Tópicos de Computação e Informática

Educação a Distância
Cruzeiro do Sul Educacional
Campus Virtual

Material teórico

Software

Responsável pelo Conteúdo:
Prof Dr Luciano Silva

Revisão Técnica:
Prof. Ms. Fábio Peppe Beraldo

Revisão Textual:
Profª. Esp. Márcia Ota

UNIDADE Software

- Introdução
- Software como um Ambiente Operacional
- Sistema Operacional como um Software Básico
- Compiladores e Interpretadores
- Linguagens de Programação

Objetivo de Aprendizado

Sejam bem-vindos a nossa Unidade de Conteúdo IV! O nosso objetivo será estudar com maiores detalhes a componente de software de um ambiente operacional, com ênfase nos sistemas operacionais, compiladores, interpretadores e linguagens de programação.

Para que possamos atingir o nosso objetivo nesta unidade, sugiro o seguinte plano de estudos:

Ler cuidadosamente a contextualização desta nossa unidade. Assistir a apresentação narrada, onde serão expostos os principais conceitos de redes de computadores. Ler o material teórico relativo a esta nossa quarta unidade.

Realizar as atividades de sistematização sobre os conceitos-chave tratados na nossa unidade. Participar ativamente da nossa atividade reflexiva, que será realizada através da produção de um documento escrito sobre linguagens de programação.

Consulte as nossas indicações de referências complementares sobre o assunto desta nossa unidade. Desejo muito sucesso a todos ao final desta unidade!

Contextualização

Nosso objetivo, nesta segunda unidade de conteúdo, será estudar, de um ponto de vista mais aprofundado, a componente de software de um ambiente operacional.

Software configura-se como um elemento essencial na organização de qualquer ambiente operacional. Um sistema operacional, por exemplo, é um exemplo típico de um software, chamado software básico. Outro exemplo importante são os compiladores e interpretadores, que permitem a implementação de um software através de linguagens de programação.

Neste contexto, a correta compreensão dos diversos tipos de software é essencial para se entender vários conceitos subjacentes em Computação como, por exemplo, Programação, Sistemas Distribuídos, dentre outros.

1. Introdução

Nesta nossa unidade, faremos um estudo mais aprofundado dos tipos de software que compõem os níveis mais altos de abstração de um ambiente operacional. Estudaremos estes tipos acompanhando a organização hierárquica de software: software básico → software aplicativo.

Para que este objetivo seja alcançado, o material está organizado da seguinte forma:

- **A Seção 2** revisa os conceitos de software estudados na nossa primeira unidade.
- **A Seção 3** revisa um tipo de software básico bastante importante: o sistema operacional.
- **A Seção 4** apresenta outros exemplos importantes de software básico: os compiladores e interpretadores.
- Finalmente, a **Seção 5** faz um pequeno estudo das principais linguagens de programação, caracterizando-as como compiladas ou interpretadas.

Ao final do estudo e das atividades desta unidade, deveremos ser capazes de:

- conhecer as principais linguagens de programação;
- entender:
  - a organização de um sistema operacional como software básico;
  - o funcionamento dos compiladores e interpretadores de linguagens de programação como software básico; e
  - e caracterizar os principais tipos de software (básico e aplicativos).

Não deixe de utilizar os fóruns associados a esta unidade para apresentar e discutir qualquer dificuldade encontrada. **Bons estudos!**

2. Software como um Ambiente Operacional

Sabemos, através da nossa primeira unidade, que os ambientes operacionais são organizados em duas categorias básicas - hardware e software -, conforme ilustra a figura abaixo:

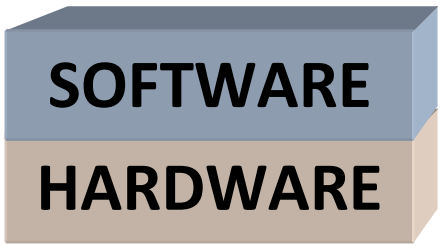

Figura 1: Organização de nível superior dos ambientes operacionais.

Pode-se pensar a camada de hardware como composta de todos os **equipamentos físicos** que suportam os processos computacionais. Por exemplo, processadores, memórias, monitores, cabos de rede e roteadores podem ser todos vistos como integrantes da camada de hardware.

