Montadores, Compiladores, Ligadores e Interpretadores

No nosso dia a dia, utilizamos programas de computadores para os mais diversos fins, seja para nos auxiliar em nosso trabalho, seja para usos pessoais.

Um **programa de computador** é um **conjunto de instruções** que:

- Possui um **determinado fim**;
- Por exemplo, o Photoshop para edição de imagens; e
- É **executado** por um **processador**.

Mas para haja essa execução pelo processador, o programa precisa estar em uma linguagem que o processador possa entender: a linguagem de máquina. Ela é a linguagem que um processador é capaz de compreender e é composta de apenas de números 0 e 1.

Para executar uma tarefa qualquer, um computador precisa receber instruções precisas sobre o que fazer. Uma seqüência adequada de instruções de computador, para a realização de uma determinada tarefa, se constitui num PROGRAMA de computador. Uma linguagem de programação é um conjunto de ferramentas, regras de sintaxe e símbolos ou códigos que nos permitem escrever programas de computador, destinados a instruir o computador para a realização de suas tarefas.

A primeira e mais primitiva linguagem de computador é a própria linguagem de máquina, aquela que o computador entende diretamente e pode ser diretamente executada pelos circuitos do processador (pelo hardware). No início da era da computação, os programas eram escritos em linguagem de máquina, isto é, as instruções eram escritas diretamente na linguagem do computador (formada apenas com 1's e 0's). Um programa em linguagem de máquina é uma longa série de 0's e 1's, ordenados de forma que alguns representam códigos de instruções e outros representam os dados que serão processados (ou indicam onde esses dados estão armazenados). Em um programa escrito em linguagem de máquina, cada instrução escrita pelo programador será individualmente executada, isto é, a cada instrução do programa corresponderá uma ação do computador

. A relação é portanto 1 para 1 – uma instrução do programa corresponde a uma operação do computador.

Imagine então um programa extenso escrito apenas usando 1's e 0's; imagine que para cada diferente marca ou modelo de computador, as regras para entender esses códigos serão totalmente diferentes; e finalmente imagine que voce teria que escrever uma a uma as instruções e os dados adequadamente codificados e ordenados, perfurar todos o programa em cartões e submeter toda a massa de cartões ao computador, para finalmente receber algumas horas depois o seu programa de volta com uma mensagem de erro tipo "erro no cartão X" … e mais nada! Um programa escrito nessa linguagem era difícil de ser escrito sem que se cometessem muitos erros, processo esse longo, difícil, entediante e principalmente caro.

Um programa em linguagem de máquina era também extremamente difícil de ser entendido por outros programadores que futuramente viessem a trabalhar na manutenção do programa. Essa complexidade levou à necessidade de se desenvolverem técnicas e ferramentas para tornar a escrita e manutenção de programas mais fácil, mais rápida e principalmente mais barata.

Cada família de computadores possui sua própria linguagem de máquina. Um programa em linguagem de máquina é dependente do computador ou seja, tendo sido escrito para um determinado computador, somente poderá ser executado em computadores da mesma família, que lhe sejam 100% compatíveis.

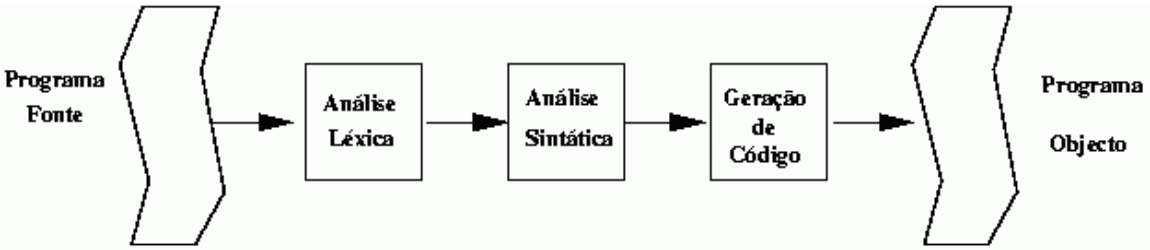
Tradução

Um programa escrito por um programador (chamado código fonte) em uma linguagem de alto nível é um conjunto de instruções que é clara para programadores, mas não para computadores. Ou seja, os computadores entendem única e exclusivamente suas linguagens nativas, as linguagens de máquina.

Programas em linguagem de alto nível, a exemplo dos programas escritos em linguagem de Montagem, também precisam ser traduzidos para linguagem de máquina para poderem ser submetidos ao computador e processados.

O processo de tradução do programa escrito em uma linguagem simbólica pelo programador, chamado código fonte (source code) para a linguagem de máquina do computador chamada código objeto (object code), é chamado compilação e é realizado por um programa chamado Compilador (Compiler).

Tradução sucessiva de uma linguagem para outra intermediária mais simples, da linguagem fonte até à linguagem objeto.



Cada tradução pode dividir-se nas seguintes fases:

Análise Léxica (scanner)

Ele separa a sequência de caracteres que representa o programa fonte em entidades ou tokens, símbolos básicos da linguagem. Durante a análise léxica, os tokens são classificados como palavras reservadas, identificadores, símbolos especiais, constantes de tipos básicos (inteiro real, literal, etc.), entre outras categorias. Basicamente é reconhecer as sequências de símbolos que representam uma unidade. Ex.: o nome de uma variável, uma constante, uma palavra chave de uma instrução (while).

Um token consiste de um par ordenado (valor, classe). A classe indica a natureza da informação contida em valor.

Outras funções atribuídas ao analisador léxico são: ignorar espaços em branco e comentários, e detectar erros léxicos.

Análise Sintática (parser)

Ele agrupa os tokens fornecidos pelo analisador léxico em estruturas sintáticas, construindo a árvore sintática correspondente. Para isso, utiliza uma série de regras de sintaxe, que constituem a gramática da linguagem fonte. É a gramática da linguagem que define a estrutura sintática do programa fonte.

