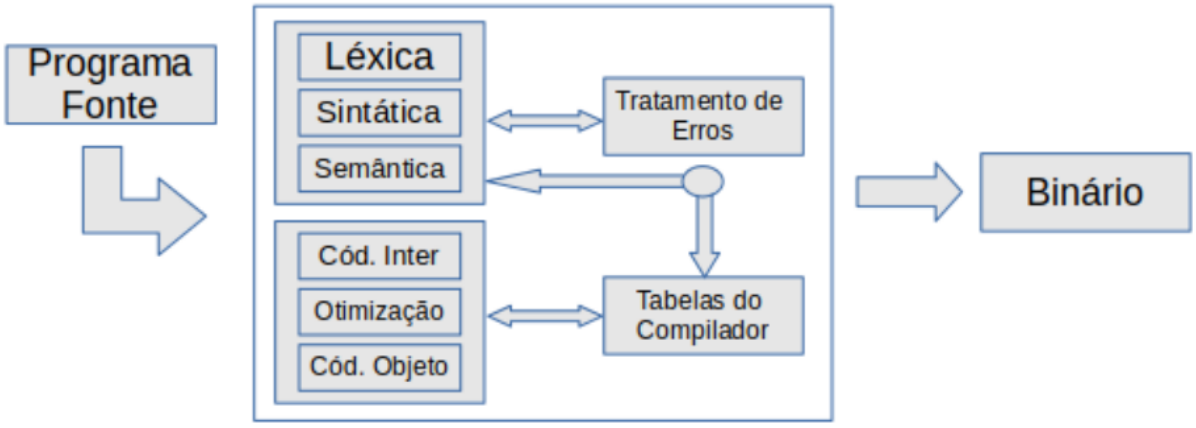
Conceitos de sistemas de computação

Compiladores



O processo de compilação de um programa consiste em transformar um programa fonte escrito em uma linguagem de programação em um programa executável, que é um arquivo binário que pode ser executado em uma máquina específica. Esse processo é realizado por um compilador, que é um programa responsável por essa transformação.

O programa fonte é o programa original escrito pelo programador em uma linguagem de programação. O compilador lê esse código fonte em uma sequência de caracteres e analisa cada palavra-chave, símbolo e operador presentes nele.

A análise léxica é a primeira etapa do processo de compilação, na qual o compilador identifica os tokens, ou símbolos, presentes no programa fonte. Isso é feito por meio de uma tabela de símbolos, que contém todas as palavras-chave e operadores da linguagem de programação.

A análise sintática é a próxima etapa, na qual o compilador verifica se a estrutura do programa está correta de acordo com as regras da linguagem de programação. Essa etapa utiliza uma gramática formal para verificar se o código fonte está de acordo com a sintaxe da linguagem.

A análise semântica é a terceira etapa, na qual o compilador verifica se o programa tem um significado válido. Isso inclui verificar se as variáveis e funções são definidas corretamente e se as operações realizadas são possíveis. Essa etapa usa uma tabela de símbolos e um conjunto de regras semânticas para verificar a validade do programa.

O tratamento de erro é uma etapa importante no processo de compilação. Durante as etapas anteriores, o compilador pode identificar erros no programa fonte. Esses erros são reportados ao programador, que deve corrigi-los antes de compilar o programa novamente.

Após a verificação semântica, o compilador gera um código intermediário, que é um código de baixo nível que representa o programa em uma forma mais abstrata do que o código fonte. Esse código intermediário é usado para facilitar a otimização do código e a geração do código objeto.

A otimização é uma etapa opcional do processo de compilação, na qual o código intermediário é analisado e transformado em um código mais eficiente, sem alterar o comportamento do programa. Essa etapa utiliza técnicas como a eliminação de código redundante, a substituição de operações por outras mais eficientes e a reordenação de instruções.

O código objeto é a última etapa do processo de compilação, na qual o compilador gera um código binário que pode ser executado na máquina alvo. Esse código contém todas as instruções do programa, bem como as informações necessárias para que o sistema operacional possa executá-lo corretamente.

As tabelas do compilador são estruturas de dados usadas pelo compilador para armazenar informações sobre o programa, como as variáveis definidas, as funções e os procedimentos, e as instruções do programa. Essas tabelas são usadas durante as etapas de análise léxica, sintática e semântica do processo de compilação.

