O nível de linguagem de montagem

Introdução à linguagem de montagem

- Uma linguagem de montagem pura é uma linguagem na qual cada declaração produz exatamente uma instrução de máquina.

- A utilização de nomes simbólicos e endereços simbólicos faz uma enorme diferença.

- O programador de linguagem de montagem tem acesso a todos os recursos e instruções disponíveis na máquina-alvo; o programador de linguagem de alto nível não tem.

- Um programa em linguagem de montagem só pode ser executado em uma família de máquinas.

Introdução à linguagem de montagem

- A figura abaixo mostra fragmentos de programas em linguagem de montagem para o x86 para efetuar o cálculo *N = I + J*.

Etiqueta	*Opcode*	Operandos	Comentários
FORMULA:	MOV	EAX,I	; registrador EAX = I
	ADD	EAX,J	; registrador EAX = I + J
	MOV	N,EAX	; N = I + J
I	DD	3	; reserve 4 bytes com valor inicial 3
J	DD	4	; reserve 4 bytes com valor inicial 4
N	DD	0	; reserve 4 bytes com valor inicial 0

Introdução à linguagem de montagem

- Algumas das pseudoinstruções disponíveis no *assembler* MASM.

Pseudoinstrução	Significado
SEGMENT	Inicie um novo segmento (texto, dados etc.) com certos atributos
ENDS	Encerre o segmento corrente
ALIGN	Controle o alinhamento da próxima instrução ou dados
EQU	Defina um novo símbolo igual a uma expressão dada
DB	Aloque armazenamento para um ou mais bytes (inicializados)
DW	Aloque armazenamento para um ou mais itens de dados (palavras) de 16 bits (inicializados)
DD	Aloque armazenamento para um ou mais itens de dados (duplos) de 32 bits (inicializados)
DQ	Aloque armazenamento para um ou mais itens de dados (quádruplos) de 64 bits (inicializados)
PROC	Inicie um procedimento
ENDP	Encerre um procedimento
MACRO	Inicie uma definição de macro
ENDM	Encerre uma definição de macro
PUBLIC	Exporte um nome definido neste módulo
EXTERN	Importe um nome definido de outro módulo
INCLUDE	Busque e inclua um outro arquivo
IF	Inicie a montagem condicional baseada em uma expressão dada
ELSE	Inicie a montagem condicional se a condição IF acima for falsa
ENDIF	Termine a montagem condicional
COMMENT	Defina um novo caractere de início de comentário
PAGE	Gere uma quebra de página na listagem
END	Termine o programa de montagem

Macros

- Uma definição de macro é um modo de dar um nome a um pedaço de texto.

- Embora *assemblers* diferentes tenham notações diferentes para definir macros, todos requerem as mesmas partes básicas em uma definição de macro:

1. Um cabeçalho de macro que dê o nome da macro que está sendo definida.

2. O texto que abrange o corpo da macro.

3. Uma pseudoinstrução que marca o final da definição (por exemplo, ENDM).

Macros

- Chamadas de macro não devem ser confundidas com chamadas de procedimento.

Item	Chamada de macro	Chamada de procedimento
Quando a chamada é feita?	Durante montagem	Durante execução do programa
O corpo é inserido no programa-objeto em todos os lugares em que a chamada é feita?	Sim	Não
Uma instrução de chamada de procedimento é inserida no programa-objeto e executada mais tarde?	Não	Sim
Deve ser usada uma instrução de retorno após a conclusão da chamada?	Não	Sim
Quantas cópias do corpo aparecem no programa-objeto?	Uma por chamada de macro	Uma

Macros

- Para implementar um processador de macros, um *assembler* deve ser capaz de realizar duas funções:

  - salvar definições de macro e
  - expandir chamadas de macro.

- **O processo de montagem**

- A principal função da passagem um é montar uma tabela denominada **tabela de símbolos**, que contém o valor de todos os símbolos.

O processo de montagem

- Um símbolo é um rótulo ou um valor ao qual é atribuído um nome simbólico por meio de uma pseudoinstrução, como
- BUFSIZE EQU 8192
- A passagem um da maioria dos assemblers usa no mínimo três tabelas internas:

1. a tabela de símbolos,
2. a de pseudoinstruções e
3. a de *opcodes*.

O processo de montagem

- A tabela de símbolos tem uma entrada para cada símbolo, como ilustrado abaixo:

Símbolo	Valor	Outras informações
MARIA	100	
ROBERTA	111	
MARILYN	125	
STEPHANY	129	

O processo de montagem

- A tabela de *opcodes* contém pelo menos uma entrada para cada *opcode* (mnemônico) simbólico na linguagem de montagem.

