STATIC AND SHARED LIBRARIES

CONTENT

- Object libraries
- Static libraries
- Linking process
- Circular dependency
- Linker flags to solve
- Overview shared libraries
- Create and use shared libraries
- Shared lib soname
- Useful tool for working with static AND shared libs
- Finding Shared Libraries at Run Time

OBJECT LIBRARIES

- Object libraries
  - Compile each source file and produce corresponding object file
  - Link all these files with main to create the execution program
  - Or group these files and created object libraries
  - Two kind of types object libraries as static and shared

```
$ cc -g -c prog.c mod1.c mod2.c mod3.c
$ cc -g -o prog_nolib prog.o mod1.o mod2.o mod3.o
```

STATIC LIBRARIES

- Archives
  - Building a single library file from multiple executables file object without recompile original source file
  - Can see contents static lib and reserve static lib back to file object
  - Static libraries has name form as libname.a

```
$ ar options archive object-file...
```

- Option
  - r – insert an object file into the archive, replace previous object file if has same name
  - t – table of content, display content of archives
  - d – delete a module from a archives

```
$ cc -g -c mod1.c mod2.c mod3.c
$ ar r libdemo.a mod1.o mod2.o mod3.o
$ rm mod1.o mod2.o mod3.o
```

```
$ ar tv libdemo.a
rw-r--r-- 1000/100 1001016 Nov 15 12:26 2009 mod1.o
rw-r--r-- 1000/100 406668 Nov 15 12:21 2009 mod2.o
rw-r--r-- 1000/100  46672 Nov 15 12:21 2009 mod3.o
```

```
$ ar d libdemo.a mod3.o
```

USE STATIC LIBRARIES

- use static lib
  - Give name of static lib in link command
  - Place to standard directories searches by linker and then specify the libraries name use –l libname option
  - Place somewhere and directive the linker to by using –L /path_to_dir option

```
$ cc -g -c prog.c
$ cc -g -o prog prog.o libdemo.a
```

```
$ cc -g -o prog prog.o -ldemo
```

```
$ cc -g -o prog prog.o -Lmylibdir -ldemo
```

LINKING PROCESS

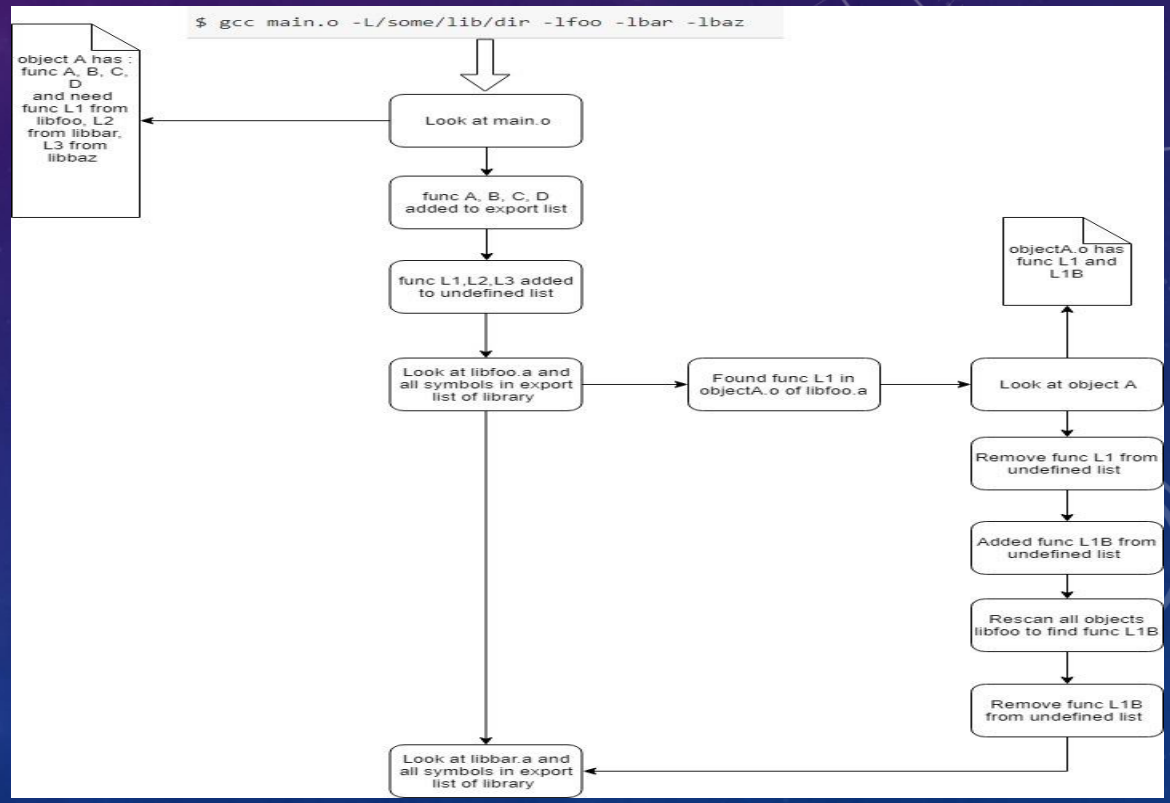
```
$ gcc main.o -L/some/lib/dir -lfoo -lbar -lbaz
```

