógica 001 (Y1, Y2, Y3 e Y4) foi carregada na página física 101. A página lógica 010 (Z1, Z2, Z3 e Z4) foi carregada na página física 000. Durante a carga é montada uma tabela de páginas para o processo. Essa tabela informa, para cada página lógica, qual a página física correspondente. No exemplo, a tabela é formada por 3 entradas, uma vez que o processo possui 3 páginas lógicas.

Quando um processo executa, ele manipula endereços lógicos. O programa é escrito com a suposição que ele vai ocupar uma área contígua de memória, que inicia no endereço zero, ou seja, vai ocupar a memória lógica do processo. Para que o programa execute

corretamente, é necessário transformar o endereço lógico especificado em cada instrução executada, no endereço físico correspondente. Isso é feito com o auxílio da tabela de páginas.

3.5. Memória Virtual

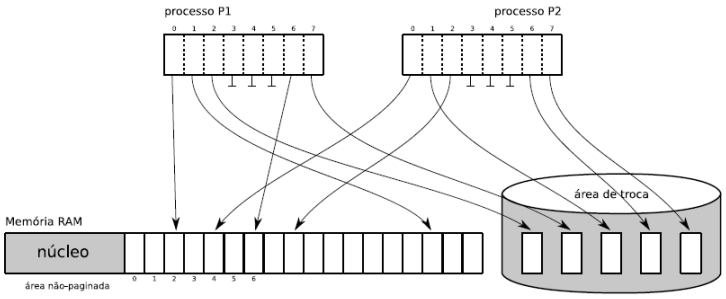
Um problema constante nos computadores é a disponibilidade de memória física: os programas se tornam cada vez maiores e cada vez mais processos executam simultaneamente, ocupando a memória disponível. Além disso, a crescente manipulação de informações multimídia (imagens, áudio, vídeo) contribui para esse problema, uma vez que essas informações são geralmente volumosas e seu tratamento exige grandes quantidades de memória livre. Como a memória RAM é um recurso caro (cerca de U$50/GByte no mercado americano, em 2007) e que consome uma quantidade significativa de energia, aumentar sua capacidade nem sempre é uma opção factível.

Observando o comportamento de um sistema computacional, constata-se que nem todos os processos estão constantemente ativos, e que nem todas as áreas de memória estão constantemente sendo usadas. Por isso, as áreas de memória pouco acessadas poderiam ser transferidas para um meio de armazenamento mais barato e abundante, como um disco rígido (U$0,50/GByte) ou um banco de memória flash (U$10/GByte), liberando a memória RAM para outros usos. Quando um processo proprietário de uma dessas áreas precisar acessá-la, ela deve ser transferida de volta para a memória RAM. O uso de um armazenamento externo como extensão da memória RAM se chama memória virtual; essa estratégia pode ser implementada de forma eficiente e transparente para processos, usuários e programadores.

Nos primeiros sistemas a implementar estratégias de memória virtual, processos inteiros eram transferidos da memória para o disco rígido e vice-versa. Esse procedimento, denominado troca (swapping) permite liberar grandes áreas de memória a cada transferência, e se justifica no caso de um armazenamento com tempo de acesso muito elevado, como os antigos discos rígidos. Os sistemas atuais raramente transferem processos inteiros para o disco; geralmente as transferências são feitas por páginas ou grupos de páginas, em um procedimento denominado paginação (paging), detalhado a seguir.

Normalmente, o mecanismo de memória virtual se baseia em páginas ao invés de segmentos. As páginas têm tamanho fixo, o que permite simplificar os algoritmos de escolha de páginas a remover, os mecanismos de transferência para o disco e também a formatação da área de troca. A otimização desses fatores seria bem mais complexa e menos efetiva caso as operações de troca fossem baseadas em segmentos, que têm tamanho variável.

A idéia central do mecanismo de memória virtual em sistemas com memória paginada consiste em retirar da memória principal os quadros pouco usados, salvando-os em uma área do disco rígido reservada para esse fim.



Algoritmos de substituição de páginas

A escolha correta dos quadros a remover da memória física é um fator essencial para a eficiência do mecanismo de memória virtual. Más escolhas poderão remover da memória quadros muito usados, aumentando a taxa de faltas de página e diminuindo o desempenho do sistema. Vários critérios podem ser usados para escolher "vítimas", páginas a transferir da memória para a área de troca no disco:

Algoritmo FIFO

Um critério básico a considerar para a escolha das páginas a substituir poderia ser sua "idade", ou seja, o tempo em que estão na memória. Assim, páginas mais antigas podem ser removidas para dar lugar a novas páginas. Esse algoritmo é muito simples de implementar: basta organizar as páginas em uma fila de números de páginas com política FIFO (First In, First Out). Os números das páginas recém carregadas na memória são registrados no final da lista, enquanto os números das próximas páginas a substituir na memória são obtidos no início da lista.

Algoritmo LRU

Uma aproximação implementável do algoritmo ótimo é proporcionada pelo algoritmo LRU(Least RecentlyUsed, menos recentemente usado). Neste algoritmo, a escolha recai sobre as páginas que estão na memória há mais tempo sem ser acessadas. Assim, páginas antigas e menos usadas são as escolhas preferenciais. Páginas antigas mas de uso frequente não são penalizadas por este algoritmo, ao contrário do que ocorre no algoritmo FIFO.