Opcode	Primeiro operando	Segundo operando	*Opcode* hexa	Comprimento da instrução	Classe da instrução
AAA	—	—	37	1	6
ADD	EAX	immed32	05	5	4
ADD	reg	reg	01	2	19
AND	EAX	immed32	25	5	4
AND	reg	reg	21	2	19

O processo de montagem

```
public static void pass_one( ) {
    // Esse procedimento é um esboço de passagem um para um assembler simples
    boolean more_input = true;                  // sinal que para a passagem um
    String line, symbol, literal, opcode;       // campos da instrução
    int location_counter, length, value, type;  // variáveis diversas
    final int END_STATEMENT = –2;               // sinaliza final da entrada

    location_counter = 0;                       // monta a primeira instrução em 0
    initialize_tables( );                       // inicialização geral

    while (more_input) {                        // more_input ajustada para falso por END
        line = read_next_line( );               // obtenha uma linha de entrada
        length = 0;                             // # bytes na instrução
        type = 0;                               // de que tipo (formato) é a instrução

        if (line_is_not_comment(line)) {
            symbol = check_for_symbol(line);    // essa linha é rotulada?
            if (symbol != null)                 // se for, registre símbolo e valor
              enter_new_symbol(symbol, location_counter);
            literal = check_for_literal(line);  // a linha contém uma literal?
            if (literal != null)                // se contiver, entre a linha na tabela
              enter_new_literal(literal);

            // Agora determine o tipo de opcode. –1 significa opcode ilegal.
            opcode = extract_opcode(line);      // localize mnemônico do opcode
            type = search_opcode_table(opcode); // ache formato, por exemplo OP REG1,REG2
            if (type < 0)                       // se não for um opcode, é uma pseudoinstrução?
              type = search_pseudo_table(opcode);
            switch(type) {                      // determine o comprimento dessa instrução
              case 1: length = get_length_of_type1(line);  break;
              case 2: length = get_length_of_type2(line);  break;
              // outros casos aqui
            }
        }
        write_temp_file(type, opcode, length, line);   // informação útil para a passagem dois
        location_counter = location_counter + length;  // atualize loc_ctr
        if (type == END_STATEMENT) {            // terminamos a entrada?
            more_input = false;                 // se terminamos, execute tarefas de preparo
            rewind_temp_for_pass_two( );        // tais como rebobinar o arquivo temporário
            sort_literal_table( );              // e ordenar a tabela de literais
            remove_redundant_literals( );       // e remover literais duplicadas
        }
    }
}
```

- Passagem um de um assembler simples.

O processo de montagem

```
public static void pass_two( ) {
    // Esse procedimento é um esboço de passagem dois para um assembler simples.
    boolean more_input = true;                  // sinal que para a passagem dois
    String line, opcode;                        // campos da instrução
    int location_counter, length, type;         // variáveis diversas
    final int END_STATEMENT = –2;               // sinaliza final da entrada
    final int MAX_CODE = 16;                            // máximo de bytes de código por instrução
    byte code[ ] = new byte[MAX_CODE];          //contém código gerado por instrução

    location_counter = 0;                       // monta a primeira instrução em 0

    while (more_input) {                        // more_input ajustada para falso por END
      type = read_type( );                      // obtém campo de tipo da próxima linha
      opcode = read_opcode( );                  // obtém campo de opcode da próxima linha
      length = read_length( );                  // obtém comprimento de campo da próxima linha
      line = read_line( );                      // obtém a linha de entrada propriamente dita

      if (type != 0) {                          // tipo 0 é para linhas de comentário
        switch(type) {                          // gerar o código de saída
          case 1: eval_type1(opcode, length, line, code);  break;
          case 2: eval_type2(opcode, length, line, code);  break;
          // outros casos aqui
        }
      }
      write_output(code);                       // escreva o código binário
      write_listing(code, line);                // imprima uma linha na listagem
      location_counter = location_counter + length;          // atualize loc_ctr
      if (type == END_STATEMENT) {              // terminamos a entrada?
        more_input = false;                     // se terminamos, execute tarefas de manutenção
        finish_up( );                           // execute tarefas de manutenção gerais e termine
      }
    }
}
```

