Dynamic Linking

TA : Sharath Gopal

'C/C++' Compilation Process

- Preprocessor
  - Expand header includes, macros, etc
  - -E option in gcc to show the resulting code
- Compiler
  - Generates machine code for certain architecture
- Linker
  - Links all modules together
  - Address resolution
- Loader
  - Loads the executable to memory to start execution

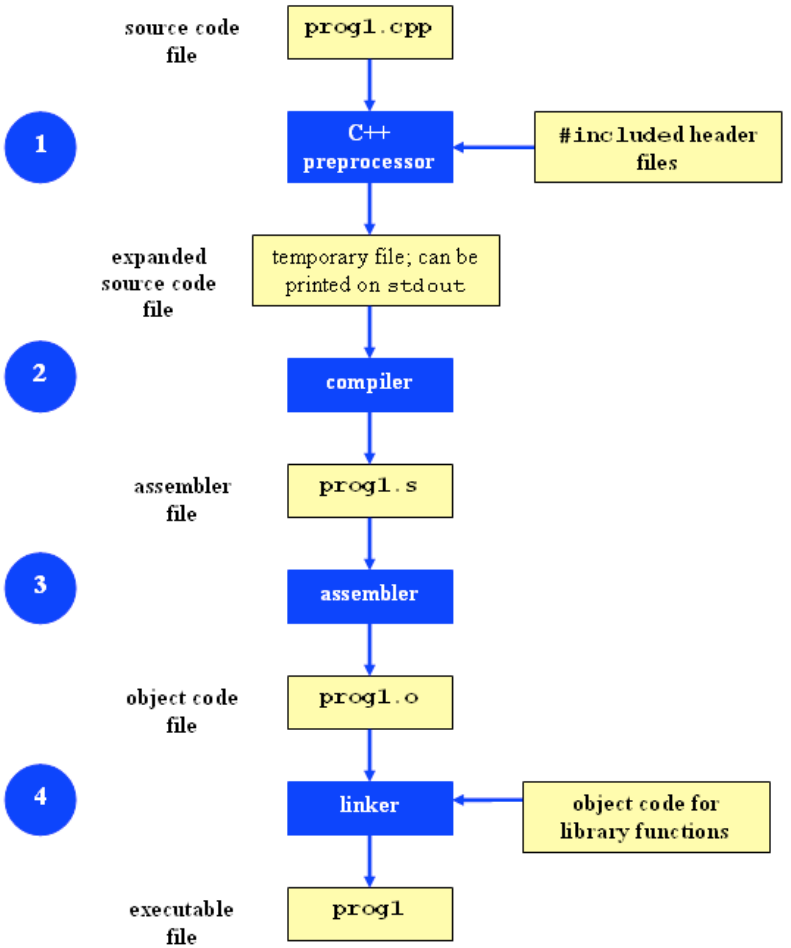
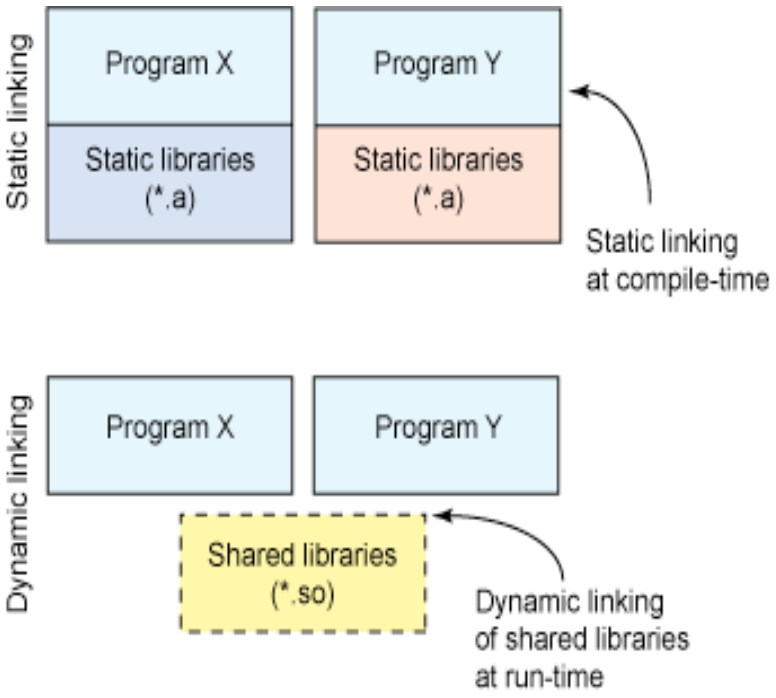