Algoritmo NRU

O algoritmo da segunda chance leva em conta somente o bit de referência de cada página ao escolher as vítimas para substituição. O algoritmo NRU (Not RecentlyUsed, ou não usada recentemente) melhora essa escolha, ao considerar também o bit de modificação (dirty bit, vide seção 3.4.1), que indica se o conteúdo de uma página foi modificado após ela ter sido carregada na memória.

4. Sistemas de Arquivos

Arquivos podem ser vistos como recipientes que contêm dados ou como um grupo de registros correlatos. Os arquivos armazenam informações que serão utilizadas, em geral, por programas aplicativos. Os arquivos são implementados através da criação, para cada arquivo no sistema, de uma estrutura de dados composta por registros. O descritor de arquivo é um registro que mantém informações sobre o arquivo, como posição de início e fim.

4.1. Arquivos e diretórios

As informações típicas (atributos) mantidos pelo sistema operacional são:

1. Nome do arquivo
2. Tamanho (bytes)
3. Data e hora da criação, do último acesso, da última modificação
4. Identificação do usuário que criou o arquivo
5. Listas de controle de acesso
6. Local do disco físico onde o conteúdo do arquivo foi colocado

As operações mais comuns relacionadas à manipulação dos arquivos são:

Create	Delete	Open	Close	Read	Write
Append	Seek	Get attributes	Set attributes	Rename	

Para controlar e organizar os arquivos, os sistemas de arquivos têm, em geral, os diretórios ou pastas, que podem ser arquivos em muitos sistemas também. Os diretórios podem ser organizados em um único nível (contendo todos os arquivos) ou em múltiplos níveis (diretórios dentro de diretórios).

Um dos principais problemas é como alocar espaço em disco para que os arquivos sejam armazenados de forma eficiente e que permita acesso rápido?

4.2. Alocação de arquivos

Existem alguns métodos que podem ser utilizados, tais como:

1. Alocação contígua
2. Alocação com Lista Ligada
3. Alocação com Lista Usando um Índice

Alocação Contígua

Este é o esquema mais simples de alocação de arquivos, onde cada arquivo é armazenado no disco como um bloco contíguo de dados. Neste esquema, em um disco com blocos de 1 KB, um pequeno arquivo de 20 KB seria armazenado em 20 blocos consecutivos.

Principais Vantagens:

1. simples implementação: controle de onde está cada arquivo no disco é feito por 1 único número (endereço em disco do 1º bloco).

2. performance: todo o bloco (arquivo) pode ser lido do disco de uma única vez. É necessário o tempo de somente um seek.

Problemas:

1. a estratégia só pode ser usada se o tamanho Máx. do arquivo for conhecido no momento de sua criação (devido a necessidade existente em saber o tamanho total do arquivo ou quantidade de blocos que ele ocupa).

2. fragmentação do disco: perde-se muito espaço útil com este esquema de alocação. Ao remover um arquivo a área ocupada pelo mesmo é liberada ocasionando lacunas por todo o disco. Necessidade de compactação (custo alto).

Alocação com Lista Ligada

Nesta estratégia de alocação, usamos uma lista ligada para indicar os espaços ocupados em disco pelo arquivo. Assim, não é mais necessário que o arquivo seja armazenado em posições contíguas do disco. A primeira palavra de cada bloco é usada com um ponteiro para o próximo bloco e o restante do bloco é usado para armazenar as informações (dados) do arquivo.

Principais Vantagens:

1. não se perde espaço por fragmentação externa.

2. qualquer bloco pode ser utilizado, permitindo que os arquivos cresçam indefinidamente enquanto houver espaço no disco.

3. a entrada do diretório só precisa armazenar o endereço do 1º bloco do arquivo (em cada bloco existirá um ponteiro para o próximo bloco do arquivo).

Problemas:

1. o acesso randômico é lento pois existe a necessidade de percorrer a lista.
2. a implementação deste método de alocação é mais complicada.

Alocação com lista ligada usando uma Tabela na Memória

Para eliminarmos os problemas apresentados por último, podemos dispor de uma tabela em memória armazenando os ponteiros para cada bloco do arquivo. Esta tabela recebe o nome de FAT (File Allocation Table) - esquema adotado pelo DOS. Com este esquema o acesso aleatório fica muito mais fácil, pois todo o esquema de ponteiros agora fica armazenado em memória, o que é muito mais rápido. O principal problema é o gasto com memória para manter a tabela com as informações. Para um disco de 40 GB e blocos de 1 KB, a tabela precisará de 40 milhões de entradas. Considerando que cada entrada tem no mínimo 3 bytes a tabela ocupará um espaço total de 120 MB.

Alocação com Lista Usando um Índice

Busca resolver o problema de "ponteiros" esparramados pelo disco que a alocação encadeada provoca. Para isso, mantém, por arquivo, um índice de blocos que o compõe. Este o índice é mantido em um bloco do disco. O diretório possui um ponteiro para o bloco onde está o índice associado a um determinado arquivo.

Este esquema elimina as desvantagens existentes na alocação com lista ligada: retira os ponteiros de cada um dos blocos e os coloca em uma tabela ou índice na memória. Apesar de o acesso ser randômico também, sua implementação é bem mais simples.

A tabela é armazenada na memória principal e pode ser seguida sem a necessidade de acessar o disco.

Desvantagem: a tabela também deve estar na memória o tempo todo, o que implica em utilização de espaço de memória.

