CUPRINS

Partea a doua

PROGRAMAREA ÎN LIMBAJUL C++

Partea a treia

UNELTE DE PROGRAMARE PENTRU LINUX

Anexe

Capitolul 2

COMPILATOARE

Eu am o regulă simplă în viață:
dacă nu înțeleg ceva, înseamnă că este rău.
(Linus Torvalds)

Compilarea programelor C este formată din trei faze independente, realizate de obicei de trei programe distincte:

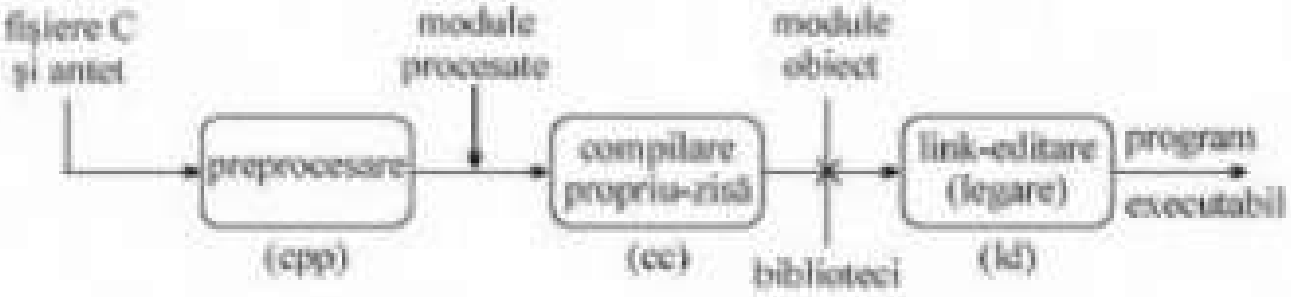

Figura 2.1. Fazele compilării

Utilizând anumite opțiuni (vezi paragraful următor), compilatorul poate fi determinat să parcurgă doar anumite faze.

Preprocesarea

Preprocesorul primește un text și generează tot un text. El analizează macrourile (care încep cu caracterul „#", vezi partea I, capitolul 8 pentru detalii) și le expandează. Astfel, rezultatul preprocesării este tot un program-sursă C/C++, dar care nu mai include directive preprocesor, ci rezultatul acestora (de exemplu, o directivă `#include` va avea ca efect includerea efectivă a codului C/C++ conținut în fișierul-antet corespunzător). Faza de preprocesare poate fi apelată prin:

```
gcc -E main.c > main.i
```

Compilarea

Rezultatul compilării este un *fișier-obiect* care reprezintă un amestec de cod mașină și alte informații necesare în faza de editare de legături (link-editare).

Fișierul-obiect conține în primul rând o listă a simbolurilor nesatisfăcute (adică declarate dar nedefinite), precum și locurile în care acestea sunt utilizate. Aceste simboluri sunt externe (definite în alte module). De asemenea, fișierul-obiect conține și o listă a simbolurilor exportate de către modul.

Fișierul-obiect mai conține și *informații de relocare*, acestea fiind necesare pentru că toate salturile se fac la niște adrese care se vor modifica în faza de editare de legături.

Nu în cele din urmă, obiectele pot conține și informații necesare pentru depanarea programului rezultat (vezi subcapitolul 3.1). Faza de compilare poate fi apelată cu:

```
gcc -c main.c
```

Rezultatul va fi un fișier-obiect cu numele `main.o`.

Editarea de legături

Legarea modulelor constă în completarea referințelor obiectelor globale, relocarea codului și adunarea tuturor modulelor într-un singur fișier executabil. Editarea de legături se face cu ajutorul programului `ld`, dar care nu trebuie apelat separat (lista fișierelor-obiect este foarte mare, fiindcă poate include numeroase biblioteci).

Bibliotecile C

O *bibliotecă* este o colecție de module-obiect deja compilate, strânse la un loc într-un *fișier-arhivă*. În general, declarațiile globale (variabile, constante, funcții etc.) se găsesc într-un fișier-antet furnizat împreună cu fișierul-arhivă. Crearea bibliotecilor se face cu ajutorul programului `ar` (pentru detalii vezi subcapitolul 2.3).

Există posibilitatea de a crea biblioteci speciale, denumite *biblioteci partajate* (*shared libraries*). Există un număr mare de funcții, în special în bibliotecile standard, foarte des utilizate în majoritatea programelor. Funcțiile trebuie încărcate în memorie o singură dată de către sistemul de operare, pentru a nu lega codul lor la toate executabilele, ceea ce ar duce la mărirea dimensiunii acestora. Ca rezultate pozitive, putem enumera: reducerea drastică a dimensiunii executabilelor, micșorarea timpului de încărcare a acestora și reducerea consumului de memorie.

În Linux, bibliotecile partajate sunt stocate în directoarele `/lib` și `/usr/lib`.

2.1. Compilarea programelor C: *GCC*

Cel mai răspândit compilator din lumea UNIX este GCC, implementare *open source* realizată de către Free Software Foundation. Acesta este un compilator extensibil, existând extensii pentru C++, Objective-C, Pascal, Fortran etc. Executabilele generate de către acest compilator nu trebuie neapărat să fie *open source*, chiar dacă includ librăriile standard C sau C++.

Prima versiune a compilatorului `GCC` a fost scrisă de către Richard Stallman, inițiatorul proiectului GNU, prin 1986.

