



UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA MECÂNICA
Curso de Graduação em Engenharia Mecatrônica



Sistemas Digitais para Mecatrônica
FEELT49081

Tarefa da Semana 02 - Ambiente de Programação Linux
Prof. Éder Alves de Moura

Hericles Felipe Ferraz - 11811EMT022

Uberlândia, 18 de Dezembro de 2021

Sumário

Questão 1 - Liste e descreva o que são as 4 etapas do processo de compilação	3
Questão 2 - Desenvolva uma aplicação simples que demonstre o uso de múltiplos arquivos para a construção de uma aplicação em C	7
Questão 3 - O compilador gcc permite fornecer parâmetros extras, que modificam desde a emissão de erros até o binário final, o otimizando para determinados comportamentos. Explique a função e crie um exemplo para demonstrar a funcionalidade dos seguintes parâmetros:	9
-static: Convencionalmente, os compiladores utilizam bibliotecas compartilhadas em sua construção. A opção com o -static, serve para compilar o código utilizando apenas bibliotecas estáticas. A figura abaixo mostra a compilação usando o método convencional e o -static na prática.	9
Observe que a execução de ambos é equivalente, mas o tamanho final dos arquivos é distinto.	9
Referências	13

1. Questão 1

Liste e descreva o que são as 4 etapas do processo de compilação

As 4 etapas do processo de compilação, são:

- Preprocessing - É o primeiro estágio da compilação. Nesta etapa, é feita uma preparação do que será compilado, onde:
 - Comentários são removidos
 - As linhas que começam com “#”, são decodificadas como comandos do pré-processador. Esses comandos estruturam uma linguagem de escala, formando uma linguagem de macro simples com sua própria sintaxe e semântica, útil para reproduzir trechos de código sem repetição desnecessária.
 - Um exemplo de um código em c, é mostrado abaixo:

```
/* Exemplo básico baseado no hello word */
#include <stdio.h>

int main(void)
{
    // Testando seção de comentário
    printf("Sistemas Digitais para mecatronica! ");
    return 0;
}
```

É possível mostrar-se os resultados do pré-processamento, com o parâmetro -E, como mostrado abaixo:

```
$: gcc -E example_pre_process.c
```

Conforme mostrado abaixo, mostra que os comentários são removidos e existe a codificação do `include <stdio.h>`.

```

extern char *ctermid (char *__s) __attribute__ ((__nothrow__ , __leaf__));
# 840 "/usr/include/stdio.h" 3 4
extern void flockfile (FILE *__stream) __attribute__ ((__nothrow__ , __leaf__));

extern int ftrylockfile (FILE *__stream) __attribute__ ((__nothrow__ , __leaf__)) ;

extern void funlockfile (FILE *__stream) __attribute__ ((__nothrow__ , __leaf__));
# 858 "/usr/include/stdio.h" 3 4
extern int __uflow (FILE *);
extern int __overflow (FILE *, int);
# 873 "/usr/include/stdio.h" 3 4

# 3 "example_pre_process.c" 2

# 4 "example_pre_process.c"
int main(void)
{
    printf("Sistemas Digitais para mecatronica! ");
    return 0;
}
hericles@Dolores ~/Disciplinas/Sistemas_Embarcados/Semana02/scripts/questao1 <master*>
$

```

- Compilation Proper - É o segundo estágio, onde o código pré-processado é convertido em comando do assembly, específico para o processador de destino. Nesta etapa, os comandos ainda são legíveis para os humanos. O resultado desta etapa, pode ser salvo em um arquivo.s, que possuirá o conjunto de instruções provenientes do assembly.

```
$: gcc -S example_pre_process.c
```

Onde o comando acima, gera o seguinte resultado, conforme mostrado na Figura 2.

```

hericles@Dolores ~/Disciplinas/Sistemas_Embarcados/Semana02/scripts/questao1 <master*>
$ gcc -S example_pre_process.c
hericles@Dolores ~/Disciplinas/Sistemas_Embarcados/Semana02/scripts/questao1 <master*>
$ ls
example_pre_process.c  example_pre_process.s
hericles@Dolores ~/Disciplinas/Sistemas_Embarcados/Semana02/scripts/questao1 <master*>
$ head -10 example_pre_process.s
.file "example_pre_process.c"
.text
.section .rodata
.align 8
.LC0:
.string "Sistemas Digitais para mecatronica! "
.text
.globl main
.type main, @function
main:

