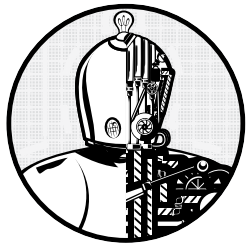
15

RAZVOJNE ALATKE



Linux i Unix su veoma popularni među programerima, ne samo zbog neverovatno bogatog izbora alatki i okruženja na raspolaganju, nego i zato što je sistem izuzetno dobro dokumentovan i transparentan. Na Linux mašini, ne morate biti programer da biste koristili razvojne alatke, ali kada radite na sistemu, trebalo bi da znate nešto o alatkama za programiranje zato što one igraju značajniju ulogu u upravljanju Unix sistemima nego u drugim operativnim sistemima. Potrebno je da barem umete da prepoznate razvojne alatke i steknete određeno znanje o njihovoj upotrebi.

Ovo poglavlje pakuje veliku količinu informacija na malom prostoru, ali nije neophodno da savladate baš sve navedeno. Pojedine delove gradiva lako možete preskočiti i vratiti se na njih kasnije. Odeljak o deljenim bibliotekama verovatno sadrži ono najvažnije što treba da znate. Ali da biste znali odakle potiču deljene biblioteke, prvo ćete se upoznati sa osnovama izrade programa.

15.1 Kompajler jezika C

Ako znate kako se koristi kompajler (prevodilac) za programski jezik C, imaćete prilično dobar uvid u poreklo programa koje vidite na svom Linux sistemu. Izvorni kôd većine alatki za Linux i mnogih aplikacija na Linux sistemima, napisan je na jeziku C ili C++. U ovom poglavlju koristićemo prevashodno primere na jeziku C, ali informacije do kojih ćete doći možete preneti i na C++.

C programi slede tradicionalni postupak razvoja: pišete programe, prevedete ih, a zatim izvršite, odnosno kada napišete C program i želite da ga izvršite, izvorni kôd koji ste napisali morate *prevesti* (engl. *compile*) u binarni oblik niskog nivoa koji rač unar razume. S druge strane, kada koristite jezik za skriptove – koji razmatramo u nastavku poglavlja – ne treba da prevodite ništa.

NAPOMENA *Većina distribucija standardno ne sadrži alatke koje su potrebne za prevođenje C koda zato što te alatke zauzimaju prilično mnogo mesta. Ako ne možete da nađete neke od alatki koje ovo poglavlje opisuje, možete instalirati paket build-essential za Debian/Ubuntu, ili sličan paket za Fedoru/CentOS pomoću komande* `yum groupinstall "Development Tools"`*. Ako to ne uspe, potražite na internetu odgovarajući paket pod odredniciom „C compiler“.*

Izvršiva komponenta kompajlera za jezik C u većini Unix sistema jeste GNU C kompajler, `gcc`, mada noviji prevodilac `clang` iz projekta LLVM stiče sve veću popularnost. Imena datoteka izvornog C koda završavaju se sa *.c*. Razmotrite samostalnu datoteku izvornog C koda čije je ime *hello.c*, a koju možete naći u knjizi *The C Programming Language*, 2. izdanje, čiji su autori Brian W. Kernighan i Dennis M. Ritchie (Prentice Hall, 1988):

```
#include <stdio.h>

main() {
    printf("Hello, World.\n");
}
```

Upišite taj izvorni kôd u datoteku koju ćete nazvati *hello.c* a zatim izvršite sledeću komandu:

```
$ cc hello.c
```

Rezultat je izvršiva datoteka *a.out*, koju možete izvršavati na isti način kao i svaku drugu izvršivu datoteku u sistemu. Međutim, verovatno bi izvršivoj datoteci trebalo nadenuti neko drugo ime (kao što je *hello*). Da biste to postigli, zadajte opciju kompajlera `-o`:

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

Za veoma kratke programe, to je otprilike sve što treba uraditi po pitanju prevođenja koda. Može biti potrebno da dodate još neki direktorijum ili biblioteku za umetanje (videti odeljke 15.1.2 i 15.1.3), ali sada ćemo razmotriti nešto opsežnije programe pre nego što počnemo da se bavimo tim temama.