Por fim, o binário é o resultado final do processo de compilação. Ele é o código executável que pode ser carregado na memória e executado pelo sistema operacional. O binário é geralmente compatível apenas com uma arquitetura específica de CPU, então pode ser necessário compilar o código fonte em diferentes plataformas para suportar múltiplas arquiteturas de hardware.

Em resumo, o processo de compilação é uma sequência de etapas que transforma o código fonte em um binário executável. Cada etapa do processo é importante para garantir que o código seja compilado com sucesso e gere um executável confiável e eficiente.

Linguagens compiladas

O slide trata dos conceitos relacionados às linguagens de programação compiladas, que incluem C, C++, C#, Objective-C, Fortran, Go, Rust, Delphi (Object Pascal), Pascal, entre outras. Vamos agora explicar detalhadamente cada um dos tópicos mencionados no slide:

Necessário "remontar" o programa sempre que necessitar realizar uma alteração

As linguagens de programação compiladas requerem que o código fonte seja traduzido para um código executável antes de ser executado. Esse processo é chamado de compilação e envolve várias etapas, como análise léxica, análise sintática, análise semântica e geração de código. Após a compilação, o código fonte é transformado em um arquivo executável, que pode ser executado diretamente pelo sistema operacional.

Uma das características das linguagens compiladas é que, se houver uma alteração no código fonte, será necessário recompilar o programa inteiro antes que a alteração entre em vigor. Isso ocorre porque cada alteração pode afetar o código em outras partes do

programa, o que significa que todo o código precisa ser "remontado" para garantir que tudo funcione corretamente.

Liga bibliotecas já compiladas (linker)

As linguagens de programação compiladas geralmente usam bibliotecas, que são coleções de código pré-compilado que podem ser usadas para realizar tarefas comuns. Por exemplo, uma biblioteca pode conter funções para realizar operações matemáticas, manipulação de arquivos ou comunicação de rede.

Quando um programa é compilado, o compilador cria referências para as funções que serão usadas a partir das bibliotecas, mas não as inclui diretamente no arquivo executável. Em vez disso, o linker é usado para ligar as referências a funções reais que estão contidas nas bibliotecas.

Essa abordagem tem várias vantagens. Em primeiro lugar, torna os programas mais eficientes em termos de espaço em disco, já que as bibliotecas podem ser compartilhadas entre vários programas. Além disso, permite que as bibliotecas sejam atualizadas independentemente dos programas que as usam, tornando a manutenção de software mais fácil.

Não é necessário um processo de análise e tradução toda vez que é executado

Uma vez que o código fonte de um programa compilado é transformado em um arquivo executável, não é necessário analisá-lo e traduzi-lo novamente toda vez que o programa é executado. Em vez disso, o sistema operacional carrega o arquivo executável diretamente na memória e o executa.

Essa abordagem tem várias vantagens. Em primeiro lugar, torna a execução de programas compilados mais rápida do que a execução de programas interpretados, já que não há necessidade de analisar e traduzir o código fonte toda vez que o programa é executado. Além disso, torna os programas mais seguros, já que o código fonte não é distribuído junto com o programa executável, o que dificulta a engenharia reversa e a descoberta de vulnerabilidades de segurança.

Assembler (Montador)

O Assembler, também conhecido como Montador, é um software que converte o código Assembly em código de máquina executável diretamente pelo processador. Assembly é uma linguagem de programação de baixo nível que utiliza uma notação simbólica para representar as instruções em linguagem de máquina.

O processo de conversão do código Assembly em código de máquina é feito em duas etapas: a montagem e a ligação.

Montagem:
Durante a etapa de montagem, o montador analisa o código Assembly linha por linha e converte cada instrução em código de máquina correspondente. O código de máquina é um

código binário que contém os comandos que o processador precisa para executar as tarefas especificadas pelo programador.
O montador também cria uma tabela de símbolos que contém informações sobre as variáveis e rótulos utilizados no código Assembly. Essa tabela é utilizada na etapa de ligação para relacionar as referências a variáveis e rotinas com seus endereços de memória correspondentes.

