Software Básico

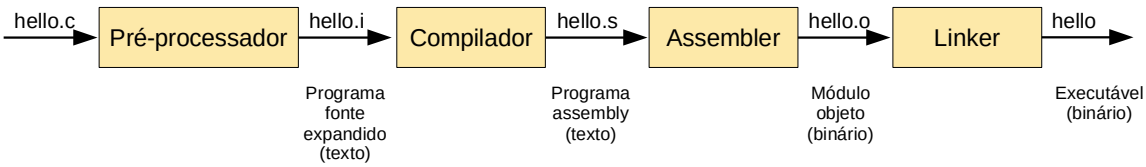
Compilação e Ligação

Reconhecimento

- Material produzido por:
  - Noemi Rodriguez – PUC-Rio
  - Ana Lúcia de Moura – PUC-Rio
- Adaptação
  - Bruno Silvestre – UFG

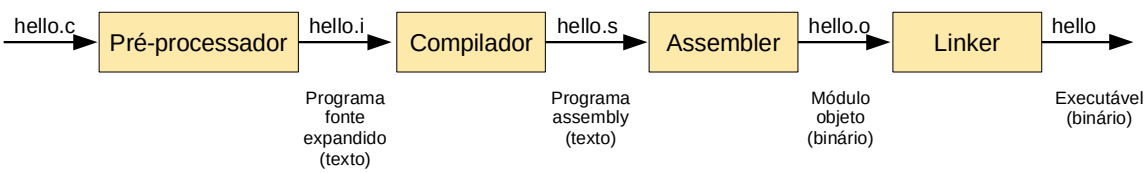
Compilação e Ligação

- A compilação de um módulo C compreende:
  - Pré-processamento
  - Geração de código Assembly
  - Geração do módulo objeto



Compilação e Ligação

- Um módulo objeto não pode ser executado:
  - Dependência de outros módulos (e bibliotecas)
  - As “lacunas” são preenchidas pela ligação (amarração)
- Endereços podem não estar (totalmente) definidos
  - As referências são corrigidas pela relocação



Ligador (*Linker*)

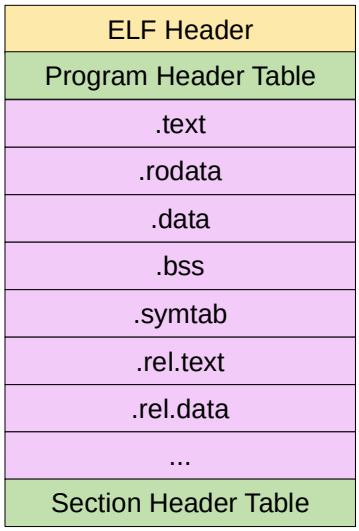
- Combina vários arquivos objeto em um arquivo executável
  - União dos arquivos (código e dados)
  - Resolução das referências externas
  - Relocação das referências à memória
- As bibliotecas estáticas são incluídas
  - Uma biblioteca reúne arquivos objeto relacionados
  - Ex: libc, libm, ...

Arquivo Objeto

- Seções de código, dados e algumas seções com informações para a “ligação” com outros arquivos
  - Tabela de Símbolos
    - Funções e variáveis definidas pelo módulo ("exportações")
    - Funções e variáveis referenciadas pelo módulo ("importações")
  - Dicionário de Relocação
    - Localização de referências a memória a serem preenchidas ou corrigidas (instruções e dados)
- Pode conter informações para um debugger

Arquivo Objeto

- Cabeçalho ELF possui informações sobre o arquivo
- Cabeçalho de programa possui informação de como carregar o programa na memória
  - Importante para o SO
- Cabeçalho de seção possui informações das posições de cada seção
  - Usado pelo *linker*



Linker: Realocação de Código

Linker: Realocação de Código

calc.c

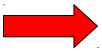
```
extern int var1;

void fun1(int x);

int var2 = 0;
int var3 = 0;

int fun2() {
  fun1(var1);
  …
}
```