O analisador sintático tem também por tarefa o reconhecimento de erros sintáticos, que são construções do programa fonte que não estão de acordo com as regras de formação de estruturas sintáticas como especificado pela gramática.

Identifica a estrutura gramatical do programa e reconhece o papel de cada componente. É normalmente construída uma árvore sintática do programa e uma tabela de símbolos, que identifica variáveis.

Analisador Semântico

O compilador executa ainda a análise semântica. O analisador semântico utiliza a árvore sintática determinada pelo analisador sintático para: identificar operadores e operandos das expressões, reconhecer erros semânticos, fazer verificações de compatibilidade de tipo, analisar o escopo das variáveis, fazer verificações de correspondência entre parâmetros atuais e formais.

Fundamentalmente, a análise semântica trata os aspectos sensíveis ao contexto da sintaxe das linguagens de programação. Por exemplo, não é possível representar em uma gramática livre de contexto uma regra como “Todo identificador deve ser declarado antes de ser usado.“, e a verificação de que essa regra foi aplicada cabe à análise semântica.

Otimização de Código

O processo de otimização de código consiste em melhorar o código intermediário de tal forma que o programa objeto resultante seja mais rápido em tempo de execução. Por exemplo, um algoritmo para geração do código intermediário gera uma instrução para cada operador na árvore sintática, mesmo que exista uma maneira mais otimizada de realizar o mesmo comando.

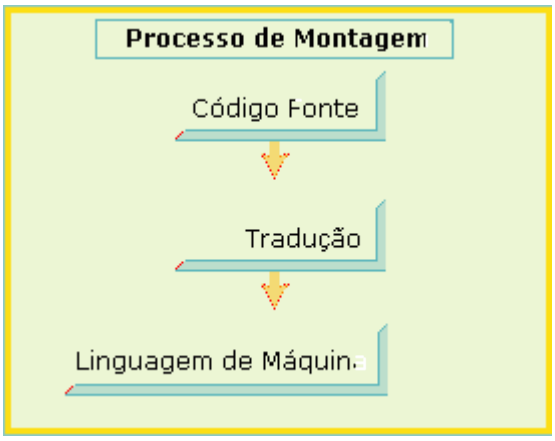
Geração de Código

A fase final do compilador é a geração do código para o programa objeto, consistindo normalmente de código em linguagem assembly ou de código em linguagem de máquina. Aqui é o processo de construir instruções da linguagem máquina (em assembly, normalmente) que simulam as instruções reconhecidas pelo analisador sintáctico. A geração de código pode englobar: análise semântica, geração de código intermédio, otimizadores e geração de código final.

Montagem

Citamos anteriormente uma forma de tradução rápida e simples: a executada pelo programa Montador. O processo de montagem traduz um programa escrito em linguagem Assembly em um programa equivalente em linguagem de máquina, possível de ser executado pelo computador.

A seguir, é apresentado o fluxo que representa o processo de montagem.



No processo de montagem, o código fonte (programa em linguagem simbólica escrito pelo programador) é examinado, instrução por instrução e é feita a tradução, gerando o código que será executado (código objeto). Os passos executados pelo programa Montador são:

1. Verificar a correção do código de instrução (se o mnemônico corresponde a uma instrução válida para o computador, se os campos definidos na estrutura da linguagem e a sintaxe estão corretos) e substituir os mnemônicos pelos códigos numéricos binários equivalentes. Qualquer erro no código acarreta a interrupção do processo e a emissão de mensagem de erro.

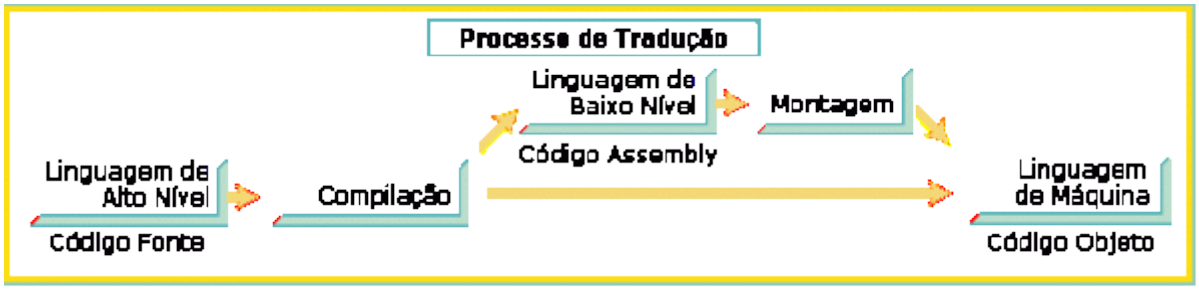
2. Resolver as referências de memória: os nomes simbólicos adotados pelo programador são convertidos para endereços reais de memória (valores numéricos binários de endereços).

3. Reservar espaço em memória para o armazenamento das instruções e dados.

4. Converter valores de constantes em binário.

Compilação

Compilação é o processo de tradução de um programa escrito em linguagem de alto nível para código em linguagem de máquina. Compilação é um processo análogo ao da montagem (verificação / análise do código fonte, resolução das referências de memória, reserva de espaço em memória e conversão para código de máquina binário). O que diferencia a compilação do processo de montagem é sua maior complexidade. No processo de montagem, há uma relação de 1:1, ou seja, cada instrução do código fonte resulta em uma instrução de máquina, enquanto na compilação a relação é múltipla, cada instrução do código fonte gerando várias instruções de máquina.



Durante a compilação, o código fonte é analisado (análise léxica, sintática e semântica), é gerado um código intermediário e são construídas tabelas de símbolos, alocam-se as áreas de memória para variáveis e atribui-se os registradores a serem utilizados, e é finalmente gerado o código objeto em linguagem binária de máquina. Em alguns compiladores, é gerado um código intermediário em Assembly (que pode ser visualizado pelo programador) e que em seguida passa pelo montador para gerar finalmente o código objeto em linguagem de máquina.

