Colégio Técnico Universitário – CTU

Curso Técnico de Informática Industrial

Professor: Stênio Sã Rosário F. Soares

Disciplina: **Introdução a Sistemas Operacionais**

Material de Apoio

Este material é o resultado da compilação de vários outros materiais utilizados na apresentação da disciplina de Introdução a Sistemas Operacionais. Os primeiros três capítulos foram extraídos de material elaborado por mim e pelas professoras Cristiane Targa e Melba Gorza quando ministramos as disciplinas de Introdução à Informática e Informática I na Universidade Federal Fluminense, bem como do material do Prof. Sandro, disponível no site do CTU. O Capítulo 4, que trata do Sistema Operacional Linux foi integralmente obtido no endereço http://200.129.176.74/~bruno/.

Capítulo 1

Evolução Histórica do Hardware

Este capítulo é para os alunos curiosos, aqueles que não perdem oportunidade de saber um pouco além do conteúdo da matéria. Não será cobrado diretamente nas avaliações, mas certamente é importante que o profissional técnico tenha uma idéia do que foi o passado das suas ferramentas de trabalho.

Centenas de computadores de diferentes tipos têm sido projetadas e construídas durante a evolução do computador digital moderno. A maioria já foi esquecida, mas alguns tiveram um impacto significativo nas idéias modernas. A seguir, daremos uma visão breve de alguns destes desenvolvimentos-chave históricos, para conseguir uma melhor compreensão de como chegamos até onde estamos agora.

1) Geração Zero – Computadores Mecânicos (1642-1945)

Antes do surgimento dos algarismos arábicos, os cálculos eram efetuados através do **ábaco**, um instrumento criado provavelmente pelos Sumérios, um povo que habitava a Mesopotâmia, por volta de 500 a.C. A palavra *ábaco* vem do grego *abax*, que significa *tábua de cálculo*. Era constituído por um conjunto de bolinhas, atravessadas por um fio pelo qual podiam deslizar. Certas regiões do Oriente mantêm ainda hoje o costume de utilizar o ábaco como instrumento para efetuar cálculos.

A primeira pessoa a construir uma máquina de calcular foi o cientista francês Blaise Pascal, em cuja honra deu-se o nome à linguagem de programação Pascal. Este aparelho, construído em 1642, quando Pascal tinha apenas 19 anos, foi projetado para ajudar seu pai, um coletor de impostos para o governo francês. Era inteiramente mecânico, utilizava engrenagens e funcionava através de uma manivela (operação manual).

A máquina de Pascal podia apenas subtrair e somar, porém trinta anos mais tarde o matemático alemão Barão Gottfried Wilhelm von Leibniz construiu uma outra máquina mecânica que podia também multiplicar e dividir. De fato, Leibniz construiu o equivalente a uma calculadora de bolso de quatro funções, há mais de três séculos.

Nada mais aconteceu durante 150 anos até que um professor de matemática na Universidade de Cambridge, Charles Babbage projetou e construiu sua **máquina de diferenças**. Este dispositivo mecânico, que do mesmo modo que o de Pascal só podia somar e subtrair, foi projetado para calcular tabelas de números úteis à navegação. A máquina foi projetada para executar um algoritmo simples, o método das diferenças finitas utilizando polinômios. O aspecto mais importante da máquina de diferenças era seu método de saída: ela perfurava os resultados numa placa de cobre. Seria o método de escrita percursor do advento dos cartões perfurados e discos.

Embora a máquina de diferenças funcionasse razoavelmente bem, Babbage logo cansou-se dela, pois podia executar apenas um algoritmo. Ele começou então a dedicar o seu tempo no projeto e construção de uma máquina sucessora, a **máquina analítica** (1834). Esta máquina possuía quatro componentes: o armazenamento (memória), o engenho (a unidade de cálculo), a seção de entrada (leitora de cartões perfurados) e a seção de saída (saída perfurada e impressa). O armazenamento tinha capacidade de guardar 1000 valores de 50 dígitos, sejam eles operandos ou resultados de operações. O engenho podia aceitar operandos do armazenamento, somá-los, subtraí-los, multiplicá-los ou dividi-los, e retornar o resultado ao armazenamento. Tal como a máquina de diferenças, ela era totalmente mecânica.

O grande avanço da máquina analítica era de ser de uso geral. Ela lia instruções de cartões perfurados e as executava. Algumas instruções comandavam a máquina para buscar dois números no armazenamento, levá-los ao engenho, sofrerem a operação (por exemplo, soma) e ter o resultado enviado de volta ao armazenamento. Perfurando um programa diferente nos cartões de entrada, era possível fazer com que a máquina analítica executasse cálculos diferentes, algo que não era verdade na máquina de diferenças.

Uma vez que a máquina analítica era programável, ela precisava de *software*. Para produzir este *software*, Babbage contratou uma jovem chamada Ada Augusta Lovelace, sendo esta a primeira programadora do mundo. A linguagem de programação *Ada* foi assim denominada em sua homenagem.

O próximo grande marco de desenvolvimento ocorreu nos anos 30, quando um estudante de engenharia alemão chamado Konrad Zuse construiu uma série de máquinas de calcular automáticas utilizando relés (aparelhos/circuitos) eletromagnéticos. Zuse não conhecia o trabalho de Babbage, e suas máquinas foram destruídas por um bombardeio a Berlim em 1944, de forma que seu trabalho não teve qualquer influência nas máquinas posteriores. Mesmo assim, ele foi um dos pioneiros neste campo.

Logo depois, nos Estados Unidos, duas pessoas também projetaram calculadoras, John Atanasoff e George Stibbitz. A máquina de Atanasoff era surpreendentemente avançada para sua época. Ela utilizava aritmética binária além de técnicas utilizadas ainda hoje na manutenção da carga de memórias. Infelizmente, a máquina nunca se tornou realmente operacional. De certa forma, Atanasoff foi como Babbage: um visionário derrotado pela inadequada tecnologia de *hardware* de seu tempo. O computador de Stibbitz, embora mais primitivo que o de Atanasoff, funcionava realmente.

Enquanto Zuse, Stibbitz e Atanasoff estavam projetando calculadoras automáticas, um jovem chamado Howard Aiken estava realizando manualmente tediosos cálculos numéricos como parte de sua pesquisa de doutorado em Harvard. Após seu doutorado, Aiken reconheceu a importância de cálculos feitos por máquina. Começou a ler sobre o assunto, descobriu o trabalho de Babbage e, baseado neste, construiu em 1944 o seu primeiro computador com relés, o Mark I. Ele era capaz de armazenar 72 valores de 23 dígitos. Para entrada e saída, utilizava-se fita de papel perfurado e foi o primeiro computador de origem americana para propósitos gerais. Quando Aiken acabou de construir o sucessor, o Mark II, computadores com relés estavam obsoletos. A era eletrônica tinha começado.

Como conseqüência do esforço de guerra, o governo inglês colocou em operação, em 1943, o COLOSSUS. Tratava-se de um equipamento destinado a decifrar as mensagens codificadas por um dispositivo construído pela Alemanha, denominado ENIGMA. Devido à sua arquitetura muito específica, voltada para a elucidação de mensagens, o COLOSSUS foi um equipamento único, cujo maior mérito é ter sido o **primeiro computador digital**.

1. Primeira Geração – Válvulas (1945-1955)

Os equipamentos desenvolvidos até o início dos anos 40 eram fundamentados na utilização de relés eletromecânicos. A utilização de válvulas mecânicas foi um marco que caracterizou esta nova era, que inicia a história dos computadores eletrônicos digitais.

Esta geração começou com a construção do ENIAC (*Eletronic Numerical Integrator and Computer*) pelo engenheiro americano John Mauchley e seu aluno de pós-graduação J. Presper Eckert. O ENIAC surgiu a partir de um projeto apresentado ao Exército Americano em 1943 para a construção de um computador destinado a efetuar cálculos que seriam utilizados nas frentes de combate. As suas 18.000 válvulas eletrônicas, seus 1.500 relés, além de 800 quilômetros de fio utilizados, contribuíram para o peso de 30 toneladas e um enorme consumo de energia elétrica. Tudo isso era necessário para fazer funcionar uma arquitetura com vinte registradores, cada um capaz de conter números decimais de até dez dígitos, sendo capaz de realizar 100.000 operações aritméticas por segundo. A programação era elaborada através da utilização de chaves com múltiplas posições. O projeto somente foi concluído em 1946 e, por isso, não pôde ser utilizado nas operações militares da Segunda Guerra.

Maurice Wilkes, pesquisador da universidade de Cambridge, inspirado no trabalho de Mauchley e Eckert, construiu o EDSAC, em 1949, o primeiro computador eletrônico funcionando com programa armazenado. Isso significa que um programa não era mais representado por um conjunto de chaves posicionadas. As instruções do programa passavam a ser inseridas no equipamento, assim como os dados a serem processados, isto é, eram sinais elétricos armazenados numa parte que recebeu a denominação de **memória**.

Enquanto isso, Mauchley e Eckert trabalhavam no sucessor do ENIAC, denominado EDVAC (*Eletronic Discret Variable Automatic Calculator*). O projeto não foi a frente, em virtude de aspectos ligados à fundação de uma empresa fabricante de computadores, que após a evolução natural dos fatos deu origem à então denominada UNIVAC (hoje transformada em UNISYS, após fusão com a Burroughs). Surgiu, em 1951, o UNIVAC I, o primeiro computador eletrônico a ser usado comercialmente.

Os projetos para a construção de computadores surgiam a partir de concepções dos seus criadores, não havendo ainda linhas gerais de comportamento quanto a possíveis padrões de arquitetura. Entre todas as tentativas, merece especial atenção aquela apresentada por John von Neumann. Tendo participado da equipe que desenvolveu o ENIAC, além de suas idéias sobre computador de programa armazenado terem sido utilizadas no EDSAC, von Neumann conduziu um projeto chamado IAS no Instituto de Estudos Avançados de Princeton. A partir desse trabalho, o processo de desenvolvimento de computadores nunca mais seria o mesmo.

2) Segunda Geração – Transistores (1955-1965)

Durante os anos 50 (mais ou menos 1955), surgiu um novo dispositivo capaz de substituir a válvula eletrônica com vantagens: o **transistor**, marcando o início da segunda geração de computadores.

Alguns projetos criados na geração anterior foram redesenhados para utilizarem transistores em substituição às válvulas eletrônicas. Entre outros aspectos de relevância, o transistor abriu a perspectiva da redução significativa dos tamanhos dos circuitos eletrônicos digitais. Esse importante fato permitiu que surgissem computadores de menor tamanho sob o ponto de vista físico, tendo, porém, capacidades de processamento rivalizadas com aquelas dos de grande porte. Talvez o mais famoso tenha sido o PDP-1, fabricado em 1961, com 4 KB de memória principal, palavras de 18 bits e ciclo de 5 microssegundos. O seu preço estava na faixa de 120.000 dólares. Esse projeto evoluiu a ponto de, em poucos anos, surgir o PDP-8, um campeão de vendas na linha dos chamados **minicomputadores**. A grande inovação desse novo

modelo era a utilização de um recurso denominado *barramento unificado*, uma novidade na época na maneira de interligar os componentes internos do computador.

Entre os computadores de segunda geração cuja existência tornou-se digna de nota, não se pode deixar de mencionar o B5000, produzido pela empresa Burroughs, hoje com a denominação de UNISYS. A preocupação dos seus projetistas, ao contrário dos que atuavam nas outras empresas do gênero, não era com o hardware e sim com o software. O equipamento foi construído com a intenção de executar programas codificados em Algol 60, uma linguagem precursora do Pascal.

1. Terceira Geração – Circuitos Integrados (1965-1980)

As pesquisas sobre a utilização do transistor continuaram tendo resultado no surgimento de um novo componente, o **circuito integrado**. Essa foi a denominação atribuída ao dispositivo resultante da montagem de um conjunto de diversos transistores, interligados de acordo com um plano definido, em uma única pastilha de silício (surge, então, o ***chip***). A principal conseqüência disso foi a possibilidade imediata da construção de computadores mais rápidos, mais baratos e menores que seus ancestrais baseados apenas em transistores.

A partir dessa geração, uma empresa em particular experimentou notável expansão: a **IBM**. Uma nova linha de produtos foi anunciada apresentando arquitetura capaz de abranger tanto soluções para problemas comerciais quanto para problemas científicos, utilizando a nova tecnologia de circuitos integrados.

Quanto aos aspectos técnicos, os conceitos de **canal** e **multiprogramação** marcaram profundamente essa geração. O *canal* representou uma grande inovação na capacidade de ligação de dispositivos periféricos à CPU. Tratava-se de um pequeno computador, cuja programação era especificamente voltada a converter sinais entre a CPU e um ou mais periféricos. Eram distinguidos em duas categorias, a saber: *canal seletor*, para periféricos de alta velocidade e *canal multiplexador*, para periféricos de baixa velocidade.

Quanto à *multiprogramação*, esta representou uma inovação ainda mais arrojada. Tratava-se de um novo conceito de uso do computador, onde vários programas podiam ser carregados ao mesmo tempo na memória e disputar a utilização do processador. Por exemplo, o programa denominado **A** usava o processador durante um pequeno intervalo de tempo, salvava os resultados intermediários e em seguida o liberava para o próximo programa da fila de espera. Passando algum tempo, chegava novamente a vez do programa **A** utilizar o processador, repetindo-se tudo até o final da execução do mesmo. A velocidade com que tudo isso era feito dava a impressão de que todos os programas estavam funcionando ao mesmo tempo.

Ainda durante esse período, os *minicomputadores* ganharam importante espaço no mercado, pois eram equipamentos capazes de atender à maioria das necessidades das organizações, porém a custo bem menor que o de um equipamento de grande porte.

3) Quarta Geração – Computadores Pessoais e VLSI (a partir de 1980)

O avanço da tecnologia eletrônica digital foi gradativamente permitindo que se colocassem cada vez mais transistores num *chip*. Por volta de 1980, já era possível colocar milhões de transistores num único *chip*, devido à técnica chamada ***Very Large Scale Integration*** (***VLSI***). A conseqüência mais imediata desse fato foi a redução dos tamanhos e dos preços dos equipamentos, apesar do aumento das suas capacidades de armazenamento. Estava estabelecida a quarta geração de computadores.

A partir dessa época, profundas mudanças começaram a ocorrer nos hábitos dos usuários, em virtude da possibilidade da miniaturização dos componentes utilizados na construção de computadores. Durante muito tempo, os computadores de grande porte (conhecidos como ***mainframes***) foram utilizados por muitas organizações como apoio às suas necessidades de informatização, muito mais devido à falta de alternativas tecnológicas do que à real necessidade de usar equipamentos com aquele porte. Computadores de menor porte tornaram-se substitutos naturais para muitos casos onde outrora um *mainframe* era recomendado.

O advento dos chamados *microcomputadores*, contudo, produziu a mais profunda mudança já ocorrida na indústria de Informática. De maneira muito repentina, o usuário passou a ter nas mãos um grande poder de processamento, que contribuiu de modo definitivo para o processo de dissipar o mito que ainda existia em torno dos grandes computadores. Essa é a situação atual em que se encontra a tecnologia da Informática, estando, cada vez mais, ocorrendo a autonomia do usuário.

Capítulo 2

Organização Interna de Computadores

1) Conceitos Básicos no Mundo do Computador

Um computador digital é uma máquina capaz de nos solucionar problemas através da execução de instruções que lhe são fornecidas. Denomina-se **programa** uma seqüência de instruções que descreve como executar uma determinada tarefa. Os circuitos eletrônicos de cada computador podem reconhecer e executar diretamente um conjunto limitado de instruções simples para as quais todos programas devem ser convertidos antes que eles possam ser executados. Estas instruções básicas raramente são mais complicadas do que:

- Somar dois valores
- Verificar se um número é igual a zero
- Mover um dado de uma parte da memória do computador para outra

Juntas, as instruções primitivas do computador formam uma linguagem que torna possível as pessoas se comunicarem com o computador. Tal linguagem é denominada **linguagem de máquina**. Ao se projetar um novo computador deve-se decidir que instruções devem estar presentes nesta linguagem de máquina. Como a maioria das linguagens de máquina é muito simples, é difícil e tedioso utilizá-las.

Este problema pode ser atacado de duas maneiras principais, ambas envolvendo o projeto de um novo conjunto de instruções de uso mais conveniente para as pessoas do que o conjunto de instruções embutidas na máquina. Juntas, estas novas instruções formam uma linguagem que chamaremos de L2, exatamente como as instruções embutidas na máquina formam uma linguagem, que chamaremos de L1. Os dois conceitos diferem na maneira com que os programas escritos em L2 são executados pelo computador, que, afinal de contas, pode apenas executar programas escritos em linguagem de máquina, L1.

