Link 4B Library search using -L and -l only

Young W. Lim

2024-07-05 Fri

Outline

1. Based on

2. Search libraries (I)
   - Compile time and run time
   - Specifying library paths in gcc
   - Dynamic linker `ld.so` library search order
   - Link time library paths : `-L` and `-l`
   - Run time library resolution : `LD_LIBRARY_PATH`

Based on

"Study of ELF loading and relocs", 1999
http://netwinder.osuosl.org/users/p/patb/public_html/elf_relocs.html

Compling 32-bit program on 64-bit gcc

- gcc -v
- gcc -m32 t.c
- sudo apt-get install gcc-multilib
- sudo apt-get install g++-multilib
- gcc-multilib
- g++-multilib
- gcc -m32
- objdump -m i386

Compile time and run time

Compile time and run time (1)

1. the compile-time linking
   `gcc` and `ld`
1. run-time linker lookups
   generally `ld.so` (`/lib64/ld-linux-x86-64.so`)

`https://stackoverflow.com/questions/1904990/what-is-the-difference-between-ld-libr`

Compile time and run time (2)

- when you compile your program,
  the compiler (gcc) checks syntax, and then
  the linker (ld) ensures that
  the symbols required for execution *exist*
  (i.e *variables*, *methods* etc)

- when you run your program,
  the run-time linker (ld.so)
  - actually *fetches* the shared libraries
  - *loads* in the shared symbols / code / etc.

https://stackoverflow.com/questions/1904990/what-is-the-difference-between-ld-libr

Compile time and run time (3)

- When you are *compiling* a program,
  you create *object files*
  and then *link* them together.
- may use GNU `ld` to link them,
  there are also other linkers, `LLVM` linker
- a linker combines *object files* into *executable*
  GNU `ld` is part of binutils with documentation

`https://stackoverflow.com/questions/67131565/how-do-i-set-dt-rpath-or-dt-runpath`

Compile time and run time (4)

- When you execute an already compiled ready to use *executable*
  then the dynamic linker `ld.so`
  finds the libraries that the *executable* depends on,
  loads them and executes the *executable*
- `ld.so` is a shared library usually distributed
  as part of C standard library,
  usually on linux that's `glibc`,
  but there are also other, like `musl`.

`https://stackoverflow.com/questions/67131565/how-do-i-set-dt-rpath-or-dt-runpath`

Linking (1)

- for linking, make sure you specify
  - object files (or source files) before libraries (`-l` options)
  - `-L` option for a given library before the `-l` option
    (`*.c *.o -L... -l...`)
- the order of libraries can matter
  - libraries listed earlier can be referenced in those listed later
  - avoid circular references between libraries

`https://stackoverflow.com/questions/31455979/how-to-specify-libraries-paths-in-gc`

Linking (2)

- The way my IDE handles the process is
  to put the -L tag up front and
  the -l tag at the end
- all of the -l tags need to come after your target
  so that the compiler knows
  which symbols need to be resolved
  before searching

https://stackoverflow.com/questions/31455979/how-to-specify-libraries-paths-in-gc

Linking (3)

- example 1

```
gcc -L/path/to/library -o target_here -lfirst -lsecond -lthird ...
```

- example 2

```
gcc imagefilter.c -o imagefilter \
    -I/home/savio/opencv-3.0.0/include/opencv \
    -L/home/savio/opencv-3.0.0/cmake_binary_dir/lib \
    -lopencv_imgcodecs \
    -lopencv_imgproc \
    -lopencv_highgui \
    -lopencv_core
```

https://stackoverflow.com/questions/31455979/how-to-specify-libraries-paths-in-gc
https://stackoverflow.com/questions/31455979/how-to-specify-libraries-paths-in-gc

