Week 08M: Dynamic Linking

Thuy Vu

Reminders

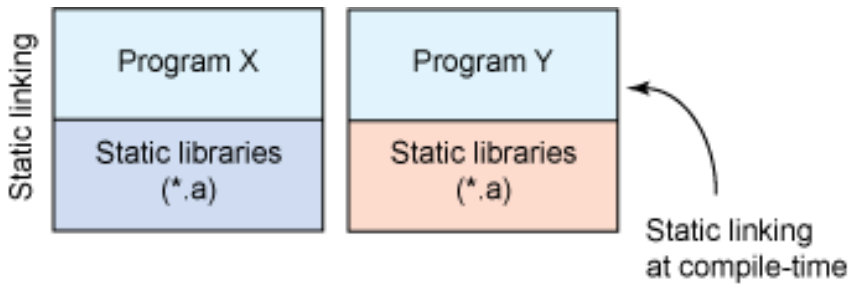
- Assignment 7: SSH setup and use in applications
  - `web.cs.ucla.edu/classes/winter17/cs35L/assign/assign7.html`
  - Time due 23:55 this Friday, March 3
- Assignment 8: Dynamic linking
  - `web.cs.ucla.edu/classes/winter17/cs35L/assign/assign8.html`
  - Time due 23:55 this Friday, March 10
- Assignment 9: Change management
  - `web.cs.ucla.edu/classes/winter17/cs35L/assign/assign9.html`
  - Time due 23:55 this Friday, March 17
- Assignment 10: Presentation

Anatomy of Linux Dynamic Libraries

1. **Libraries** – to package similar functionality → modular programming
2. Linux supports **two types**

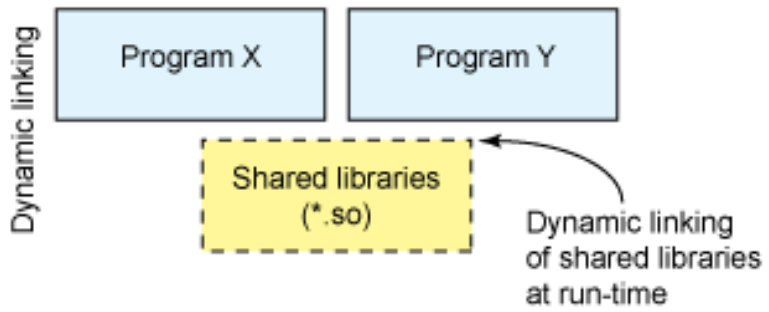
static library

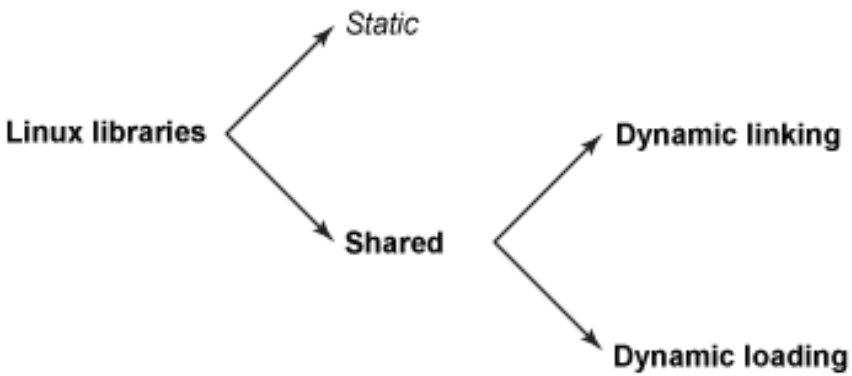
- functionality to bind to a program statically at compile-time



dynamic library

- functionality to bind to a program dynamically at run-time





- **dynamic linking** – have Linux load the library upon execution
- **dynamic loading** – selectively call functions with the library in a process

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

Dynamic Loading

to let an application load and link libraries itself

- application can specify a particular library to load, then
- application can call functions within that library

load shared libraries from disk (file) into memory and re-adjust its location

- done by a library named `ld-linux*.so.2`
  1. load-time relocation
  2. position independent code (PIC)

the Dynamic Loading API

1. `dlopen` – makes an object file accessible to a program

   ```
   void *dlopen( const char *file, int mode );
   ```

   - `RTLD_NOW` → relocate now; `RTLD_LAZY` → to relocate when needed;
2. `dlsym` – gives resolved address to a symbol within this object

   ```
   void *dlsym( void *restrict handle, const char *restrict name );
   ```

   - check `char *dlerror();` if an error occurs
3. `dlerror` – returns a string error of the last error that occurred
4. `dlclose` – closes an object file

   ```
   char *dlclose( void *handle );
   ```

the lab – to find out which programs linked to which libraries

1. a C program computes `cos(0.5)` and prints using `%.17g` on the SEASnet GNU/Linux servers
2. `ldd` to find which dynamic libraries a program uses
3. `strace` to find which system calls a program makes
4. automate the process (?)

the homework – to split an application into dynamically linked modules
`randall.c = randcpuid.c + randlibhw.c + randlibsw.c + randmain.c`

1. `randall.c =`
   `randcpuid.c + randlibhw.c + randlibsw.c + randmain.c`

```
gcc -shared -fPIC greeting-fr.c -o greeting-fr.so
gcc -ldl -Wl,-rpath=.  greeting-dl.c -o greet-dl
```

1. `-shared` to build a shared object
2. `-fPIC` to output position independent code
3. `-lmylib` to link with "libmylib.so" (`-ldl` is what?)
   - `-L` to find `.so` files from this path, default is `/usr/lib`
   - `-Wl,rpath=dir` to set **rpath** to **dir** to linker (by using `-Wl`)

`attribute` of functions

- `__attribute__ ((__constructor__))` to run when `dlopen()` is called
- `__attribute__ ((__destructor__))` to run when `dlclose()` is called
- `__attribute__ ((__destructor__))`
  `void to_run_before_open(void) { ... }`

Dynamic Loading – An Example

1. build the libraries
2. load the libraries
3. run the functions in libraries