



UNIVERSIDADE FEDERAL DE UBERLÂNDIA-UFU
FACULDADE DE ENGENHARIA MECÂNICA
ENGENHARIA MECATRÔNICA
SISTEMAS DIGITAIS PARA MECATRÔNICA



Relatório Semana 2

GABRIEL RODRIGUES BARBOSA

11821EMT011

UBERLÂNDIA

2023

1- **Após ver os vídeos, podemos ir para a segunda atividade:**

2. Utilizando o arquivo o PDF auxiliar, da fonte “Kurt Wall. Linux Programming Unleashed. SAMS, 2007. Capítulo 3” e em pesquisas realizadas na internet, desenvolva as atividades propostas. Crie um arquivo ‘exercicio02.txt’ e coloque as repostas desta atividade:

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

1 – Pré-compilação: pré processa vários quesitos do script, como substituições de macros e compilação condicional, ou seja, tudo que não é encargo do compilador, mas que precisa ser feito.

2 – Compilação: Verificação de sintaxe e análise lexical (verificação estática de código), e posterior tradução em código intermediário ou código de conjunto.

3 – Qualificação: Conversão do código intermediário ou código de conjunto em binário, para interpretação pela máquina.

4 - Conexão: Essa etapa liga todos os arquivos necessários, como objetos, inicializadores e bibliotecas, montando um executável, que pode ser carregado em memória para execução.

b) 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 dos seguintes parâmetros:

i)-static: A etapa de conexão da compilação é feita de forma estática, não levando em conta a execução dinâmica de bibliotecas durante seu funcionamento

ii)-g: Requisita informação do debug de código

iii)-pedantic: Esse parâmetro desliga as permissões de certas extensões, gerando warnings adicionais na compilação caso elas existam.

iv)-Wall: Habilita todos warnings durante o processo de compilação.

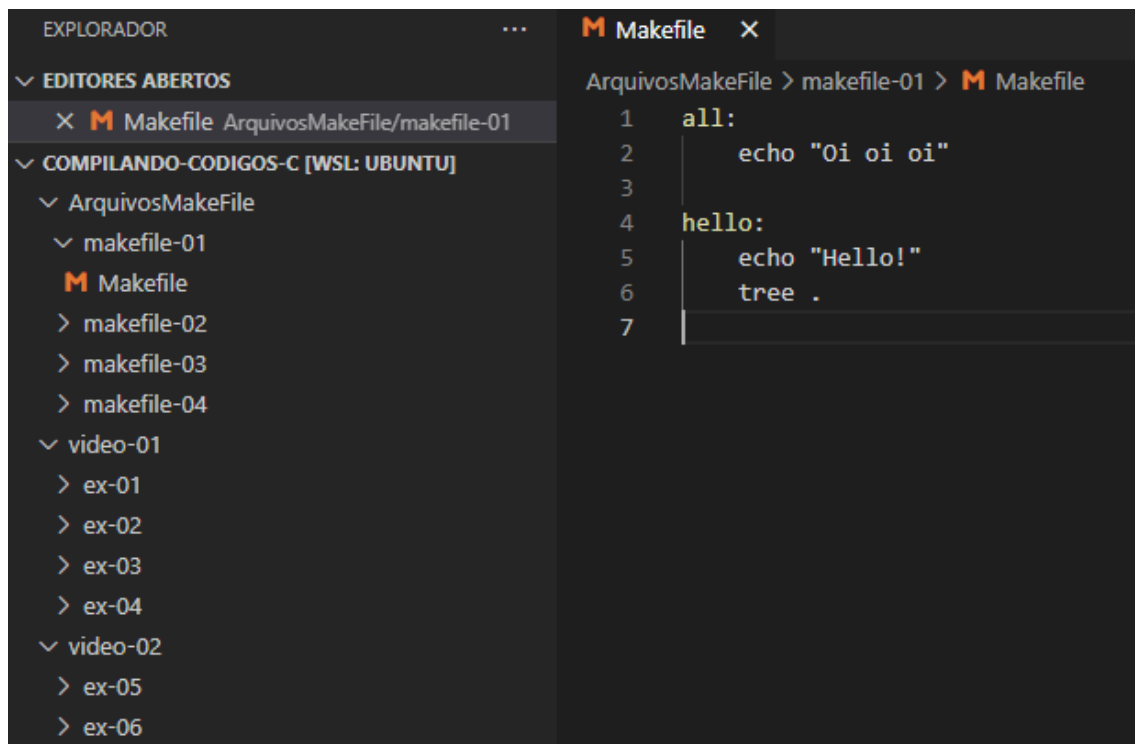
v)-Os: Faz com que o compilador tente otimizar a compilação voltado à tamanho do arquivo final.

vi) -O3: Faz com que o compilador tente otimizar a compilação voltado à performance o máximo possível

3- Veja os cinco primeiros vídeos da seguinte lista e reproduza os exemplos de cada vídeo. Crie uma subpasta 'Exercicio03' e organize os arquivos desta atividade dentro dessa pasta

