CS242: System Software Lab

Logging, Static/Dynamic linking and Makefile

Tut02 : 5th Aug 2024

Dr. A. Sahu

Dept of Comp. Sc. & Engg.

Indian Institute of Technology Guwahati

Outline

- Logging
- Compiling multiple file in C/C++
  - Creating Library, Static/Dynamic
- Makefile:
  - Introduction
  - Example and Example
  - Rules
  - Dependency

Logging

Logging

- A Log file is a file that records events occurs in a operating system or other system software or the messages between the different user on a communication software.
- Logging is a act of keeping log files.

Significance 

System admin need to know what is happening on the system.

Logging

- A general purpose facility called syslog is used.
- Programme send their log entry to syslog by consulting two configuration files **/etc/syslogd.conf** and **/etc/syslog**.
- Syslog contain four basic terms:
  - **Facility:** Describe the type of application. For example: mail, kernel, ftp etc.
  - **Priority or Levels**: Describe the importance of log message. For example: emerg, crit, alert etc.
  - **Selector**: Combination of facility and levels.
  - **Action**: Whenever there is a match of selector, a action is performed eg: message to log file, echo the message to console.

Logging (Contd.)

- The syslog.conf controls where message are logged. A typical syslog.conf look like this:

```
*.err;kern.debug;auth.notice /dev/console
daemon,auth.notice           /var/log/messages
lpr.info                     /var/log/lpr.log
mail.*                       /var/log/mail.log
ftp.*                        /var/log/ftp.log
auth.*                       @prep.ai.mit.edu
auth.*                       root,amrood
netinfo.err                  /var/log/netinfo.log
install.*                    /var/log/install.log
*.emerg                      *
*.alert                      |program_name
mark.*                       /dev/console
```

Group Management

- Three type of account in the linux system:
  - **Root account**:Have complete control of system.
  - **System account**: Have some system specific control. For example: sshd account and mail account.
  - **User account**: Have interactive access to the system and provide very limited access to critical data.
- To manage user and groups, four type of administration files are needed:
  - /etc/passwd - Keep user account and password information.
  - /etc/shadow - Hold encrypted password.
  - /etc/group - contain group information.
  - /etc/gshadow - Hold secure information related to groups.

Fun time

Static Linking
and
Dynamic Linking

Compiling multiple Files

```
//foo.c
int foo3x(int x){
        return 3*x;
 }
```

