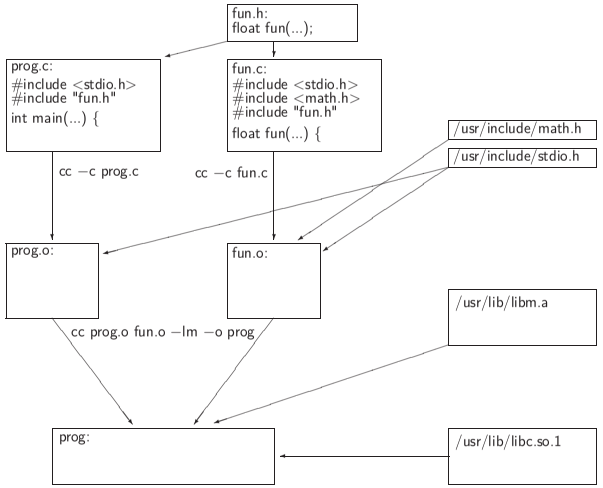
Unix: kompilator C i narzędzia pomocnicze

Witold Paluszyński
witold.paluszynski@pwr.wroc.pl
http://kcir.pwr.wroc.pl/~witold/

Niniejszy dokument zawiera materiały do wykładu na temat systemu kompilatora C i narzędzi pomocniczych w systemie Unix. Jest on udostępniony pod warunkiem wykorzystania wyłącznie do własnych, prywatnych potrzeb i może być kopiowany wyłącznie w całości, razem z niniejszą stroną tytułową.

System kompilatora C



wywołanie pełnej kompilacji:

```
cc prog.c fun.c -lm -o prog
```

```
cc -c prog.c
cc -c fun.c
cc prog.o fun.o -lm -o prog
```

Opcje wywołania kompilatora C (wspólne)

Tradycyjnie, kompilatory C rozpoznają te opcje jednakowo:

-o*nazwa* umieść postać wynikową kompilacji w pliku *nazwa*, domyślnie *a.out* dla postaci programu wykonywalnego, *nazwazrodla.s* dla postaci asemblerowej, i *nazwazrodla.o* dla postaci binarnej

-c pomiń ostatnią fazę kompilacji (linker), nie twórz programu wynikowego, pozostaw postać binarną .o

-g wpisz w program binarny dodatkowe informacje dla debuggera

-l*bib* powoduje przeglądanie przez linker biblioteki *bib*, w pliku o nazwie `libbib.a` lub `libbib.so` w kartotece `/usr/lib` lub w innych zdefiniowanych ścieżką linkera

-S wykonaj tylko pierwszą fazę kompilacji do kodu asemblera .s

-O*n* wykonaj optymalizację kodu poziomu *n* (domyślnie poziom 2, który jest na ogół bezpieczny)

-w pomiń ostrzeżenia (opcja zwykle szkodliwa)

Opcje wywołania kompilatorów (różne)

Niestety, niektóre ważne i pożyteczne opcje występują tylko dla niektórych kompilatorów, lub mają inną postać:

-V wyświetlaj wywołania kolejnych faz kompilacji (Sun cc)

-v wyświetlaj wywołania kolejnych faz kompilacji (HP cc, GNU gcc)

-Xc ścisłe przestrzeganie standardu ANSI C (Sun cc)

-Aa ścisłe przestrzeganie standardu ANSI C (HP cc)

-ansi przestrzeganie standardu ANSI C (GNU gcc)

-pedantic ścisłe przestrzeganie standardu ANSI C (GNU gcc)

-Wall wyświetlanie ostrzeżeń o wszelkich „dziwnych" konstrukcjach programowych (GNU gcc)

Narzędzie `make`

skrypt do kompilacji programu:

```
cc -c prog.c
cc -c fun.c
cc prog.o fun.o -lm -o prog
```

specyfikacja Makefile:

```
prog: prog.o fun.o
        cc prog.o fun.o -lm -o prog
prog.o: prog.c fun.h
        cc -c prog.c
fun.o: fun.c fun.h
        cc -c fun.c
```

alternatywna specyfikacja Makefile:

```
CC = gcc
LDFLAGS = -lcurses

prog: prog.o fun.o
prog.o: prog.c
fun.o: fun.c
```