INode

A cada arquivo associa-se uma pequena tabela, denominada INode, que lista os atributos e os endereços em discos dos blocos do arquivo. Os primeiros endereços de disco são armazenados no próprio INode. Se o arquivo for pequeno, toda a informação é encontrada diretamente no Inode. O conteúdo do arquivo só é transferido do disco para a memória quando o arquivo for aberto (esquema utilizado pelo UNIX).

5. Gerência de Dispositivos

Cada periférico do computador possui suas peculiaridades, assim, o procedimento de interação com uma placa de rede é completamente diferente da interação comum disco rígido SCSI. Todavia, existem muitos problemas e abordagens em comum para o acesso aos periféricos. Por exemplo, é possível criar uma abstração única para a maioria dos dispositivos de armazenamento como pen-drives, discos SCSI ou IDE, disquetes e etc, na forma de um vetor de blocos de dados. A função da gerência de dispositivos (também conhecida como gerência de entrada/saída) é implementar a interação com cada dispositivo por meio de drivers e criar modelos abstratos que permitam agrupar vários dispositivos distintos sob a mesma interface de acesso.

5.1. Dispositivos de entrada e saída

Os dispositivos de entrada e saída (E/S) são utilizados para permitir a comunicação entre o computador e o mundo externo. Através desses dispositivos a UCP e a memória principal podem se comunicar tanto com usuários quanto com memórias secundárias, a fim de realizar qualquer tipo de processamento.

Os dispositivos de E/S podem ser divididos em duas categorias: os que são utilizados como memória secundária e os que servem para a interface homem-máquina. Os dispositivos utilizados como memória secundária como discos se caracterizam por armazenar grande volume de informações, seu custo é relativamente baixo e seu tempo de acesso é maior que o acesso a memória principal.

Alguns dispositivos servem para a comunicação homem-máquina, como teclados, monitores de vídeo, impressoras, entre outros. Com o avanço no desenvolvimento de aplicações de uso cada vez mais geral procura-se aumentar a facilidade de comunicação entre o usuário e o computador. A implementação de interfaces mais amigáveis permite cada vez mais que pessoas sem conhecimento específico sobre informática possam utilizar o computador. Scanner, caneta ótica, mouse, dispositivos sensíveis a voz humana e ao calor do corpo humano são alguns exemplos desses tipos de dispositivos.

5.2. Device drivers

Device driver é o comando de um dispositivo ou programa. É a forma a partir da qual uma unidade periférica cria uma interface com o sistema operacional para se conectar com o dispositivo do hardware.

Vejamos um exemplo prático: quando se conecta uma impressora a um computador, esta impressora requer a instalação do "driver" (que é instalado a partir de um CD ou de um disquete que vem junto com o equipamento), sem o qual ela não conseguirá fazer a interface com o Computador. O "driver" é o elemento que faz esse comando. É, literalmente, o dirigente.

Foi a solução encontrada para que os Sistemas Operacionais sejam compatíveis com diferentes tipos de equipamentos. Cada impressora, por exemplo, tem suas pecualiaridades de hardware, logo torna-se inviável que o Sistema Operacional tenha conhecimento sobre todos os equipamentos disponíveis. O Sistema Operacional disponibiliza bibliotecas de programação, para que o fabricante possa criar uma interface entre seu equipamento e o software.

5.3. Acesso Direto à Memória (DMA)

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

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

O SO somente pode usar o DMA se o hardware tem o controlador de DMA. O DMA melhora o desempenho do sistema, pois poupo tempo ocioso da CPU, que poderia muito bem executar a tarefa do DMA, porém como o tempo de E/S é grande principalmetne para

grandes quantidades de dados pode fazer com que a CPU fique muito tempo ociosa. Quando a quantidade é pequena as vezes é até mais viável fazer a transferência direto pela CPU que é um hardware mais rápido que o DMA, isso pode causar concorrência no barramento, pois o barramento utilizado pelo DMA para acessar a memória é o mesmo utilizado pela CPU. Utilizando o DMA, a CPU requisita ao DMA de onde começar a ler os bytes, quantos bytes devem ser lidos/escritos e fica livre para executar outras tarefas que sejam CPU Bound, então quando o DMA termina de realizar sua tarefa, ele transmite um sinal de interrupção para a CPU que simplesmente usa os bytes.

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

5.4. Pedido de Interrupção (IRQ)

O Sistema Operacional (SO) chaveia entre os aplicativos ativos para que o usuário tenha a sensação de que estes estão executando em paralelo. O SO permite que um aplicativo utilize o processador durante um determinado período de tempo e então permite que outra aplicação o utilize. Como o chaveamento é feito de uma forma muito rápida temos a impressão de que os aplicativos estão sendo executados ao mesmo tempo.

Um pedido de interrupção (abreviação IRQ (em inglês)) é a forma pela qual componentes de hardware requisitam tempo computacional da CPU. Um IRQ é a sinalização de um pedido de interrupção de hardware.

Por exemplo: caracteres digitados no teclado, operações de leitura sobre o HD, dados recebidos pelo modem ou mesmo movimentos do mouse devem ser executados mesmo que a máquina esteja processando alguma tarefa.

Dessa forma IRQ´s são canais de comunicação com o processador. Ao receber um pedido através de algum destes canais, o processador percebe a solicitação de interrompimento de um dispositivo.