Um método de execução de um programa escrito em L2 consiste, em primeiro lugar, em substituir cada instrução por uma seqüência equivalente de instruções L1. O programa resultante é composto inteiramente de instruções L1. O computador então executa o novo programa em L1 em vez do programa em L2. Esta técnica é conhecida como **tradução**.

A outra técnica consiste em receber o programa escrito em L2 como dado de entrada e efetuar a execução examinando uma instrução de cada vez e executando a seqüência equivalente de instruções L1 diretamente. Esta técnica não requer a geração de um novo programa em L1. Denomina-se **interpretação**, e o programa que a executa denomina-se **interpretador**. Vamos ver estes conceitos mais detalhadamente no capítulo 3.

A tradução e a interpretação são similares. Em ambos os métodos, as instruções em L2 são executadas pelas seqüências equivalentes de instruções em L1. A diferença é que, na tradução, o programa inteiro em L2 é primeiramente convertido para um programa em L1, o programa em L2 é abandonado, e então o novo programa em L1 é executado. Na interpretação, depois de cada instrução L2 ser examinada e decodificada, ela é executada imediatamente. Nenhum programa traduzido é gerado.

Os programas escritos na linguagem de máquina de um computador podem ser executados diretamente pelos circuitos eletrônicos do computador, sem quaisquer interpretadores ou tradutores intermediários. Esses circuitos eletrônicos, juntamente com a memória e os dispositivos de entrada/saída, constituem o ***hardware*** do computador. O *hardware* é composto de objetos tangíveis – circuitos integrados, placas de circuito impresso, cabos, fontes de alimentação, memórias, leitoras de cartões, impressoras, terminais, etc. – em lugar de idéias abstratas, algoritmos ou instruções.

O ***software***, ao contrário, consiste em **algoritmos** (instruções detalhadas que dizem como fazer algo) e suas representações para o computador – ou seja, os **programas**. Os programas podem estar representados em cartões perfurados, fitas magnéticas, discos e outros meios, mas a essência do *software* está no conjunto de instruções que constitui os programas, não nos meios físicos sobre os quais eles estão gravados.

Uma forma intermediária entre o *hardware* e o *software* é o ***firmware***, que consiste no *software* embutido em dispositivos eletrônicos durante a fabricação. O *firmware* é utilizado quando se espera que os programas raramente ou nunca serão mudados, por exemplo, em brinquedos ou instrumentos.

Um tema central dentro da computação que também será adotado por nós é:

Hardware e software são logicamente equivalentes.

Qualquer operação efetuada pelo *software* pode também ser implementada diretamente no *hardware*, e qualquer instrução executada pelo *hardware* pode também ser simulada pelo *software*. A decisão de se colocar certas funções em *hardware* e outras em *software* baseia-se em fatores tais como: custo, velocidade, confiabilidade e freqüência esperada de alterações.

2) Organização Interna de Computadores

Um computador é formado por:

- Processadores
- Memórias
- Dispositivos de entrada e saída

Esses três componentes são conectados através de barramentos através dos quais se faz possível a comunicação. A organização de um computador simples, orientado a barramento é mostrada na Figura 2.1.

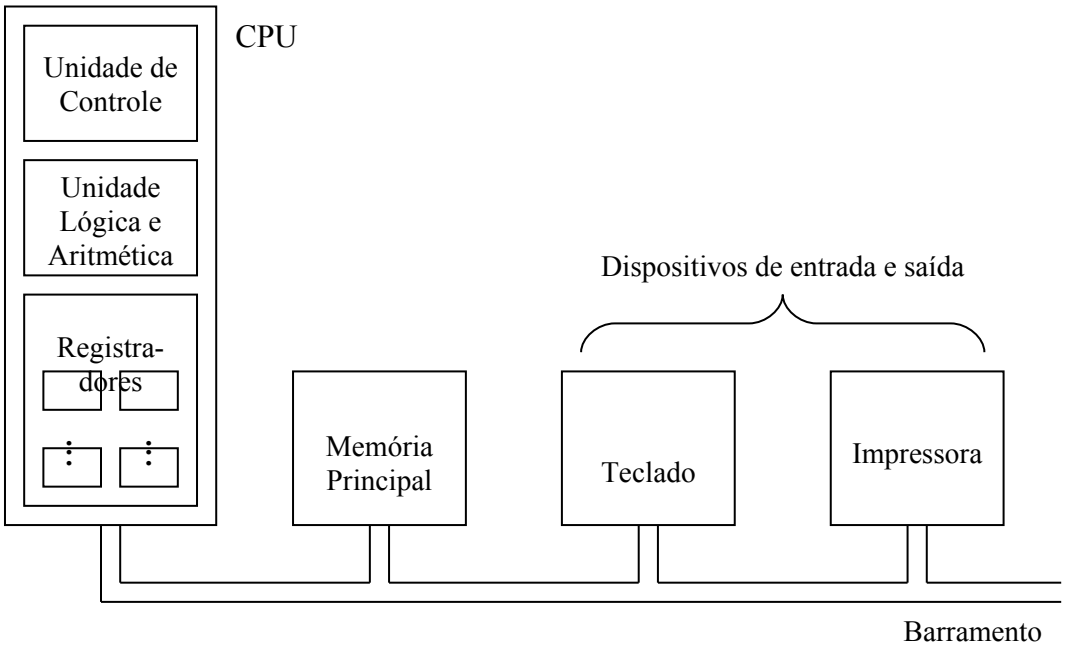


Figura 2.1: A organização de um computador simples com uma CPU e dois dispositivos de E/S

2.1)Processadores (CPU)

A **unidade central de processamento** (***Central Processing Unit - CPU***) é o "cérebro" do computador. Sua função é executar programas armazenados na memória principal, buscando suas instruções, examinando-as, e então executando uma após a outra.

A CPU é composta por várias partes distintas. A **unidade de controle** (***UC***) é responsável pela busca de instruções da memória principal e determinação de seus tipos. A **unidade lógica e aritmética** (***ALU***) faz operações (tais como *adição* e o *ou*) necessárias à execução das instruções. A CPU contém ainda uma memória pequena, de alta velocidade, usada para armazenar dados (entrada ou saída), resultados temporários obtidos durante a execução das instruções e certas informações de controle. Esta memória é formada por um conjunto de **registradores**, cada qual com uma função específica.

O registador mais importante é o **contador de programa** (***program counter - PC***), que aponta para a próxima instrução a ser executada. O **registrador de instrução** (***instruction register - IR***) é também importante. Ele contém a instrução que está sendo executada. A maioria dos computadores tem outros registradores para, por exemplo, o armazenamento de resultados intermediários.

2.1.1)Execução de Instruções

A CPU executa cada instrução através de uma série de passos:

1. Buscar a próxima instrução da memória para o IR.
2. Atualizar o PC, para que ele aponte para a próxima instrução.
3. Determinar o tipo de instrução.
4. Se a instrução usar dados de memória, determinar onde eles estão.
5. Buscar os dados (se houver algum) para registradores internos da CPU.
6. Executar a instrução.
7. Armazenar os resultados em locais apropriados.
8. Voltar ao passo 1 para iniciar a execução da próxima instrução.

Esta seqüência de passos é freqüentemente referida como o ciclo **busca-decodifica-executa**. Ela é o centro da operação de todos os computadores.

2.1.2)Organização da CPU

A organização interna de parte de uma CPU von Neumann é mostrada na Figura 2.2. Esta parte é chamada de **fluxo de dados** e é constituído por registradores e pela ALU.

Os registradores alimentam os dois registradores de entrada da ALU, denominados, na figura, de registradores *A* e *B*. Estes registradores mantêm as entradas da ALU enquanto ela está executando a operação.

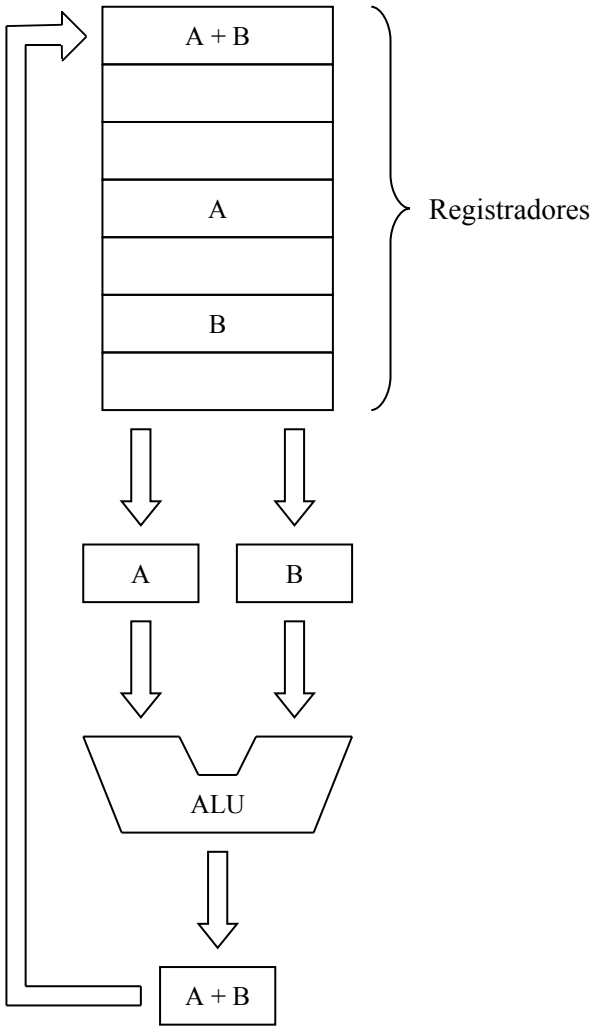


Figura 2.2: O fluxo de dados para uma típica máquina von Neumann

A própria ALU executa adição (como na figura), subtração e outras operações simples com suas entradas, produzindo um resultado no seu registrador de saída. Este resultado pode ser armazenado de volta em um registrador da CPU e, de lá, de volta à memória, se desejado.

As instruções podem ser divididas em três categorias:

- Registrador-memória – essas instruções permitem que dados sejam buscados da memória para registradores, onde podem ser usados como entradas para a ALU em instruções seguintes, por exemplo.
- Registrador-registrador – um exemplo típico deste tipo de instrução consiste em buscar dois operandos dos registradores, levá-los para os registradores de entrada da ALU, executar alguma operação com eles e armazenar o resultado de volta num registrador (situação ilustrada na Figura 2.2).
- Memória-memória – sua execução é similar à execução de uma instrução registrador-registrador.

2.2)Memória Principal

A *memória* é a parte do computador onde programas e dados são armazenados. Consiste no local de onde a CPU apanha os dados a serem processados, onde guarda valores intermediários e para onde envia os resultados finais do processamento, sendo assim, um elemento básico para o funcionamento da CPU. Sem uma memória de onde a CPU possa ler e escrever informações não existiriam computadores de programa armazenado.

A unidade básica de armazenamento é o digito binário, chamado **bit** (***bi****nary digi****t***). Um bit pode conter os valores 0 ou 1.

Principais Tipos de Memórias

- RAM (*Random Access Memory – Memória de Acesso Direto*): memórias nas quais as operações de escrita e leitura são possíveis.
- ROM (*Read-Only Memory – Memória Somente de Leitura*): os dados presentes nessa memória não podem ser modificados sob quaisquer circunstâncias. Tais dados são inseridos na memória na hora de sua fabricação e a única forma de alterar os seus dados é a sua substituição.

Endereços de Memória

As memórias são compostas de um determinado número de **células** (ou **posições**), cada uma podendo armazenar uma parte da informação. Cada célula tem um número, chamado de seu **endereço**, pelo qual os programas podem referenciá-la. Se a memória tem *n* células, elas terão endereços de *0* a *n-1*. Todas as células em uma memória possuem o mesmo número de bits. A figura 2.3 mostra três organizações diferentes para uma memória de 96 bits. Note que células adjacentes têm endereços consecutivos.

Um *byte* consiste num conjunto de 8 *bits*. *Bytes*, por sua vez, podem ser agrupados em *palavras* (*words*), sendo que o número de *bytes* por *palavra* pode variar de computador para computador.

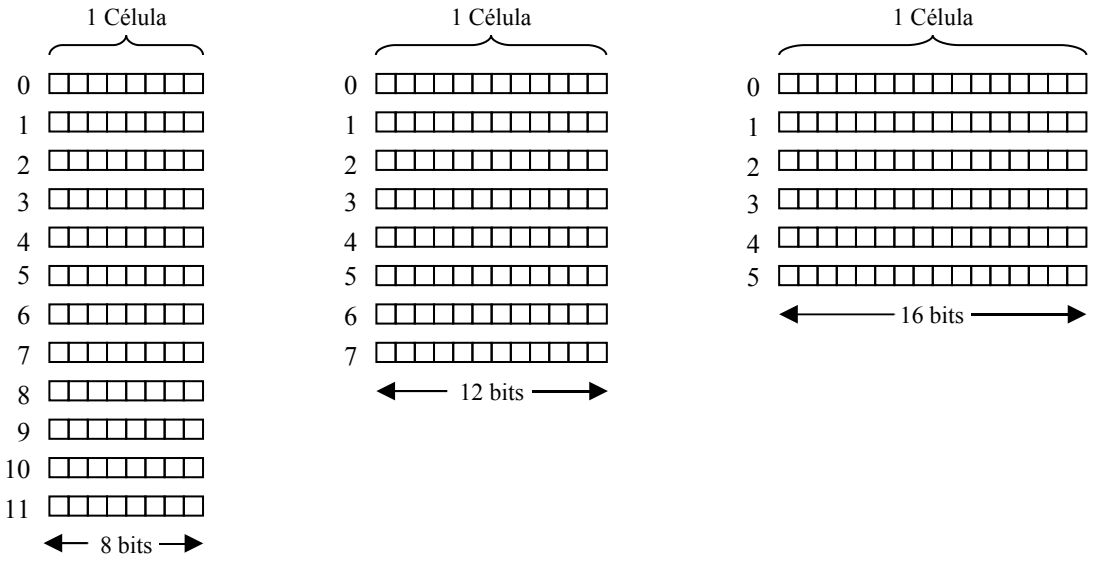


Figura 2.3: Três formas de organizar uma memória de 96 bits

O significado da célula é que ela é a menor unidade endereçável. Até recentemente, a maioria dos fabricantes de computador padronizavam a célula de 8 bits que, como já visto, é chamada de *byte*. Com a necessidade de maior espaço de endereçamento, alguns fabricantes começaram a utilizar a *word* como unidade endereçável, onde uma *word* pode ser igual a um ou mais bytes, dependendo do fabricante.

A unidade de medida de memória é igual a *1 byte*. A seguir, mostramos as relações entre a unidade de medida de memória e seus múltiplos:

Múltiplo	**Abreviatura**	**Valor**
1 Kilobyte	1KB	2^{10} B = 1024 B
1 Megabyte	1MB	2^{20} B = 1024 KB
1 Gigabyte	1GB	2^{30} B = 1024 MB
1 Terabyte	1TB	2^{40} B = 1024 GB

2.3) Memória Secundária

Como toda palavra na memória (principal) precisa ser diretamente acessível em um intervalo de tempo muito curto, ela é relativamente cara. Consequentemente, os computadores possuem também *memórias secundárias* que são memórias mais lentas, baratas e também são muito grandes. Tais memórias são utilizadas para armazenar grande quantidade de dados que a memória principal não é capaz de suportar.

i) Fitas Magnéticas

A fita magnética foi o primeiro tipo de memória secundária. Uma unidade de fita de computador é análoga a um gravador de rolo convencional: uma fita é enrolada em um rolo alimentador, passa por uma cabeça de gravação e chega a um rolo receptor. Variando a corrente

na cabeça de gravação, o computador pode gravar informações na fita em forma de pequenos pontos magnetizados.

A Figura 2.4 mostra como a informação é organizada em uma fita magnética. Cada ***frame*** possui um byte, além de um bit extra, chamado de **bit de paridade**, para aumentar a confiabilidade. Após a unidade de fita terminar a gravação de um **registro físico** (uma seqüência de *frames*), ela deixa um espaço na fita, enquanto reduz a velocidade. Se o programa escreve pequenos registros físicos na fita, a maior parte da fita será gasta por esses espaços, denominados ***gaps***. A utilização da fita pode ser mantida alta gravando-se registros físicos bem maiores que o *gap*.

