Projektovanje elektronskih sistema

Predavanje 8

Doc.dr Borisav Jovanović

preuzeto iz predavanja prof. Milunke Damnjanovic i prof. Miluna Jevtica

Sadržaj:

- Izvorni (source) C i C++ fajlovi
- Proces kompajliranja
- Kompajliranje C i C++ izvornih fajlova
- Opcije gcc i g++
- Povezivanje/linkovanje
- Automatizacija procesa kompajliranja,
  GNU Make
- Pokretanje GDB za debagovanje/otklanjanje grešaka

- Programski jezik C razvijen je davne 1973. godine i ostaje jedan od najpopularniijh programskih jezika u 2020. godini.

- Samo da bismo vam dali ideju, evo nekoliko sistema programiranih na jeziku C: Microsoft Windows, Linux, Mac, iOS, Android, Windows Phone kernels, Oracle Database, MySQL, MS SQL Server

- C ++ je vrlo blizak jeziku C.

- Koristi objektno orijentisano programiranje i kombinuje ga sa sintaksom i snagom jezika C.

Izvorni (source) C i C++ fajlovi

- Imena izvornih C fajlova se završavaju sa .c ili .h.
- Ako želimo da kreiramo novi izvorni C++ fajl, ime datoteke treba se završava sa .cpp ili .hpp.
- Kompajler je program koji pretvara izvorni (source) kod u mašinski kod koji mašina/računar može da razume i izvrši.
- Originalni GNU C Compiler (GCC) je razvijen od strane Ričarda Stolmana, osnivača GNU projekta, koji se pored Linusa Torvaldsa, smatra začetnikom Linux OS.
- 1984. godine GNU projekat je kreiran kako bi omogućio Unix-like OS kao free software, u cilju promovisanja slobode programiranja.

Svi kompajleri na Linux sistemima su deo GNU-a Compiler Collection, koji je poznat kao GCC.

GCC sadrži kompajlere za C, C ++, Java, Objective-C, Fortran itd.

GNU Toolchain sadrži:

- GCC kompajler (osnovna komponenta)
- GNU Make (automatizovan alat za kompajliranje i build-ovanje aplikacija
- GNU Binutils (binarni alati uključujući linker i asembler)
- GNU Debugger (GDB)
- GNU Autotools
- GNU Bison

Pretpostavimo da imamo projekat koji sadrži dve izvorne datoteke:

- C izvornu datoteku **main.c** (Listing 1)
- izvornu C++ datoteku **reciprocal.cpp** (Listing 2)

Ove dve datoteke treba da se kompajliraju i povežu (linkuju) zajedno da bi se dobio **izvršni program** koji se zove **reciprocal**.

Ovaj program izračunava recipročnu vrednost nekog celog broja.

```
// Listing 1 (main.c)
// C source file — main.c
#include <stdio.h>
#include <stdlib.h>
#include “reciprocal.hpp”
int main (int argc, char **argv)
{
   int i;
   i = atoi (argv[1]);
   printf (“The reciprocal of %d is %g\n”,i,reciprocal(i));
   return 0;
}
```