Quando um programa de usuário emite uma chamada ao sistema, esta é encaminhada ao driver apropriado. Para evitar que a CPU fique ocupada interrogando se dispositivo terminou a operação de E/S (espera ociosa), o driver pede ao dispositivo que lhe sinalize quando isso ocorrer. Dessa forma, o Sistema Operacional poderá executar outras tarefas enquanto o programa que o chamou pedindo o serviço se encontra bloqueado. Ao final da operação, o controlador do dispositivo gera uma interrupção, ao chip controlador de interrupção, para sinalizar à CPU.

Caso nenhuma outra interrupção esteja pendente ou em tratamento e nenhum outro dispositivo fez uma requisição de maior prioridade no mesmo momento, a interrupção é tratada imediatamente. Caso contrário, ela é ignorada, e o dispositivo continuará emitindo sinal de interrupção.

Após executar uma instrução, a CPU verifica as linhas de interrupções para saber se alguma interrupção foi sinalizada. Caso tenha sido sinalizada uma interrupção, o hardware grava os registradores da CPU na pilha do programa que estava em execução e carrega o contador de programa com o endereço da rotina referente à interrupção sinalizada.

Interrupções podem ser geradas por hardware ou por software. No caso do hardware, pelo dispositivos periféricos ou pelo relógio (timer). As interrupções geradas por software são conhecidas como system calls (chamadas ao sistema) ou trap (armadilha). A diferença primordial entre as interrupções geradas por software e as geradas por hardware,

está no fato de que, conhecendo um programa e seus dados, é possível prever quando as interrupções de software irão acontecer - o que não é possível no caso das interrupções de hardware.

6. Sistemas Operacionais Distribuídos

O que é um sistema distribuído?

É uma coleção de computadores independentes que aparenta ser um único computador aos seus usuários.

Conceitos relacionados a hardware e software, respectivamente.

Exemplos: Rede com estações de trabalho e processadores stand-alone alocados por demanda com um sistema de arquivos unificado. Sistema bancário com terminais espalhados geograficamente e um único banco de dados.

Como se tornou possível o surgimento desses sistemas?

Computadores mais rápidos e baratos e o surgimento de redes de computadores de alta velocidade.

Vantagens de sistemas distribuídos sobre sistemas centralizados.

Lei de Grosch – O poder de computação de um computador é proporcional ao quadrado do seu preço. (Aplicável a mainframes).

No caso de microprocessadores, é mais barato se comprar vários processadores e montá-los em um sistema multi-processador. (preço)

Em alguns casos, é teoricamente impossível de se construir um computador centralizado que possa ser comparado a um sistema distribuído com uma grande quantidade de processadores. (velocidade)

Algumas aplicações são inerentemente distribuídas - sistema de uma cadeia de lojas, jogos ou outras aplicações cooperativas. (distributividade inerente)

Maior confiabilidade, útil em aplicações de alto risco. (confiabilidade)

Crescimento incremental – acaba com necessidade da compra de mainframes cada vez maiores, agora o sistema pode ser trocado aos poucos. (crescimento incremental)

Vantagens de sistemas distribuídos sobre micros independentes:

Compartilhamento de dados. (colaboração e preço)

Compartilhamento de dispositivos. (preço)

Comunicação.

Mistura de computadores pessoais e compartilhados pode permitir uma distribuição de tarefas mais eficiente. (flexibilidade)

Desvantagens de sistemas distribuídos:

Software – sistemas operacionais, linguagens de programação e aplicações.

Comunicação – tratamento e recuperação de mensagens. Melhoria da rede pode acarretar em custos altos.

Segurança – Compartilhamento de dados implica em esquemas especiais para proteção de dados sigilosos.

Ideal – Pessoas trabalhando juntas e compartilhando informações sem se preocupar com distribuição física dos dados, máquinas e outros usuários.

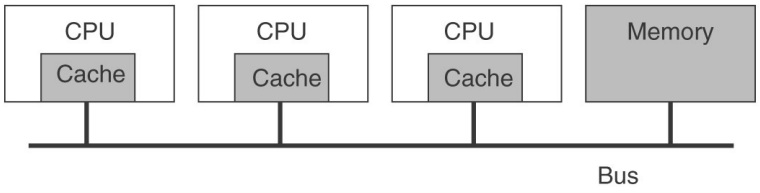
6.1. Conceitos de Hardware

Tanenbaum propõe a divisão de máquinas MIMD (múltiplas instruções e múltiplos dados) em multiprocessadores, que usam memória compartilhada e multicomputadores que possuem somente memória própria. Exemplos de multiprocessadores e multicomputadores. Outra subdivisão dessa classificação é em sistemas com barramento ou switches (ligações ponto-a-ponto). Outro ponto da taxonomia de Tanenbaum é a caracterização dos sistemas pelo grau de ligação entre as máquinas, que podem ser fortemente acopladas ou fracamente acopladas. Máquinas fortemente acopladas possuem um baixo retardo no envio de mensagens e uma alta taxa de transmissão, o oposto de máquinas fracamente acopladas. Geralmente, sistemas fortemente acoplados são usados como sistemas paralelos, trabalhando em um único problema enquanto que sistemas fracamente acoplados são utilizados como um sistema distribuído, trabalhando em diversos problemas. Geralmente multiprocessadores são sistemas fortemente acoplados enquanto que multicomputadores são sistemas fracamente acoplados.

Multiprocessadores baseados em barramento.

Consiste em um número de CPUs (que pode ter alguma memória local - cache) ligadas através de um barramento. Sem caches locais, o barramento tende a ser sobrecarregado rapidamente. Solução: adicionar caches locais. Novo problema: A coerência dos dados que estão armazenados em cada cache é fundamental.