As fitas magnéticas são dispositivos de acesso seqüencial. Se a fita está posicionada no início, para ler o registro *n* é necessário ler antes os registros físicos de *1* a *n-1*, um de cada vez. Se a informação desejada se encontra próxima ao fim da fita, o programa terá que ler quase a fita inteira, o que poderá levar vários minutos. Forçar a CPU, que pode executar milhões de instruções por segundo, a esperar, por exemplo, 200 segundos enquanto a fita avança, é um desperdício. Por isso, as fitas são mais adequadas quando os dados precisam ser acessados seqüencialmente.

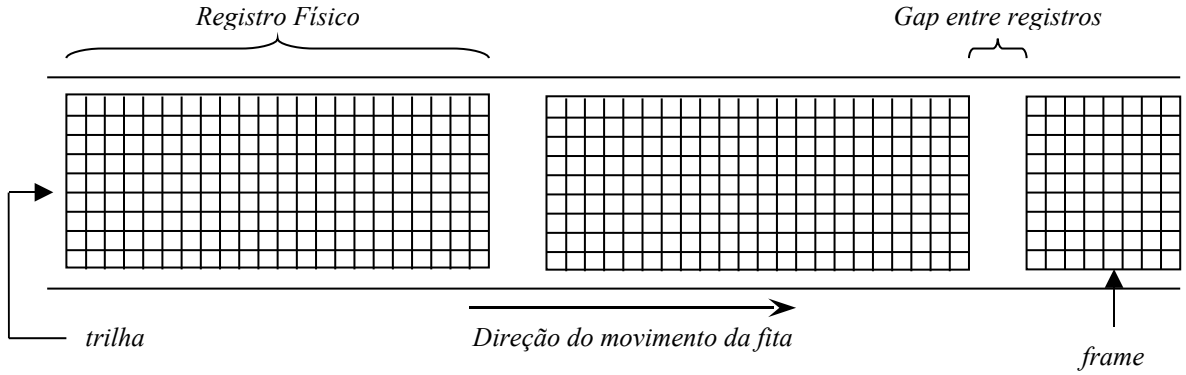


Figura 2.4: A informação é gravada na fita como uma seqüência de matrizes retangulares de bits

ii) Discos Rígidos

Um **disco** é um pedaço metal ao qual uma cobertura magnetizável foi aplicada no processo de fabricação, geralmente de ambos os lados. A informação é gravada em um certo número de círculos concêntricos chamados **trilhas** (veja Figura 2.5). Os discos possuem tipicamente entre 40 e algumas centenas de trilhas por superfície. Cada unidade de disco possui uma cabeça móvel que pode aproximar-se ou afastar-se do centro do disco. A cabeça é suficientemente pequena para ler ou gravar informações exatamente em uma trilha. A unidade de disco possui freqüentemente vários discos empilhados verticalmente. Em tal configuração, o braço de leitura terá uma cabeça próxima a cada superfície, e todas se movem juntas. A posição radial das cabeças (distância do eixo) é chamada **cilindro**. Uma unidade de disco com *n* pratos (e *2n* faces) terá *2n* cabeças, e, portanto *2n* trilhas por cilindro.

As trilhas são divididas em **setores**, e há normalmente entre 10 e 100 setores por trilha. Um setor consiste em um certo número de bytes, tipicamente 512.

Para especificar uma transferência, o programa deve fornecer as seguintes informações:

- O cilindro e a superfície, que juntos definem uma única trilha;
- O número do setor onde começa a informação;
- O número de palavras a serem transferidas;

- O endereço na memória principal de onde a informação vem ou para onde vai;
- Se a informação é para ser lida do disco ou se é para ser escrita nele.

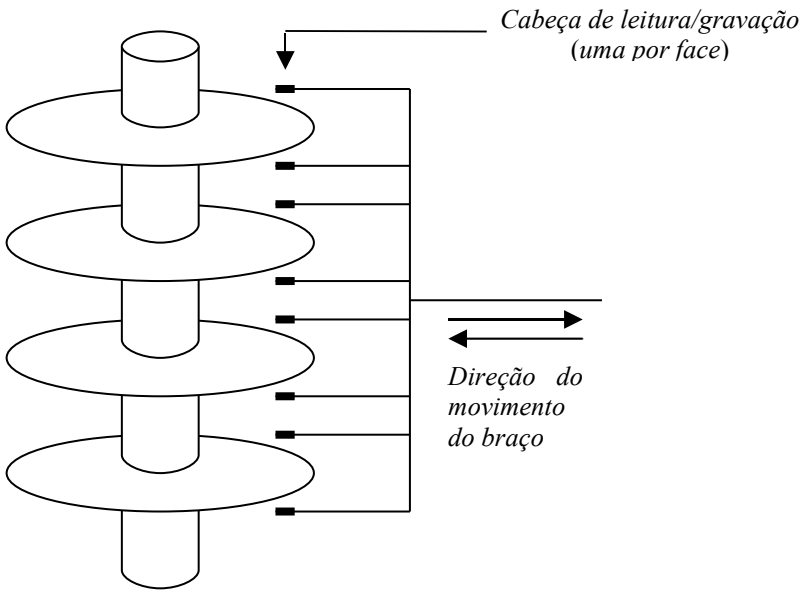


Figura 2.5: Um disco com quatro pratos

As transferências de informação no disco sempre começam no início de um setor, nunca no meio. Se uma transferência de múltiplos setores ultrapassa o limite de uma trilha dentro de um cilindro, nenhum tempo é perdido porque a mudança de uma cabeça para outra é feita eletronicamente. Entretanto, se a transferência atravessa o limite de um cilindro, o tempo de uma rotação pode ser perdido no reposicionamento das cabeças no próximo cilindro e na espera pelo setor 0.

Se a cabeça de leitura está sobre um cilindro errado, ela precisa primeiramente ser movida. Este movimento é chamado ***seek*** (procura), e o seu tempo é denominado ***seek time***. Uma vez que a cabeça está posicionada corretamente, é necessário esperar até que o setor inicial passe sob a cabeça, antes de iniciar a transferência. Este tempo, por sua vez, é conhecido como **latência rotacional**. O tempo total de acesso é igual ao *seek time* mais a latência rotacional mais o tempo de transferência.

Quase todos os computadores utilizam discos com vários pratos, como descrito anteriormente, para o armazenamento de dados. Eles são freqüentemente denominados **discos rígidos** (***hard disks - HD***). O tipo mais conhecido é o **disco *winchester***, que é uma unidade selada (para evitar contaminação com poeira). As cabeças em uma unidade *winchester* têm forma aerodinâmica e flutuam em um colchão de ar gerado pela rotação dos pratos.

iii) Discos Óticos

Nos últimos anos, discos óticos (em oposição aos magnéticos) tornaram-se disponíveis. Tais discos foram inicialmente desenvolvidos para gravar programas de televisão, mas eles logo passaram a também ser utilizados como dispositivos de armazenamento de computadores.

Tais discos possuem uma capacidade de armazenamento muito superior aos discos magnéticos e devido a esta capacidade, os discos óticos têm sido objeto de grande pesquisa e vêm sofrendo uma evolução incrivelmente rápida.

A primeira geração de discos óticos se baseia na mesma tecnologia dos *compact discs* usados em áudio, e os discos presentes nesta geração são chamados ***CD-ROMs*** (***Compact Disk Read Only Memory***).

Informações em um CD-ROM são escritas como um espiral único contínuo, diferentemente dos discos magnéticos com suas trilhas e cilindros discretos.

Os CD-ROMs são potencialmente úteis na distribuição de grandes bases de dados, especialmente aquelas que misturam textos e imagens. Uma biblioteca completa de leis, medicina ou literatura de 250 livros grossos cabe facilmente em um CD-ROM, assim como uma enciclopédia totalmente ilustrada. Um curso por computador formado por milhares de *slides* coloridos, cada qual acompanhado por 10 segundos de narração, é outro candidato para CD-ROM.

Apesar desse enorme potencial, os CD-ROMs não são escrevíveis, o que limita sua utilidade como dispositivos de armazenamento de computadores. A intenção de se ter um meio escrevível conduziu à próxima geração: os discos óticos ***WORM*** (***Write Once Read Many***). Este dispositivo permite que os usuários escrevam informações nos discos óticos, mas isso só pode ser feito uma vez. Tais discos são bons para arquivar dados, relatórios, e outras informações que são (semi) permanentes. Eles não são muito apropriados para guardar arquivos temporários de rascunho. Mas, dada a grande capacidade desses discos, o estilo descartável de apenas acrescentar arquivos temporários até que o disco esteja cheio, e então jogá-lo fora, é aceitável.

É claro que a existência de discos que só podem ser escritos uma vez provoca um impacto na forma em que os programas são escritos.

A terceira fase de evolução dos discos óticos é o **meio ótico apagável**.

Discos óticos escrevíveis não irão substituir os discos *winchester* convencionais por algum tempo, por duas razões. Primeiro, o seu *seek time* é muito superior ao dos discos *winchester*. Depois, sua taxa de transferência é bem inferior à apresentada pelos discos magnéticos. Por isso, o desempenho dos discos magnéticos é simplesmente muito melhor. Enquanto os discos óticos irão sem dúvida melhorar com o tempo, os discos magnéticos irão, provavelmente, melhorar tão rápido quanto, mantendo-se na frente.

Capítulo 3
Sistemas Operacionais

As atuais aplicações dos computadores exigem, freqüentemente, que uma única máquina execute tarefas que possam competir pela posse dos recursos disponíveis na máquina. Por exemplo, uma máquina pode ser conectada a vários terminais ou estações de trabalho atendendo, simultaneamente, a diversos usuários. Até mesmo em computadores pessoais, os usuários podem executar várias atividades ao mesmo tempo, tais como a impressão de um documento, a modificação de outro e a criação de um gráfico a ser inserido em algum trabalho. Este tipo de utilização exige um alto grau de coordenação para evitar que atividades independentes interfiram umas nas outras, garantindo ainda uma comunicação eficiente e confiável entre atividades interdependentes. Tal coordenação é efetuada por um programa denominado *Sistema Operacional*.

1) Evolução dos Sistemas Operacionais

→ Sistemas Monoprocessados:

As máquinas de um único processador, das décadas de quarenta e cinqüenta, não eram flexíveis nem muito eficientes. A execução de programas exigia uma considerável preparação de equipamentos, incluindo a montagem de fitas e a colocação física de cartões perfurados em leitoras de cartão, o posicionamento de chaves, e assim por diante. Logo, a exececução de cada programa, chamada de *job*, se tornava uma tarefa bem desgastante. Quando vários usuários desejavam compartilhar uma mesma máquina, era comum o emprego de folhas de reserva de horário. Durante o período de tempo alocado a um usuário, a máquina ficava totalmente sob sua responsabilidade e controle. A sessão normalmente começava com a instalação do programa, seguida de curtos períodos de execução do mesmo, e era freqüentemente finalizada com um esforço desesperado de fazer alguma coisa adicional, enquanto o usuário seguinte já começava a exibir indícios de impaciência.

Em ambientes assim, os sistemas operacionais começaram a surgir como ambientes de software destinados a simplificar a instalação dos programas do usuário e a tornar mais suave a transição de um *job* e outro. Um primeiro progresso consistiu na separação entre usuários e equipamentos, eliminando a movimentação física de pessoas para dentro e para fora da sala do computador. Para isso, um operador de computadores era contratado para executar a operação propriamente dita da máquina. Aos que desejassem usar o computador, era solicitado que os programas a serem executados fossem submetidos ao operador, juntamente com os dados necessários e com eventuais instruções especiais sobre a operação do programa, devendo o usuário retornar posteriormente para receber os resultados. O operador, por sua vez, transferia esse material para a memória, de onde o sistema operacional poderia acessá-los, para promover

sua execução. Os *jobs* presentes na memória aguardavam para serem executados em uma *fila de jobs* (*processamento em batch/lotes*).

A principal desvantagem dos tradicionais processamentos em lotes é a falta de interação do usuário com o programa uma vez submetido este último à fila de *jobs*. Este procedimento é aceitável para algumas aplicações, como é o caso do processamento de folhas de pagamento, em que os dados e todas as decisões do processo estão estabelecidos a priori. Contudo, não é aceitável quando o usuário necessita interagir com o programa durante a sua execução. Como exemplo, pode-se citar os sistemas de reserva de passagens ou similares, pois tanto as reservas como os cancelamentos devem ser notificados assim que ocorrerem. Outro exemplo corresponde aos sistemas de processamento de texto, pois neles os documentos são escritos e reescritos dinamicamente. Um outro exemplo ainda são os jogos para computadores, para os quais a interação com a máquina é a característica mais significativa.

Para satisfazer essas necessidades, novos sistemas operacionais foram desenvolvidos, capazes de proporcionar tal *processo interativo*. Esses sistemas permitem a execução de programas que mantenham diálogo com o usuário através de terminais ou estações de trabalho. Estes sistemas interativos originaram o conceito conhecido como *processamento em tempo real*. Essa denominação se deve à necessidade de coordenação das atividades executadas pela máquina com outras que se passam no ambiente em que a máquina está imersa. Insatisfeitos com a espera por toda uma noite pelos resultados dos seus *jobs*, os atuais usuários passaram a exigir uma resposta muito rápida do sistema, com o qual se comunicam remotamente a partir de suas estações de trabalho.

Se os sistema interativos permitissem atender a apenas um usuário de cada vez, o processamento em tempo real não deveria apresentar qualquer problema. Porém, computadores eram caros, e, por isso, cada um deles deveria servir simultaneamente a mais de um usuário. Por outro lado, sendo comum o fato de vários usuários solicitarem serviços interativos ao computador ao mesmo tempo, as características exigidas de um sistema de tempo real passaram a constituir um obstáculo concreto. Se o sistema operacional, neste ambiente de multiusuários, insistisse em executar somente um *job* de cada vez, um único usuário apenas acabaria recebendo um atendimento satisfatório em tempo real.

Uma solução para este problema podera ser trazida por um sistema operacional que revezasse a execução dos vários *jobs* através de um processo denominado *compartilhamento de tempo* (*time-sharing*). Mais precisamente, *time-sharing* refere-se à técnica de dividir o tempo em intervalos ou fatias (*time slices*), e restringir a execução, dentro de cada uma dessas frações de tempo, a um *job* de cada vez. Ao término de cada um desses intervalos, o *job* corrente é retirado do processamento e um outro é acionado. Revezando rapidamente, desta maneira, a execução dos *jobs*, cria-se a ilusão de que vários *jobs* estão sendo executados simultaneamente. Dependendo dos tipos de *jobs* que estivessem sendo executados, os antigos sistemas *time-sharing* chegaram a atender simultaneamente cerca de 30 usuários, com uma aceitável resposta em tempo real.

Atualmente, a técnica de *time-sharing* é utilizada tanto em sistemas com um único usuário como também em ambientes multiusuários, embora o primeiro seja, usualmente, denominado *multitasking*, referindo-se à ilusão que propiciam de haver mais de uma tarefa sendo executada ao mesmo tempo. Independentemente do número de usuários do ambiente, constatou-se que o conceito de *time-sharing* promovia o aumento da eficiência global de uma máquina. Esta constatação mostrou-se particularmente surpreendente ao se levar em conta o considerável processamento adicional exigido para a implementação do revezamento que caracteriza a técnica de *time-sharing*. Por outro lado, na ausência de *time-sharing*, um computador acaba gastando mais tempo enquanto espera que seus dispositivos periféricos completem suas tarefas, ou que um

usuário faça sua próxima solicitação ao sistema. Em ambientes com *time-sharing* este tempo pode ser cedido a alguma outra tarefa. Assim, enquanto uma tarefa espera pela sua vez de utilizar o processador, a outra pode prosseguir a sua execução. Como resultado, em um ambiente *time-sharing*, um conjunto de tarefas pode ser concluído em tempo menor do que o faria se executado de modo seqüencial.

→ Sistemas Multiprocessados:

Mais recentemente, a necessidade de compartilhar informação e recursos entre diferentes máquinas suscitou o desejo de unir tais máquinas para o intercâmbio de informação. Para preencher tal necessidade, popularizaram-se os sistemas com computadores interconectados, conhecidos como *redes de computadores*. De fato, o conceito de uma máquina central grande, servindo a muitos usuários, foi substituído pelo conceito de muitas máquinas pequenas, conectadas por uma rede na qual os usuários compartilham recursos espalhados pela rede (tais como serviço de impressão, pacotes de software, equipamentos de armazenamento de dados). O principal exemplo é a *Internet*, uma rede de redes que hoje une milhões de computadores do mundo todo.

