



Ambiente de Desenvolvimento

Walter Fetter Lages

fetter@ece.ufrgs.br

Universidade Federal do Rio Grande do Sul

Escola de Engenharia

Departamento de Engenharia Elétrica

Programa de Pós-Graduação em Engenharia Elétrica

ELE213 Programação de Sistemas de Tempo Real



Tópicos

- Editor de Textos
- Compilador
- Montador
- Linker
 - Linkagem Incremental
 - Linkagem Estática
 - Linkagem Dinâmica
- Make
- Gerenciador de Bibliotecas
- Depurador
- Ambiente Integrado de Desenvolvimento



Introdução

- Linguagem de máquina
- Assembly
- Montador
- Compilador
- Sistema *Host*
- Sistema *Target*



Editor de Textos

- Produz o arquivo fonte (.c, .pas, .java, .cpp, ...)
 - MS-DOS Editor (edit)
 - Windows notepad (notepad)
 - Norton Editor (ne)
 - Brief (brief)
 - Windows Programmer's Editor (wpe)
 - Joe's Own Editor (joe)
 - Visual Editor (vi)
 - VI Improved (vim)
 - KDE Text Editor (kedit)
 - Editor of Macros (emacs)



Compilador

- Traduz os módulos fonte para outra linguagem (.asm, .s) ou para módulos objeto (.obj, .o)
 - Microsoft C (cc)
 - Turbo Pascal (tpc)
 - Turbo C++ (tcc)
 - Borland C++ (bcc)
 - GNU Compiler Collection (gcc, g++, g77, gjc, ...)



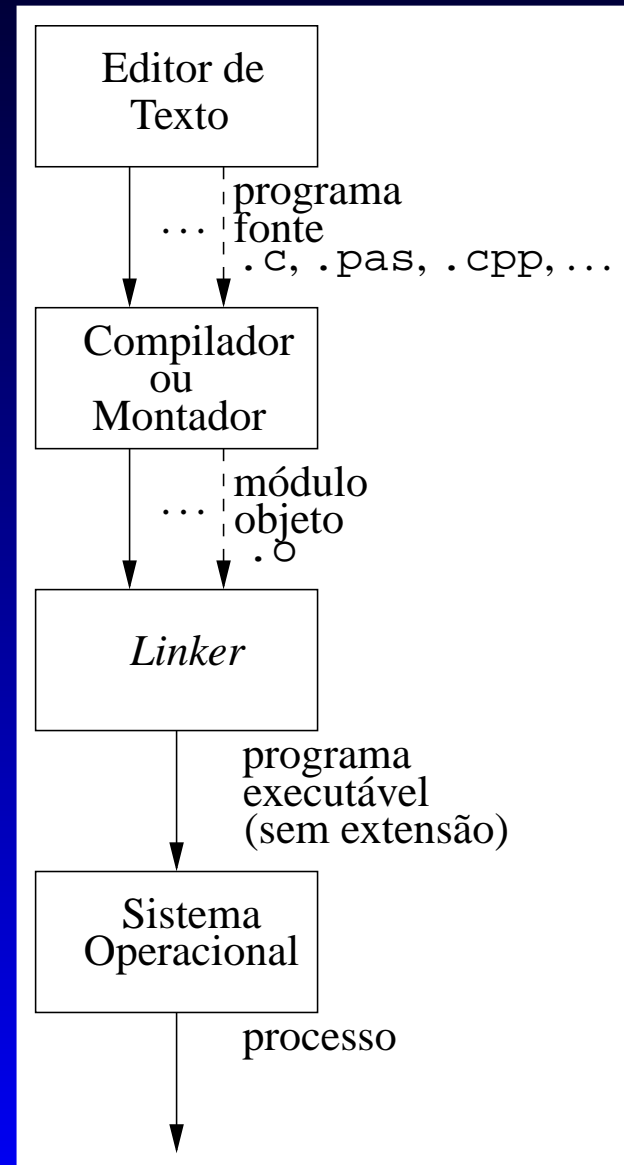
Montador

- Traduz os módulos fonte em Assembly (.asm, .s) para módulos objeto (.obj, .o)
 - Microsoft Macro Assembler (ml, masm)
 - Turbo Assembler (tasm)
 - GNU Assembler (as)
 - Netwide Assembler (nasm)