Ligação:
Na etapa de ligação, o linker ou ligador, é responsável por combinar o código de máquina gerado pelo montador com as bibliotecas externas necessárias para executar o programa. Isso é necessário porque muitos programas dependem de funções de bibliotecas externas para funcionar corretamente.
O linker também resolve as referências a variáveis e rotinas que foram definidas em outros arquivos de código-fonte. Para isso, ele utiliza a tabela de símbolos gerada pelo montador para encontrar os endereços de memória corretos das variáveis e rotinas.

Por fim, o linker gera um arquivo executável que contém todo o código de máquina necessário para executar o programa.

Em resumo, o processo do Assembler começa com a escrita do código Assembly, seguido pela etapa de montagem, onde o código Assembly é convertido em código de máquina. Na etapa de ligação, o código de máquina é combinado com as bibliotecas externas e as referências a variáveis e rotinas são resolvidas. O resultado final é um arquivo executável que pode ser executado diretamente pelo processador.

Tradutores

O processo de tradução de um programa de uma linguagem de alto nível para outra linguagem de alto nível geralmente envolve a utilização de um programa de tradução (ou compilador).

O compilador é um software que recebe o código-fonte de um programa escrito em uma linguagem de alto nível, como a linguagem A, e o traduz para uma linguagem de baixo nível, como a linguagem de máquina. Essa tradução é realizada em várias etapas.

A primeira etapa é a análise léxica, na qual o compilador examina o código-fonte para identificar os elementos básicos da linguagem, como palavras-chave, operadores, nomes de variáveis e constantes. Esses elementos são organizados em uma estrutura de dados chamada de tabela de símbolos.

A segunda etapa é a análise sintática, na qual o compilador usa a tabela de símbolos para analisar a estrutura do código-fonte e determinar se ele está de acordo com as regras da linguagem. Essa etapa envolve a criação de uma árvore de análise sintática, que representa a estrutura do programa.

A terceira etapa é a análise semântica, na qual o compilador verifica se o código-fonte faz sentido do ponto de vista da semântica da linguagem. Por exemplo, se o programa tenta fazer uma operação inválida em uma variável, o compilador detectará esse erro.

A quarta etapa é a geração de código, na qual o compilador traduz o código-fonte para a linguagem de destino, a linguagem B. Isso envolve a conversão da árvore de análise sintática em código de máquina ou em outra linguagem de alto nível.

Por fim, o código gerado pelo compilador é otimizado para melhorar seu desempenho e reduzir seu tamanho, se necessário.

Todo esse processo pode ser automatizado por um software de tradução ou compilador, que realiza as etapas de análise, tradução e otimização automaticamente. Dessa forma, os desenvolvedores podem escrever programas em uma linguagem de alto nível e, em seguida, traduzi-los para outras linguagens sem precisar entender os detalhes técnicos da tradução.

Linguagens Traduzidas

O tópico "Linguagens Traduzidas" se refere a um tipo de linguagem de programação que não é compilada diretamente para código de máquina, mas sim traduzida em tempo de execução por um interpretador ou tradutor.

As linguagens de programação que são traduzidas incluem Python, PHP, Ruby, Javascript, TypeScript, R, Lua, VBScript, ActionScript e outras. Em todas essas linguagens, um tradutor é invocado em toda execução do programa. Algumas bibliotecas podem estar pré-compiladas em algumas linguagens, como no caso do CPython.

Ao contrário das linguagens compiladas, que passam por um processo de compilação que gera código de máquina diretamente a partir do código fonte, as linguagens traduzidas passam por um processo geral de compilação, mas não geram código de máquina. Em vez disso, o código fonte é traduzido em tempo de execução em um código intermediário, que é interpretado e executado pelo sistema.

Algumas linguagens de programação usam um modo híbrido de tradução/compilação. Nesse caso, parte do código é compilada diretamente para código de máquina e outra parte é interpretada. Exemplos de linguagens de programação que usam esse modo híbrido incluem Java, Jython, Clang/Clang++, Julia e outras.

No caso do Java, por exemplo, o código fonte é compilado em bytecode, que é então interpretado pela máquina virtual Java (JVM). A compilação em bytecode permite uma execução mais rápida do que a interpretação direta do código fonte, mas ainda permite que o código seja executado em diferentes plataformas. Já o Jython é uma implementação da linguagem Python que é executada na JVM. O Jython compila o código Python em bytecode Java, que é então executado pela JVM. Esse processo permite que o código Python seja executado em qualquer plataforma que tenha uma JVM instalada.