O código objeto pode ser absoluto (os endereços constantes são endereços reais de memória) ou relocável (os endereços são relativos, tendo como referência o início do programa, e os endereços reais de memória são definidos apenas em tempo de execução).

Tipos de Compiladores:

- **Single-Pass:** compilação numa única leitura do programa fonte
- **Multi-Pass:** compilação através de várias leituras do programa fonte
- **Load-And-Go:** compilação e a execução do programa fonte
- **Debugging:** compilação permitindo a depuração do programa fonte
- **Optimizing:** compilação e a otimização do programa alvo

Vantagens:

- O código compilado é mais rápido de ser acessado;
- Impossibilita ou pelo menos dificulta ser quebrado e visualizado o código-fonte original;
- Permite otimização do código por parte do compilador;
- Compila o código somente se estiver sem algum erro.

Desvantagens:

- Para ser utilizado o código precisa passar por muitos níveis de compilação;
- Assim como vantagem a possibilidade de não poder visualizar o código-fonte, pode ser uma desvantagem;
- Processo de correção ou alteração do código requer que ele seja novamente recompilado.

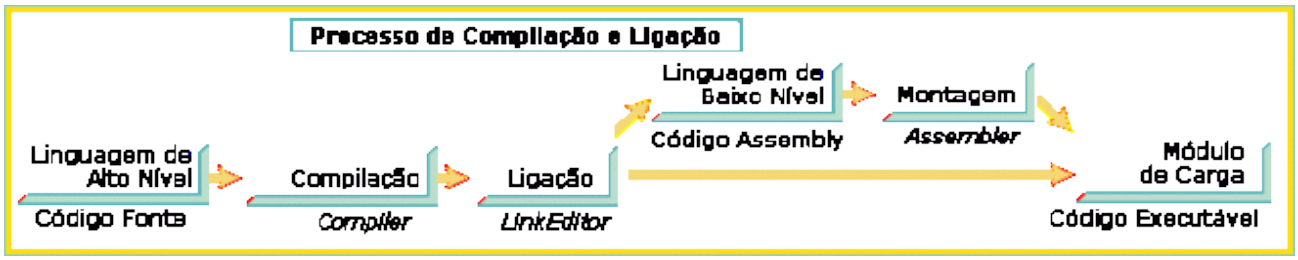
Bibliotecas

O desenvolvimento de um programa certamente utilizará diversas operações que são comuns a muitos outros programas. Por exemplo, a execução de uma instrução de entrada e saída, a classificação dos dados de um arquivo, o cálculo de funções matemáticas, etc. Uma linguagem de alto nível geralmente incorpora diversas rotinas prontas (que fazem parte da linguagem) e que compõem bibliotecas (librarys) de funções pré-programadas que poderão ser utilizadas pelo programador, poupando tempo, aumentando a eficiência e evitando erros. Dessa forma, um programa em alto nível possivelmente conterá diversas chamadas de biblioteca (library calls). Essas funções não devem ser confundidas com as instruções da linguagem – na realidade, são pequenos programas externos que são chamados através de instruções especiais de chamada de biblioteca. Para serem executadas, essas rotinas precisam ser incorporadas ao código do programador, isto é, a chamada de biblioteca precisa ser substituída pelo código propriamente dito, incluindo os parâmetros necessários.

Ligação

Assim, o código objeto preparado pelo compilador em geral não é imediatamente executável, pois ainda existe código (as rotinas de biblioteca) a ser incorporado ao programa. A cada chamada de biblioteca encontrada no código fonte, o compilador precisará incluir uma chamada para a rotina e o endereço dos dados que devam ser passados para a rotina.

A tarefa de examinar o código objeto, procurar as referências a rotinas de biblioteca (que constituem referências externas não resolvidas), buscar a rotina da biblioteca, substituir a chamada pelo código (“resolver as referências externas”) e obter os parâmetros para incluí-los no código objeto é executada por um programa chamado Ligador (LinkEditor). O resultado da execução do Ligador é o código final pronto para ser executado pelo computador, chamado módulo de carga ou código executável.

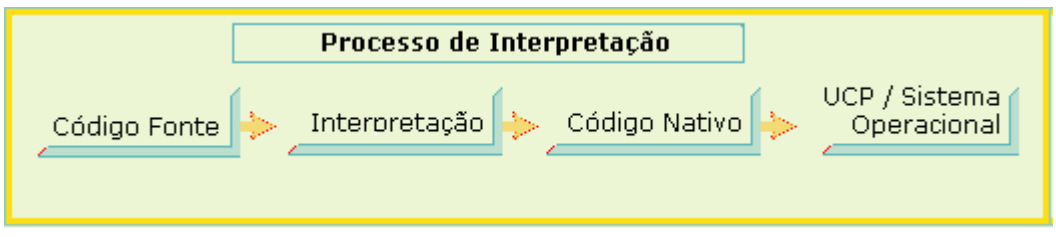


O módulo de carga, após testado e depurado (isto é, depois de resolvidos todos os erros, também chamados “bugs”) é armazenado em memória de massa para ser executado quando necessário. O processo de compilação e ligação é executado apenas pelo programador na fase de desenvolvimento e não mais precisará ser executado pelo usuário, quando da execução do programa.

Interpretação

Com o processo de execução de um programa em fases distintas (compilação / ligação / execução) apresentado, um programa para ser executado precisa primeiro ter sido convertido para código objeto pelo compilador e depois ter passado pelo ligador. Esse processo é o mais largamente utilizado, porém não é o único.

O método alternativo chama-se de interpretação e, a partir do programa fonte, realiza as três fases (compilação, ligação e execução), comando por comando, em tempo de execução. Não existem fases distintas nem se produzem códigos intermediários. Todo o processo de conversão é efetuado em tempo de execução e imediatamente executado. Ou seja, cada comando é lido, verificado, convertido em código executável e imediatamente executado, antes que o comando seguinte seja sequer lido.