Linker

- Liga os diversos módulos objeto (`.obj`, `.o`) para criar:
 - um programa executável, ou
 - um módulo maior, ou
 - uma biblioteca de linkagem dinâmica
- As referências internas são resolvidas
 - Microsoft Linker (`link`)
 - Turbo Linker (`tlink`)
 - GNU Linker (`ld`, `collect2`)

Fluxo Mínimo





Gerenciador de Bibliotecas

- Agrupa diversos módulos objeto (.obj, .o) em um arquivo de biblioteca (.lib, .a)
- As referências internas **não** são resolvidas
- A unidade de linkagem é o módulo objeto
 - Microsoft Library Manager (lib)
 - Turbo Library Manager (tlib)
 - GNU Archive (ar)



Depurador

- debug
- Codeview (cv)
- Turbo Debugger (td)
- GNU Debugger (gdb)
 - Front-ends (xxgdb, kdbg)



Linkagem Incremental

- Diversos módulos objeto são ligados para formar um módulo objeto maior
- São resolvidas as referências internas



Linkagem Estática

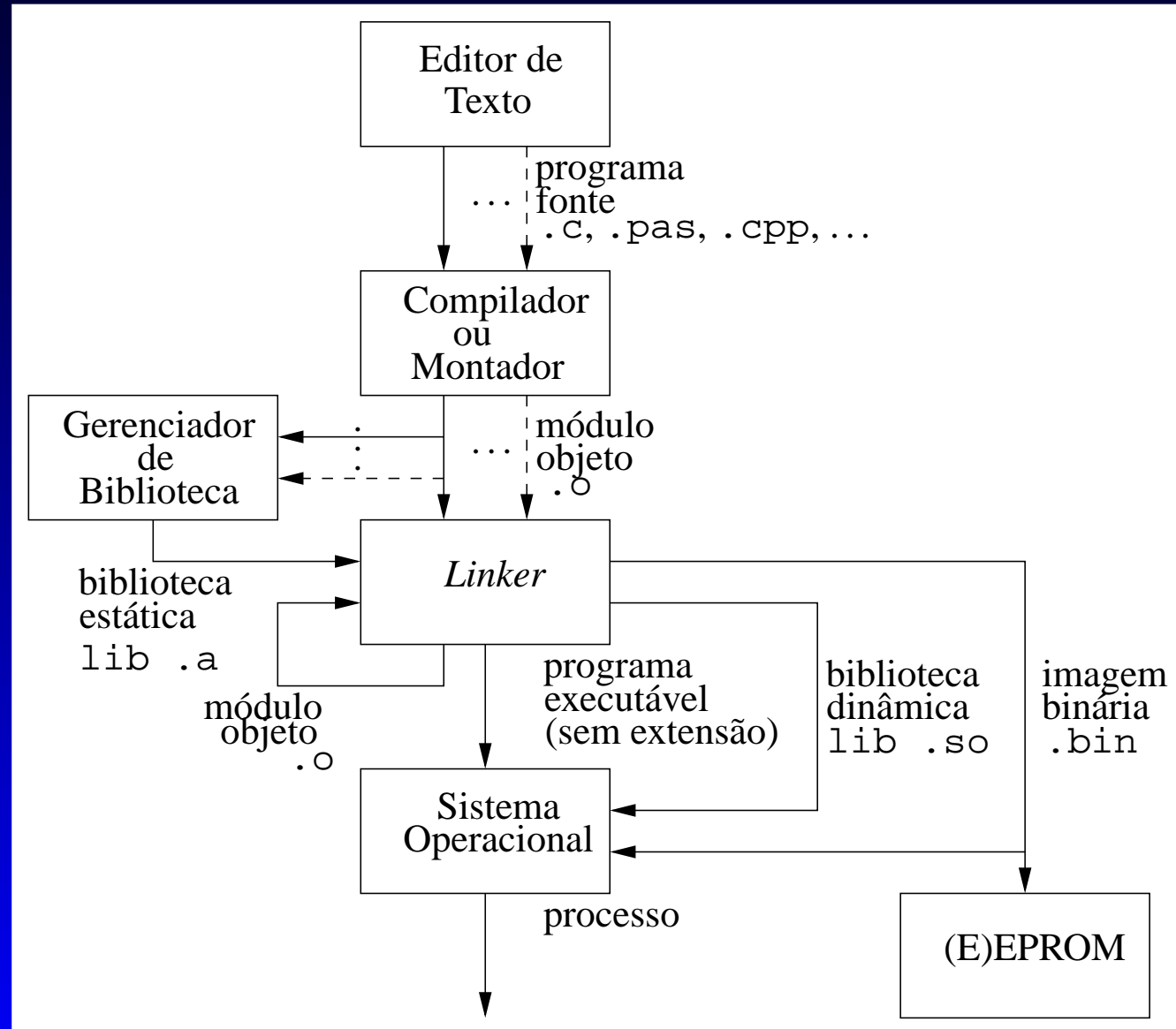
- Um ou mais módulos são ligados para criar um programa executável (.exe) ou imagem binária (.com)
- Podem ser ligados módulos em si ou módulos contidos em bibliotecas estáticas
- Cada programa tem a sua própria copia dos módulos objeto (.obj, .o)