Image source - http://faculty.cs.niu.edu/~mcmahon/CS241/Notes/compile.html

Linux Libraries

- **Static Library**
  - Statically linked
  - Every program has its own copy
  - More space in memory
  - Tied to a specific version of the lib. New version of the lib requires recompile of source code.
- **Shared Library (binding at run-time)**
  - Dynamically loaded/linking
    - **Dynamic Linking** – The OS loads the library when needed. A dynamic linker does the linking for the symbol used.
    - **Dynamic Loading** – The program "actively" loads the library it needs (DL API – dlopen(), dlclose()). More control to the program at run-time. Permits extension of programs to have new functionality.
  - Library is shared by multiple programs
  - Lower memory footprint
  - New version of the lib does not require a recompile of source code using the lib



Img Source : http://www.ibm.com/developerworks/library/l-dynamic-libraries/

`extern` and `static` qualifiers

- Declaration Vs Definition
  - Definition allocates memory, whereas a declaration just tells the compiler about the existence of a function or a variable.
- `extern`
  - Functions – Functions are 'extern' by default.
  - Variables – `extern int j;`
    - This is just a declaration, not a definition. So, no memory is allocated for 'j' by this line.
- `static` (Limiting visibility)
  - Functions – Static function is visible only in that file, and hence cannot be invoked from outside the file it is defined in.
  - Global Variables – Variable is visible just in that file, not outside.
  - What happens if you have static on local variables?

Dynamic Linking/Loading tools

ELF – Executable and Linking format

- Format of executables, shared libraries, object code on Linux

- **ldd** – Shows shared library dependencies
  - libdl.so (the DL API), the GNU C library (libc.so) ,etc
- **readelf** – parsing ELF files
- **objdump** – Info about object files
- **nm** – Lists symbols from object files

GCC options

- -c : compile and create object files
- -o : name of output file
- -I (upper case i) : Additional folders to search for header files
- -L : Additional folders to search for libraries to link with
- -shared : create shared library
- -l (lower case 'ell') : Name of additional library to link with
- -fpic : Output position independent code. This is required for shared libraries (generating relative addresses instead of absolute addresses)

Creating static and shared libs in GCC

- mymath.h

```
#ifndef __MY_MATH_H__
#define __MY_MATH_H__
void mul5(int *i);
void add1(int *i);
#endif
```

- mul5.c

```
#include "mymath.h"
void mul5(int *i)
{
  *i *= 5;
}
```

- add1.c

```
#include "mymath.h"
void add1(int *i)
{
  *i += 1;
}
```

- gcc -c mul5.c -o mul5.o
- gcc -c add1.c -o add1.o
- ar -cvq libmymath.**a** mul5.o add1.o ----> (static lib)
- gcc **-shared** -fpic -o libmymath.**so** mul5.o add1.o -----> (shared lib)

Dynamic loading

```c
#include<stdio.h>
#include<dlfcn.h>

int main(int argc, char* argv[])
{
  int i = 10;
  void (*myfunc)(int *);
  void *dl_handle;
  char *error;
  dl_handle = dlopen("libmymath.so", RTLD_LAZY);//RTLD_NOW
  if(!dl_handle) {
    printf("dlopen() error - %s\n", dlerror());
    return 1;
  }
  //Calling mul5(&i);
  myfunc = dlsym(dl_handle, "mul5");
  error = dlerror();
  if(error != NULL) {
    printf("dlsym mul5 error - %s\n", error);
    return 1;
  }
  myfunc(&i);

  //Calling add1(&i);
  myfunc = dlsym(dl_handle, "add1");
  error = dlerror();
  if(error != NULL) {
    printf("dlsym add1 error - %s\n", error);
    return 1;
  }
  myfunc(&i);
  printf("i = %d\n", i);

  dlclose(dl_handle);
  return 0;
}
```

- Copy the code into main.c
- gcc main.c -o main -ldl
- You will have to set the environment variable LD_LIBRARY_PATH to include the path that contains libmymath.so

Resources

- http://www.ibm.com/developerworks/library/l-dynamic-libraries/
- http://www.ibm.com/developerworks/library/l-lpic1-102-3/
- http://tldp.org/HOWTO/Program-Library-HOWTO/index.html
- http://www.yolinux.com/TUTORIALS/LibraryArchives-StaticAndDynamic.html
- http://man7.org/linux/man-pages/man7/vdso.7.html