Os vídeos foram assistidos e os arquivos foram criados, aqui o resumo com os prints dos arquivos Makefile:

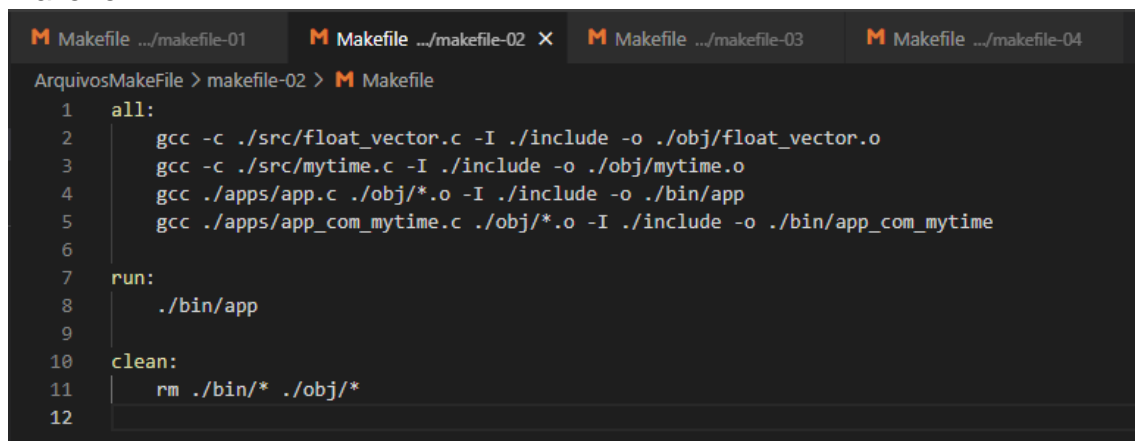
Makefile1:



```
EXPLORADOR
...
EDITORES ABERTOS
  X M Makefile ArquivosMakeFile/makefile-01
COMPILANDO-CODIGOS-C [WSL: UBUNTU]
  v ArquivosMakeFile
    v makefile-01
      M Makefile
    > makefile-02
    > makefile-03
    > makefile-04
  v video-01
    > ex-01
    > ex-02
    > ex-03
    > ex-04
  v video-02
    > ex-05
    > ex-06

ArquivosMakeFile > makefile-01 > M Makefile
1  all:
2      echo "Oi oi oi"
3
4  hello:
5      echo "Hello!"
6      tree .
7
```

Makefile2:



```
M Makefile .../makefile-01  M Makefile .../makefile-02 X  M Makefile .../makefile-03  M Makefile .../makefile-04

ArquivosMakeFile > makefile-02 > M Makefile
1  all:
2      gcc -c ./src/float_vector.c -I ./include -o ./obj/float_vector.o
3      gcc -c ./src/mytime.c -I ./include -o ./obj/mytime.o
4      gcc ./apps/app.c ./obj/*.o -I ./include -o ./bin/app
5      gcc ./apps/app_com_mytime.c ./obj/*.o -I ./include -o ./bin/app_com_mytime
6
7  run:
8      ./bin/app
9
10 clean:
11     rm ./bin/* ./obj/*
12
```

Makefile3:

```
ArquivosMakeFile > makefile-03 > M Makefile
1 APPS = ./apps
2 BIN = ./bin
3 OBJ = ./obj
4 SRC = ./src
5 INCLUDE = ./include
6
7 all: libed myapps
8
9 libed:
10 gcc -c $(SRC)/float_vector.c -I $(INCLUDE) -o $(OBJ)/float_vector.o
11 gcc -c $(SRC)/mytime.c -I $(INCLUDE) -o $(OBJ)/mytime.o
12
13 myapps:
14 gcc $(APPS)/app.c $(OBJ)/*.o -I $(INCLUDE) -o $(BIN)/app
15 gcc $(APPS)/app_com_mytime.c $(OBJ)/*.o -I $(INCLUDE) -o $(BIN)/app_com_mytime
16
17 run:
18 ./bin/app
19
20 clean:
21 rm ./bin/* ./obj/*
22
```

