Sistemas Operacionais

Prof. Marcos Ribeiro Quinet de Andrade
Universidade Federal Fluminense - UFF
Pólo Universitário de Rio das Ostras – PURO

Bibliografia

Livros-Texto:

TANENBAUM, A. S. **Sistemas Operacionais Modernos**. São Paulo : Prentice-Hall, 2004.
SILBERSCHATZ, A., GAGNE, G., GALVIN, P. B. **Sistemas Operacionais com Java**: conceitos e aplicações. Rio de Janeiro : Campus, 2004.

Livros de Referência:
DEITEL, H. M., DEITEL, P.J., CHOFINES, D.R. **Sistemas Operacionais**. São Paulo : Pearson Prenticce-Hall, 2005.
OLIVEIRA, R. S., CARISSIMI, A. S., TOSCANI, S. S. **Sistemas Operacionais**. Porto Alegre : Instituto de Informática da UFRGS: Editora Sagra Luzzatto, 2004.
TANENBAUM, A. S., WOODHULL. **Sistemas Operacionais: projeto e implementação.** 2a. ed.. Porto Alegre : Bookman, 2000.

Capítulo 1

Introdução aos Sistemas Operacionais

Um sistema operacional é um programa que atua como uma *interface* entre o *hardware* do computador e o usuário do sistema. Seu propósito é fornecer um ambiente no qual se possa executar programas. Suas metas são tornar o sistema do computador conveniente ao uso e que a utilização do *hardware* seja feita de modo eficiente.

Neste capítulo são apresentados uma definição do que é um sistema operacional, seus objetivos e divisão por áreas de aplicação. A seguir, é feito um breve histórico do desenvolvimento dos sistemas operacionais.

1. Introdução aos Sistemas Operacionais

1.1. Definição de Sistemas Computacionais

Um **sistema computacional** é um conjunto de recursos computacionais, sendo uma parte implementada em *hardware* e outra parte em *software*, dependendo do projeto, para interagirem cooperativamente.

Em torno de um sistema computacional, existem usuários com problemas distintos para serem resolvidos, por exemplo, um usuário precisa editar texto, enquanto outro precisa fazer a contabilidade da empresa. Deste modo, o problema de cada usuário deve ser atendido por um programa, ou aplicativo, específico que, neste caso, seriam, respectivamente, um editor de textos e um programa de contabilidade. Além disso, o sistema computacional deve possuir o *hardware* necessário, ou seja, os dispositivos físicos ou recursos físicos, para a execução desses programas.

De um modo geral, programas possuem muito em comum, por exemplo, tanto editor de texto como o sistema de contabilidade precisam acessar o disco, e a forma de acesso aos periféricos é a mesma para todos os programas.

Além disso, para um melhor aproveitamento dos recursos computacionais, os sistemas computacionais são projetados para que vários usuários possam compartilhar os recursos simultaneamente. Neste caso, os aplicativos podem apresentar necessidades conflitantes, pois muitas vezes disputam os mesmos recursos. Por exemplo, o editor de texto e o sistema de contabilidade podem solicitar, ao mesmo tempo, a única impressora do sistema computacional.

Assim, qualquer pessoa que utilize um sistema computacional deve saber que existe um software chamado **sistema operacional** (ao longo do curso, será referenciado muitas vezes apenas como S.O.), que de alguma forma controla os recursos do sistema computacional, qualquer que seja o seu tipo ou porte. Sistemas operacionais são necessários nos mais simples microcomputadores domésticos ou nos mais sofisticados supercomputadores de institutos de pesquisa ou de grandes corporações.

1.2. Objetivos da Disciplina

O correto entendimento dos mecanismos presentes nos S.O. permite ao profissional de informática uma melhor compreensão do seu ambiente de trabalho, resultando no desenvolvimento de soluções com maior qualidade e eficiência. Assim, os objetivos da disciplina são permitir ao profissional de informática a atuar nas áreas de suporte, seleção e recomendação de S.O., a obter visão básica do desenvolvimento e implementação de um S.O. e, principalmente, fornecer subsídios necessários para a compreensão do funcionamento de S.O.

1.3. Definição do Sistema Operacional

O sistema operacional é um programa que controla e coordena o uso do *hardware* do computador entre os vários programas de aplicação para os vários usuários. Assim, podemos dizer que o sistema operacional é um conjunto de módulos de *software* que regem os recursos do sistema, resolvem seus conflitos, simplificam o uso da máquina e otimizam seu desempenho global.

Um sistema computacional é constituído de vários elementos, que atuam em diferentes níveis, como exemplificado na figura 1.1. Observe que o sistema operacional, juntamente com aplicações que constituem o software do sistema, pertence a uma mesma camada, cujo objetivo fundamental é atender às solicitações dos programas de aplicação do usuário por funcionalidades desempenhadas pelo *hardware* do sistema.

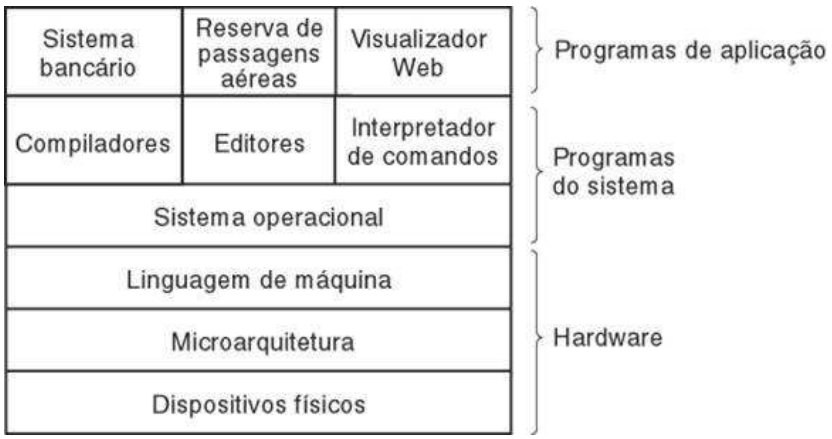


Figura 1.1. Estrutura simplificada de um sistema computacional

Um sistema operacional deve ter o completo domínio sobre os recursos da máquina. O escalonamento de recursos, o controle de entrada e saída (E/S), a gerência da memória, a gerência do processador, o escalonamento de processos e a segurança são funções que o sistema operacional deve exercer.

No projeto de um sistema operacional deve ter em mente dois objetivos principais:

- Apresentar ao usuário do computador uma forma amena de utilizar a máquina. Criar uma máquina virtual, de fácil compreensão para o usuário, com características diferentes da máquina física;
- Realizar o melhor uso possível do *hardware* disponível, aumentando o desempenho do sistema e diminuindo o custo.

Finalmente, o sistema operacional pode ser visto como uma **máquina estendida** (ou máquina virtual), ocultando detalhes complexos do funcionamento do hardware que constitui o sistema computacional, e como um **gerenciador de recursos**, que podem ser compartilhados no **tempo** (diferentes programas ou usuários aguardam sua vez de usá-los) ou no **espaço** (vários programas ou usuários utilizam simultaneamente uma parte do recurso), por exemplos, várias aplicações ocupam partes da memória do sistema.

Como sistemas computacionais são empregados em ambientes de tarefas bastante heterogêneos, é razoável assumir que existam diferentes sistemas operacionais (ou diferentes versões de um mesmo), que sejam mais adequadas para a realização de um conjunto de tarefas específico. É possível dividir as **áreas de aplicação** de um sistema operacional em **sistemas de tempo real** e **sistemas de processamento de tarefas**, definidos a seguir.

1.4. Classificação de Sistemas de Computação em Relação ao Funcionamento

Os diversos tipos de sistemas computacionais diferem entre si, principalmente nas tarefas de administração dos processos e capacidade de atendimento aos processos, por exemplo, alguns sistemas computacionais tratam um só processo de usuário por vez, ou seja, mantém apenas um processo de usuário na memória, enquanto outros sistemas computacionais já suportam múltiplos processos de simultaneamente em memória.

Entretanto, o simples fato de existir mais de um processo em memória, não garante que todos, ou alguns, serão atendidos ao mesmo tempo, ou seja, paralelamente. A estrutura de funcionamento de um sistema computacional depende dos recursos de *hardware* presentes, bem como da capacidade do sistema operacional em atender as solicitações dos processos, em disponibilizar os recursos aos processos e ao escalonamento de uso do(s) processador(es).

A seguir veremos uma classificação de sistemas computacionais, considerando a estrutura de funcionamento dos mesmos, porém lembramos que devemos complementar esse estudo em outros textos, pois existem diferenças de classificação entre diferentes autores. Assim, o que apresentamos a seguir são conceitos básicos, sem seguir especificamente um único autor.

a) Sistemas monoprogramados:

São os mais simples e foram largamente utilizados. Permitiam que somente um programa por vez fosse processado, ou seja, somente um programa tinha posse de todos os recursos do sistema computacional, indiferente da sua necessidade de uso, consequentemente, a utilização dos recursos não será otimizada. Além disso, o programa em execução deverá administrar todo o sistema computacional. Se este tipo de controle for utilizado em equipamentos caros, a relação custo x benefício tende a ser desvantajosa.

b) Sistemas multiprogramados:

Com o objetivo de melhorar a relação custo x benefício e visando a otimização do uso dos recursos, surge, no início dos anos 70, o conceito e a primeira implementação de multiprogramação. A idéia era otimizar a utilização de recursos destinando determinadas tarefas a componentes específicos do sistema computacional, por exemplo, quando executa-se uma rotina de entrada e saída (E/S ou I/O), pode-se atribuir o procedimento de E/S a um hardware/software dedicado à função. Assim, o processador ficaria ocioso, podendo ser utilizado por outro processo.

Para tanto, deve ser mantido em memória, simultaneamente, uma quantidade *n* de processos, capazes de ocupar o processador durante todo o tempo. Uma variação um pouco mais sofisticada dos sistemas multiprogramados, são os sistemas que fazem multitarefa cooperativa, que seguem o mesmo conceito com algumas melhorias.

c) Sistemas de tempo compartilhado:

Os sistemas de tempo compartilhado (*time sharing* ou *time shared*), da mesma forma que os sistemas multiprogramados, possibilitavam que, em um instante de tempo vários processos fossem alocados na memória, entretanto, a execução se daria em intervalos cíclicos de tempo, determinando uma "fatia" de tempo (*time slice*) para cada processo.

O tempo compartilhado é largamente utilizado nos equipamentos de grande porte e permitiu a implementação de sistemas multiusuários, onde vários usuários, de forma simultânea e *online*, concorrem pelo tempo de execução. Como exemplo de sistemas de tempo compartilhado, podemos citar os sistemas de multitarefa preemptiva.

d) Sistemas multiexecutados:

Outra evolução "natural" dos sistemas monoprogramados são os sistemas multiexecutados, pois a própria evolução do hardware permitiu a execução de várias tarefas concorrentemente, através de tempo compartilhado, sendo que essas tarefas são comandadas por um único usuário (monousuário). Ou seja, os sistemas multiexecutados são similares aos de tempo compartilhado, porém não permitem multiusuários.

e) Sistemas multiprocessados:

Os sistemas multiprocessados permitem que durante um determinado instante de tempo, vários processos estejam alocados na memória, sendo executados simultaneamente (processamento paralelo), não de forma concorrente, como em outros sistemas multitarefas (por exemplo, sistemas exclusivamente multiprogramado ou exclusivamente por tempo compartilhado). Os sistemas computacionais que suportam multiprocessamento possuem mais do que um elemento processador.

A execução de dos programas nos diversos processadores não impede a utilização dos periféricos em outras tarefas independente dos processadores (assim como, nos sistemas de tempo compartilhado, multiprogramados e multi-executados) de forma a otimizar o uso destes. Sistemas multiprocessados são ambientes com sistemas operacionais complexos e sofisticados destinados a sistemas computacionais de alto desempenho.

1.5. Conceitos Básicos de Sistemas Operacionais

Um sistema computacional é composto por um ou mais processadores, uma certa quantidade de memória, terminais, discos magnéticos, interfaces de rede e dispositivos de E/S, etc., ou seja, estamos lidando com um sistema extremamente

complexo, e desenvolver software que gerencie e integre esses componentes, fazendo-os trabalhar corretamente e de forma otimizada, não é tarefa fácil.

O sistema operacional é uma camada de software colocada entre os recursos de hardware e os programas que executam tarefas para os usuários, sendo que uma de suas principais responsabilidades é permitir e promover o acesso aos periféricos, sempre que um programa solicitar uma operação de E/S. Através dessa intervenção do sistema operacional, o programador não precisa conhecer detalhes do hardware, por exemplo, informações de como “enviar um caractere para a impressora”, são internas ao sistema operacional. Além disso, como todos os acessos aos periféricos são feitos através do sistema operacional, fica mais fácil controlar a disponibilização dos recursos, buscando uma distribuição justa, eficiente e conveniente, não só dos recursos de E/S, mas de todo o sistema computacional.

Uma utilização mais eficiente do sistema computacional é obtida a partir da distribuição de seus recursos entre os programas, por exemplo, a distribuição do espaço em disco, da memória principal, do tempo do processador, do acesso a disco, etc. Já a utilização mais conveniente é obtida a partir da disponibilização dos recursos do sistema computacional, sem que os usuários conheçam os detalhes dos recursos.

Como exemplo, vamos imaginar um usuário especializado, um programador, que ao desenvolver um programa precisa colocar um caractere na tela. Para tanto, em geral, é necessária toda uma sequência de acessos à interface do vídeo, diversos registradores devem ser lidos ou escritos e, além disso, podem existir diferentes tipos de interfaces que exigirão diferentes sequências de acesso. Porém, através do sistema operacional, o programador apenas informa, no programa, qual caractere deve ser colocado na tela e todo o trabalho de acesso ao periférico é feito pelo sistema operacional.

Ao esconder detalhes dos periféricos, muitas vezes são criados recursos de mais alto nível, ou seja, as abstrações. Por exemplo, os programas utilizam o espaço em disco através do conceito de arquivo. Arquivos não existem no hardware, mas formam um recurso criado a partir do que o hardware oferece. Para o programador é muito mais confortável trabalhar com arquivos do que receber uma área de espaço em disco que ele próprio teria que organizar.

Como exemplo de abstração, considere uma instrução de E/S, por exemplo, READ ou WRITE em um IBM PC, que deve ser acompanhada de 13 parâmetros, especificando o endereço do bloco a ser lido, o número de setores por trilha, o modo de gravação no meio físico, o espaçamento entre setores, etc., sem contar com as tarefas de ordem física/mecânica, por exemplo, verificar se o motor do drive já está acionado. A grande maioria dos programadores não se envolve com tais detalhes, pois lida com uma abstração de alto nível e, portanto mais simples. No exemplo em questão, a abstração feita nos discos é visualizá-los como uma coleção de arquivos identificados por nomes, onde os eventos, a manipulação dos arquivos, não consideram maiores detalhes e restringem-se a simplesmente abrir, ler/escrever e fechar.

1.6. Arquitetura de Sistemas Operacionais

A arquitetura de um sistema operacional é a estrutura básica sobre a qual é projetado o sistema operacional, de como as abstrações são realmente implementadas, como o sistema computacional deve ser solicitado e atender aos aplicativos, como interagem as partes do sistema operacional entre si e como o sistema operacional responde às solicitações dos aplicativos.

1.6.1. Arquitetura monolítica:

É a arquitetura mais antiga e mais comum. Cada componente do sistema operacional é contido no núcleo (*kernel*) e pode comunicar-se com qualquer outro componente diretamente. Essa intercomunicação direta permite rapidez na resposta de sistema operacional monolíticos, entretanto como núcleos monolíticos agrupam os componentes todos juntos, é difícil identificar a origem de um determinado problema ou erro. Além disso, todo o código do sistema operacional é executado com acesso irrestrito ao sistema, o que pode facilitar a ocorrência de danos provocados intencionalmente, ou não, por outros aplicativos.

1.6.2. Arquitetura em camadas:

À medida que os sistemas operacionais tornaram-se mais complexos e maiores, projetos puramente monolíticos tornaram-se inviáveis e, então a arquitetura em camada, ou modular, tornou-se uma boa opção, agrupando “camadas” de componentes, ou seja, conjunto de procedimentos, que realizam tarefas similares.

Cada camada comunica-se somente com as suas camadas imediatamente inferior e superior. Uma camada inferior sempre presta um serviço à sua camada superior, sendo que a camada superior não sabe como o serviço é feito, apenas o solicita. A implementação de uma camada pode ser modificada sem exigir modificação em outra camada, pois possuem componentes autocontidos.

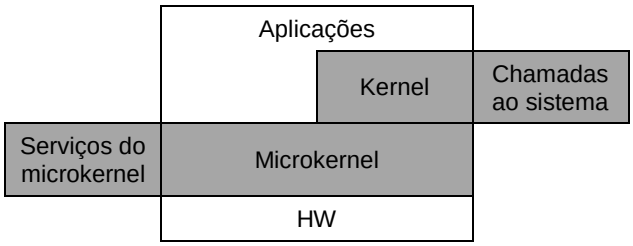
Em uma abordagem em camadas, a solicitação de um serviço pode precisar passar por muitas camadas antes de ser atendida, assim o desempenho se degrada em comparação ao de núcleos monolíticos.

1.6.3. Arquitetura de micronúcleo:

A arquitetura de micronúcleo (*microkernel*) também é uma forma de arquitetura modular ou em camadas. Na tentativa de reduzir os procedimentos mais fundamentais, somente um pequeno número de serviço tais como parte do gerenciamento de memória, a sincronização entre processos e a comunicação entre processos, terá acesso direto ao HW, como representado na figura 1.2.

Por sua vez, o serviço de comunicação entre processo, que está dentro do micronúcleo, é o responsável por habilitar os serviços de, p.ex., redes, sistemas de arquivos, gerenciamento de dispositivos, etc., que normalmente, podem ser implementados no núcleo do sistema operacional, não no micronúcleo, ou até como procedimentos (aplicativos) externos ao núcleo.

Alguns sistemas operacionais permitem que as aplicações acessem diretamente os serviços oferecidos pelo micronúcleo. Por também ser uma arquitetura modular, a arquitetura em micronúcleo possui, em geral, as mesmas vantagens e desvantagens da arquitetura em camadas.



1.2. Representação da organização de um SO com arquitetura de microkernel.

1.7. A Função do Sistema Operacional

O sistema operacional é um programa que controla e coordena o uso do *hardware* do computador entre os vários programas de aplicação para os vários usuários. Assim, podemos dizer que o sistema operacional é um conjunto de módulos de *software* que regem os recursos do sistema, resolvem seus conflitos, simplificam o uso da máquina e otimizam seu desempenho global.

Um sistema operacional deve ter o completo domínio sobre os recursos da máquina. O escalonamento de recursos, o controle de entrada e saída (E/S), a gerência da memória, a gerência do processador, o escalonamento de processos e a segurança são funções que o sistema operacional deve exercer.

No projeto de um sistema operacional deve-se ter em mente dois objetivos principais:

- Apresentar ao usuário do computador uma forma amena de utilizar a máquina. Criar uma máquina virtual, de fácil compreensão para o usuário, com características diferentes da máquina física;
- Realizar o melhor uso possível do *hardware* disponível, aumentando o desempenho do sistema e diminuindo o custo.

É possível dividir as áreas de aplicação de um sistema operacional em **sistemas de tempo real** e **sistemas de processamento de tarefas**, que estão definidos a seguir.

17.1. Sistemas de Tempo Real

Os sistemas de tempo real são aqueles que devem fornecer uma resposta a estímulos externos num período de tempo extremamente pequeno. Podem-se citar os seguintes exemplos:

- **controle de processos:** Os computadores são utilizados para controlar processos industriais, tais como o refino de petróleo bruto, controle de temperatura em alto fornos entre outros. Em comum, tais aplicações têm a necessidade de receber uma resposta rápida após a emissão de um sinal de controle.

- **consulta a base de dados:** O computador aqui, é utilizado para obter informações armazenadas em grandes bancos de dados. Geralmente, o usuário deste tipo de aplicação desconhece como se processam as operações do sistema, e espera um tempo de resposta pequeno para obter suas informações. Um exemplo de tal aplicação seria uma consulta a informações sobre censo.

- **processamento de transações:** Neste caso, o computador é utilizado para realizar acessos a bancos de dados que estão frequentemente sendo atualizados, talvez várias vezes em um segundo. Aplicações típicas são reservas de passagens e consultas bancárias.

1.7.2. Sistemas de Processamento de Tarefas

Os sistemas de processamento de tarefas são projetados para manipular um fluxo contínuo de programas que serão executados pelo computador. Tendo em vista a grande variedade de programas que podem ser processados, tal sistema precisa suportar um grande número de utilitários. Esses utilitários podem ser compiladores para diversas linguagens, montadores (*assemblers*), *linkers*, editores de texto entre outros. É necessário também dar suporte a um sistema de arquivos para gerenciar o armazenamento de informações.

Podemos classificar os sistemas de processamento de tarefas em dois grupos:

- ***Batch***: A principal característica desse grupo é o fato de que o usuário perde o controle do programa a partir do momento em que ele o submete ao sistema.
- **Interativo:** A característica marcante desse grupo é permitir a monitoração e o controle do programa, através de um terminal, enquanto durar o processamento.

1.8. A Evolução dos Sistemas Operacionais

Antigamente existia somente o *hardware* do computador. O operador e programador da máquina eram uma só pessoa. Todo o controle do sistema era feito através de botões e *displays* no console. O operador/programador monitorava a execução de um programa interativamente. A ativação (*setup*) do computador era muito lenta e a depuração era extremamente trabalhosa[1]. Assim, era necessário buscar soluções que tornassem mais fácil e mais eficiente à utilização do computador. Os sistemas operacionais foram então criados com estas finalidades.

Os sistemas operacionais, assim como os dispositivos eletrônicos, vêm sofrendo mudanças ao longo das últimas décadas. Podem ser identificadas várias gerações, que estão descritas a seguir.

Primeira Geração (Anos 50)

Em 1953 surgiu o primeiro sistema operacional. Desenvolvido pela GM Laboratories, ele foi desenvolvido para um computador IBM modelo 701. Os primeiros

[1] Procure saber mais a respeito do funcionamento do ENIAC, o primeiro computador digital eletrônico, e ainda, sobre o Altair 8800, um dos primeiros computadores pessoais

sistemas eram voltados para o processamento em *batchs* (lotes). O sistema operacional era responsável pela entrada de um programa e saída de outro, isto é, o sequenciamento de *jobs.* Uma vez que houvesse um processo rodando, ele tinha completo controle sobre a máquina. Ao término (normal ou anormal) do processo, o controle retornava ao sistema operacional que preparava a maquina para receber o próximo programa. Ainda assim, o tempo de ativação (*setup*) do computador era enorme.

Segunda Geração (Anos 60)

Nesta época, várias empresas já fabricavam sistemas operacionais. O principal objetivo era terminar o maior número de processos por unidade de tempo, aumentando o desempenho de um sistema de computador.

Foram desenvolvidos também nesta época os primeiros sistemas com as seguintes características:

- **Multiprogramação:** A multiprogramação permite que vários processos sejam executados simultaneamente.
- **Multiprocessamento:** No multiprocessamento, vários programas são processados ao mesmo tempo em processadores diferentes.
- ***Time-sharing*:** São ditos de tempo compartilhado. Usuários interagem com a máquina de uma maneira conversacional através de terminais.
- **Tempo real:** Dentre eles se destaca o SABRE para reservas de passagens da *American Airlines*.