Linguagens como C, Pascal, COBOL, etc, são linguagens tipicamente compiladas, enquanto o BASIC foi desenvolvido como linguagem interpretada (hoje também existem linguagens BASIC compiladas e o programador pode optar). As linguagens de programação tipicamente de usuário, tais como das planilhas Excel, o Word Basic (linguagem de construção de Macros do Word), o Access, etc, são todas linguagens interpretadas.

Uma vantagem é que o ciclo de escrita, execução, modificação é mais rápido em relação à compilação. Por outro lado, a execução é mais lenta.

No programa em linguagem de alto nível, os interpretadores executam os passos definidos para cada instrução e produzem o mesmo resultado que o do programa compilado. Entretanto, a execução de um programa em linguagem de alto nível com o uso de interpretadores é mais lenta que a execução de um programa compilado, uma vez que precisa examinar cada instrução no programa-fonte, à medida que ela ocorre, e desviar para a rotina que executa a instrução.

Vantagens:

- Correções e alterações são mais rápidas de serem realizadas;
- Código não precisa ser compilado para ser executado;
- Consomem menos memória.

Desvantagens:

- Execução é mais lenta do programa;
- Necessita sempre ser lido o código original para ser executado;

Veja o comparativo abaixo:

	Vantagens	Desvantagens
Compiladores	Execução mais rápida	Várias estapas de tradução
	Permite estruturas de programação mais completas para a sua execução	Programação final é maior necessitando mais memória
	Permite a otimização do código fonte	Processo de correção de erros e depuração é mais demorado
Interpretadores	Depuração do programa é mais simples	Execução do programa é mais lenta
	Consome menos memória	Estruturas de dados demasiado simples
	Resultado imediato do programa ou rotina desenvolvida	Necessário fornecer o programa fonte ao utilizador

Bytecode

Traduzindo ao pé da letra, código em bytes — não confundir com código-máquina, é o resultado de um processo semelhante ao dos compiladores de código-fonte que não é imediatamente executável. Em oposição, o bytecode irá ser interpretado numa máquina virtual, que fará a execução. Assim, o bytecode é um estágio intermédio entre o código-fonte (escrito numa linguagem de programação específica) e a aplicação final, sendo a sua vantagem principal a dualidade entre a portabilidade — o bytecode irá produzir o mesmo resultado em qualquer arquitetura— e a ausência da necessidade do pré-processamento típico dos compiladores — o bytecode é encarado como um produto final, cuja validação da sintaxe e tipos de dados (entre outras funções dos compiladores) não será necessária.

Compilador:

Compilador é um programa ou um grupo de programas que escrito por uma linguagem (esta sendo necessária de compilação para sua execução) ao ser compilado gera outro código que é interpretado pelo computador. Este código compilado é chamado de código objeto, podendo ser um arquivo executável que é reproduzido em um sistema operacional, por exemplo. Este tipo de tradutor é um dos mais utilizados.

Os compiladores analisam o código em três partes, de forma sintática ou hierárquica, análise léxica ou linear e anâlise semântica.

Alguns compiladores contam com um pré-processamento. Este pré-processamento é responsável por modificar o código-fonte conforme necessidades que o compilador identifique que sejam necessárias, por exemplo, otimização de código.

Vantagens:

O código compilado é mais rápido de ser acessado;

- Impossibilita ou pelo menos dificulta ser quebrado e visualizado o código-fonte original;

- Permite otimização do código por parte do compilador;

- Compila o código somente se estiver sem algum erro.

Desvantagens:

Para ser utilizado o código precisa passar por muitos níveis de compilação;

- Assim como vantagem a possibilidade de não poder visualizar o código-fonte, pode ser uma desvantagem;

- Processo de correção ou alteração do código requer que ele seja novamente recompilado.

Interpretador:

O interpretador ao contrário do compilador roda o código-fonte escrito como sendo o código objeto, ele traduz o programa linha a linha, o programa vai sendo utilizado na medida em que vai sendo traduzido. Cada execução do programa precisa ser novamente traduzido e interpretado.

O interpretador analisa sintaticamente e semanticamente o código, se estas duas etapas forem realizadas e executadas de forma correta o código está pronto para funcionar.

Vantagens:

- Correções e alterações são mais rápidas de serem realizadas;

- Código não precisa ser compilado para ser executado;

- Consomem menos memória.

Desvantagens:

- Execução é mais lenta do programa;

- Necessita sempre ser lido o código original para ser executado;

s processadores CISC (Complex Instruction Set Computers) já possuem instruções complexas que são interpretadas por micro-programas. O número de registradores é pequeno e qualquer instrução pode referenciar a memória principal. Neste tipo de arquitetura, a implementação do pipeline é mais difícil. São exemplos de processadores CISC o VAX (DEC), 80x86 e o Pentium (Intel), e o 68xx (Motorola).

Curso de Técnico em Informática – ETE Taquarituba

Software

O Hardware por si só não tem a menor utilidade. Para torná-lo útil existe um conjunto de programas, utilizado como interface entre as necessidades do usuário e as capacidades do hardware. A utilização de softwares adequados às diversas tarefas e aplicações (conceitos de camadas) torna o trabalho do usuários muito mais simples e eficiente.

Tradutor

Nos sistemas operacionais antigos, o ato de programar era bastante complicado, já que o programador deveria possuir conhecimento do hardware e programar em painéis através de fios. Esses programas eram desenvolvidos em linguagem de máquina e carregados diretamente na memória principal para execução.

Com o surgimento das primeiras linguagens de montagem (assembly languages) e das linguagens de alto nível, o programador deixou de se preocupar com muitos aspectos pertinentes ao hardware, como em qual região da memória o programa deveria ser carregado ou quais endereços de memória seriam reservados para as variáveis. A utilização dessas linguagens facilitou a construção de programas em muitos aspectos. Desse modo, um programa poderia ser escrito de uma forma bem documentada e com facilidades para realizar alterações.

O tradutor, pelo tipo de linguagem de programação utilizada, pode ser chamado de montador ou compilador (Figura 6).