Linkagem Dinâmica

- Bibliotecas dinâmicas (.dll, .so) não são ligadas com o programa pelo linker
- Carregada pelo sistema operacional
- Compartilhada por todos os programas que usam a biblioteca dinâmica

Fluxo Completo



Make

- Programa para automatizar e otimizar o processo de compilação e linkagem
- Executa apenas os procedimentos que são necessários
- Funciona baseado na comparação das datas e horas dos arquivos
- `Makefile` = Arquivo interpretado pelo make
 - Microsoft Program Maintenance Utility (`make`, `nmake`)
 - Borland Make (`make`)
 - GNU Make (`make`)



IDE

- Editor, Compilador, Linker, Make, ... integrados
- Alguns podem ser configurados para utilizar diversos compiladores, linkers, ...
 - Microsoft Visual C (msc)
 - Turbo Pascal (turbo)
 - Borland C++ (bc)
 - Windows Programmer's Editor (wpe)
 - KDE Development Environment (kdevelop)



Exemplo 1

- Hello, World! em C
 - hello.c
 - Makefile



hello.c

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    printf("Hello, world!\n");
    return 0;
}
```



Makefile

```
1 TARGET=hello
2 SRCS=$(TARGET).c
3
4 FLAGS=-O2 -Wall
5 INCLUDE=-I. -I$(HOME)/include
6 LIBDIR=-L$(HOME)/lib
7 LIBS=
8
9 CC=gcc
10 CFLAGS=$(FLAGS) $(INCLUDE)
11 LDFLAGS=$(LIBDIR) $(LIBS)
```



Makefile – Continuação

```
13 all: $(TARGET)
14
15 $(TARGET): $(SRCS:.c=.o)
16     $(CC) -o $@ $^ $(LDFLAGS)
17
18 %.o: %.c
19     $(CC) -MMD $(CFLAGS) -c -o $@ $<
20
21 -include $(SRCS:.c=.d)
22
23 clean:
24     rm -f *~ *.bak *.o *.d
25
26 distclean: clean
27     rm -f $(TARGET)
```



Exemplo 2

- Hello, World! em C++
 - hello.cpp
 - Makefile

hello.cpp

```
#include <iostream>

int main(int argc, char *argv[])
{
    std::cout << "Hello, world!" << std::endl;
    return 0;
}
```



Makefile

```
1 TARGET=hello
2 SRCS=$(TARGET).cpp
3
4 FLAGS=-O2 -Wall
5 INCLUDE=-I. -I$(HOME)/include -I$(HOME)/include/cpp
6 LIBDIR=-L$(HOME)/lib
7 LIBS=
8
9 CXX=g++
10 CXXFLAGS=$(FLAGS) $(INCLUDE)
11 LDFLAGS=$(LIBDIR) $(LIBS)
```

Makefile – Continuação