Já a camada de software corresponde, essencialmente, aos **programas** utilizados nos processos computacionais. Comumente, os programas também são também conhecidos como **softwares**.

Normalmente, a maioria dos programas não se comunica diretamente com o hardware. Para intermediar esta comunicação, existe uma categoria especial de software, denominada **sistema operacional**, que disponibiliza uma forma padrão.

Essencialmente, o software pode ser classificado em dois grandes tipos:

1. **Software básico ou de sistema:** inclui os programas encontrados nas BIOS dos computadores (firmaware), os compiladores e interpretadores de linguagens, os sistemas operacionais e vários tipos de interfaces gráficas que, em conjunto, permitem ao usuário interagir com o computador e seus periféricos. **Exemplos:** Microsoft Windows (sistema operacional), Linux (sistema operacionals), Visual C++ (compilador), Java (interpretador), KDE (interface gráfica).
2. **Software aplicativo:** corresponde a programas, cujo objetivo é resolver problemas específicos de um usuário como, por exemplo, edição de textos (Microsoft Word), cálculos (Microsoft Excel) ou armazenamento/gerenciamento de dados (Microsoft Access).

3. Sistema Operacional como um Software Básico

Do ponto de vista:

- **Dos ambientes operacionais**: os sistemas operacionais representam uma das categorias mais importantes de programas. Conforme já colocado anteriormente, um sistema operacional funciona como um mediador entre diversos programas e os sistemas de hardware.
- **Estrutural:** um sistema operacional é formado por quatro módulos básicos, conforme mostrado na figura abaixo:

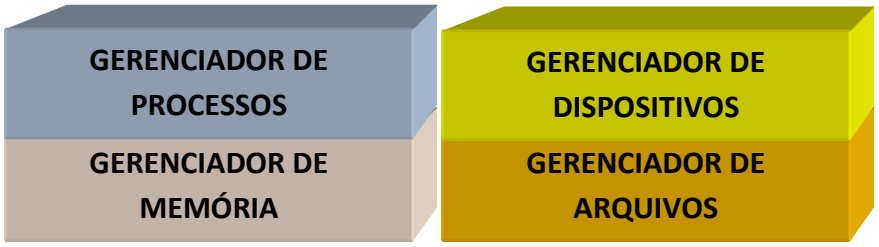

Figura 2: Módulos de um sistema operacional.

Cada programa em execução em um sistema operacional é chamado de processo. O módulo capaz de gerenciar estes programas é denominado **gerenciador de processos**. Para que os programas possam ser carregados em memória, é necessário decidir como eles serão carregados e qual a porção de memória será alocada ao programa. Todas estas decisões são tomadas pelo **gerenciador de memória.**

O controle dos dispositivos de E/S é efetuado pelo **gerenciador de dispositivos** que, dentre outras tarefas, é responsável pelo acesso aos *device drivers*, programas que permitem a comunicação com os dispositivos. Os dispositivos com grande capacidade de armazenamento, como um HD por exemplo, necessitam de meios mais eficientes de acesso aos dados armazenados. Normalmente, utiliza-se a estrutura de arquivos e diretórios para aumentar a eficiência deste acesso. Estas estruturas são controladas pelo **gerenciador de arquivos**.

4. Compiladores e Interpretadores

Outra categoria muito importante, principalmente para os desenvolvedores de programas, são os compiladores e interpretadores.

Um compilador permite, por exemplo, transformar programas escritos em uma linguagem de programa de alto nível (C, C++, Pascal, dentre outras) para a linguagem binária, que o computador consegue entender e executar. Já o interpretador, transforma diretamente as instruções do programa de alto nível para ações de execução diretas do processador.

Um compilador é um programa de computador (ou um grupo de programas) que, a partir de um código-fonte escrito em uma linguagem de alto ou médio nível, cria um programa semanticamente equivalente, porém escrito em outra linguagem, código objeto.

O nome "compilador" é usado, principalmente, para os programas que traduzem o código de fonte de uma linguagem de programação de alto nível para uma linguagem de programação de baixo nível (por exemplo, linguagem de montagem *Assembly* ou código de máquina).

Um programa que traduz uma linguagem de programação de baixo nível para uma linguagem de programação de alto nível é um descompilador.