Programa-Fonte Programa-Fonte Programa-Objeto

Linguagem de Montagem Módulo-Objeto

Módulo-Objeto

Montador

Compilador

Linguagem de Alto Nível

Figura 6 - Tradutor

Compilador

É o utilitário responsável por gerar, a partir de um programa escrito em uma linguagem de alto nível, um programa em linguagem de máquina não executável.

As linguagens de alto nível, como pascal, fortran, cobol não tem nenhuma relação direta com a máquina, ficando essa preocupação exclusivamente com o compilador.

Interpretador

O interpretador é considerado um tradutor que não gera código-objeto. A partir de um programa fonte, escrito em linguagem de alto nível, o interpretador, no momento da execução do programa, traduz cada instrução e a executa em seguida.

Linker

O linker (ligador), também chamado de linkagem, é o utilitário responsável por gerar, a partir de um ou mais módulos-objetos, um único programa executável.

Curso de Técnico em Informática – ETE Taquarituba

Módulo

Objeto Compilador

Módulo Fonte

Módulo Objeto

Módulo Fonte LinkerCompilador

Módulo

Objeto Compilador

Módulo Fonte

Programa Executável

Figura 7 - Linker.

Loader

Também chamado carregador é o utilitário responsável por colocar fisicamente na memória um programa para execução. O procedimento de carga varia com o código gerado pelo linker e, em função deste, o loader é classificado como sendo do tipo absoluto ou relocável.

Tipo absoluto - o loader só necessita conhecer o endereço de memória inicial e o tamanho do módulo para realizar o carregamento. Então, ele transfere o programa da memória secundária para a memória principal e inicia sua execução.

No caso de código relocável, o programa pode ser carregado em qualquer posição de memória, e o loader é responsável pela relocação no momento do carregamento.

Depurador

O desenvolvimento de programas está sujeito a erros de lógica, independentemente de metodologias utilizadas pelo programador. A depuração é um dos estágios desse desenvolvimento, e a utilização de ferramentas adequadas é essencial para acelerar o processo de correção de programas.

O depurador (debbuger) é o utilitário que permite ao usuário controlar a execução de um programa a fim de detectar erros na sua estrutura. Este utilitário oferece ao usuário recursos como: • Acompanhar a execução de um programa instrução por instrução;

• Possibilitar a alteração e visualização do conteúdo de variáveis;

• Implementar pontos de parada dentro do programa (break-point), de forma que, durante a execução, o programa pare nesses pontos;

• Especificar que, toda vez que o conteúdo de uma variável for modificado, o programa envie uma mensagem (watchpoint).

Linguagem de Controle

Também denominada a linguagem de comando, e a forma mais direta de um usuário se comunicar com o sistema operacional. Esta linguagem é oferecida por cada sistema operacional para que, através de comandos simples, o usuário possa ter acesso a rotinas especificas do sistema.

Curso de Técnico em Informática – ETE Taquarituba

Interpretador de Comandos (Shell)

O sistema operacional é o código executor de chamadas de sistema. Os editores, compiladores, montadores, ligadores e interpretadores de comando não fazem parte do sistema operacional, apesar de serem softwares muito importantes e muito úteis. Esses comandos quando digitados pelos usuários, são interpretados pelo Shell, verifica sua sintaxe, envia mensagens de erro e faz chamadas a

rotinas do sistema. Dessa forma o usuário dispõe de uma interface interativa com o sistema operacional, para realizar tarefas como acessar uma arquivo em disco ou consultar um diretório.

Linguagem de Máquina

A linguagem de máquina de um computador é a linguagem de programação que o processador realmente consegue entender. Cada processador possui um conjunto único de instruções de máquina, definido pelo próprio fabricante. As instruções especificam detalhes, como registradores, modos de endereçamento e tipos de dados, que caracterizam um processador e suas potencialidades.

Microprogramação

Um programa em linguagem de máquina é executado diretamente pelo hardware em processadores de arquitetura RISC, porém em máquinas CISC isto não acontece. Neste caso, como podemos observar na Figura 3, entre os níveis de linguagem de máquina e do hardware, existem ainda o da microprogramação.

Os microprogramas definem a linguagem de máquina de cada computador. Apesar de cada computador possui níveis de microprogramação diferentes, existem muitas semelhanças nessa camada se compararmos os diversos equipamentos. Uma máquina possui, aproximadamente 25 microintruções básicas, que são interpretadas pelos circuitos eletrônicos.

Processos

Um conceito chave da teoria dos sistemas operacionais é o conceito de processo. Um processo é basicamente um programa em execução, sendo constituído do código executável, dos dados referentes ao código.

Chamadas de Sistema

Os programas de usuário solicitam serviços do sistema operacional através da execução de chamadas de sistema. A cada chamada corresponde um procedimento de uma biblioteca de procedimentos que o programa do usuário pode chamar.

Arquivos

Arquivos são mecanismos de abstração que fornece uma forma de armazenar recuperar informações em disco. Isto deve ser feito de uma forma que mantenha o usuário isolado dos detalhes a respeito de como as informações são armazenadas, e de como os discos efetivamente trabalha.

Curso de Técnico em Informática – ETE Taquarituba

Tipos de Sistemas Operacionais

Os tipos de sistemas operacionais e sua evolução estão intimamente relacionados com a evolução do hardware e das aplicações por ele suportadas.

Muitos termos inicialmente introduzidos para definir conceitos e técnicas acabam sendo substituídos por outros, na tentativa de refletir uma nova maneira de interação ou processamento. Isto fica muito claro quanto tratamos da unidade de execução do processador. Inicialmente, os termos programa ou job eram os mais utilizados, depois surgiu o conceito de processo e sub-processo e, mais recentemente, os conceitos de tarefa e de thread.

A evolução dos sistemas operacionais para computadores pessoais e estações de trabalho popularizou vários conceitos e técnicas, antes só conhecidos em ambientes de grande porte. A nomenclatura, no entanto, não se manteve a mesma. Surgiram novos termos para conceitos já conhecidos, que foram apenas adaptados para uma nova realidade.