Cache write-trough: Toda escrita na cache acarreta em escrita na memória. Escritas sempre geram tráfego no barramento, enquanto que leituras só geram tráfego quando a palavra não está na cache (cache miss). Para manter a consistência, as outras caches escutam o barramento e invalidam as posições que são escritas por outras caches na memória (snoopy caches). Um design deste tipo é coerente e invisível ao programador. É um esquema difícil de funcionar com um grande número de processadores.



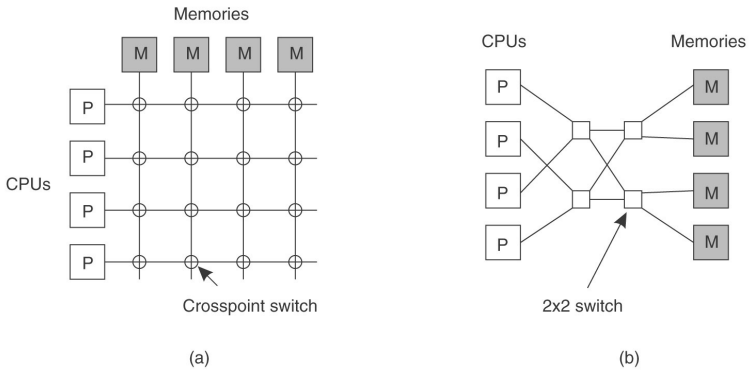
Multiprocessadores com switch.

Podem ser usados com barras cruzadas ou com pontos de cruzamento. Memórias são localizadas de um lado e os processadores do outro. Caminho em uma comunicação é switched e a memória em questão tem o acesso bloqueado para os outros processadores.

Número de switches pode tornar custo do sistema proibitivo. Rede ômega diminui número de switches necessários de n2 para n log2n. O retardo em redes ômegas com muitos níveis pode se tornar considerável, ou o seu custo caso switches ultra-rápidos sejam utilizados.

Solução: contruir um sistema que use uma hierarquia de memórias. Um design deste tipo é o NonUniform Memory Access (NUMA) onde cada CPU tem uma memória local além de memórias que servem a várias CPUs. O retardo diminui, mas a localização de software se torna crucial para o bom funcionamento do sistema.

Conclusão: construir um grande sistema de multiprocessadores fortemente acoplados com memória compartilhada é possível, mas é muito caro e complicado.

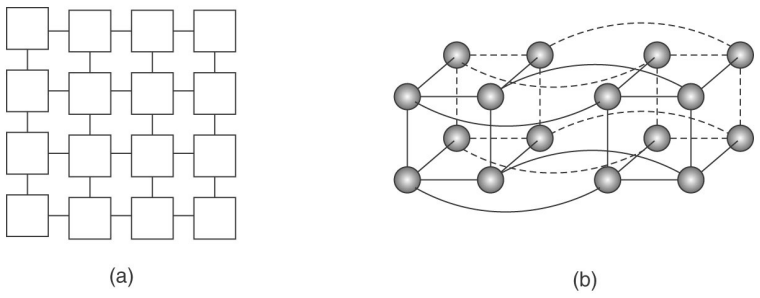


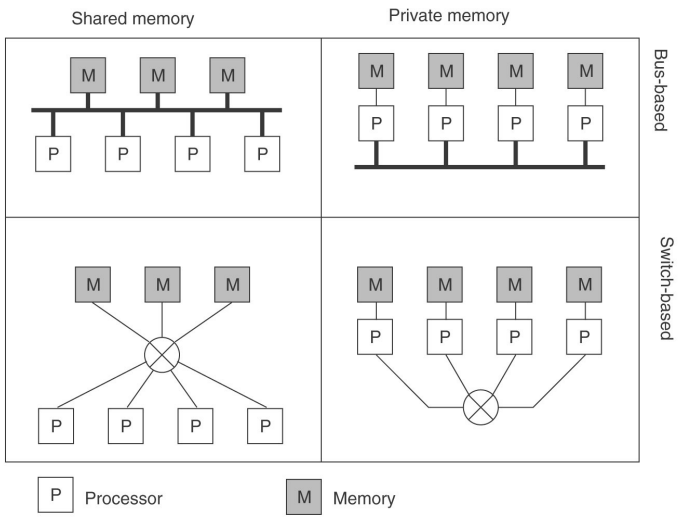
Multicomputadores baseados em barramento.

Conjunto de CPUs com memória local trocando mensagens através de um barramento. Rede local ou CPUs conectadas com um barramento rápido.

Multicomputadores com switch.

CPUs tem um certo número de conexões para outras CPUs e mensagens são trocadas através de CPUs que intermediam a comunicação quando necessário. Abaixo estão as topologias de grid e hipercubo. No grid, número de conexões e número máximo de passos em uma comunicação cresce com a raiz quadrada do número de CPUs enquanto que no hipercubo esse número cresce com o logaritmo do tamanho. Atualmente já são usados grids com dezenas de milhares CPUs. A Teragrid conecta milhares de computadores usando uma rede de 40 Gb/s.





6.2. Conceitos de Software

A imagem que o sistema apresenta aos usuários é quase que completamente determinada pelo sistema operacional. Software fracamente acoplado (loosely-coupled) permite que máquinas e usuários sejam fundamentalmente independentes dos outros. Exemplo: LAN. Software fortemente acoplado (tightly-coupled) funciona como uma única máquina. Exemplo: Computador multiprocessador que executa um programa de xadrez em paralelo.