```

- Assembly - É a terceira etapa, onde o código em assembly é transformado em instruções que são lidas diretamente pelo processador (em linguagem binária). Nesta fase, a saída gerada já não é mais entendível por humanos, e é chamada de object code, resultante do do código de origem, com o mesmo nome exceto pela extensão, que “.o”

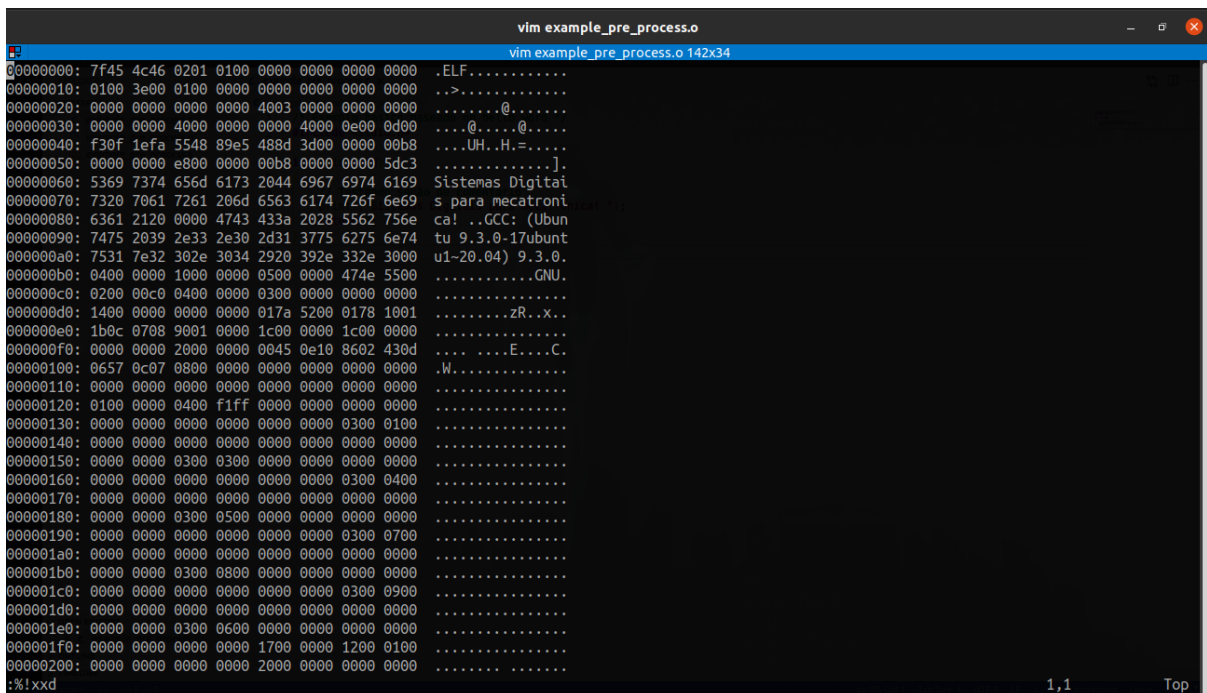
Executando o comando abaixo, tem-se:

```
$: gcc -c example_pre_process.c
```

O código example_pre_process.c será compilado, e um example_pre_process.o será gerado.

```
hericles@Dolores ~/Disciplinas/Sistemas_Embarcados/Semana02/scripts/questao1 <master*>
$ ls
example_pre_process.c  example_pre_process.o  example_pre_process.s
```

Uma forma de abrir esse arquivo.o para verificação, é utilizando o vim com o modo “:%!xxd” do vim.