Tipos de Sistemas Operacionais

Sistemas

Monoprogamáveis/ Monotarefa

Sistemas

Multiprogramáveis/ Multitarefa

Sistemas com

Múltiplos Processadores

Figura 8 - Tipos de sistemas operacionais

Tipos de Sistemas Operacionais Sistemas Monoprogramáveis/Mono-tarefa Sistemas Multiprogramáveis/Multitarefa Sistemas Batch

Sistemas de Tempo Compartilhado Sistemas de Tempo Real

Sistemas com Múltiplos Processadores

Sistemas Fortemente Acoplados Sistemas Simétricos Sistemas Assimétricos Sistemas Fracamente Acoplados Sistemas Operacionais de Rede Sistemas Operacionais Distribuídos

Sistemas Monoprogramáveis/Monotarefa

Os primeiros sistemas operacionais eram tipicamente voltados para a execução de um único programa (job). Qualquer outro programa, para ser executado, deveria aguardar o término do programa corrente. Os sistemas monoprogramáveis, como vieram a ser conhecidos, se caracterizam por permitir que o processador, a memória e os periféricos permaneçam exclusivamente dedicados à execução de um único programa.

Curso de Técnico em Informática – ETE Taquarituba

Neste tipo de sistema, enquanto um programa aguarda por um evento, como a digitação de um dado, o processador permanece ocioso, sem realizar qualquer tipo de processamento. A memória é subtilizada caso o programa não a preencha totalmente, e os periféricos, como discos e impressoras, estão dedicados a um único usuário.

Comparados a outros sistemas, os sistemas monoprogramáveis/monotarefa são de simples implementação, não existindo muita preocupação com problemas de proteção.

UCP Memória

Dispositivos de E/S

Programa/ Tarefa

Figura 9 - Sistemas monoprogramáveis/monotarefa

Sistemas Multiprogramáveis/Multitarefa

Os Sistemas Multiprogramáveis, que vieram a substituir os monoprogramáveis, são mais complexos e eficientes. Enquanto em sistemas monoprogramáveis existe apenas um programa utilizando seus diversos recursos, nos multiprogramáveis vários programas dividem esses mesmos recursos.

As vantagens do uso de sistemas multiprogramáveis são o aumento da produtividade dos seus usuários e a redução de custos, a partir do compartilhamento dos diversos recursos do sistema.

A partir do número de usuários que interagem com o sistema, podemos classificar os sistemas multiprogramáveis como monousuário e multiusuário.

O conceito de sistemas multiprogramável está tipicamente associado aos mainframes e minicomputadores, onde existe a idéia do sistema sendo utilizado por vários usuários (multiusuário). No mundo dos computadores pessoais e estações de trabalho, apesar de existir apenas um único usuário interagindo como sistema (monousuário), é possível que ele execute diversas tarefas concorrentemente ou mesmo simultaneamente. Os sistemas multitarefa, como também são chamados, se caracterizam por

permitir que o usuário edite um texto, imprima um arquivo, copie um arquivo pela rede e calcule uma planilha. Abaixo estão relacionados os tipos de sistemas em função do número de usuários

Um usuário Dois ou mais usuários

Monoprogramação/Monotarefa Monousuário N/A Multiprogramação/Multitarefa Monousuário Multiusuário

Tabela 2 - Sistemas X Usuários

Os sistemas multiprogramáveis/multitarefa podem ser classificados pela forma com que suas aplicações são gerenciadas, podendo ser divididos em sistemas batch, de tempo compartilhado ou de tempo real. Um sistema operacional pode suportar um ou mais desses tipos de processamento.

Curso de Técnico em Informática – ETE Taquarituba

Sistemas Multiprogramáveis/Multirefa

Sistemas

Batch

Sistemas de Tempo compartilhado

Sistemas de Tempo Real

Figura 10 - Tipos de sistemas multiprogramáveis/multitarefa

Sistemas Batch

Os sistemas batch (lote) foram os primeiros sistemas multiprogramáveis a serem implementados e caracterizam-se por terem seus programas, quando submetidos, armazenados em disco ou fita, onde esperam para ser executados seqüencialmente.

Normalmente, os programas, também chamados de jobs, não exigem interação com os usuários, lendo e gravando dados em discos e fitas. Alguns exemplos de aplicações originalmente processadas em batch são compilações, linkedições, sorts, backups e todas aquelas onde não é necessária a interação com o usuário.

Sistemas de Tempo Compartilhado

Os sistemas de tempo compartilhado (time-sharing) permitem a interação dos usuários com o sistema, basicamente através de terminais que incluem vídeo, teclado e mouse. Dessa forma, o usuário pode interagir diretamente com o sistema em cada fase do desenvolvimento de suas aplicações e, se preciso, modificá-las imediatamente. Devido a esse tipo de interação, os sistemas de tempo compartilhado também ficaram conhecidos como sistemas on-line.

Para cada usuário, o sistema operacional aloca uma fatia de tempo (time-slice) do processador. Caso o programa do usuário não esteja concluído nesse intervalo de tempo, ele é substituído por um de outro usuário, e fica esperando por uma nova fatia de tempo. Não só o processador é compartilhado nesse sistema, mas também a memória e os periféricos, como discos e impressoras. O sistema cria para o usuário um ambiente de trabalho próprio, dando a impressão de que todo o sistema está dedicado, exclusivamente, a ele.

Sistemas de tempo compartilhado são de implementação complexa, porém, se levado em consideração o tempo de desenvolvimento e depuração de uma aplicação, aumentam consideravelmente a produtividade dos seus usuários, reduzindo os custos de utilização do sistema.

Sistemas de Tempo Real

Os sistemas de tempo real (real time) são bem semelhantes em implementação aos sistemas de tempo compartilhado. A maior diferença é o tempo de resposta exigido no processamento das aplicações.