Apareceu também nesta geração, o conceito de dispositivo independente. Nos sistemas da primeira geração, o usuário que precisasse escrever dados em uma fita, deveria referenciar, no seu programa, especificamente qual fita ele desejava Na segunda geração, o usuário deveria apenas referenciar a necessidade de uma fita. O sistema, então, ficaria responsável por reservar uma fita disponível para ele.

Em abril de 1964, a IBM lançou a série de computadores System/360 que eram compatíveis em termos de arquitetura e tinham o mesmo sistema operacional OS/360. Esse sistema era adequado tanto para aplicações científicas quanto para aplicações comerciais. Antes disso, cada vez que fosse necessário um sistema computacional mais potente, eram oferecidos sistemas totalmente diferentes, o que implicava em uma conversão de *hardware e* de *software* lenta e muito cara. Com a série 360 era possível passar anos e anos sem precisar de conversões. Foi um verdadeiro sucesso.

Terceira Geração (meados dos anos 60 a meados dos anos 70)

A terceira geração começou efetivamente com a introdução da família 360. Os computadores dessa geração foram desenvolvidos para serem **sistemas de propósito geral**. Eram sistemas que suportavam simultaneamente o processamento *batch, sime-sharing,* tempo-real e multiprocessado. Eram sistemas grandes e caros. Este conceito vendeu muitos computadores, mas existia um *overhead* muito alto relativo ao tempo em que a máquina gastava executando rotinas do sistema operacional. Além disso, determinadas aplicações não necessitavam de todos os modos de processamento.

Unia exceção deste conceito foi o sistema operacional UNIX, que foi construído nesta época. No final dos anos 60, Ken Thompson e Dennis Ritchie, entre outros membros da equipe do Bell Laboratories desenvolveram e implementaram um ambiente interativo, o UNIX. Usando esse sistema, desenvolveram também a linguagem C. Uma grande parte do sistema operacional foi escrita em C, o que contribuiu para a popularidade de ambos.

Os sistemas começaram a ser escritos em linguagens de alto nível e surgiram as linguagens de controle que permitem controlar a execução de processos.

Outro marco importante foi o surgimento da engenharia de *software*. Essa viria a ditar regras para a construção de sistemas. Os sistemas operacionais que eram um aglomerado de programas escritos por pessoas com muito mais noção de *hardware* do que *software* passaram a ser escritos de uma maneira mais disciplinada.

Quarta Geração (Meio dos anos 70 ao final dos anos 80)

Nesta fase apareceram os sistemas operacionais para redes de computadores, onde o usuário ganha acesso a redes locais ou geograficamente dispersas. O ponto importante era transferir informação entre computadores interconectados. Correio eletrônico, transferência de arquivo e aplicações de acesso a banco de dados proliferaram nesta época. O modelo **cliente/servidor** tornou-se difundido. Os **clientes** são os processos dos usuários que necessitam de vários serviços e os **servidores** são os componentes de *hardware/software* da rede que realizam estes serviços. Os servidores são geralmente dedicados a um tipo de tarefa tais como impressão, acesso a banco de dados entre outros.

O conceito de processamento distribuído tornou-se largamente difundido. Quando necessários, dados eram trazidos para serem processados em alguma instalação de computador central de larga escala.

Com o advento do microprocessador, surgem os computadores pessoais, um dos mais importantes desenvolvimentos com consequências sociais das últimas décadas. Um computador poderia ser adquirido por um preço acessível a muitos usuários que passariam a ter o seu próprio computador.

A quantidade de pessoas com acesso a um computador tornou-se consideravelmente maior e o termo *user friendly* começou a ser muito utilizado. Ele significa que o sistema apresenta, para usuários leigos, um ambiente de acesso fácil ao computador.

O conceito de máquina virtual tornou-se largamente utilizado. Os usuários não precisavam mais se preocupar com detalhes físicos do sistema do computador que está sendo usado, ao invés disso, o usuário via uma máquina virtual criada pelo sistema operacional. O campo da engenharia de *software* continuou a ter uma importância significativa.

Quinta Geração (Final dos anos 80 ao presente)

Nos anos 90 entramos na verdadeira era da **computação distribuída.** As computações são divididas em subcomputações. Essas são executadas em diferentes processadores, que podem ser computadores multiprocessadores ou redes de computadores. As subcomputações podem ser distribuídas de tal maneira que se possa obter vantagens utilizando computadores de propósito especial através das redes.

As redes podem ser configuradas dinamicamente. Elas continuam operando ainda que novos dispositivos e *softwares* sejam adicionados ou removidos. Quando cada novo servidor for adicionado, ele dará a rede, através de um procedimento de registro, informações sobre suas capacidades, políticas de ligação, acesso entre outras. Os clientes podem então usar os servidores, quando necessário, de acordo com os termos descritos durante o registro. Para atingir flexibilidade real, clientes não teriam conhecimento dos detalhes da rede.

Este tipo de conectividade é facilitada por padrões de sistemas abertos e protocolos. Esses padrões estão sendo desenvolvidos por alguns grupos internacionais como *International Organization for Standardization (*ISO*), Open Software Foundation*, X/Open, entre outros. Eles pretendem chegar a um acordo sobre um ambiente internacionalmente aceito para padrões de comunicação e de computação. A tendência é a computação tornar-se muito poderosa e portátil.

Nos anos recentes, foram introduzidos os computadores *laptop*, que possibilitam às pessoas transportarem seus computadores por toda parte. Com o desenvolvimento de protocolos de comunicação os computadores *laptop* podem ser ligados em redes de comunicação e transmitir dados com alta confiabilidade.

1.9. Resumo

Os sistemas operacionais foram desenvolvidos ao longo das últimas décadas com dois propósitos principais. Primeiro, fornecer um ambiente conveniente ao desenvolvimento e a execução de programas. Segundo, controlar as atividades computacionais (recursos do *hardware) para* garantir um bom desempenho do sistema do computador.

Existem várias definições para os sistemas operacionais. Dentre elas, uma define os sistemas operacionais como um programa que atua como uma interface entre o *hardware* da máquina e o usuário do sistema.

Inicialmente os computadores eram manipulados através de consoles. Não havia nenhum mecanismo que facilitasse a sua utilização. Em uma fase posterior, foram desenvolvidos os montadores, carregadores, compiladores que melhoraram a tarefa da programação do sistema. Porém, o tempo de *setup* era grande, o que levou então ao desenvolvimento de sistemas em *batch.*

Depois desta época é que começaram a surgir os sistemas mais elaborados como os sistemas multiprogramados, multiprocessados, de tempo compartilhado e de tempo real. Um dos grandes destaques foi o conceito desenvolvido nos sistemas da IBM, a

série 360. Com essa série era possível mudar de um sistema para outro sem precisar de mudanças radicais.

Um outro destaque foi o desenvolvimento do sistema operacional UNIX, escrito em sua maior parte com uma linguagem de alto nível, o C. Mais recentemente surgiram os sistemas operacionais para rede de computadores. O conceito de processamento distribuído tornou-se difundido. Computadores pessoais tornaram-se acessíveis e possuem *softwares* amigáveis.

Atualmente, as redes de computadores têm obtido uma importância real. A computação distribuída vem sendo submetida a computadores multiprocessadores e a redes. O modelo cliente/servidor vem sendo desenvolvido, buscando portabilidade e melhor desempenho. Um outro tipo de computador que surgiu é denominado *laptop*. Os *laptops* são pequenos e podem ser transportados com facilidade[2].

[2] Existe alguma diferença entre notebooks e laptops? Faça uma pesquisa a respeito.

Capitulo 2

Serviços do Sistema Operacional

Uma visão bastante comum do sistema operacional é aquela que encara este *software* como uma extensão da máquina, fornecendo mais serviços para os aplicativos e outros programas básicos. Além disso, o sistema operacional pode ser considerado também um administrador e fornecedor de recursos (incluindo serviços). Ele cuida de todos os recursos que estão disponíveis no computador, permitindo ao usuário utilizar a máquina (*hardware* + SO) de maneira amigável. Isso é fundamental em qualquer máquina, tornando-se mais crítico naquelas que permitem mais de um usuário ao mesmo tempo.

Neste capítulo são apresentados os serviços que são oferecidos pelo sistema operacional e mostradas a visão do usuário e a visão do sistema operacional, em relação à máquina.

2.1. Tipos de Serviços

O sistema operacional fornece um ambiente para a execução de programas através de serviços para os programas e para os usuários desses programas. Alguns serviços não têm como preocupação apenas tornar a máquina mais confortável para o usuário, mas também para que o próprio sistema seja mais eficiente e seguro. Esse é o caso dos serviços oferecidos nos sistemas que permitem vários usuários compartilhando todos os recursos da máquina. Apesar da forma como esses serviços são oferecidos variar de sistema para sistema existem algumas classes de serviços que são comuns a todos os sistemas operacionais. Estes compõem a sua própria definição. Como exemplo, temos:

- **Execução de programas:** O sistema operacional é o responsável por carregar um programa na memória principal da máquina e executá-lo. O programa é o responsável pelo término da sua própria execução;

- **Operações de entrada/saída:** Durante a sua execução, um programa pode ter necessidade de se comunicar com o meio externo à máquina. Esta operação recebe o nome de entrada/saída (E/S) e pode envolver qualquer dispositivo de E/S (disco, impressora). Como um programa não pode executar estas operações diretamente, o sistema operacional é o responsável por fornecer meios adequados para isso;

- **Manipulação de sistema de arquivos:** Os usuários de uma máquina têm necessidade de realizar acessos aos arquivos pelo nome para saber se eles existem, para apagá-los ou até para renomeá-los. Um programa em execução pode querer ler ou escrever num arquivo qualquer. O sistema operacional é o responsável por gerenciar o sistema de arquivos da máquina. Este gerenciamento inclui a alocação de espaço no dispositivo de armazenamento secundário, a busca otimizada a um determinado arquivo e o armazenamento de todas as informações necessárias sobre cada arquivo.

- **Detecção de erros:** O sistema operacional é o responsável por detectar erros possíveis que podem comprometer a execução de qualquer programa e a segurança da máquina. Estes erros podem envolver o próprio processador a memória principal (acesso a uma área proibida), os dispositivos de entrada/saída (falta de papel na impressora), ou até mesmo o programa do usuário (uma divisão por zero). Para cada tipo de erro, o sistema operacional tem uma ação apropriada para garantir a corretude e a consistência da computação.

- **Alocação de recursos**: O sistema operacional é o responsável pela alocação dos diversos recursos em sistemas com um ou mais usuários. Estes recursos incluem a memória principal, a própria UCP, arquivos e os dispositivos de E/S A alocação deve ser feita da forma mais eficiente possível para não prejudicar o desempenho do sistema.

- **Proteção:** O sistema operacional é o responsável pela proteção a todo o sistema computacional. Essa proteção se torna necessária tanto em sistemas monousuários quanto em sistemas multiusuários. A única diferença é a sua complexidade. Quando vários usuários estão usando o sistema, a execução de um programa não pode interferir na execução de outro. Além disso, o próprio sistema operacional deve ser protegido de erros cometidos pelos usuários.

2.2. Usuários x Sistema Operacional

A seguir serão apresentadas as duas formas de o sistema operacional fornecer serviços aos usuários: chamadas ao sistema e programas utilitários.

Chamadas ao Sistema

O nível mais fundamental de serviços fornecido pelo sistema operacional é realizado através de **chamadas ao sistema** (*system calls*) As chamadas fornecem uma interface entre um programa em execução e o sistema operacional. Estão, geralmente, disponíveis como instruções nas linguagens de baixo nível ou até mesmo em linguagens de alto nível, como C. As chamadas ao sistema podem ser classificadas, em duas categorias principais:

- **Controle de processos**. Nessa categoria existem chamadas ao sistema para a criação e a finalização de processos, a manipulação de tempo para manter a sincronização entre processos concorrentes, o carregamento e a execução de programas, a obtenção e a ativação de atributos dos processos, entre outras.

- **Gerenciamento de arquivos e de dispositivos de E/S**. Nesta categoria existem chamadas ao sistema para criar, apagar, abrir e fechar um arquivo, ler e escrever em arquivos, e ainda obter e modificar os atributos de um arquivo, entre outras. Os atributos de um arquivo incluem, por exemplo, o seu nome, tipo, códigos de proteção e tamanho. Em geral, a única informação necessária para o sistema realizar um acesso a um arquivo é o seu nome. No caso de dispositivos de E/S, existem ainda as chamadas para requisitar e liberar um dispositivo.

Na figura 2.1 é mostrado um esquema de atendimento do sistema operacional às chamadas ao sistema. Inicialmente (1), o processo escreve nos registradores da máquina o código da chamada e os seus parâmetros. A seguir (2), o processo executa uma instrução ***trap*** e o sistema operacional interpreta a chamada (3), executa o serviço solicitado (4) e devolve o controle para o processo (5).

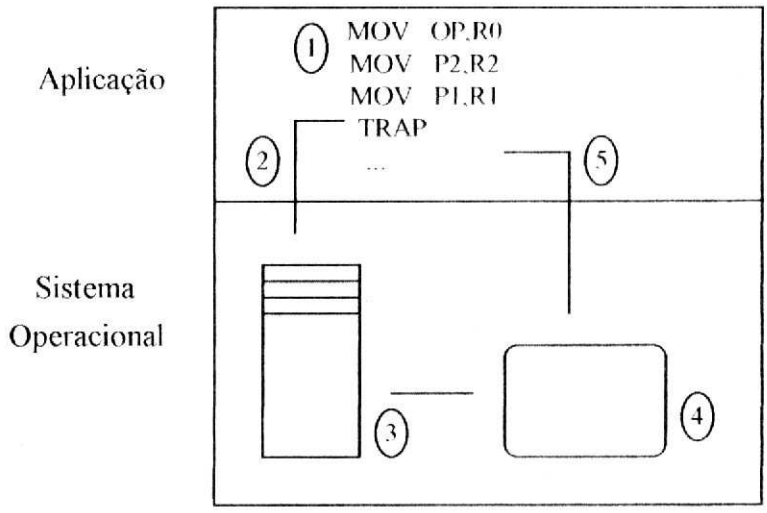


Figura 2.1. Exemplo de chamada ao sistema

Uma instrução *trap* muda o estado do processador do modo usuário para o **modo supervisor** (ou **sistema**). No primeiro modo, as instruções que afetam o *hardware* não podem ser executadas e, no segundo, todas as instruções podem ser executadas A distinção entre o modo usuário e o supervisor ou sistema existe para proteção do sistema operacional e para proteção entre usuários.

Na implementação de uma chamada ao sistema são necessárias mais informações do que simplesmente identificar a chamada desejada. A quantidade e o tipo exato de informações variam de acordo com a chamada e o sistema operacional em particular. Em geral, existem dois métodos usados para passar parâmetros ao sistema operacional. A abordagem mais simples é passá-los em registradores. Entretanto, em alguns casos, pode haver mais parâmetros do que registradores. Nestes casos os parâmetros são geralmente armazenados em blocos ou tabelas na memória principal e o endereço do bloco ou tabela é passado como um parâmetro em um registrador.

Para exemplificar o uso das chamadas ao sistema podemos considerar um programa em execução que leia informações de um arquivo e escreva em outro. As primeiras informações de que o programa necessita são os nomes dos arquivos que serão manipulados. Esses nomes podem ser pré-definidos dentro do próprio programa em execução ou passados pelo usuário durante a execução (isso exige uma outra sequência de chamadas ao sistema). Uma vez que os nomes dos arquivos tenham sido obtidos, o programa deve abrir o arquivo de entrada, que vai ser lido (*open*) e criar *o* arquivo de saída (*create*). Algum tipo de tratamento de erro pode ser previsto, como por exemplo, se o arquivo de entrada ainda não foi criado, se ele possui algum tipo de proteção ou se já existe algum arquivo com nome igual ao do arquivo de saída. Cada uma dessas operações requer uma chamada ao sistema.

Depois que os dois arquivos já foram abertos, o próximo passo é ler do arquivo de entrada (*read*) e escrever no arquivo de saída Neste ponto, tanto na leitura como na escrita, outras situações também podem ser previstas, como a chegada ao final do arquivo, erro no dispositivo de armazenamento secundário, falta de espaço no disco para o arquivo de saída.

Finalmente, depois que o arquivo de entrada foi totalmente copiado para o arquivo de saída, o programa em execução deve fechar os dois arquivos (*close*) e terminar normalmente (*end*).

Programas Utilitários

Outros tipos de serviços oferecidos pelo sistema operacional são os chamados **programas utilitários** *(systems programs).* Esses serviços são de mais alto nível e fornecem uma interface entre o usuário e o sistema operacional. Como foi visto no exemplo anterior, um programador pode escrever um programa que copie o conteúdo de um arquivo de entrada para um arquivo de saída, e utilizá-lo sempre que necessário. Embora possível esta não seja a forma mais confortável, pois obriga o usuário a conhecer detalhes complexos da máquina, e a ter conhecimento de técnicas de programação. Além disso, vários usuários têm necessidade deste tipo de operação e seria bastante interessante que existisse alguma padronização.

Dessa forma então, a maioria dos sistemas operacionais oferece um conjunto de programas utilitários que tornam o ambiente mais conveniente para o desenvolvimento e execução de programas. Podem-se agrupar os programas utilitários da seguinte forma:

- **Manipulação de arquivos:** Inclui apagar, copiar, ver o conteúdo, renomear e imprimir um arquivo e ainda verificar a existência do arquivo;

- **Informações sobre o sistema:** Inclui obter e modificar data e hora da máquina, conhecer a quantidade de memória secundária disponível e outras informações de estado.

- **Suporte para linguagens de programação:** Junto com o sistema operacional podem sem fornecidos, ou vendidos separadamente, programas como compiladores, montadores, interpretadores para as linguagens de programação mais comuns como Pascal, Fortran ou C;

- **Carregamento e execução de programas:** Uma vez compilado, montado e livre de erros, um programa deve ser carregado na memória principal e executado. O sistema pode oferecer instrumentos para que essas duas operações sejam efetuadas mais eficientemente (carregadores absolutos ou realocáveis, *linkage editors* e depuradores).

Existe ainda um programa utilitário muito importante que recebe o nome de **interpretador de comandos.** Esse programa começa a executar quando o sistema é iniciado e fica esperando que um usuário digite um comando para interpretá-lo. Existem duas alternativas possíveis na implementação do interpretador de comandos. O interpretador pode conter o código que executa o comando pedido (ativa os parâmetros e invoca as chamadas ao sistema necessárias) ou então cada comando é implementado por um programa independente. Cada programa possui um nome próprio, e a função do interpretador é carregá-lo na memória principal e executá-lo.

No primeiro caso, o tamanho do interpretador de comandos depende da quantidade de comandos existentes, já que uma parte do seu código contém o código do próprio comando. No segundo caso, o interpretador apenas ativa o programa que contém o comando. Além disso, outros comandos podem ser adicionados ao sistema através da criação de arquivos novos e o tamanho do interpretador pode ser relativamente pequeno.

A visão que os usuários têm do sistema operacional é definida mais pelos programas utilitários que pelas chamadas ao sistema, particularmente pelo interpretador de comandos. A visão que um projetista do sistema operacional tem do sistema como um todo é bastante diferente. Ele vê os recursos físicos (disco, impressora, teclado, vídeo) e deve convertê-los em facilidades lógicas fornecidas aos usuários.

2.3. Comportamento do Sistema Operacional

Os sistemas operacionais têm uma característica bastante interessante: se não existir nenhum programa a executar, nenhum dispositivo de entrada/saída a ser atendido e nenhum usuário aguardando uma resposta, o sistema operacional fica esperando a ocorrência de algum evento. Eventos são, em geral, representados por **interrupções.**

Quando uma interrupção ocorre, o *hardware* transfere o controle para o sistema operacional. Nesse momento, o sistema operacional salva o estado da máquina (registradores e contador de programa) e determina qual foi o tipo de interrupção que ocorreu, já que existem diferentes tipos de interrupção. Por exemplo, uma chamada ao sistema *(create)*, uma interrupção de um dispositivo de E/S (impressora), uma interrupção de erro (tentativa de executar uma instrução privilegiada). Para cada tipo de interrupção um tratamento diferente tem de ser dado.

Na visão do sistema operacional, as chamadas ao sistema são agrupadas de acordo com o seu tipo. Para cada chamada um segmento de código é executado. Apesar de o sistema tratar muitas chamadas, a maioria dos eventos que ocorrem pertencem à classe de interrupções dos dispositivos de E/S. Uma operação de E/S é resultante de uma chamada ao sistema requisitando tal serviço. Uma situação prática seria um programa abrir um arquivo e escrever alguma informação nele. Neste caso, o dispositivo de E/S usado poderia ser um disco e várias operações de E/S e uma sequência de chamadas ao sistema seriam realizadas.

Uma vez que uma operação de entrada/saída tem início, dois cenários podem ocorrer. O primeiro é quando o controle só retorna para o programa do usuário, que gerou a chamada ao sistema, quando a operação tiver sido terminada. No segundo, o controle retorna ao programa do usuário sem esperar que a operação de entrada/saída tenha sido terminada. Ambas as situações têm vantagens e desvantagens.

O primeiro caso é mais simples, pois apenas uma operação de entrada/saída fica pendente a cada momento. Em compensação, limita a quantidade de operações simultâneas que podem ser feitas. O segundo caso é mais, complexo, pois, várias operações de entrada/saída podem ficar pendentes ao mesmo tempo e o sistema operacional precisa identificá-las para poder tratá-las.

Outro tipo de interrupções que merecem uma atenção especial são as interrupções de erro. Elas são geradas quando o programa que está sendo executado tenta realizar algo não permitido pelo sistema operacional Por exemplo, realizar o acesso uma posição de memória protegida. Sempre que uma interrupção de erro ocorre e o sistema operacional deve terminar o programa em execução de forma anormal. Uma mensagem de erro é enviada e a área de memória utilizada pode ser copiada num arquivo *(dump)* para que o usuário possa tentar descobrir a causa do erro.

A figura 2.2 a seguir ilustra o fluxo geral do comportamento de um sistema operacional.

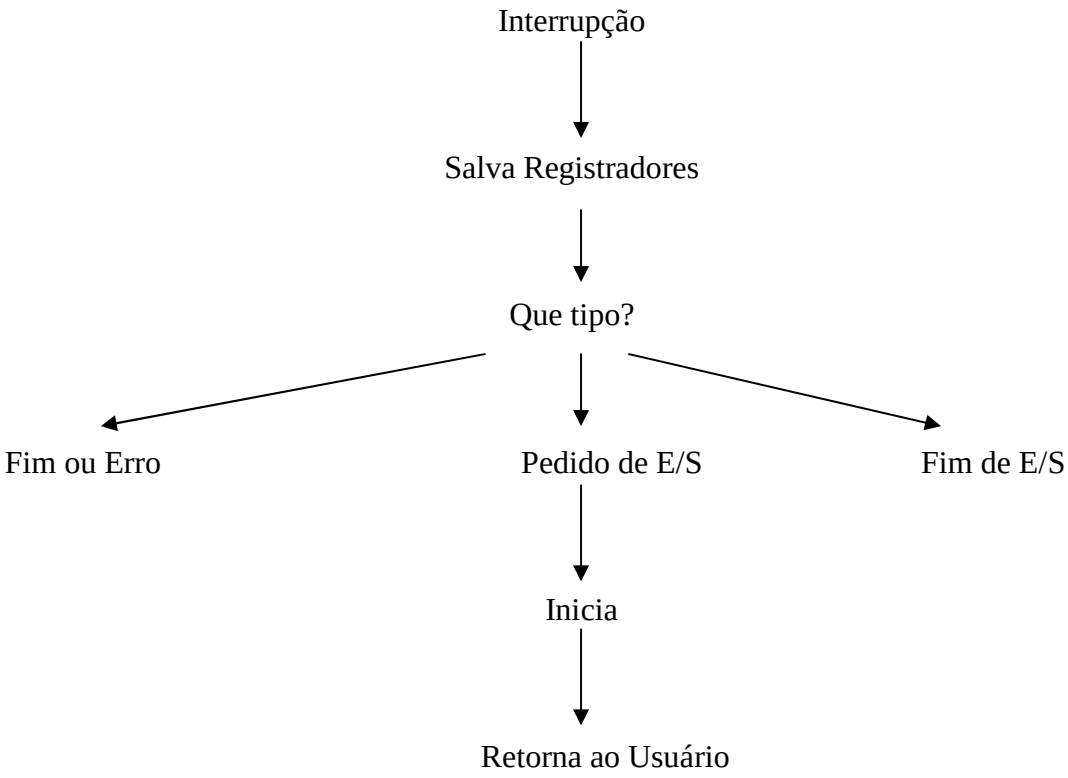


Figura 2.2. Fluxo geral do sistema operacional

2.4. Resumo

O sistema operacional oferece diversos serviços aos usuários. Num nível mais baixo, existem as chamadas ao sistema, que fornecem as funções básicas e permitem a um programa em execução fazer pedidos diretamente ao sistema operacional. Num nível mais alto, existem os programas utilitários. Entre eles se destaca o **interpretador de comandos**. O interpretador fornece ao usuário um meio mais amigável de requisitar serviços ao sistema. Neste nível um pedido satisfeito pelo interpretador de comandos ou por algum utilitário é traduzido numa sequência de chamadas ao sistema.