`make` — zasady ogólne

`make` jest uogólnieniem skryptu do kompilacji pakietów wybierającym tylko niezbędne do rekompilacji moduły i właściwe elementy kompilatora:

- ułatwia kompilację pakietów składających się z wielu części, gdzie wprowadzenie poprawek powoduje konieczność rekompilacji niektórych tylko części całego pakietu
- plik opisowy Makefile zawiera specyfikację zależności pomiędzy elementami pakietu, które następnie są sprawdzane przez program `make` względem dat utworzenia odpowiednich plików
- gdy jakaś zależność nie jest zachowana, `make` „produkuje" ponownie dany obiekt (plik) według reguł również zawartych w pliku Makefile
- `make` jest przygotowany do kompilacji programów w C i posiada wbudowany zestaw reguł dla kompilacji tych programów włącznie z całym zestawem narzędzi Unixowych (np. `yacc`, `lex`), lecz równie dobrze nadaje się do dowolnego przetwarzania pakietów o dowolnym charakterze (np. skład tekstu skomplikowanego dokumentu) dzięki możliwości zdefiniowania odpowiednich reguł i poleceń

Przykładowy plik Makefile

```
SHELL=/bin/sh
.SUFFIXES: .txt .tex .dvi .ps .ps2 .pdf .bib .bbl .idx .ind

.txt.tex:
	/usr/bin/sed -f $(HOME)/lib/txt2cudzy.sed $< >$@
.tex.dvi:
	latex $<
.dvi.ps:
	dvips -T 11in,8.5in -t landscape $<
.ps.ps2:
	psnup -2 $< $@
	mgland2print $< $@
.dvi.pdf:
	dvipdfm -l -p letter $*
.bib.bbl:
	bibtex $*
.idx.ind:
	makeindex $<

all: upr_shell upr_tools upr_libs upr_jsys upr_llio upr_proc upr_advio

upr.ps: upr.dvi
upr.dvi: upr.tex
```

```
# pomijamy zaleznosc upr.dvi od upr.bbl i upr.ind bo powoduje petlenie
upr.tex: upr.txt

bib biblio: upr.bbl
upr.bbl: upr.aux Unixprog.bib
	bibtex upr

ind indeks index: upr.ind
upr.ind: upr.idx
upr.idx: upr.tex

upr_shell: upr_shell.ps upr_shell.ps2 upr_shell.pdf
upr_shell.ps: upr_shell.dvi
upr_shell.ps2: upr_shell.ps
upr_shell.pdf: upr_shell.dvi
upr_shell.dvi: upr_shell.tex
upr_shell.tex: upr_shell.txt

upr_tools: upr_tools.ps upr_tools.ps2 upr_tools.pdf
upr_tools.ps: upr_tools.dvi
upr_tools.ps2: upr_tools.ps
upr_tools.pdf: upr_tools.dvi
upr_tools.dvi: upr_tools.tex
upr_tools.tex: upr_tools.txt
```

```
upr_libs: upr_libs.ps upr_libs.ps2 upr_libs.pdf
upr_libs.ps: upr_libs.dvi
upr_libs.ps2: upr_libs.ps
upr_libs.pdf: upr_libs.dvi
upr_libs.dvi: upr_libs.tex upr_libs_intro.tex upr_libs_string.tex \
              upr_libs_curses.tex upr_libs_search.tex upr_libs_ndbm.tex \
              upr_libs_regcomp.tex upr_libs_crypt.tex
upr_libs.tex: upr_libs.txt
upr_libs_intro.tex: upr_libs_intro.txt
upr_libs_string.tex: upr_libs_string.txt
upr_libs_curses.tex: upr_libs_curses.txt
upr_libs_search.tex: upr_libs_search.txt
upr_libs_ndbm.tex: upr_libs_ndbm.txt
upr_libs_regcomp.tex: upr_libs_regcomp.txt
upr_libs_crypt.tex: upr_libs_crypt.txt

upr_jsys: upr_jsys.ps upr_jsys.ps2 upr_jsys.pdf
upr_jsys.ps: upr_jsys.dvi
	dvips -t a4 upr_jsys
upr_jsys.pdf: upr_jsys.dvi
upr_jsys.dvi: upr_jsys.tex
upr_jsys.tex: upr_jsys.txt

upr_llio: upr_llio.ps upr_llio.ps2 upr_llio.pdf
upr_llio.ps: upr_llio.dvi
```

```
	dvips -t a4 upr_llio
upr_llio.pdf: upr_llio.dvi
upr_llio.dvi: upr_llio.tex
upr_llio.tex: upr_llio.txt

upr_advio: upr_advio.ps upr_advio.ps2 upr_advio.pdf
upr_advio.ps: upr_advio.dvi
	dvips -t a4 upr_advio
upr_advio.pdf: upr_advio.dvi
upr_advio.dvi: upr_advio.tex
upr_advio.tex: upr_advio.txt

upr_proc: upr_proc.ps upr_proc.ps2 upr_proc.pdf
upr_proc.ps: upr_proc.dvi
	dvips -t a4 upr_proc
upr_proc.ps2: upr_proc.ps
	psnup -pa4 -Pa4 -2 upr_proc.ps upr_proc.ps2
upr_proc.pdf: upr_proc.dvi
upr_proc.dvi: upr_proc.tex
upr_proc.tex: upr_proc.txt

upr_thr: upr_thr.ps upr_thr.ps2 upr_thr.pdf
upr_thr.ps: upr_thr.dvi
	dvips -t a4 upr_thr
upr_thr.ps2: upr_thr.ps
```

```
	psnup -pa4 -Pa4 -2 upr_thr.ps upr_thr.ps2
upr_thr.pdf: upr_thr.dvi
upr_thr.dvi: upr_thr.tex
upr_thr.tex: upr_thr.txt
```