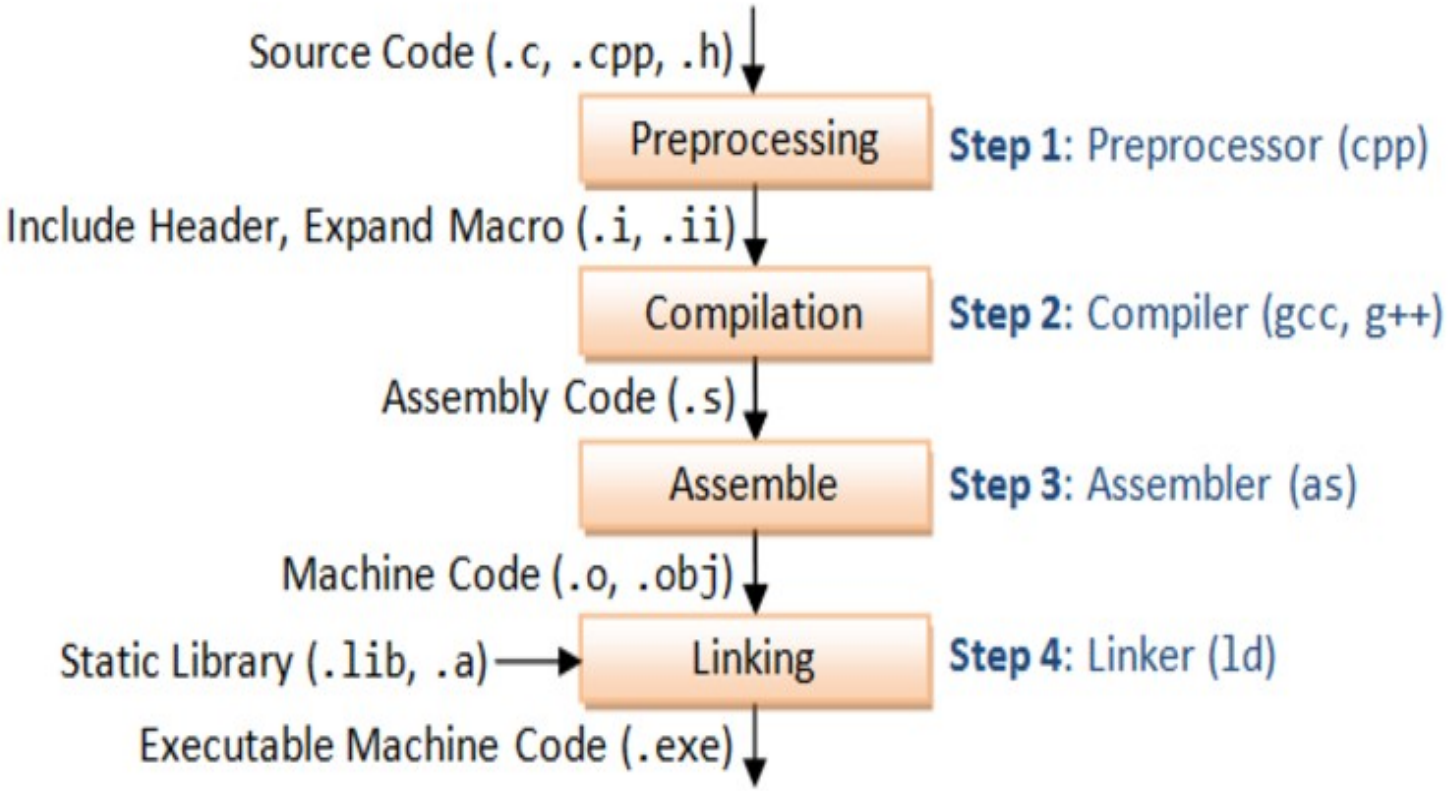
```
// Listing 2 (reciprocal.cpp)
// C++ source file — reciprocal.cpp

#include <cassert>
#include “reciprocal.hpp”
double reciprocal (int i) {
  // I should be non-zero.
  assert (i != 0);
  return 1.0/i;
}
```

```
// Listing 3 (reciprocal.hpp)
// Header file — reciprocal.hpp
#ifdef __cplusplus
extern “C” {
#endif
extern double reciprocal (int i);
#ifdef __cplusplus
}
#endif
```

- U operativnom sistemu Windows, izvršni fajlovi (programi) se obično završavaju sa .exe.
- Linux programi, na sa druge strane, obično nemaju ekstenziju /produžetak.
- Dakle, Windows ekvivalent ovog programa bi verovatno bi se zvao r**eciprocal.exe;** Linux verzija je jednostavno r**eciprocal.**
- Pored dva izvorna fajla u projektu postoji još jedna *header* datoteka koja se zove r**eciprocal.hpp** (Listing 3).