O modo híbrido de tradução/compilação é útil porque permite que os desenvolvedores aproveitem os benefícios da compilação de código, mas ainda assim mantenham a portabilidade e flexibilidade de uma linguagem interpretada.

Linkers

O slide apresenta informações sobre linkers, que são programas responsáveis por juntar diferentes partes de um código compilado em um único executável ou biblioteca. Existem dois tipos principais de linkers: estático e dinâmico.

O linker estático cria um executável único que contém todo o código necessário para a execução do programa. Isso significa que o executável é auto-suficiente e não depende de nenhuma biblioteca externa para funcionar. O linker estático é mais fácil de gerenciar e instalar, e também permite melhores otimizações, pois todas as partes do código são conhecidas em tempo de compilação. No entanto, o linker estático não permite um sistema de plugin, pois todas as partes do código precisam estar presentes no executável.

Já o linker dinâmico trabalha de forma diferente. Ele compila cada parte do código separadamente em bibliotecas dinâmicas (DLL ou SO) que podem ser adicionadas à execução do programa em tempo de execução. As bibliotecas dinâmicas já estão otimizadas e podem ser substituídas independentemente (por exemplo, para aplicar patches). No entanto, o linker dinâmico é fortemente dependente das bibliotecas e permite indireções, o que pode tornar a depuração mais difícil.

Uma falácia comum é que somente o linker dinâmico carrega o código necessário para a execução do programa, enquanto o linker estático carrega todo o código, aumentando o overhead. No entanto, sistemas operacionais modernos implementam mecanismos de paginação de memória que permitem que somente as páginas relevantes sejam carregadas em memória, mesmo com o linker estático.

O processo de linkagem envolve a colocação do código e dos dados simbolicamente em memória, a determinação dos endereços dos rótulos de dados e instruções e a junção de referências internas e externas. A linkagem é importante para garantir que todas as partes do programa sejam corretamente integradas em um único executável ou biblioteca.

Loaders

O tópico "Loaders" se refere a um processo utilizado pelos sistemas operacionais para carregar e executar programas executáveis. Quando um programa é criado e compilado em um determinado ambiente de desenvolvimento, ele é gerado em formato binário, que é um conjunto de instruções e dados que o computador pode entender e executar diretamente. No entanto, esse binário ainda não pode ser executado diretamente pelo sistema operacional, pois precisa ser carregado em memória e configurado corretamente para que possa ser executado.

É nesse ponto que o Loader entra em ação. Ele é um executável "pronto" que é capaz de ler o cabeçalho do programa binário, que contém informações importantes como o tamanho de código e segmentos que o programa utiliza. Com base nessas informações, o Loader aloca a quantidade necessária de memória para o programa e copia as instruções e dados para a memória alocada.

Além disso, o Loader também é responsável por iniciar os registradores e o ponteiro de stack no primeiro local livre da memória e saltar para a rotina inicial do programa, carregando os argumentos necessários para que o programa comece a ser executado. Quando o programa finaliza sua execução, ele invoca uma system call exit, indicando ao sistema operacional que sua execução foi concluída e que a memória alocada pode ser liberada.

Em resumo, os Loaders são um componente importante dos sistemas operacionais, pois permitem que programas executáveis sejam carregados e executados corretamente em memória, garantindo que a execução do programa ocorra de forma segura e eficiente.

Máquinas Virtuais (VMs)

O tópico "Máquinas Virtuais (VMs)" se refere a um conceito de computação que envolve a criação de um ambiente de execução isolado e seguro, que é chamado de "virtual". Esse ambiente virtual é criado por meio da duplicação dos recursos de hardware da máquina hospedeira, permitindo que múltiplos sistemas operacionais e aplicativos sejam executados simultaneamente em um único hardware.

Existem diversas implementações de máquinas virtuais, cada uma projetada para uma finalidade específica. Entre as mais conhecidas estão a JVM (Java Virtual Machine) e a LLVM (Low-Level Virtual Machine). A JVM é projetada para executar programas Java, mas também suporta outras linguagens de programação, como Quercus (PHP), jRuby, Nashorn, Clojure, Scala, Groovy e Jython. A LLVM, por sua vez, é projetada para compilar e executar código de baixo nível em diversas linguagens de programação, como Julia, Clang, Lua, CUDA, OpenCL, Objective-C, Swift, C# e outras.