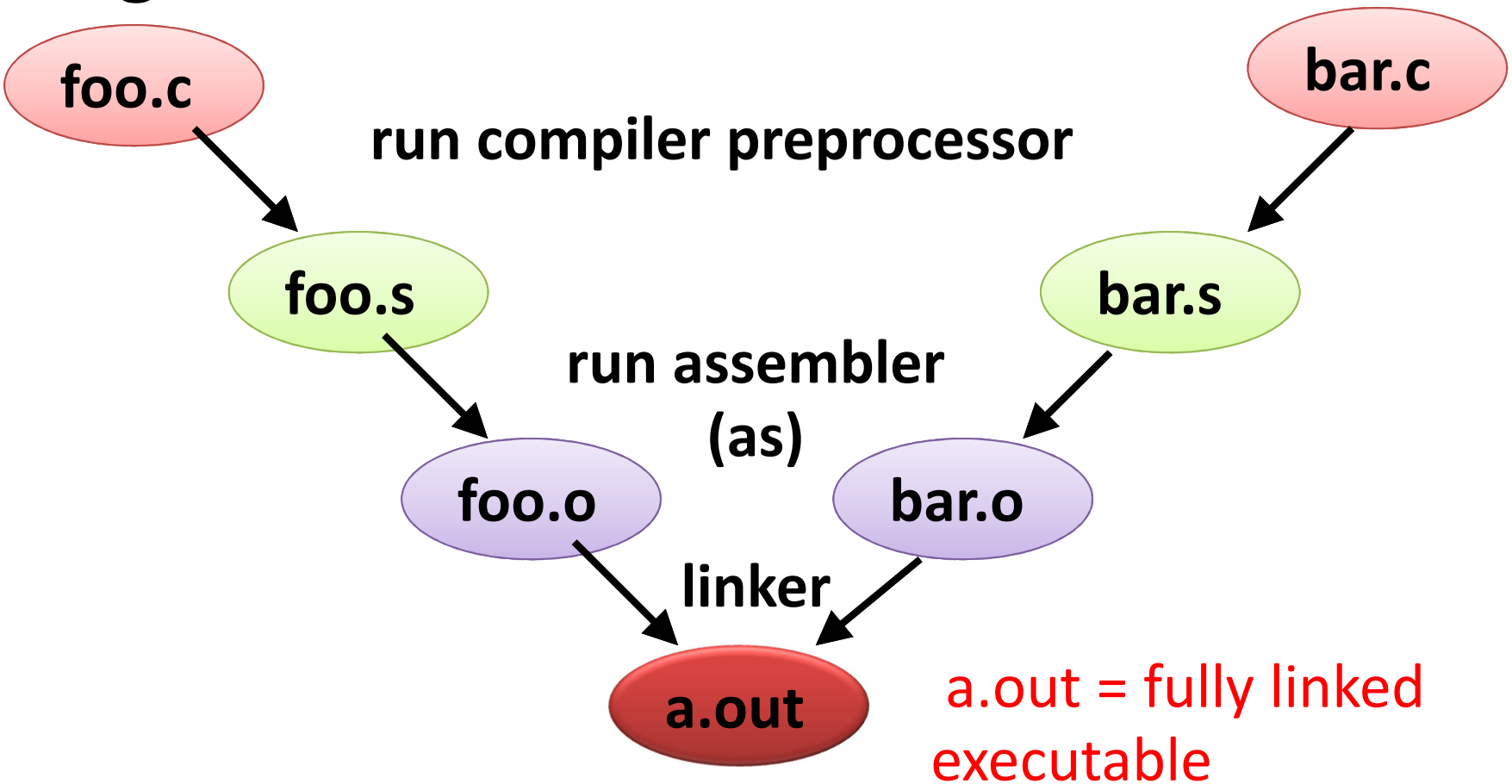
```
//bar.c
int main(){
        int x;
        x=foo3x(10);
        printf("%d",x);
        return  0;
 }
```

- $ gcc –c foo.c
- $ gcc –c bar.c
- $ gcc foo.o bar.o
- $./a.out

Linker and Loader

- Compiler in Action…

$gcc foo.c bar.c –o a.out



What is Linker ?