Muitos dos problemas de coordenação que ocorrem em projetos de redes são iguais ou semelhantes aos enfrentados pelos sistemas operacionais. De fato, o software de controle de rede pode ser visto como um sistema operacional projetado para grandes redes. Deste ponto de vista, o desenvolvimento de software de redes é uma extensão natural do campo dos sistemas operacionais. Enquanto as redes mais antigas foram construídas como um conjunto de computadores individuais, levemente interligados, cada qual sob o controle do seu próprio sistema operacional, as recentes pesquisas na área de redes estão se concentrando nos sistemas estruturados como grandes redes, cujos recursos são igualmente compartilhados entre as tarefas presentes na rede. Essas tarefas são designadas para serem executadas nos processadores da rede, de acordo com a necessidade, sem levar em consideração a real posição física de tais processadores.

As redes representam apenas um exemplo dos projetos de multiprocessadores que estão inspirando o desenvolvimento dos modernos sistemas operacionais. Enquanto uma rede produz um sistema multiprocessado mediante a combinação de máquinas, cada qual contendo apenas um único processador, outros sistemas multiprocessadores são projetados como computadores únicos, contendo, porém, mais de um processador. Um sistema operacional para tais computadores não apenas coordenará a competição entre as várias tarefas que são de fato executadas simultaneamente, mas também controlará a alocação de tarefas aos diversos processadores. Este processo envolve problemas de *balanceamento de carga* (para garantir que os processadores sejam utilizados de modo eficiente) bem como *escalonamento* (divisão das tarefas em várias subtarefas, cujo número seja compatível com o número de processadores disponíveis na máquina).

Desta forma, o desenvolvimento de sistemas multiprocessados criou dimensões adicionais no estudo de sistemas operacionais, uma área que deverá manter-se em franca atividade pelos anos vindouros.

2) Arquitetura dos Sistemas Operacionais

Para entender a arquitetura de um sistema operacional típico, é útil analisar todos os tipos de software que ele contém. Para tanto, iniciemos com um estudo de como agrupar programas segundo uma classificação. Tal classificação invariavelmente, separa em diferentes classes, módulos semelhantes de software, da mesma forma como os fusos horários vizinhos definem, no

relógio, mesmo para comunidades geograficamente próximas, a diferença de uma hora, mesmo que não haja, entre tais localidades, diferenças significativas no horário solar. Além disso, no caso de classificação de software, a dinâmica e a falta de uma autoridade definitiva no assunto conduzem a classificações e terminologias contraditórias. Logo, a classificação seguinte (Figura 3.1) deveria ser vista mais como uma forma de enfrentar um assunto complexo do que como uma taxonomia universalmente aceita.

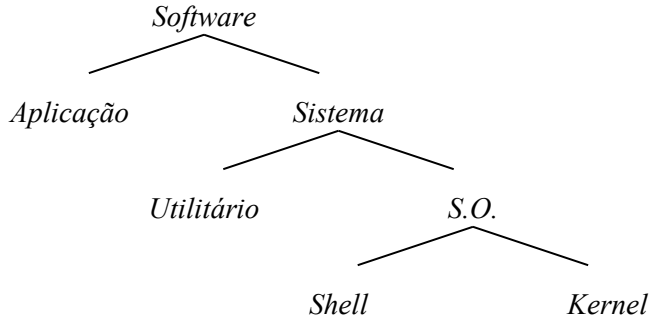


Figura 3.1: Classificação de software

→ Software:

Primeiramente, dividamos o software de um computador em duas grandes categorias: *software aplicativo* e *software de sistema*. Software aplicativo consiste de programas que executam determinadas tarefas no computador. Um computador utilizado para manter o cadastro de uma companhia industrial conterá softwares aplicativos diferentes dos encontrados em um computador de um autor de livros didáticos. Exemplos de software aplicativo incluem planilhas eletrônicas, sistemas de banco de dados, editores de textos, jogos e software de desenvolvimento de programas.

Em contraste com o software aplicativo, um software de sistema executa aquelas tarefas que são comuns nos sistemas computacionais em geral. Num certo sentido, o software de sistema desenha o ambiente em que se desenvolve o software aplicativo.

Na classe de software de sistema, podem ser consideradas duas categorias, uma das quais consiste do próprio *sistema operacional*, e a outra, de módulos de software do tipo conhecido como *utilitário*. A maioria dos softwares de instalação consiste de programas que executam tarefas, embora não incluídas no sistema operacional, essenciais à instalação de algum software no computador. De certo modo, softwares utilitários consistem de módulos de software que ampliam as capacidades do sistema operacional. Por exemplo, a capacidade de efetuar a formatação de um disco ou a cópia de um arquivo não são, em geral, implementadas pelo próprio sistema operacional, mas sim por programas utilitários. Outros exemplos de softwares utilitários incluem os programas para comunicação por modem, para linhas telefônicas, software para mostrar a previsão do tempo na tela, e, em número cada vez mais crescente de máquinas, o software de tratamento das atividades relacionadas com as redes de computadores.

A distinção entre software aplicativo e software utilitário não é muito nítida. Muitos usuários consideram que a classe de software utilitário deve incluir todos os softwares que acompanham o sistema operacional por ocasião de sua compra. Por outro lado, eles também classificam os sistemas de desenvolvimento de programas como software utilitário, desde que tais pacotes sempre acompanhem o sistema operacional por ocasião de sua venda. A distinção entre o software utilitário e o sistema operacional é igualmente vaga. Alguns sistemas consideram os softwares que executam serviços básicos, tais como listar os arquivos do disco,

como sendo softwares utilitários; outros os incluem como partes integrantes do sistema operacional.

É fácil verificar que, implementando-se certos programas na forma de software utilitário, o sistema operacional resulta mais simples do que seria se tivesse de contabilizar todos os recursos básicos exigidos pelo sistema computacional. Além disso, as rotinas implementadas como software utilitário podem ser mais facilmente personalizadas de acordo com as necessidades de uma particular instalação. Não é incomum encontrar companhias ou indivíduos que tiveram que alterar ou ampliar algum dos softwares utilitários que acompanham originalmente o sistema operacional.

→ *Shell*:

A parte do sistema operacional que define a interface entre o sistema operacional e seus usuários é chamada *shell*. O trabalho do shell é o de prover uma comunicação natural com os usuários do computador. Os shells modernos executam tal função por meio de uma *interface gráfica com o usuário* na qual os objetos a serem manipulados, tais como arquivos e programas, são representados através de ícones. Essas interfaces permitem que os usuários solicitem a execução de comandos ao sistema apontando e deslocando tais itens com o auxílio do mouse. Os shells mais antigos se comunicavam com os usuários via teclado e monitor, através de mensagens textuais.

Embora o shell de um sistema operacional represente um papel importante na definição da funcionalidade de um computador, ele é somente uma interface entre o usuário e o verdadeiro núcleo do sistema operacional, como mostrado na Figura 3.2. Esta distinção entre o shell e as partes internas do sistema operacional é enfatizada pelo fato de alguns sistemas operacionais permitirem ao usuário selecionar, entre diversos shells, aquele que lhe for mais adequado. Usuários do sistema operacional UNIX, por exemplo, podem selecionar uma variedade de shells, incluindo o *Borne shell*, o *C Shell* e o *Korn shell*. Versões mais antigas do *Windows* eram, essencialmente, shells de substituição ao *MS-DOS*. Em casos como esse, o sistema operacional permanece o mesmo, exceto quanto à forma em que se comunica com o usuário.

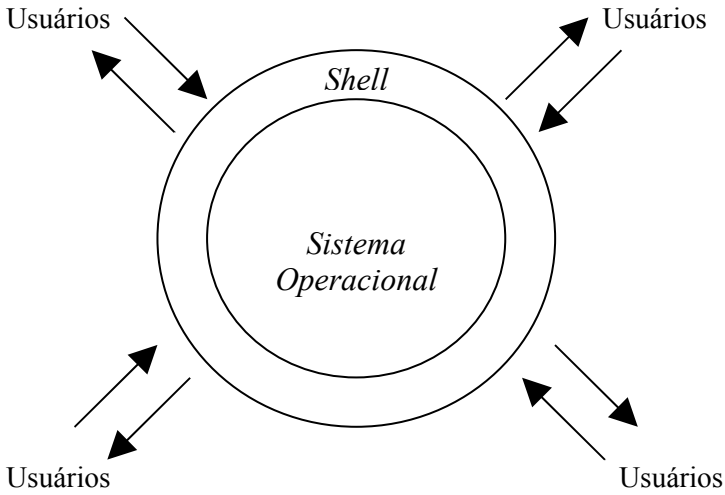


Figura 3.2: O shell como interface entre o usuário e o sistema operacional

→ *Kernel* (Núcleo):

A parte interna de um sistema operacional é, em geral, chamada *kernel*. O kernel de um sistema operacional contém os componentes de software que executam as funções mais básicas necessárias ao funcionamento de cada instalação computacional em particular. Um destes módulos básicos é o *gerenciador de arquivos*, cuja função é coordenar o uso dos recursos de armazenamento de massa do computador. Mais precisamente, o gerenciador de arquivos mantém "cópias" de todos os arquivos armazenados no dispositivo de armazenamento, mantém informação sobre a localização de cada arquivo, sobre os usuários autorizados a acessar os diversos arquivos e sobre as áreas disponíveis no dispositivo de armazenamento, para novos arquivos ou para a extensão de arquivos existentes.

Para auxiliar os usuários, a maioria dos gerenciadores de arquivos permite que estes sejam agrupados em conjuntos chamados *pastas* ou *diretórios* (*folders*). Tal procedimento permite ao usuário organizar seus arquivos de acordo com as respectivas finalidades, agrupando em cada mesmo diretório arquivos referentes a um mesmo assunto. Além disso, é possível criar uma organização hierárquica, possibilitando que cada diretório possa conter, por sua vez, subdiretórios. Uma seqüência de aninhamentos de níveis de diretórios é denominada *path* ou *caminho*.

Qualquer acesso a arquivos, por parte de algum módulo de software, é efetuado através do gerenciador de arquivos. O procedimento inicial consiste em solicitar ao gerenciador a autorização para fazer acesso ao arquivo. Este procedimento é conhecido como "abrir o arquivo". Se o gerenciador de arquivos aceitar o pedido, ele fornecerá a informação necessária para encontrar e manipular tal arquivo. Tal informação é mantida em uma área da memória principal denominada *descritor de arquivos*. É com base na informação contida nesse descritor de arquivo que operações elementares individuais são executadas sobre o arquivo.

Outro componente do *kernel* corresponde a um conjunto de *device drivers* (acionadores de dispositivos) que são os módulos de software que executam a comunicação com os controladores promovendo nos dispositivos periféricos a execução das operações desejadas. Cada *device driver*, exclusivamente projetado para um dado tipo de dispositivo, converte solicitações de alto nível em comandos mais elementares, diretamente reconhecíveis pelos controladores ou dispositivos associados àquele *device driver*.

Dessa maneira, os detalhes técnicos associados aos dispositivos são confinados aos *device drivers*, ficando transparentes aos demais módulos de software, seus usuários. Estes devem apenas enviar requisições de alto nível aos *device drivers* e deixar a cargo destes a resolução dos detalhes. Por exemplo, o *driver* para uma unidade de disco é capaz de converter (com base na informação extraída do descritor de arquivo) um pedido de gravação de um trecho de arquivo em disco para uma seqüência de passos referenciando trilhas e setores, e transferir toda essa informação para o controlador apropriado.

O controlador, por sua vez, se responsabiliza pelo posicionamento da cabeça de leitura e gravação e pelo acompanhamento do processo. Em contraste, o *driver* para uma impressora efetua a conversão de um pedido de impressão de um trecho de arquivo para um conjunto de operações mais básicas, envolvendo transferências de caracteres, tipos de letras (fontes) e controles de impressão. De fato, os passos intermediários necessários variam, inclusive, de impressora para impressora. É por isso que, quando se compra uma impressora nova ela geralmente vem acompanhada do *driver* correspondente.

Outro componente do *kernel* de um sistema operacional é o *gerenciador de memória*, encarregado de coordenar o uso da memória principal. Em computadores monoprogramados,

esta tarefa é trivial, pois nestes casos o programa a executar é alocado na memória principal, executado e, então, substituído pelo programa que executa a tarefa seguinte. Porém, em ambientes multiusuários, ou em ambientes multitarefa, nos quais a máquina se encarrega de várias tarefas ao mesmo tempo, os deveres do gerenciador de memória são mais complexos. Nestes casos, muitos programas e blocos de dados devem coexistir na memória principal, cada qual em uma área própria, determinada pelo gerenciador de memória. Na medida das necessidades das diferentes atividades, o gerenciador de memória vai providenciando as áreas necessárias, e mantendo um mapa das regiões de memória não utilizadas.

A tarefa do gerenciador de memória torna-se mais complexa quando a área total de memória principal solicitada excede o espaço realmente disponível na máquina. Neste caso, o gerenciador de memória pode criar uma ilusão de espaço adicional alternando os programas e os dados entre a memória principal e o disco. Este espaço ilusório de memória é chamado de *memória virtual*. Por exemplo, suponha que seja solicitada uma área, na memória principal, de 64 MB, e que somente 32 MB estejam realmente disponíveis. Para criar a ilusão de um espaço maior de memória, o gerenciador de memória divide a área solicitada em partes chamadas *páginas* e armazena em disco o conteúdo destas páginas. À medida que as diversas páginas forem sendo solicitadas, o gerenciador de memória pode armazená-las na memória física, em substituição a outras que já não sejam mais necessárias, de modo que as demais partes do software possam ser executadas, como se houvesse, de fato, 64 MB de memória principal.

No *kernel* de um sistema operacional estão situados, também, o *escalonador* (*scheduler*) e o *despachante* (*dispatcher*) que serão estudados mais adiante.

→ Conceitos Básicos:

Vimos como um sistema operacional se comunica com os usuários e como os seus componentes trabalham conjuntamente para coordenar a execução de atividades dentro da máquina. Entretanto, ainda não analisamos como se inicia a execução de um sistema operacional. Isto é feito através de um procedimento conhecido como *booting*, executado pela máquina todas as vezes que esta é ligada. Para compreender tal procedimento, deve-se, antes de tudo, compreender a razão de ser ele executado.

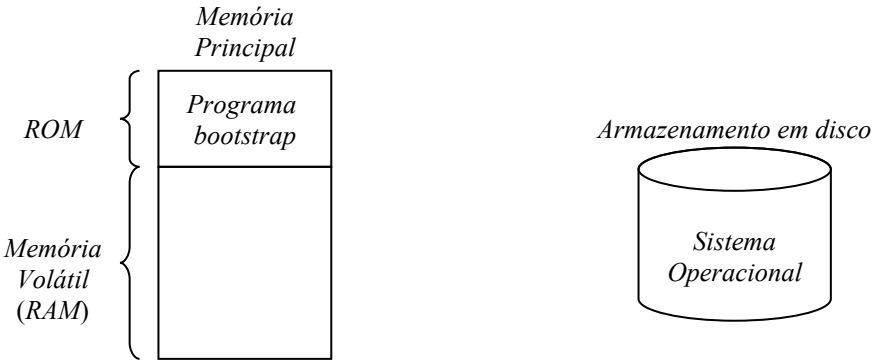
Um processador é projetado de forma tal que, todas as vezes que for ligado, o conteúdo do seu contador de instruções (PC) seja devidamente preenchido com um endereço predeterminado. É nesse endereço que o processador encontra o programa a ser executado. Para assegurar que tal programa esteja sempre presente, a área de memória é, normalmente, projetada de tal modo que seu conteúdo seja permanente. Tal memória é conhecida como memória *ROM* (*Read-only memory*). Uma vez que os padrões de bits sejam instalados na ROM, o que se faz mediante um processo de gravação, tal informação se conserva indefinidamente, mesmo com o desligamento da máquina.

No caso de computadores pequenos, como os utilizados como dispositivos em fornos de microondas, em sistemas de ignição de automóveis e em receptores estéreo de rádio, é possível implementar áreas significativamente grandes da memória principal em tecnologia ROM, uma vez que, nestes casos, o objetivo principal não é a flexibilidade, já que o programa a ser executado por tais dispositivos será sempre o mesmo, todas as vezes que for acionado. Contudo, isto não ocorre em computadores de propósito geral, não sendo, pois, prático transformar grandes áreas de memória principal em ROM, nessas máquinas. De fato, em máquinas de propósito geral, a maior parte da memória é volátil, o que significa que o seu conteúdo é perdido sempre que a máquina for desligada.