Leitura Obrigatória: Capítulo 1 do livro “Sistemas Operacionais Modernos”, 2ª edição, TANEMBAUM, A.

Capitulo 3

Sistema de Arquivos

É através do sistema de arquivos que o usuário mais nota a presença do sistema operacional. Os dados e programas são armazenados em arquivos. O armazenamento de informações possibilita a recuperação, reutilização e modificação nos dados e nos programas. Os computadores podem armazenar informações em vários dispositivos físicos diferentes, tais como fitas magnéticas, discos magnéticos e ópticos. Neste capítulo, apresentamos o conceito de arquivos e suas formas de armazenamento. Discutiremos os métodos de acesso e de armazenamento de arquivos no dispositivo físico. E ainda, abordaremos as estruturas de diretórios que mantêm uma organização no acesso, e as formas de proteção a arquivos.

3.1. Conceito de Arquivos

Arquivos (*files*) podem ser definidos como uma unidade lógica de armazenamento de informação destinada a abstrair as propriedades físicas dos meios de armazenamento Ou ainda, uma sequência de registros cujo significado é definido pelo seu criador.

Um arquivo é referenciado por seu nome e tem propriedades tais como tipo, tempo de criação, tamanho, nome do proprietário entre outras. Essas informações ficam armazenadas em um **diretório,** que é uma tabela de símbolos que permite identificar tais informações. As estruturas de diretórios serão vistas com mais detalhes adiante

Diferentes tipos de informação podem ser armazenados em um **arquivo** (programas fonte e objeto, dados). Arquivos podem ser numéricos, alfabéticos ou alfanuméricos. Um arquivo tem certa estrutura definida de acordo com seu uso. Por exemplo, um arquivo texto e uma sequência de caracteres organizada em linhas e possivelmente em páginas.

Se o sistema operacional reconhece a estrutura de um arquivo, ele pode então operá-lo de maneira satisfatória. A desvantagem neste caso é que o sistema operacional pode se tornar muito grande, pois haverá a necessidade de um trecho de código para cada tipo de estrutura diferente. No outro extremo, o sistema operacional não fornece nenhum suporte aos tipos de arquivo. Essa abordagem foi adotada no sistema operacional UNIX.

Um arquivo é um tipo abstrato de dados. Para defini-lo propriamente, é necessário considerar, também, as operações que podem ser realizadas sobre os mesmos. Essas operações são realizadas através de chamadas ao sistema operacional (ver capítulo 2) e são:

- **Criação:** São necessários dois passos para se criar um arquivo: encontrar um espaço para ele no dispositivo de armazenamento e colocar a entrada do arquivo no diretório informando seu nome e sua localização no dispositivo.

- **Escrita:** A escrita é feita através de uma chamada ao sistema especificando o nome do arquivo e a informação a ser escrita.

- **Leitura:** A leitura é realizada por unia chamada ao sistema especificando o nome do arquivo e a localização onde o bloco lido será colocado.

- **Reposicionamento para o início:** É buscada no diretório a entrada associada ao arquivo e a sua posição corrente é simplesmente colocada no início do arquivo.

- **Apagar:** O diretório é pesquisado e quando a entrada associada ao arquivo é encontrada, é liberado todo o espaço destinado ao arquivo e invalidada sua entrada no diretório.

3.2. Armazenamento

As formas físicas mais comuns que os sistemas de computadores utilizam para armazenar informações são meios **magnéticos** (discos, fitas, etc) **e ópticos** (CD, DVDs, Blu-rays, etc.)**.** Cada um desses dispositivos tem suas próprias características e organização física.

A vantagem da utilização de meios magnéticos é a simplicidade, porém algumas vezes é ineficiente. No caso da fita, por exemplo, quando há arquivos muito grandes pode ser necessário o armazenamento em várias fitas.

A fita possui um diretório para determinar quais são os arquivos contidos nela e a localização de cada um deles. Podem ser guardadas, também, outras Informações adicionais, tal como o tamanho do arquivo. O diretório normalmente é colocado no inicio da fita para aumentar a velocidade de acesso. Pela natureza física das fitas magnéticas pode-se perder muito tempo quando arquivos armazenados distantes um do outro são realizados acessos alternados, pois a fita possui acesso sequencial.

Fisicamente os discos são relativamente simples, como ilustrado na Figura 3 1. Eles possuem duas superfícies recobertas com um material magnético, similar ao das fitas magnéticas. Os discos são divididos em **trilhas** que variam de tamanho segundo o *disk driver* Cada trilha é dividida em **setores.** Um setor é a menor unidade de informação que pode ser lida ou escrita em um disco. Um acesso ao disco então deve especificar superfície trilha e setor. Os discos como as fitas, também possuem um diretório com informações sobre cada arquivo armazenado.

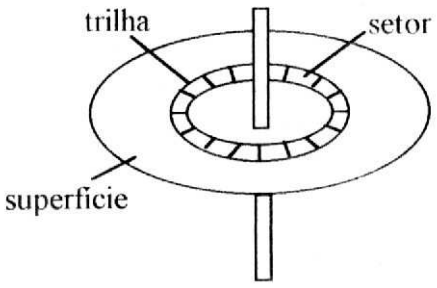


Figura 3.1 Características físicas de um disco

3.3. Gerenciamento de Espaço em Disco

O sistema de arquivos deve ser capaz de controlar a área de espaço livre nos discos, utilizar métodos de acesso às informações armazenadas e métodos de alocação que sejam convenientes e eficientes.

Uma vez que os discos possuem uma quantidade limitada de espaço, é necessário reutilizar os espaços liberados pelos arquivos que foram apagados Para manter informações sobre os espaços livres em disco, o sistema operacional tem uma **lista de espaços livres.**

As informações armazenadas em arquivos devem ser buscadas e colocadas dentro da memória do computador para serem utilizadas. Os métodos de acesso às informações de um arquivo podem ser feitos de modo **sequencial ou direto.** Alguns sistemas fornecem somente um dos métodos, outros oferecem os dois. Esta escolha é uma decisão de projeto.

Por permitir acesso direto aos arquivos, os discos possibilitam flexibilidade na implementação de arquivos. Muitos arquivos podem ser armazenados em um disco. Os três principais métodos de alocação de espaço em disco que são utilizados são o **contíguo, o encadeado** *(linked)* e o **indexado.**

3.3.1. Lista de Espaços Livres

A **lista de espaços livres** *(free space list)* registra o endereço de todos os blocos que estão livres no disco. Para criar um arquivo, faz-se uma busca a essa lista para procurar a quantidade de blocos necessária e alocá-la ao novo arquivo. O endereço desses blocos então é retirado da lista de espaços livres. Quando um arquivo é apagado, seu espaço é adicionado à lista.

Frequentemente, a lista de espaços livres é implementada como um vetor de *bits*. Cada bloco é representado por um *bit.* Se o *bit* estiver desativado (0) o bloco está livre. Por exemplo, na sequência:

00111011...

Os blocos 0, 1 e 5 estão livres. As vantagens desta implementação são a simplicidade e a eficiência para encontrar *n* blocos livres e consecutivos no disco. A desvantagem é um *overhead* em termos de ocupação de espaço de disco extra para armazenar o mapa de bits.

Uma outra maneira de implementação da lista é encadear todos os blocos livres, mantendo o ponteiro do primeiro bloco livre na lista. Este bloco contém um ponteiro para o próximo bloco e assim por diante. Para atravessar a lista devemos ler cada bloco. Isso torna este esquema pouco eficiente, pois requer um tempo substancial de E/S.

Podemos ainda, armazenar o endereço dos *n* blocos livres no primeiro bloco livre, e colocar o ponteiro deste primeiro bloco na lista. Se houver mais blocos livres do que o número que pode ser armazenado em um bloco, a última posição do bloco é o endereço de um outro bloco contendo os ponteiros para outros *n* blocos livres. Nesta implementação podemos encontrar blocos livres mais rapidamente.

Existe outro esquema que pode tirar vantagem do fato de que, geralmente diversos blocos contíguos podem ser alocados e liberados simultaneamente. Dessa forma pode-se manter uma lista de endereços livres, guardando o endereço do primeiro bloco livre e o número *n* de blocos contíguos livres. Cada entrada, na lista então consiste de um endereço e um contador.

Os discos permitem acesso direto aos arquivos. Assim, necessitamos de métodos para alocar os arquivos nos discos que utilizem de modo apropriado o espaço do disco e permitam acesso rápido aos arquivos.

3.3.2. Métodos de Acesso

Os acessos aos arquivos tanto podem ser sequenciais quanto diretos. A seguir descreveremos estes dois métodos de acesso.

Acesso Sequencial

O acesso sequencial é o modo de acesso de arquivos mais comum. A informação é buscada em ordem, uma posição após a outra. Após um registro avança-se o ponteiro para o próximo registro no arquivo.

O grande volume de operações em um arquivo são leituras e escritas. Tal arquivo pode ser reposicionado, e em alguns sistemas um programa pode ser capaz de deslocar *n* registros para frente ou para trás, por algum valor de *n* inteiro como mostra a figura 3.2 a seguir.

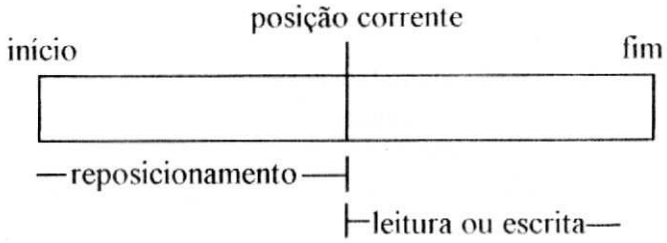


Figura 3.2 Arquivo de acesso sequencial.

Acesso Direto

No acesso direto o arquivo é visto como uma sequência numerada de blocos. Um bloco é geralmente uma quantidade de informação de tamanho fixo, definida pelo sistema operacional. O acesso direto não tem restrições na ordem de acesso a cada bloco. Dessa forma, qualquer bloco pode ser lido ou escrito aleatoriamente. O método de acesso direto é bastante utilizado para acesso imediato a grandes quantidades de informação.

3.3.3. Métodos de Alocação

Alocação Contígua

O método de **alocação contígua** *(contiguos allocation)* requer que cada arquivo ocupe um conjunto de endereços contíguos no disco. A alocação contígua é definida pelo endereço de disco do primeiro bloco e o tamanho do arquivo. Se um arquivo possui tamanho *n* e começa na localização *b*, então ele ocupa os blocos *b*, *b*+1, *b*+2, ... *b* + *n*-1. Essas informações são armazenadas no diretório. A figura 3.3 a seguir ilustra esta forma de alocação.

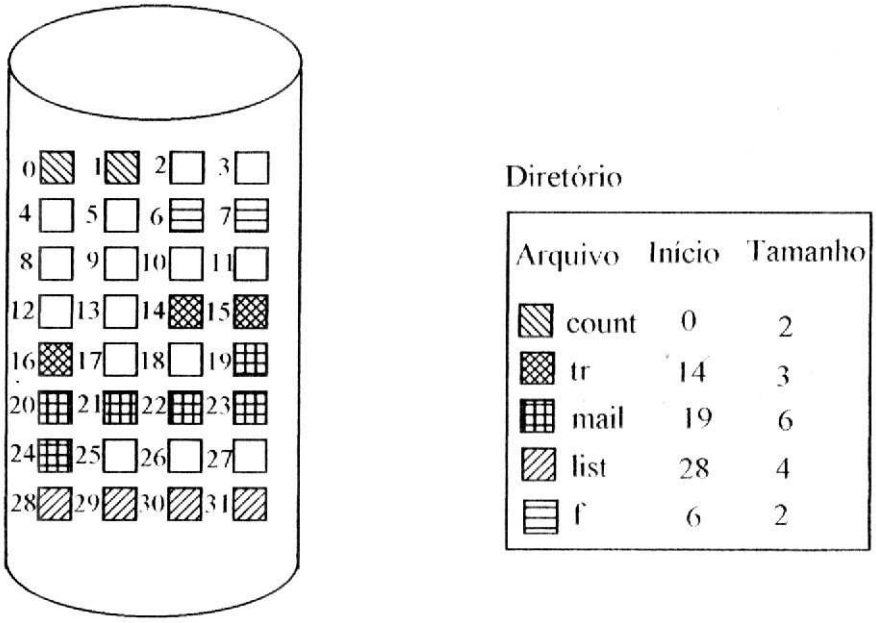


Figura 3.3. Alocação contígua.

Os acessos aos arquivos são razoavelmente fáceis e podem ser feitos sequencialmente ou diretamente. No acesso sequencial, o sistema de arquivo conhece o último endereço de acesso e quando necessário, lê o próximo bloco. No acesso direto a um bloco *i* de um arquivo que começa num bloco *b*, podemos imediatamente realizar o acesso ao bloco *b + i*.

Uma das dificuldades com alocação contígua é encontrar espaço livre para um novo arquivo. Essa decisão depende da implementação da lista de espaços livres. Outra dificuldade é saber o tamanho do arquivo na hora de ser criado. É trivial em uma cópia, mas difícil de estimar num arquivo novo que esteja sendo gerado.

Uma vez que a lista de espaços livres é definida podemos decidir como encontrar espaço para um arquivo alocado contiguamente. Para um arquivo a ser criado de comprimento de *n* setores, devemos buscar na lista por *n* setores livres que sejam contíguos. Se a lista de espaços livres for implementada como um vetor de bits, devemos encontrar *n* bits '0's em uma linha. Para uma lista de espaços livres implementada com endereços e contadores, devemos encontrar um contador com pelo menos *n*. Este problema pode ser visto como uma instância particular do problema geral de **alocação de memória dinâmica**.

O problema de alocação de memória dinâmica trata de satisfazer o pedido de tamanho *n* de uma lista de espaços livres. As soluções padrão são ***first-fit***, ***best-fit*** e ***worst-fit***. A solução *first-fit* encontra o primeiro espaço livre que suficiente para atender ao pedido. A solução *best-fit* encontra o menor espaço livre que seja grande o suficiente para atender ao pedido. *Worst-fit* utiliza o maior espaço livre disponível.

Estudos com relação à eficiência destes algoritmos mostram que o algoritmo *first-fit* é mais rápido que o *best-fit*, pois o algoritmo *best-fit* necessita realizar uma

busca pela lista inteira. A desvantagem do algoritmo *first-fit* é a fragmentação que pode ser maior que no algoritmo *best-fit.*

A alocação contígua provoca **fragmentação externa**. A fragmentação existe quando há espaço total de disco suficiente para satisfazer um pedido, mas este espaço não é contíguo, isto é, a memória do dispositivo está fragmentada em um grande número de espaços pequenos.

Para solucionar o problema da fragmentação externa pode ser utilizada a **técnica de compactação**. Ela consiste basicamente em copiar o sistema de arquivos inteiro para outro disco ou fita e depois copiá-lo de volta ao disco, deixando novamente um espaço contíguo maior. O custo desta compactação é o tempo, que pode ser relevante.

Alocação Encadeada

O método de **alocação encadeada** *(linked allocation)* soluciona o problema da necessidade de estimar o tamanho de um arquivo no armazenamento contíguo. Neste método, cada arquivo é uma lista encadeada de blocos de disco. Os blocos do disco podem estar espalhados em qualquer lugar. O diretório contém um ponteiro para o primeiro e o último bloco do arquivo. E ainda, cada bloco possui um ponteiro para o próximo bloco. A figura 3.4 a seguir mostra este método de alocação.

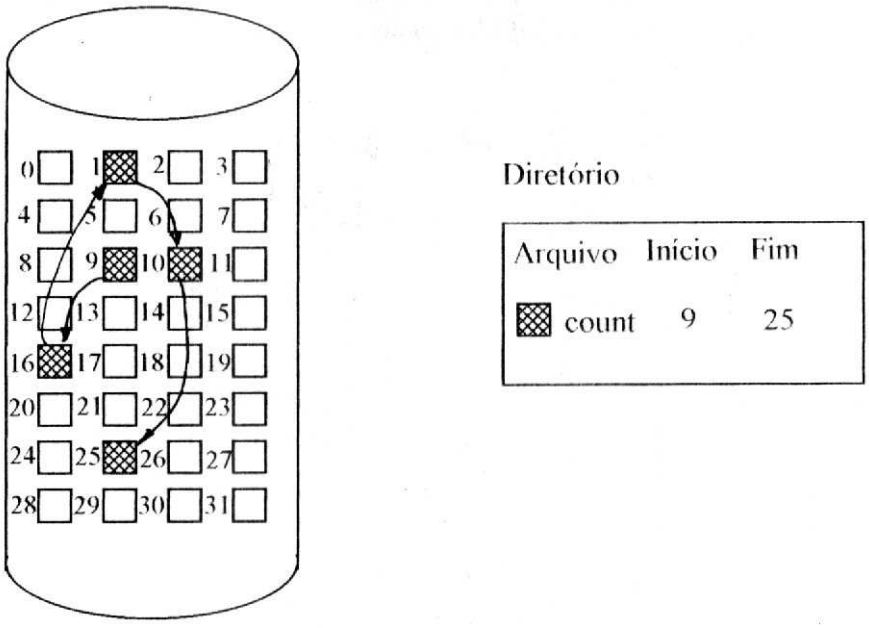


Figura 3.4 Alocação encadeada.

A criação de um arquivo é fácil, basta somente criar uma nova entrada no diretório do dispositivo. O ponteiro do início do arquivo deve ser iniciado com valor nulo, para significar arquivo vazio. Na medida em que o arquivo vai sendo escrito, o endereço do primeiro bloco é retirado da lista de espaços livres e colocado no diretório. Os ponteiros dos blocos seguintes vão sendo retirados da lista de espaços livres e encadeados ao final do arquivo. Para ler um arquivo, lemos os blocos seguindo os

ponteiros. Com o método de alocação encadeada não é necessário saber o tamanho do arquivo. Além disso, este método não provoca fragmentação externa.

Uma desvantagem na alocação encadeada é somente permitir o acesso sequencial em cada arquivo. Para encontrar o bloco *i* de um arquivo, devemos começar no inicio daquele arquivo e seguir os ponteiros até o i-ésimo bloco. Outras desvantagens são: a perda do espaço ocupado pelos ponteiros e a confiabilidade. Se perdermos um ponteiro todo o restante do arquivo após o ponteiro ficará perdido.

Alocação Indexada

O método de **alocação indexada (***indexed allocation*) soluciona os problemas de declaração do tamanho do arquivo e de fragmentação externa, relativos a alocação contígua, e do acesso sequencial e dos ponteiros espalhados pelo disco, relativos a alocação encadeada. A alocação indexada resolve este problema colocando todos os ponteiros juntos em um local chamado **bloco de índices (***index block*).

Cada arquivo tem o seu próprio bloco de índices, que é um de endereços de blocos de disco. A i-ésima entrada no bloco de índices aponta para o i-ésimo bloco do arquivo. No diretório é armazenado o endereço do bloco de índices, como pode ser visto na figura 3.5 a seguir.

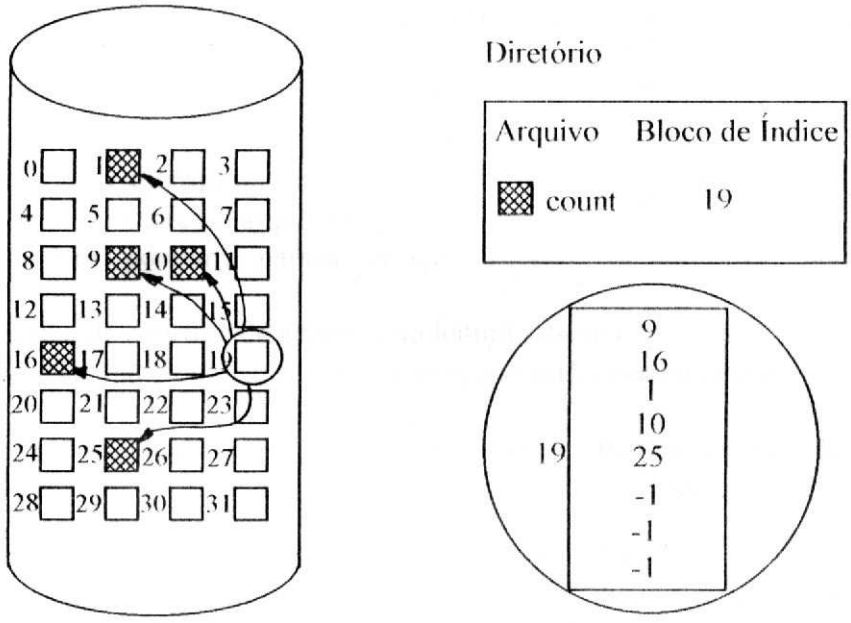


Figura 3.5 Alocação indexada.

Quando um arquivo é criado, todos os ponteiros, no bloco de índices são iniciados com valor nulo. Quando o i-ésimo bloco é escrito, é removido um bloco da lista de espaços livres e seu endereço é colocado na i-ésima entrada do bloco de índices.

A alocação indexada suporta o acesso direto sem sofrer de fragmentação externa. Qualquer bloco livre em qualquer lugar do disco pode satisfazer um pedido por mais memória. Porém, este método de alocação sofre de perda de espaço. Geralmente,

o *overhead* no armazenamento dos ponteiros no bloco de índices é maior do que na alocação encadeada. Muitos arquivos são pequenos, Dependendo do tamanho do bloco de índices a perda de espaço pode ser um problema.

Neste ponto a questão é quão grande deveria ser um bloco de índices. Ele deveria ser o menor possível, porém se for muito pequeno não será suficiente para armazenar os ponteiros para um arquivo maior. Um bloco de índices normalmente é do tamanho de um setor de disco. Para arquivos grandes que necessitam de mais de um setor, os blocos de índice usam ou uma alocação encadeada dos blocos de índices ou blocos de índices que apontam para outros blocos de índices.

Para que o sistema operacional seja capaz de recuperar dados rapidamente de um sistema de memória de armazenamento secundário, é necessária a utilização de um sistema de endereçamento, denominado genericamente de formatação. A formatação organiza trilhas e setores do disco em regiões onde os dados são, de fato, gravados; o tamanho destas regiões varia segundo o processo de formatação utilizado.