15.1.1 Kombinovanje više izvornih datoteka

Većina C programa je prevelika da bi mogli u celini stati u jednu datoteku izvornog koda. Datoteke mamutske veličine postaju suviše nepraktične za programera, a kompajleri ponekad čak imaju teškoća da analiziranju preopsežne datoteke. Iz tih razloga, projektanti grupišu više komponeneta izvornog koda i svaki njegov deo smeštaju u vlastitu datoteku.

Kada prevodite većinu *.c* datoteka, ne pravite odmah izvršivu datoteku. Umesto toga, upotrebite opciju `-c` kompajlera da biste napravili *objektne datoteke*. Da biste videli kako se to radi, recimo da imate dve datoteke, *main.c* i *aux.c*. Sledeće dve komande kompajlera obavljaju najveći deo sklapanja (engl. *building*) programa:

```
$ cc -c main.c
$ cc -c aux.c
```

Prethodne dve komande prevode dve izvorne datoteke u dve objektne datoteke *main.o* i *aux.o*.

Objektna datoteka je binarna datoteka koju procesor gotovo da razume, osim što i dalje postoji nekoliko nedorečenosti. Prvo, operativni sistem ne zna kako da izvrši objektnu datoteku, a drugo, verovatno ćete morati da kombinujete više objektnih datoteka i određene sistemske biblioteke kako biste sklopili kompletan (i upotrebljiv) program.

Da biste od jedne ili više objektnih datoteka sklopili jednu izvršivu datoteku, morate izvršiti *linker* (povezivač), što je u Unixu komanda `ld`. Programeri retko koriste `ld` s komandne linije, zato što sam C kompajler zna kako da pozove program za povezivanje (linker). Da biste od dve navedene objektne datoteke formirali izvršiv program koji se zove *myprog*, izvršite sledeću komandu kako biste ih povezali:

```
$ cc -o myprog main.o aux.o
```

Mada više izvornih datoteka možete prevesti i ručno, kao što prikazuje prethodni primer, može biti teško da ih sve pratite tokom postupka prevođenja kada se broj izvornih datoteka poveća. Sistem `make` opisan u odeljku 15.2 tradicionalni je Unixov standard za upravljanje postupkom prevođenja koda. Taj sistem je naročito važan za upravljanje datotekama koje su opisane u naredba dva odeljka.

15.1.2 Datoteke zaglavlja i direktorijumi za umetanje

C *datoteke zaglavlja* (engl. *header files*) dodatne su datoteke sa izvornim kodom koje obično sadrže deklaracije bibliotečkih tipova funkcija. Na primer, *stdio.h* je datoteka zaglavlja (videti jednostavan primer u odeljku 15.1).

Nažalost, veliki broj problema pri prevođenju koda potiče od datoteka zaglavlja. Većina grešaka nastaje kada kompajler ne uspe da nađe datoteke zaglavlja i biblioteke. Ponekad čak i programer zaboravi da umetne odgovarajuću datoteku zaglavlja, usled čega izvorni kôd ne može da se prevede.

Rešavanje problema s datotekama za umetanje

Određivanje ispravnih datoteka za umetanje nije uvek jednostavno. Ponekad je navedeno više datoteka za umetanje sa istim imenima ali iz različitih direktorijuma, a nije jasno koja je ispravna. Kada kompajler ne uspeva da pronađe potrebnu datoteku za umetanje, poruka o grešci izgleda ovako:

```
badinclude.c:1:22: fatal error: notfound.h: No such file or directory
```

Ova poruka vas obaveštava da kompajler ne može da nađe datoteku zaglavlja *notfound.h* koju datoteka *badinclude.c* referencira. Ta konkretna greška je direktan rezultat sledeće direktive u prvom redu datoteke *badinclude.c*:

```
#include <notfound.h>
```

Podrazumevani direktorijum za datoteke za umetanje na Unixu jeste */usr/include*; kompajler ih uvek tamo traži, osim ako mu izričito zadate da to ne čini. Međutim, kompajleru možete zadati i da potraži u drugim direktorijumima za umetanje (većina putanja koje sadrže datoteke zaglavlja imaju reč *include* negde u imenu).