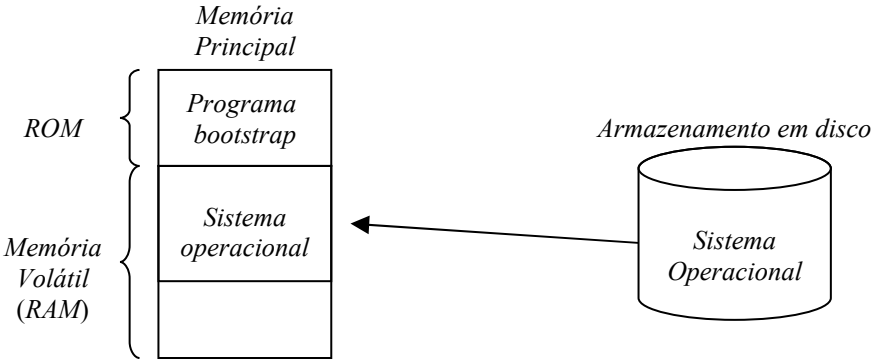
Para ser possível dar partida em uma máquina de propósito geral, a sua área de memória ROM é pré-programada com um pequeno programa chamado *bootstrap*, executado automaticamente toda vez que a máquina é ligada. Ele faz o processador transferir o conteúdo de uma área predeterminada do disco para uma região volátil da memória principal (Figura 3.3). Na maioria dos casos, este material é o sistema operacional. Uma vez transferido o sistema operacional para a memória principal, o *bootstrap* prepara o processador para executá-lo, e lhe transfere o controle da máquina. A partir de então, todas as atividades da máquina passam a ser controladas pelo sistema operacional.

3) Coordenação das Atividades da Máquina

Nesta seção, com base no conceito de processos, examinaremos como um sistema operacional coordena a execução dos softwares de aplicação, dos utilitários e dos diversos módulos internos ao próprio sistema operacional.



Passo 1: a máquina se inicia executando o programa bootstrap, residente na memória. O sistema operacional está armazenado em disco.



Passo 2: o programa bootstrap controla a transferência do sistema operacional para a memória principal e, em seguida, lhe transfere o controle.

Figura 3.3: O processo de *booting*

Conceito de Processo

Um dos conceitos mais fundamentais dos sistemas operacionais modernos é a distinção entre um programa e a atividade de executá-lo. O programa é apenas um conjunto estático de comandos, enquanto sua execução é uma atividade dinâmica, cujas propriedades mudam à medida que o tempo avança. Esta atividade é conhecida como *processo*. Um processo leva em conta a situação corrente da atividade, conhecida como *estado do processo*. Este estado inclui a posição do programa que está sendo correntemente executada (o valor do PC), bem como os valores contidos nos outros registradores do processador, e as posições associadas à memória. Grosseiramente falando, o estado do processo fornece uma fotografia da situação da máquina num dado momento.

Para enfatizar a distinção entre um programa e um processo, note que um único programa pode ser associado a mais de um processo em um mesmo instante. Por exemplo, em um sistema multiusuário, de tempo compartilhado, dois usuários podem, ao mesmo tempo, editar documentos separados. Ambas as atividades utilizam um mesmo programa editor de textos, mas cada qual caracteriza um processo separado, com seu próprio conjunto de dados. Nesta situação, o sistema operacional pode manter na memória principal uma só cópia do programa editor, e permitir que cada processo o utiliza à sua maneira, durante a fatia de tempo que lhe couber.

Em um computador de tempo compartilhado, é natural ter vários processos competindo pelas fatias de tempo. Estes processos englobam a execução de programas aplicativos e utilitários, bem como porções do sistema operacional. É tarefa do sistema operacional coordenar todos estes processos. Essa atividade de coordenação inclui garantir que cada processo tenha acesso aos recursos de que necessita (dispositivos periféricos, área na memória principal, acesso a dados e acesso ao processador), que processos independentes não interfiram uns com os outros, e que processos que se intercomunicam tenham a possibilidade de trocar informação entre si.

→ Administração de Processos

As tarefas associadas à coordenação de processos são manuseadas pelo escalonador (*scheduler*) e pelo despachante (*dispatcher*) no interior do kernel do sistema operacional. Assim, o escalonador mantém um registro dos processos presentes no sistema computacional, inclui novos processos nesse conjunto, e remove processos que já completaram sua execução. Para cuidar de todos os processos, o escalonador mantém, na memória principal, um conjunto de dados numa estrutura denominada *tabela de processos*. Cada vez que a máquina recebe uma nova tarefa, o escalonador cria para ela um processo, acrescentando uma nova linha à tabela de processos. Esta linha contém diversos indicadores: a área de memória designada para o processo (obtida por meio do gerenciador de memória), a prioridade do processo e um indicador de que o processo está pronto para ser executado ou à espera de algum evento. Diz-se que um processo está *pronto* para ser executado se estiver em um estado a partir do qual sua atividade possa prosseguir; o processo estará em *estado de espera* se seu progresso estiver sendo bloqueado até que seja registrada a ocorrência de algum evento externo, tal como a conclusão de um acesso ao disco ou recebimento de uma mensagem, enviada por algum outro processo. O escalonador mantém, então, atualizada esta informação à medida que o processo vai progredindo. O mais provável é que um dado processo irá alternar seu estado entre *pronto para execução* e *aguardando evento*, sua prioridade irá variar ao longo de sua execução e, certamente, o escalonador irá removê-lo da tabela de processos assim que suas atividades se completarem.

O despachante é o módulo do kernel do sistema operacional cuja função é a de assegurar que os processos escalonados sejam de fatos executados. Em um sistema de tempo compartilhado esta tarefa é realizada dividindo-se o tempo físico em pequenas *fatias*, tipicamente

inferior a 50 milissegundos, cada qual denominada *quantum* ou *time slice*. A "atenção" do processador é revezada entre os processos, a cada qual é concedido um intervalo de tempo não superior à duração de um quantum (Figura 3.4). O procedimento de alternar o processador de um processo para outro é denominado *chaveamento de processos*.

Cada vez que um processo inicia o uso de sua fatia de tempo, o despachante dispara um circuito temporizador, encarregado de medir o próximo quantum. Ao término desse quantum, esse relógio gera um sinal denominado *interrupção*. O processador reage a este sinal de uma forma muito parecida àquela pela qual alguém reage quando interrompido durante a execução de alguma tarefa: interrompe-se o que se estiver fazendo, registra-se o ponto da tarefa no qual se foi interrompido, e então passa-se a cuidar do atendimento ao evento que provocou a interrupção. Quando o processador recebe uma interrupção, completa o seu ciclo corrente de execução de instrução de máquina, quarda a posição em que se encontra e começa a executar um programa, chamado *rotina de tratamento de interrupção*, o qual deve ter sido depositado previamente em uma região predeterminada da memória principal.

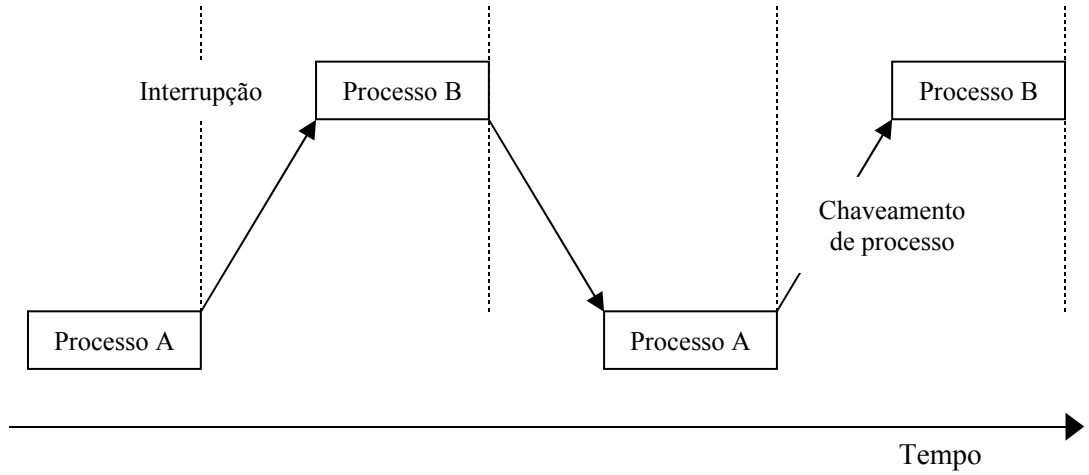


Figura 3.4: Compartilhamento de tempo entre os processos A e B

Em nosso ambiente de tempo compartilhado, o programa de tratamento de interrupção faz parte do próprio despachante. Assim, como efeito do sinal de interrupção, tem-se o bloqueio da continuidade do processo em andamento, devolvendo-se o controle ao despachante. Neste momento, o despachante permite que o escalonador atualize a tabela de processos (por exemplo, a prioridade do processo que acaba de esgotar a sua fatia de tempo poderá ser reduzida, e as prioridaes dos demais, aumentadas). O despachante, então, seleciona o processo de maior prioridade dentre os que se encontram prontos, e reinicia a operação do temporizador, dando início a uma nova fatia de tempo.

Um ponto alto de um sistema de tempo compartilhado é a capacidade de parar um processo para continuá-lo mais tarde. Caso ocorra uma interrupção durante a leitura de um livro, a capacidade do leitor de continuar a leitura mais tarde depende de sua habilidade de relembrar o ponto em que parou, bem como de reter a informação acumulada até tal ponto. Em suma, deverá ser capaz de recirar o ambiente existente imediatamente antes da ocorrência da interrupção. Tal ambiente é denominado *estado* do processo. Este estado inclui o valor do contador de instruções, o conteúdo dos registradores e os das posições de memória relevantes. Processadores projetados para operarem sistemas de tempo compartilhado incluem recursos para guardarem tal estado a cada ocorrência de interrupção. Possuem também instruções, em linguagem de máquina, para recarregar um estado anteriormente armazenado. Tais características simplificam a tarefa, de responsabilidade do despachante, de efetuar a alternância de processos e ilustram até que ponto o

projeto das máquinas modernas pode ser influenciado pelas necessidades dos sistemas operacionais.

Às vezes, a fatia de tempo de um processo termina antes do tempo determinado pelo temporizador. Por exemplo, se um processo executar uma solicitação de entrada ou saída, por exemplo, solicitando dados de um disco, sua fatia de tempo será "truncada" pelo sistema, uma vez que, de outra forma, tal processo desperdiçará o tempo restante da fatia, aguardando que o controlador terminasse de executar a operação solicitada. Neste caso, o escalonador atualizará a tabela de processos, marcando o processo corrente como estando em estado de espera, e o despachante fornecerá um novo quantum a outro processo que já esteja pronto para ser executado. Depois (talvez várias centenas de milissegundos mais tarde), quando o controlador indicar que aquela operação de entrada ou saída foi completada, o escalonador reclassificará o processo como estando pronto para a execução, habilitando-o assim a competir novamente por outra fatia de tempo.

→ **SISTEMA DE ARQUIVO.**

Todos nós sabemos que dados - sejam eles partes de programas ou dados propriamente dito, como um texto ou uma planilha - devem ser armazenados em um sistema de memória de massa, já que a memória (RAM) do micro é apagada quando desligamos o computador. Memória de massa é o nome genérico para qualquer dispositivo capaz de armazenar dados para uso posterior, onde incluímos disquetes, discos rígidos, CD-ROMs, ZIP drives e toda a parafernália congênere.

Dados são armazenados em forma de arquivos e a maneira com que os arquivos são armazenados e manipulados dentro de um disco (ou melhor dizendo, dentro de um sistema de memória de massa) varia de acordo com o sistema operacional.

Na maioria das vezes, um disco é dividido em pequenas porções chamados setores. Dentro de cada setor cabem 512 bytes de informação. Multiplicando-se o número total de setores de um disco por 512 bytes, teremos a sua capacidade de armazenamento. No caso de um disco rígido, ele possui na verdade vários discos dentro dele. Cada face de cada disco é dividida em círculos concêntricos chamados cilindros ou trilhas. Em cada trilha temos um determinado número de setores. É claro que toda esta divisão é invisível, pois é feita magneticamente. Para sabermos qual o número total de setores de um disco rígido, basta multiplicarmos sua geometria, ou seja, o seu número de cilindros, lados (parâmetro também chamado de "cabeças") e setores por trilha. Um disco rígido que possua a geometria de 2448 cilindros, 16 cabeças e 63 setores por trilha, terá 2448 x 16 x 63 = 2.467.584 setores. Multiplicando-se o número total de setores por 512 bytes, teremos sua capacidade total, no caso 1.263.403.008 bytes.

Importante notar que 1 KB não representa 1.000 bytes, mas sim 1.024, assim como 1 MB não representa 1.000.000 de bytes, mas sim 1.048.576. Muita gente arredonda e acaba errando nas contas. Lembre-se: 1 KB = 2^{10}, 1 MB = 2^{20} e 1 GB = 2^{30}. No exemplo dado, o disco rígido seria de 1,18 GB (basta dividir a capacidade que encontramos em bytes por 2^{30} para encontrarmos o resultado em gigabytes) e não 1,26 GB como seria de se supor.

O Sistema de Arquivos FAT-16

O sistema de arquivos utilizado pelo MS-DOS chama-se FAT-16. Neste sistema existe uma Tabela de Alocação de Arquivos (File Allocation Table, FAT) que na verdade é um mapa de utilização do disco. A FAT mapeia a utilização do espaço do disco, ou seja, graças a ela o sistema operacional é capaz de saber onde exatamente no disco um determinado arquivo está armazenado.

Existem várias posições na FAT, sendo que cada posição aponta a uma área do disco. Como cada posição na FAT-16 utiliza uma variável de 16 bits, podemos ter, no máximo, 216 = 65.536 posições na FAT. Como em cada setor cabem apenas 512 bytes, concluímos que, teoricamente, poderíamos ter discos somente de até 65.536 x 512 bytes = 33.554.432 bytes ou 32 MB.

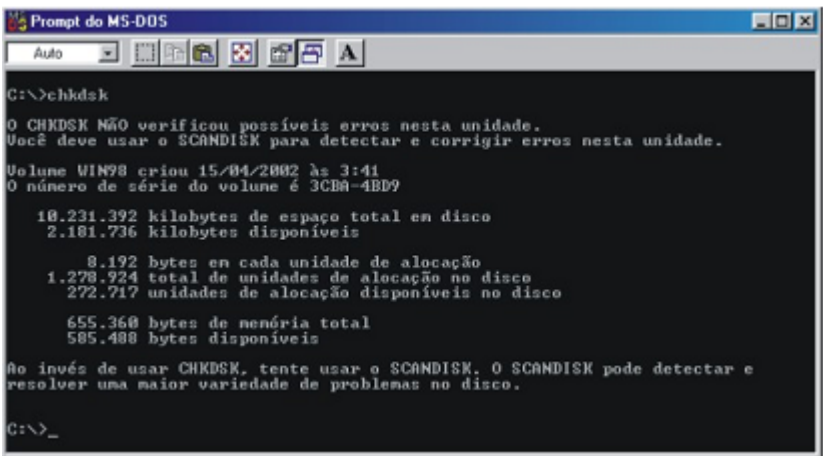
Por este motivo, o sistema FAT-16 não trabalha com setores, mas sim com unidades de alocação chamadas clusters, que são conjuntos de setores. Em vez de cada posição da FAT apontar a um setor, cada posição aponta para um cluster, que é um conjunto de setores que poderá representar 1, 2, 4 ou mais setores do disco.

Tamanho do Cluster	Capacidade Máxima de Armazenamento
2 KB	128 MB
4 KB	256 MB
8 KB	512 MB
16 KB	1 GB
32 KB	2 GB

O tamanho do cluster é definido automaticamente pelo sistema operacional quando o disco é formatado, seguindo a tabela. Um disco rígido de 630 MB utilizará clusters de 16 KB, enquanto um de 1,7 GB utilizará clusters de 32 KB. Como a menor unidade a ser acessada pelo sistema operacional será o cluster, os arquivos deverão ter, obrigatoriamente, tamanhos múltiplos do tamanho do cluster.

Isto significa que um arquivo de 100 KB em um disco rígido que utilize clusters de 8 KB obrigatoriamente ocupará 13 clusters, ou 104 KB, pois este é o valor mais próximo de 100 KB que conseguimos chegar utilizando clusters de 8 KB. Neste caso, 4 KB serão desperdiçados.