Um programa que faz uma tradução entre:

- entre linguagens de alto nível é normalmente chamado de um tradutor, filtro ou conversor de linguagem.
- uma linguagem de montagem e o código de máquina é denominado montador (*Assembler*).

O processo de compilação é realizado em diversas fases:

- **Análise léxica**

  É a primeira fase do compilador. A função do analisador léxico, também denominado scanner, é ler o código fonte, caracter a caracter, buscando a separação e identificação dos elementos componentes do programa fonte, denominados símbolos léxicos ou *tokens*.

  É também de responsabilidade desta fase a eliminação de elementos "decorativos" do programa, tais como espaços em branco, marcas de formatação de texto e comentários.

- **Análise sintática**

  É o processo de se determinar se uma cadeia de símbolos léxicos pode ser gerada por uma gramática. No caso de analisadores sintáticos *top-down*, temos a opção de escrevê-los à mão ou gerá-los de forma automática, mas os analisadores *bottom-up* só podem ser gerados automaticamente . A maioria dos métodos de análise sintática cai em uma dessas duas classes denominadas *top-down e bottom-up*.

  Entre os métodos *top-down*, os mais importantes são a análise sintática descendente recursiva e a análise sintática preditiva não-recursiva. Entre os métodos de análise sintática *bottom-up*, os mais importantes são a análise sintática de precedência de operadores, análise sintática LR canônico, análise sintática LALR e análise sintática SLR.

- **Análise semântica**

  As análises léxica e sintática não estão preocupadas com o significado ou semântica dos programas que elas processam. O papel do analisador semântico é prover métodos pelos quais as estruturas construídas pelo analisador sintático possam ser avaliadas ou executadas.

  As gramáticas livres de contexto não são suficientemente poderosas para descrever uma série de construções das linguagens de programação, como por exemplo, regras de escopo, regras de visibilidade e consistência de tipos.

  É papel do analisador semântico assegurar que todas as regras sensíveis ao contexto da linguagem estejam analisadas e verificadas quanto à sua validade. Um exemplo de tarefa própria do analisador semântico é a checagem de tipos de variáveis em expressões.

  Um dos mecanismos comumente utilizados por implementadores de compiladores é a Gramática de Atributos, que consiste em uma gramática livre de contexto acrescentada de um conjunto finito de atributos e um conjunto finito de predicados sobre estes atributos.

- **Geração de código intermediário**

  Na fase de geração de código intermediário, ocorre a transformação da árvore sintática em uma representação intermediária do código fonte. Um tipo popular de linguagem intermediária é conhecido como código de três endereços. Neste tipo de código, uma sentença típica tem a forma X := A op B, onde X,A e B são operandos e op uma operação qualquer.

Uma forma prática de representar sentenças de três endereços é através do uso de quádruplas (operador, argumento-1, argumento-2 e resultado). Este esquema de representação de código intermediário é preferido por diversos compiladores, principalmente aqueles que executam extensivas otimizações de código, uma vez que o código intermediário pode ser rearranjado de uma maneira conveniente com facilidade.

- **Otimização de código**

A otimização de código é a estratégia de examinar o código intermediário, produzido durante a fase de geração de código com objetivo de produzir, através de algumas técnicas, um código que execute com bastante eficiência.

O nome otimizador deve sempre ser encarado com cuidado, pois não se pode criar um programa que leia um programa P e gere um programa P´ equivalente, sendo melhor possível, segundo o critério adotado.

Várias técnicas/tarefas reúnem-se sob o nome de Optimização e consistem em detectar padrões dentro do código produzido e substituí-los por códigos mais eficientes. Entre as técnicas usadas estão:

- a substituição de expressões, que podem ser avaliadas durante o tempo de compilação pelos seus valores calculados;
- eliminação de subexpressões redundantes;
- desmembramento de laços;
- substituição de operações (multiplicação por *shifts*), entre outras.

Uma das técnicas de optimização mais eficazes e independente de máquina é a otimização de laços, pois laços internos são bons candidatos para melhorias. Por exemplo, em caso de computações fixas dentro de laços, é possível movê-las para fora dos mesmos, reduzindo o processamento.

- **Geração de código Assembly**

A fase de geração de código Assembly é a última fase da compilação.