Existem 8 possíveis combinações de hardware e software, entretanto somente quatro são distinguíveis pelos usuários já que os mesmos não tem a noção de qual tecnologia de interconexão está sendo utilizada (com barramento ou switches). Das 4 opções restantes, a combinação hardware fortemente acoplado (tightly-coupled) e software fracamente acoplado (loosely-coupled) não faz sentido, já que se estaria indo de encontro ao objetivo principal da construção de um SD.

Sistemas operacionais de rede

Loosely coupled – comunicação é explicitamente solicitada pelo usuário e se dá através de troca de mensagens. A distribuição do sistema é clara para o usuário.

Servidores de arquivos fornecem arquivos a máquinas clientes. Usuários diferentes tem visões diferentes do sistema, devido a diferenças em como os sistemas de arquivo locais estão organizados, quais dispositivos locais as máquinas possuem, etc.

Máquinas podem rodar diferentes versões de um mesmo SO ou até diferentes SOs desde que os protocolos de comunicação e serviços sejam usados por todas as máquinas. A "coordenação" (mínima) de serviços entre as máquinas é feita através a aderência aos protocolos.

Sistemas distribuídos verdadeiros

As vezes são chamados de sistemas operacionais distribuídos. Software fortemente acoplado (tightly-coupled) em um hardware fracamente acoplado (loosely-coupled). O objetivo é a criação de uma ilusão para os usuários que o sistema de multicomputadores

funciona como uma grande máquina de tempo compartilhado. As vezes são referenciados como sistemas de imagem única ou uniprocessador virtual.

Características de SDs:

Sistema de comunicação interprocessos único, independente se os dois processos estão localizados na mesma máquina ou não.

Esquema de proteção global.

Gerenciamento de processos único, isto é, como eles são criados, destruídos, iniciados ou parados. A idéia usada nos sistemas operacionais de rede de se estabelecer protocolos para a comunicação cliente-servidor não é suficiente. Deve haver um único conjunto de chamadas de sistema em todas as máquinas e as chamadas devem ser projetadas levando em conta o ambiente distribuído.

Sistema de arquivos deve ter a mesma visão e regras em todas as máquinas. Segurança deve ser implementada em relação aos usuários e não as máquinas.

Geralmente, o mesmo núcleo (kernel) roda em todas as máquinas, já que os mesmos implementam as mesmas funções e chamadas. Entretanto cada núcleo pode ter um controle razoável sobre os recursos locais, por exemplo, no gerenciamento de memória, swapping, ou até escalonamento de processos, no caso de uma CPU possuir mais de um processo sendo executado.

Sistemas multiprocessadores de tempo compartilhado

Software fortemente acoplado (tightly-coupled) em um sistema fortemente acoplado (tightly-coupled). Exemplo: multiprocessadores rodando Unix. Implementação mais simples que um verdadeiro sistema distribuído devido ao projeto do SO poder ser centralizado. Único sistema de arquivos, e fila de execução, que é mantida em um espaço de memória compartilhada. Somente uma cópia do SO é executada, ao contrário dos outros dois tipos de sistema.

Como nenhuma CPU possui memória local e todos os programas são armazenados em memória global compartilhada. Teoricamente um processo que seja executado por um longo período passaria aproximadamente o mesmo tempo sendo executado em cada CPU do sistema, entretanto é preferível a execução contínua na mesma CPU (caso esteja disponível) devido a pater do espaço de endereçamento do processo permanecer na cache da CPU. Processos bloqueados esperando por E/S podem ser suspensos ou marcados como esperando (busy waiting), dependendo da carga total do sistema. Implementa um sistema de arquivos tradicional.

6.3. Questões de Projeto

1- Transparência.

O objetivo final é passar a impressão de um sistema único aos usuários, logo a transparência é o aspecto mais importante em SOD. Transparência pode ser atingida em dois níveis: Escondendo-se a distribuição dos usuários e escondendo-se a distribuição dos programas (muito mais difícil). Qual o significado de transparência?

Transparência de Acesso – Esconde as diferenças de representação de dados e como os recursos são acessados.

Transparência de localização – Os usuários não sabem onde os recursos estão localizados.

Transparência de migração – Recursos podem ser movidos dentro do sistema sem que os usuários percebam.

Transparência de relocação – Mudança de local de um recurso é transparente aos usuários.

Transparência de replicação – Usuários não sabem quantas cópias de um programa ou arquivo existem.

Transparência de concorrência – Vários usuários dividem recursos sem saber.

Transparência de paralelismo – Atividades ocorrem em paralelo ser que os usuários saibam. Muito complicado. Geralmente o usuário tem que programar sua aplicação para que ela tome vantagem do paralelismo do sistema.

2- Flexibilidade.

Utilização de núcleos monolíticos ou de micro-núcleos associados com servidores a nível de usuário.

O núcleo monolítico é basicamente um SO com aplicações de conexão via rede e a integração de serviços remotos. Sistemas adaptados do Unix utilizam essa abordagem, pois o Unix usa um núcleo monolítico.

Os micro-núcleos são mais flexíveis e implementam quatro serviços:

1. Um mecanismo de comunicação inter-processo.
2. Algum gerenciamento de memória.
3. Um pouco de gerenciamento de baixo-nível de processos e escalonamento.
4. Entrada e saída de baixo-nível.