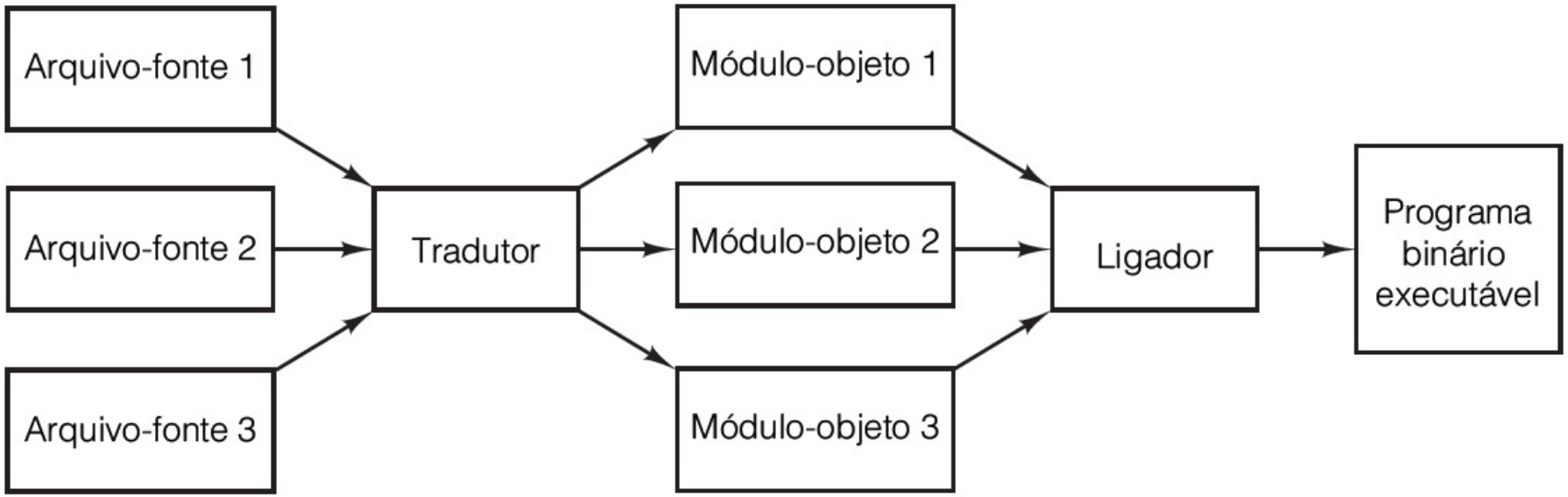
- A função da passagem dois é gerar o programa-objeto e talvez imprimir a listagem de montagem.

O processo de montagem

- Durante a passagem um do processo de montagem, o *assembler* acumula informações sobre símbolos e seus valores.
- A técnica de implementação mais simples é, de fato, executar a tabela de símbolos como um arranjo de pares.
- Outro modo de organizar a tabela de símbolos é ordenar por símbolos e usar o algoritmo de busca binária para procurar um símbolo.
- Um modo completamente diferente de simular uma memória associativa é uma técnica conhecida como codificação *hash* ou *hashing*.

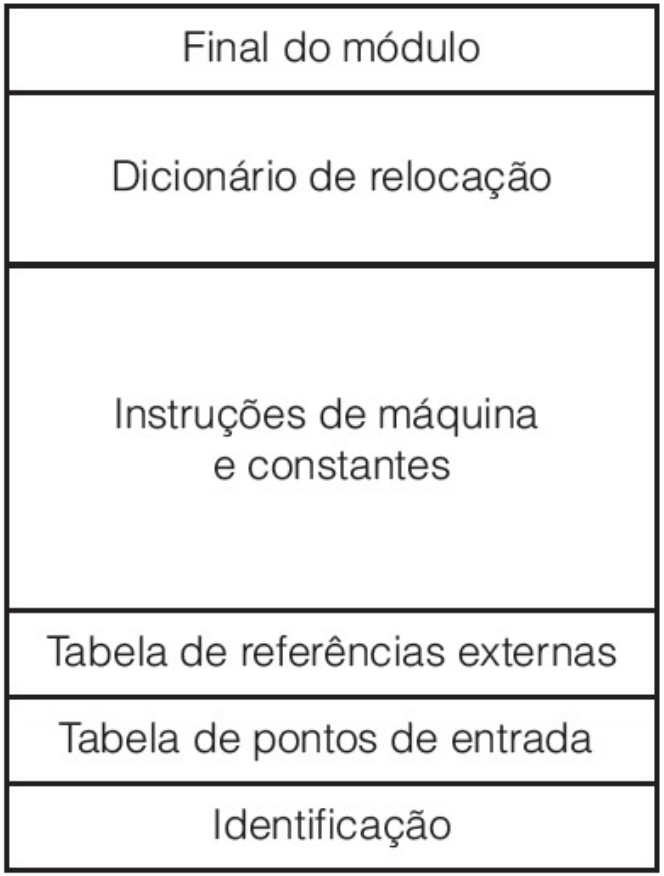
Ligação e carregamento

- A geração de um programa binário executável a partir de um conjunto de procedimentos-fonte traduzidos independentemente requer a utilização de um ligador.