- Linker - ld
  - Invoked through gcc
  - Object file and libraries provided in a certain order on command-line from left to right
  - Linker maintain a symbol table that has two list
    - A list exported by all the objects and libraries encountered so far
    - A list of undefined symbols that the encountered object and libraries requested to import and not found yet
  - When the linker encounter a new object file, it looks at
    - The symbol it exports : these added to the list of exported symbol
    - If any symbol in undefined list, it removed from there because it has been found, if any symbol in exported list, it terminate with multiple definition error : two different object export the same simple and linker confused
    - The symbol it import , it added to the list of undefined symbols, unless it can be found in list of exported symbol
  - When the linker encounter a new libraries, it first look at the symbol it export
    - If any the symbol it exports are on undefined list, the object is added to the link and the next step executed, otherwise the next step skipped
    - If object added to link, it treat as described above, its undefined and export symbols added to symbol tables
    - Finally, if any of the objects in the library has been included in the link, the library rescanned again, it is possible that symbols imported by included object can be found in other objects within the libraries
  - When the linking finished, it lookup the symbol table. If any symbols remain in the undefined list, the linker throw an “undefined reference” error

```
/usr/lib/x86_64-linux-gnu/crt1.o: In function '_start':
(.text+0x20): undefined reference to 'main'
collect2: ld returned 1 exit status
```

LINKING PROCESS (2)

- After linker looked at a library, it won't look up again. Event if it exports symbol that may be need by some later library
- As an library is examined, an object file within it can be lelf out of the link if it does provide symbols the symbol table need
- An object file use function strlen, linker will load strlen.o will be taken into the link from libc.a, this called an-object-per-function and keep the execution bin small
- If object or library AA need a symbol from library BB then AA should come before library BB in the command line invocation of linker