(2) Running

- both the compiler (gcc) / linker (ld)
  and the runtime system (ld.so)
  need to be able to *find* the shared objects
  - the -L option is used to tell the linker (ld)
    where to find the libraries (shared objects)
  - lots of ways of telling the runtime (dynamic loader ld.so)
    where to find the libraries (shared objects)
    - -R
    - LD_LIBRARY_PATH
    - LD_RUN_PATH

https://stackoverflow.com/questions/31455979/how-to-specify-libraries-paths-in-gc

Specifying library paths in gcc

Specifying library paths by LD_LIBRARY_PATH (1)

- Add the directory
  to LD_LIBRARY_PATH environment variable or its equivalent

```
LD_LIBRARY_PATH=\
  /home/savio/opencv-3.0.0/cmake_binary_dir/lib\
  :$LD_LIBRARY_PATH ./imagefilter
```

or:

```
export LD_LIBRARY_PATH=\
         /home/savio/opencv-3.0.0/cmake_binary_dir/lib\
         :$LD_LIBRARY_PATH ./imagefilter
```

https://stackoverflow.com/questions/31455979/how-to-specify-libraries-paths-in-gc

Specifying library paths by LD_LIBRARY_PATH (2)

- The first notation sets the environment variable
  just for as long as *the program is running*
  - useful if you need to compare
    the behaviour of two versions of a library, for example.

```
LD_LIBRARY_PATH=\
  /home/savio/.../lib\
  :$LD_LIBRARY_PATH ./imagefilter
```

https://stackoverflow.com/questions/31455979/how-to-specify-libraries-paths-in-gc

Specifying library paths by LD_LIBRARY_PATH (3)

- The second notation sets the environment variable for *the session*.
  - might include that in your `.profile` or equivalent so it applies to every session.

```
export LD_LIBRARY_PATH=\
         /home/savio/.../lib\
         :$LD_LIBRARY_PATH ./imagefilter
```

https://stackoverflow.com/questions/31455979/how-to-specify-libraries-paths-in-gc

Specifying library paths by RPATH/RUNPATH (1-1)

- When creating the elf file with GNU `ld -rpath=path`,
  path is added to the runtime library search path RUNPATH
  (DT_RUNPATH entry in .dynamic section)
- with `ld --disable-new-dtags -rpath=path`
  path is added to the runtime library search path RPATH
- with `ld --enable-new-dtags -rpath=path`
  path is added to the runtime library search path RUNPATH

https://stackoverflow.com/questions/67131565/how-do-i-set-dt-rpath-or-dt-runpath

Specifying library paths by RPATH/RUNPATH (1-2)

- RPATH is deprecated, so normally
  -rpath adds a path to RUNPATH
- DT_RPATH is an old dynamic tag,
  DT_RUNPATH a new dynamic tag
- tools such as chrpath and patchelf can also
  create or modify RPATH / RUNPATH
  (DT_RPATH / DT_RUNPATH entry in .dynamic section)
  in any ELF file after compilation

https://stackoverflow.com/questions/67131565/how-do-i-set-dt-rpath-or-dt-runpath

Specifying library paths by RPATH/RUNPATH (1-3)

- ld option
  --disable-new-dtags
  - this linker can create the new dynamic tags in ELF.
  - But the older ELF systems may not understand them
  - if you specify --enable-new-dtags,
    the *new* dynamic tags will be created as needed
    and *older* dynamic tags will be omitted.
  - if you specify --disable-new-dtags,
    no new dynamic tags will be created.
  - by default, the *new* dynamic tags are not created.
  - Note that those options are only available for ELF systems.

https://stackoverflow.com/questions/67131565/how-do-i-set-dt-rpath-or-dt-runpath

Specifying library paths by RPATH/RUNPATH (2)

- The difference between `-rpath` and `-rpath-link`
  - directories specified by `-rpath` options
    are included in the executable
    and used at runtime
  - the `-rpath-link` option is only effective
    at link time.
- And the `ld` documentation
  also explains how `-rpath-link` works.
- It's to specify directories
  for searching dependent shared libraries.