Um sistema operacional possui uma parte chamada de sistema de arquivos, que implementa diversas funcionalidades da gerência de arquivos. Dentre estas funções, é mantida uma tabela na memória, Na prática, os métodos mais empregados para a referência a arquivos são: o uso de tabelas de alocação de arquivos (FAT), para alocação por lista encadeada e o uso de *i-nodes*, que são estruturas indexadas.

Tabela de alocação de arquivos

A FAT é uma tabela que permanece carregada na memória principal contendo as referências aos clusters utilizados para armazenar os dados do início de cada arquivo. Embora ainda seja necessário seguir o encadeamento para encontrar um dado deslocamento dentro do arquivo, o encadeamento permanece inteiramente na memória, podendo ser seguido sem fazer qualquer referência ao disco

Toda a área de arquivos é divida em clusters. Arquivos são alocados nessa área um cluster de cada vez, mas não necessariamente em clusters adjacentes. Utilizamos uma tabela de alocação de arquivos (FAT- *file allocation table*) para encadear todos os clusters de um arquivo.

Para cada cluster na área de arquivos existe uma entrada na FAT, contendo um ponteiro, que nada mais é do que um endereço de cluster. Desta forma, um arquivo é representado por uma cadeia de entradas na FAT, cada entrada apontando para a próxima entrada na cadeia. No exemplo ilustrado na figura 3.6, o arquivo identificado como “arquivo.txt” tem seu início no cluster 0003, conforme entrada na FAT, e ao acessar este cluster, além dos dados armazenados em seus setores correspondentes, é encontrada uma referência ao próximo cluster que armazena a última parte do arquivo. Uma referência especial é reservada para o EOF (‘*End of File*’), que no exemplo é a string ‘FFFF’, e os clusters livres estão representados a seguir pela string ‘0000’.

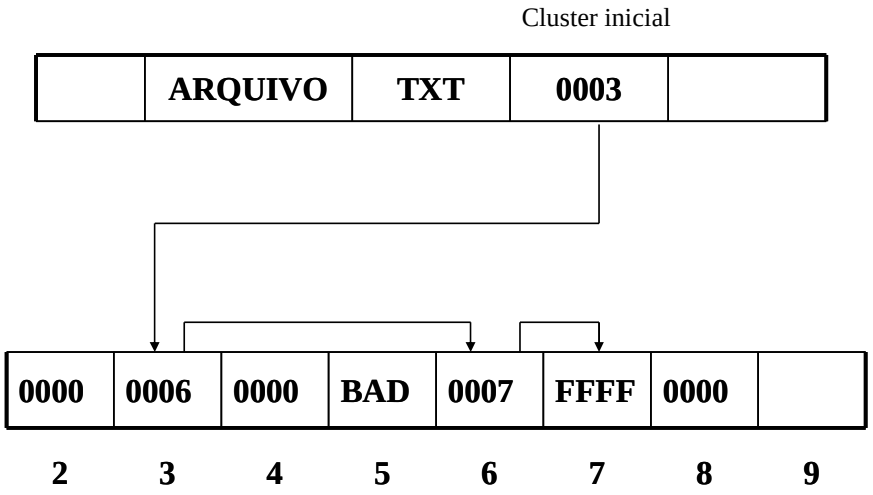


Figura 3.6. Exemplo de FAT

I-nodes

A cada arquivo é associado uma estrutura de dados chamada *i-node*, que relaciona os atributos e os endereços em disco dos blocos de arquivo. Dado o *i-node*, é possível encontrar todos os blocos do arquivo, como ilustrado na figura 3.7. A grande vantagem deste método é que o *i-node* é carregado na memória somente quando o arquivo correspondente se encontrar aberto.

Para resolver a limitação dos *i-nodes* de um arquivo ser maior do que o número de endereços disponíveis no *i-node* seria reservar o último endereço não para um bloco de dados, mas para o endereço de um bloco contendo mais endereços de blocos de disco.

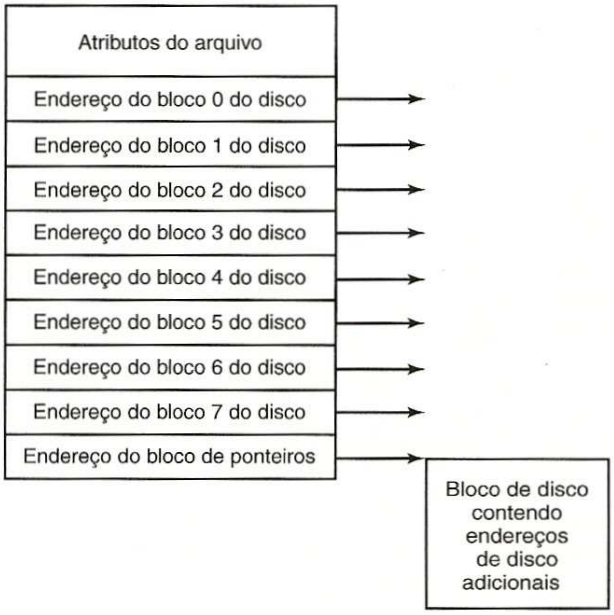


Figura 3.7. Exemplo da estrutura de um *i-node*

3.4. Sistema de Diretório

Os arquivos são representados por entradas no diretório do dispositivo. O diretório armazena informações tais como nome, localização, tamanho, tipo, para todos os arquivos daquele dispositivo.

Quando existe um aumento expressivo da quantidade de memória e do número de usuários no sistema do computador, é necessário impor uma estrutura de diretório no sistema de arquivos. A estrutura de diretório é um meio de organizar os muitos arquivos presentes no sistema.

No diretório são armazenados dois tipos de informação. A primeira está relacionada com o dispositivo físico (a localização do arquivo seu tamanho e modo de alocação). A segunda, por sua vez, está relacionada à organização lógica dos arquivos (nome, tipo, proprietário, códigos de proteção),

As informações mantidas para cada arquivo nos diretórios variam de um sistema operacional para outro sistema operacional, Algumas das informações que podem existir são listadas a seguir.

- **Nome do arquivo**: O nome simbólico de um arquivo.
- **Tipo do arquivo:** Para aqueles sistemas que suportam tipos diferentes de arquivos.
- **Localização**: Localização do arquivo no dispositivo.
- **Tamanho**: o tamanho corrente do arquivo e o tamanho máximo.
- **Posição corrente**: Um ponteiro para a posição corrente de leitura ou escrita no arquivo.
- **Proteção**: Informação de controle de acesso para leitura, escrita, execução.
- **Contabilidade de uso**: Indica o número de processos que estão correntemente usando o arquivo.
- **Tempo, data e identificação do processo**: Estas informações podem ser mantidas para criação, última modificação e último uso. Podem ser úteis para proteção e monitoração de uso.

Muitas estruturas de diretório diferentes têm sido utilizadas. O diretório é essencialmente uma tabela de símbolos. O sistema operacional utiliza o nome do arquivo simbólico para achar o arquivo. Quando considerarmos uma estrutura de diretório em particular, devemos ler em mente as operações que podem ser realizadas no diretório. Essas operações são: busca, criação, apagar, listar seu conteúdo e fazer cópias para *backup*.

Diretório de um único Nível

A estrutura de **diretório de um único nível** é a mais simples. Todos os arquivos estão contidos no mesmo diretório. É fácil de dar suporte e entender. Este tipo de diretório tem a limitação de que todos os arquivos devem ter nomes distintos. A figura 3.8 ilustra esta estrutura.

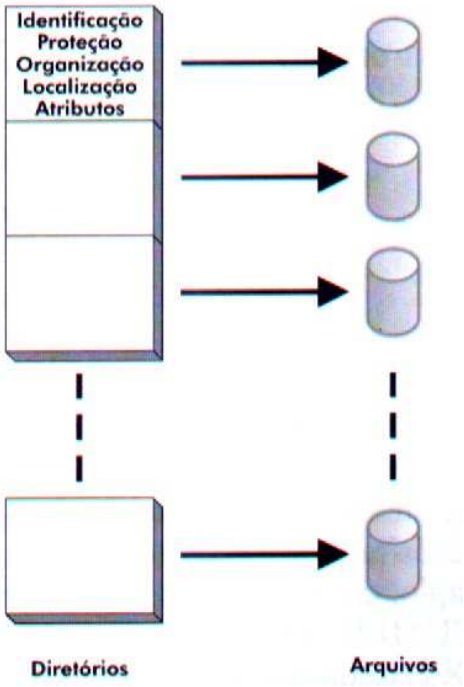


Figura 3.8. Diretório de um único nível.

A desvantagem desta estrutura é a possibilidade de confusão entre os nomes de arquivos de usuários diferentes, pois todos os usuários compartilham o diretório.

Diretório em Dois Níveis

Uma solução possível para o problema do diretório de um único nível é criar um diretório para cada usuário. Este diretório é lógico, uma vez que todos os arquivos estão fisicamente no mesmo dispositivo.

Na estrutura de diretório de dois níveis, cada usuário tem seu próprio diretório de arquivo de usuário *(user file directory – UDF).* Cada diretório de usuário tem uma estrutura similar. Quando um usuário entra no sistema, o diretório de arquivo mestre *(master file directory - MFD)* do sistema é pesquisado. O MFD é indexado pelo nome do usuário ou um número de contabilidade, Cada entrada aponta para o diretório de um usuário. Quando um usuário se refere a um arquivo em particular, somente o seu diretório é pesquisado. A figura 3.9 ilustra esta estrutura.

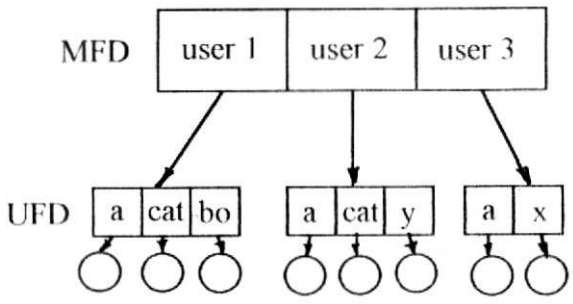


Figura 3.9 Diretório em dois níveis.

Esta estrutura isola um usuário de outro. A desvantagem da estrutura em dois níveis existe quando um usuário quer utilizar arquivos de outros usuários. Alguns sistemas não permitem este acesso. Se o acesso é permitido, ele é feito através do nome do usuário e do nome do arquivo, que definem o nome do caminho *(path name).* Todo arquivo no sistema tem um único *path name*; por exemplo, se o usuário 1 deseja realizar um acesso ao seu próprio arquivo '*cat'* basta referenciá-lo. Para se referenciar ao arquivo '*cat'* do usuário 2 ele deve indicar o caminho como /user2/cat.

Diretório Estruturado em Árvore

O diretório de dois níveis pode ser visto como uma árvore com profundidade dois. É natural estender esta estrutura para uma árvore arbitrária, que são os **diretórios estruturados em árvore.** Nesta estrutura existe um diretório raiz da árvore. Os nós intermediários da árvore são os diretórios dos usuários, que podem ainda criar seus próprios subdiretórios e organizar seus arquivos. Esta estrutura pode ser vista na figura 3.10.

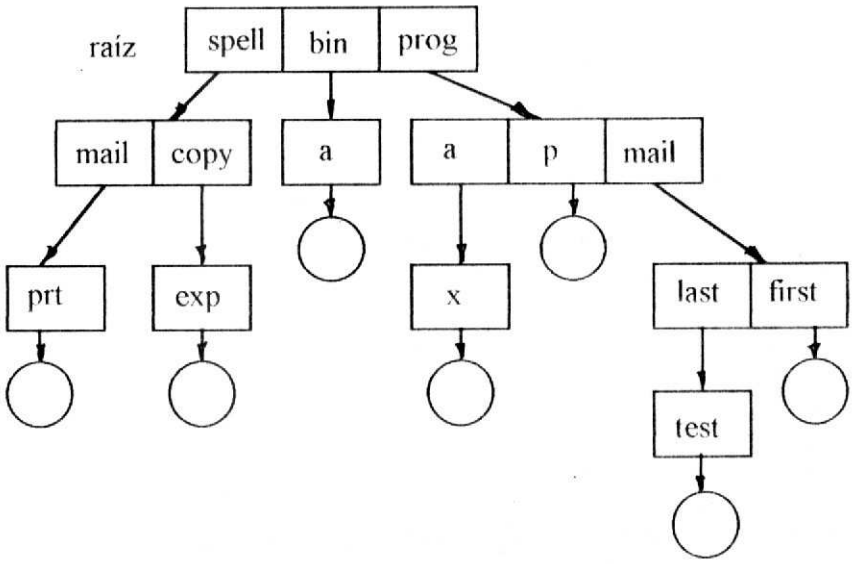


Figura 3.10. Diretório em árvore.

Todo o arquivo no sistema tem um único nome considerando o seu *path name.* O *path name* é o caminho da raiz através de todos os subdiretórios ao arquivo especificado. A sequência de diretórios pesquisada quando um arquivo é buscado é chamada de **caminho de busca** *(search path).* O *path name* pode ser apresentado de duas maneiras diferentes:

- **Completo**: Define um caminho da raiz ao arquivo.
- **Relativo**: Define um caminho do diretório corrente ao arquivo.

Os arquivos podem ser facilmente compartilhados, por exemplo, um usuário pode criar um subdiretório contendo os arquivos que serão compartilhados com outros usuários. Um destes usuários pode fazer o acesso aos arquivos compartilhados especificando o *path name* dos arquivos.

A maneira de se apagar um diretório estruturado em árvore é uma política de decisão interessante. Se um diretório está vazio ele pode ser apagado. Se ele não estiver vazio podemos decidir entre duas abordagens. A primeira é só apagar o diretório se estiver vazio. Isso implica em apagar todos os arquivos e subdiretórios contidos nele primeiro. A segunda abordagem é assumir que quando é pedido para se apagar um diretório, deva-se apagar também todos os seus arquivos e subdiretórios. A escolha da forma de implementação é uma decisão de projeto e ambas são utilizadas nos sistemas.

Diretório em Grafo Acíclico

Um diretório em grafo acíclico permite que subdiretórios e arquivos sejam **compartilhados,** ao contrário da estrutura em árvore que não permite o compartilhamento explícito. Um grafo acíclico é uma generalização do esquema de diretório estruturado em árvore e não contém ciclos. A figura 3.11 ilustra esta estrutura.

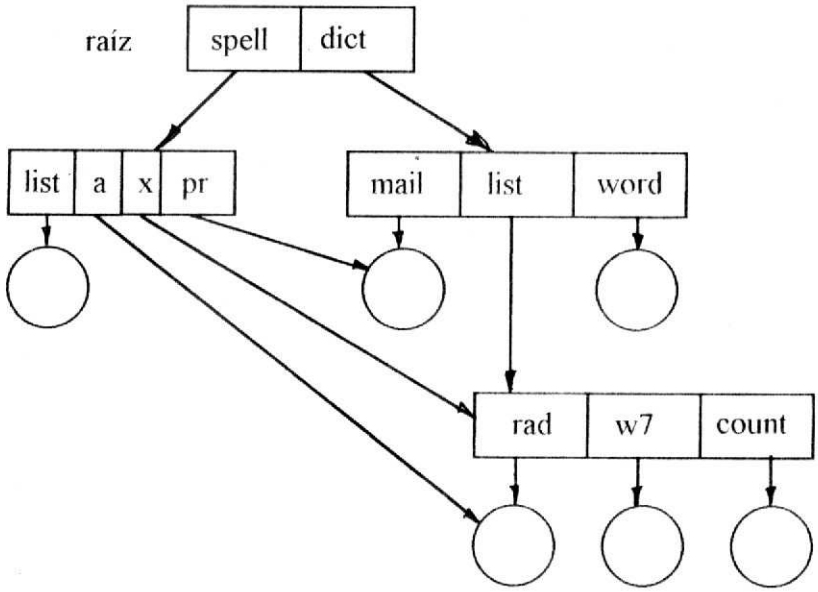


Figura 3.11 Diretório em grafo acíclico

O compartilhamento de arquivos ou diretórios pode ser implementado de diversas formas. A maneira mais comum é criar uma entrada de diretório nova chamada *link.* Um *link* é um ponteiro para outro subdiretório ou arquivo. Uma outra implementação é duplicar as informações em ambos os diretórios. O problema nesta abordagem é manter a consistência nas informações se o arquivo for modificado.

Uma estrutura de diretório de grafo acíclico é mais flexível que uma estrutura em árvore simples, mas também é mais complexa. Os problemas como buscar um determinado arquivo ou apagá-lo devem ser cuidadosamente considerados.

Um arquivo pode ter vários nomes completos. Dessa forma, nomes com caminhos diferentes podem se referenciar ao mesmo arquivo. Como cada arquivo tem mais de uma trajetória de busca, a eliminação de um arquivo pode ser feita de várias maneiras. Quando são utilizados *links,* apagar um arquivo implica na retirada do *link,* não afetando o arquivo. Esse só é removido quando forem retirados todos *os links.*

Uma outra possibilidade é preservar o arquivo até que todas as referências a ele tenham sido apagadas. Poderia haver uma lista de referências ao arquivo (entradas no diretório *ou links* simbólicos). Quando uma referência fosse feita, ela seria adicionada à lista. Quando um *link* ou entrada fosse apagado, a referência seria retirada da lista. O arquivo só seria apagado quando a lista estivesse vazia. O tamanho da lista pode crescer e tornar-se um problema, mas na verdade, somente é necessário manter um contador de referências. Quando este tiver valor zero, o arquivo pode ser apagado.

3.5. Proteção de Arquivos

Num sistema de computador é importante manter a proteção das informações contra danos físicos (confiabilidade) e contra acessos impróprios (proteção). A confiabilidade geralmente é obtida realizando uma cópia dos arquivos em intervalos regulares. A proteção de arquivos é um resultado direto da habilidade de realizar acessos a eles. Sistemas que não permitem a um usuário referenciar os arquivos de outro usuário, não precisam de proteção.

Um extremo seria fornecer proteção completa, proibindo o acesso. O outro extremo é fornecer livre acesso sem nenhuma proteção. Essas abordagens são multo radicais. Na verdade, o que é necessário é um controle de acesso. Uma maneira é associar uma senha (*password*) a cada arquivo. Só podem ser feitos acessos este arquivo o usuário que conhecer a senha.

Outros mecanismos de proteção fornecem acesso controlado, limitando os tipos de acesso que podem ser feitos ao arquivo. Existem diversas operações que podem ser controladas como leitura, escrita, execução, cópia, edição, entre outras.

O controle de acesso é feito com a identidade do usuário. Vários usuários podem ter necessidade de acessos de tipos diferentes a arquivos e a diretórios. Uma **lista de acesso** (*access list*) é associada com cada arquivo e com cada diretório, especificando o nome do usuário e os tipos de acesso permitidos. Quando é feito um pedido, o sistema operacional verifica a lista de acesso, permitindo-o ou não. Uma desvantagem com este mecanismo é o tamanho da lista de acesso. Uma solução para reduzir o tamanho da lista de acesso é tentar classificar os usuários em três grupos:

- **Proprietário:** O usuário que criou o arquivo.
- **Grupo:** Um conjunto de usuários que compartilham o arquivo e necessitam de acesso similar.
- **Universo:** Todos os usuários do sistema.

Com esta classificação, somente três campos são necessários para garantir a proteção. Cada campo é uma coleção de bits que permitem ou previnem a operação associada com cada bit Por exemplo, no sistema operacional UNIX existem três campos (proprietário, grupo, universo) de três *bits* cada: ***rwx***. O *bit* ***r*** controla o acesso de leitura, o bit ***w*** controla o acesso de escrita e o bit ***x*** controla a execução. Neste esquema nove bits por arquivos são necessários para registrar a proteção da informação.

3.6. Resumo

Um arquivo é um tipo abstrato de dados definido e implementado pelo sistema operacional. O sistema operacional pode dar suporte a vários tipos de arquivos ou pode deixar para o programa de aplicação resolver este problema.

As informações podem ficar armazenadas em dispositivos físicos como fitas magnéticas ou discos. Neste capítulo abordamos as formas de armazenamento em discos magnéticos.

São necessários mecanismos para controlar a forma de armazenamento e o acesso às informações. O acesso às informações pode ser de modo sequencial ou direto

Os arquivos podem ser armazenados nos discos de três modos diferentes: contíguo, encadeado e indexado.

Os diretórios são estruturas criadas para armazenar todas as informações relativas aos arquivos e para manter uma organização sobre eles. Apresentamos as seguintes estruturas de diretório: em um único nível, em dois níveis, em árvore, em grafo acíclico e em grafo geral.

Geralmente, necessitamos proteger as informações de danos físicos ou de acessos impróprios. Isso é resolvido através de mecanismos de proteção.

Leitura Obrigatória: Capítulo 6 do livro “Sistemas Operacionais Modernos”, 2ª edição, TANEMBAUM, A.

Capitulo 4

Processos

O escalonamento da Unidade Central de Processamento (UCP ou CPU - *Central Processing Unit*) é o conceito mais relevante de sistemas operacionais multiprogramados. É através do chaveamento do processador entre os vários processos, que o sistema operacional pode tornar a máquina mais eficiente e produtiva. Neste capítulo serão introduzidos os conceitos básicos de processos, apresentados alguns problemas e algumas soluções propostas na literatura relacionadas ao escalonamento de processos.

4.1. Conceito de Multiprogramação

Sem dúvida, o conceito de multiprogramação é um dos mais importantes nos sistemas operacionais modernos. Se existirem vários programas carregados na memória ao mesmo tempo, a UCP pode ser compartilhada entre eles, aumentando a eficiência da máquina e produzindo mais resultados em menos tempo.

A idéia por trás da multiprogramação é bastante simples. Quando um programa libera a UCP, seja para realizar alguma operação de E/S ou por outro motivo, ela fica parada. Enquanto espera que o programa volte para executar, a UCP não realiza nenhum trabalho útil. Para acabar com a ociosidade deste tempo vários programas são mantidos ao mesmo tempo na memória e o sistema operacional se encarrega de escolher um deles para executar. Assim, sempre que um programa é interrompido, outro é escolhido para ser executado em seu lugar. Com isso, a UCP estará durante grande parte do tempo ocupada processando instruções de programas.

Os benefícios da multiprogramação são vários: aumento da utilização da UCP e da taxa de saída do sistema computacional, isto é, dia quantidade de trabalho realizada dentro de um intervalo de tempo *(throughput).*

Para exemplificar a utilidade da multiprogramação, imaginemos um sistema onde três programas estão carregados na memória para serem executados, como na figura 4.1. Na primeira parte da figura, é mostrada a execução sequencial dos três processos.

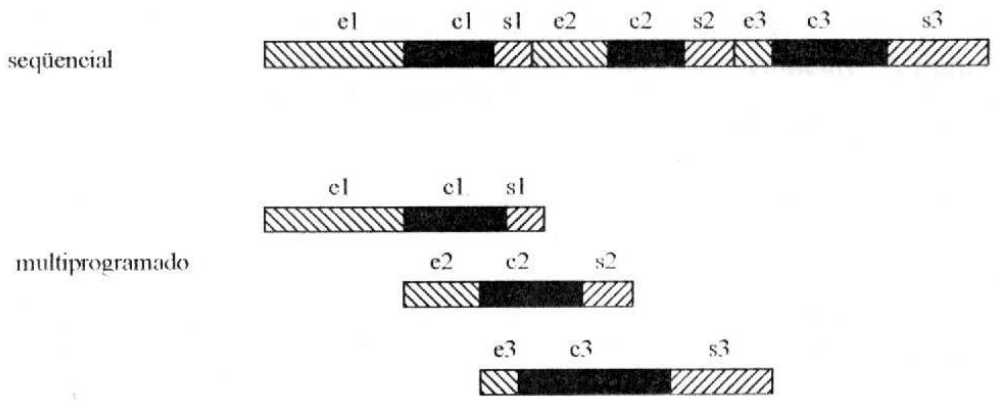


Figura 4.1 Execução sequencial e concorrente de três processos

Se o sistema for multiprogramado, como é sugerido na segunda parte da figura 4.1, menos tempo será gasto na execução dos três programas. Assim, a UCP não ficaria ociosa em nenhum momento e sua taxa de utilização aumenta para 100%. Este exemplo é um caso extremo e dificilmente acontece na prática, mas ilustra bem a idéia de multiprogramação.