Proces kompajliranja



Kompajliranje C i C++ izvornih fajlova

- Naziv C kompajlera Linux-a je **gcc**.
- **gcc** je portabilan i može se izvršavati na mnogim operativnim sistemima.
- Predstavlja prirodno okruženje za Linux i Unix-like OS. Na Windows operativnim sistemima MinGW i Cygwin.
- Da biste kompajlirali neku C izvornu datoteku, koristi se opcija *-c*.
- Na primer, unosom sledeće komande kompajlira se datoteka **main.c.** Kao rezultat, dobija se objektni fajl **main.o**

%gcc -c main.c.

- Naziv C++ kompajlera u Linux-u je **g++**.
- Kompajliranje izvornog C++ fajla **reciprocal.cpp** se postiže:

%g++ -c reciprocal.cpp

- Opcija -c govori kompajleru da program treba da se iskompajlira samo u objektnu datoteku; bez ove opcije toga **g++** će pokušati da linkuje/poveže program kako bi stvorio krajnji izvršni program.
- Nakon izvršavanja kompajlera **g++** dobija se objektni fajl koji se zove **reciprocal.o**.
- Za kompajliranje bilo kog razumno velikog C i C++ programa potrebno je znati opcije gcc i g++ programa.
- Recimo, opcija -I se koristi da bi se **gcc**-u reklo gde da traži *header* (.h) datoteke.
- Za svaki .h fajl, korišćen od strane .c fajlova (korišćenjem #include direktive) kompajler traži *include* putanju. Ove putanje se specificiraju korišćenjem –I opcije ili *environment* varijable CPATH.
- Ako se ne kaže drugačije, kompajler pretražuje samo trenutni direktorijum i podrazumevane standardne direktorijume (gde se standardno nalaze header fajlovi):
  - /usr/include
  - /usr/include/sys.

- Dakle, ako želimo da dodamo *header* datoteke (iz nekog drugog a ne trenutnog direktorijuma) potrebna je **opcija -I**.
- Na primer, pretpostavimo da projekat ima poseban direktorijum za smeštanje izvornih datoteka (.c i .cpp) koji se zove **source** i drugi folder koji se zove **include** za smeštanje *header* (.h) fajlova.
- Za kompajliranje **reciprocal.cpp** treba da se ukaže kompajleru da koristi direktorijum **../include** za pronalaženje foldera u kome je **reciprocal.hpp**:

% g++ -c reciprocal.cpp -I ../include/

% gcc -c main.c -I ../include/

Opcije gcc i g++

- Ponekad je potrebno da se definišu **makronaredbe** u komandnoj liniji koje se izvršavaju sa gcc i g++.
- Na primer, ako se NE ŽELI da se izvršavaju naredbe ASSERT u reciprocal.cpp, koje su korisne pri otklanjanju grešaka ove se provere isključuju definisanjem makronaredbe NDEBUG u komandnojh liniji.
- Postoji i drugi način za preskakanje ASSERT naredbi, a to je preko eksplicitne naredbe #define u reciprocal.cpp, ali to zahteva promenu samog izvornog programa.
- Lakše je ako se definiše NDEBUG u komandnoj liniji:
- **g++ -c -D NDEBUG reciprocal.cpp -I ../include/**

- Ako hoćemo da kompajler optimizuje kod da se on izvršava što je moguće brže, to postiže pomoću opcije -O2 u komandnoj liniji.
- **gcc** ima nekoliko različitih nivoa optimizacije; drugi nivo je prikladan za većinu programa.
- Na primer, sledeća naredba kompajlira izvorni fajl **reciprocal.cpp** sa uključenom optimizacijom za brzinu izvršavanja:

- **% g++ -c -O2 reciprocal.cpp -I ../include/**

- GCC opcija **-Wall** omogućava prikazivanje svih *Warnin*g poruka;
- Opcija -g generiše strukture podataka koje su potrebne za efikasno debagovanje programa korišćenjem GDB debagera.