`https://stackoverflow.com/questions/67131565/how-do-i-set-dt-rpath-or-dt-runpath`

Specifying library paths by RPATH/RUNPATH (3)

- The -rpath command line option used
  to add a path to a DT_RPATH entry in the .dynamic section,
- DT_RPATH was deprecated in favor of DT_RUNPATH
- modern linker versions use DT_RUNPATH instead.
  - using -rpath on a really old linker,
    you will modify dynamic section entry with .d_val = DT_RPATH,
  - but if your linker is up to date,
    you will modify with .d_val = DT_RUNPATH instead.

https://stackoverflow.com/questions/67131565/how-do-i-set-dt-rpath-or-dt-runpath

Specifying library paths by RPATH/RUNPATH (4)

- The `-rpath-link` option is an option
  which does not create any entry,
  but is used to supersede the DT_RUNPATH entry
  present in the dynamic section of a library
  that is being linked.
- Therefore, when compiling, you should usually not need it.

`https://stackoverflow.com/questions/67131565/how-do-i-set-dt-rpath-or-dt-runpath`

Specifying library paths by LD_RUN_PATH (1)

- Some systems have an LD_RUN_PATH environment variable too.
  - some have 32-bit and 64-bit variants
  - fiddly for users and installers alike;
  - how do you *ensure* the environment variable is set for everyone that uses your code?
  - an environment-setting shell script that then runs the real program can help here.

https://stackoverflow.com/questions/31455979/how-to-specify-libraries-paths-in-gc

Specifying library paths by LD_RUN_PATH (2)

- add the directory to the configuration file
  that specifies the list of known directories
  for the dynamic loader to search.
- platform specific
  - file name, format, location (usually under /etc somewhere)
    and mechanism used to edit it.
  - the file might be /etc/ld.so.conf.
  - there might well be a program to edit the config file correctly.

https://stackoverflow.com/questions/31455979/how-to-specify-libraries-paths-in-gcc

Specifying library paths by default library path

- install the libraries in a location
  that will be searched anyway
  - default library path
    - /usr/lib
    - /usr/local/lib
  - without reconfiguring the dynamic loader

https://stackoverflow.com/questions/31455979/how-to-specify-libraries-paths-in-gc

Specifying library paths by -R (1)

- On some systems,
  a -R option can be added to the command line
  to specify where libraries (shared objects)
  may be found at runtime :
- not all systems support this option.

```
$ gcc imagefilter.c -o imagefilter \
>     -I/home/savio/opencv-3.0.0/include/opencv \
>     -L/home/savio/opencv-3.0.0/cmake_binary_dir/lib \
>     -R/home/savio/opencv-3.0.0/cmake_binary_dir/lib \
>     -lopencv_imgcodecs -lopencv_imgproc \
>     -lopencv_highgui -lopencv_core
```

https://stackoverflow.com/questions/31455979/how-to-specify-libraries-paths-in-gc

Specifying library paths by -R (2)

- the disadvantage of this -R option is
  that the location you specify is embedded in the binary.
  - If the libraries on the customers' machines
    is not in the same place, the library won't be found.
  - Consequently, a path under someone's home directory
    is only appropriate for that user on their machines
  - not general
    if the software is installed by default
    in, say, /opt/packagename/lib,
    then specifying that with -R is probably appropriate.

https://stackoverflow.com/questions/31455979/how-to-specify-libraries-paths-in-gc

Dynamic linker `ld.so` library search order

Library search order

Library search order (1)

- The dynamic linker of the GNU C Library searches for shared libraries in the following locations in order:

  1. `DT_RPATH`
  2. `LD_LIBRARY_PATH`
  3. `DT_RUNPATH`
  4. `ldconfig` cache file
  5. default path `/lib` and then `/usr/lib`

- Failing to find the shared library in all these locations will raise the following error

  ```
  cannot open shared object file: No such file or directory
  ```

https://en.wikipedia.org/wiki/Rpath#+end_src

Library search order (2)

