16. Compilação no Linux

*16.1 C*ompilador X Interpretador

- Um código fonte pode ser compilado ou interpretado.
- Compiladores e interpretadores tratam o código de maneira diferente.

- Interpretador:
  - Lê o código fonte, uma linha por vez, executando a instrução dessa linha.
  - Deve estar presente toda vez que se quiser executar o programa. Ex.: Basic
  - Como a cada execução, o código é examinado linha por linha, o processo é mais lento.

- Compilador:
  - Lê o programa inteiro e o converte para código objeto.
  - Código objeto é um código binário, também conhecido como código de máquina. Fica armazenado em um arquivo e pode ser executado pelo processador.
  - O código compilado, após ser ligado, pode ser executado diretamente, sem necessitar de nova compilação e sem depender do compilador.

16.2 - Tarefas do compilador

Um compilador executa basicamente 3 tarefas:

- Pré-processamento
- Compilação: conversão de C para assembly
- Tradução: conversão de assembly para código de máquina (binário)
- Ligação-edição; junção dos arquivos objeto e bibliotecas para formar o arquivo executável.

    - Pré-processamento e ligação-edição são tarefas executadas por dois programas, o pré-processador e o ligador respectivamente
    - Em geral os compiladores mais novos contêm essas duas ferramentas acopladas.

1.6.21 - Pré-processamento

O pré-processamento é feito antes da compilação e consiste em:

- Retirada de comentários
- Inclusão de arquivos de cabeçalhos (headers)
- Inclusão de macros no meio do código.
- Algumas otimizações:
    - Funções pequenas do mesmo arquivo são inseridas no meio do código, pois a chamada de funções é um processo caro
    - Valores constantes calculados dentro de um loop são movidos para fora.

16.2.2 - Compilação

- As instruções são convertidas para código de máquina (código objeto ; extensão .o)
- É feita a verificação da sintaxe do código, erros de sintaxe são detectados nessa fase.
- Ligações com outras funções e variáveis externas não é feita.
- As rotinas (bibliotecas) do sistema necessárias para executar o programa ainda não são incluídas.

- Obs.: Bibliotecas do sistema:
  - Fornecem suporte em tempo de execução.
  - Permitem acesso a serviços do SO, ex.: I/O
  - Permitem acesso às rotinas de ponto flutuante.

- Durante a compilação de um programa, ocorrem os eventos de tempo de compilação. Exemplos: Avisar que uma variável não foi declarada.

- Código relocável:
  - Pode ser executado em qualquer região da memória disponível.
  - Contém informações de quantos bytes, a partir do inicio do arquivo, vai precisar para executar suas instruções.
  - A produção de código relocável facilita o processo de ligação/edição.

16.2.3 - Ligação/edição

- Feita pelo ligador ou ainda *linker* ou *loader*.
- O ligador é responsável por ligar (*linkar*) as várias partes do código objeto:
  - códigos objeto gerados pelo programador
  - bibliotecas padrão da linguagem
  - rotinas (bibliotecas) do sistema.

- Após a ligação/edição é gerado o código executável.
- Durante a ligação/edição são atribuídos os endereços das instruções de "jump" (desvio) e "call" (chamadas), ou seja, resolvem-se as referências externas.

Exemplo:

arquivo1.c

```
int count;   /*declaraçoes*/
void display(void);
   /*programa principal*/
void main(void)
{
  count = 10;
  display();
}
```

arquivo2.c

```
#include <stdio.h>
extern int count;   /* decl */
   /*função display*/
void display(void)
{
  printf("%d \n", count);
}
```

- Na compilação são criados dois códigos objetos: arquivo1.o e arquivo2.o.

- A localização da função *display()* no arquivo 1 fica em aberto após a compilação. Idem para a variável externa *count* no arquivo

- Essas localizações (endereços) serão resolvidas durante a fase de ligação.

- O ligador "avisa" se algum símbolo está indefinido. Exemplos:
  - Uso de uma função que não foi definida.
  - Associação errada de bibliotecas.

- Os arquivos de biblioteca padrão (*.a) contêm um conjunto de códigos objetos (.o).
- Arquivos objetos são incluídos totalmente em um código executável.
- Arquivos de biblioteca são parcialmente incluídos. Somente as partes (*.o) que são utilizadas pelo programa são incluídas.

16.3 - Compilação separada

- Um programa pode estar contido em muitos arquivos.

- Se apenas uma parte do código é alterada, somente o arquivo correspondente necessita ser recompilado.

- A atualização e manutenção do programa é bem mais fácil e rápida.

- Porém existe o trabalho inicial de organizar o programa em diversos arquivos.

16.4 - Execução e mapa de memória

- Durante a execução de um programa, ocorrem os eventos de tempo de execução.

  Ex.: erro de divisão por zero, causando a interrupção do programa.

- Um compilador também oferece rotinas para gerenciar o ambiente durante o tempo de execução.

- Durante a execução são criadas e utilizadas 4 regiões distintas na memória com finalidades específicas:
  - Região de armazenamento de código.
  - Região de armazenamento das variáveis globais.
  - Pilha:

    Armazena informações relacionadas com funções, por exemplo:
    - endereço de retorno das chamadas às funções
    - argumentos de funções
    - variáveis locais
  - Heap:
    - Região de alocação dinâmica.
    - Armazena dados criados em tempo de execução, ex.: listas encadeadas, árvores.

- A disposição física dessas 4 regiões varia em função do tipo da CPU e do compilador.

- Colisão entre pilha e heap:
  - Funções recursivas são aquelas que chamam a si próprias.

- A cada chamada, um novo conjunto de variáveis locais é criado e armazenado na pilha.

- Um grande número de chamadas recursivas podem "estourar" a pilha e causar o problema de colisão entre pilha e heap.

- Isto geralmente ocorre devido à erros de elaboração do código fonte.

16.5 - Ambientes integrados de programação

- O processo de programar consiste basicamente em:
  - Criar o programa fonte através de um editor.
  - Compilar (pre-processamento, compilação e ligação/edição).
  - Fazer depuração através de um *debbuger.*
  - Executar o programa.

- Alguns compiladores, fornecem um ambiente integrado de programação contendo:
  - editor de texto
  - make: programa para organizar a compilação separada de arquivos dependentes
  - compilador
  - debugger: permite acompanhar a execução do programa passo a passo, verificando o valor de variáveis, endereços e expressões.

- Outros compiladores, por exemplo: *gcc* do Unix, executam somente a tarefa principal de compilar

- No entanto existem alguns utilitários destinados a executar as outras tarefas. Alguns desses utilitários são:
  - *vi* e *emacs:* editores de texto.
  - *make*: funciona de modo semelhante ao make dos ambientes integrados
  - xxgdb, dbxtool: auxiliam a depuração.

- Programas muito extensos devem ser colocados em diferentes arquivos.

- Vantagens:
  - É mais fácil trabalhar com arquivos pequenos.
  - Permite reutilização.
  - Se um arquivo com código fonte é modificado, somente este é recompilado.

- Os programadores do Unix utilizam uma maneira padrão para particionar o código.