Universidade Federal do Ceará
CC - Campus Quixadá
Aluno: Antônio Andson
https://github.com/andsonsilv

25/03/2019

Descomplicando o místico Makefile

1 Makefile

O objetivo do Makefile é definir regras de compilação p/ projetos de software. Tais regras são definidas em arquivo chamando **Makefile**. O programa **make** interpreta o conteúdo do Makefile e executa as regras lá definidas. O programa make pode variar de um sistema a outro, visto que não faz parte de nenhuma normalização.

1.1 Makefile na prática

Vamos primeiramente criar um arquivo.c básico, pode ser um "Olá Mundo".

```
#include<stdio.c>
int main(void){
    printf("Ola Mundo\n");
    return 0;
}
```

Compilando pelo terminal o arquivo.c: "gcc -o binario arquivo.c". Verificar se o gcc está instalado (você pode instalar assim: sudo apt-get install gcc). Consequentemente se executamos o ./binario, a execução será concluída perfeitamente.

Agora vamos compilar e executar via Makefile. Cria-se um arquivo chamado **makefile** na pasta onde o arquivo.c está inserido. O Makefile funciona de acordo com regras. Observe a seguir a relação inicial do **makefile**.

```
regra:dependencia
        comando
```

o primeira regra é o **all**. All é o nome das regras a serem executadas. O **all** chama um "função", no caso por padrão vamos chama-lá de **"main"**. A **main** vai ser a dependência, logo a **main** vai se relacionar com o arquivo.c e sequencialmente o **comando** para compilação deve ser escrito. Segue **makefile** abaixo:

```
all: main

main: arquivo.c
        gcc -o binario arquivo.c
```

Agora é só digitar **make** no terminal, no local onde o **makefile** foi criado que o novo binário será criado. Certifique-se que o **make** esteja instalado, caso contrário, instale com os comandos: sudo apt-get install make e sudo apt-get install make-guile. Se aparecer o erro: **make: Nothing to be done for 'all'.**, é só remover o binário gerado na primeira: rm binario. Pronto, repita o comando **make** novamente e o novo binario foi gerado, agora é só executar normalmente: ./binario.

1.2 Compilando e montando programas que têm vários arquivos fontes

Para um arquivofonte .c é bem entendível e simples né?! Para vamos também é!

Vamos falar sobre compilação e montagem, apesar de muitos acharem que é a mesma coisa, não é. A compilação é a etapa que transforma seu programa em código de máquina, e a montagem junta todos os pedaços necessários para fazer seu programa (incluí até algumas coisas que envolve baixo nível, e que neste momento você apenas precisa saber que existe para rodar seu programa por completo). Na compilação os objetos são criados, e na montagem, os objetos criados anteriormente são "fundidos".

Agora vamos para prática. vamos criar outro arquivo.c chamado arquivocomplementar.c. Veja abaixo seu código.

```
#include<stdio.h>
#include"headers/arquivocomplementar.h"

int main(){
    andstore();
    printf("@andstorevirtual\n");
    return 0;
}
```

O header (cabeçalho) é bem simples, não vamos entrar em detalhes, mas para quem nunca viu, ele serve para que possamos colocar as funções de um arquivo em outros arquivos.c. A função **andstore()** pertencente ao arquivocomplementar.c foi usada no **arquivo.c**.

Código arquivocomplementar.h

```
#ifndef ARQUIVOCOMPLEMENTAR_H_INCLUDED
#define ARQUIVOCOMPLEMENTAR_H_INCLUDED

void andstore();

#endif //ARQUIVOCOMPLEMENTAR_H_INCLUDED
```

Para apenas compilar um arquivo você deve usar o comando:

```
$ gcc -c arquivo.c
```

O gcc criará um arquivo: **arquivo.o** (a extensão .o significa programa-objeto ou simplesmente objeto). Não foi necessário especificar o nome do arquivo gerado. Faça consequentemente para o arquivocomplementar.c.

```
$ gcc -c arquivocomplementar.c
```

Nesse caso, os dois objetos precisam ser montados para formar o executável/binário, porém reforçando, para n arquivos.c, é necessário montar n objetos.

```
gcc -o binario arquivo.o arquivocomplementar.o
```

(binario foi o nome dado intuitivamente, mas poderia ser outro nome). O binário foi gerado, agorá e só executar o próprio.

```
./binario
```

Lembrando do código do arquivocomplementar.c, visto que não foi mencionado acima:

```c
#include<stdio.h>
#include"headers/arquivocomplementar.h"

void andstore(){
        printf("@andstorevirtual @andstorevirtual\n");
}
```

Após a execução do executável/binário, temos a saída:

```
$ @andstorevirtual @andstorevirtual
$ @andstorevirtual
```

1.2.1 Makefile é vida!

Agora imagine, será que precisamos fazer esse processo todas as vezes. Makefile entra em ação de novo. Como foi explicado passo-a-passo o processo acima, o makefile vai parecer que é muito fácil. Então "sem mais delongas",

```
all: prog

prog: arquivo.o arquivocomplementar.o
    gcc -o binario arquivo.o arquivocomplementar.o

arquivo.o: arquivo.c
        gcc -c arquivo.c

arquivocomplementar.o: arquivocomplementar.c
                gcc -c arquivocomplementar.c

clean:
   rm -rf *.o

rmproper: clean
   rm -rf binario

run: prog
   ./binario
```

- clean: usado para apagar arquivos-objeto (*.o) e outros arquivos temporários.
- rmproper: usado para apagar tudo que precisa ser modificado.
- run: para executar o programa.

Para usar os alvos é só dá o camando:

```
$ make clean
$ make rmproper
$ make run
```