A geração de um bom código objeto é difícil devido aos detalhes particulares das máquinas para os quais o código é gerado. Contudo, é uma fase importante, pois uma boa geração de código pode ser, por exemplo, duas vezes mais rápida que um algoritmo de geração de código ineficiente.

Nem todas as técnicas de optimização são independentes da arquitetura da máquina-alvo. Optimizações dependentes da máquina necessitam de informações, tais como os limites e os recursos especiais da máquina-alvo a fim de produzir um código mais compacto e eficiente. O código produzido pelo compilador deve se aproveitar dos recursos especiais de cada máquina-alvo.

Um interpretador, por sua vez, normalmente, efetua os passos de análise léxica, sintática e semântica. A partir da análise semântica, ele pode intepretar diretamente as instruções ou gerar um código para uma máquina virtual (*virtual machine*).

A linguagem Java e as linguagens do ambiente .NET são exemplos típicos de linguagens que dependem de máquinas virtuais (*JVM-Java Virtual Machine* ou *Framework .NET*).

Uma **linguagem de programação** é chamada de **compilada,** quando depender de um compilador para gerar os seus programas executáveis. Uma **linguagem de programação** é chamada de **interpretada,** quando depende de um interpretador para gerar os seus programas executáveis.

A seguir, faremos um pequeno estudo das linguagens mais comuns no mundo dos ambientes operacionais.

5. Linguagens de Programação

Nesta seção, faremos um pequeno passeio pelas principais linguagens de programação existentes no mercado, dando exemplos de programas para cada uma delas.

A linguagem C é uma linguagem de programação compilada de propósito geral, estruturada, imperativa, procedimental, de médio nível e padronizada, criada em 1972, por Dennis Ritchie, no AT&T Bell Labs, para desenvolver o sistema operacional UNIX (que foi originalmente escrito em *Assembly*). É classificada de médio nível pela própria definição desse tipo de linguagem.

A programação em linguagens de médio nível tem como característica não ser necessário conhecer o processador, ao contrário das linguagens de baixo nível.

As linguagens de baixo nível estão fortemente ligadas ao processador.

A linguagem C permite acesso de baixo nível com a utilização de código Assembly no meio do código fonte. Assim, o baixo nível é realizado por Assembly e não C. Desde então, espalhou-se por muitos outros sistemas, e tornou-se uma das linguagens de programação mais usadas, e influenciou muitas outras linguagens, especialmente C++, que foi, originalmente, desenvolvida como uma extensão para C. A seguir, temos um exemplo de programa escrito em C:

```
#include <stdio.h>

struct pessoa
{
   unsigned short int idade;
   char nome[51];                  /* vetor de 51 chars para o nome */
   unsigned long int rg;
};

int main(void)
{
   struct pessoa exemplo = {16, "Fulano", 123456789};
   printf("Idade: %hu\n", exemplo.idade);
   printf("Nome: %s\n", exemplo.nome);
   printf("RG: %lu\n", exemplo.rg);
   return 0;
}
```

A linguagem C++ (em português, lê-se "cê mais mais") é uma linguagem de programação multiparadigma e de uso geral. A linguagem é considerada de médio nível, pois combina características de linguagens de alto e baixo níveis. Desde os anos 1990, é uma das linguagens comerciais mais populares, sendo bastante usada também na academia por seu grande desempenho e base de utilizadores.

Bjarne Stroustrup desenvolveu o C++ (originalmente com o nome *C with Classes*, que significa C com classes em português) em 1983 no Bell Labs como um adicional à linguagem C. Novas características foram adicionadas com o tempo, como funções virtuais, sobrecarga de operadores, herança múltipla, templates e tratamento de exceções.

Após a padronização ISO realizada em 1998 e a posterior revisão realizada em 2003, uma nova versão do padrão da linguagem está em desenvolvimento. A seguir, tem-se um exemplo de programa escrito em C++:

```
#include <iostream>

class Passaro                    // classe base
{
public:
  virtual void MostraNome()
  {
    std::cout << "um passaro";
  }
  virtual ~Passaro() {}
};

class Cisne: public Passaro      // Cisne é um pássaro
{
public:
  void MostraNome()
  {
    std::cout << "um cisne";     // sobrecarrega a função virtual
  }
};

int main()
{
  Passaro* passaro = new Cisne;
   passaro->MostraNome();        // produz na saída "um cisne", e não "um pássaro"
   delete passaro;
}
```

Java é uma linguagem de programação orientada a objeto desenvolvida na década de 90 por uma equipe de programadores, chefiada por James Gosling, na empresa Sun Microsystems. Diferentemente das linguagens convencionais, que são compiladas para código nativo, a linguagem Java é compilada para um *bytecode*, que é executado por uma máquina virtual.

