Edição e Compilação C em UNIX

No processo de codificação de um software, são atividades básicas:

- Edição
- Compilação
- Execução
- Depuração

Além destas, também são atividades necessárias, sobretudo à medida em que os softwares desenvolvidos crescem em tamanho e complexidade:

- Construção de bibliotecas
- Gerência de versões
- Documentação

Neste módulo serão enumeradas algumas ferramentas (a maioria de código aberto -*open source*) disponíveis para auxiliar nas diversas etapas do desenvolvimento de programas C/C++ em plataformas UNIX, mais particularmente no mundo Linux.

Edição de código

Para a edição de código-fonte pode-se optar por editores de código-fonte como Emacs (ou XEmacs), Vi, Nano, Jed, Kate (no ambiente KDE), Gedit (no ambiente Gnome), Sublime Text, Atom e muitos outros. Pode-se também optar por ambientes de desenvolvimento integrado (IDEs) como Eclipse, Kdevelop, Anjuta, Code::Blocks e diversos outros.

Qual é a melhor opção ? Esta é uma pergunta difícil de responder, pois há muitas controvérsias...

Compilação

Existem vários compiladores C/C++ disponíveis livremente em ambiente Linux, mas o GCC (GNU Compiler Collection) é de longe o compilador mais popular e certamente um dos melhores. O GCC compreende compiladores de front-end e geradores de código para várias linguagens de programação (C, C++, ObjC, Fortran, Ada, Java, ...) em diversas plataformas. Além dele, sugere-se o compilador CLANG (para C/C++/ObjC), pela qualidade do código gerado e suas mensagens de erro bem mais compreensíveis que as do GCC.

O GCC é um compilador que opera em linha de comando. Por default, o compilador irá realizar a compilação e ligação do(s) fonte(s), gerando um arquivo executável denominado `a.out` (que tem esse nome por razões históricas). Para executar esse arquivo basta invocá-lo, em uma linha de comando:

hello.c

```
#include <stdio.h>

int main ()
{
  printf ("Hello, world\n") ;
  return (0) ;
}
```

```
$ gcc hello.c

$ ls -l
-rwxr-xr-x 1 prof 11724 Set 28 16:25 a.out
-rw-r--r-- 1 prof 45 Set 28 16:13 hello.c

$ a.out
Hello, world
```

Pode-se redefinir o arquivo de saída com a opção `-o`:

```
$ cc -o hello hello.c
```

É possível compilar diversos arquivos interdependentes simultaneamente, como mostra o seguinte exemplo:

hello.c

```
int main ()
{
  escreva ("Hello, world\n") ;
  return (0) ;
}
```

escreva.c

```
#include <stdio.h>

void escreva (char *msg)
{
  printf ("%s", msg) ;
}
```

```
$ cc -o hello hello.c escreva.c
```

Também é possível efetuar somente a compilação dos códigos-fonte, sem efetuar a ligação e a geração do executável. Nesse caso, serão gerados os arquivos com o código-objeto (`*.o`) dos correspondentes arquivos de código fonte (`*.c`), como mostra o exemplo abaixo:

```
$ cc -c hello.c escreva.c

$ ls -l
-rw-r--r--  1  prof     50  Set 28 16:13  escreva.c
-rw-r--r--  1  prof    788  Set 28 16:16  escreva.o
-rw-r--r--  1  prof     45  Set 28 16:13  hello.c
-rw-r--r--  1  prof    800  Set 28 16:16  hello.o
```

Esses arquivos objeto poderão posteriormente ser ligados, gerando o executável:

```
$ cc -o hello *.o

$ ls -l
-rw-r--r--  1  prof     50  Set 28 16:13  escreva.c
```

```
-rw-r--r--  1  prof    788  Set 28 16:16  escreva.o
-rwxr-xr-x  1  prof  11724  Set 28 16:25  hello
-rw-r--r--  1  prof     45  Set 28 16:13  hello.c
-rw-r--r--  1  prof    800  Set 28 16:16  hello.o
```

As opções usuais de execução do compilador GCC podem ser consultadas em sua página de manual (`man gcc`).

Opções de compilação

O compilador GCC (e os demais compiladores, em geral) aceitam uma grande quantidade de opções na linha de comando, para ativar funcionalidades específicas. As opções mais usuais são:

- `-Wall` : gera avisos sobre problemas no código que não impedem a compilação, mas podem constituir erros, como o uso de variáveis não inicializadas.
- `-Wextra` : versão ainda mais exigente do `-Wall`.
- `-Werror` : transforma todos os avisos de `-Wall` ou `-Wextra` em erros, abortando a compilação.
- `-o` : define o nome do arquivo executável.
- `-c` : faz somente a etapa de compilação, gerando o código-objeto do fonte informado.
- `-E` : faz somente a etapa de preprocessamento.
- `-S` : gera um arquivo com o código *Assembly* do programa.
- `-l` : serve para ligar o código com bibliotecas dinâmicas.
- `-D` : serve para definir macros na linha de comando (substitui o `#define`).
- `-O` : define níveis de otimização do código (ex: `-O1`)
- `-ansi` : força o uso do padrão C89/C90 (C ANSI).
- `--std=` : força o uso de um padrão específico da linguagem (ex: `--std=c99`).

Todos os programas desta disciplina devem ser compilados com `-Wall`; a ocorrência de avisos gera perda de pontos na nota.