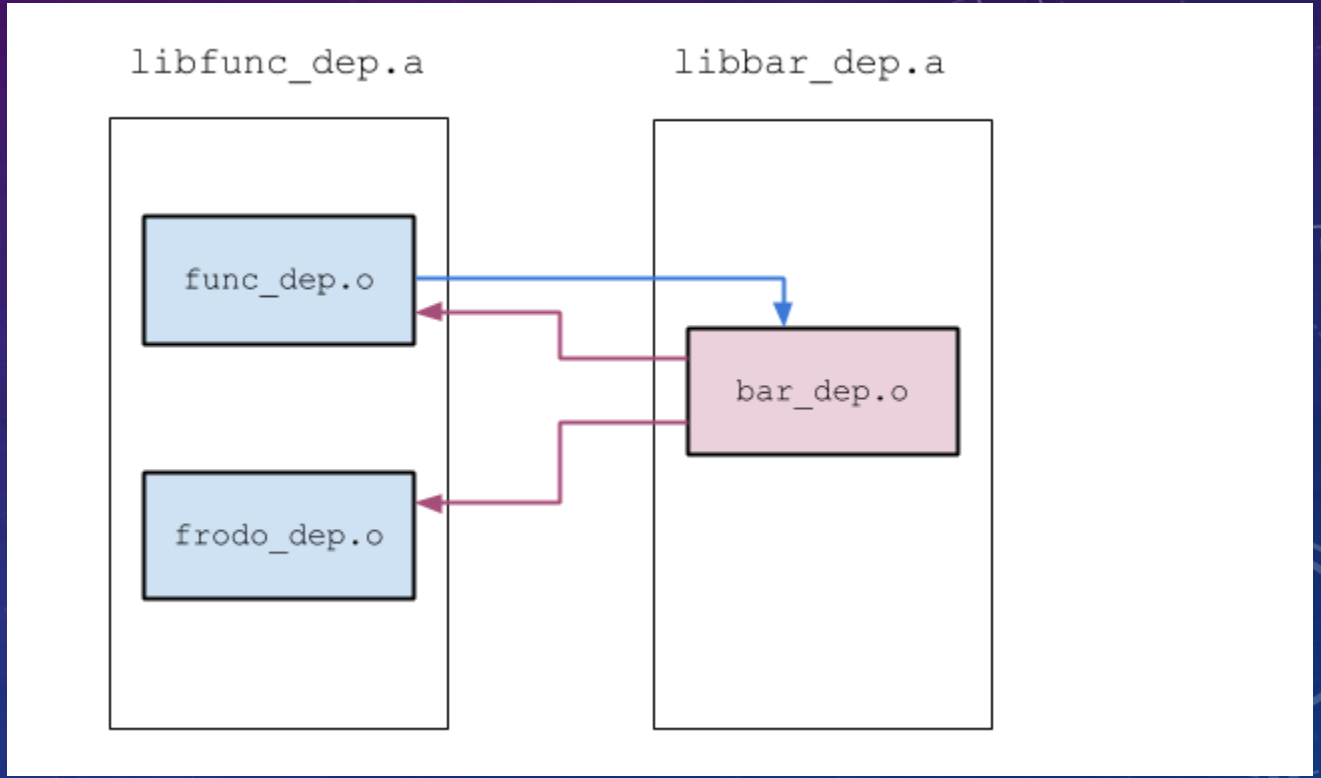
CIRCULAR DEPENDENCY

```
$ cat func_dep.c
int bar(int);

int func(int i) {
    return bar(i + 1);
}
$ cat bar_dep.c
int func(int);

int bar(int i) {
    if (i > 3)
        return i;
    else
        return func(i);
}
```

- what happens if AA needs a symbol from BB, but BB also needs a symbol from AA

```
$ gcc  -L. simplemain.o -lfunc_dep -lbar_dep
./libbar_dep.a(bar_dep.o): In function 'bar':
bar_dep.c:(.text+0x17): undefined reference to 'frodo'
collect2: ld returned 1 exit status
$ gcc  -L. simplemain.o -lbar_dep -lfunc_dep
./libfunc_dep.a(func_dep.o): In function 'func':
func_dep.c:(.text+0x14): undefined reference to 'bar'
collect2: ld returned 1 exit status
```

```
$ gcc  -L. simplemain.o -lfunc_dep -lbar_dep -lfunc_dep
$ ./a.out ; echo $?
24
```



LINKER FLAGS TO SOLVE

- --start-group archive –end-group
  - Linker research all lib to find all symbols in undefined list
  - Cost to performance but it cheaps

```
$ gcc simplemain.o -L. -Wl,--start-group -lbar_dep -lfunc_dep -Wl,--end-group
$ ./a.out ; echo $?
24
```

OVERVIEW SHARED LIBRARIES

- Disadvantages of static lib
  - Disk space wasted to storing multiple copies of the same object modules
  - If several difference programs using the same modules are running at the same time, each hold a copies of object modules in memory, it increase RAM program consumes
  - If a change require in lib, all executables use that module must be relinked
  -

- Advantages and disadvantages of shared lib
  - Program size is small but program use shared lib take longer to start
  - Because separation between program and shared lib, then when function in lib change, we don’t need relink program, just update shared lib but believe me, something it is inconvenient for programmer
  - Shared lib more complex than static and require more registers to run, it also require object file compile with Position Independent Code
  - Symbol relocation must be performed in run time

CREATE AND USE SHARED LIBRARIES

```
$ gcc -g -c -fPIC -Wall mod1.c mod2.c mod3.c
$ gcc -g -shared -o libfoo.so mod1.o mod2.o mod3.o
```

- Use option –fPIC to build object file and –shared to build shared lib
- Shared lib has prefix lib and the suffix .so
- Unlike static lib, it is not possible to add and remove individual object from a previously built shared library

- Position Independent Code
  - This option change the way the compiler generates code for operations (access global, static variables …), allow the code to be located at any virtual address at run time
  - This is necessary for shared mem since there no way to know where shared lib code will be located in memory
  - Verify by check GLOBAL_OFFSET_TABLE_

```
$ nm mod1.o | grep _GLOBAL_OFFSET_TABLE_
$ readelf -s mod1.o | grep _GLOBAL_OFFSET_TABLE_
```