- Postoji mnogo drugih opcija za gcc i g++.
- Najbolji način da se dobiju kompletne informacije o opcijama je:
- **% info gcc**

Povezivanje/linkovanje, biblioteke

- Nakon kompajliranja **main.c** i **reciprocal.cpp i stvaranja objeknih fajlova,** treba ih linkovati/povezati. To se radi takođe korišćenjem gcc i g++.
- Za linkovanje programa koji sadrži C++ kod treba koristiti g++, čak iako kod sadrži C delove koda.
- Ako program sadrži samo C kod, za linkovanje treba da se koristi gcc. Budući da ovaj program sadrži i C i C ++, koristi se g++:

- **% g++ -o reciprocal main.o reciprocal.o**

- Opcija -o specificira ime **izvršne datoteke** koja se kreira nakon linkovanja.

- Izvršni program se pokreće:

- **% ./reciprocal 7**
- **The reciprocal of 7 is 0.142857**

- U Linux-u se izvršni fajl uvek pokreće sa **./ime_binarnog_fajla**
- Ovo praktično znači: pozovi fajl sa nazivom ime_binarnog_fajla iz trenutnog direktorijuma.
- Moguće je pokrenuti program samo sa **ime_binarnog_fajla** samo ukoliko se putanja fajla nalazi u PATH environment varijabli Linux OS.

Korišćenje biblioteka

- Biblioteka je kolekcija pred-kompajliranih objektnih fajlova koji mogu da se povezuju sa izvršnim fajlom pomoću linker-a.
- Postoje dva tipa biblioteka: **statičke i deljene**.
- **Statička biblioteka** ima ekstenziju .a (Linux) odnosno .lib (Windows) i prilikom povezivanja sa objektnim fajlom kod iz biblioteke se kopira u izvršni fajl.
- Ovakav tip biblioteke se može kreirati korišćenjem programa ar.
- **Deljene biblioteke** imaju ekstenziju .so (Linux) odnosno .dll (Windows). Kada se program povezuje sa ovakvom bibliotekom, samo mala tebela se kreira u izvršnom fajlu. Pre nego što se krene sa izvršavanjem, operativni sistem učitava mašinski kod zahtevan za eksterne funkcije i ovaj proces je poznat kao dinamičko povezivanje (dynamic linking).
- Na ovaj način se dobijaju manji izvršni fajlovi i čuva se prostor na disku jer se deljene biblioteke dele između više izvornih fajlova. Osim toga, nadogradnja sistema je moguća bez ponovnog rekompajliranja izvornog koda (samo je potrebno zameniti deljenu biblioteku)

- Linker traži tzv. putanju biblioteke za sve biblioteke koje su potrebne kako bi se kreirao izvršni fajl. Ova putanja se specificira sa **–L opcijom** ili environment varijablom LIBRARY_PATH.
- U našem primeru kompajler je automatski linkovao standardnu C biblioteku koja sadrži implementaciju funkcije **printf()**.
- Ako je potrebno povezivanje neke druge biblioteke, tu biblioteku treba navesti sa uključenom opcijom -l.
- U Linux-u, nazivi biblioteka gotovo uvek počinju sa **lib**. Na primer, biblioteka Pluggable Authentication Module (PAM) je **libpam.a**. Da biste povezali biblioteku libpam.a, koristite naredbu:
- **% g++ -o reciprocal main.o reciprocal.o -lpam**
- **Kompajler automatski dodaje prefiks lib i .a sufiks**.