Sintaxa generală de apelare a compilatorului este:

```
gcc [ opțiuni nume_fișier ]
```

Tipul fișierelor de intrare este determinat după sufixul acestora, care sunt prelucrate după cum urmează:

- `.c` – sursă C: preprocesare, compilare, asamblare;
- `.C` – sursă C++: preprocesare, compilare, asamblare;
- `.cc` – sursă C++: preprocesare, compilare, asamblare;
- `.cxx` – sursă C++: preprocesare, compilare, asamblare;
- `.m` – sursă Objective-C: preprocesare, compilare, asamblare;
- `.i` – sursă C preprocesată: compilare, asamblare;
- `.ii` – sursă C++ preprocesată: compilare, asamblare;
- `.s` – sursă în limbaj de asamblare: asamblare;
- `.S` – sursă în limbaj de asamblare: preprocesare, asamblare.

Fișierele având alte sufixe, cum ar fi `.o` (fișier-obiect) sau `.a` (fișier-arhivă), sunt trimise către link-editor.

Pentru a se evita o parte din etapele prelucrării pot fi utilizate următoarele opțiuni:

- `-c` compilează și asamblează fișierele-sursă, dar nu le link-editează;
- `-S` compilează fișierele-sursă, fără însă a le asambla;
- `-E` nu lansează compilarea, ci doar preprocesarea.

Implicit, `gcc` generează un fișier executabil având denumirea `a.aout`; fișierele-obiect corespunzătoare intrării *`sursa.sufix`* sunt denumite *`sursa`*`.o`, iar fișierele asamblate *`sursa`*`.s`.

Denumirea ieșirii poate fi schimbată cu ajutorul opțiunii `-o` *`fisier`*.

Spre exemplu, pentru compilarea unui program C:

```
gcc -o cap7_1 cap7_1.c
```

Alte opțiuni des utilizate:

Opțiuni la compilare:

- `-I`*`director`* adaugă *director* la lista directoarelor în care sunt căutate fișierele `.h`;
- `-L`*`director`* adaugă *director* la lista directoarelor în care sunt căutate bibliotecile;
- `-g` include informații de depanare în fișierul-obiect generat. Această opțiune este necesară dacă se intenționează depanarea programului cu `GDB`, de exemplu;

- `-O`, `-O1`, `-O2`, `-O3` activează diverse niveluri de optimizare a codului. `-O3` este nivelul cel mai avansat de optimizare;
- `-O0` dezactivează optimizarea codului.

Opțiuni la editarea de legături:

- `-lbiblioteca` utilizează biblioteca specificată în cadrul etapei de editare de legături.

Opțiuni la preprocesare:

- `-Dmacro` definește macroul specificat ca fiind `1`;
- `-Dmacro=valoare` definește macroul cu valoarea specificată;
- `-Umacro` anulează macroul specificat.

Pentru exemplificare, compilarea unui program care utilizează biblioteca `ncurses`:

```
gcc -o test test.c -I/usr/include/ncurses -lncurses
```

2.2. Compilarea programelor C++: *G++*

Compilatorul G++ se utilizează pentru compilarea programelor C++. Sintaxa de utilizare este identică cu cea a GCC.

Opțiunile mai des utilizate sunt:

- `-fall-virtual` tratează toate funcțiile membre ca fiind virtuale;
- `-fthis-is-variable` permite utilizarea pointer-ului `this` ca o variabilă obișnuită;
- `-fexternal-templates` produce cod obiect mai mic pentru declarațiile de template-uri, generând doar o singură copie a fiecărei funcții template acolo unde aceasta este definită. Pentru a putea utiliza această opțiune, trebuie marcate toate fișierele-sursă care utilizează template-uri cu directiva `#pragma implementation` (acolo unde sunt definite efectiv template-urile), respectiv `#pragma interface` (acolo unde sunt declarate template-urile). Atunci când este utilizată această opțiune, toate instanțierile de template-uri sunt considerate `external`. De aceea, toate aceste instanțieri trebuie să fie realizate în cadrul fișierului marcat cu `implementation`, de exemplu prin utilizarea unor declarații `typedef` care să facă referire la fiecare instanțiere;

- `-falt-external-templates` are un comportament similar cu cel al opțiunii precedente, cu excepția faptului că este generată o singură copie a fiecărei funcții template acolo unde aceasta este definită pentru prima oară.

2.3. Crearea de biblioteci: *ar*

Utilitarul `ar` se folosește pentru a crea, modifica și extrage fișiere dintr-o arhivă, în speță bibliotecă de tip static.

Pentru a crea o bibliotecă statică, trebuie mai întâi generate fișierele-obiect, apoi creată arhiva:

```
ar -r libtest.a functii.o test.o
ranlib libtest.a
```

Prima comandă va crea arhiva `libtest.a`, conținând fișierele-obiect `functii.o` și `test.o`. Cea de-a doua comandă generează un index pentru arhiva nou creată.

Sintaxa programului `ar` este următoarea:

```
ar comanda arhiva [membri...]
```

unde *comandă* poate fi:

- `d` șterge module din arhivă;
- `p` afișează membrii specificați (sau toți membrii dacă nu este specificat nici unul) din arhivă;
- `r` inserează un nou membru în arhivă;
- `t` afișează conținutul unei arhive;
- `x` extrage un membru din arhivă.

Pentru a crea biblioteci partajate, se va folosi compilatorul `gcc` (sau `g++` pentru programe C++):

```
gcc -shared -o libtest.so functii.o test.o
```

Bibliotecile partajate trebuie instalate în genere în directorul `/usr/lib`, apoi trebuie executată comanda `ldconfig` pentru a actualiza baza de date de biblioteci (*cache*).