Departamento de Engenharia Informática

Sistemas Operativos 1

Utilitário Make

Novembro 2002

1. Modelo de Compilação da Linguagem C

Compilar um programa muito simples em C requer, pelo menos, o ficheiro de texto com extensão .c que contém o programa e as bibliotecas apropriadas (ficheiros com extensão .h). Até à obtenção do ficheiro executável são precisos 3 passos (pelo menos!), como ilustrado na Figura 1.

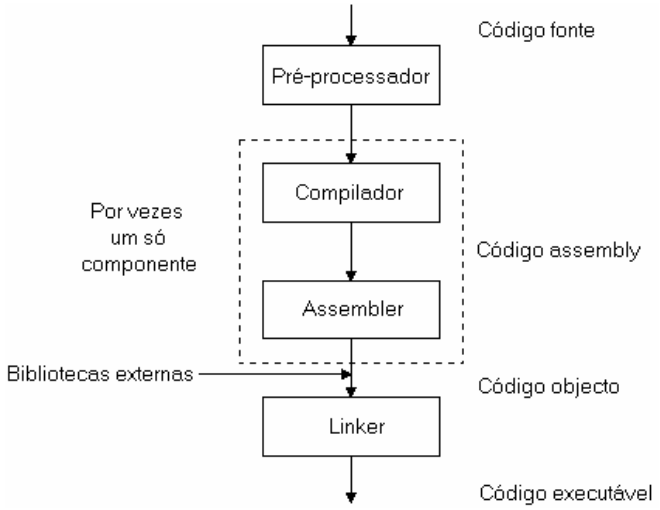

Figura 1 - Modelo de compilação em C

O processo da Figura 1 pode ser despoletado pela seguinte linha de comandos:

gcc exemplo.c

1.1. 1º Passo - o Pré-processador

O pré-processador modifica o código fonte com dois objectivos principais: o de remover os comentários e de interpretar directivas a si dirigidas, ou seja, linhas que começam com o caracter #, nomeadamente #include e #define. A primeira directiva insere o conteúdo de um ficheiro de texto, normalmente designados por cabeçalhos (header files) e têm a extensão .h no ficheiro corrente. A segunda directiva define um nome simbólico cujas ocorrências serão substituídas por outro nome ou constante.

1.2. 2º Passo - o Compilador e o Assembler

O compilador converte os ficheiros fonte que contém linguagem de alto nível, como o C, recebidos do pré-processador, em ficheiros com linguagem de baixo nível ou assembly, transformando os ficheiros .c em .s. Existem compiladores capazes de gerar directamente código objecto, ou seja, instruções do processador já em código binário.

O assembler traduz código em linguagem assembly (texto) para código objecto. Pode estar integrado no compilador. O código objecto é geralmente armazenado em ficheiros com a extensão .o, em sistemas operativos unix ou .obj no ms-dos.

1.3. 3º Passo - o Linker

Esta última fase envolve a combinação do código objecto ao código das bibliotecas standard da linguagem que contém funções "built-in", como o printf. As referências a variáveis globais externas também são resolvidas pelo linker. Este último passo produz um programa executável, que por defeito se chama **a.out**.

2. Quando o Programa Começa a Ficar Muito Grande…

Começa a fazer sentido dividi-lo em ficheiros mais pequenos todos com extensão .c. O processo de compilação é em tudo idêntico ao apresentado na Figura 1, mas os dois ficheiros .o passam a ser juntos pelo Linker num único ficheiro executável **a.out**. À linha de comandos é acrescentado o novo ficheiro:

gcc exemplo1.c exemplo2.c

A criação de um único ficheiro executável a partir de dois ficheiros com código fonte implica a criação de dois ficheiros separados .o e da sua ligação durante processo de "linkagem". No caso do **gcc** o processo de "linkagem" é "escondido" mas pode-se inibir o **gcc** de criar o código objecto, usando a opção **-c**.

gcc -c exemplo1.c

gcc -c exemplo2.c

gcc exemplo1.o exemplo2.o

2.1. Separação de um Programa em Vários Ficheiros

A separação de um programa em C implica alguma atenção nos seguintes pontos:

- Não devem existir funções com o mesmo nome em ficheiros diferentes (para não confundir o compilador);
- As variáveis globais também só devem ser declaradas num ficheiro;
- Para usar funções de outro ficheiro deve ser criado um ficheiro .h com o protótipos das funções e deve ser usada a directiva #include para incluir os ficheiros .h nos ficheiros .c;
- Pelo menos um dos ficheiros tem de ter a função main ().

3. Comando make

A separação de um programa grande em vários ficheiros traz a grande vantagem de facilitar a manuseamento de ficheiros distintos mais pequenos, mas, por outro lado, implica maior perda de tempo na recompilação de todos os ficheiros. No entanto é vulgar que apenas se torne necessário alterar, com alguma frequência, uma pequena parte do programa, muitas vezes apenas uma função, porque as restantes permanecem inalteradas de início ao fim da implementação.

O comando make ajuda a gerir todo o processo de compilação, porque verifica as datas de alteração dos ficheiros e recompila apenas os ficheiros que sofreram alteração desde a última compilação.