1. The (colon-separated) paths
   in the DT_RPATH dynamic section attribute of the binary
   - if present and
   - the DT_RUNPATH attribute does not exist

https://en.wikipedia.org/wiki/Rpath#+end_src

Library search order (3)

2. the (colon-separated) paths
   in the environment variable LD_LIBRARY_PATH,

   - LD_LIBRARY_PATH is ignored,
     if the executable is a setuid / setgid binary

   - LD_LIBRARY_PATH can be overridden
     if the dynamic linker is called
     with the option --library-path

     ```
     /lib/ld-linux.so.2 --library-path $HOME/mylibs myprogram
     ```

https://en.wikipedia.org/wiki/Rpath#+end_src

Library search order (4)

3. The (colon-separated) paths
   in the DT_RUNPATH dynamic section attribute of the binary
   - if present.

https://en.wikipedia.org/wiki/Rpath#+end_src

Library search order (5)

3. Lookup based on the `ldconfig cache file`
   (often located at `/etc/ld.so.cache`)

   - which contains a compiled list of candidate libraries
     previously found in the augmented library path
     (set by `/etc/ld.so.conf`).

   - if, however, the binary was linked
     with the `-z nodefaultlib` linker option,
     libraries in the default library paths are skipped

`https://en.wikipedia.org/wiki/Rpath#+end_src`

Library search order (6)

5. In the trusted default path `/lib`,
   and then `/usr/lib`.
   - if the binary was linked with the `-z nodefaultlib` linker option,
     this step is skipped

`https://en.wikipedia.org/wiki/Rpath#+end_src`

Detailed library search order

Detailed library search order (1)

1. Any directories specified by `-rpath-link` options.
2. Any directories specified by `-rpath` options.

- The difference between `-rpath` and `-rpath-link` is
  - that directories specified by `-rpath` options are included in the executable and used at runtime,
  - whereas the `-rpath-link` option is only effective at link time
  - Searching `-rpath` in this way is only supported by native linkers and cross linkers which have been configured with the `--with-sysroot` option.

`https://sourceware.org/binutils/docs-2.40/ld.pdf`

Detailed library search order (2)

3. On an ELF system, for native linkers,
   if the `-rpath` and `-rpath-link` options were not used,
   search the contents of the environment variable `LD_RUN_PATH`
4. On *SunOS*, if the `-rpath` option was not used,
   search any directories specified using `-L` options.
5. For a native linker, search the contents of
   the environment variable `LD_LIBRARY_PATH`

`https://sourceware.org/binutils/docs-2.40/ld.pdf`

Detailed library search order (3)

6. For a native ELF linker, the directories in DT_RUNPATH or DT_RPATH of a shared library are searched for shared libraries needed by it.
   - The DT_RPATH entries are ignored if DT_RUNPATH entries exist.
7. For a linker for a Linux system, if the file /etc/ld.so.conf exists, the list of directories found in that file.
   - Note: the path to this file is prefixed with the sysroot value, if that is defined, and then any prefix string, if the linker was configured with the --prefix=<path> option.

https://sourceware.org/binutils/docs-2.40/ld.pdf

Detailed library search order (4)

8. For a native linker on a *FreeBSD* system,
   any directories specified by the `_PATH_ELF_HINTS` macro
   defined in the `elf-hints.h` header file.
9. Any directories specified by a `SEARCH_DIR` command
   in a linker script given on the command line,
   including scripts specified by `-T` (but not `-dT`).
10. The default directories, normally `/lib` and `/usr/lib`

`https://sourceware.org/binutils/docs-2.40/ld.pdf`

Detailed library search order (5)

10. Any directories specified
    by a plugin `LDPT SET EXTRA LIBRARY PATH`
1. Any directories specified
    by a `SEARCH_DIR` command in a default linker script

`https://sourceware.org/binutils/docs-2.40/ld.pdf`

Link time library paths : -L and -l

Overview

- it is assumed
  `libdemo.so` : a shared library file
- `-L. -ldemo` provides
  - the name of the library file (`libdemo.so`)
  - the location of the library file (`.`)
- `ld`
  the name of the shared library is embedded
  in the executable
  (`.dynamic` section, dynamic dependencies, NEEDED)
- `ld.so` : the final linker
  actually fetches the shared libraries
  loads in the shared symbols / code / etc.