Quanto maior o tamanho do cluster, maior o desperdício. Se o mesmo arquivo de 100 KB for armazenado em um disco rígido que utilize clusters de 16 KB, ele obrigatoriamente utilizará 7 clusters, ou 112 KB. E, para o caso de um disco rígido com clusters de 32 KB, este mesmo arquivo ocupará 4 clusters, ou 128 KB. O desperdício em disco é um dos maiores problemas do sistema FAT, característica que chamamos de slack space. Quando maior o tamanho do cluster, mais espaço em disco é desperdiçado. Para saber qual o tamanho do cluster que está sendo utilizado em seu disco rígido, basta utilizar o comando CHKDSK, observando a linha "xxxxxx bytes em cada unidade de alocação", onde "xxxxx" é o tamanho do cluster em bytes.



Desse jeito vemos que o grande vilão do sistema FAT-16 é o desperdício em disco. Há, contudo, outro grande problema: o sistema FAT-16 não reconhece diretamente discos maiores que 2 GB. Para que discos com mais de 2 GB possam ser utilizados, devemos particioná-los, ou seja, dividi-los logicamente em outros

menores que 2 GB. No caso de um disco rígido de 2,5 GB devemos obrigatoriamente dividi-lo em dois, podendo esta divisão ser, por exemplo, uma unidade de 2 GB e outra de cerca de 500 MB.

FAT-32: A solução definitiva?

Junto com a última versão do Windows 95 (chamado Windows 95 OSR2), a Microsoft lançou um novo sistema de arquivos, denominado FAT-32. Este sistema estará presente também no sistema operacional Windows 98. Com o sistema FAT-32 o tamanho dos clusters é sensivelmente menor, o que faz com que haja bem menos desperdício. Este sistema permite, também, que discos rígidos de até 2 terabytes (1 TB = 2^40 bytes) sejam reconhecidos e acessados diretamente, sem a necessidade de particionamento.

Tamanho do Cluster	Capacidade Máxima de Armazenamento
512 bytes	256 MB
4 KB	8 GB
8 KB	16 GB
16 KB	32 GB
32 KB	2 TB

O sistema FAT-32 apresenta, porém, uma série de pequenos problemas:

1) Discos que utilizem o sistema FAT-32 não são "enxergados" por outros sistemas operacionais que não sejam o Windows 95 OSR2 ou superiores. Até mesmo o Windows 95 tradicional (e também o MS-DOS) não acessa discos que estejam formatados com o sistema FAT-32.

2) Utilitários de manutenção de disco rígido mais antigos também não acessam discos formatados em FAT-32, como, por exemplo, o Norton Utilities (As versões atuais já reconhecem discos em FAT-32).

3) Não é mais rápido. No geral é cerca de 6% mais lento que o sistema FAT-16. Quanto mais clusters o disco rígido tiver e quanto menor eles forem, mais lento será o sistema de armazenamento de dados.

HPFS e NTFS: As verdadeiras soluções!

A verdadeira solução para o problema de desperdício em disco é a utilização de um outro sistema de arquivos que não o FAT. O sistema operacional OS/2, por exemplo, possuía um excelente sistema de arquivos denominado HPFS (High Performance File System). O sistema operacional Windows NT (e também o Windows 2000 e Windows XP) também possui o seu próprio (e também excelente) sistema de arquivos, denominado NTFS (New Technology File System).

No caso do OS/2 e do Windows NT, na hora de sua instalação o usuário pode optar em utilizar o sistema FAT-16 ou então o HPFS/NTFS. A vantagem destes sistemas de arquivo é que não há desperdício em disco, pois não há clusters: a menor unidade de alocação é o próprio setor de 512 bytes.

A desvantagem óbvia destes sistemas de arquivos: só podem ser utilizados em conjunto com os seus sistemas operacionais. Ou seja, não há como instalar o HPFS no Windows 95, MS-DOS, etc... Outra desvantagem: assim como o sistema FAT-32, não são "enxergados" por outros sistemas operacionais diretamente (há, contudo, alguns "macetes" que permitem com que esta limitação seja transposta).

Capítulo 4

Introdução ao Linux

Linux é um Sistema Operacional baseado em UNIX assim como o AIX da IBM, Solaris, SunOS, BSD e outros.

Os direitos autorais pertencem a **Linus Torvalds** (**Linux = Linu**s + Unix) e outros colaboradores e pode ser livremente distribuído sob os termos da GNU (General Public License - GPL). Essa licença preserva os direitos autorais do software, mas assegura a distribuição dos programas com código-fonte.

Linux em si é apenas o Kernel (Núcleo) do sistema operacional, a parte que controla o hardware, gerencia arquivos, separa processos, etc.

Existem diversas combinações de Linux com vários utilitários e aplicativos de modo a formar um sistema operacional completo. Cada uma dessas combinações é chamada de **distribuição**.

DISTRIBUIÇÕES

Distribuições são empresas que pegam o Linux e adicionam facilidades ou recursos, juntam com aplicativos e utilitários desenvolvidos também por outras empresas e colocam seus CD's à disposição dos interessados, juntamente com suportes manuais etc.

Dentre as principais distribuições, podemos citar:

_ Caldera, Conectiva, Corel, Debian, Mandrake, Redhat, Slackware, Suse, Tech Linux, Turbo Linux, Ubuntu Linux, Mandriva, etc.

Características

Free Software – tem todo seu código fonte(incluindo o Kernel, drivers, bibliotecas, etc.) aberto para qualquer um.

Além de ser um software livre o Linux possui outras características como:

_ **Modularidade** – O UNIX é o único em seu desenho modular, que permite usuários adicionar ou remover partes para adaptá-lo as suas necessidades específicas.

_ **Multitasking** (Capacidade de Multitarefa) – A capacidade de Multitasking do UNIX permite que mais de uma tarefa seja realizada simultaneamente.

_ **Multiuser** (Multiusuário) – Um sistema multiusuário permite que vários usuários utilizem o computador simultaneamente.

_ **Portabilidade** – A portabilidade é a possibilidade dos softwares que operam em uma máquina operarem em uma outra diferente.

_ **Multiprocessamento** – O UNIX foi aprimorado para trabalhar em máquinas de grande porte, suportando assim a utilização de vários processadores em um único computador, e com a possibilidade de compartilhamento do processamento entre vários usuários ao mesmo tempo.

Estrutura Organizacional

A estrutura organizacional pode ser divida em 4 partes principais:
_ *Kernel, Shell*, processos e estrutura de arquivos.

Kernel

É a base do Sistema Operacional, suporta os programas dos usuários e programas utilitários como compiladores, linkers, sistemas de controle, código fonte, etc. e implementa as seguintes funções:

- Escalonamento de processos;
- Gerenciamento de memória;
- Gerenciamento de dispositivos;
- Gerenciamento de arquivos;
- Interface de chamada do sistema;
- Interface de controle do operador.

VERSÕES DO KERNEL

Convenção para distinguir as versões estáveis das não estáveis:

•Versão v.x.y, onde:
v – número da versão.
x – número par ➔ versão estável
número impar ➔ versão não estável (versões para beta teste – características novas sendo acrescentada todo o tempo)
y – correções de bugs – nenhuma característica nova implementada.

De tempos em tempos o Kernel apresenta-se como uma performance confiável passando a ser denominado como uma Kernel estável. O desenvolvimento continua em uma nova versão do Kernel.

Kernel é mudado conforme novos devices e/ou drivers são adicionados e erros são corrigidos.

Shell

Shell é o programa que conecta e interpreta os comandos digitados por um usuário. Ele é o mediador entre o usuário e o sistema LINUX usado para interpretar os comandos. Esse programa lê os comandos digitados pelo usuário e os executa utilizando os serviços e/ou outros programas do sistema operacional.

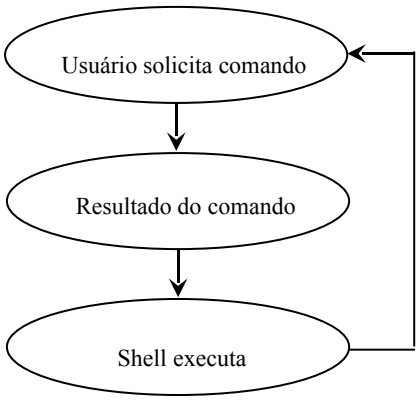


Figura 1: Interação Usuário - Shell

Dentre os Shells mais conhecidos pode-se citar:

- **sh** Bourne Shell – o mais tradicional(prompt → $);
- **Bash** Bourne – Again Shell – Shell padrão do Linux;
- **ksh** Korn Shell – muito utilizado atualmente;
- **csh** C Shell – considerado o mais poderoso – largamente utilizado(prompt → %);
- **rsh** Remote Shell – Shell remoto;
- **Rsh** Restricted Shell – versão restrita do sh

Processos

Um processo é um simples programa que está rodando em seu espaço de endereçamento virtual próprio. Os processos são executados em background ou foreground e podem ser divididos em três grupos principais:

_ Interativos, Batch e Deamons.

TIPOS DE PROCESSOS

- **Processos interativos** são iniciados a partir de, e controlados por uma sessão terminal.
- **Processos batch**, ou em lote, não são associados a nenhum terminal. Ao invés disso são submetidos a uma fila, da qual jobs são executados seqüencialmente.
- **Deamons** são processos servidores, inicialmente iniciados durante o boot, que rodam continuamente enquanto o sistema estiver ativo, esperando, em background, até que um processo requisite seus serviços.

ATRIBUTOS

- Process ID (PID)
- Parent Process ID (PPID)
- TTY
- UID real e efetiva (RUID, EUID)
- GID real e efetiva (RGID, EGID)

Process ID, PID é um número que identifica unicamente esse processo e é usado para referir-se a ele.

Parent Process ID, PID é o processo pai do processo, ou seja, o processo que o criou.

TTY é o dispositivo de terminal associado com o processo.

UID real de um processo é o UID do usuário que o criou. O UID efetivo, é o UID que é utilizado para determinar o acesso do processo a recursos do sistema. Usualmente, RUID e EUID são os mesmos e o processo tem os mesmos acessos que o usuário que os disparou.

GID real e efetivo de um processo é o grupo primário ou corrente do usuário.

Estrutura de Arquivos

Arquivos são centrais para o UNIX de uma maneira não encontrada em outros sistemas operacionais.

O UNIX tem uma organização de diretórios hierárquica em forma de árvore conhecida como filesystem. A base desta árvore é um diretório chamado root directory. Em sistemas UNIX, todo espaço em disco disponível é combinado em uma única árvore de diretórios abaixo do "/" (root).

O acesso a arquivos é organizado através de propriedades e proteções. Toda segurança do sistema depende, em grande parte, da combinação entre a propriedade e proteções setadas em seus arquivos e suas contas de usuários e grupos.

Resumindo:

- Tudo em UNIX é um arquivo;
- O UNIX tem uma organização de diretórios hierárquica chamada filesystem;
- O acesso a arquivos é organizado através de propriedades e proteções.

Tipos de arquivos

Existem três tipos de arquivos no UNIX:

Arquivos Ordinários (ou arquivos comuns):

São arquivos que contém dados binários ou caracteres ASCII. Consiste de uma string de bytes de dados. Por exemplo, um arquivo criado através de um editor de texto é um arquivo do tipo ordinário. Uma subclasse de arquivos ordinários são os arquivos **hidden** (oculto). O nome dos arquivos hidden começam sempre com um ponto (`.profile, .kshrc`) e possuem funções especiais. Esses arquivos recebem a denominação de *hidden* porque normalmente não podem ser vistos em uma consulta a um diretório.

Arquivos Diretórios

São responsáveis pela manutenção da estrutura hierárquica do sistema de arquivos. As informações são armazenadas em arquivos ordinários, estes são agrupados em diretórios, que por sua vez também são agrupados em outros diretórios.

Todo diretório contém os nomes de arquivos "." e ".." , que correspondem respectivamente ao próprio diretório e ao seu diretório pai. Todo usuário possui um diretório default (**home directory**), e quando ele se conecta ao sistema o UNIX automaticamente o posiciona neste diretório.

Arquivos Especiais

O UNIX trata todos os dispositivos físicos do sistema como arquivos especiais. Cada dispositivo, como terminais, unidades de fita, disco e impressoras, possui um arquivo especial associado à ele. Estes arquivos residem no diretório **/dev,** e não referenciam dados, eles possuem especificações sobre o tipo de dispositivo, como terminal ou impressora, e suas características, como configuração, densidade de gravação, etc. Os arquivos especiais podem ser acessados da mesma forma que os arquivos ordinários.

Obs: O nome do arquivo pode possuir até 255 caracteres, em sistemas BSD, ou até 14 caracteres, em sistemas System V. Alguns caracteres não devem ser usados por possuírem significado especial para o UNIX, são eles: "/", "*", "?", "[", "]", ">", "<", "-", "$", "'","""", "&", "!" e "\". Para quem está acostumado com o DOS, notará algumas diferenças como os nomes de arquivos, que no DOS tem apenas 8 caracteres de nome e 3 de extensão. No UNIX não tem limite podendo ter várias extensões. Os arquivos no UNIX não possuem extensões, porém podemos incluir um ponto seguido de uma extensão no nome de qualquer arquivo, embora esta extensão não possua nenhum significado especial para os comandos UNIX.

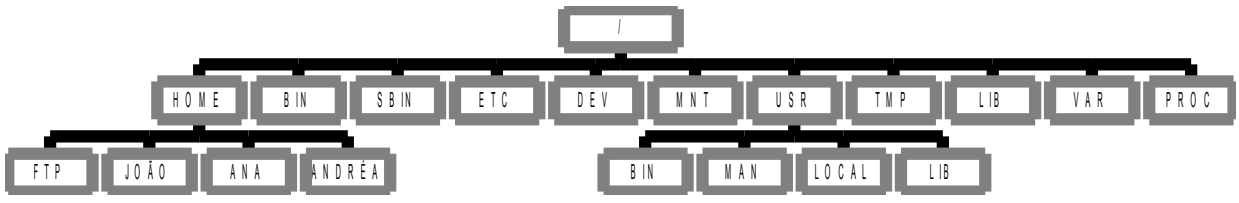
EX: relatório.txt.zip.tar

Hierarquia de arquivos

Como foi dito anteriormente os arquivos diretórios são organizados hierarquicamente em forma de árvore em que a base é chamada de root. Esta árvore pode ser facilmente observada na figura 2.

Estrutura hierárquica

Figura 2: Estrutura Hierárquica



Diretório	Descrição dos arquivos que estão nesse diretório.
/	Diretório raiz do sistema de arquivos. é abaixo dele que se situam todos os outros.
/bin	Arquivos executáveis de comandos essenciais.
/boot	Arquivos estáticos necessários à inicialização do sistema.
/dev	Arquivos de dispositivos (drives) do sistema.
/etc	Arquivos de configuração do sistema.
/home	Lugar onde ficam os diretórios locais(diretórios pessoais) dos usuários.
/lib	Arquivos de bibliotecas essenciais ao sistema, utilizados pelos programas em /bin.
/mnt	Usualmente é o ponto de montagem de dispositivos(drives) na máquina.
/proc	Informações do kernel e dos processos.
/root	Diretório local do superusuário.
/sbin	Arquivos essenciais (comandos especiais) ao sistema. Normalmente só o superusuário tem acesso a estes arquivos.
/tmp	Diretório de arquivos temporários.
/usr	Arquivos pertencentes aos usuários. (é a segunda maior hierarquia de diretórios presente no Linux, só perdendo para o diretório raiz).

Tabela 1: Estrutura de Diretórios do Linux

Conexão e Desconexão com o Sistema

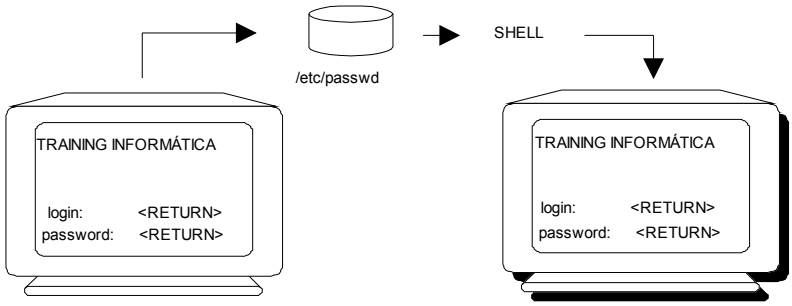


Figura 2: Conexão com o sistema

Conexão com o sistema

Para nos conectarmos ao sistema necessitamos de uma identificação de usuário (login) e uma senha (password) de acesso, como mostra a figura acima. Caso ocorra algum erro na digitação do login ou da password, o sistema responde com a mensagem 'login incorrect' e proporciona uma nova oportunidade ao usuário de se conectar.