Usando bibliotecas

A linguagem C é rica em poder de expressão, mas relativamente pobre em funcionalidades. Para construir aplicações que fazem uso de funcionalidades específicas, como interfaces gráficas, comunicação via rede, fórmulas matemáticas complexas, etc, devem ser usadas bibliotecas. Algumas bibliotecas encapsulam chamadas do sistema operacional, sendo então chamadas de *bibliotecas de sistema*, enquanto outras provêm funcionalidades no espaço de usuário, como funções matemáticas e interfaces gráficas.

As bibliotecas mais comuns, utilizadas por todas as aplicações e utilitários do sistema, são:

- `libc`: na verdade um grande pacote de bibliotecas que provê funcionalidades básicas de entrada/saída, de acesso a serviços do sistema, à rede, etc.
- `ld-linux`: provê as funções necessárias para a carga de bibliotecas dinâmicas, durante a inicialização do programa.

Por default, essas duas bibliotecas são automaticamente incluídas e ligadas em todos os programas. Para compilar um programa que utiliza outras bibliotecas externas, algumas opções devem ser informadas ao compilador. Por exemplo, considere o seguinte programa, que faz o cálculo de Pi através de uma série de Gregory:

pi.c

```
#include <stdio.h>
#include <math.h>

#define TERMOS 1000000

int main ()
{
  int i ;
  double pi = 0 ;

  for (i=0; i < TERMOS; i++)
    pi += pow (-1,i) / (2*i+1) ;
  pi *= 4 ;

  printf ("O valor aproximado de Pi é: %f\n", pi) ;
  return (0) ;
}
```

Ao compilar esse programa, obtemos:

```
$ cc pi.c -o pi
/tmp/ccCANYTf.o(.text+0x42): In function `main':
: undefined reference to `pow'
collect2: ld returned 1 exit status
```

Esse erro ocorre porque o ligador não conseguiu encontrar a função pow em nenhum dos arquivos fonte ou objeto informados no comando, nem nas bibliotecas padrão. Essa função se encontra na biblioteca matemática `/usr/lib/libm`, que deve ser informada ao ligador da seguinte forma:

```
$ cc pi.c -o pi -lm
```

A opção `-lm` indica o uso da biblioteca `libm`. Da mesma forma, `-lpthread` indica o uso da biblioteca `libpthread`, e assim por diante. O ligador irá procurar por arquivos de bibliotecas nos diretórios padrão (`/lib`, `/usr/lib`, ...). Pode-se informar outros diretórios de bibliotecas ao ligador através da opção `-L` (detalhada mais adiante neste texto).

Ligação estática e dinâmica

Há duas formas básicas de ligar as bibliotecas a um programa executável:

- Na **ligação estática** (*static linking*), o código da biblioteca é incorporado ao executável. O executável fica maior, mas não depende de bibliotecas instaladas no sistema para poder executar.
- Na **ligação dinâmica** (*dynamic linking*), o executável guarda apenas referências às bibliotecas necessárias, que são resolvidas somente quando o programa executável for lançado. O executável fica muito menor, mas precisa que todas as bibliotecas necessárias estejam presentes no sistema para executar.

A ligação dinâmica é feita por default. Para compilar um programa, ligando-o estaticamente à bibliotecas, devemos executar:

```
$ cc -static pi.c -o pi -lm
```

O utilitário `ldd` permite verificar de quais bibliotecas dinâmicas um executável depende:

```
$ ldd pi
    libm.so.6 => /lib/i686/libm.so.6 (0x40028000)
    libc.so.6 => /lib/i686/libc.so.6 (0x4004b000)
    /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

Para encontrar as bibliotecas dinâmicas, são percorridos os diretórios indicados pelo arquivo de configuração `/etc/ld.so.conf` e pela variável de ambiente `LD_LIBRARY_PATH`. O utilitário `ldconfig` permite atualizar as informações sobre bibliotecas dinâmicas nos diretórios padrão (ou nos diretórios informados via linha de comando).

Documentação

O sistema UNIX implementa um sistema de documentação *on-line* simples, mas bastante útil e eficiente, chamado **páginas de manual** (*man pages*). As páginas de manual estão estruturadas em sessões:

- Sessão 1: Comandos do usuário.
- Sessão 2: Chamadas ao sistema
- Sessão 3: Bibliotecas e funções standard
- Sessão 4: Descrição de dispositivos e formatos de arquivos de dados
- Sessão 5: Formato de arquivos de configuração
- Sessão 6: Jogos
- Sessão 7: Diversos
- Sessão 8: Comandos de administração do sistema

O acesso às páginas de manual é normalmente efetuado através do comando `man`. Assim, `man ls` apresenta a página de manual do comando `ls`, enquanto `man man` apresenta a página de manual do próprio comando `man`. Ambientes gráficos normalmente oferecem ferramentas mais versáteis para consulta às páginas de manual.

Além do comando `man`, outros comandos são úteis para a busca de informações no sistema:

- `whatis`, `apropos` : para localizar páginas de manual que contenham informações sobre algum assunto específico.
- `locate` : para localizar arquivos no sistema
- `which` : para informar onde se encontra um determinado executável
- `info` : sistema de informações mais completo que o `man`, mas não disponível em todas as plataformas UNIX.

From:
https://wiki.inf.ufpr.br/maziero/ - **Prof. Carlos Maziero**

Permanent link:
https://wiki.inf.ufpr.br/maziero/doku.php?id=c:edicao_e_compilacao_c_em_unix

Last update: **2023/08/01 17:04**