```
13 all: $(TARGET)
14
15 $(TARGET): $(SRCS:.cpp=.o)
16         $(CXX) -o $@ $^ $(LDFLAGS)
17
18 %.o: %.cpp
19         $(CXX) -MMD $(CXXFLAGS) -c -o $@ $<
20
21 -include $(SRCS:.cpp=.d)
22
23 clean:
24         rm -f *~ *.bak *.o *.d
25
26 distclean: clean
27         rm -f $(TARGET)
```




Exemplo 3

- Hello, World! em Assembly
- Utiliza serviço do Linux para mostrar *string*
- Sintaxe do gas
 - hello.s
 - Makefile

hello.s

```
.intel_syntax noprefix

.text

.global _start

_start: mov     edx,offset len    # message length
        mov     ecx,offset msg   # pointer to message to write
        mov     ebx,1           # file handle (stdout)
        mov     eax,4           # system call number (sys_write)

        int     0x80            # call kernel

        mov     ebx,0           # first argument: exit code
        mov     eax,1           # system call number (sys_exit)
        int     0x80            # call kernel

.data

msg:     .string "Hello, world!\n"
len=     . - msg
```



Makefile

```
1 TARGET=hello
2 SRCS=$(TARGET).s
3
4 FLAGS=-gstabs -a='echo $@ | cut -f 1 -d.`.lst
5 INCLUDE=
6 LIBDIR=
7 LIBS=
8
9 AS=as
10 ASFLAGS=$(FLAGS) $(INCLUDE)
11 LD=ld
12 LDFLAGS=
```



Makefile – Continuação

```
14 all: $(TARGET)
15
16 $(TARGET): $(SRCS:.s=.o)
17     $(LD) $(LDFLAGS) -o $@ $(LIBS) $^
18
19 %.o: %.s
20     $(AS) $(ASFLAGS) -o $@ $<
21
22 clean:
23     rm -f *~ *.bak *.o *.d *.lst
24
25 distclean: clean
26     rm -f $(TARGET)
```



Exemplo 4

- Hello, World! com interface gráfica
- Utiliza o *toolkit* Qt versão 4
 - hello.cpp
 - makefile
 - makefile tem prioridade sobre o Makefile
 - Utiliza qmake para gerar Makefile

hello.cpp

```
#include <QApplication>
#include <QPushButton>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

    QPushButton hello("Hello, world!", 0);
    hello.show();

    return app.exec();
}
```



makefile

```
1 TARGET=hello
2
3 QMAKE=qmake
4
5 all: $(TARGET)
6
7 $(TARGET): Makefile
8     $(MAKE) -f Makefile
9
10 Makefile: $(TARGET).pro
11     $(QMAKE) -o $@
12
13 $(TARGET).pro: $(PWD)
14     $(QMAKE) -project -o $@
```



Referências

- [1] S. Chamberlain and I. L. Taylor. *Using ld, the GNU Linker*, 2003.
<<http://sourceware.org/binutils/docs>>.
- [2] D. Elsner, J. Fenlason, and friends. *Using as, the GNU Assembler*, 2002.
<<http://sourceware.org/binutils/docs>>.
- [3] R. H. Pesch, J. M. Osier, and Cygnus Support. *The GNU Binary Utilities*, 1993.
<<http://sourceware.org/binutils/docs>>.
- [4] R. M. Stallman, R. McGrath, and P. Smith. *GNU Make, A Program for Directing Recompilation*. Boston, 2002.
<<http://www.gnu.org/software/make/manual/make.pdf>>.
- [5] R. M. Stallman, R. Pesch, S. Shebs, and etal. *Debugging with GDB, The GNU Source-Level Debugger*. Boston, 2002.
<<http://sourceware.org/gdb/current/onlinedocs/gdb.pdf.gz>>.
- [6] R. M. Stallman and The GCC Developer Community. *Using the GNU Compiler Collection*, 2004. <<http://gcc.gnu.org/onlinedocs/gcc>>.