4.2. Conceito de Processo

Informalmente, um processo é um programa em execução. Um programa é uma sequência de instruções executáveis, e pode conter diferentes fluxos de execução, frutos

da existência de comandos condicionais e interativos. Ele pode ser visto como uma entidade passiva. Um processo pode ser visto ainda como uma entidade ativa que descreve a execução de um dos possíveis fluxos, do programa. A execução de um processo se dá de maneira sequencial, ou seja, em qualquer instante de tempo no máximo uma instrução está sendo executada.

Como discutido anteriormente, a definição de processo como um programa em execução, apesar de adequada sob uma visão mais simplista do funcionamento de sistemas operacionais, terá de ser expandida, a fim de nos permitir considerar outros aspectos importantes do uso prático, como o funcionamento de *threads*.

Em sistemas multiprogramados, diferentes programas são formados por conjuntos de instruções, que no momento que o programa tem o direito de uso da unidade processadora, são armazenados em registradores do processador. Estes processos concorrem entre si pelo uso da UCP, e muitas vezes o conjunto de instruções que devem ser executadas ao longo de sua execução não são todas realizadas em um mesmo intervalo de tempo. Um processo pode ter que parar sua execução, dando a vez a outro processo, que executará outro conjunto de instruções que o compõe, que por sua vez pode dar a vez a outro (que pode inclusive ser o primeiro que estava sendo executado), e assim sucessivamente, até que as instruções que compõem um processo sejam, finalmente, concluídas.

Para que seja possível funcionar corretamente em um ambiente com outros processos em execução, cada processo deve possuir informações que não são somente relacionadas a si, mas também ao ambiente onde está sendo executado. Neste sentido, as informações que formam um processo podem ser classificadas em:

Contexto de software: São especificados limites e características dos recursos que podem ser alocados pelo processo, como o número máximo de arquivos que podem ser abertos simultaneamente, a prioridade de execução e o tamanho de buffers para operações de E/S. As informações do contexto de software são separadas nos grupos **identificação**, **quotas** e **privilégios**.

Contexto de hardware: armazena o conteúdo dos registradores gerais da UCP, além de registradores de uso específico, como o *program counter* (PC), *stack point* (SP) e registrador de status. Sua função é salvar o conteúdo dos registradores em uma operação de troca de contexto, para que posteriormente possa ser restaurado o mesmo conjunto de informações que o processo utilizava no momento que parou sua execução.

Espaço de endereçamento: consiste na área de memória destinada ao processo durante sua execução, aonde dados e instruções serão armazenados para serem executados. Cada processo possui sua área de endereçamento, que o S.O. deve proteger do acesso a demais processos.

Outra idéia associada à definição de processo está na necessidade de descrever a existência de várias atividades que ocorrem em paralelo dentro do sistema computacional. Num ambiente multiprogramado vários usuários podem estar executando seus programas simultaneamente, dificultando a gerência de múltiplas atividades paralelas. Dessa forma o modelo de processos é uma maneira de se

decompor este problema em componentes mais simples. Todo o *software* no computador é organizado em processos sequenciais ou apenas processos.

Com a multiprogramação, cada usuário tem a sensação de ler uma máquina (processador) só para si, o que na prática não acontece. Na verdade, o processador central se reveza entre os vários usuários e uma das tarefas do sistema operacional é tornar isto o mais transparente possível. Quando o processador muda de um processo para outro, é necessário que o sistema salve todas as informações necessárias para a retomada do processo interrompido. Dentre essas informações pode-se destacar: a identificação do processo, o seu estado, o valor do *Program Counter*, o valor dos registradores que estavam sendo usados (acumulador, de uso geral), as informações para gerência da memória (registradores de limite, endereço do início da tabela de páginas), as informações para escalonamento (prioridade, ponteiros para as filas de escalonamento), entre outras. Assim, pode-se dizer que ao conceito de processo estão associadas informações que caracterizam o seu contexto de execução. Estas informações são armazenadas numa estrutura que recebe o nome de **Bloco de Controle do Processo** (*Process Control Block - PCB)* e que deve estar armazenada na área de memória destinada ao sistema operacional, para evitar que o usuário possa realizar o acesso a ele. Os PCB's representam os processos para o sistema operacional e a sua quantidade varia com o tempo à medida que processos são e terminam dinamicamente. A figura 4.2 ilustra algumas informações armazenadas num BCP.

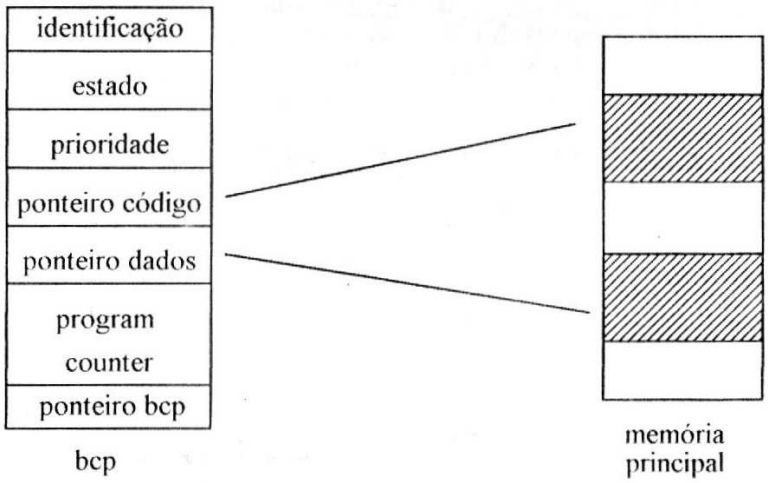


Figura 4.2 Bloco de controle.

Em sistemas multiprogramados, quando ocorre a parada da execução de um programa por qualquer razão prevista (parada para a ocorrência de uma operação de E/S, término da fatia de tempo atribuída ao processo, etc.) é dito que aconteceu uma **troca de contexto**, ou seja, as informações referentes ao processo em execução devem ser salvas, para que o processo retome sua execução futuramente do mesmo ponto que parou, e que as informações referentes ao novo processo que terá o direito de uso da UCP retomem de um ponto anterior (caso já tivesse utilizado a UCP anteriormente).

A troca de contexto é uma operação que apresenta um custo computacional que não pode ser ignorado. Entre a operação de salvar o contexto atual de um processo em seu PCB, e carregar o contexto de outro processo a ser executado, existe um intervalo de ociosidade, que não pode ser eliminado, como ilustrado na figura 4.3.

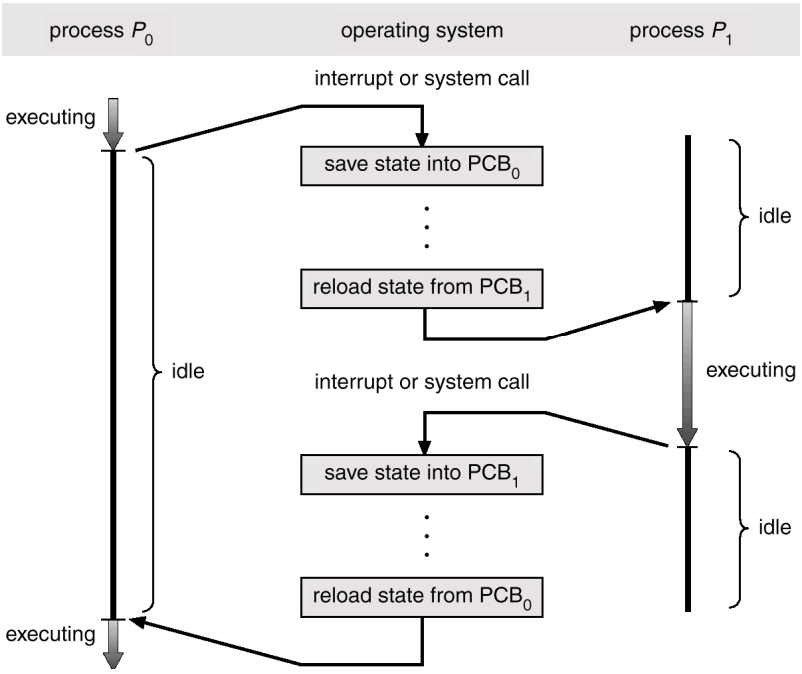


Figura 4.3. Operação de troca de contexto entre processos.

4.3. Threads

Threads são um mecanismo que aumenta a eficiência do sistema operacional reduzindo o *overhead* causado pela troca de contexto dos processos. Elas podem ser vistas como a menor unidade de execução do sistema e compartilham todos os recursos com o processo que as criou. **Cada thread possui seu próprio contexto de hardware, mas compartilha o mesmo contexto de software e espaço de endereçamento com os demais threads do processo.**

Em sistemas com *threads*, um processo com exatamente uma *thread* é equivalente a um processo clássico. Cada *thread* pertence a exatamente um único processo. Sob o ponto de vista de *threads*, processos são estáticos e apenas as *threads* são escalonadas para execução. Cada *thread* representa um fluxo de controle separado com sua pilha e seu estado da máquina. Como todos os recursos, exceto o processador são controlados pelo processo que cria a *thread*, o chaveamento da UCP entre *threads* é mais rápido e eficiente. Entretanto, o chaveamento entre *threads* de processos diferentes gastam um tempo maior, pois envolve todo o processo de troca de contexto.

Threads são eficientes para a exploração de concorrência dentro de uma aplicação e podem se comunicar usando a memória comumente compartilhada. Deve ser tomado muito cuidado no que diz respeito à sincronização a regiões críticas. O conceito e funcionamento de threads será discutido com mais detalhes em outro capítulo.

4.4. Criação de Processos

Nos sistemas mais antigos apenas o sistema operacional podia criar novos processos. Atualmente, a maioria dos sistemas permite que os usuários criem e

destruam seus próprios processos. Um grande número de programas podem se beneficiar disto, como programas de simulação, ordenação e computação científica.

Para um programa poder criar novos processos o sistema operacional deve fornecer primitivas específicas (**chamadas ao sistema**) para tal finalidade. Quando um novo processo é criado, inicialmente o seu BCP é preparado com as informações básicas. Em seguida é feita uma chamada ao sistema operacional para a sua criação. Algumas linguagens de programação oferecem suporte para o controle de processos.

4.5. Estados de um Processo

A execução de um processo começa por um ciclo de execução na UCP (*CPU-Burst*). Este ciclo é seguido por um ciclo de E/S do qual a UCP participa no início e no final. Novamente há um ciclo de UCP e outro de E/S até que o processo termine a sua execução. O tamanho dos ciclos de UCP e de E/S varia de processo para processo. Sendo assim, podemos identificar dois tipos de processos:

- Os que requisitam mais operações de E/S e têm ciclos de UCP muito pequenos;
- Os que gastam mais tempo executando instruções e têm ciclos de E/S pequenos.

Um programa associado a um processo do primeiro tipo é classificado como **I/O-*bound*** (apresenta vários *bursts* de UCP curtos) e um programa associado a um processo do segundo tipo é classificado como **CPU-*bound*** (poucos *bursts* de UCP, mas de longa duração). Programas científicos são geralmente CPU-*bound* enquanto que programas comerciais são I/O-*bound*. A figura 4.4. ilustra uma sequência de bursts típica da execução de um programa.

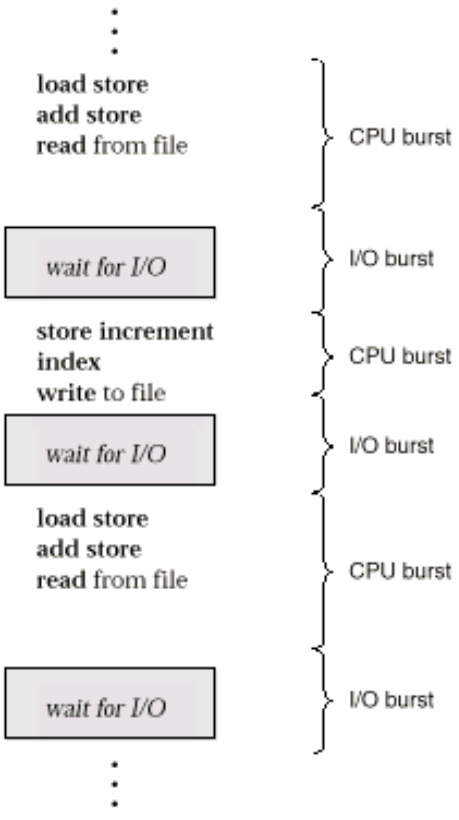


Figura 4.4. Sequência de *bursts* da execução de um programa.

Durante a execução de um programa o processo pode se encontrar em um dos estados a seguir:

- **Executando:** Diz-se que um processo está executando se ele está usando o processador. Em sistemas com uma única UCP, no máximo um processo pode estar neste estado em qualquer instante de tempo.
- **Pronto:** Um processo está no estado pronto quando espera que a UCP seja liberada pelo processo que a está usando. Este estado pode ser considerado como a entrada para que um processo novo possa competir pelo processador com outros processos, que também estejam neste estado.
- **Bloqueado:** Um processo está bloqueado quando não pode ser executado porque espera que alguma condição externa aconteça. Por exemplo, o fim de uma operação de E/S. Assim que a condição é satisfeita, o processo volta para o estado pronto.
- **Terminado:** Um processo está terminado quando a última instrução do programa foi executada.

A maneira como um processo passa de um estado para o outro é mostrada na figura 4.5 e na tabela 4.1, a seguir.

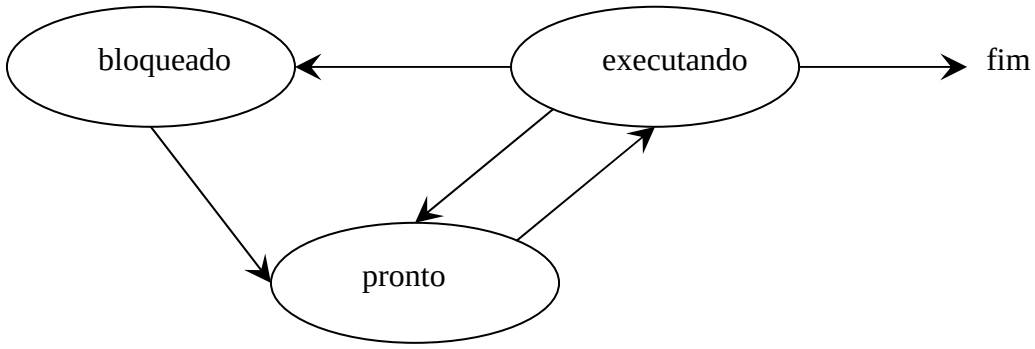


Figura 4.5. Estados de um processo.

Evento	Transição	Significado
tempo	executando → pronto	fatia de tempo do processo executando expirou.
despacho	pronto → executando	sistema operacional entrega o processador a um processo no estado pronto.
bloqueio	executando → bloqueado	processo solicitou operação de E/S.
despertar	bloqueado → pronto	evento esperado acontece.

Tabela 4.1. Transição dos estados de um processo.

4.6. O Conceito de Escalonamento

A visão que o sistema operacional tem dos processos e seus respectivos estados pode ser representada por filas formadas pelos BCP's. Assim, um processo que esteja no estado pronto é mantido numa **fila de processos prontos** *(ready queue).* A forma como os processos são colocados e retirados dessa fila será discutida a seguir e é um dos pontos mais importantes na implementação de um sistema operacional.

Quando um processo sai do estado executando e vai para o estado bloqueado é porque alguma interrupção ocorreu e alguma condição externa deve ser satisfeita. Ele é colocado na fila associada ao dispositivo que será usado durante a operação de E/S. Por exemplo, se o acesso for ao disco, o processo deverá ser colocado na fila desse dispositivo, caso ele esteja, sendo usado por algum outro processo. A figura 4.6. representa bem a visão que o sistema operacional tem dos processos.

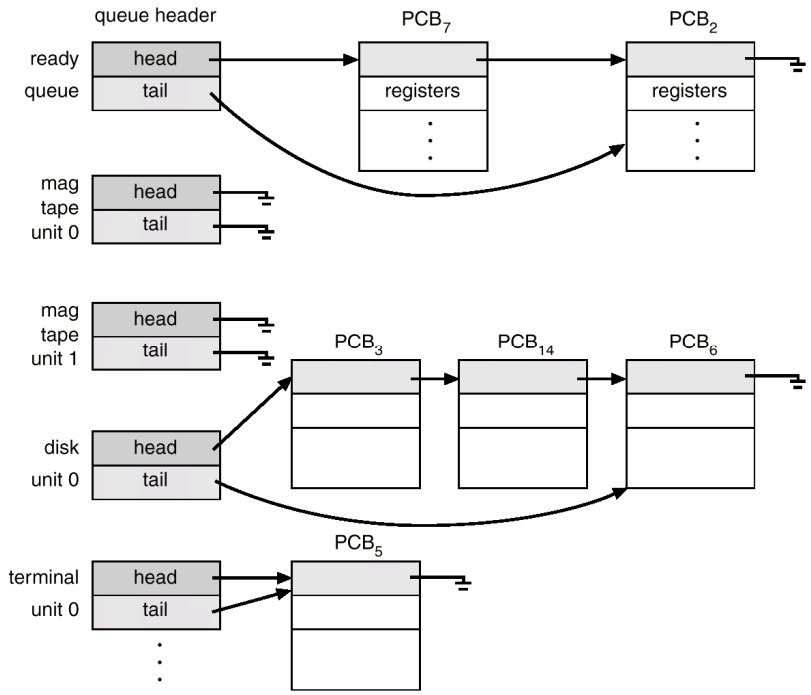


Figura 4.6. Representação das filas de processos, no sistema operacional.

De acordo com as figuras 4.5 e 4.6, um processo entra no sistema e é colocado na fila de prontos. Ele espera nesta fila até ser selecionado para executar seu primeiro ciclo de UCP, passando para o estado executando. Num determinado momento uma operação de E/S poderá ser requisitada e a execução do processo será interrompida. Ele passa então para o estado bloqueado e é colocado na fila do dispositivo que realizará a operação. Quando a operação de E/S terminar, o processo retorna ao estado pronto para mais uma vez disputar o processador com os outros processos. Em alguns sistemas, há ainda a possibilidade de um processo passar do estado executando para o estado pronto, caso haja algum limite máximo de tempo que o processo pode executar na UCP. Isto é para evitar que um processo utilize a UCP por um período de tempo muito longo, fazendo com que os outros processos que se encontram no estado pronto esperem muito. Isso acarreta o aumento do tempo total de processamento.

A escolha do processo que vai ser retirado de uma fila recebe o nome de **escalonamento** e é implementada por um componente do sistema operacional chamado de **escalonador.**

Existem dois tipos de escalonadores: de **longo prazo e de curto prazo.** O primeiro é o responsável por escolher quais programas vão ser carregados na memória e passar para o estado pronto. Esse escalonador é executado em intervalos de tempo

maiores, já que um programa só libera a área de memória quando termina a sua execução. O segundo é o responsável por selecionar qual processo vai ser retirado da fila de prontos e ser executado na UCP. A grande importância dos escalonadores de longo prazo é que eles definem o **nível de multiprogramação** do sistema, controlando o número de programas carregados na memória ao mesmo tempo. Logo, controlando a quantidade de processos que entram na fila de prontos, Usando os conceitos vistos anteriormente de *CPU-bound* e de *I/O-bound* é importante que o escalonador de longo prazo faça uma boa mistura de programas na memória. O objetivo desta mistura é manter o sistema balanceado e impedir que a fila de prontos não fique a maioria do tempo vazia ou muito cheia. Este capítulo dará ênfase ao segundo tipo de escalonadores e assume que os programas já estão carregados na memória e se encontram no estado pronto. O diagrama mostrado na figura 4.7 mostra como os escalonadores atuam no sistema.

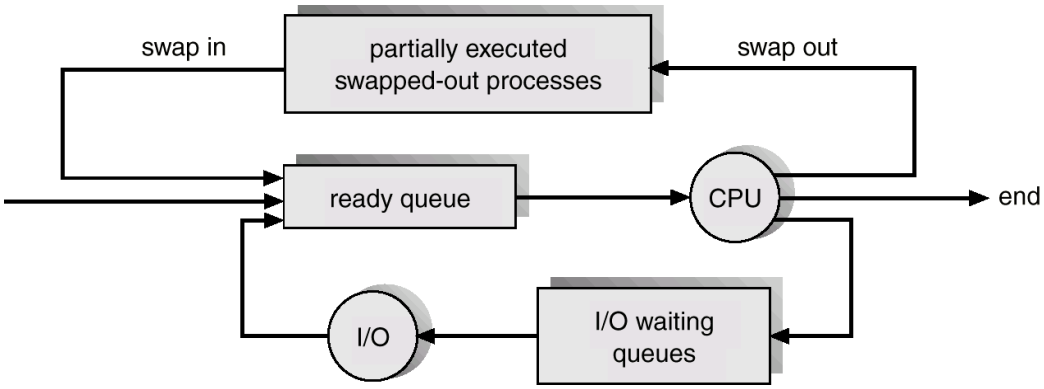


Figura 4.7. Escalonadores do sistema operacional.

Outro componente envolvido no escalonamento da UCP recebe o nome de **despachador (*dispatcher*).** Ele é o responsável por passar o controle da UCP para o processo selecionado pelo escalonador de curto prazo. Essa operação envolve o carregamento dos registradores do processador com os dados do processo selecionado e o desvio para o ponto do programa de onde a execução deve se iniciar (nem sempre é do início, dado que durante a execução, um processo pode ser suspenso, pois pode ter que esperar por algum evento). Estas informações estão guardadas no PCB do processo selecionado e o despachador deve ser o mais rápido possível.

4.7. Escalonamento de Processos

O escalonamento da UCP lida com o problema de decidir qual dos processos que se encontram na fila de prontos será o escolhido para executar. Existem várias maneiras de se implementar essa escolha, mas é importante saber que ela influencia tia eficiência do sistema operacional. Como os algoritmos possuem propriedades diferentes, há necessidade de alguns critérios para se selecionar qual o melhor algoritmo a ser usado numa situação particular. Esses critérios são apresentados a seguir:

- **Utilização da UCP:** Como a UCP é um recurso caro e influencia diretamente na eficiência da máquina, ela deve ser mantida ocupada o maior tempo possível. Em sistemas reais, a taxa de ocupação do processador varia de 40% (baixa utilização) a 90% o (alta utilização).

- ***Throughput* (taxa de saída):** Uma maneira de se medir o trabalho realizado pelo sistema é contabilizar-se a quantidade de processos que terminam num determinado intervalo de tempo.

- ***Turnaround time*:** Do ponto de vista dos processos, um critério importante é a quantidade de tempo que cada um dos processos passa dentro do sistema para executar. O *turnaround time* é contado a partir do momento em que o processo é submetido ao sistema até o instante em que a última instrução é executada. Inclui todos os intervalos gastos esperando na memória, esperando na fila de prontos, executando na UCP e realizando E/S.