- Kao i kod header fajlova, linker traži biblioteke na standardnim mestima koji sadrže standardne sistemske biblioteke:
  - /lib i
  - /usr/lib.
- Ako se želi da linker pretražuje i druge direktorijume sem standardnih, treba da se koristi opcija -L, koja je slična sa opcijom -I o kojoj smo ranije govorili.
- Sledeća komanda se koristi da linker traži biblioteke u direktorijumu **/usr/local/lib/pam**:
- **% g++ -o reciprocal main.o reciprocal.o –L /usr/local/lib/pam -lpam**

- Razlika je između –I i -L sledeća. Iako se ne mora koristiti opcija -I da bi se pretraživao trenutni direktorijum, mora da se koristi opcija -L da bi linker pretražio tekući direktorijum.
- Konkretno, koristi se sledeća naredba da se ukaže linkeru da pronađe biblioteku **test** koja se nalazi u trenutnom direktorijumu:
- **% gcc -o app app.o -L –ltest**

Automatizacija procesa kompajliranja, GNU Make

- Program GNU Make se koristi za automatizaciju kompajliranja programskog izvornog C/C++ koda. Koristi ga većina profesionalnih Linux programera.
- Osnovna ideja koja stoji iza programa make-a je jednostavna. Treba reći **make**-u koje ciljeve **(targets)** želimo da kompajliramo, a zatim, definisati pravila **(rules)** koja objašnjavaju kako to treba raditi. Takođe, treba odrediti zavisnosti (**dependences**) koje navode kada određeni cilj treba kompajlirati.
- U našem projektu, postoje tri cilja: objektni kodovi **reciprocal.o**, **main.o** i izvršni fajl **reciprocal**.
- Već znamo kako se prave ovi fajlovi korišćenjem opcija gcc i g++ i komandne linije.
- Treba malo razmisliti o zavisnostima (**dependences**).

- Izvršni fajl **reciprocal** zavisi od objektnih kodova **reciprocal.o** i **main.o** jer se ne može linkovati izvršni program dok se ne naprave sve objektne datoteke.
- Objektne datoteke treba ponovo kompajlirati kad god se promene odgovarajuće izvorne .c i .cpp datoteke.
- Pored toga, promena u **header** fajlu **reciprocal.hpp** takođe treba da izazove ponovno kompajliranje obe objektne .o datoteke, jer je u obe izvorne datoteke isti **reciprocal.hpp** fajl uključen.
- Pored očiglednih ciljeva, u *Make* fajlu uvek treba da postoji definicija čišćenja projekta, koja treba da izbriše sve generisane objektne datoteke i programe.
- Pravilo je da se koristi **rm naredba** za čišćenje fajlova.

```
reciprocal: main.o reciprocal.o
	g++ $(CFLAGS) -o ./../bin/reciprocal main.o reciprocal.o

main.o: main.c ../include/reciprocal.hpp
	gcc $(CFLAGS) -c main.c -I ../include/

reciprocal.o: reciprocal.cpp ../include/reciprocal.hpp
	g++ $(CFLAGS) -c reciprocal.cpp -I ../include/

clean:
	rm -f *.o ./../bin/reciprocal
```

- Može se videti da su ciljevi navedeni sa leve strane, iza cilja je dvotačka, a zatim i sve zavisnosti.
- Pravilo za realizaciju cilja nalazi se u sledećem redu koji počinje znakom Tab.
- Za pokretanje **make** fajla samo treba ukucati **make**:
- **% make**
- u komandnoj liniji se vidi sledeće:
- **% make**
- **gcc -c main.c -I ../include/**
- **g++ -c reciprocal.cpp -I ../include/**
- **g++ -o ./../bin/reciprocal main.o reciprocal.o**
-

- **make** je prvo napravio objektne .o datoteke, a zatim ih linkovao.
- Ako se promeni izvorni kod **main.c** i ponovo ukuca **make**, dobija se:

- **% make**
- **gcc -c main.c -I ../include/**
- **g++ -o ./../bin/reciprocal main.o reciprocal.o**

- Objektni fajl **reciprocal.o** se nije rekompajlirao, nije bilo potrebe za tim jer nijedna zavisnost vezana **reciprocal.o** se nije promenila. Samo se rekompajlirao objektni fajl **main.o** i linkovao program **reciprocal**.