USE A SHARED LIBRARIES

- A program maintain a shared library dependencies as dynamic dependency list
- At runtime dynamic linker will find corresponding shared library file for program
  - In linux its name /lib/ld-linux.so.2
  - Dynamic linker exanimate dynamic dependency list and find shared lib in /usr/lib or /lib or user-defined directory
  - User-defined directory shared lib notify by LD_LIBRARY_PATH

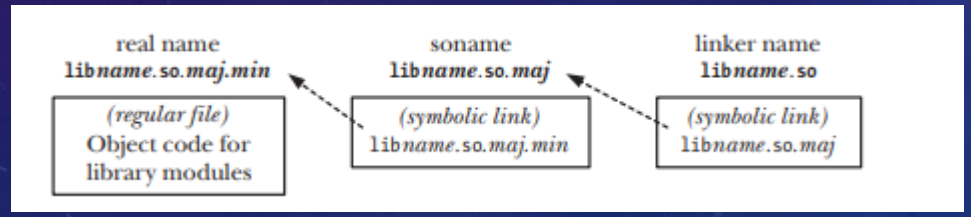
```
$ gcc -g -Wall -o prog prog.c libfoo.so
```

```
$ ./prog
./prog: error in loading shared libraries: libfoo.so: cannot
open shared object file: No such file or directory
```

```
$ LD_LIBRARY_PATH=. ./prog
Called mod1-x1
Called mod2-x2
```

SHARED LIB SONAME

- Realname vs soname and vs linkername
  - Soname as alias realname and use for linker
  - Option –soname pass to linker
  - Check soname with objectdump and readelf
  - Create soft-link between real-name and soname




```
$ gcc -g -c -fPIC -Wall mod1.c mod2.c mod3.c
$ gcc -g -shared -Wl,-soname,libbar.so -o libfoo.so mod1.o mod2.o mod3.o
```

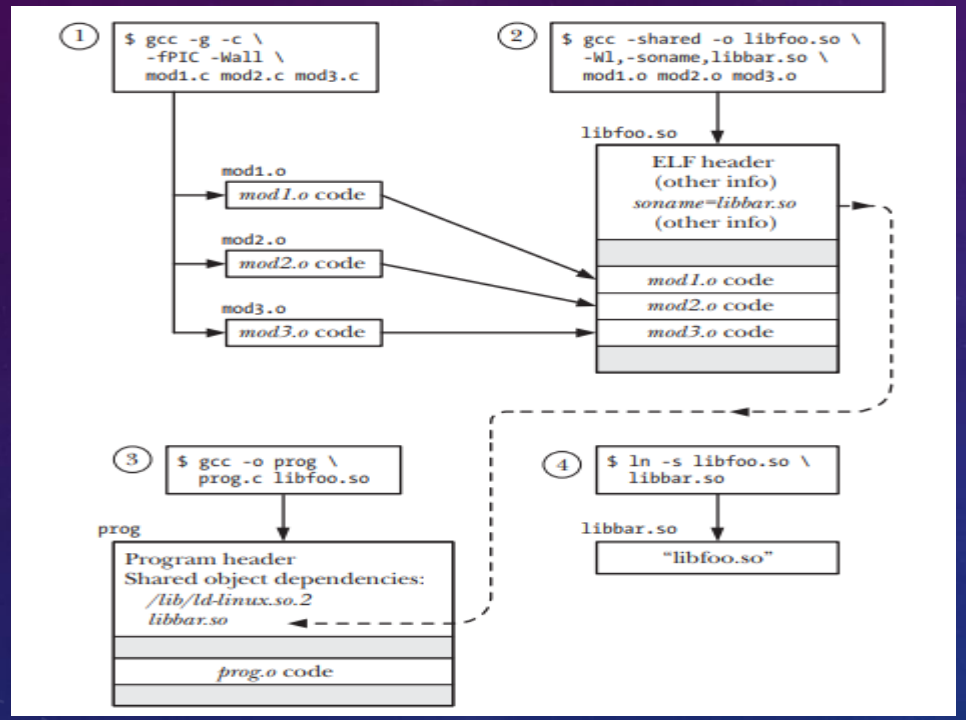
```
$ objdump -p libfoo.so | grep SONAME
  SONAME       libbar.so
$ readelf -d libfoo.so | grep SONAME
 0x0000000e (SONAME)       Library soname: [libbar.so]
```

