What Is A Library?

- A library is a packaged collection of object modules that is included as a unit as needed in a linked program.

What Is A Library?

- A library is a packaged collection of object modules that is included as a unit as needed in a linked program.
- Fundamentally, no more than a set of object modules.
  - Links within these modules are statically resolved.
  - Usually also contains some kind of added directory information to make it faster to search.

What Is A Library?

- A library is a packaged collection of object modules that is included as a unit as needed in a linked program.
- Fundamentally, no more than a set of object modules.
  - Links within these modules are statically resolved.
  - Usually also contains some kind of added directory information to make it faster to search.
- Static libraries
  - A collection of concatenated relocatable object modules, with a header describing the size and location of each module.
  - Identified by filename extension `.a` (“archive”) on Linux.

What Is A Library?

- A library is a packaged collection of object modules that is included as a unit as needed in a linked program.
- Fundamentally, no more than a set of object modules.
  - Links within these modules are statically resolved.
  - Usually also contains some kind of added directory information to make it faster to search.
- Static libraries
  - A collection of concatenated relocatable object modules, with a header describing the size and location of each module.
  - Identified by filename extension `.a` ("archive") on Linux.
- Shared libraries
  - An object module *that can be loaded at an arbitrary memory address* and linked (using a dynamic linker) with a program in memory at either load time or run time.
  - Identified by filename extension `.so` ("shared object") on Linux.

What Is Shared

- There is only one copy of the `.so` file for the entire system.
  - This is unlike static libraries, which are copied, embedded into, and linked with each executable that calls them.

What Is Shared

- There is only one copy of the `.so` file for the entire system.
  - This is unlike static libraries, which are copied, embedded into, and linked with each executable that calls them.

- There is a single copy of the `.text` section of the shared library in physical memory, which is shared by all executables that reference them contemporaneously.
  - This effect is accomplished by mapping these (read-only) physical pages into the virtual address spaces of these processes.
- The `.data` section of the shared library must of course be replicated, one for each executable.

What Is Shared

- There is only one copy of the `.so` file for the entire system.
  - This is unlike static libraries, which are copied, embedded into, and linked with each executable that calls them.

- There is a single copy of the `.text` section of the shared library in physical memory, which is shared by all executables that reference them contemporaneously.
  - This effect is accomplished by mapping these (read-only) physical pages into the virtual address spaces of these processes.
- The `.data` section of the shared library must of course be replicated, one for each executable.

- The requirement that the shared library be loadable at an arbitrary memory address means that all code in the library must be position-independent.

Binding Time Choices for Libraries

1. Static linker fully links library with application object modules.
   - All references resolved by static linker.
   - Loaded by OS loader as part of monolithic executable image.

Binding Time Choices for Libraries

1. Static linker fully links library with application object modules.
   - All references resolved by static linker.
   - Loaded by OS loader as part of monolithic executable image.
2. Static linker uses library information, but only links it partially.
   1. Loaded by OS loader as part of `fork()/exec()`.
      1. References resolved in bulk before program execution starts.
      2. References resolved individually at points of first use.
   2. Loaded by dynamic linker, triggered by first use.
      - References resolved individually at points of first use.

Binding Time Choices for Libraries

1. Static linker fully links library with application object modules.
   - All references resolved by static linker.
   - Loaded by OS loader as part of monolithic executable image.
2. Static linker uses library information, but only links it partially.
   1. Loaded by OS loader as part of `fork()/exec()`.
      1. References resolved in bulk before program execution starts.
      2. References resolved individually at points of first use.
   2. Loaded by dynamic linker, triggered by first use.
      - References resolved individually at points of first use.
3. Static linker doesn't have library information.
   1. System calls allow the process to request the dynamic linker to load, link, and resolve an arbitrary shared library *during application execution*.
      - See `<dlfcn.h>` and `dlopen()`, `dlsym()`, `dlclose()`, `dlerror()` on Linux.

Dynamic Linking Workflow on Linux