3.1. Dependências Entre Ficheiros

Tome-se como exemplo um programa designado por **media** cujo código fonte está dividido em 2 ficheiros o **main.c** e o **media.c**.

main.c

#include <math.h>

#include <stdio.h>

```
void main(void) {
        float a=4, b=2, result;
        result = media(a, b);
        printf("Média = %f\n", result);
}
```

media.c

```
#include <math.h>
float media (float a, float b) {
        float med;
        med = (a + b) / 2;
        return med;
}
```

As dependências entre ficheiros podem ser descritas no grafo da Figura 2.

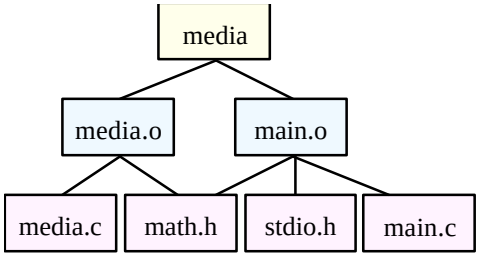

Figura 2 - Grafo de dependências

Se em alguma altura o ficheiro media.c for alterado, o ficheiro main.o também tem de ser alterado e o ficheiro executável também. Mas os restantes ficheiros presentes no grafo não precisam de sofrer nenhuma alteração.

O comando make verifica o grafo de dependências de acordo com um ficheiro de texto designado por **makefile**. O make verifica se algum ficheiro é mais recente do que outro que depende dele e compila os ficheiros de acordo com as necessidades.

3.2. Estrutura de Ficheiros - makefile

O ficheiro makefile tem a seguinte estrutura:

target *: ficheiro(s) fonte*

        comando *(é sempre precedido por um tab)*

Para o exemplo apresentado acima teríamos:

```
# makefile para o executável media

media: main.o media.o

        gcc -o media main.o media.o

main.o: main.c

        gcc -c main.c

media.o: media.c

        gcc -c media.c
```

A partir do momento que se cria a makefile a compilação do código fonte passa a resumir-se à seguinte linha de comandos

```
make
```

3.3. Macros

Numa makefile é possível usar macros, que não são mais do que variáveis. No exemplo usado até aqui pode-se alterar a makefile, usando a macro FICHEIROS para:

```
# makefile para o executável media

# agora com a macro FICHEIROS

FICHEIROS = main.o media.o

media: $(FICHEIROS)

        gcc -o media $FICHEIROS

main.o: main.c

        gcc -c main.c


media.o: media.c

        gcc -c media.c
```

Existem, ainda, algumas macros predefinidas que podem ser usadas para simplificação na construção da makefile:

- $@ Nome completo da *target* actual;

```
# makefile para o executável media

# agora com a macro $@

media: main.o media.o

        gcc -o $@ main.o media.o
```

main.o: main.c

gcc -c main.c

media.o: media.c

gcc -c media.c

- $? Lista dos ficheiros de dependência que estão desactualizados em relação à *target*:

 # *makefile para o executável media*

 # *agora com a macro $?*

media: main.o media.o

chmod +w $?

media: main.o media.o

gcc -o media main.o media.o

main.o: main.c

gcc -c main.c

media.o: media.c

gcc -c media.c

No caso, por exemplo, do ficheiro main.o estar desactualizado em relação à *target* **media** então a linha de comandos que contém o chmod passa a ser interpretada como:

chmod +w main.o

De notar que as macros $@ e $? Só podem ser usadas nas linhas de comando e nunca nas linhas que especificam dependências.

- $$@ - igual ao $@, mas pode ser usado na linha que especifica as dependências, usando o exemplo anterior:

 # *makefile para o executável media*

 # *agora com as macros $$@ e $@*

media: main.o $$@.o

gcc -o media main.o $@.o

main.o: main.c

gcc -c main.c

media.o: media.c

gcc -c media.c

- $< Ficheiro fonte da dependência actual (única):

 # makefile para o executável media

 # agora com a macro $<

media: main.o media.o

gcc -o media main.o media.o

.c .o:

gcc -c $<

Esta macro define que todos os ficheiros com extensão **.o** são construídos a partir de ficheiros com o mesmo nome, mas com extensão **.c**.

3.4. Executar Parte da makefile

O comando make também pode ser usado para executar apenas uma *target* da makefile e não o ficheiro completo como nos exemplos anteriores. Imagina-se uma nova *target* que elimine todos os ficheiros criados durante o processo de compilação e linkagem, que já não serão mais utilizados:

 # makefile para o executável media

 # agora com a operação limpeza

media: main.o media.o

gcc -o media main.o media.o

.c .o:

gcc -c $<

limpeza:

rm *.o

A última *target* pode ser executada de forma simples com a seguinte linha de comandos:

make limpeza

4. Exercício

4.1. Crie uma função ***squares.c*** que mostre no ecrã o quadrado dos números de 0 a 5. Crie uma função ***main*** num ficheiro separado de modo a invocar a primeira função. Pode usar o código abaixo:

```
void squares (void)
{
  int i;

  for (i=1; i<=5; i++)
        printf("%d\n", i*i);
}
```

```
void main (void)
{
        squares();
}
```

4.2. Crie uma ***makefile*** de modo a que, usando o comando ***make***, crie um executável chamado ***squares***. Depois de criado, o executável só deve ser alterado se um dos ficheiros squares.c ou main.c também o tiver sido.

4.3. Acrescente à makefile uma target que altere as permissões do ficheiro executável de modo a que o ficheiro executável resultante possa ser enviado por mail para o utilizador XXX.