A vantagem do micro-núcleo é que é altamente modular: existem interfaces bem definidas para cada serviço e cada serviço é igualmente acessível para cada usuário independente da localização. Também facilita a implementação, instalação e depuração de novos serviços - flexibilidade. O teste de um novo serviço não acarreta em uma recompilação e reinicialização do núcleo. Usuários que não estão satisfeitos com os serviços podem implementar os seus próprios serviços.

Pode-se ter mais de um servidor de arquivos, com cada um funcionando com um formato. As únicas vantagens em potencial dos núcleos monolíticos são a performance e o custo inicial, no entanto um estudo experimental comparando o Sprite com o Amoeba não mostrou diferenças significativas em performance e o vantagem de custo inicial do núcleo monolítico tende a desaparecer a medida que serviços vão sendo implementados e/ou atualizados.

3- Confiabilidade.

Um dos objetivos originais em se construir SODs era o de se obter um sistema mais confiável que sistemas com um único processador. A idéia e que se uma máquina quebra, outra máquina do sistema pode assumir as suas funções. Teoricamente, a confiabilidade total do sistema seria um OU das confiabilidades dos componentes. “Um sistema distribuído é um sistema onde eu não consigo trabalhar porque alguma máquina na qual eu nunca ouvi falar não está funcionando” Leslie Lamport.

Aspectos de confiabilidade:

1. Disponibilidade (availability). Replicação de dados pode melhorar disponibilidade.
2. Segurança. Mais problemático do que em sistemas uniprocessador.
3. Tolerância a falhas. É desejável que a ocorrência de falhas seja transparente aos usuários. Servidores cooperativos podem ser usados para este fim.

4- Performance.

Problema: comunicação entre CPUs (essencial em um SD) é bem mais lenta que execução nas CPUs, devido ao overhead de protocolos. Reduzir o número de mensagens? Seria bom se isso não afetasse a execução paralela de processos, além da confiabilidade. Uso de granularidade de paralelismo na alocação de processos. Processos com paralelismo fine-grained, isto é, com um grande número de pequenas computações devem ser executados localmente enquanto que os coarse-grained, com grandes computações e pouca interação e dados são mais adequados a execução em paralelo.

5- Capacidade de expansão (scalability).

A maioria dos SDs são projetados para trabalhar com algumas centenas de CPUs. Soluções viáveis para sistemas com 200 máquinas podem falhar bisonhamente para um sistema com 200.000.000 máquinas.

Arquiteto do sistema deve evitar tabelas (catálogo telefônico), componentes (servidor de correio eletrônico) e algoritmos (que necessitem de um servidor centralizado) centralizados.

Algoritmos descentralizados tem as seguintes características:

1. Nenhuma máquina tem informação completa sobre o estado do sistema.
2. Máquinas tomam decisões baseadas somente em informações locais.
3. A quebra de uma máquina não faz com que o algoritmo falhe.
4. Não trabalham com nenhuma suposição de que existe um clock global.

6.4. Chamada remota a procedimento

Chamada remota de procedimento (RPC, acrônimo de Remote Procedure Call) é uma tecnologia de comunicação entre processos que permite a um programa de computador chamar um procedimento em outro espaço de endereçamento (geralmente em outro computador, conectado por uma rede). O programador não se preocupa com detalhes de implementação dessa interação remota: do ponto de vista do código, a chamada se assemelha a chamadas de procedimentos locais.

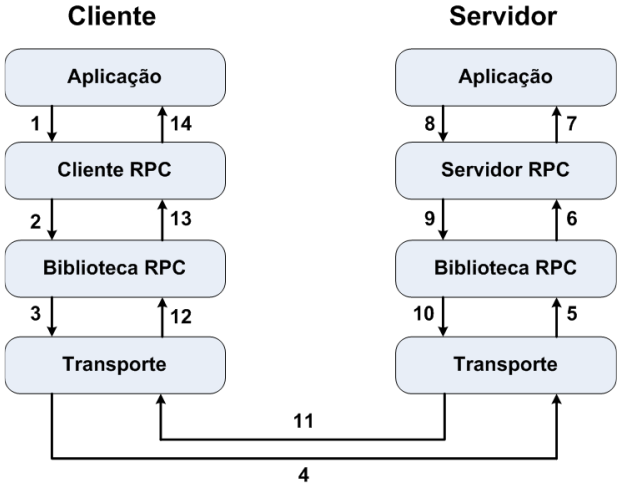
RPC é uma tecnologia popular para a implementação do modelo cliente-servidor de computação distribuída. Uma chamada de procedimento remoto é iniciada pelo cliente enviando uma mensagem para um servidor remoto para executar um procedimento específico. Uma resposta é retornada ao cliente. Uma diferença importante entre chamadas de procedimento remotas e chamadas de procedimento locais é que, no primeiro caso, a chamada pode falhar por problemas da rede. Nesse caso, não há nem mesmo garantia de que o procedimento foi invocado.

O modelo de Chamada Remota de Procedimento é similar ao modelo de chamadas locais de procedimentos, no qual a rotina que invoca o procedimento coloca os argumentos em uma área de memória bem conhecida e transfere o controle para o procedimento em execução, que lê os argumentos e os processa. Em algum momento, a rotina retoma o

controle, extraindo o resultado da execução de uma área bem conhecida da memória. Após isso, a rotina prossegue com a execução normal.