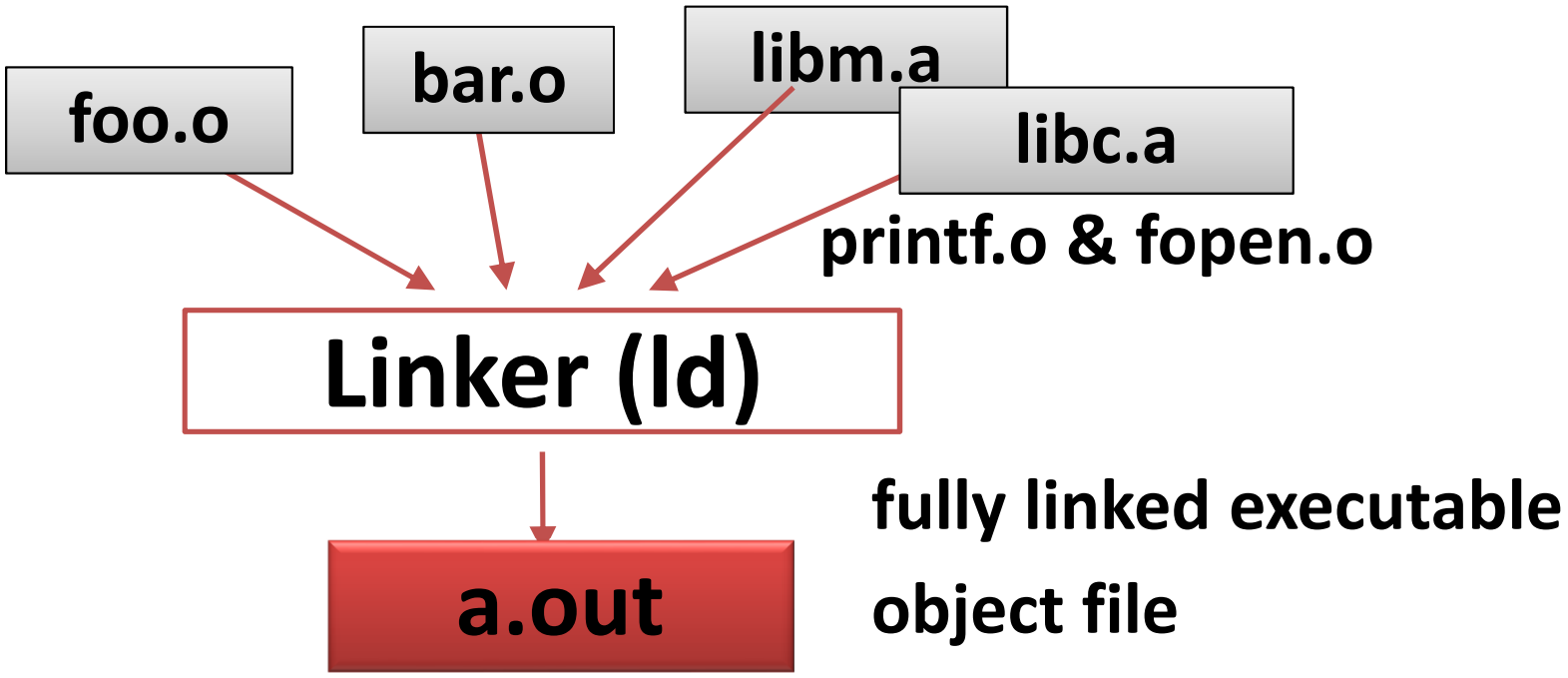
- Combines multiple relocatable object files
- Produces fully linked executable – directly loadable in memory
- How?
  - Symbol resolution – associating one symbol definition with each symbol reference
  - Relocation – relocating different sections of input relocatable files

Object files

- Types –
  - Relocatable : Requires linking to create executable
  - Executable : Loaded directly into memory for execution
  - Shared Objects : Linked dynamically, at run time or load time

Linking with Static Libraries

- Collection of concatenated object files – stored on disk in a particular format – archive
- An input to Linker
  - Referenced object files copied to executable



Creating Static Library

```
//foo.c
int foo3x(int x){
        return 3*x;
 }
```

```
int main(){//bar.c
        int x;
        x=foo3x(10);
        printf("%d",x);
        return  0;
 }
```