- **Tempo de espera:** O escalonamento da UCP não afeta a quantidade de tempo que o processo gasta executando no processador nem a quantidade de tempo gasto realizando E/S. Esses tempos dependem do próprio processador e dos dispositivos de E/S. Por outro lado, os algoritmos de escalonamento afetam diretamente o tempo que os processos gastam esperando nas filas. Esse tempo é medido através do tempo de espera.

- **Tempo de resposta:** Em sistemas interativos (*on-line*) o *turnaround time* às vezes não é o melhor critério de medida. Para estes sistemas uma boa medida seria o intervalo de tempo entre a submissão de um pedido e a resposta obtida. Esta medida é o tempo de resposta.

Uma vez selecionado um algoritmo para implementar o escalonamento, é desejável que a taxa de utilização e a taxa de saída sejam maximizadas, que o *turnaround time*, os tempos de espera e de resposta sejam minimizados. E ainda que todos os processos recebam o processador em algum momento, para que possam ser executados. Além disso, é muito importante que o sistema operacional atenda a estes objetivos mantendo total transparência para os usuários. A quantidade máxima de programas executando concorrentemente deve ser tal que não degrade a eficiência do sistema computacional.

Os algoritmos de escalonamento podem ser divididos em duas classes: **preemptivos e não-preemptivos.** No primeiro caso, o processo que está sendo executado pela UCP pode ser interrompido e perder o controle do processador para outro processo mais prioritário. No segundo caso, o processador não pode ser retirado do processo que está sendo executado, a não ser quando o seu ciclo de UCP terminar.

O escalonamento de processos pode ser representado de forma visual através de um modelo intitulado **diagrama (ou carta) de Gannt**, que simplesmente representa a ordem de execução dos processos e o tempo de execução de cada parte de um processo. A seguir serão mostrados alguns dos mais importantes algoritmos de escalonamento de cada uma das classes.

***First- Come-First-Served* (FCFS)**

Este é o algoritmo de implementação mais simples. O processo que primeiro requisita o processador é atendido primeiro. Neste caso, a fila de prontos é implementada seguindo o modelo *First-In-First-Out* onde o primeiro a entrar na fila é o primeiro a sair. Quando um processo entra na fila de prontos o seu PCB é sempre colocado no final da fila. Quando a UCP fica livre ela é entregue ao primeiro processo da fila.

Apesar de simples de entender e implementar, a eficiência do algoritmo pode ficar bastante comprometida porque é muito dependente dos processos que formam a fila de prontos. Este é um algoritmo naturalmente não-preemptivo, já que o critério de seleção entre os processos está baseado no tempo de chegada à fila de prontos. Dessa forma, se um processo já está usando a UCP, o seu tempo de chegada tem de ser menor que qualquer outro processo que chegue depois dele.

Na figura 4.8 é mostrada a fila de prontos com quatro processos *(p1, p2, p3, p4).* A fila foi organizada de acordo com a ordem de chegada dos processos, o que significa que o primeiro processo a ser executado é o p1, depois o *p2* e assim por diante. Neste exemplo, a quantidade de tempo gasta para executar os quatro ciclos de UCP, correspondentes a cada processo, é de 24 ut (unidades de tempo) e o tempo total de espera é de 41 ut.

Processo	T. *burst*
p1	6 ut
p2	8 ut
p3	7 ut
p4	3 ut

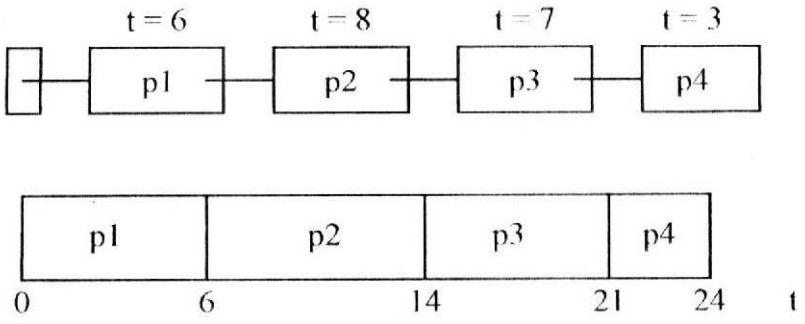
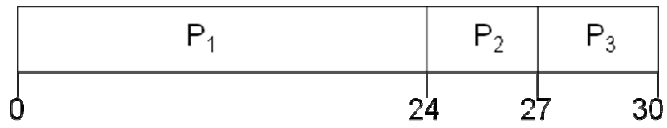


Figura 4.8 Algoritmo *First- Come-First-Served*

Uma forma muito empregada para medir a eficiência de um algoritmo de escalonamento é o tempo médio de espera, que representa uma média do tempo que cada processo teve que aguardar para ter sua execução iniciada. O exemplo a seguir ilustra como a simples mudança da ordem de chegada dos processos altera significativamente o tempo médio de espera.

Processo	T. *burst*
P1	24 ut
P2	3 ut
P3	3 ut

Suponha que os processos chegam na seguinte ordem : P1, P2, P3. A carta de Gannt a seguir ilustra a situação:



Tempo de espera para P1 = 0; P2 = 24; P3 = 27
Tempo médio de espera: (0 + 24 + 27)/3 = 17

Suponha agora que os processos cheguem na ordem: P2, P3, P1. A carta de Gannt para esta situação será:

Tempo de espera para P1 = 6; P2 = 0; P3 = 3
Tempo médio de espera: (6 + 0 + 3)/3 = 3

Como podemos observar, a simples mudança na ordem que os processos são submetidos ao sistema ocasiona uma situação melhor que caso anterior

Shortest Job First (SJF)

Este algoritmo associa a cada processo o tempo aproximado do seu próximo ciclo de UCP a ser executado. Quando o processador fica disponível ele é entregue ao processo que possui o menor próximo ciclo de UCP que foi estimado.

Este algoritmo obtém o menor tempo de espera médio. Apesar disso, a sua implementação é dificultada pelo fato de haver necessidade de se fazer uma previsão do tamanho do próximo ciclo de UCP de cada processo que entra para a fila de prontos. A organização da fila está diretamente relacionada com este tempo. Este algoritmo é mais utilizado na implementação de escalonadores de longo prazo. O usuário pode passar para o sistema operacional uma estimativa do tempo de execução do seu programa. Quando usado na implementação de um escalonador de curto prazo, algumas formulações matemáticas são utilizadas para se prever o tempo associado a cada processo. O problema dessa solução é que o tempo pode ser superestimado ou subestimado.

Este algoritmo pode ser preemptivo ou não-preemptivo. No primeiro caso, se chegar à fila de prontos algum processo p1 cujo próximo ciclo de UCP seja menor do que o tempo que falta para o processo p2, que está sendo executado, terminar, então p1 pode retirar o processador do processo p2 e começar a executar. O processo p2 deve voltar para a fila de prontos. No segundo caso essa interrupção não e permitida.

No exemplo a seguir, mostrado na figura 4.9, existem quatro processos na fila de prontos. Cada um deles com o tempo do próximo ciclo de UCP já calculado. De acordo com o algoritmo, a ordem de execução dos processos é p4, p1, p3, p2. O tempo total gasto para executar os ciclos é igual a 24 unidades de tempo e o tempo total de espera é de 28 ut. Os tempos de espera e o tempo de execução de cada processo são:

Processo	T. espera	T. execução
p1	3 ut	6 ut
p2	16 ut	8 ut
p3	9 ut	7 ut
p4	0 ut	3 ut

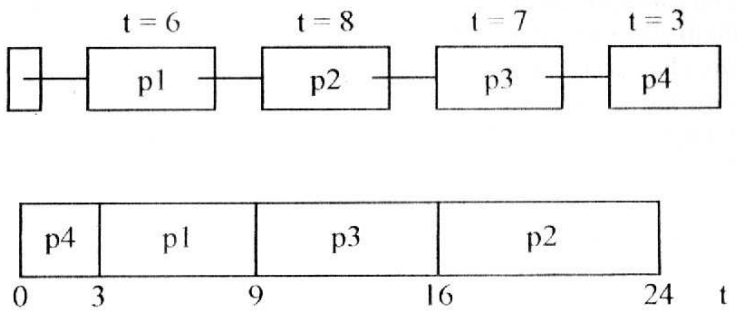
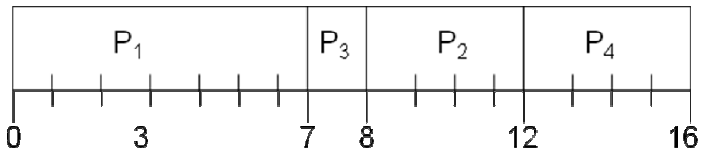


Figura 4.9. Algoritmo *Shortest Job First.*

Novamente, o tempo médio de espera irá variar, mas agora devido ao uso ou não de preempção. A seguir são exemplificados os dois métodos.

SJF (não preemptivo):

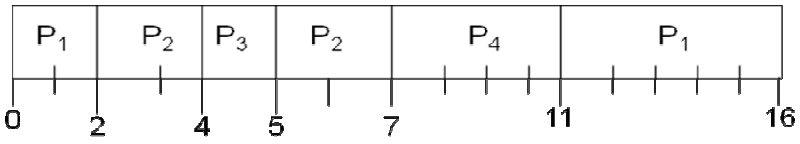
Processo	T. chegada	T. *burst*
P_1	0.0	7 ut
P_2	2.0	4 ut
P_3	4.0	1 ut
P_4	5.0	4 ut



Tempo médio de espera = (0 + 6 + 3 + 7)/4 = 4

SJF (preemptivo):

Processo	T. chegada	T. *burst*
P_1	0.0	7 ut
P_2	2.0	4 ut
P_3	4.0	1 ut
P_4	5.0	4 ut



Tempo médio de espera = (9 + 1 + 0 +2)/4 = 3

Como podemos observar, o uso de preempção torna o processo de escalonamento de processos mais eficaz, porém, como determinar com precisão o tempo de duração do próximo *burst* de UCP? Na verdade, não há como determinar este tempo com exatidão, podendo apenas ser **estimado**, realizando-se uma **média** exponencial dos valores de duração de *bursts* de UCP prévios. A figura 4.10 mostra uma estimativa realizada pelo sistema de escalonamento na tentativa de "adivinhar" os tempos de *burst* futuros.

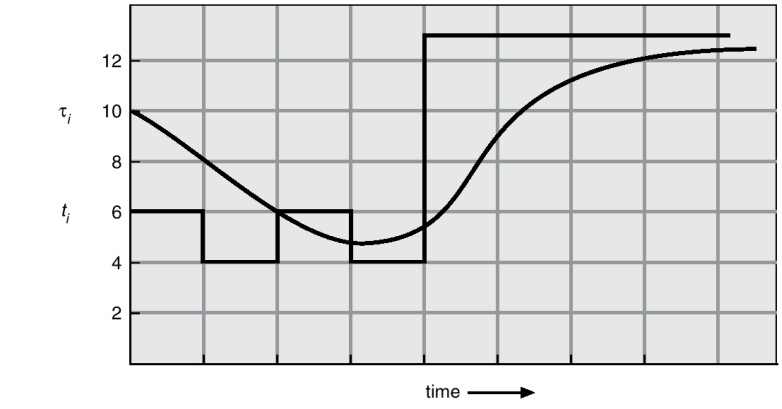


Figura 4.10. Estimativa do tempo do próximo *burst* de UCP.

Prioridade

A cada processo é atribuída uma prioridade, representada por um número inteiro (se é o maior ou o menor número inteiro que representa a maior ou menor prioridade varia de implementação para implementação). A UCP é entregue ao processo que tiver maior prioridade. Processos que tiverem prioridades iguais são escalonados usando-se a idéia de FCFS.

Fazendo uma analogia com os dois algoritmos já apresentados, pode-se considerar que no caso do FCFS prioridade é diretamente relacionada com o tempo de

chegada à fila de prontos. No caso do SJF, a prioridade é inversamente proporcional ao tamanho previsto para o próximo ciclo de UCP do processo,

No caso geral, as prioridades podem ser definidas internamente, pelo sistema operacional, ou externamente, pela própria equipe de usuários, e podem ser atribuídas estaticamente ou dinamicamente. Se a atribuição for estática, a prioridade se mantém constante durante todo o tempo de vida do processo. Se for dinâmica, a prioridade pode mudar ao longo da existência do processo, de acordo com o seu comportamento.

Existem duas vantagens principais na política de escalonamento baseada em prioridades. Primeiro, é possível diferenciar entre os processos segundo sua prioridade. Segundo, existe uma adaptabilidade em relação ao comportamento do processo (no caso da atribuição ser dinâmica). Por outro lado, deve-se cuidar para que todos os processos (com alta ou baixa prioridade) tenham chance de serem executados.

O uso deste método pode causar um problema conhecido como ***starvation***, que é quando processos de baixa prioridade nunca são executados, porque sempre existem processos mais prioritários do que eles. A solução para este caso é chamada de ***aging***, que consiste em fazer com que a medida que o tempo passe a prioridade dos processos aumente, até que ocasionalmente estes venham a ser executados.

O escalonamento com prioridade pode ser preemptivo ou não-preemptivo. No primeiro caso, quando um processo chega à fila de prontos, sua prioridade é comparada com a de todos os processos, inclusive com o que está executando. Se sua prioridade for maior, ele retira o processador do processo que o está utilizando e o processo retirado é retornado a fila de prontos. No caso de não preempção, o processo espera que o processo que está executando correntemente termine.

Múltiplas Filas

Em sistemas onde esta política de escalonamento é usada, a fila de prontos é implementada não só por uma única fila, mas por várias. Cada fila tem uma prioridade e um algoritmo de escalonamento próprio. Os processos são colocados nas filas através de alguma propriedade e, em alguns sistemas, podem migrar de uma fila menos prioritária para outra mais prioritária. Dessa forma, todos os processos serão executados.

Estas filas podem ser executadas em *foreground* (se forem processo interativos) ou *background* (processos em *batch*). Nestes casos, utiliza-se o algoritmo de escalonamento *Round Robin* (a ser visto em seguida) para as que executam em *foreground* e o *First-come-first-served* para as de *background*.

No exemplo mostrado na figura 4.11 existem várias filas, cada uma com a sua prioridade e diversos processos em cada uma delas. Enquanto os processos da fila 1 não tiverem terminado, os das outras filas não podem executar, já que a sua prioridade é maior que as das outras.

Para a execução, deve ser realizado um escalonamento entre as filas. Pode ser um e**scalonamento de prioridades fixas** (serve todos da fila *foreground*, depois da de *background*). O uso deste método traz a possibilidade de *starvation*. Para evitar que isto

ocorra, uma fatia de tempo pode ser usada. Cada fila recebe uma certa quantidade de tempo da UCP, que é escalonada entre os processos da fila (por exemplo, 80% para *foreground* em RR, e 20% para *background* em FCFS)

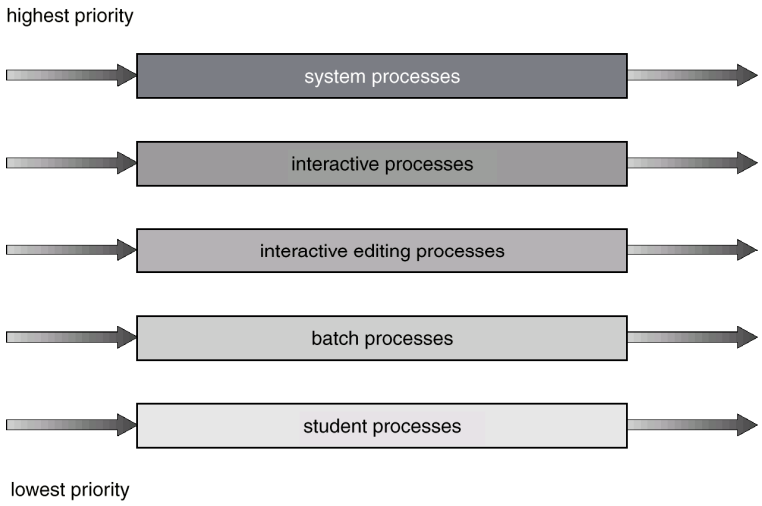


Figura 4.11. Múltiplas filas.

***Round Robin* (RR)**

O algoritmo *Round Robin* é naturalmente preemptivo e especialmente projetado para sistemas de *time-sharing*. Cada processo pode ocupar a UCP durante uma quantidade de tempo definida pelo sistema. Essa fatia de tempo recebe o nome de ***time-slice***. Ao final desse tempo o processo executando perde o direito de continuar usando a UCP e é inserido no final da fila de prontos. Neste caso, o processo passa do estado **executando** para o estado **pronto,** sem ter passado pelo estado **bloqueado.** O primeiro processo da fila recebe o processador e começa a sua execução. Novos processos que entram na fila de prontos também são adicionados ao seu final.

A eficiência deste algoritmo depende muito do tamanho da fatia de tempo. Se ela for muito grande (no caso extremo igual a infinito) o algoritmo se comporta como o FCFS e o tempo de resposta dos processos que ficam no final da fila é aumentado. Se a fatia de tempo for muito pequena, cada um dos *n* processos tem a impressão de estar usando o processador a uma velocidade de 1/*n* **e o tempo despedido na troca de contexto torna-se significativo**. Essa troca de contexto está relacionada com a operação de salvar as informações do processo que foi interrompido, e carregar as informações do processo que recebeu o acesso ao processador. Na figura 4.12 é mostrado um exemplo do uso deste algoritmo.

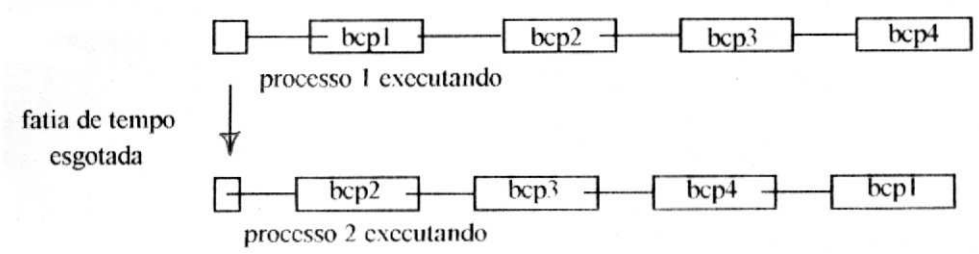
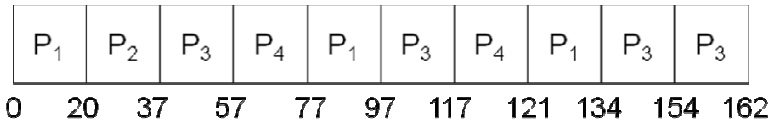


Figura 4.12. Escalonamento Circular (*Round Robin*).

Inicialmente três processos (p2, p3, p4) encontram-se na fila de prontos e o processo p1 está sendo executado. Quando a fatia de tempo deste processo terminar, p2 começa a ser executado e p1 volta para o final da fila. Outro exemplo é apresentado a seguir, mostrando a execução de quatro processos com fatia de tempo igual a 20 ut:

Processo	T. *burst*
P1	53 ut
P2	17 ut
P3	68 ut
P4	24 ut



4.8. Resumo

Um processo é um programa em execução e é representado dentro do sistema operacional como um conjunto de informações que recebe o nome de *Process Control Block (PCB)*. A multiprogramação permite que vários processos permaneçam carregados simultaneamente na memória principal. Esses processos podem compartilhar a UCP. *A* idéia é aumentar o aproveitamento da máquina minimizando os intervalos de tempo em que a UCP fica ociosa.

Cada processo pode se encontrar em um determinado estado, de acordo com a sua atividade. Esses estados são tratados pelo sistema operacional como filas, onde cada elemento é um (PCB). Uma dessas filas é a fila de prontos que contém os processos que estão esperando para serem executados pela UCP. *A* escolha de um processo recebe o nome de escalonamento, e o processo selecionado recebe o controle do processador. Existem diferentes algoritmos para implementar o escalonamento. O objetivo principal é aumentar a taxa de uso do processador e a taxa de saída, e minimizar o tempo total que cada processo gasta para executar, o tempo de espera e o tempo de resposta. Cada um dos algoritmos tem propriedades próprias e apresenta vantagens e desvantagens.

Leitura Obrigatória: Capítulo 2 do livro “Sistemas Operacionais Modernos”, 2ª edição, TANEMBAUM, A., itens 2.1 e 2.5

Capitulo 5

Gerenciamento de Memória

A memória possui uma importância fundamental num sistema de computador. É na memória principal que ficam armazenados os programas que serão executados e a maior parte dos dados que serão manipulados. A memória interage com a CPU e com o subsistema de entrada e saída.

O sistema operacional é responsável pelo controle das atividades da memória, tais como manter informações sobre a sua ocupação, armazenando e liberando espaços, decidir quais processos serão carregados quando houver espaços livres. Existem diferentes esquemas de gerenciamento de memória.

Neste capítulo abordaremos varias estratégias de gerenciamento que têm sido utilizadas nos sistemas de computadores. A maior parte das estratégias tem como objetivo manter vários processos simultaneamente na memória principal para permitir a multiprogramação.

5.1. Introdução

Num sistema computacional tanto a CPU quanto o subsistema de E/S interagem com a memória. Dado que cada conteúdo (palavra ou byte) armazenado na memória possui seu próprio endereço, a interação é feita através de uma sequência de leituras e escritas a endereços de memória específicos.

Em um sistema existem tipos de memória com diferentes características, que formam uma hierarquia, como a ilustrada na figura 5.1, a seguir.

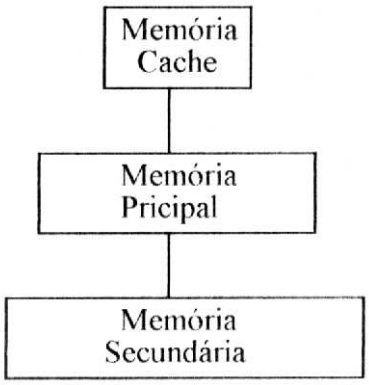


Figura 5.1. Hierarquia de memória.

A **hierarquia de memória** pode ser analisada segundo suas capacidades de armazenamento, custo por bit e tempo de acesso. Partindo do nível mais inferior da Figura 5.1, as memórias secundárias são capazes de armazenar uma grande quantidade de informação, seu custo por *bit é* menor e o seu tempo de acesso é relativamente maior do que as memórias dos outros níveis.

No segundo nível, está a memória principal. Ela é capaz de armazenar uma quantidade menor de informação, seu custo por bit é maior e seu tempo de acesso é menor do que as memórias secundárias.

No nível superior estão as memórias *cache*. Essas memórias são as mais rápidas e com o maior custo por *bit.* Por serem muito caras as memórias *cache são* pequenas, isto *é,* são as que têm menor capacidade de armazenamento.

A abordagem desse capítulo são as diversas políticas de gerenciamento para a memória primária ou memória principal. O gerenciamento de memória usa regras, ou políticas de gerenciamento para:

- **Busca:** Determina quando um bloco de informação deve ser transferido da memória secundária para a memória principal.
- **Armazenamento:** Determina onde o bloco de informação deve ser colocado na memória principal.
- **Substituição:** Determina qual bloco de informação deve ser substituído por um novo bloco.

Existem muitos esquemas diferentes de gerenciamento de memória. A seleção de um esquema, em particular, depende de muitos fatores, mas especialmente depende do suporte de *hardware* do sistema. Iniciaremos com o *hardware* menos complexo até atingirmos os esquemas mais sofisticados.