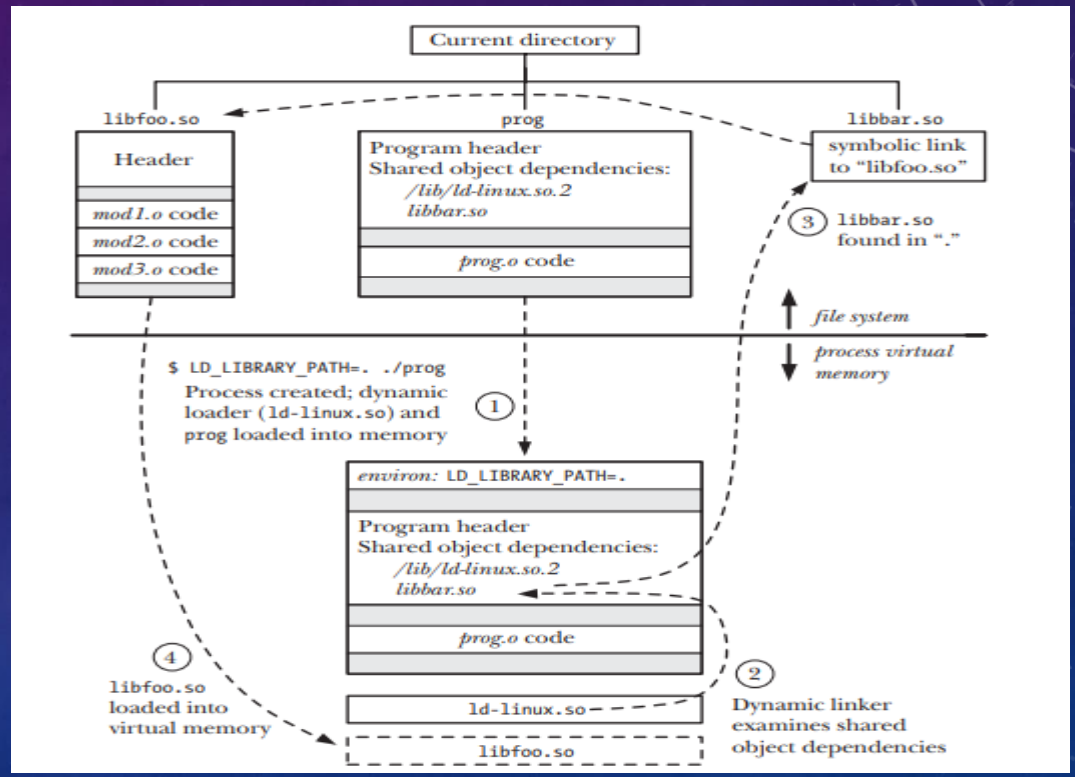
```
$ gcc -g -Wall -o prog prog.c libfoo.so
```

```
$ LD_LIBRARY_PATH=. ./prog
prog: error in loading shared libraries: libbar.so: cannot open
shared object file: No such file or directory
```

```
$ ln -s libfoo.so libbar.so        Create soname symbolic link in current directory
$ LD_LIBRARY_PATH=. ./prog
Called mod1-x1
Called mod2-x2
```

SHARE LIB LINKING PROCESS AND EXECUTION OF PROGRAM THAT LOAD SHARED LIB REVIEW





USEFUL TOOL FOR WORKING WITH STATIC AND SHARED LIBS

- ldd command
  - List dynamic dependencies and all shared lib that a program required to run
- objdump and readelf command
  - To find symbol undefined ref
  - Merge two static libs into one
  - Provide a lot of information
- nm command
  - To find where is one function located
  - Provide a lot of information

```
thientran@ubuntu:~/Desktop/training/static_shared_lib$ ldd prog_fdman
        linux-gate.so.1 =>  (0x0035f000)
        libc.so.6 => /lib/tls/i686/cmov/libc.so.6 (0x0062b000)
        /lib/ld-linux.so.2 (0x00185000)
```

```
thientran@ubuntu:~$ nm -A /usr/lib/lib* 2> /dev/null | grep ' crypt$'
/usr/lib/libcrypt.a:crypt-entry.o:00000180 T crypt
```