- Čišćenje projekta:
- **% make clean**
- rm -f *.o ./../bin/reciprocal

- U make fajlovima moguće je koristiti promenljive.
- $ (CFLAGS) je **make** promenljiva. Promenljiva može da se definiše u Makefile ili preko komandne linije.
- Tokom kompajliranja **make** će zameniti vrednost promenljive. Na primer, da bi se iskompajlirao program sa uključenom optimizacijom:

% make CFLAGS=-O2

- **gcc -O2 -c main.c -I ../include/**
- **g++ -O2 -c reciprocal.cpp -I ../include/**
- **g++ -O2 -o ./../bin/reciprocal main.o reciprocal.o**
- Vidi se da je Flag -O2 umetnut umesto $(CFLAGS).

Pokretanje programa GDB za debagovanje/otklanjanje grešaka

- make CFLAGS=-g
- Cd ../bin
- Za pokretanje opcije debagovanja:

- **% gdb reciprocal**
- **(gdb)**

- Prvi korak je pokretanje programa unutar **gdb**.
- To se postiže komandom **run** i bilo kojim argumentom programa.
- Prvo ćemo da pokrenemo **reciprocal** program, ali na pogrešan način, bez navođenja argumenta:

```
(gdb) run
Starting program: reciprocal
Program received signal SIGSEGV, Segmentation
fault.
__strtol_internal (nptr=0x0, endptr=0x0, base=10,
group=0)
at strtol.c:287
287 strtol.c: No such file or directory.
(gdb)
```

- Problem je u tome što u main.c ne postoji kod za proveru grešaka.
- Program očekuje jedan argument, ali u ovom slučaju program je pokrenut bez argumenata.
- The SIGSEGV poruka ukazuje na pucanje programa.
- GDB zna da se stvarni pad dogodio u funkciji koja se zove __strtol_internal. Ta funkcija se nalazi u standardnoj biblioteci, njen izvorni kod nije dostupan, što objašnjava poruku „ No such file or directory “.
- Može da se **stek** pogleda pomoću komande **where**:

```
(gdb) where
#0 __strtol_internal (nptr=0x0, endptr=0x0, base=10, group=0)
at strtol.c:287
#1 0x40096fb6 in atoi (nptr=0x0) at ../stdlib/stdlib.h:251
#2 0x804863e in main (argc=1, argv=0xbffff5e4) at main.c:8
```

- Može se videti da je funkcija main programa *atoi()* pozvala NULL pokazivač, što je izvor greške.
- Možemo se popeti za dva nivoa iznad u steku dok se ne dođe do glavne funkcije.
- To se radi korišćenjem u**p** naredbe:

```
(gdb) up 2
#2 0x804863e in main (argc=1,
argv=0xbffff5e4) at main.c:8
8 i = atoi (argv[1]);
```

- **gdb** može da pronađe izvorni kod main.c i da prikaže liniju gde je došlo greške, tj. do pogrešnog poziva funkcije.

- Pored toga, vrednost promenljivih može se videti pomoću naredba **print**

```
(gdb) print argv[1]
$2 = 0x0
```

Breakpoint možete postaviti pomoću naredbe **break**:

```
%(gdb) break main
Breakpoint 1 at 0x804862e: file main.c, line 8.
```

Ova komanda postavlja breakpoint na prvom redu glavnog programa.
Ako se ponovo pokrene program sa argumentom:

```
(gdb) run 7
Starting program: reciprocal 7
Breakpoint 1, main (argc=2, argv=0xbffff5e4) at
main.c:8
8 i = atoi (argv[1]);
```

Može se videti da se program **gdb** zaustavio na **breakpoint**.
Možete preći preko poziva na *atoi()* pomoću naredbe **next**:

```
(gdb) next
9 printf (“The reciprocal of %d is %g\n”, i, reciprocal (i));
```

Ako želite da vidite šta se dešava unutar **reciprocal**, koristite naredbu **step**:

```
(gdb) step
reciprocal (i=7) at reciprocal.cpp:6
6 assert (i != 0);
```