Enquanto em sistemas de tempo compartilhado o tempo de resposta pode variar sem comprometer as aplicações em execução, nos sistemas de tempo real os tempos de resposta devem estar dentro de limites rígidos, que devem ser obedecidos, caso contrário poderão ocorrer problemas irreparáveis.

Não existe idéia de fatia de tempo, um programa detém o processador o tempo que for necessário, ou até que apareça outro prioritário em função de sua importância no sistema. Esta importância ou prioridade de execução é controlada pela própria aplicação e não pelo sistema operacional, como nos sistemas de tempo compartilhado.

Esses sistemas, normalmente, estão presentes em controle de processos, como no monitoramento de refinarias de petróleo, controle de tráfego aéreo, de usinas termelétricas e nucleares, ou em qualquer aplicação onde o tempo de resposta é fator fundamental.

Curso de Técnico em Informática – ETE Taquarituba

Sistemas com Múltiplos Processadores

Os sistemas com múltiplos processadores caracterizam-se por possuir duas ou mais

UCPS interligadas, trabalhando em conjunto. Um fator-chave no desenvolvimento de sistemas operacionais com múltiplos processadores é a forma de comunicação entre as UCPs e o grau de compartilhamento da memória e dos dispositivos de entrada e saída. Em função desses fatores, podemos classificar os sistemas em fortemente acoplados ou fracamente acoplados.

Sistemas com Múltiplos Processadores

Sistemas Fracamente

Acoplados

Sistemas Fortemente Acoplados

Sistemas

Operacionais Distribuídos

Sistemas

Operacionais de Rede

Sistemas Simétricos

Sistemas Assimétricos

Figura 1 - Sistemas com múltiplos processadores.

Sistemas Fortemente Acoplados

Nos sistemas fortemente acoplados (tightly coupled) existem vários processadores compartilhando uma única memória e gerenciados por apenas um sistema operacional. Múltiplos processadores permitem que vários programas sejam executados ao mesmo tempo, ou que um programa seja dividido em subprogramas, para execução simultânea em mais de um processador. Dessa forma, é possível ampliar a capacidade de computação de um sistema, adicionando-se apenas novos processadores, com um custo muito inferior à aquisição de outros computadores.

Com o multiprocessamento, novos problemas de concorrência foram introduzidos, pois vários processadores podem estar acessando as mesmas áreas de memória. Além disso, existe o problema de organizar de forma eficiente os processadores, a memória e os periféricos.

Uma conseqüência do multiprocessamento foi o surgimento dos computadores voltados, principalmente, para processamento científico, aplicado, por exemplo, ao desenvolvimento aeroespacial, prospeção de petróleo, simulações, processamento de imagens e CAD. A princípio qualquer aplicação que faça uso intensivo da UCP será beneficiada pelo acréscimo de processadores ao sistema.

Curso de Técnico em Informática – ETE Taquarituba

UCP Memória

Dispositivos de E/S

Dispositivos de E/S

Figura 12 - Sistemas fortemente acoplados

Memória Dispositivos de E/S

Memória Dispositivos de E/S

Link de Comunicação

Figura 13 - Sistemas fracamente acoplados

Sistemas Assimétricos

Na organização assimétrica ou mestre/escravo(master/slave), somente um processador (mestre) pode executar serviços do sistema operacional, como, por exemplo, realizar operações de entrada/saída. Sempre que um processador do tipo escravo precisar realizar uma operação de entrada/saída, terá de requisitar o serviço ao processador mestre. Dependendo do volume de operações de entrada/saída destinadas aos processadores escravos, o sistema pode se tornar ineficiente, devido ao elevado número de interrupções que deverão ser tratadas pelo mestre.

UCP SlaveUCP Master

Dispositivos de E/S S.O Usuários Usuários

Figura 14 - Sistemas assimétricos.

Curso de Técnico em Informática – ETE Taquarituba

Se o processador falhar, todo o sistema ficará incapaz de continuar o processamento.

Neste caso, o sistema deve ser reconfigurado, fazendo um dos processadores escravos assumir o papel do mestre.

Mesmo sendo uma organização simples de implementar e quase um extensão dos sistemas multiprogramáveis, esse tipo de sistema não utiliza eficientemente o hardware, devido à assimetria dos processadores, que não realizam as mesmas funções.

Sistemas Simétricos

O multiprocessamento simétrico (Simmetric Multiprocessing- SMP), ao contrário da organização mestre/escravo, implementa a simetria dos processadores, ou seja, todos os processadores realizam as mesmas funções. Apenas algumas poucas funções ficam a cargo de um único processador, como, por exemplo, a inicialiazação (boot) do sistema.

Dispositivos de E/S.OUsuários

Figura 15 - Sistemas simétricos.

Como vários processadores estão utilizando, independentemente, a mesma memória e o mesmo sistema operacional, é natural a ocorrência de acessos simultâneos às mesmas áreas de memória. A solução desses conflitos fica a cargo do hardware e do sistema operacional.

No processamento simétrico, um programa pode ser executado por qualquer processador, inclusive por vários processadores ao mesmo tempo (paralelismo). Além disso, quando um processador falha, o sistema continua em funcionamento sem nenhuma interferência manual, porém com menor capacidade de computação.

Os sistemas simétricos são mais poderosos que os assimétricos, permitindo um melhor balanceamento do processamento e das operações de entrada/saída, apesar de sua implementação ser bastante complexa.

Multiprocessamento

Desde sua criação, os computadores têm sido vistos como máquinas seqüências, onde a UCP executa a instruções de um programa, uma de cada vez. Na realidade, essa visão não é totalmente verdadeira, pois, em nível de hardware, múltiplos sinais estão ativos simultaneamente, o que pode ser entendido como uma forma de paralelismo.

Com a implementação de sistemas com múltiplos processadores, o conceito de simultaneidade ou paralelismo pode ser expandido a um nível mais amplo, denominado multiprocessamento, onde uma tarefa pode ser dividida e executada, ao mesmo tempo, por mais de um processador.