Compilador



calc.s

```
.data
.globl var2
var2: .int 0

.globl var3
var3: .int 0

.text
.globl fun2
fun2:
   pushq %rbp
   movq  %rbp, %rsp
   movl var1, %edi

   call fun1
```

Linker: Realocação de Código

calc.s

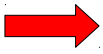
```
.data
.globl var2
var2: .int 0

.globl var3
var3: .int 0

.text
.globl fun2
fun2:
   pushq %rbp
   movq  %rbp, %rsp
   movl var1, %edi

   call fun1
```

Assembler



Arquivo Objeto (ELF)

```
Seção .data:
---------------- <var2>:
   0:   00 00
---------------- <var3>:
   4:   00 00

Seção .text:
---------------- <fun2>:
   0:   55                      push %rbp
   1:   48 89 ec                mov %rbp,%rsp
   4:   8b 3c 25 00 00 00 00    mov 0x0,%edi
   b:   e8 00 00 00 00          callq <fun2>
```

Espaço para endereço de func2

Espaço para endereço de var1

Linker: Realocação de Código

```
Seção .data:
---------------- <var2>:
   0:   00 00
---------------- <var3>:
   4:   00 00

Seção .text:
---------------- <fun2>:
   0:   55                      push %rbp
   1:   48 89 ec                mov %rbp,%rsp
   4:   8b 3c 25 00 00 00 00    mov 0x0,%edi
   b:   e8 00 00 00 00          callq <fun2>
```

Tabela de Símbolos

OFFSET	TIPO	SÍMBOLO
00	D	var2
04	D	var3
00	T	fun2
-	U	var1
-	U	fun1

Tabela de Realocação

OFFSET	TIPO	SÍMBOLO
07	R_X86_64_32S	var1
0c	R_X86_64_PC32	fun1

Linker: Realocação de Código

calc.c

```
extern int var1;

void fun1(int x);

int var2 = 0;
int var3 = 0;

int fun2() {
  fun1(var1);
  …
}
```

main.c

```
extern int var2;
extern int var3;

int fun2();

int var1 = 0;

void fun1(int x) {
  …
}

int main() {
  var2 = fun2();
  var3 = fun2();
  …
}
```

Linker: Realocação de Código

main.c

```
extern int var2;
extern int var3;

int fun2();

int var1 = 0;

void fun1(int x) {
  …
}

int main() {
  var2 = fun2();
  var3 = fun2();
  …
}
```

Tabela de Realocação

OFFSET	TIPO	SÍMBOLO
10	R_X86_64_PC32	fun2
17	R_X86_64_32S	var2
1c	R_X86_64_PC32	fun2
23	R_X86_64_32S	var3

Tabela de Símbolos

OFFSET	TIPO	SÍMBOLO
-	U	var2
-	U	var3
-	U	fun2
00	T	fun1
0b	T	main
00	D	var1

main.o

Tabela de Realocação

OFFSET	TIPO	SÍMBOLO
10	R_X86_64_PC32	fun2
17	R_X86_64_32S	var2
1c	R_X86_64_PC32	fun2
23	R_X86_64_32S	var3

Tabela de Símbolos

OFFSET	TIPO	SÍMBOLO
-	U	var2
-	U	var3
-	U	fun2
00	T	fun1
0b	T	main
00	D	var1

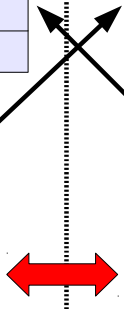
calc.o

Tabela de Realocação

OFFSET	TIPO	SÍMBOLO
07	R_X86_64_32S	var1
0c	R_X86_64_PC32	fun1

Tabela de Símbolos

OFFSET	TIPO	SÍMBOLO
00	D	var2
04	D	var3
00	T	fun2
-	U	var1
-	U	fun1



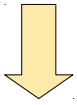
Resolução de Referências

- Cada referência externa deve ser associada a uma definição única (exportação) de um símbolo com o **mesmo nome**
  - Se nenhuma definição com esse nome é encontrada, o ligador termina com erro
  - Se há duas (ou mais) definições com o mesmo nome, o ligador também termina com erro