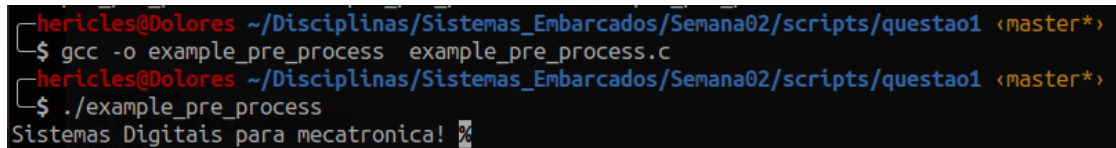
```
vim example_pre_process.o
vim example_pre_process.o 142x34
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF.....
00000010: 0100 3e00 0100 0000 0000 0000 0000 0000  ..>.....
00000020: 0000 0000 0000 0000 4003 0000 0000 0000  .....@.....
00000030: 0000 0000 4000 0000 0000 4000 0e00 0d00  ....@....@....
00000040: f30f 1efa 5548 89a5 488d 3d00 0000 00b8  ....UH..H.=....
00000050: 0000 0000 e800 0000 00b8 0000 0000 5dc3  .....].
00000060: 5369 7374 656d 6173 2044 6967 6974 6169  Sistemas Digitai
00000070: 7320 7061 7261 206d 6563 6174 726f 6e69  s para mecatroni
00000080: 6361 2120 0000 4743 433a 2028 5562 756e  ca! ..GCC: (Ubn
00000090: 7475 2039 2e33 2e30 2d31 3775 6275 6e74  tu 9.3.0-17ubunt
000000a0: 7531 7e32 302e 3034 2920 392e 332e 3000  u1-20.04) 9.3.0.
000000b0: 0400 0000 1000 0000 0500 0000 474e 5500  ....GNU.
000000c0: 0200 00c0 0400 0000 0300 0000 0000 0000  .....
000000d0: 1400 0000 0000 0000 017a 5200 0178 1001  ....zR..X..
000000e0: 1b0c 0708 9001 0000 1c00 0000 1c00 0000  .....
000000f0: 0000 0000 2000 0000 0045 0e10 8602 430d  ....E....C.
00000100: 0657 0c07 0800 0000 0000 0000 0000 0000  .W.....
00000110: 0000 0000 0000 0000 0000 0000 0000 0000  .....
00000120: 0100 0000 0400 f1ff 0000 0000 0000 0000  .....
00000130: 0000 0000 0000 0000 0000 0000 0300 0100  .....
00000140: 0000 0000 0000 0000 0000 0000 0000 0000  .....
00000150: 0000 0000 0300 0300 0000 0000 0000 0000  .....
00000160: 0000 0000 0000 0000 0000 0000 0300 0400  .....
00000170: 0000 0000 0000 0000 0000 0000 0000 0000  .....
00000180: 0000 0000 0300 0500 0000 0000 0000 0000  .....
00000190: 0000 0000 0000 0000 0000 0000 0300 0700  .....
000001a0: 0000 0000 0000 0000 0000 0000 0000 0000  .....
000001b0: 0000 0000 0300 0800 0000 0000 0000 0000  .....
000001c0: 0000 0000 0000 0000 0000 0000 0300 0900  .....
000001d0: 0000 0000 0000 0000 0000 0000 0000 0000  .....
000001e0: 0000 0000 0300 0600 0000 0000 0000 0000  .....
000001f0: 0000 0000 0000 0000 1700 0000 1200 0100  .....
00000200: 0000 0000 0000 0000 2000 0000 0000 0000  .....
:%!xxd
1,1 Top
```

- Linking - A quarta e última etapa é onde todas as ligações de chamadas são feitas. Um arquivo de saída é gerado, que pode ser diretamente executado com o comando:

```
$: gcc -o example_pre_process example_pre_process.c
```

Um exemplo do que é feito nesta etapa, é na utilização da função printf, no código de exemplo. Quando o linking passa pela função printf, é efetuado uma conexão entre as chamadas de funções internas do sistema operacional com o do executável.

Um exemplo desta etapa, é mostrado na imagem abaixo.



```
hericles@Dolores ~/Disciplinas/Sistemas_Embarcados/Semana02/scripts/questao1 <master*>  
$ gcc -o example_pre_process example_pre_process.c  
hericles@Dolores ~/Disciplinas/Sistemas_Embarcados/Semana02/scripts/questao1 <master*>  
$ ./example_pre_process  
Sistemas Digitais para mecatronica! %
```

2. Questão 2

Desenvolva uma aplicação simples que demonstre o uso de múltiplos arquivos para a construção de uma aplicação em C

Neste exercício, foi inserido como exemplo o trabalho final da disciplina de Sistemas Operacionais. Cada grupo possuía como objetivo um distinto tipo de desafio, e o que foi escolhido pelo grupo ao qual fiz parte foi de desenvolver códigos que simulasse o funcionamento básico do interpretador de comandos da família Unix, como o comando “ls, pwd, cd, mkdir”, dentre outros.