`https://stackoverflow.com/questions/28230983/gcc-l-command-confusion`

-ld and -ld.so

- linking is done by two different instances of *linker*
  - when you *compile* and *link* your program
    linker `ld` (`/usr/bin/ld`)
    - checks external references
    - builds your executable
      by adding external reference `libdemo.so`
  - when you *run* your program
    run-time linker `ld.so` (`/lib64/ld-linux-x86-64.so.2`)
    - loads all needed *shared objects*

`https://stackoverflow.com/questions/28230983/gcc-l-command-confusion`

-L *path* is not saved

- the reasons why -L *path* is not saved
  - `libdemo.so` is not necessarily located
    at the same path where it was compiled
    - you could copy your binary unto another host
    - that path was internal build path, etc
  - it may be unsafe to save -L *path*
    - `ld.so` ususally seeks over list of trusted paths
      where non-root users cannot write

`https://stackoverflow.com/questions/28230983/gcc-l-command-confusion`

To provide the name of the library and the location

- since the executable file
  does not *contains* copies of the shared object files,
  it needs some way to *identify* the *necessary* shared library
  - during the link, only the name of the shared library
    is embedded in the executable
    (`.dynamic` section, dynamic dependencies, `NEEDED`)
    but the specific location is not yet specified.
  - So the `-L. -ldemo` is really just to provide
    the name of the library file (`libdemo.so`) and
    the location (`.`)

`https://stackoverflow.com/questions/28230983/gcc-l-command-confusion`

-l and -L (4)

- -Ldir adds directory dir
  to the list of directories
  to be searched for -l

- -ldemo is only to provide
  the name of the library file

https://stackoverflow.com/questions/28230983/gcc-l-command-confusion

-l and -L (5)

- `-L. -ldemo` is not required when using the `-rpath`
  - because in `-rpath dir` command,
    the name of the library `libdemo.so` is passed directly

    (`.dynamic` section, dynamic dependencies, `NEEDED`)
    (`.dynamic` section, rpath, `DT_RPATH`)
  - otherwise specifying it with `-L. -ldemo` is necessary.
- The run-time library path is subsequently provided
  to specify the exact location at the time of execution

`https://stackoverflow.com/questions/28230983/gcc-l-command-confusion`

-l and -L (6)

- in some cases, saving -L is useful
  when software installed into /opt
- therefore RPATH was introduced

https://stackoverflow.com/questions/28230983/gcc-l-command-confusion

-l and -L (7)

- if -rpath is used, -L is not needed
- rpath=dir adds a directory to the runtime library search path
- used when linking an ELF executable with shared objects.
- all arguments are concatenated and
  passed to the runtime linker,
  which uses them to locate shared objects at runtime

https://stackoverflow.com/questions/28230983/gcc-l-command-confusion

Run time library resolution : LD_LIBRARY_PATH

LD_LIBRARY_PATH (1)

- the predefined environmental variable
- contains the paths
  which the linker should look into
- in order to link
  shared / dynamic libraries
- a colon separated list of paths
- which the dynamic loader
  should look for shared libraries

https://stackoverflow.com/questions/7148036/what-is-ld-library-path-and-how-to-use

LD_LIBRARY_PATH (2)

- the standard library paths
  /lib and /usr/lib
- the paths in LD_LIBRARY_PATH have
  higher priority than
  the standard library paths
  - the standard paths will still be searched,
    but *only after* the paths in LD_LIBRARY_PATH
    have been searched

https://stackoverflow.com/questions/7148036/what-is-ld-library-path-and-how-to-use

LD_LIBRARY_PATH (3)