Uma das principais vantagens das máquinas virtuais é a facilidade de migração e portabilidade de linguagens de programação entre diferentes sistemas operacionais e plataformas. Isso significa que um aplicativo que foi desenvolvido em uma determinada linguagem de programação pode ser executado em diferentes sistemas operacionais sem a necessidade de modificar o código fonte. Além disso, as máquinas virtuais também oferecem maior segurança e isolamento de aplicativos, uma vez que cada ambiente virtual é separado e independente.

É importante destacar que a execução em máquinas virtuais não é necessariamente híbrida. Híbrido é um termo utilizado para descrever um tipo de aplicativo que combina diferentes tecnologias, como código nativo e interpretado, em um único ambiente. Embora algumas máquinas virtuais possam suportar a execução de aplicativos híbridos, a execução em ambientes virtuais geralmente é projetada para executar código compilado ou interpretado de forma isolada e independente.

Depuradores

O tópico "Depuradores" se refere a ferramentas de software que permitem aos programadores encontrar e corrigir problemas em seus códigos. O depurador é uma ferramenta importante para desenvolvedores, pois permite rastrear a execução do programa em tempo real, interromper a execução em um determinado ponto e examinar os valores dos dados e variáveis armazenados em memória.

O depurador pode ser utilizado para encontrar erros lógicos, falhas de programação ou erros de sintaxe. Ele pode ser usado para visualizar o fluxo de execução do programa, examinar o conteúdo da memória, definir pontos de interrupção em partes específicas do código e examinar os valores das variáveis. Os depuradores também podem ser utilizados para examinar o código de outros programas e bibliotecas que foram incluídos no programa sendo depurado.

Os depuradores geralmente são projetados para interagir com a máquina em que o programa está sendo executado. Isso significa que, em alguns casos, eles podem interromper a execução de máquina para permitir que o programador examine endereços de registradores e memória, a fim de entender o comportamento do programa. Em máquinas virtuais, como mencionado anteriormente, os depuradores são ainda mais flexíveis, permitindo que o programador examine a execução do programa dentro do ambiente virtual.

Apesar da importância dos depuradores, eles também podem ser um problema para os programadores. A incerteza de Heisenberg, também conhecida como Heisenbug, é um exemplo de problema que pode ocorrer durante a depuração de um programa. Isso ocorre quando a tentativa de depurar um problema altera o comportamento do programa de tal forma que o problema não pode mais ser reproduzido ou identificado.

Exemplos de depuradores incluem o GDB (GNU Debugger) para sistemas Unix-like, o PDB (Python Debugger) para programas Python, o JSwat para programas Java, o Eclipse Debugger para programas em geral e o Valgrind para detecção de vazamento de memória e erros de acesso em baixo nível.

Just-in-Time (JIT)

O tópico "Just-in-Time (JIT)" se refere a uma técnica de otimização de código usada por algumas máquinas virtuais (VMs) e linguagens de programação (LPs). Ela combina os benefícios da compilação e interpretação de código, permitindo uma execução mais rápida e eficiente.

A técnica JIT é usada em VMs que não precisam compilar o código fonte em bytecode repetidamente. Em vez disso, o JIT permite que o bytecode seja compilado sob demanda, ou seja, somente quando é necessário para a execução do programa. Isso é

particularmente útil em casos em que o mesmo trecho de código é executado várias vezes, pois permite que o código seja compilado apenas uma vez e, em seguida, reutilizado.

A execução JIT geralmente é mais rápida do que a execução de LPs puramente interpretadas, pois o código compilado é executado diretamente pelo processador, em vez de ser interpretado pela VM. No entanto, pode haver atrasos com a compilação de bytecode para código de máquina, que pode afetar o desempenho inicial do programa.

Algumas das LPs mais comuns que fornecem suporte a JIT incluem a JVM (a partir da versão 4.0), RPython, .NET, LLVM e outras. Essas LPs usam o JIT para otimizar o desempenho do programa, compilando código sob demanda e permitindo a execução direta do código de máquina. A técnica JIT é uma das principais razões pelas quais as VMs são capazes de fornecer desempenho comparável ao das linguagens compiladas.