Foi inserido a documentação do trabalho efetuado, caso seja de interesse. Neste projeto, existem os diretórios:

- Documentação - Relatório com explicação do que foi desenvolvido
- include - Todos os .h que foram criados para posterior importação
- src - Todos os .c desenvolvidos

A ideia, foi de deixar cada código o mais separado possível, de forma semelhante ao que é feito na própria família Unix.

Para compilar o projeto, é necessário efetuar o comando:

```
$ gcc -w -o shell -pthread shell.c
```

Em seguida, é necessário executar o projeto:

```
$ ./shell
```

A imagem abaixo mostra o projeto sendo compilado e executado:

```
hericles@dolores ~/Disciplinas/Sistemas_Embarcados/Semana02/scripts/questao2/shell_linux/src <master*>
$ gcc -w -o shell -pthread shell.c
/usr/bin/ld: /tmp/ccnrb6cf.o: in function `main':
shell.c:(.text+0x31fd): warning: the `gets' function is dangerous and should not be used.
hericles@dolores ~/Disciplinas/Sistemas_Embarcados/Semana02/scripts/questao2/shell_linux/src <master*>
$ ./shell
```

Um exemplo do funcionamento do shell desenvolvido, é mostrado abaixo:

```
Desenvolvido pelos Alunos :
Hericles Felipe Ferraz - 11811EMT022
Luiz da Silva Moura - 11611EMT028
Marcus Vinicius Miata - 11811BCC017
Pedro Henrique Rabis Diniz - 11811BCC024

hericles:/home/hericles/Disciplinas/Sistemas_Embarcados/Semana02/scripts/questao2/shell_linux/src
$ ls
List Directory: .
cp.c  pipe.c  shell  mkdir.c  cd.c  ls.c  shell.c  pwd.c  exec.c  stop.c  rmdir.c  re_direct.c

hericles:/home/hericles/Disciplinas/Sistemas_Embarcados/Semana02/scripts/questao2/shell_linux/src
$ pwd
/home/hericles/Disciplinas/Sistemas_Embarcados/Semana02/scripts/questao2/shell_linux/src

hericles:/home/hericles/Disciplinas/Sistemas_Embarcados/Semana02/scripts/questao2/shell_linux/src
$ cd ..

hericles:/home/hericles/Disciplinas/Sistemas_Embarcados/Semana02/scripts/questao2/shell_linux
$ cd src

hericles:/home/hericles/Disciplinas/Sistemas_Embarcados/Semana02/scripts/questao2/shell_linux/src
$ ls -l
List Directory: . -l
cp.c      -rw-rw-r--  1  hericles  hericles  3471  Mon Dec 21 17:56:08 2020
pipe.c    -rw-rw-r--  1  hericles  hericles  2885  Mon Dec 21 18:45:51 2020
shell     -rwxrwxr-x  1  hericles  hericles  40328 Sat Dec 18 19:43:24 2021
mkdir.c   -rw-rw-r--  1  hericles  hericles   904  Sat Dec 19 14:10:24 2020
cd.c      -rw-rw-r--  1  hericles  hericles  1252  Mon Dec 21 16:12:09 2020
ls.c      -rw-rw-r--  1  hericles  hericles  4217  Sat Dec 19 14:10:24 2020
shell.c   -rw-rw-r--  1  hericles  hericles  4369  Sat Dec 18 19:31:46 2021
pwd.c     -rw-rw-r--  1  hericles  hericles  1045  Mon Dec 21 16:14:11 2020
exec.c    -rw-rw-r--  1  hericles  hericles  1923  Sat Dec 19 14:10:24 2020
```


3. Questão 3 -

O compilador gcc permite fornecer parâmetros extras, que modificam desde a emissão de erros até o binário final, o otimizando para determinados comportamentos. Explique a função e crie um exemplo para demonstrar a funcionalidade dos seguintes parâmetros:

-static: Convencionalmente, os compiladores utilizam bibliotecas compartilhadas em sua construção. A opção com o -static, serve para compilar o código utilizando apenas bibliotecas estáticas. A figura abaixo mostra a compilação usando o método convencional e o -static na prática.

```
hericles@Dolores ~/Disciplinas/Sistemas_Embarcados/Semana02/scripts <master*>
$ mv ../scripts/questao1/example_pre_process.c questao3/example_static.c
hericles@Dolores ~/Disciplinas/Sistemas_Embarcados/Semana02/scripts <master*>
$ cd questao3
hericles@Dolores ~/Disciplinas/Sistemas_Embarcados/Semana02/scripts/questao3 <master*>
$ gcc -o compile_default example_static.c
hericles@Dolores ~/Disciplinas/Sistemas_Embarcados/Semana02/scripts/questao3 <master*>
$ gcc -static -o compile_static example_static.c
hericles@Dolores ~/Disciplinas/Sistemas_Embarcados/Semana02/scripts/questao3 <master*>
$ ./compile_default
Sistemas Digitais para mecatronica! %
hericles@Dolores ~/Disciplinas/Sistemas_Embarcados/Semana02/scripts/questao3 <master*>
$ ./compile_static
Sistemas Digitais para mecatronica! %
hericles@Dolores ~/Disciplinas/Sistemas_Embarcados/Semana02/scripts/questao3 <master*>
$ ls -l
total 876
-rwxrwxr-x 1 hericles hericles 16704 dez 18 20:50 compile_default
-rwxrwxr-x 1 hericles hericles 871696 dez 18 20:50 compile_static
-rw-rw-r-- 1 hericles hericles 178 dez 18 18:05 example_static.c
```

Observe na figura acima, que a execução de ambos é equivalente, mas o tamanho final dos arquivos é distinto.

-g: Quando compila-se utilizando o parâmetro -g, são inseridas informações de debug no binário resultante. A imagem abaixo, mostra a diferença entre a compilação sem o -g, denominada de compile_default, e a utilizando o -g, denominada compile_g.

```
hericles@Dolores ~/Disciplinas/Sistemas_Embarcados/Semana02/scripts/questao3 <master*>
$ gcc -g -o compile_g example_static.c
hericles@Dolores ~/Disciplinas/Sistemas_Embarcados/Semana02/scripts/questao3 <master*>
$ ./compile_g
Sistemas Digitais para mecatronica!
hericles@Dolores ~/Disciplinas/Sistemas_Embarcados/Semana02/scripts/questao3 <master*>
$ ls -l
total 896
-rwxrwxr-x 1 hericles hericles 16704 dez 18 20:50 compile_default
-rwxrwxr-x 1 hericles hericles 19240 dez 18 21:12 compile_g
-rwxrwxr-x 1 hericles hericles 871696 dez 18 20:50 compile_static
-rw-rw-r-- 1 hericles hericles 178 dez 18 18:05 example_static.c
```

É interessante notar, que mesmo com as informações de debug, o executável compile_g, é consideravelmente menor do que o compile_static utilizando - static na compilação, e um pouco maior do que o comando padrão que resultou no compile_default.

-pedantic: Serve para mostrar todos os warning necessários com as conformidades do padrão ANSI/ISO C. Caso o programa seja compilado com o parâmetro -pedantic-erros, todos os warning são considerados como erros. Um exemplo disso, é mostrado considerando o código abaixo:

```
/* Exemplo com pedant */
#include <stdio.h>

void main(void)
{
    long long int i = 0l;
    fprintf(stdout, "Programa sem as conformidades do padrao ANSI/ISO C:
\n");
}
```

Executando os comandos conforme mostrado na figura abaixo, tem-se:

```
hericles@Dolores ~/Disciplinas/Sistemas_Embarcados/Semana02/scripts/questao3 <master*>
$ gcc -pedantic -o pendant example_pedantic.c
example_pedantic.c:4:6: warning: return type of 'main' is not 'int' [-Wmain]
  4 | void main(void)
    | ~~~~~
hericles@Dolores ~/Disciplinas/Sistemas_Embarcados/Semana02/scripts/questao3 <master*>
$ gcc -pedantic-errors -o pendant example_pedantic.c
example_pedantic.c:4:6: error: return type of 'main' is not 'int' [-Wmain]
  4 | void main(void)
    | ~~~~~
```