No modelo RPC, o processo de invocação ocorre de maneira similar. Uma Thread é responsável pelo controle de dois processos: invocador e servidor. O processo invocador primeiro manda uma mensagem para o processo servidor e aguarda (bloqueia) uma mensagem de resposta. A mensagem de invocação contém os parâmetros do procedimento e a mensagem de resposta contém o resultado da execução do procedimento. Uma vez que a mensagem de resposta é recebida, os resultados da execução do procedimento são coletados e a execução do invocador prossegue.

Do lado do servidor, um processo permanece em espera até a chegada de uma mensagem de invocação. Quando uma mensagem de invocação é recebida, o servidor extrai os parâmetros, processa-os e produz os resultados, que são enviados na mensagem de resposta. O servidor, então, volta a esperar por uma nova mensagem de invocação.



Nesse modelo, apenas um dos dois processos permanece ativo, em um dado instante de tempo. No entanto, esse modelo serve apenas de ilustração. O protocolo ONC RPC não faz restrições à implementações que permitam concorrência entre esses processos. Por exemplo, uma implementação poderia optar por chamadas assíncronas, que permitiriam ao cliente continuar com o trabalho útil, enquanto estivesse aguardando a mensagem de resposta.

Uma chamada remota de procedimento difere das chamadas locais em alguns pontos:

- Tratamento de erros: falhas do servidor ou da rede devem ser tratadas.
- Variáveis globais e efeitos colaterais: Uma vez que o servidor não possui acesso ao espaço de endereços do cliente, argumentos protegidos não podem ser passados como variáveis globais ou retornados.
- Desempenho: chamadas remotas geralmente operam a velocidades inferiores em uma ou mais ordens de grandeza em relação às chamadas locais.
- Autenticação: uma vez que chamadas remotas de procedimento podem ser transportadas em redes sem segurança, autenticação pode ser necessário.

Dessa forma, mesmo havendo diversas ferramentas que geram automaticamente o cliente e o servidor, os protocolos precisam ser desenvolvidos cuidadosamente.

Hino Nacional

Ouviram do Ipiranga as margens plácidas
De um povo heróico o brado retumbante,
E o sol da liberdade, em raios fúlgidos,
Brilhou no céu da pátria nesse instante.

Se o penhor dessa igualdade
Conseguimos conquistar com braço forte,
Em teu seio, ó liberdade,
Desafia o nosso peito a própria morte!

Ó Pátria amada,
Idolatrada,
Salve! Salve!

Brasil, um sonho intenso, um raio vívido
De amor e de esperança à terra desce,
Se em teu formoso céu, risonho e límpido,
A imagem do Cruzeiro resplandece.

Gigante pela própria natureza,
És belo, és forte, impávido colosso,
E o teu futuro espelha essa grandeza.

Terra adorada,
Entre outras mil,
És tu, Brasil,
Ó Pátria amada!
Dos filhos deste solo és mãe gentil,
Pátria amada,Brasil!

Deitado eternamente em berço esplêndido,
Ao som do mar e à luz do céu profundo,
Fulguras, ó Brasil, florão da América,
Iluminado ao sol do Novo Mundo!

Do que a terra, mais garrida,
Teus risonhos, lindos campos têm mais flores;
"Nossos bosques têm mais vida",
"Nossa vida" no teu seio "mais amores."

Ó Pátria amada,
Idolatrada,
Salve! Salve!

Brasil, de amor eterno seja símbolo
O lábaro que ostentas estrelado,
E diga o verde-louro dessa flâmula
- "Paz no futuro e glória no passado."

Mas, se ergues da justiça a clava forte,
Verás que um filho teu não foge à luta,
Nem teme, quem te adora, a própria morte.

Terra adorada,
Entre outras mil,
És tu, Brasil,
Ó Pátria amada!
Dos filhos deste solo és mãe gentil,
Pátria amada, Brasil!

Hino do Estado do Ceará

Poesia de Thomaz Lopes
Música de Alberto Nepomuceno
Terra do sol, do amor, terra da luz!
Soa o clarim que tua glória conta!
Terra, o teu nome a fama aos céus remonta
Em clarão que seduz!
Nome que brilha esplêndido luzeiro
Nos fulvos braços de ouro do cruzeiro!

Mudem-se em flor as pedras dos caminhos!
Chuvas de prata rolem das estrelas...
E despertando, deslumbrada, ao vê-las
Ressoa a voz dos ninhos...
Há de florar nas rosas e nos cravos
Rubros o sangue ardente dos escravos.
Seja teu verbo a voz do coração,
Verbo de paz e amor do Sul ao Norte!
Ruja teu peito em luta contra a morte,
Acordando a amplidão.
Peito que deu alívio a quem sofria
E foi o sol iluminando o dia!

Tua jangada afoita enfune o pano!
Vento feliz conduza a vela ousada!
Que importa que no seu barco seja um nada
Na vastidão do oceano,
Se à proa vão heróis e marinheiros
E vão no peito corações guerreiros?

Se, nós te amamos, em aventuras e mágoas!
Porque esse chão que embebe a água dos rios
Há de florar em meses, nos estios
E bosques, pelas águas!
Selvas e rios, serras e florestas
Brotem no solo em rumorosas festas!
Abra-se ao vento o teu pendão natal
Sobre as revoltas águas dos teus mares!
E desfraldado diga aos céus e aos mares
A vitória imortal!
Que foi de sangue, em guerras leais e francas,
E foi na paz da cor das hóstias brancas!

GOVERNO DO
ESTADO DO CEARÁ

Secretaria da Educação