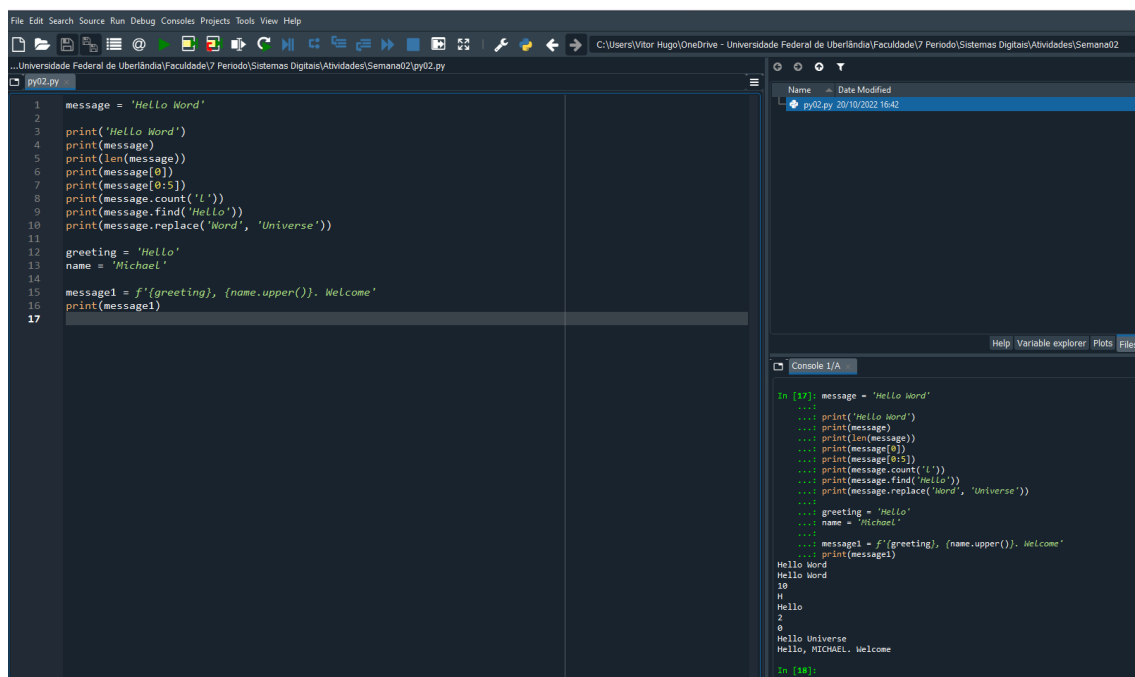
Makefile4:

```
ArquivosMakeFile > makefile-04 > M Makefile
1 APPS = ./apps
2 BIN = ./bin
3 INCLUDE = ./include
4 OBJ = ./obj
5 SRC = ./src
6
7 all: libed myapps
8
9 libed: \
10 float_vector.o \
11 mytime.o
12
13 myapps:
14 gcc $(APPS)/app.c $(OBJ)/*.o -I $(INCLUDE) -o $(BIN)/app
15 gcc $(APPS)/app_com_mytime.c $(OBJ)/*.o -I $(INCLUDE) -o $(BIN)/app_com_mytime
16
17 %.o: $(SRC)/%.c $(INCLUDE)/%.h
18 gcc -c $< -I $(INCLUDE) -o $(OBJ)/$@
19
20
21 run:
22 ./bin/app
23
24 clean:
25 rm -rf ./bin/* ./obj/*
```

4 - Essa atividade consiste em assistir os vídeos do curso introdutório de Python, disponível em:

<https://www.youtube.com/watch?v=k9TUPpGqYTo&list=PLosiE80TeTskrapNbzxHwoFUIlCjGgY7&index=2> e reproduzir os códigos de cada aula. Os vídeos da lista estão numerados de 1 até 26. Você deve reproduzir APENAS os códigos disponíveis nos vídeos de 2 até 20 (exceto os dos vídeos 12 e 13). Crie uma subpasta 'Exercicio04' e organize os arquivos desta atividade dentro dessa pasta. Você deverá criar um arquivo 'pyXX.py', onde o XX corresponde ao número do vídeo na lista, contendo todo o desenvolvimento proposto.

Os vídeos foram vistos e todos os exercícios resolvidos. Para isso, utilizei o software Spyder, o print do código **py02.py** está como exemplo:



```
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Vitor Hugo\OneDrive - Universidade Federal de Uberlândia\Faculdade\7 Período\Sistemas Digitais\Atividades\Semana02\py02.py
py02.py
1 message = 'Hello Word'
2
3 print('Hello Word')
4 print(message)
5 print(len(message))
6 print(message[0])
7 print(message[0:5])
8 print(message.count('l'))
9 print(message.find('Hello'))
10 print(message.replace('Word', 'Universe'))
11
12 greeting = 'Hello'
13 name = 'Michael'
14
15 message1 = f'{greeting}, {name.upper()}. Welcome'
16 print(message1)
17
```

```

Name      Date Modified
py02.py   20/10/2022 16:42

Help Variable explorer Plots Files

Console 1/A

In [17]: message = 'Hello Word'
...: print('Hello Word')
...: print(message)
...: print(len(message))
...: print(message[0])
...: print(message[0:5])
...: print(message.count('l'))
...: print(message.find('Hello'))
...: print(message.replace('Word', 'Universe'))
...: greeting = 'Hello'
...: name = 'Michael'
...: message1 = f'{greeting}, {name.upper()}. Welcome'
...: print(message1)
Hello Word
Hello Word
10
H
Hello
2
0
Hello Universe
Hello, MICHAEL. Welcome

In [18]:
```