Ligação e carregamento

- Estrutura interna de um módulo-objeto produzido por um tradutor.



Ligação e carregamento

- Quando um programa é escrito, ele contém nomes simbólicos para endereços de memória, por exemplo, BR L.

- O momento em que é determinado o endereço da memória principal correspondente a L é denominado tempo de vinculação.

- Há pelo menos seis possibilidades para o momento de vinculação:

1. Quando o programa é escrito.
2. Quando o programa é traduzido.

Ligação e carregamento

3. Quando o programa é ligado, mas antes de ser carregado.
4. Quando o programa é carregado.
5. Quando um registrador de base usado para endereçamento é carregado.
6. Quando a instrução que contém o endereço é executada.

- **Ligação dinâmica**

- Um modo mais flexível de ligar procedimentos compilados em separado é ligar cada procedimento no momento em que ele é chamado pela primeira vez. Esse processo é denominado **ligação dinâmica**.

Ligação dinâmica em MULTICS

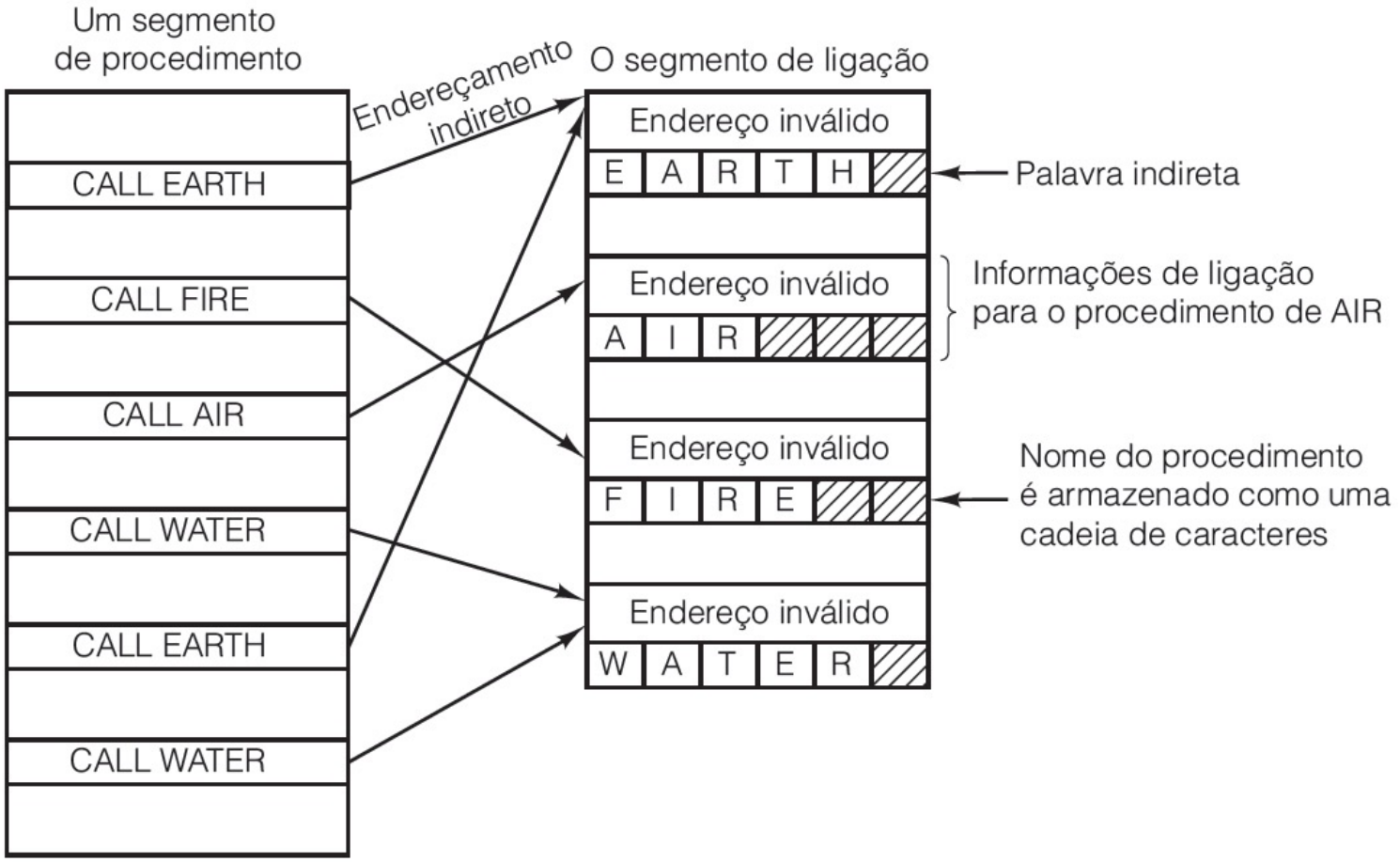
- Na forma MULTICS de ligação dinâmica, há um segmento associado a cada programa, denominado **segmento de ligação**, que contém um bloco de informações para cada procedimento que poderia ser chamado.