System kontroli wersji RCS

Funkcje systemu RCS (**R**evision **C**ontrol **S**ystem):

- pamiętanie pełnego zestawu archiwalnych wersji danego pliku z automatyczną numeracją wersji, dokumentacją i rejestracją czasu utworzenia i autora; dostęp do dowolnej wersji poprzez: numer, datę, autora, bądź nazwę,
- obsługa dodatkowych (bocznych) odgałęzień wersji pliku,
- koordynacja pracy pomiędzy wieloma programistami pracującymi nad projektem poprzez ustawianie praw dostępu oraz mechanizm wypożyczania i zwracania plików,
- ograniczona możliwość automatycznego łączenia dwóch różnych zmodyfikowanych wersji jednego pliku; powstanie takich wersji może być wynikiem nieporozumienia programistów, bądź chęci przeniesienia na jedną gałąź rozwojową pliku poprawek dokonanych na innej gałęzi
- korzystanie z automatycznie aktualizowanych znaczników w plikach.

System RCS: polecenia

- `ci` przekazuje plik systemowi RCS, nadaje mu numer wersji (kolejny) oraz pobiera i prosi o podanie opisu dokonanej modyfikacji; oryginalny plik zostaje skasowany (z wyjątkiem `ci -u` lub `ci -l`, to ostatnie wywołanie od razu blokuje plik do edycji)
- `co` pobiera plik z systemu RCS; z opcją -l również blokuje plik tak, że wypożyczoną kopię można edytować
- `rlog` wyświetla kompletną historię pliku włącznie z opisami istniejących wersji i aktualnym stanem wypożyczenia pliku
- `rcsdiff` wyświetla różnice pomiędzy bieżącym plikiem a jego ostatnią wersją przechowywaną przez system RCS
- `rcsmerge` służy do połączenia dwóch zmodyfikowanych wersji jednego pliku
- `rcs` – komenda administracyjna, służy do modyfikacji ustawień