- The best way to use LD_LIBRARY_PATH is to set it on the command line or script *immediately before* executing the program.
- this way the new LD_LIBRARY_PATH isolated from the rest of your system.
- Example:

```
$ export LD_LIBRARY_PATH="/list/of/library/paths:/another/path"
$ ./program
```

https://stackoverflow.com/questions/7148036/what-is-ld-library-path-and-how-to-use

LD_LIBRARY_PATH and -L (3)

- LD_LIBRARY_PATH has the side-effect of *altering*
  - the way gcc and ld behave
  - the way the the run-time linker behaves

  by modifying the search path.
- LD_LIBRARY_PATH *affects* this search path implicitly (sometimes not a good thing)

https://stackoverflow.com/questions/1904990/what-is-the-difference-between-ld-libr

LD_LIBRARY_PATH and -L (4)

- without using LD_LIBRARY_PATH on most Linux systems
  - to *add* the path that contains your shared libraries
    in /etc/ld.so.conf file
  - create a file in /etc/ld.so.conf.d/
    with the path in it
  - run ldconfig (/sbin/ldconfig as root)
    to update the runtime linker bindings cache.

```
$ cat ld.so.conf
include /etc/ld.so.conf.d/*.conf

$ ls
fakeroot-x86_64-linux-gnu.conf  libc.conf
i386-linux-gnu.conf             x86_64-linux-gnu.conf
```

https://stackoverflow.com/questions/1904990/what-is-the-difference-between-ld-libr

LD_LIBRARY_PATH and -L (5)

```
$ cat fakeroot-x86_64-linux-gnu.conf
/usr/lib/x86_64-linux-gnu/libfakeroot

$ cat libc.conf
# libc default configuration
/usr/local/lib

$ cat i386-linux-gnu.conf
# Multiarch support
/usr/local/lib/i386-linux-gnu
/lib/i386-linux-gnu
/usr/lib/i386-linux-gnu
/usr/local/lib/i686-linux-gnu
/lib/i686-linux-gnu
/usr/lib/i686-linux-gnu

$ cat x86_64-linux-gnu.conf
# Multiarch support
/usr/local/lib/x86_64-linux-gnu
/lib/x86_64-linux-gnu
/usr/lib/x86_64-linux-gnu
```

https://stackoverflow.com/questions/1904990/what-is-the-difference-between-ld-libr

LD_LIBRARY_PATH and -L (6)

- when the program is executed, the run-time linker
  will look in those directories for libraries
  that your binary has been linked against.

```
Example on Debian:

jewart@dorfl:~$ cat /etc/ld.so.conf.d/usrlocal.conf
/usr/local/lib
```

https://stackoverflow.com/questions/1904990/what-is-the-difference-between-ld-libr

LD_LIBRARY_PATH and -L (7)

- If you want to know what libraries
  the run-time linker knows about,
  you can use:

```
$ ldconfig -v

/usr/lib:
libbfd-2.18.0.20080103.so -> libbfd-2.18.0.20080103.so
libkdb5.so.4 -> libkdb5.so.4.0
libXext.so.6 -> libXext.so.6.4.0
```

https://stackoverflow.com/questions/1904990/what-is-the-difference-between-ld-libr

LD_LIBRARY_PATH and -L (8)

- And, if you want to know what libraries
  a binary is linked against,
  you can use ldd like such,
  which will tell you which library
  your runtime linker is going to choose:

```
$ ldd /bin/ls
linux-vdso.so.1 =>  (0x00007fffda1ff000)
librt.so.1 => /lib/librt.so.1 (0x00007f5d2149b000)
libselinux.so.1 => /lib/libselinux.so.1 (0x00007f5d2127f000)
libacl.so.1 => /lib/libacl.so.1 (0x00007f5d21077000)
libc.so.6 => /lib/libc.so.6 (0x00007f5d20d23000)
```

https://stackoverflow.com/questions/1904990/what-is-the-difference-between-ld-libr