Em um sistema monoprogramado a gerência da memória principal é menos complexa que nos sistemas multiprogramados. Nos sistemas monoprogramados uma parte do sistema operacional permanece sempre residente na memória, e a cada instante somente um único programa do usuário está carregado na memória principal. O compartilhamento da memória entre o sistema operacional e o programa do usuário, necessita que o sistema possua mecanismos de proteção. Esses mecanismos permitem verificar a validade de acessos à memória gerados pelo programa do usuário para evitar danos ao sistema operacional.

A maioria dos sistemas operacionais atuais é multiprogramado. A multiprogramação permite a execução de vários processos simultaneamente. Dessa forma, existe a necessidade de que vários processos estejam armazenados na memória principal ao mesmo tempo. Neste caso também, além da proteção da área do sistema operacional, é necessário proteger os outros processos residentes na memória, de acessos inválidos gerados por um outro processo.

Um outro problema que surge com a multiprogramação diz respeito à localização dos processos na memória (relocação). Em alguns sistemas, pode-se prever as posições exatas de variáveis e procedimentos na memória que serão ocupadas em tempo de execução (alocação estática). Já em outros sistemas, os espaços ocupados pelos processos podem variar entre execuções (alocação dinâmica). Dessa forma, a geração de endereços de acesso à memória pode ser feita em:

- **Tempo de compilação:** Nesse tempo pode ser gerado um código absoluto se for conhecido onde o programa residirá na memória.
- **Tempo de carga:** Se não for conhecido onde armazenar o programa em tempo de compilação, o compilador deve gerar um código realocável. Nesse caso, somente no tempo de carga será conhecido onde o programa residirá na memória.
- **tempo de execução:** Se um programa pode ser movido de um segmento de memória para outro, durante sua execução então a geração de endereço deve ser retardada até este tempo.

5.2. Modelos de gerenciamento de memória em ambientes monoprogramados

5.2.1. *Bare Machine*

O esquema de memória mais simples é nenhum. Ao usuário é fornecido uma máquina básica e ele tem completo controle sobre a memória. As vantagens desta abordagem são: a flexibilidade máxima para o usuário, já que ele pode utilizar a memória da maneira que desejar, simplicidade e menor custo. Não há necessidade de nenhum hardware especial para se gerenciar a memória. Sua maior desvantagem é não possuir nenhum tipo de proteção. Este esquema é utilizado somente quando há a necessidade de um sistema dedicado.

5.2.2. Monitor Residente

Este esquema também é bem simples e divide a memória em duas partes, uma para o usuário e outra para o sistema operacional. Geralmente é possível armazenar o sistema operacional tanto na memória baixa quanto na memória alta. Porém o mais comum é armazená-lo na parte baixa.

É necessário proteger o sistema operacional de trocas acidentais ou não, provocadas pelo programa do usuário. Esta proteção deve ser fornecida pelo *hardware* e pode ser implementada de diversas maneiras. A abordagem geral é mostrada na figura 6.2. Todo endereço gerado pelo programa do usuário é comparado com um endereço limite (*fence*). Se o endereço gerado é um endereço válido, a referência à memória é realizada normalmente. Caso contrário, se o endereço é menor que o limite, o acesso à memória é negado. A referência à memória é interceptada, e é gerada uma interrupção ao sistema operacional indicando erro de acesso à memória.

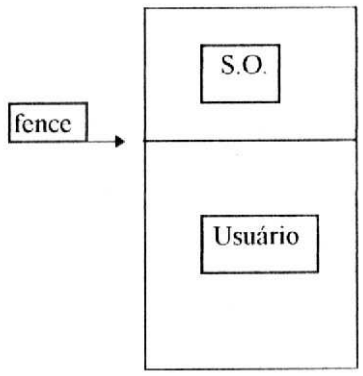


Figura 5.2. Monitor Residente.

O endereço de limite pode ser especificado de vários modos diferentes nos sistemas de computadores. Uma das abordagens é ter o endereço de limite construído *hardware* como uma constante fixa. A dificuldade com o endereço de limite fixo é a seleção correta deste endereço. Se ele for muito pequeno o sistema operacional não ficará completamente protegido. Se ele for muito grande, uma parte da memória não será utilizada pelo sistema operacional e nem pelo usuário.

Para solucionar o problema na variação do tamanho do sistema operacional, pode-se utilizar um registrador de limite. O conteúdo deste registrador é o endereço usado para verificar a correção das referências à memória feita pelo usuário. Este conteúdo pode ser carregado pelo sistema operacional utilizando uma instrução privilegiada especial. Este esquema permite a troca do conteúdo do registrador de limite sempre que o sistema operacional necessite.

5.3. Modelos de gerenciamento de memória em ambientes multiprogramados

5.3.1. Organização de Memória: Partições Fixas X Partições Variáveis

A **organização de memória** é o modo como a memória principal é vista pelo sistema. Isto leva a considerações como:

- Quantos usuários utilizarão a memória?
- Se forem vários usuários, eles ocuparão a memória ao mesmo tempo?
- Quanto de espaço de memória será dado a cada um deles?
- Como a memória será dividida? Em quantas partições?
- Os processos poderão ser executados em qualquer partição?
- Os processos deverão ser colocados na memória de forma contígua ou poderão estar espalhados pela memória principal?

Nos esquemas apresentados a seguir, é necessário que todo o espaço de endereço lógico de um processo esteja na memória física antes que o processo possa ser executado. Dessa forma, o tamanho de um programa fica restrito ao tamanho da memória física do sistema. Estas organizações são ditas organizações de memória real. É importante perceber que o grau de multiprogramação é limitado pelo número de partições da memória principal e influencia diretamente o desempenho do sistema.

5.3.1.1. Partições Fixas

Nesta organização a memória é dividida em número fixo de partições ou regiões. A cada partição pode ser atribuído um processo para ser executado Quando existe uma partição livre, um processo é selecionado de uma fila e carregado naquela partição. Quando ele termina sua execução, a partição torna-se livre para um outro processo.

Em relação à fila de processos, esta pode ser única para cada partição ou única para as várias partições. A figura 5.3 a seguir ilustra estas abordagens.

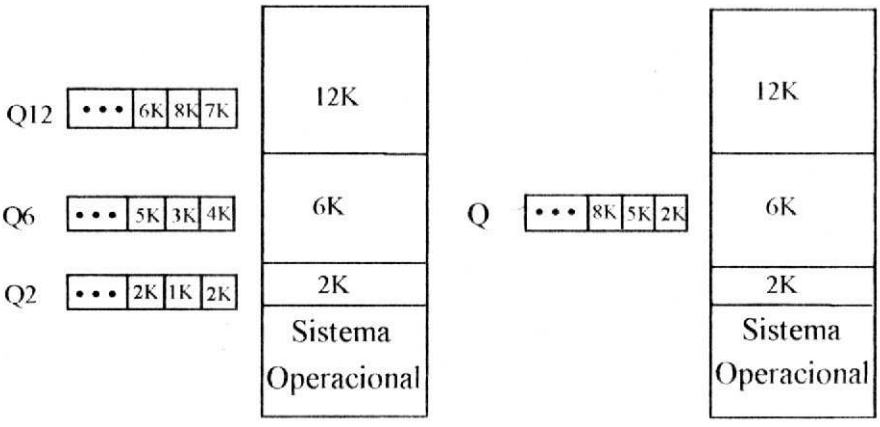


Figura 5.3. Fila de processos separadas e unificada.

Quando existe uma única fila para cada partição, a estratégia é classificar os processos segundo sua quantidade de memória necessária e colocá-lo na fila correspondente. Quando existe uma única fila para todas as partições, o escalonador de processos seleciona o próximo processo que será executado e em qual partição será

carregado. A escolha de um processo para ser armazenado numa partição, depende da política de alocação de memória utilizada, A escolha pode ser *best-fit-only* (menor espaço, dentre os disponíveis, que seja suficiente) ou *best-available-fit* (primeiro espaço, dentre os disponíveis que seja suficiente).

Outro fator importante é a proteção das áreas de memória. Como existem diversos processos residentes na memória simultaneamente, devem haver mecanismos para proteger tanto o sistema operacional quanto os processos. A proteção na relocação estática e na relocação dinâmica é implementada de forma diferente.

A proteção na **relocação estática** é realizada em tempo de montagem ou tempo de carga. Para isto, são utilizados os dois registradores de limite inferior e superior como mostra a figura 5.4. Cada endereço lógico gerado deve ser maior ou igual ao conteúdo armazenado no registrador de limite inferior, e menor ou igual ao conteúdo armazenado no registrador de limite superior. Se for um acesso válido o endereço é então enviado à memória.

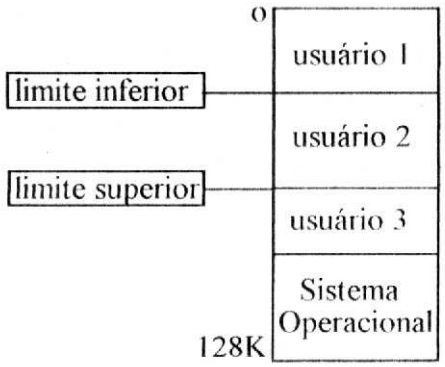


Figura 5.4. *Hardware* de suporte para os registradores de limite superior e inferior.

A proteção na **relocação dinâmica** é feita em tempo de execução. Nesta proteção, são empregados um registrador base e um registrador limite, como ilustrado na figura 5.5. O registrador base contém o valor do menor endereço físico. O registrador limite contém a faixa dos endereços lógicos. Com registradores base e limite, cada endereço lógico deve ser menor que o conteúdo armazenado no registrador limite. Esse endereço, se válido, é então calculado dinamicamente adicionando-se o valor contido no registrador base. O endereço calculado é então enviado à memória,

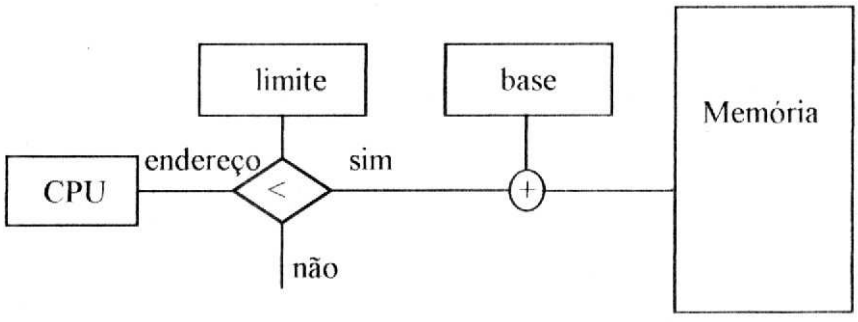


Figura 5.5. Hardware de suporte para os registradores de base e limite

Um dos problemas no projeto de partições fixas envolve a determinação dos tamanhos das partições. Como encontrar uma boa divisão que atenda as necessidades de memória dos processos? Estudos estatísticos podem indicar que conjunto de partições seria conveniente.

Com a divisão da memória em partições fixas podemos ter ainda, o problema da fragmentação da memória. Existe **fragmentação interna** quando há um processo sendo executado em uma partição e ele não a ocupa por completo. Já a **fragmentação externa** existe quando uma partição não é utilizada, por ser pequena demais para qualquer processo que esteja esperando. Tanto a fragmentação interna quanto externa são prejudiciais ao desempenho do sistema. Na organização de memória em partições fixas pode ocorrer os dois tipos de fragmentação.

5.3.1.2. Partições Variáveis

O problema principal das partições fixas é a determinação do tamanho das partições de modo que, a fragmentação interna e externa seja mínima. A organização da memória com partições variáveis visa solucionar este problema permitindo que os tamanhos das partições variem dinamicamente

Para a implementação da organização com partições variáveis o sistema operacional mantém uma tabela indicando quais partes da memória estão disponíveis e quais estão ocupadas. A princípio, toda a memória está disponível e é considerada um grande bloco de memória *(hole)*. Quando um processo chega e necessita de memória, é realizada uma busca por um espaço que seja grande o suficiente para o armazenar. Se existe tal espaço, é atribuído ao processo somente a quantidade de memória necessária. O restante do espaço, que pode haver, é deixado disponível para futuras requisições. Sempre que um processo termina sua execução ele libera seu espaço de memória. Esse espaço liberado é colocado de volta junto com os espaços de memória disponíveis. Neste ponto, procura-se verificar se há áreas adjacentes que possam ser recombinadas de modo a formar espaços de tamanhos maiores.

A figura 5.6 a seguir mostra um exemplo de alocação de memória *e o* escalonamento dos processos.

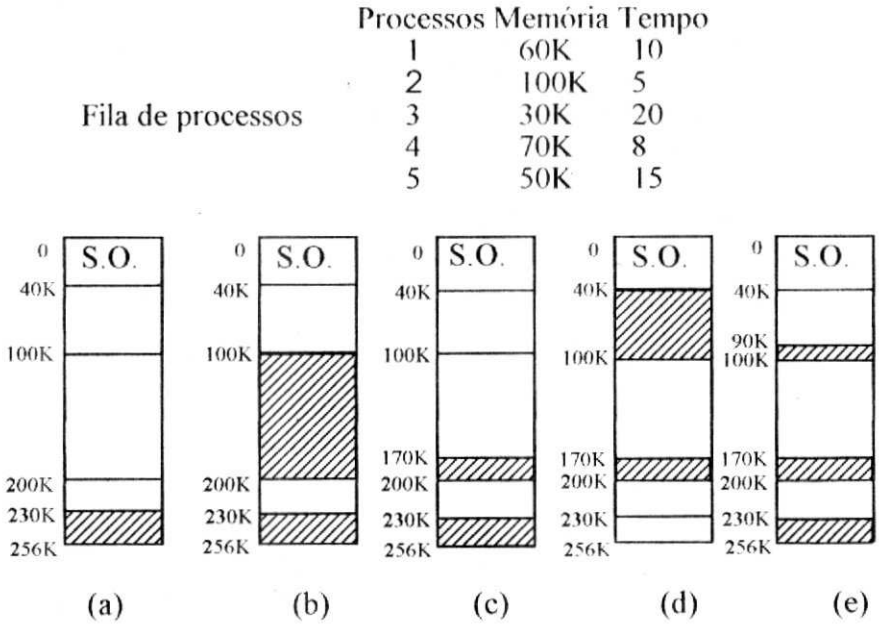


Figura 5.6. Exemplo de alocação de memória e escalonamento de processos.

Neste exemplo, está ilustrada uma memória de 256K, onde existe um monitor residente que ocupa 40K e restam 216K de memória para a execução dos processos. Supondo que exista uma fila de processos como a da figura 5.6, podemos armazenar imediatamente os três primeiros processos na memória (figura 5.6(a)). Assim, temos uma fragmentação externa de 26K, Utilizando o algoritmo de escalonamento de *CPU Round Robin com* um quantum de 1 unidade de tempo, o processo 2 terminará no tempo 14, liberando seu espaço de memória (figura 5.6(b)). E feito um novo acesso a fila de processos, e o processo 4 é armazenado na memória (figura 5.6(c)). O processo 1 terminará no tempo 28 levando a figura 5.6(d), sendo então o processo 5 escalonado produzindo a figura 5.6(e).

Como pode ser observado no exemplo, em qualquer tempo há um conjunto de espaços livres, de tamanhos variados e espalhados pela memória. Além disso, existe um conjunto de processos esperando para serem executados. Esta situação pode ser vista como uma aplicação geral do problema de alocação de memória dinâmica. O problema é como satisfazer um pedido de tamanho *n* de uma lista de espaços livres. As estratégias mais comuns para selecionar um espaço de memória são:

- ***First-fit*:** Aloca o primeiro espaço grande o suficiente. Não necessita de uma busca por toda a lista. É a estratégia mais rápida.
- ***Best-fit**:* Aloca o menor espaço que seja grande o suficiente. É necessário percorrer toda a lista, a menos que esta esteja ordenada. Esta estratégia é a que provoca menor fragmentação da memória
- ***Worst-fit*:** Aloca o maior bloco. Também é necessária uma busca pela lista inteira, a menos que ela esteja ordenada por tamanho. Esta estratégia visa deixar espaços de memória maiores livres.

A proteção das áreas de memória é feita do mesmo modo que nas partições fixas. O *hardware* é idêntico. O que diferencia as partições fixas e as partições variáveis é o *software*.

Em geral, a utilização da memória com partições variáveis é melhor do que com partições fixas. Praticamente não existe fragmentação interna, só fragmentação externa, o que ainda assim, pode ser muito ruim. Suponha tinia memória com vários espaços livres, mas espalhados pela memória. Suponha ainda que, esses espaços estivessem juntos e formassem um bloco grande o suficiente, onde pudessem ser executados outros processos. Essa fragmentação é prejudicial ao desempenho do sistema.

Para solucionar a fragmentação externa podemos utilizar a técnica de **compactação.** O objetivo dessa técnica é juntar os espaços espalhados pela memória em um único grande bloco. Essa técnica só é possível de ser utilizada se houver relocação dinâmica. Deve-se levar em consideração também, a estratégia a ser escolhida e o custo da compactação.

5.4. Memória Virtual

A técnica de **memória virtual** foi criada para permitir a execução de vários processos que não necessariamente estejam armazenados por inteiro na memória principal. Uma das vantagens mais importantes é que os programas dos usuários podem ser maiores do que a memória física. Além disso, como cada programa de usuário não necessita estar completamente armazenado na memória pode ser executado um maior número de programas simultaneamente. Isso leva a uma maior utilização da CPU e um aumento no *throughput*.

A memória virtual é a separação da memória lógica do usuário da memória física. Isso torna a tarefa de programação mais fácil na medida em que o programador não precisa se preocupar com a quantidade de memória física disponível. A implementação da memória virtual é comumente realizada com paginação sob demanda (*demand paging*) ou com segmentação sob demanda (*demand segmentation*). Nas seções seguintes descreveremos as técnicas de *swapping*, paginação e segmentação necessárias à implementação da memória virtual.

5.4.1. Swapping

A técnica de **swapping** requer que o sistema possua um *backing store*. O *backing store* é uma memória secundária, geralmente um disco de rápido acesso (leitura e gravação). Ele deve ser grande o suficiente para armazenar cópias de todos os programas de usuários e fornecer acesso direto a esses programas.

O *swapping* nada mais é do que a troca do conteúdo de um determinado espaço de memória. A figura 5.7 a seguir ilustra o *swapping* de dois programas utilizando um disco como *backing store.*

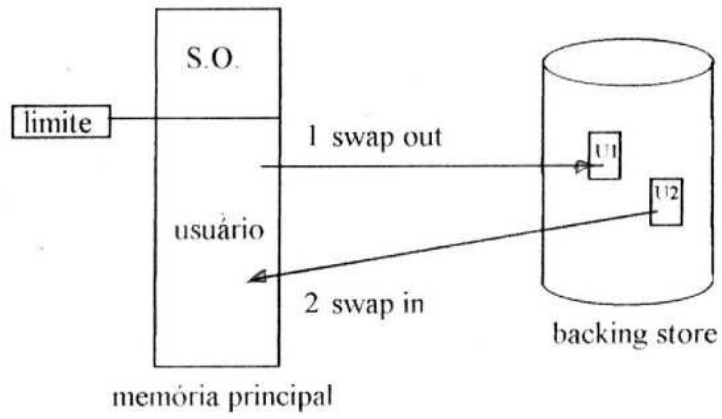


Figura 5.7. *Swapping* de dois programas utilizando um disco.

A fila de prontos (*ready queue)* consiste de todos os processos que estão no *backing store* e que estão prontos para serem executados. Um conjunto de variáveis do sistema indica quais processos estão correntemente na memória

Sempre que o escalonador da CPU decide executar um processo, ele chama o *dispatcher. Esse* verifica se o processo escolhido está presente na memória. Caso não esteja, ele troca um dos processos correntes na memória pelo processo desejado. O *dispatcher é* responsável pela troca de contexto dos processos, isto é, salva o conteúdo dos registradores do processo que está sendo retirado, armazena nos registradores os valores correspondentes ao novo processo e transfere o controle para o processo.

O processo escolhido para ser retirado da memória e levado ao *backing store* deve estar completamente ocioso. Nenhum processo com entrada e saída pendente deve ser trocado.

5.4.2. Paginação

A **paginação** é uma organização de memória que permite que a memória física seja vista como se estivesse dividida em blocos de tamanho fixo, chamados *frames*. A memória lógica é também dividida em blocos do mesmo tamanho, chamados páginas (*pages*). Quando um programa vai ser executado, suas páginas são trazidas do *backing store* e carregadas nos *frames* disponíveis. O disco é dividido em blocos de tamanho fixo, cujo tamanho é o mesmo dos *frames* da memória.

O *hardware de* suporte para a paginação é mostrado a seguir na figura 6.8. Todo endereço gerado pela CPU é dividido em duas partes: o número de página (*page number p*) e deslocamento dentro da página (*page offset - d*). O *page number* é utilizado como um índice para a tabela de páginas. A **tabela de páginas** contém o endereço base de cada página da memória física Este endereço base é combinado com o *page offset* para definir o endereço enviado a memória física.

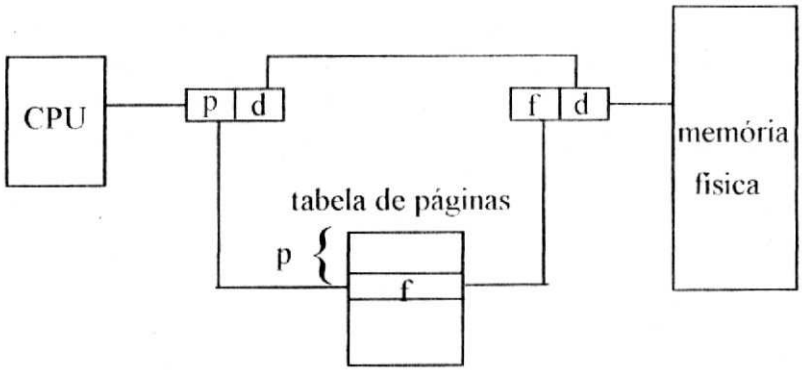


Figura 5.8. Hardware para paginação.

O tamanho da página (e do frame) é definido pelo hardware. Tipicamente varia entre 512 e 2048 palavras, dependendo da arquitetura do computador. A figura 5.9. a seguir, exemplifica a visão do usuário sobre como a memória lógica pode ser mapeada na memória física.

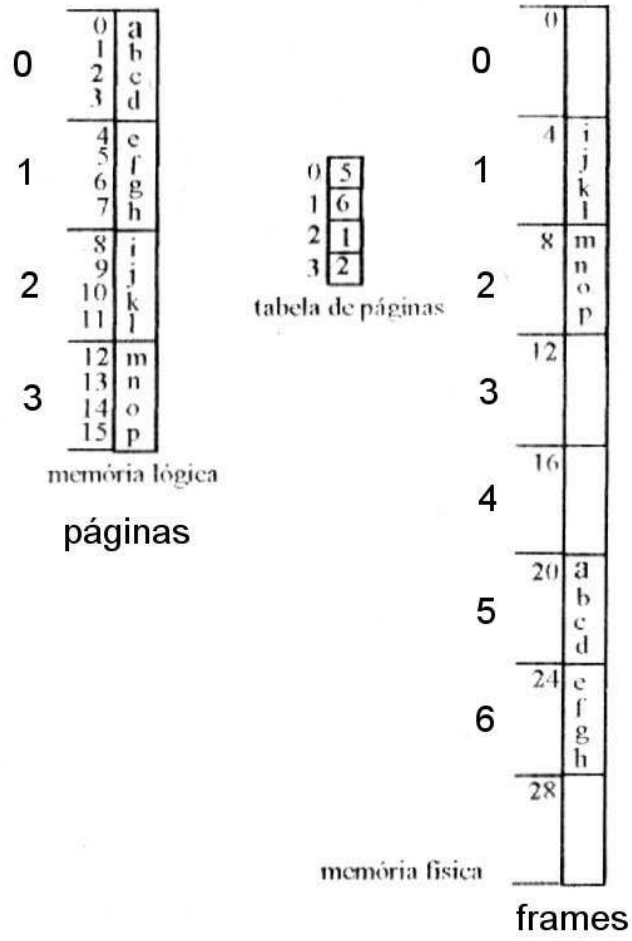


Figura 5.9. Exemplo de paginação para uma memória de 32 palavras com 8 páginas.