-Wall: O parâmetro -Wall, serve para mostrar todos os warning que forem possíveis do gcc. A imagem abaixo mostra este parâmetro em execução.

```
hericles@Dolores ~/Disciplinas/Sistemas_Embarcados/Semana02/scripts/questao3 <master*>
$ gcc -Wall -o compile_pendant example_pedantic.c
example_pedantic.c:4:6: warning: return type of 'main' is not 'int' [-Wmain]
  4 | void main(void)
    |      ^~~~~~
example_pedantic.c: In function 'main':
example_pedantic.c:6:16: warning: unused variable 'i' [-Wunused-variable]
  6 | long long int i = 0l;
    |                ^
hericles@Dolores ~/Disciplinas/Sistemas_Embarcados/Semana02/scripts/questao3 <master*>
$ gcc -o compile_pendant example_pedantic.c
```

Onde é mostrado o destaque dos warnings utilizando-se o -Wall, e nenhum warning sendo mostrado na versão default.

-Os: As otimizações de nível -Os, são equivalentes a de nível O3, porém com recursos direcionados aos entry points. Conforme mostrado no manual do programa, na figura abaixo.

```
-fdeclone-ctor-dtor
The C++ ABI requires multiple entry points for constructors and destructors: one for a base subobject, one for a complete object, and one for a virtual destructor that calls operator delete afterwards. For a hierarchy with virtual bases, the base and complete variants are clones, which means two copies of the function. With this option, the base and complete variants are changed to be thunks that call a common implementation.

Enabled by -Os.
```

-O3: As otimizações de nível O3 incluem toda a otimização de nível O2, com ajustes de velocidade, controle de loops e ações específicas do processador.

```
This tells GCC to deduce the hardware multiply support based upon the MCU name provided by the -mmcu option. If no -mmcu option is specified or if the MCU name is not recognized then no hardware multiply support is assumed. "auto" is the default setting.
```

```
Hardware multiplies are normally performed by calling a library routine. This saves space in the generated code. When compiling at -O3 or higher however the hardware multiplier is invoked inline. This makes for bigger, but faster code.
```

```
The hardware multiply routines disable interrupts whilst running and restore the previous interrupt state when they finish. This makes them safe to use inside interrupt handlers as well as in normal code.
```

No caso do O3 e do OS, a principal diferença seria no tempo de execução. Conforme mostrado na figura abaixo, ambos possuem o mesmo tamanho.

```
hericles@Dolores ~/Disciplinas/Sistemas_Embarcados/Semana02/scripts/questao3 <master*>
$ ls -l
total 1000
-rwxrwxr-x 1 hericles hericles 16752 dez 18 23:44 a.out
-rwxrwxr-x 1 hericles hericles 16704 dez 18 23:34 compile_default
-rwxrwxr-x 1 hericles hericles 19240 dez 18 21:12 compile_g
-rwxrwxr-x 1 hericles hericles 16752 dez 18 23:50 compile_o3
-rwxrwxr-x 1 hericles hericles 16752 dez 18 23:50 compile_os
-rwxrwxr-x 1 hericles hericles 16752 dez 18 23:36 compile_pendant
-rwxrwxr-x 1 hericles hericles 871696 dez 18 20:50 compile_static
-rw-rw-r-- 1 hericles hericles 163 dez 18 23:29 example_pedantic.c
-rw-rw-r-- 1 hericles hericles 178 dez 18 18:05 example_static.c
-rwxrwxr-x 1 hericles hericles 16752 dez 18 23:29 pendant
```

4. Referências

- 1 - <https://teachcomputerscience.com/the-four-stages-of-compilation/>
- 2 - <https://www.calleluks.com/the-four-stages-of-compiling-a-c-program/>
- 3 - <https://www.linkedin.com/pulse/four-stages-compiling-c-program-leine-francisca-valente/>
- 4- <https://qastack.com.br/programming/2855121/what-is-the-purpose-of-using-pedantic-in-gc-c-g-compiler>
- 5- <https://gcc.gnu.org/onlinedocs/gcc-11.2.0/gcc.pdf>