System RCS: przykład

```
shasta-201> ci upr_shell.txt
upr_shell.txt,v  <--  upr_shell.txt
enter description, terminated with single '.' or end of file:
NOTE: This is NOT the log message!
>> Prezentacja na temat Bourne shella i filtrow tekstowych
>> .
initial revision: 1.1
done
shasta-202> ls -l upr_shell.txt
upr_shell.txt: No such file or directory
shasta-203> co -l upr_shell.txt
upr_shell.txt,v  -->  upr_shell.txt
revision 1.1 (locked)
done
shasta-206> ls -l upr_shell*
-rw-r--r--   1 witold   gurus        180 Nov  5 11:58 upr_shell.aux
-rw-r--r--   1 witold   gurus      41556 Nov  5 10:59 upr_shell.dvi
-rw-r--r--   1 witold   gurus       9681 Nov  5 11:58 upr_shell.log
-rw-r--r--   2 witold   gurus     129537 Nov  5 11:58 upr_shell.pdf
-rw-r--r--   2 witold   gurus     215917 Nov  5 10:59 upr_shell.ps
-rw-r--r--   1 witold   gurus     243737 Nov  5 10:59 upr_shell.ps2
-rw-r--r--   1 witold   gurus      30300 Nov  5 10:59 upr_shell.tex
-rw-r--r--   1 witold   gurus      30300 Nov 12 15:17 upr_shell.txt
-r--r--r--   1 witold   gurus      30555 Nov 12 15:17 upr_shell.txt,v
```

```
shasta-207> rlog upr_shell.txt

RCS file: upr_shell.txt,v
Working file: upr_shell.txt
head: 1.1
branch:
locks: strict
access list:
symbolic names:
keyword substitution: kv
total revisions: 1;     selected revisions: 1
description:
Prezentacja na temat Bourne shella i filtrow tekstowych
----------------------------
revision 1.1    locked by: witold;
date: 2003/11/12 14:17:50;  author: witold;  state: Exp;
Initial revision
=============================================================================
```

Często użycie systemu RCS polega na utworzeniu podkartoteki o nazwie RCS, w której polecenia systemu RCS umieszczają swoje archiwa, co pozwala np. na łatwą archiwizację. Często również każde wprowadzenie pliku do archiwum od razu związane jest z zatrzymaniem jego oryginalnej wersji źródłowej do dalszej edycji przez autora: `ci -l upr_shell.txt`

Biblioteki programowe

Biblioteka jest plikiem zawierającym zbiór skompilowanych modułów programowych `.o` w formacie, z którego linker może je wydobyć w czasie kompilacji jakiegoś programu, i wykorzystać funkcje i symbole potrzebne temu konkretnemu programowi.

Biblioteki są stosowane najczęściej jako biblioteki systemowe, zawierające większą liczbę modułów, z którymi programista nie chciałby mieć do czynienia indywidualnie.

W podobnej roli jednak można również tworzyć biblioteki własne, gdy posiadamy zbiór modułów i chcemy korzystać z nich, lub udostępnić je komuś, jako całość, nie wnikając w którym module jest która funkcja, albo z których innych funkcji (i z jakich modułów) ona korzysta.

Biblioteki programowe: przykład użycia

Poniższy przykład zakłada, że istnieje biblioteka programowa w pliku `${HOME}/mylibs/liballmath.a` zawierająca funkcje matematyczne z dwóch binarnych modułów programowych `${HOME}/mylibs/geometry.o` i `${HOME}/mylibs/basicmath.o`.

Rozdzielona faza kompilacji bez linkowania, i linkowania programu z modułami binarnymi, oraz z zawierającą je wszystkie biblioteką, zadaną jako bezwzględna ścieżka pliku biblioteki, oraz jako biblioteka z wykorzystaniem ścieżki wyszukiwania bibliotek linkera:

```
# program compilation, no linking
cc -c -o mymathprog.o mymathprog.c

# version 1 - linking with binary object modules
cc -o mymathprog mymathprog.o ${HOME}/mylibs/geometry.o \
                              ${HOME}/mylibs/basicmath.o

# version 2 - linking with library file using full file path
cc -o mymathprog mymathprog.o ${HOME}/mylibs/liballmath.a

# version 3 - linking with library file using linker's search path
cc -o mymathprog mymathprog.o -L ${HOME}/mylibs/ -lallmath.a
```

Biblioteki statyczne, dynamiczne, i współdzielone

Tradycyjny model tworzenia bibliotek uniksowych polega na zapisaniu ich w jednym pliku archiwum. Biblioteka może posiadać indeks, ułatwiający linkerowi wyszukanie i skompletowanie wszystkich modułów i funkcji niezbędnych danemu programowi.

Program zlinkowany z biblioteką statyczną zawiera w sobie cały niezbędny kod z tej biblioteki, i może np. być przeniesiony i uruchomiony na innym systemie, w którym dana biblioteka nie istnieje.

Nowsze systemy posiadają również mechanizmy dynamicznego linkowania bibliotek. Szczególnie przydatne są biblioteki współdzielone, wykorzystywane przez wiele procesów jednocześnie. Program zlinkowany z taką biblioteką nie zawiera w sobie jej kodu, a jedynie informacje o niej: nazwę, numer wersji, i lokalizację w systemie.