- Quando a ligação dinâmica está sendo usada, chamadas de procedimento na linguagem-fonte são traduzidas para instruções que endereçam indiretamente a primeira palavra do bloco de ligação correspondente, conforme mostra a figura a seguir.

- Depois, é atribuído um endereço virtual a esse procedimento, conforme figura seguinte.

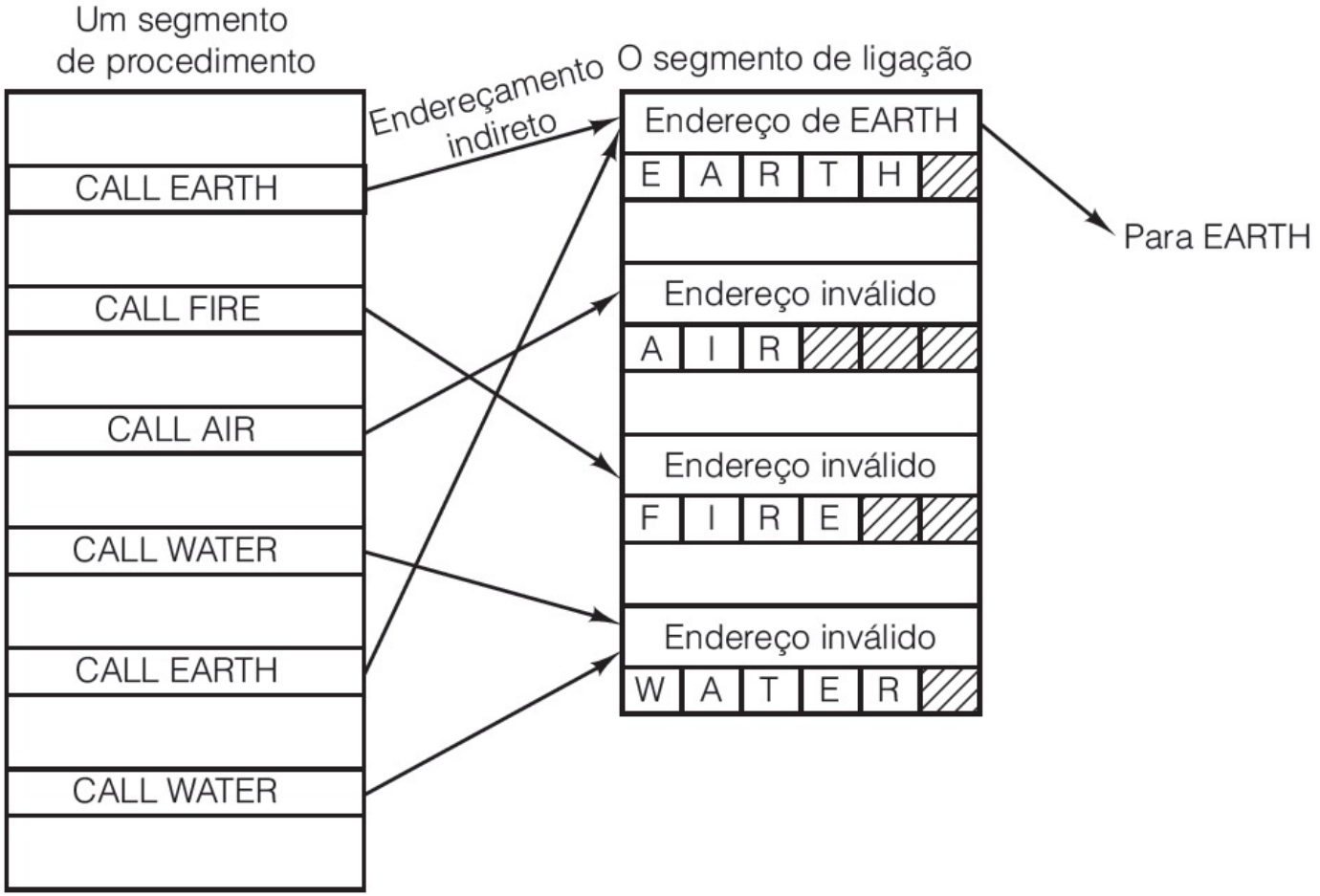
Ligação dinâmica em MULTICS

- Ligação dinâmica. Antes de EARTH ser chamado.



Ligação dinâmica em MULTICS

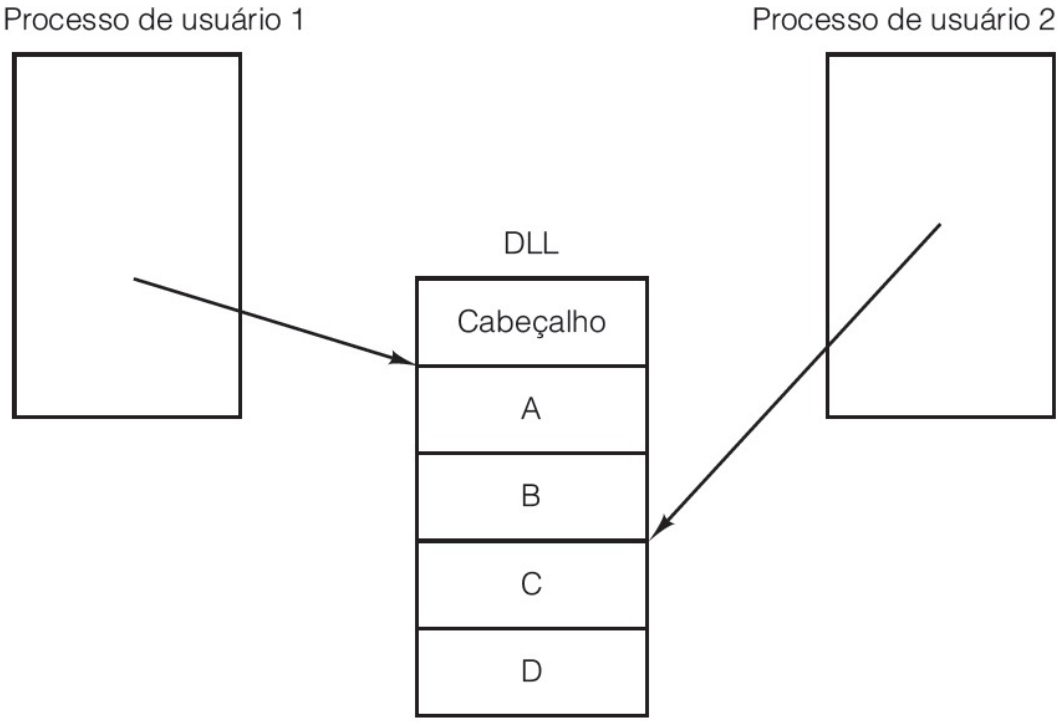
- Ligação dinâmica. Após EARTH ser chamado e ligado.



Ligação dinâmica no Windows

- A ligação dinâmica usa um formato de arquivo especial denominado **DLL**.

- Utilização de um arquivo DLL por dois processos.



Ligação dinâmica no UNIX

- O sistema UNIX tem um mecanismo que é, em essência, semelhante às DLLs no Windows, denominado **biblioteca compartilhada**.

- Como um arquivo DLL, uma biblioteca compartilhada é um arquivamento que contém vários procedimentos ou módulos de dados que estão presentes na memória durante o tempo de execução e que podem ser vinculados a vários processos ao mesmo tempo.

- O UNIX suporta apenas ligação implícita, portanto, uma biblioteca compartilhada consiste em duas partes: uma **biblioteca hospedeira** e uma **biblioteca-alvo**.