Resolução de Referências

```
void foo(void);

int main() {
  foo();
  return 0;
}
```



```
/tmp/cc0cCWXd.o: In function `main':
main.c:(.text+0x7): undefined reference to `foo'
collect2: ld returned 1 exit status
```

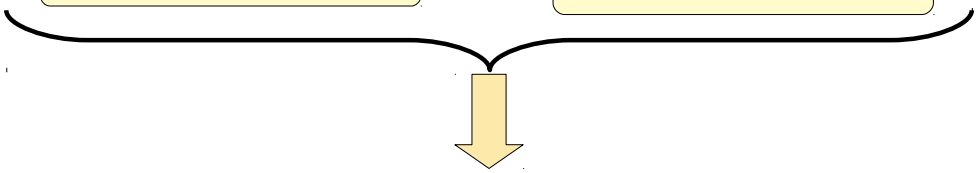
Resolução de Referências

```
int i = 1;

int foo(int j) {
  return i + j;
}
```

```
int foo(int);
int i = 0;

int main() {
  i = foo(3);
  return 0;
}
```



```
main.o:(.bss+0x0): multiple definition of `i'
foo.o:(.data+0x0): first defined here
collect2: ld returned 1 exit status
```

Referências em C

- É importante distinguir definição e referência
  - Uma definição estabelece a representação na memória (localização, tamanho, …)

Referências em C

- Em ISO C:
  - Uma declaração **com inicialização** é uma definição

    int value = 1024;
  - Uma declaração com classe **extern** e sem inicialização é uma referência

    extern int value;
  - Uma declaração **sem inicialização** (e sem *extern*) é uma tentativa de definição

    int value; /* Não deve ser usada! */

Cuidados e Boas Práticas

- Um símbolo global deve ser declarado com o mesmo tipo em todos os módulos
  - A resolução de referências é feita com base apenas no nome
  - O compilador não tem como garantir consistência a não ser com o uso de arquivos de cabeçalho (“interface”)

Cuidados e Boas Práticas

modulo-01.h

```
int foo(void);
```

modulo-02.h

```
extern int contador;
```

modulo-01.c

```
#include "modulo-01.h"
#include "modulo-02.h"

int foo() {
  contador++;
    …
}
```

modulo-02.c

```
#include "modulo-01.h"
#include "modulo-02.h"

int contador = 0;
int main (void) {
  int l = foo();
  …
}
```

Carga de Código
e
Relocação Dinâmica

Carga de Código

- O *linker*, depois de resolver todas as referências, gera o executável já indicando a posição de memória de cada elemento
- Carregar o programa na memória consiste em ler essas informações do executável e colocar os elementos em suas devidas posições
  - Geralmente faz parte do SO
- O executável pode ser visto como o espelho da memória gravada em disco

Carga de Código

- Na ligação, as referências à memória são calculadas em relação ao endereço “virtual” do programa
  - Considera o uso de memória virtual pelo SO
- Quando o programa é carregado na memória, essas referências devem ser relocadas (mapeadas) em relação ao endereço “real”
  - Essa relocação pode ser feita através de uma tradução de endereços feita pelo hardware (MMU)

Bibliotecas Dinâmicas

- Carregadas na memória e “ligadas” em tempo de execução a um programa
  - *Shared Objects* (.so), *Dynamic Link Libraries* (.dll)
- Melhor aproveitamento de memória
  - Bibliotecas estáticas são incorporadas no programa
  - Bibliotecas dinâmicas são compartilhadas
- A carga e ligação da biblioteca é realizada por um ligador dinâmico (*dynamic linker*)