W oczywisty sposób, plik binarny programu może być w ten sposób dużo mniejszy (np. jeśli korzysta z ogromnych bibliotek graficznych), ale w czasie wykonywania musi mieć dostęp do biblioteki dynamicznej. Oznacza to, że musi ona być obecna w systemie, w którym wykonywany jest program, a ponadto linker dynamiczny musi być w stanie znaleźć ją w tym systemie.

Tworzenie bibliotek współdzielonych wygląda inaczej, i wymaga innych narzędzi, na różnych systemach i dla różnych kompilatorów.

Tworzenie bibliotek statycznych

Tradycyjny model tworzenia bibliotek (statycznych) polega na zapisaniu skompilowanych modułów `.o` we wspólnym pliku archiwum. Wykorzystuje się do tego program archiwizacyjny ogólnego przeznaczenia `ar` tworzący archiwum w pliku o nazwie zwyczajowo z końcówką `.a`.

```
cc -c -o geometry.o geometry.c
cc -c -o basicmath.o basicmath.c
ar -r allmath.a basicmath.o geometry.o
ranlib allmath.a
```

Jedyny problem w powyższej procedurze wiąże się z procesem kompletowania niezbędnych procedur w czasie linkowania programów przez linker, i koniecznością wieloprzebiegowego czytania archiwum. Z tego względu wprowadzono indeksowanie biblioteki, i do tworzenia takiego indeksu służy program `ranlib`, aczkolwiek jego użycie jest opcjonalne i przyczynia się tylko do poprawienia szybkości linkowania.

Nowsze wersje archiwizatora `ar` potrafią tworzyć i aktualizować indeks archiwum, i obecnie zwykle nie stosuje się programu `ranlib`.

Biblioteki dynamicznie ładowane i współdzielone

W systemach uniksowych istnieją dwa mechanizmy dynamicznego linkowania bibliotek: biblioteki dynamicznie ładowane (*dynamically loaded libraries*) i biblioteki wspóldzielone (*shared object libraries*).

Biblioteki dynamicznie ładowane nie różnią się od innych bibliotek pod względem struktury; każda biblioteka może być dynamicznie ładowana. Mechanizm dynamicznego ładowania pozwala na ich dołączanie do programu na komendę, a nie w chwili startu programu. Umożliwia to implementację wtyczek, i/lub modułów kompilowanych przez program ze źródeł na swój własny użytek. Dynamiczne ładowanie bibliotek jest podobne w systemach Solaris i Linux.

Biblioteki współdzielone (o specjalnym formacie, zwyczajowo w plikach z końcówką `.so`) mają dwie cechy różniące je od bibliotek statycznych.

Po pierwsze, linker nie dołącza do tworzonego pliku programu binarnego kodu biblioteki, a jedynie informacje o niej: nazwę (która opcjonalnie może zawierać numer wersji biblioteki, jeśli dla danego programu ma to znaczenie), i lokalizację w systemie. W chwili startu programu uruchamiany jest **linker dynamiczny**, który musi odnaleźć bibliotekę na dysku, i załadować ją do pamięci.

Po drugie, kod biblioteki *shared object* jest ładowany do pamięci raz, i współdzielony przez wszystkie procesy linkujące z daną biblioteką, co ma zasadnicze znaczenie jeśli w systemie wiele procesów korzysta z dużych bibliotek.

Tworzenie bibliotek współdzielonych - Solaris/SunStudio

W poniższym przykładzie najpierw kompilujemy w podkatalogu `./lib` plik źródłowy biblioteki współdzielonej do modułu binarnego `mysharedlib.o`, a następnie tworzymy bibliotekę współdzieloną `libmysharedlib.so.1` w tym samym podkatalogu.

```
# compile library source file into a binary module
(cd lib; cc -c -o mysharedlib.o mysharedlib.c)

# link the library module into a shared object file
ld -G -o lib/libmysharedlib.so.1 lib/mysharedlib.o

# another way of doing the same
cc -G -o lib/libmysharedlib.so.1 lib/mysharedlib.o

# now arrange the "version 1" library file to be the default
(cd lib; ln -s libmysharedlib.so.1 libmysharedlib.so)
```

Linker `ld` tworzy biblioteki *shared object* po zadaniu opcji `-G`. Opcję tę można również zadać programowi `cc`, który przekazuje ją linkerowi, jednak wywołując linker za pośrednictwem programu `cc` należy przekazywać inne opcje linkerowi za pomocą opcji `-W`.