A linguagem de programação Java é a linguagem convencional da Plataforma Java, mas não sua única linguagem. Abaixo, tem-se um exemplo de programa escrito em Java:

```
import javax.swing.JOptionPane;
public class Soma{

 public static void main(String[]args){

  //declaração das variáveis
  String numeroA, numeroB;
  int numero1, numero2, soma;

  //pede dois números inteiros
  numeroA = JOptionPane.showInputDialog("Entre com o primeiro número inteiro");
  numeroB = JOptionPane.showInputDialog("Entre com o segundo número inteiro");

  //converte os números de string para inteiro
  numero1 = Integer.parseInt(numeroA);
  numero2 = Integer.parseInt(numeroB);

  //outra forma de conversão seria utilizar o método valueOf
  numero1 = Integer.valueOf(numeroA);
  numero2 = Integer.valueOf(numeroB);

  //efetua a soma dos números
  soma = numero1 + numero2;

  //mostra o resultado da soma para o usuário
  JOptionPane.showMessageDialog(null,"A soma dos números é: " +
  soma,"Resultado",JOptionPane.PLAIN_MESSAGE);
 } }
```

C# (ou C Sharp) é uma linguagem de programação orientada a objetos desenvolvida pela Microsoft como parte da plataforma .NET. A sua sintaxe orientada a objetos foi baseada no C++, mas inclui muitas influências de outras linguagens de programação, como Delphi e Java. Abaixo, tem-se um exemplo de programa escrito em C#:

```
using System;

public class Elefante {

        private Int32 _peso;

        public Int32 Peso
        {
          get { return this._peso; }
          set { this._peso = value; }
        }

        public Elefante() { }
                public Elefante(Int32 Peso)
        {
           this._peso = Peso;
        }

        public static Elefante operator ++(Elefante e)
        {
           return new Elefante(e.Peso + 1);
        }

      }
      class Program
      {
         static void Main(string[] args)
         {
            Elefante e = new Elefante(10);
            System.Console.WriteLine(e.Peso.ToString());
            e++;
            System.Console.WriteLine(e.Peso.ToString());
            System.Console.ReadKey();

            // @OUTPUT:
            //  10
            //  11
         }
      }
```

Pascal é uma linguagem de programação estruturada, que recebeu este nome em homenagem ao matemático Blaise Pascal. Foi criada em 1970 pelo suíço Niklaus Wirth, tendo em mente encorajar o uso de código estruturado. Esta linguagem dá suporte a um ambiente importante de programação chamado Delphi. A seguir, tem-se um exemplo de programa escrito em Pascal:

```
program name;

 uses
   crt; (* biblioteca do pascal *)

 var
   n1,n2,q1,q2:string; (*variáveis criadas pelo usuario do tipo string*)

 begin

   {Limpa a tela}
   clrscr;

   writeln('Digite o primeiro nome:');
   readln(n1);

   writeln('Digite o segundo nome:');
   readln(n2);

   writeln('Digite uma qualidade para o primero nome:');
   readln(q1);

   writeln('Digite uma qualidade para o segundo nome:');
   readln(q2);

   writeln(n1,' ',q1);
   writeln(n2,' ',q2);

   {Aguarda até uma tecla ser pressionada}
   readkey;

 end.
```

LISP é uma família de linguagens de programação concebida por John McCarthy em 1958. Num célebre artigo, ele mostra que é possível usar exclusivamente funções matemáticas como estruturas de dados elementares (o que é possível a partir do momento em que há um mecanismo formal para manipular funções: o Cálculo Lambda de Alonzo Church. Durante os anos de 1970 e 1980, Lisp tornou-se a principal linguagem da comunidade de Inteligência Artificial, tendo sido pioneiro em aplicações como administração automática de armazenamento, linguagens interpretadas e programação funcional. Existem dois dialetos importantes de LISP:

- Common LISP e
- Scheme

A seguir, tem-se um exemplo de um mesmo programa escrito nestes dois dialetos:

Common Lisp:

```
(defun fatorial (n)
  (do ((i n (- i 1))
       (resultado 1 (* resultado i)))
      ((= i 0) resultado)
)
)
```

Scheme:

```
(define fatorial
   (lambda (n)
     (let f ((i n) (resultado 1))
       (if (= i 0)
         resultado
         (f (- i 1) (* resultado i))))
)
)
```

PROLOG é uma linguagem de programação que se enquadra no paradigma de Programação em Lógica Matemática. É uma linguagem de uso geral que é especialmente associada com a inteligência artificial e linguística computacional. Consiste numa linguagem puramente lógica, que pode ser chamada de PROLOG puro, e numa linguagem concreta, a qual acrescenta o PROLOG puro com componentes extralógicos. Trata-se de uma linguagem muito importante para Inteligência Artificial.

O programa abaixo ilustra uma utilização da linguagem PROLOG:

```
filho(X,Y) :- pai(Y,X).

filho(X,Y) :- mae(Y,X).

 mae(marcia, ana).

 pai(tomas, ana).

 pai(tomas, erica).

 pai(marcos, tomas).
```

Neste exemplo, definiu-se uma base de conhecimento para definir que é mãe e quem é pai de quem. A partir desta base de conhecimento, pode-se definir regras para quem é filho de quem:

- X é filho de Y se Y é pai de X ou
- X é filho de Y se Y é mãe de X

A partir deste conjunto de regras e base de conhecimento, a linguagem PROLOG pode responder a consultas.

Finalmente, a linguagem Ada é uma Linguagem de programação estruturada, de tipagem estática, é uma linguagem imperativa, orientada a objetos e é uma linguagem de alto nível, originada de Pascal e outras linguagens. Foi, originalmente, produzida por uma equipe liderada por Jean Ichbiah da CII Honeywell Bull, contratados pelo Departamento de Defesa dos Estados Unidos durante a década de 70, com o intuito de substituir as centenas de linguagem de programação usadas pelo DoD.

Ada é uma aplicação com compiladores validados para uso confiável em missões criticas, tais como softwares de aviação. Normatizada internacionalmente pela ISO, sua versão mais atual é de 2005. Abaixo, tem-se um exemplo de programa escrito em ADA:

```
With text_IO;
Use text_IO;
With Ada.Integer_Text_IO;
Use Ada.Integer_Text_IO;

Procedure usando_for_while_loop is
        a: natural;

begin
        New_Line(3);

        for a in 1..3 loop
                Put_Line("Usando o Comando For em Ada.");
        end loop;
        New_Line(3);
        a := 1;
        while a /=5 loop
                Put_Line("Usando o Comando While em Ada.");
                a := a + 1;
        end loop;
        New_Line(3);
        a := 1;
        loop
                Put_Line("Usando o Comando Loop em Ada.");
                exit when a=5;
                a := a + 1;
        end loop;
        New_Line(3);

end usando_for_while_loop;
```

Isto encerra o nosso conteúdo da disciplina de Tópicos de Computação e Informática. Na próxima e última unidade, faremos uma revisão de todo o conteúdo desenvolvido até o momento nestas cinco unidades.

Material Complementar

Quer conhecer quais são todas as linguagens de programação catalogadas no mundo? Então, acesse o site abaixo:

http://www.scriptol.com/programming/list-programming-languages.php

Anotações

Referências

MMOKARZEL, F.; SOMA, N. **Introducao a Ciencia da Computacao**. São Paulo: Campus, 2008.

MONTEIRO, M. **Introducao a Organizacao de Computadores**. 5. ed. Rio de Janeiro: LTC, 2007.

Educação a Distância
Cruzeiro do Sul Educacional
Campus Virtual

www.cruzeirodosulvirtual.com.br
Campus Liberdade
Rua Galvão Bueno, 868
CEP 01506-000
São Paulo SP Brasil
Tel: (55 11) 3385-3000

Universidade Cruzeiro do Sul

Módulo Centro Universitário

UDF Centro Universitário

UNICID Universidade Cidade de S. Paulo