Havendo sucesso o shell apresenta um prompt, indicando que está pronto para receber comandos do usuário. O prompt apresentado depende do shell que se está utilizando, no Bourne Shell e Korn Shell o prompt é o caracter "$" enquanto no C Shell é o caracter "%", sendo que para o usuário **root** o prompt apresentado será sempre o caracter "#".

As mensagens que são enviadas ao usuário durante a conexão com o sistema variam de instalação para instalação. Normalmente incluem a data e hora da ultima conexão do usuário ao sistema.

Desconexão do sistema

A desconexão do sistema é feita através do comando logout, ou ainda digitando-se as teclas [Ctrl+D]. Existe também a opção de executar o comando exit.

Atenção: O sistema operacional UNIX é case sensitive, ou seja, faz distinção entre letras maiúsculas e minúsculas, tratando-as como caracteres diferentes.

Criação e exclusão de usuários e grupos

Uma conta de usuário é um conjunto de nome de acesso mais uma senha que possibilitam que o usuário acesse a sua área do sistema. Sendo assim, uma mesma pessoa pode possuir várias contas, basta que sejam criadas com nomes de acesso diferentes.

Basicamente, existem dois tipos de contas: a conta do usuário comum, que utiliza o sistema e suas ferramentas, e a conta de superusuário ou conta de root, onde é possível realizar as configurações do sistema. A senha de root é muito importante, pois algumas configurações só são possíveis tendo a mesma em mãos.

Um grupo é basicamente um conjunto de usuários. Geralmente ele é criado quando se cria o usuário. Mas também podemos criá-lo quando queremos que os usuários tenham permissão restrita a arquivos em comum.

Existem várias formas de criar e manter usuários em um sistema. Uma destas formas é através da linha de comando. O comando utilizado para criar uma nova conta de usuário é o comando useradd (ou adduser). Portanto, para criar uma conta com o nome de acesso roberto como no exemplo, basta digitar o comando (como superusuário):

```
# useradd roberto
```

E para criar uma senha, basta utilizar o comando **passwd**. A senha pode conter qualquer caractere e é desejável que ela tenha no mínimo 6 caracteres.

```
# passwd roberto
Changing password for user roberto
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully
#
```

Com isto, a nova conta estará criada. Uma linha será incluída no arquivo /etc/passwd com os dados do novo usuário, incluindo o seu diretório de trabalho (/home/roberto), sendo colocado o /bin/bash como o interpretador de comandos padrão, entre outros dados.

Do mesmo modo, para remover contas de usuários basta digitar o comando:

```
# userdel -r usuario
```

Todos os dados do usuário serão apagados do arquivo /etc/passwd, e o diretório de trabalho do usuário será apagado.

Através do modo texto o comando **groupadd** cria um novo grupo. Para remover um grupo basta executar o comando **groupdel**.

Existem dois modos de incluir usuários em um grupo pelo modo texto: através do comando **usermod** ou diretamente no arquivo /etc/group. Vejamos primeiramente como incluir usuários em um grupo editando diretamente o arquivo:

```
superteste:x:504:teste,roberto
```

Note, portanto, que os usuários teste e roberto foram incluídos no grupo superteste. Para incluir mais usuários basta incluir uma vírgula no fim da linha e o nome do usuário a seguir. Vejamos agora o mesmo exemplo, utilizando o comando **usermod**:

```
# usermod -G superteste teste
# usermod -G superteste roberto
```

O comando **usermod** inclui apenas um usuário por comando. É possível colocar vários grupos, mas deve-se tomar cuidado pois isto apaga a configuração anterior.

Comandos

Assim como outros Sistemas Operacionais, o UNIX/Linux necessita de comandos para ser utilizado em modo texto. Neste curso você aprenderá alguns comandos básicos e algumas comparações para usuários do DOS.

Os comandos estudados serão os seguintes:

cat _____

chmod _____

chown _____

chgrp _____

cp _____

kill _____

man _____

mkdir _____

more _____

less _____

mv _____

passwd _____

ps _____

pwd _____

rmdir _____

rm _____

clear _____

who _____

whoami _____

cd _____

find _____

ls _____

date _____

login _____

logout _____

exit _____

vi _____

su _____

Formato Padrão de uma Linha de Comando

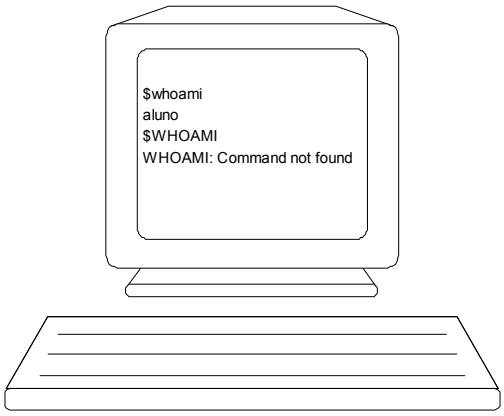


Figura 3: Utilização correta de comandos

Como já foi dito anteriormente, o UNIX faz distinção entre letras maiúsculas e minúsculas. A maioria dos comandos deve ser escrita com letras minúsculas, caso contrário o Shell não os interpretará corretamente. Esta distinção também é válida para nomes de arquivos, ou seja, os nomes carta.txt e CARTA.TXT são considerados arquivos diferentes. Os comandos UNIX não podem ser abreviados, e o formato padrão de uma linha de comando é o seguinte:

```
comando [- opção...] [argumento...]
```

A opção, ou opções, são letras que redefinem ou modificam a ação do comando, enquanto que o argumento, ou argumentos, são os objetos que sofrerão a ação do comando. Os nomes dos argumentos devem ser separados por espaços em branco, não podendo ser utilizada a vírgula. Podemos porém digitar mais de um comando em uma única linha, desde que sejam separados por ponto e vírgula (;).

Comparação DOS x LINUX

Tabela simplificada de correspondência DOS - LINUX	
Comando do DOS	**Correspondente no LINUX**
ATTRIB	chmod
CD	cd
CLS	clear
COMP	diff
COPY	cp
DATE	date
DEL	rm
DELTREE	rm –rf
DIR	ls
ECHO	echo
EDIT	vi
EDLIN	ed
HELP	man
MD	mkdir
MORE	more
MOVE	mv
PRINT	lp, lpr
RD	rmdir
REN	mv
SORT	sort
TIME	date
TYPE	cat
VER	uname –a
XCOPY	cp –r

Tabela 2: Correspondência DOS - LINUX

Obs: Esta tabela mostra uma correspondência entre alguns comandos do DOS e os comandos do UNIX e não uma equivalência. Os comandos do UNIX são, em geral, mais complexos e poderosos do que os correspondentes no DOS.

Redirecionamento, filtros e pipes

REDIRECIONAMENTO

Normalmente, quando um comando é executado, a saída é direcionada para a tela do terminal e a entrada direcionada do teclado. Em várias situações, se faz necessário redirecionar a entrada e/ou saída de um comando. O UNIX permite fazê-lo de maneira simples e elegante, utilizando símbolos de redirecionamento de entrada e saída:

Símbolo	Descrição
>	Redireciona a saída. Caso o arquivo já exista, seu conteúdo é sobreposto, caso contrário o arquivo é criado.
>>	Redireciona a saída. Caso o arquivo já exista, seu conteúdo não é sobreposto, mas sim inserido no final do arquivo.
<	Redireciona a entrada.
2>	Redireciona a saída de mensagens de erros.

Tabela 3: símbolos de Redirecionamento

Exemplos:

```
$ ls -l > diretor.lis
$ cat diretor.lis
$
```

Redireciona a saída do comando ls da tela para o arquivo diretor.lis.

```
$ ls -l >> diretor.lis
$ cat diretor.lis
$
```

Redireciona a saída do comando ls da tela para o final do arquivo diretor.lis.

```
$ cat < arq.txt
$
```

Redireciona a entrada do comando cat do teclado para o arquivo arq.txt.

```
$ date > arq.lis 2> arq.err
$
```

Redireciona a saída do comando date para o arquivo arq.lis e redireciona qualquer mensagem de erro para arquivo arq.err.

Filtros

Filtros são comandos ou utilitários que manipulam dados de entrada e geram uma nova saída. A seguir apresentamos os principais filtros:

Sort:

O comando sort permite ordenar um ou mais arquivos de entrada e gerar um novo arquivo de saída.

Exemplo:

```
$ cat alunos.txt
Francis
```

```
LP
Bessa
$ sort -o alunos.ord alunos.txt
$ cat alunos.ord
Bessa
Francis
LP
$
```

Grep:

O comando grep (global regular expression printer) procura todas as ocorrências de um determinado padrão (string) em um ou mais arquivos de entrada. A saída do comando é uma lista das linhas a onde o padrão foi encontrado.

Exemplo:

```
$ cat arq.txt
amo
amor
amora
amoroso
clamor
namoro
ramo
$ grep 'moro' arq.txt
amoroso
namoro
$
```

Pipes

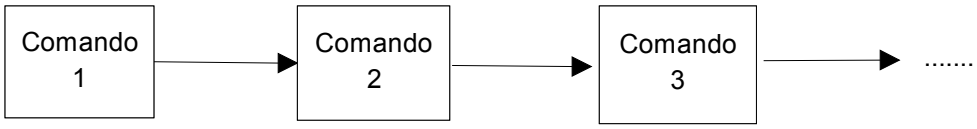
Suponha que você queira utilizar a saída de um comando como entrada de um segundo comando. Uma maneira é criar um arquivo temporário para armazenar a saída do primeiro comando, utilizar o mesmo arquivo como entrada no comando seguinte e, posteriormente, eliminar o arquivo temporário.

Exemplo:

```
$ who > temp.txt
$ wc -l < temp.txt
10
$ rm temp.txt
$
```

O comando who, que lista todos os usuários conectados ao sistema, cria o arquivo temporário temp.txt. O comando wc -l, que conta o número de linhas do arquivo, utiliza o mesmo arquivo temporário para informar quantos usuários estão conectados ao sistema.

O Unix permite que esse tipo de processamento seja implementado de forma bastante simples, utilizando o conceito de pipeline. Através da barra vertical (|), ou pipe, a saída de um comando pode ser direcionada para entrada de um outro comando sem a utilização de arquivos temporários. A idéia do pipeline pode ser comparada a uma linha de montagem, onde uma linha de produção fornece insumos para outra linha, sucessivamente.



Exemplos:

```
$ who | wc -l
10
```

```
$ ls -l | grep '\.txt$' | sort | lpr
```

O comando ls gera uma listagem completa dos arquivos e subdiretórios do diretório de trabalho. O comando grep seleciona os arquivos que terminam com '.txt', a partir da saída comando ls. O comando sort ordena os arquivos selecionados pelo grep. Finalmente a lista dos arquivos é impressa através do comando lpr.

```
$ ls -l | sort +4n | grep '\.c$'
```

O comando ls gera uma listagem completa dos arquivos e subdiretórios de trabalho. O comando sort ordena a saída em função do tamanho de cada arquivo (+4n). O comando pega a saída ordenada e seleciona apenas aqueles que terminem com '.c'.

```
$ ls -l | sort +8 | grep '^d'
```

O comando exibe uma lista de subdiretórios ordenados crescentemente.

```
$ who | sort | tee usuarios.txt | wc -l
```

O comando tee permite recuperar resultados intermediários de pipes, permitindo, inclusive, armazenar a saída de um pipe em um arquivo.

Montando e Desmontando Dispositivos

Muitos usuários, ao visitarem sites sobre o sistema operacional ou ao ouvirem o relato de colegas que já usam o Linux, sentem-se incentivados a experimentarem o sistema. A grande maioria já tem o sistema operacional Windows instalado e decide compartilhar o computador também com o Linux. Mas quase sempre, o usuário necessita acessar arquivos presentes na partição do Windows. Isso é perfeitamente possível, através de um processo conhecido como montagem da partição. Algumas distribuições Linux montam as partições Windows automaticamente, mas em outras, é necessário que o usuário faça isso manualmente. Esse processo é simples e será

explicado aqui. Os procedimentos a seguir devem ser executados como usuário root ou outro que tenha permissões de administrador.

Discos

O primeiro passo é saber como identificar os discos (HD, CD-ROM, disquete) no Linux. Tais dispositivos, além de outros (como a porta LPT1) são tidas como existentes no diretório /dev/. O HD é identificado como o dispositivo hda0 (hdb para outro HD no mesmo computador e assim por diante). O número 0, indica a partição no HD. Com isso, a segunda partição deve ser identificada como hda1, a terceira como hda2, enfim. No caso de HDs SCSI, as letras hd devem ser trocadas por sd, ficando da seguinte forma: sda0, sda1, sda2, etc.

No caso do drive de disquete, a sigla para sua identificação é fd0. Se houver outro drive, este deve ser identificado como fd1. No caso dos CD-ROMS, eles são identificados como HDs. Assim, se por exemplo, em seu computador há um HD e um CD-ROM, o HD poderá ser reconhecido como hda e o CD como hdb.

Montando a partição

Agora que já se sabe o que são e como funcionam os dispositivos no Linux, serão estudados os comandos **mount** e **umount**, que são os responsáveis pela montagem dos dispositivos. Mas, antes de montar qualquer dispositivo, é preciso saber quais dispositivos estão ligados a ele. Primeiramente, os arquivos que mapeiam dispositivos estão no diretório */dev* da estrutura de diretórios de seu Linux. Serão mostrados a seguir os dispositivos que são mais freqüentemente utilizados para montagem. Veja na Tabela 4. Periféricos os dispositivos mais usados no cotidiano, com seus respectivos periféricos.

Arquivo	**Mapeia qual periférico**
hda	Primeiro disco rígido instalado na máquina (*master*).
hdaX	A partição X do primeiro disco rígido instalado. *hda1* mapeia a primeira partição do disco.
hdb	O segundo disco rígido/CD-ROM instalado na máquina (slave).
hdbX	A partição X do primeiro disco rígido instalado. *hdb1* mapeia a primeira partição do disco.
fd0	O primeiro drive de disquete.
fd1	O segundo drive de disquete.
cdrom	O drive de CD-ROM instalado na máquina (caso exista).

Tabela 4: Periféricos

Vale aqui uma lembrança: em *hd?X*, o item **X** pode variar de acordo com o número de partições existentes no disco rígido e **?** pode variar de acordo com o número de discos rígidos instalados na máquina.

Para a partição Windows, você deve saber qual das existentes é ela. Em nosso exemplo, vamos supôr que o Windows está na partição hda0 e o Linux na partição hda1. Tendo ciência disso, agora é necessário que você crie um diretório no Linux por onde a partição Windows deverá ser acessada, ou seja, o *ponto de montagem*. Geralmente, este diretório é criado dentro da pasta /mnt/ mas pode ser criado em outro. Para o nosso exemplo, vamos chamar este diretório de win. Assim, seu caminho é /mnt/win.

O próximo passo é comando de montagem:

mount - t [tipo] [caminho da partição] [ponto de montagem]

Em nosso exemplo, o comando acima ficaria assim:

mount -t vfat /dev/hda0 /mnt/win

O *tipo* indica o sistema de arquivos utilizados na partição. Partições fat e fa32 são identificados como vfat. No caso de CD-ROM, o tipo deve ser especificado como iso9660. Por exemplo: mount -t iso9660 /dev/hdb /mnt/cdrom. As partições NTFS podem, teriocamente, serem montadas do mesmo jeito. No entanto, existem problemas de compatibilidade entre o kernel do Linux e o sistema de arquivos NTFS, motivo pelo qual, a montagem desse tipo de partição deve seguir procedimentos especiais, que não serão explicadas aqui. Se a partição que você deseja montar for ext2 ou ext3, basta especificar estes nomes como tipo.

No exemplo a seguir, será demonstrado o procedimento para a montagem de um disquete formatado para Windows:

```
# mount -t vfat /dev/fd0 /mnt/floppy
```

Este comando instrui o kernel para que ele inclua o sistema de arquivos encontrado em */dev/fd0*, que é do tipo *vfat*, disponibilizando-o no diretório */mnt/floppy*. Caso haja algum arquivo no diretório onde foi montado o disquete (ou outro sistema de arquivos), esse conteúdo ficará indisponível enquanto o sistema de arquivos estiver montado. O diretório */mnt/floppy* referenciará o diretório raiz ("/") do sistema de arquivos montado.