Nesta figura 5.9 está ilustrada uma memória física de 32 palavras dividida em 8 páginas. Cada página possui 4 palavras. Suponha um acesso ao endereço lógico 3, que pertence à página 0. Realizando a indexação na tabela de páginas, encontramos que a

página 0 está armazenada no frame 5. Dessa forma, o endereço lógico 3 é mapeado ao endereço físico 23 (= 5 x 4 + 3).

Para a alocação de um processo na memória é necessário expressar seu tamanho em número de páginas. Se o processo tiver tamanho *n*, ele necessitará de no máximo *n* frames. A cada frame alocado, seu número é colocado na tabela de páginas, para posterior geração de endereço físico.

Cada processo possui a sua própria tabela de páginas. A implementação da tabela de páginas pode ser feita com um conjunto de registradores dedicados ou ser mantida na memória e utilizar um registrador, que armazena um apontador para a tabela de páginas. Este registrador é chamado Page *Table Base Register (PTBR),* e o seu conteúdo é armazenado junto com outros valores de registradores no *PCB (Process Control Block).*

O problema com o uso de uma tabela de páginas é o tempo de acesso a uma locação de memória do usuário. São necessários na realidade dois acessos, um para realizar o acesso à tabela de páginas e outro para a posição de memória da palavra desejada. A solução para diminuir este tempo de acesso é utilizar uma memória *cache* para translação de endereços lógicos em endereços físicos.

O sistema operacional gerencia a memória física mantendo informações de quais *frames* estão alocados, quais deles estão disponíveis, quanto *frames* existem no total, entre outras, Estas informações são mantidas, geralmente, em urna estrutura de dados denominada **tabela de frames** (frame *table).* A tabela de frames possui uma entrada para *cada frame da* página física, indicando se o frame está disponível ou não, e se alocado, qual página de que processo está associada ao frame.

Uma das vantagens da paginação e a possibilidade do compartilhamento de código. Todo código reentrante**,** isto é, aquele que não muda durante a execução, pode ser compartilhado. Basta que, cada processo tenha na sua tabela de páginas o mesmo endereço do código armazenado na memória física. Podem ser compartilhados então compiladores, montadores, *Iinkers entre* outros.

A proteção de memória no ambiente paginado é realizada por *bits* de proteção associados com cada página. Esses *bits* são normalmente mantidos na tabela de páginas e definem permissão de leitura/escrita e acesso válido/inválido. Com a paginação não há fragmentação externa, porém pode haver alguma fragmentação interna, que ocorre se o último frame alocado não estiver completo.

5.4.3. Segmentação

A **segmentação** é um esquema de gerenciamento de memória que pode dar suporte à visão que o usuário possui sobre a memória. O usuário pensa na memória como um conjunto de subrotinas, procedimentos, tabelas, variáveis e assim por diante, Cada um desses segmentos tem um nome, é de tamanho variável e não possui uma ordenação específica. Os elementos dentro de cada segmento são identificados por sua posição dentro do segmento, como por exemplo, a primeira entrada numa tabela de símbolos. Dessa forma, um espaço de endereço lógico é uma coleção de segmentos. Os endereços especificam tanto o nome do segmento quanto a posição dentro do segmento.

O usuário especifica então cada endereço por estas duas quantidades: um nome de segmento e uma posição.

Por simplicidade de implementação, os segmentos são numerados e referenciados por seu número. Normalmente, o programa do usuário é montado (ou compilado) e o montador (ou compilador) automaticamente constrói segmentos refletindo o programa de entrada.

Embora o usuário possa referenciar objetos dentro do programa por endereços bidimensionais, a memória física é uma sequência unidimensional de palavras. Assim, é necessário definir um mapeamento entre estas duas visões. Este mapeamento é efetuado por uma **tabela de segmentos.** O uso da tabela de segmentos é mostrado na figura 6.10.

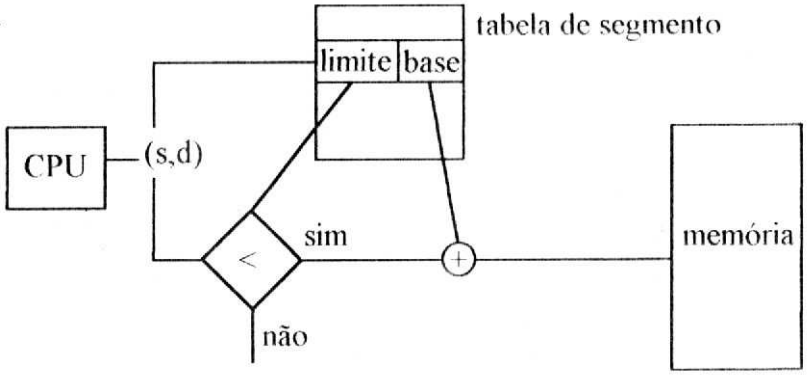


Figura 5.10. *Hardware* para segmentação.

O endereço lógico consiste de duas partes: um número de segmentos e um *offset d*, dentro do segmento. O número do segmento é usado como um índice para a tabela de segmentos. Cada entrada na tabela possui uma base e um limite do segmento. O off*set d* do endereço lógico deve estar entre 0 e este limite. Se for um acesso válido, o *offset é* adicionado à base do segmento para produzir um endereço físico. A tabela de segmentos é um *array* de pares de registradores base/limite. Para exemplificar a segmentação, considere a figura 5.11 mostrada a seguir.

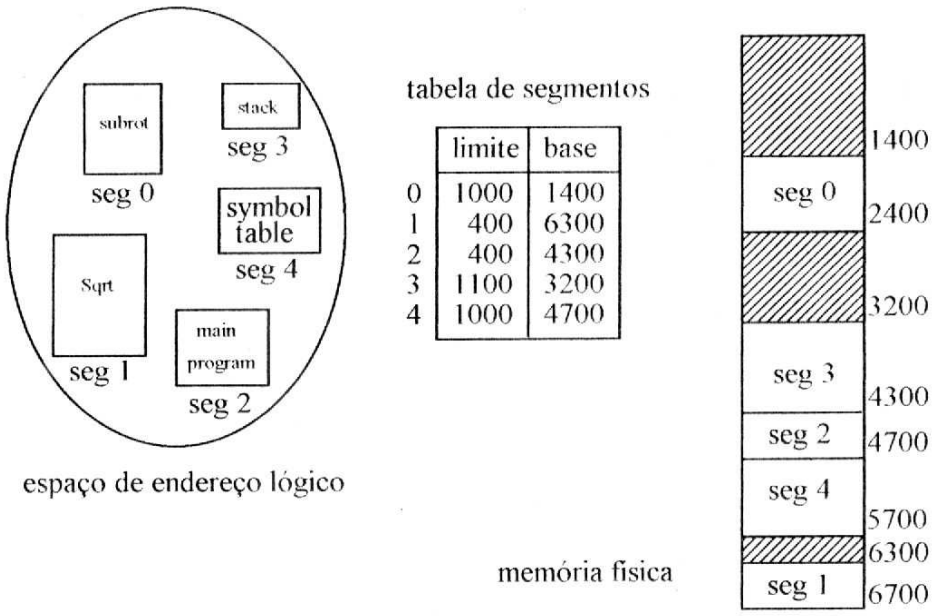


Figura 5.11. Exemplo de segmentação.

Na figura 5.11 são ilustrados cinco segmentos numerados de 0 a 4. Suponha um acesso à palavra 53 do segmento *2*. Seu endereço físico é dado pela base 4300 mais a posição dentro do segmento 53, gerando então o endereço fisico 4353 (4300 + 53). Da mesma forma que a tabela de páginas, a tabela de segmentos pode ser armazenada em registradores ou na memória. O registrador que armazena o apontador para a tabela de segmentos, quando essa está armazenada na memória, é chamado *Segment Table Base Register* (STBR).

A segmentação também permite o compartilhamento de código. Os segmentos são compartilhados quando, entradas nas tabelas de segmento de diferentes processos apontam para as mesmas localizações físicas. A proteção na segmentação é associada a cada segmento. Cada entrada na tabela de segmentos contém informações para prevenir acesso ilegal de leitura/escrita ou de acesso fora dos limites do segmento.

Como os segmentos são de tamanhos variados e armazenados corno tal, não existe fragmentação interna no gerenciamento de memória com segmentação. Porém, a segmentação pode causar fragmentação externa quando todos os espaços livres de memória são muito pequenos para armazenar um segmento.

5.4.4. Paginação Sob Demanda

Como mencionado anteriormente, a memória virtual é comumente implementada com **paginação sob demanda ou** com **segmentação sob demanda.** Descreveremos a seguir a paginação sob demanda. Raciocínio análogo pode ser feito na segmentação sob demanda.

A paginação sob demanda é similar a um sistema paginado com *swapping*. Os programas residem na memória secundária. Quando se inicia a execução, os programas são trazidos para a memória principal. Porém, nunca uma página é trazida para a

memória se ela não for necessária. Com isso, diminuímos o tempo de troca e a quantidade de memória física necessária.

Para controlar o armazenamento das páginas trazidas para a memória, a tabela de páginas possui um bit de válido/inválido. Esse bit é ativado quando a página está presente na memória. Se o programa tenta acessar uma página que ainda não foi trazida para a memória, então é gerada uma interrupção por falta de página (*page fault*). Neste caso, é utilizado o seguinte procedimento para carregar uma página:

- Busca-se um frame na lista de *frames* livres;
- Escalona-se uma operação de disco para ter a página desejada para o frame alocado;
- Quando a leitura do disco se completar, a tabela de páginas é modificada para indicar que a página agora está presente na memória e reinicia-se a instrução que foi interrompida pelo acesso ilegal. Todo este procedimento é ilustrado pela figura 5.12.

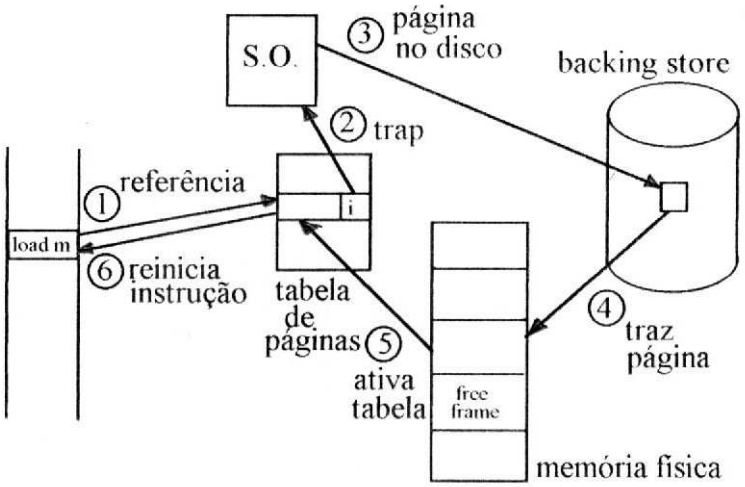


Figura 5.12. Passos na manipulação de uma falta de página.

No procedimento descrito, não abordamos a situação onde a memória estivesse completa, isto é, não houvessem *mais frames livres* para serem alocados. Esta situação pode ser solucionada com a substituição de alguma página. A escolha de uma página a ser trocada leva a considerar algumas estratégias de substituição de páginas que serão abordadas na seção seguinte.

Em resumo, podemos dizer que os dois principais problemas que devem ser solucionados para implementar a paginação sob demanda são a alocação de *frames* e a substituição de páginas. Se tivermos vários processos na memória, devemos decidir quantos *frames* alocar a cada um deles. E quando necessário substituir uma página, devemos selecionar quais *frames* serão substituídos.

5.4.5. Estratégias de Substituição

As estratégias de **substituição de páginas** visam escolher uma pagina para ser trocada por outra, se não existirem *frames* livres. *Um frame* é liberado escrevendo seu

conteúdo num disco e trocando a tabela de páginas para indicar que a pagina não está mais residente na memória. *O frame* liberado pode ser então utilizado por outra página.

Existem diversos algoritmos de substituição diferentes. A escolha de um, em particular leva em consideração a **taxa de falta de páginas** *(page fault rate).* Devemos escolher aquele que apresentar a menor taxa. Uma quantidade de trocas de páginas elevada degrada o desempenho do sistema. A seguir descreveremos alguns algoritmos de substituição de páginas.

First-In-First-Out (FIFO)

O algoritmo *First-In-First-Out* é o mais simples. Ele associa a cada página, o tempo em que ela foi trazida para a memória. Quando uma página tiver que ser substituída, a pagina mais antiga na memória é escolhida.

Apesar deste algoritmo ser simples de entender e implementar nem sempre seu desempenho é bom. Suponha que temos uma página em uso constante e há bastante tempo na memória e uma outra trazida mais recentemente, mas que não será mais utilizada. Com este algoritmo a escolha recai sobre a página em uso. Esta página provavelmente provocará uma nova falta de páginas. Isso aumenta a taxa de falta de paginas e torna a execução do programa mais lenta. A figura 5.13 a seguir ilustra o algoritmo FIFO para um *string* de referências arbitrário. Com esta sequência de referências temos 15 faltas de página.

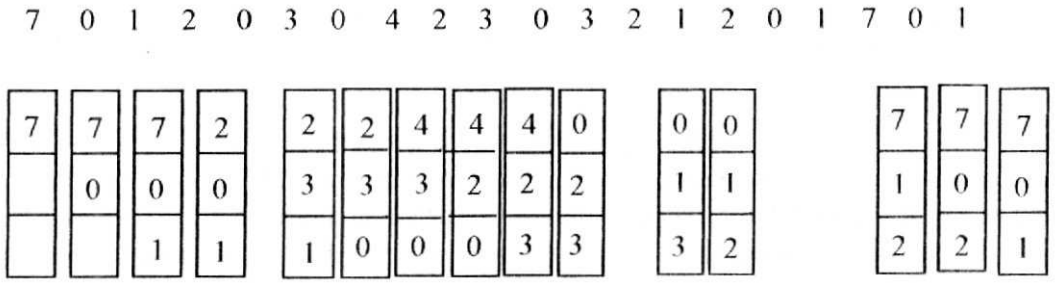


Figura 5.13. Substituição FIFO.

Optimal Replacement (OPT)

O algoritmo de substituição de páginas ótimo (OPT) é o que apresenta a menor taxa de falta de páginas. Neste algoritmo, é substituída a página que não será utilizada pelo maior período de tempo. Infelizmente, o algoritmo de substituição ótimo é difícil de implementar, uma vez que ele requer um conhecimento futuro das referências à memória. Por esse motivo, o algoritmo ótimo é utilizado principalmente em estudos comparativos. Como exemplo, utilizamos o mesmo *string* de referências da figura 5.13, na figura 5.14, que ilustra o algoritmo ótimo. Aqui são causadas nove faltas de página.

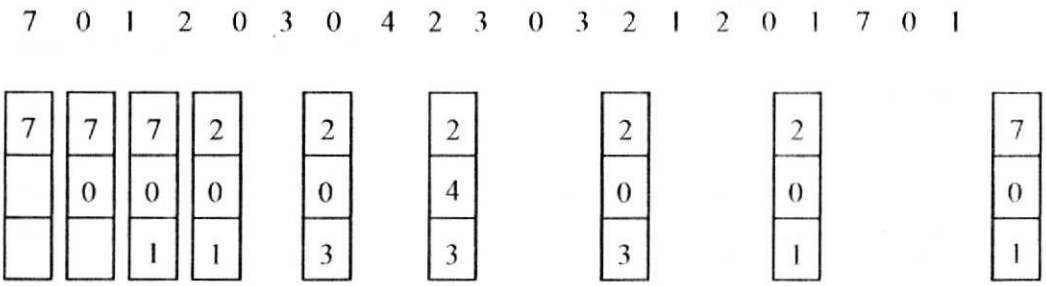


Figura 5.14. Substituição OPT.

Least Recently Used (LRU)

O algoritmo de substituição de páginas *Least Recently Used* é uma tentativa de aproximação ao algoritmo ótimo. O algoritmo ótimo utiliza na sua concepção o conhecimento futuro das referências à memória. O algoritmo LRU utiliza o conhecimento da história do passado recente das referências à memória, como uma aproximação do futuro.

Este algoritmo associa a cada página seu último tempo de uso. Quando houver a necessidade de substituir uma página, é escolhida aquela que não foi utilizada pelo maior período de tempo. Essa estratégia é conveniente ao princípio da localidade. Por esse princípio, quando uma página é referenciada, existe uma grande chance de que ela seja novamente referenciada em breve. A figura 5.15 a seguir ilustra o comportamento deste algoritmo, onde ocorreram 12 faltas de página.

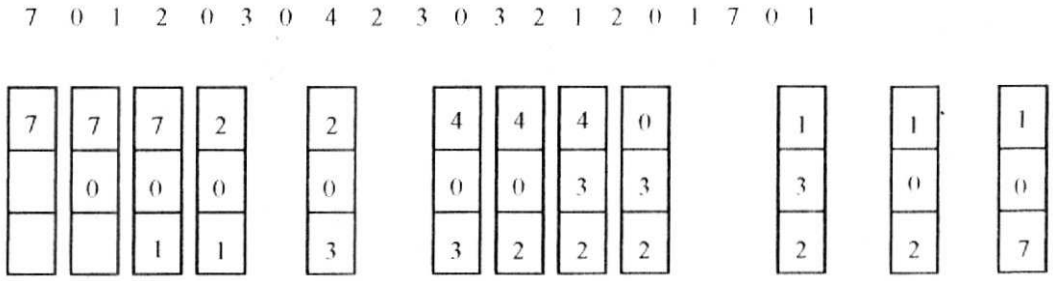


Figura 5.15. Substituição LRU.

O algoritmo LRU é bastante utilizado e considerado muito bom. O problema com ele está na sua forma de implementação. O sistema necessita manter uma lista das páginas da memória, ordenada por último uso. Há duas formas de implementação:

- **contador.** A cada entrada na tabela de páginas é associado um registrador de tempo de uso. Sempre que uma referência a pagina é feita, o valor do tempo é carregado no registrador. A página substituída deve ser aquela com o menor valor de tempo.
- **pilha.** Nessa abordagem é mantida uma estrutura de pilha dos números das páginas. Quando a pagina é referenciada, ela é removida da pilha e colocada no topo. Dessa forma, o fundo da pilha sempre contém a página usada menos recentemente.

Note que as implementações do LRU necessitam de algum auxílio do *hardware*. A atualização dos registradores ou da pilha deve ser feita a cada referência à memória. Se utilizarmos uma interrupção a cada referência à memória, para permitir ao *software* atualizar as estruturas de dados, o sistema se degrada. Poucos sistemas tolerariam tal nível de *overhead* no gerenciamento de memória.

5.5. Outras Considerações

Número Mínimo de *Frames*

Devemos ainda pensar sobre como a quantidade de memória livre pode ser alocada aos vários processos. Existe um número mínimo de *frames* que devem ser alocados Este número mínimo é definido pelo conjunto de instruções da arquitetura. Devemos em qualquer tempo, ter *frames* suficientes para manter todas as páginas diferentes que qualquer única instrução possa referenciar. Por exemplo, considere uma maquina na qual todas as suas instruções de referência à memória possibilitem um nível de indireção. São necessários então pelo menos três *frames*, um para a instrução um para o endereço e outro para a informação propriamente dita.

Resumindo, o número mínimo de *frames* por processo é definido pela arquitetura, enquanto o número máximo é definido pela quantidade de memória física disponível na maquina.

Alocação Local x Global

Com vários processos competindo por *frames* podemos classificar a substituição de pagina em duas categorias denominadas substituição local e substituição global.

A substituição global permite que seja selecionado para substituição um *frame* do conjunto de todos os *frames*, ainda que, aquele frame esteja alocado a um outro processo que não seja o processo que está requisitando um frame. Isto é, um processo pode utilizar um *frame* pertencente a outro processo. Já a substituição local requer que cada processo selecione para substituição somente os *frames* pertencentes ao seu proprio conjunto de *frames* alocados.

Thrashing

Se a quantidade de *frames* alocada a um processo cai abaixo do número mínimo requerido pela arquitetura do computador, devemos então suspender a sua execução. Embora seja possível tecnicamente reduzir o número de *frames* alocados ao valor mínimo, existe um número de páginas que estão em uso ativo. A consequência deste fato é ter uma alta taxa de *page faults*. Por exemplo, as páginas em uso ativo podem gerar referências à páginas que não estejam presentes na memória, logo elas precisam ser carregadas. Por sua vez, estas páginas recém carregadas geram novas referências e assim por diante.

Esta alta atividade de paginação é chamada de *thrashing*. Dizemos que um processo está em *thrashing se* ele passa mais tempo paginando do que executando Este estado causa sérios problemas de desempenho no sistema.

Modelo de Conjunto de Trabalho

O modelo do **conjunto de trabalho** *(working set model)* é baseado na suposição da localidade. Este modelo utiliza um parâmetro Δ, para definir o conjunto de trabalho. A idéia é examinar as Δ referências de página mais recentes. Este conjunto é o *working set*. Se a página está em uso ativo, ela será incluída no conjunto. Caso contrário, após *n* unidades de tempo depois da última referência, ela será retirada do conjunto de trabalho. Dessa forma, o conjunto de trabalho é uma aproximação da localidade do programa.

Por exemplo, dada a sequência de referências mostrada na figura 5.16, se Δ = 3 referências de memória, então o conjunto de trabalho no tempo t1 é {1}. No tempo t2, o conjunto é trocado para {5,6}

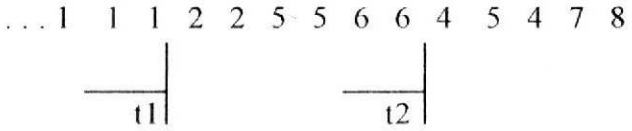


Figura 5.16. Modelo do conjunto de trabalho

5.6 Resumo

O gerenciamento de memória para sistemas monoprogramados é menos complexo do que nos sistemas multiprogramados. O grau de multiprogramação influencia diretamente no desempenho do sistema.

Foram discutidas algumas estratégias de gerenciamento de memória como *bare machine monitor* residente, partições fixas, partições variáveis, paginação e segmentação. Um dos fatores determinantes na escolha de uma política de gerenciamento, em particular, é o *hardware do* sistema.

Tanto com *bare machine*, monitor residente, partições fixas quanto com partições variáveis, é necessário que um processo esteja completamente armazenado na memória para que seja executado.

A técnica de memória virtual permite que processos maiores do que a memória física disponível sejam executados. Essa técnica é implementada com paginação sob demanda ou com segmentação sob demanda. Na paginação sob demanda, uma página só é trazida para a memória principal quando ela é necessária. Uma vez que pode ocorrer que não existam mais *frames* livres na memória, e necessário utilizar algum algoritmo que substitua as páginas na memória. Os algoritmos de substituição de páginas descritos foram o OPT, o FIFO e o LRU.

Abordamos ainda algumas considerações como qual a quantidade de *frames* a ser atribuído a cada processo, o problema do *thrashing* e o modelo de conjunto de trabalho.

Leitura Obrigatória: Capítulo 4 do livro “Sistemas Operacionais Modernos”, 2ª edição, TANEMBAUM, A.