SHARED LIBRARY VERSIONS AND NAMING CONVENTIONS

- Minor versions and major version
  - Differ but compatible or incompatible
    - The semantics of each public function and variables unchanges
      - Same argument list
      - Same effect and result to global variables
      - Same result value of all funcs
    - No have func or variables is removed
    - Structures allocated and exported unchanged but some cases we can added some additional field to structures
  - Form libname.so.major-id.minor-id format

```
libdemo.so.1.0.1
libdemo.so.1.0.2        Minor version, compatible with version 1.0.1
libdemo.so.2.0.0        New major version, incompatible with version 1.*
libreadline.so.5.0
```

```
libdemo.so.1        -> libdemo.so.1.0.2
libdemo.so.2        -> libdemo.so.2.0.0
libreadline.so.5    -> libreadline.so.5.0
```

INSTALL AND UPDATE SHARE LIB

- Standard libraries directory
  - /usr/lib - Most standard libraries directory
  - /lib - Use for system startup
  - /usr/local/lib - Nonstandard
  - LD_PATH_LIBRARY
- ldconfig command
  - Build /etc/ld.so.cache from search a standard set of directory /usr/lib, /lib and dir in /etc/ld.so.conf
  - Exminates the lastest major/minor version to find embedded soname and create relative symbollink for each soname
  - We should run ldconfig each time we update new shared lib
  - Ldconfig –p to show cache and –v to show command output

```
$ su
Password:
# mv libdemo.so.1.0.1 /usr/lib

# cd /usr/lib
# ln -s libdemo.so.1.0.1 libdemo.so.1
# ln -s libdemo.so.1 libdemo.so
```

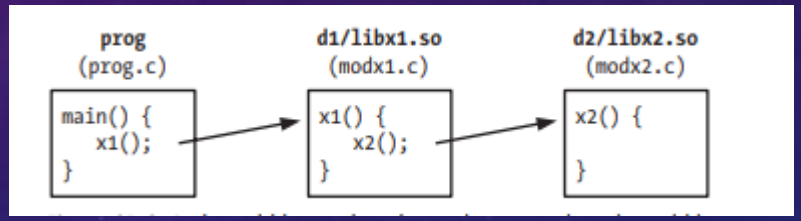
SPECIFYING LIBRARY SEARCH DIRECTORIES BY RPATH

```
$ cd /home/mtk/pdir/d2
$ gcc -g -c -fPIC -Wall modx2.c
$ gcc -g -shared -o libx2.so modx2.o
```

```
$ cd /home/mtk/pdir/d1
$ gcc -g -c -Wall -fPIC modx1.c
$ gcc -g -shared -o libx1.so modx1.o -Wl,-rpath,/home/mtk/pdir/d2 \
          -L/home/mtk/pdir/d2 -lx2
```

```
$ cd /home/mtk/pdir
$ gcc -g -Wall -o prog prog.c -Wl,-rpath,/home/mtk/pdir/d1 \
            -L/home/mtk/pdir/d1 -lx1
```

```
$ objdump -p prog | grep PATH
  RPATH       /home/mtk/pdir/d1          libx1.so will be sought here at run time
$ objdump -p d1/libx1.so | grep PATH
  RPATH       /home/mtk/pdir/d2          libx2.so will be sought here at run time
```

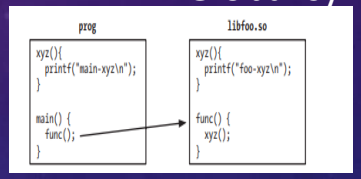



```
$ ldd prog
        libx1.so => /home/mtk/pdir/d1/libx1.so (0x40017000)
        libc.so.6 => /lib/tls/libc.so.6 (0x40024000)
        libx2.so => /home/mtk/pdir/d2/libx2.so (0x4014c000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

FINDING SHARED LIBRARIES AT RUN TIME

- Dynamic linker searches for the shared lib using following order:
  - DT_RPATH (rpath) first
  - LD_LIBRARY_PATH
  - DT_RUNPATH
  - /etc/ld.so.cache
  - /lib and /usr/lib

- Runtime symbol resolution
  - Global symbol reference




```
$ gcc -g -c -fPIC -Wall -c foo.c
$ gcc -g -shared -o libfoo.so foo.o
$ gcc -g -o prog prog.c libfoo.so
$ LD_LIBRARY_PATH=. ./prog
main-xyz
```

  - A definition of a global symbol in the main program override a definition in a library
  - If a global symbol is defined in multiple libraries, then a reference to symbol is bound to first definition found by scanning libraries in the lelf-to-right order in which they were listed on static link command line