NAPOMENA *U poglavlju 16 saznaćete više o tome kako se pronalaze nedostajuće datoteke za umetanje.*

Na primer, našli ste datoteku *notfound.h* u direktorijumu */usr/junk/include*. Kompajleru možete naložiti da traži u tom direktorijumu pomoću opcije `-I`:

```
$ cc -c -I/usr/junk/include badinclude.c
```

Sada kompajler više ne bi trebalo da se „sapliće“ o red u datoteci *badinclude.c* koji referencira datoteku zaglavlja.

Trebalo bi da obratite pažnju i na datoteke za umetanje navedene između navodnika (`" "`) umesto između šiljatih zagrada (`< >`), kao u sledećem primeru:

```
#include "myheader.h"
```

Navodnici znače da se datoteka za umetanje ne nalazi u sistemskom direktorijumu za umetanje nego da kompajler treba da je potraži u svom direktorijumu za umetanje. To često znači da se tražena datoteka za umetanje nalazi u istom direktorijumu kao datoteka izvornog koda. Ako naiđete na problem zbog navodnika, verovatno pokušavate da prevedete nekompletan izvorni kôd.

Šta je C pretprocesor (cpp)?

Posao traženja svih datoteka za umetanje zapravo ne obavlja C kompajler. Time se bavi *C pretprocesor*, što je program koji kompajler poziva da obradi vaš

izvorni kôd pre nego što kompajler analizira vaš program. Pretprocesor konvertuje izvorni kôd u oblik koji kompajler razume; to je alatka koja izvorni kôd čini lakšim za čitanje (i umeće u njega prečice).

Pretprocesorove komande u izvornom kodu zovu se *direktive* i počinju znakom `#`. Postoje tri osnovne vrste direktiva:

Datoteke za umetanje Direktiva `#include` nalaže pretprocesoru da umetne celu navedenu datoteku. Imajte u vidu da opcija `-I` kompajlera čini da pretprocesor traži datoteke za umetanje u zadatom direktorijumu, kao što ste videli u prethodnom odeljku.

Definicije makroa Red nalik na `#define BLABLA nešto` nalaže pretprocesoru da gde god u izvornom kodu naiđe na `BLABLA`, to treba da zameni sa `nešto`. Konvencije nalažu da se makroi pišu velikim slovima, ali nemojte se iznenaditi kada vidite da programeri ponekad koriste makroe čija imena podsećaju na funkcije ili promenljive. (S vremena na vreme, to je uzrok teških glavobolja. Mnogi programeri čak uživaju u zloupotrebi pretprocesora.)

NAPOMENA *Umesto da makroe definišete unutar izvornog koda, možete ih definisati i tako što ih prosledite kompajleru u obliku parametara: `-DBLABLA=nešto` ima isti efekat kao prehodna direktiva.*

Uslovno prevođenje Određene delove koda možete obeležiti sa `#ifdef`, `#if` i `#endif`. Direktiva `#ifdef` *`MAKRO`* ispituje da li je definisan pretprocesorov *`MAKRO`*, a direktiva `#if` *`uslov`* ispituje da li je *`uslov`* ispunjen. U obe direktive, ako uslov koji sledi „naredbi if“ nije ispunjen, pretprocesor ne prosleđuje kompajleru ceo tekst u opsegu od `#if` do sledećeg `#endif`. Ako nameravate da proučavate C kôd, počnite da se navikavate na to.