Organização Funcional

O esquema de comunicação interna das UCPs, memória e dipositivos de E/S (unidades funcionais) é fundamental no projeto de sistemas com múltiplos processadores, pois termina quantas UCPs o sistema poderá ter e como será o acesso à memória.

Para permitir múltiplos acessos simultâneos à memória (interliving), é comum que esta dividida em módulos, podendo assim ser compartilhada por várias unidades funcionais. As

Curso de Técnico em Informática – ETE Taquarituba organizações funcionais de multiprocessadores podem ser divididas basicamente em três tipos: barramento comum, barramento cruzado e memória multiport.

Sistemas Fracamente Acoplados

Os sistemas fracamente acoplados caracterizam-se por possuir dois ou mais sistemas de computação interligados, sendo que cada sistema possui o seu próprio sistema operacional, gerenciando os seus recursos, como processador, memória e dispositivos de entrada/saída.

Até meados da década de 80, os sistemas operacionais e as aplicações suportadas por eles eram tipicamente concentradas em sistemas de grande porte, com um ou mais processadores. Nos sistemas centralizados, os usuários utilizam terminais não inteligentes conectados a linhas seriais dedicadas ou linhas telefônicas públicas para a comunicação interativa com esses sistemas.

No modelo centralizado, os terminais não têm capacidade de processamento. Sempre um usuário deseja alguma tarefa, o pedido é encaminhado ao sistema, que realiza o processamento e retorna uma resposta, utilizando as linhas de comunicação.

Com a evolução dos computadores pessoais e das estações de trabalho, juntamente com o avanço das telecomunicações e da tecnologia de redes, surgiu um novo modelo de computação, chamado de modelo de rede de computadores.

Rede

Nó Nó Figura 16- Sistemas fracamente acoplados

Sistemas Operacionais de Rede

Em sistemas operacionais de rede (SOR), cada nó possui seu próprio sistema operacional, além de um hardware e software que possibilitam ao sistema ter acesso a outros componentes da rede, compartilhando seus recursos. O SOR permite entre outras funções: • Cópia remota de arquivos

• Emulação de terminal

• Impressão remota

• Gerência remota

• Correio eletrônico. Cada nó é totalmente independente do outro, podendo inclusive possuir sistemas operacionais diferentes. Caso a conexão entre os nós sofra qualquer problema, os sistemas podem continuar operando normalmente, apesar de alguns recursos se tornarem indisponíveis.

O melhor exemplo da utilização dos sistemas operacionais de rede são as redes locais.

Nesse ambiente, cada estação pode compartilhar seus recursos com o restante da rede. Caso uma estação sofra qualquer, os demais componentes da rede podem continuar o processamento, apenas não dispondo dos recursos oferecidos por ela.

Curso de Técnico em Informática – ETE Taquarituba

Figura 17 - Sistemas operacionais de rede.

Sistemas Operacionais Distribuídos

Em sistemas distribuídos, cada componente da rede também possui seu próprio sistema operacional, memória, processador e dispositivos. O que define um sistema distribuído é a existência de um relacionamento mais forte entre os seus componentes, onde geralmente os sistemas operacionais são os mesmos. Para o usuário e suas aplicações, é como se não existisse uma rede de computadores, mas sim um único sistema centralizado.

Rede Usuário

Figura 18 - Sistemas Operacionais Distribuídos.

A grande vantagem desses sistemas é a possibilidade do balanceamento de carga, ou seja, quando um programa é admitido para execução, a carga de processamento de cada sistema é avaliada e o processador mais livre é escolhido. Depois de aceito para processamento, o programa é executado no mesmo processador até o seu término. Também é possível o compartilhamento de impressoras, discos e fitas, independentemente do sistema em que a aplicação esteja sendo processada. Este tipo de sistema distribuído é muitas vezes chamado de cluster.

Curso de Técnico em Informática – ETE Taquarituba COMP 2COMP 1

Figura 19 - Cluster.

Suponha, por exemplo, uma configuração de dois computadores (COMP 1 e COMP 2), formando um cluster. Qualquer usuário conectado ao cluster poderá ter acesso aos dispositivos compartilhados, que permitem a ele imprimir uma listagem ou copiar um arquivo. Nesse tipo de configuração, se um dos sistemas falhar, o acesso aos dispositivos não será interrompido.

Os sistemas distribuídos podem ser considerados como uma evolução dos sistemas fortemente acoplados, onde uma aplicação pode ser executada por qualquer processador. Os sistemas distribuídos permitem que uma aplicação seja dividida em diferentes partes (aplicações distribuídas), que se comunicam através de linhas de comunicação, podendo cada parte ser processada em um sistema independente.

Organização Funcional

A organização funcional dos sistemas fracamente acoplados ou topologia define como são interligados fisicamente os diversos sistemas da rede.

Barramento

Na organização de barramento, os sistemas são conectados a uma única linha de comunicação e todos compartilham o mesmo meio, tanto para receber como para enviar mensagens. Esse tipo de organização é utilizada geralmente em redes locais (Figura 20).

Neste tipo de topologia, caso haja algum problema com o meio de transmissão, todos os nós da rede ficarão incomunicáveis.

Curso de Técnico em Informática – ETE Taquarituba

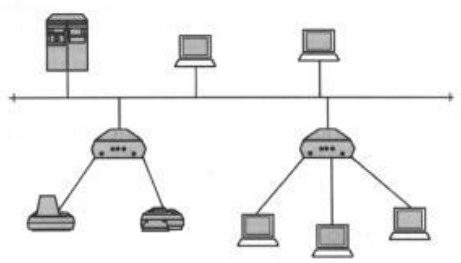


Figura 20 - Organização de Barramento

Organização Distribuída

Na organização distribuída existem linhas de comunicação ponto-a-ponto que ligam os sistemas e caminhos alternativos entre os diversos nós da rede. Caso uma linha de comunicação apresente problema, linhas alternativas permitirão que a rede continue em funcionamento. Este tipo de organização é utilizada geralmente em redes distríbuídas