Listing breakpoints.
(gdb) info break

Delete breakpoints.
(gdb) delete 1

Statičke i dinamičke biblioteke

- Arhiva (ili statička biblioteka) je jednostavno kolekcija objektnih datoteka sačuvanih u vidu jedne datoteke.
- Arhiva u Linux-u je ekvivalent Windows .LIB datoteke.
- Možete kreirati arhivu pomoću komande ar.
- Arhivske datoteke tradicionalno koriste .a ekstenziju, a ne .o, koju koriste obične objektne datoteke.

- Možete da kombinujete dva objektna fajla test1.o i test2.o u jednu arhivu libtest.a:

% ar cr libtest.a test1.o test2.o

- Fleg cr kaže komandi ar da kreira arhivu.
- Sada možete da linkujete izvršni fajl koristeći kreiranu arhivu.
- Pritom se koristi opcija –ltest

gcc -o app app.o -L –ltest

- Zajednička biblioteka (***shared library*** ili kao dinamički povezana biblioteka) slična je arhivi po tome što predstavlja grupisanje objektnih datoteka.
- Razlika je u tome što kada je ***shared*** biblioteka povezana u konačni izvršni program, tada on ne sadrži kod koji je prisutan u deljenoj biblioteci.
- Umesto toga, izvršna datoteka sadrži samo referencu na ***shared*** biblioteku.
- Ako je nekoliko programa na sistemu povezano sa istom deljenom bibliotekom, svi će referencirati biblioteku, ali nijedan neće biti uključen. Stoga se biblioteka „deli“ među svim programima koji su povezani sa njom.
- Da biste kreirali zajedničku biblioteku, morate prvo kompajlirati objekte koristeći opciju -fPIC \:

gcc -c -fPIC test1.c

- Opcija -fPIC govori kompajleru da ćete koristiti test.o kao deo zajedničkog objekta

- Objektne datoteke se kombinuju u zajedničku biblioteku ovako:

% gcc -shared -fPIC -o libtest.so test1.o test2.o

- Dobija se shared biblioteka libtest.so
- Povezivanje sa zajedničkom bibliotekom je isto kao povezivanje sa statičkom arhivom.
- Na primer, sledeći red će se linkovati program sa libtest.so.
- Pretpostavlja se da je biblioteka u trenutnom direktorijumu ili jednom od standardnih direktorijuma za pretragu biblioteka na sistemu:
- % gcc -o app app.o -L –ltest


ONE AFTERNOON, IN METROPOLIS...
AUGUST GORMAN? ACCORDING TO OUR RECORDS, YOU HAVE BEEN UNEMPLOYED FOR 36 WEEKS! MR. GORMAN, THE CITY OF METROPOLIS IS GENEROUS TO A FAULT, BUT...
METROPOLIS UNEMPLOYMENT BUREAU
I KNOW, I'M THE FAULT.
36 WEEKS OF CHRONIC UNEMPLOYMENT, LIVING OFF THE TAXES OF HARD-WORKING CITIZENS. DO YOU KNOW WHAT YOU ARE--?
CASHIER
A-G
DON'T CALL ME NO BUM. I AIN'T A BUM!
YOU ARE-- I WAS ABOUT TO SAY-- NO LONGER ELIGIBLE!
CASHIER
A-G


NOT MY FAULT, ALL THOSE TEMP JOBS FALLIN' THROUGH ONE AFTER ANOTHER. NEVER THOUGHT IT'D BE SO HARD TO EARN A BUCK IN THIS TOWN...
HEY, BROTHER, GOT A MATCH?
SURE THING, MY MAN. KEEP IT, WHY DON'T YOU?
HMMMM. TALK ABOUT GOOD TIMING... WHAT'VE YOU GOT TO LOSE?
EARN BIG MONEY!
BECOME A COMPUTER PROGRAMMER
4