Existem vários tipos de dispositivos que podem ser montados. Na Tabela 5. Tipos de sistema de arquivos constam os tipos mais utilizados.

Tipo	Descrição
vfat	Disquete formatado para Windows.
ext2	Disquete formatado para Linux.
ext3	Disquete formatado para Linux.
iso9660	CD-ROM.

Tabela 5: Tipos de sistema de arquivos

Dica
Para montar dispositivos automaticamente, ou melhor, na hora do boot na máquina, você deve editar o arquivo /etc/fstab, onde deverá acrescentar uma nova linha, indicando a montagem do dispositivo. Esta opção é interessante para HDs com mais de uma partição.

Nota
É importante não remover um dispositivo (físico), como um disquete de seu drive, enquanto ele permanecer montado. Existem operações que são efetuadas quando um dispositivo é desmontado. Uma delas é a gravação de dados que possam estar no buffer de armazenamento, esperando ser gravados.

Depois que um sistema de arquivos é montado, pode-se utilizá-lo normalmente. Ao final da utilização do sistema de arquivos, deve-se desmontá-lo para que se possa, por exemplo, remover o CD do drive. Para isso é utilizado o comando **umount**, que desmonta o sistema de arquivos. A sintaxe do comando **umount** é:

```
umount
dispositivo
```

ou então:

```
umount
diretório
```

Normalmente, apenas o superusuário poderá montar e desmontar um sistema de arquivos. Existe uma maneira simples para que o usuário comum acesse o disquete. São os comandos do pacote *mtools*. O mtools é uma coleção de ferramentas que permite ao Linux manipular arquivos MS-DOS. Sempre que possível o comando tenta simular o comando equivalente a esse sistema. Por exemplo, comandos como **mdir a:** funcionam na unidade de disquetes *a:* sem qualquer montagem ou inicialização prévia[46]. Alguns comandos do pacote *mtools* são: **mattrib, mbadblocks, mcd, mcopy, mdel, mdeltree, mdir, mdu, mformat, mkmanifest, mlabel, mmd, mmount, mmove, mrd, mread, mren, mtoolstest** e **mtype**. Para utilizá-los, basta digitar o comando desejado em um terminal como no exemplo abaixo:

```
$
mdir
```

Pacotes RPM

O sistema operacional Linux, até hoje, tem a fama de ter instalações complicadas, o que, muitas vezes, não deixa de ser verdade. Na tentativa de resolver esses problemas de instalação, A empresa Red Hat criou uma tecnologia chamada RPM, que significa **R**edHat **P**ackage **M**anager (Gerenciador de Pacotes RedHat).
O RPM é um poderoso gerenciador de pacotes, que pode ser usado para criar, instalar, desinstalar, pesquisar, verificar e atualizar pacotes individuais de software. Um pacote consiste em armazenagem de arquivos e informações, incluindo nome, versão e descrição. Veja a seguir, alguns comandos e suas respectivas funções, usadas nos pacotes RPM:

rpm -ivh - Instalação de pacotes;
rpm -Uvh - Atualização de pacotes;
rpm -qi - Informações sobre o pacote;

rpm -ql - Lista os arquivos do pacote;
rpm -e - Desinstala o pacote;
rpm -qa - Lista os pacotes instalados;

Utilizando o RPM

O RPM tem 5 modos básicos de operação, excluindo-se o modo de confecção de pacotes: instalação, desinstalação, atualização, consulta e verificação. Você pode obter mais informações usando *rpm --help* ou *man rpm*. Vejamos cada modo:

Instalação

Pacotes RPM têm nomes de arquivos tais como foo-1.0-1/.i386.rpm, que incluem o nome do pacote (foo), versão (1.0), release (1) e plataforma (i386). A instalação de um pacote é feita através de uma linha de comando, como por exemplo:
$ rpm -ivh foo-1.0-1.i386.rpm
foo ##################################
Como se pode observar, o RPM apresenta o nome do pacote e apresenta uma sucessão de caracteres # atuando como uma régua de progresso do processo de instalação. O processo de instalação foi desenvolvido para ser o mais simples possível, porém eventualmente alguns erros podem acorrer:
- Pacotes já instalados
Se o pacote já tiver sido instalado anteriormente será apresentada a seguinte mensagem:
$ rpm -ivh foo-1.0-1.i386.rpm foo package foo-1.0-1 is already installed error: foo-1.0-1.i386.rpm cannot be installed
Caso se deseje ignorar o erro, pode-se usar o parâmetro *--replacefiles* na linha de comando;
- Dependências não resolvidas
Pacotes RPM podem depender da instalação de outros pacotes, o que significa que eles necessitam destes pacotes para poderem ser executados adequadamente. Caso deseje instalar um pacote que dependa de outro pacote não localizado, será apresentada a seguinte mensagem:
$ rpm -ivh bar-1.0-1.i386.rpm
failed dependencies:
foo is needed by bar-1.0-1
Para corrigir este erro, será necessário instalar o pacote solicitado. Caso deseje que a instalação ocorra de qualquer forma, pode-se utilizar o parâmetro *--nodeps* na linha de comando.

Desinstalação

Para desinstalar um pacote utilize o comando:
$ rpm -e foo
Onde foo é o nome do pacote. Pode ser encontrado um erro de dependência durante o processo de desinstalação de um pacote (outro pacote necessita de sua existência para funcionar corretamente). Neste caso será apresentada a seguinte mensagem:
$ rpm -e foo removing these package
would break dependencies:
foo is needed by bar-1.0-1

Para ignorar a mensagem de erro e desinstalar o pacote deve ser utilizado o parâmetro *--nodeps* na linha de comando.

Atualização

Para atualizar um pacote utilize o comando:
$ rpm -Uvh foo-2.0-1.i386.rpm
foo ##
O RPM desinstalará qualquer versão anterior do pacote e fará a nova instalação preservando as configurações. Sugerimos utilizar sempre a opção *-U*, já que ela funciona perfeitamente, mesmo quando não há uma versão anterior do pacote. Uma vez que o RPM executa um processo de atualização inteligente, é apresentada uma mensagem do tipo:
saving /etc/foo.conf as /etc/foo.conf.rpmsave
O que significa que os arquivos de configuração existentes estão salvos, porém mudanças no software podem tornar este arquivo de configuração incompatível com o pacote. Neste caso, as adequações necessárias devem ser feitas pelo usuário. Como o processo de atualização é uma combinação dos processos de desinstalação e instalação, alguns erros podem ocorrer. Por exemplo, quando se quer atualizar um pacote com uma versão anterior à versão corrente, será apresentada a seguinte mensagem:
rpm -Uvh foo-1.0-1.i386.rpm
foo package foo-2.0-1 (which is never) is already installed error: foo-1.0-1.i386.rpm
cannot be installed
Para forçar uma atualização de qualquer forma, deve-se usar o parâmetro *--oldpackage* na linha de comando.

Consulta

A consulta à base de dados de pacotes instalados é feita através do comando *rpm -q*. Na sua utilização, são apresentados o nome do pacote, versão e release. Por exemplo:
$ rpm -q foo foo-2.0-1
Ao invés de especificar o nome do pacote, pode-se utilizar as seguintes opções após o parâmetro *q* :

- a - Consulta todos os pacotes instalados;
- f <file> - Consulta o pacote que contém o arquivo <file>;
- F - Funciona como o parâmetro -f, exceto que funciona a partir do stdin (entrada padrão), como por exemplo find /usr/bin | rpm -qF;
- p <arquivo do pacote> - Consulta o pacote originado pelo <arquivo do pacote>;
- P - Funciona como o parâmetro -p, exceto que funciona a partir do stdin (entrada padrão), como por exemplo find /mnt/cdrom/RedHat/RPMS | rpm -qP.

Há diversas formas de especificar que informações devem ser apresentadas pelas consultas. As opções de seleção são:

- i - Apresenta as informações do pacote, tais como nome, descrição, release, tamanho, data de criação, data de instalação e outras;
- l - Apresenta a lista de arquivos relacionados com o pacote;
- s - Apresenta o status dos arquivos do pacote. Há dois estados possíveis: normal ou missing (não localizado);
- d - Apresenta uma lista dos arquivos de documentação (páginas de manual, páginas info, README, etc.);
- c - Apresenta uma lista dos arquivos de configuração. Estes arquivos podem ser alterados após a instalação para personalização. Exemplos sendmail.cf, passwd, inittab, etc.

Verificação

A verificação de um pacote provoca a comparação dos arquivos instalados de um pacote com as informações localizadas nas bases de dados do RPM. Entre outras coisas a verificação compara o tamanho, MD5 sum, permissões, tipo, dono e grupo de cada arquivo.
Para verificar um pacote deve-se utilizar o comando:
rpm -V <nome do pacote
Pode-se usar as mesmas opções disponíveis no processo de consultas.
Exemplos:
Para verificar um pacote que contenha um arquivo em especial:
rpm -Vf /bin/vi
Para verificar todos os pacotes instalados:
rpm -Va
Para verificar um pacote instalado e o arquivo de pacote RPM:
rpm -Vp foo-1.0-1.i386.rpm

Permissões

Cada arquivo no UNIX apresenta três níveis de proteção definidos por três categorias de usuários. Todo arquivo ou diretório tem um dono (user) e pertence a um grupo (group). Qualquer usuário do sistema que não seja o dono do arquivo e não pertença ao grupo do arquivo enquadra-se na categoria outros (others). O administrador do sistema, o chamado superusuário (root), não pertence a nenhuma das categorias acima, tendo acesso irrestrito a todos os arquivos do sistema.

Para cada categoria de usuário, três tipos de acesso podem ser concedidos: leitura (read), escrita (write) e execução (execute). O sistema UNIX não faz distinção entre os acessos de escrita e eliminação, ou seja, um arquivo que pode ser alterado também pode ser eliminado.

As tabelas abaixo mostram as permissões para arquivos e diretórios:

Arquivos	Descrição
r	Permissão para ler e copiar o arquivo.
w	Permissão para alterar ou eliminar o arquivo.
x	Permissão para executar o arquivo.

Tabela 6: Permissões de arquivos

Diretórios	Descrição
r	Permissão para ler o conteúdo do diretório (listar o nome dos arquivos).
w	Permissão para criar, remover e renomear arquivos no diretório.
x	Permissão de busca: O usuário que não possuir esta permissão não poderá se posicionar no diretório nem acessar os arquivos na árvore abaixo deste diretório.

Tabela 7: Permissões de diretórios

Através da opção -l do comando ls podemos visualizar o código de proteção dos arquivos. Este código está representado pelos 9 caracteres que seguem o tipo de cada arquivo (primeiro caractere). Os três primeiros caracteres correspondem a proteção para o dono (user) do arquivo, os três próximos definem a proteção para a categoria grupo (group) e os últimos três caracteres especificam a proteção do arquivo em relação ao demais usuários (other).

Exemplo:

```
$ ls -al
total 9
drwxr-x--x   7     bessa  instrutores  512 Dec 12 11:32   .
drwxr-xr-x   9     root   instrutores  512 Oct 20 09:11   ..
-rwxr-x--x   1     bessa  instrutores  380 May 12 16:03   .cshrc
-rwxr-x--x   1     bessa  instrutores  160 May 12 18:25   .login
drwxr-xr-x   2     bessa  instrutores  512     Jan     05     10:55
      avaliacao
drwxr-xr-x   2     bessa  instrutores  512 Jan 09 15:53   curso
-rwxrw-r--   1     bessa  instrutores  970 Jun 16 11:03 dados.dat
-rwxrw-r--   1     bessa  instrutores  979     Sep     29     13:07
      manual.doc
drwxr-xr-x   2     bessa  instrutores  512 Feb 08 08:55   prog
$
```

No exemplo acima analisemos o arquivo dados.dat. Ele é do tipo ordinário. O usuário bessa (dono do arquivo) pode ler, alterar e executar o arquivo. Todos os usuários que pertençam ao grupo instrutores podem ler e alterar o arquivo mas não podem executa-lo. Qualquer outro usuário só poderá ler o arquivo.

A proteção assinalada ao arquivo diretório determina o primeiro nível de proteção para todos os arquivos deste diretório e tem prioridade sobre as proteções associadas individualmente a cada arquivo. Por exemplo, caso um usuário não tenha permissão de escrita em um arquivo diretório, não poderá eliminar nenhum arquivo deste diretório, ainda que as proteções dos arquivos indiquem o contrário.

Alteração das Permissões

Para alterarmos as permissões em um arquivo utilizamos o comando chmod (change file mode). Ele possui dois formatos:

Formato 1

```
$ chmod [ugoa] [+=-] [rwx ugo] arquivos
```

A letra u representa os privilégios do dono do arquivo; a letra g os privilégios do grupo e a letra o os privilégios dos demais usuários. A letra a (all) representa todas as categorias de usuários. O sinal + significa a adição de um privilégio e o sinal - a retirada de um privilégio. A opções r, w e x representam respectivamente as permissões de leitura, alteração e execução. Podemos alterar vários privilégios em um único comando, separando-os por vírgulas. Classes de usuários e níveis de proteção que não forem especificados no comando não serão alterados.

Exemplo:

```
$ ls -l
-rw-rw-rw-   1     jose   alunos 138    Jan 14 16:15 arquivo
$ chmod g-w,o-rw arquivo
$ ls -l
```

```
-rw-r-----   1     jose   alunos 138    Jan 14 16:30 arquivo
$
```

Formato 2

```
$ chmod nnn arquivo
```

Onde **nnn** é o código em octal que representa os tipos de acesso concedidos à cada categoria de usuário. Cada permissão tem um número associado: execute = 1, write=2 e read = 4. A tabela abaixo ilustra as combinações possíveis:

Número em Octal	Permissões	Descrição
0	---	Nenhum acesso permitido.
1	--x	Acesso execute
2	-w-	Acesso write
3	-wx	Acesso write e execute
4	r--	Acesso read
5	r-x	Acesso read e execute
6	rw-	Acesso read e write
7	rwx	Acesso read, write e execute

Tabela 8: Representação de permissões através de número octal

Exemplo:

```
$ ls -l
-rw-rw-rw-   1     jose   alunos 138    Jan 14 16:15 arquivo
$ chmod 640 arquivo

$ ls -l
-rw-r-----   1     jose   alunos 138    Jan 14 16:30 arquivo
$
```

Alteração do Grupo e Dono

Se você é membro de mais de um grupo, você pode eventualmente precisar alterar o grupo de um arquivo. Isto é feito através do comando chgrp. Sua sintaxe é:

```
$ chgrp grupo arquivo
```

Exemplos:

```
$ ls -l arquivo
-rw-r-----   1     jose   alunos       138    Jan 14 16:15 arquivo
$ chgrp monitores arquivo
$ ls -l arquivo
-rw-r-----   1     jose   monitores    138    Jan 14 16:25 arquivo
$
```

Caso desejemos saber todos os grupos existentes podemos listar o arquivo /etc/group com

o comando:

```
$ cat /etc/group.
```

É importante ressaltar que para utilizar o comando chgrp, o usuário deve pertencer ao novo grupo especificado e ser o dono do arquivo, ou então o superusuário.

Enquanto o comando chgrp permite a alteração do grupo do arquivo, o comando chown permite alterar o seu owner (dono).

```
# chown dono arquivo
```

Exemplos:

```
# ls -l arquivo
-rw-r-----   1     jose   alunos 138    Jan 14 16:15 arquivo
# chown luiz arquivo
# ls -l arquivo
-rw-r-----   1     luiz   alunos 138    Jan 14 16:25 arquivo
#
```

Este comando só pode ser utilizado pelo superusuário, pois só ele tem acesso a todas as contas e usernames do sistema.

Sites para consulta

Como o Linux é um software gratuito, existem vários sites com manuais e explicações gratuitas sobre o mesmo, ou seja, grande parte dos assuntos discutidos sobre o Linux pode ser encontrada nestes sites.

Abaixo podemos conferir alguns desses sites:

http://focalinux.cipsga.org.br/ → site com vários manuais para downloads.

http://www.linuxiso.org/ → site oficial do Linux onde todas as distribuições podem ser baixadas gratuitamente sem dificuldades.

http://www.softwarelivre.gov.br/links → site com vários links importantes sobre o Linux.

http://www.conectiva.com.br/cpub/pt/incConectiva/suporte/pr/hardware.html → site oficial de suporte da Conectiva com explicações de várias dúvidas comuns sobre diversas configurações de hardware e software no Linux.

http://www.rau-tu.unicamp.br/linux/ → site de perguntas e respostas sobre Linux. Caso não encontre a resposta no site, você pode formular e enviar sua pergunta e aguardar uma resposta de um colaborador.