Tworzenie bibliotek współdzielonych — Linux/gcc

Ponownie tworzymy w podkatalogu `./lib` bibliotekę współdzieloną `libmysharedlib.so.1` zawierającą pojedynczy moduł binarny `mysharedlib.o`:

Gnu linker (zwany pieszczotliwie `collect2`) podobnie tworzy biblioteki *shared object* po zadaniu opcji `-G`. W tym przypadku jednak wygodniej jest nie wywoływać go bezpośrednio, a skorzystać z pośrednictwa programu `gcc`:

```
# compile library source file into a binary module
(cd lib; gcc -fpic -c mysharedlib.c)

# link the library module into a shared object file
gcc -shared -o lib/libmysharedlib.so.1 lib/mysharedlib.o

# now arrange the "version 1" library file to be the default
(cd lib; ln -s libmysharedlib.so.1 libmysharedlib.so)
```

Więcej informacji:

`http://www.faqs.org/docs/Linux-HOWTO/Program-Library-HOWTO.html`

Linkowanie z wykorzystaniem bibliotek *shared object*

W poniższym przykładzie najpierw kompilujemy program `mymain.c` korzystający z biblioteki dynamicznej, a następnie linkujemy go z tą biblioteką z różnymi wariantami podawania ścieżki do biblioteki.

```
# first compile our simple program into an object module
cc -c mymain.c

# and then link with the shared object previously created

# version 1: path to the library not recorded during linking
cc mymain.o -L lib/ -lmysharedlib
# the following fails:
./a.out
# but this works:
LD_LIBRARY_PATH=. ./a.out

# version 2: relative path to the library stored in the binary file
cc mymain.o -L lib/ -lmysharedlib -R lib/
# the following works
./a.out
# but this fails
(cd lib; ../a.out)
```

Biblioteki współdzielone: więcej o nazwach i numerach wersji

Biblioteki współdzielone mają mechanizm numerów wersji. W przypadku pojedynczej, indywidualnie wykorzystywanej biblioteki, mechanizm ten może być zignorowany i całkowicie pominięty. Jednak w przypadku, gdy biblioteki używa wiele programów, a jej twórca chce bez zmiany nazwy zmienić np. interfejs niektórych funkcji, i umożliwić jednoczesną pracę w systemie programów korzystających z wielu wersji biblioteki, ten mechanizm jest niezwykle przydatny.

Polega on na tym, że biblioteki współdzielone posiadają nazwę tzw. SONAME, która składa się z nazwy pliku biblioteki poczynając od 'lib*' a kończąc na pojedynczym numerze wersji następującym po rozszerzeniu `.so`, czyli np. `.so.1`. Nazwa SONAME nie jest jedyną nazwą biblioteki; linker musi odnaleźć jej plik przez nazwę kończącą się na `.so`. Jednak jeśli nazwa SONAME jest zapisana w bibliotece, to jest następnie zapisywana we wszystkich programach oryginalnie z nią zlinkowanych, przez co w chwili startu każdego programu linker dynamiczny może odnaleźć wersję biblioteki dla niego niezbędną.

```
# Solaris - nazwa SONAME zadana argumentem -h
ld -G -h libmysharedlib.so.1 -o libmysharedlib.so.1 mysharedlib.o

# Linux - nazwa SONAME zadana argumentem -soname linkera
gcc -shared -o lib/libmysharedlib.so.1 -Wl,-soname,libmysharedlib.so.1 lib/mysharedlib.
```

```
# version 3: absolute path to the library stored in the binary file
cc mymain.o -L lib/ -lmysharedlib -R `pwd`/lib/
# anything works now
./a.out
(cd lib; ../a.out)

# version 4: like version 3, but library path in env.variable
LD_RUN_PATH=`pwd`/lib/ cc mymain.o -L lib/ -lmysharedlib
# again, anything works
./a.out
(cd lib; ../a.out)

# version 5: now use the particular version number of the library
LD_RUN_PATH=`pwd`/lib/ cc mymain.o -L lib/ -lmysharedlib
# anything works, as long as the required version of the library exists
./a.out
(cd lib; ../a.out)
```