Sledi primer uslovne direktive. Kada pretprocesor vidi sledeći kôd, ispituje da li je definisan makro `DEBUG`, pa ako jeste, kompajleru prosleđuje red koji sadrži funkciju `fprintf()`. Ako nije, pretprocesor preskače taj red i nastavlja da obrađuje datoteku iza `#endif`:

```
#ifdef DEBUG
  fprintf(stderr, "Ovo je dijagnostička poruka.\n");
#endif
```

NAPOMENA *C pretprocesor ne zna ništa o sintaksi jezika C, promenljivama, funkcijama i drugim elementima jezika. On razume samo svoje makroe i direktive.*

Na Unixu, ime C pretprocesora je `cpp`, ali ga možete pokrenuti i pomoću komande `gcc -E`. Međutim, retko je potrebno da pretprocesor pokrećete kao samostalan program.

15.1.3 Povezivanje s bibliotekama

Kompajler za jezik C ne poznaje vaš sistem dovoljno da bi mogao sam da sklopi neki koristan program. Potrebne su vam *biblioteke* da biste sastavljali kompletne programe. C biblioteka je kolekcija zajedničkih unapred prevedenih

funkcija koju možete ugraditi u svoj program. Na primer, mnogi izvršivi programi koriste biblioteku math zato što ona sadrži trigonometrijske i druge matematičke funkcije.

Biblioteke dolaze do izražaja prvenstveno u vreme povezivanja koda, kada program za povezivanje (linker) formira od objektnih datoteka jednu izvršivu datoteku. Na primer, ako imate program koji koristi biblioteku gobject ali vi zaboravite da obavestite kompajler da poveže tu biblioteku, videćete greške pri povezivanju kao što je sledeća:

```
badobject.o(.text+0x28): undefined reference to 'g_object_new'
```

Najvažniji delovi tih poruka o greškama istaknuti su podebljanim slovima. Kada je program za povezivanje analizirao objektnu datoteku *badobject.o*, nije uspeo da nađe funkciju čije je ime istaknuto, a posledica toga je da nije mogao da sklopi izvršivu datoteku. U ovom konkretnom slučaju, mogli biste posumnjati da ste zaboravili biblioteku gobject zato što je ime nedostajuće funkcije `g_object_new()`.

NAPOMENA *Nedefinisane reference ne znače uvek da vam nedostaje određena biblioteka. Moguće je i da neka od objektnih datoteka programa nije navedena u komandi za povezivanje programa. Mada, lako je razlikovati funkcije iz biblioteka i funkcije iz objektnih datoteka.*

Da biste rešili taj problem, morate prvo pronaći biblioteku gobject i zatim zadati opciju kompajlera `-l` da biste svoj program povezali s tom bibliotekom. Isto kao i datoteke za umetanje, biblioteke su rasute po celom sistemu (*/usr/lib* je sistemska podrazumevana lokacija za biblioteke), mada se mnoge biblioteke nalaze u poddirektorijumu *lib*. Za prethodni primer, osnovna datoteka biblioteke gobject jeste *libgobject.a*, a ime biblioteke je `gobject`. Znajući to, svoj program biste povezali s tom bibliotekom na sledeći način:

```
$ cc -o badobject badobject.o -lgobject
```

Program za povezivanje (linker) morate obavestiti o bibliotekama na nestandardnim lokacijama; parametar za tu namenu je `-L`. Recimo da je programu `badobject` potrebna biblioteka *libcrud.a* koja se nalazi u */usr/junk/lib*. Da biste preveli kôd i napravili izvršivu datoteku, zadajte komandu sličnu sledećoj:

```
$ cc -o badobject badobject.o -lgobject -L/usr/junk/lib -lcrud
```

NAPOMENA *Ako želite da pronađete biblioteku u kojoj se nalazi data funkcija, upotrebite komandu* `nm`*. Očekujte veliku količinu podataka. Na primer, pokušajte sledeće:* ***`nm libgobject.a`****. (Možda će vam trebati komanda* `locate` *da biste pronašli biblioteku* libgobject.a*; mnoge distribucije danas smeštaju biblioteke u poddirektorijume* /usr/lib *koji su specifični za određene arhitekture.)*