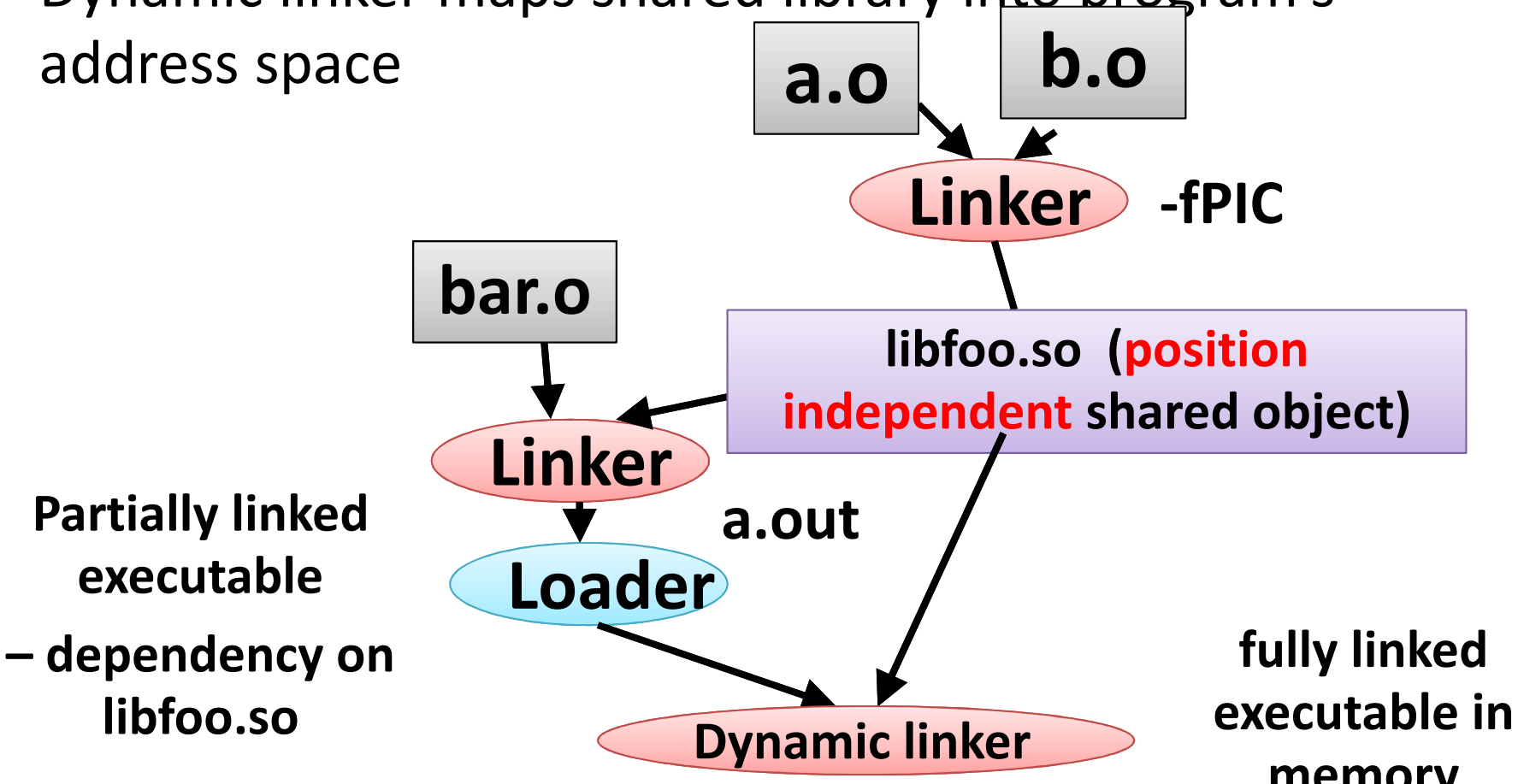
- **$ gcc -c foo.c**
- **$ ar rcs libfoo.a foo.o //it create libfoo.a**
- **$ gcc bar.c -L. -lfoo**
- **$ ./a.out**

Dynamic Linking – Shared Libraries

- Addresses disadvantages of static libraries
  - Ensures one copy of text & data in memory
  - Change in shared library does not require executable to be built again
  - Loaded at run-time by dynamic linker, at arbitrary memory address, linked with programs in memory
  - On loading, dynamic linker relocates text & data of shared object

Shared Libraries ..(Cntd)

- Linker creates **libfoo.so** (PIC) from **a.o** and **b.o**
- **a.out** – partially executable – depend on **libfoo.so**
- Dynamic linker maps shared library into program's address space



Creating Dynamic Library

```
//foo.c
int foo3x(int x){
        return 3*x;
 }
```

```
int main(){//bar.c
        int x;
        x=foo3x(10);
        printf("%d",x);
        return  0;
 }
```

- $gcc -c -fPIC foo.c
- $gcc -shared -Wl,-soname,libfoo.so.1 -o libfoo.so.1 foo.o
- $ gcc bar.c -L. –lfoo
- $ export LD_LIBRARY_PATH=.
- $ ./a.out

Makefile

Introduction

- "make" utility in Unix
  - Is one of the original tools designed by S. I. Fieldman of AT&T Bell labs 1977.
- What is "make"?
  - The tool is designed to allow programmers to
  - efficiently compile large complex programs with many components easily.
- You can place the commands to compile a program in a Unix script
  - but this will cause ALL modules to be compiled every time.
- The "make" utility allows us to only compile those
  - that have changed and the modules that depend upon them.

Motivation

- Small programs ⟶ single file

- “Not so small” programs :
  - Many lines of code
  - Multiple components
  - More than one programmer

Motivation – continued

- Problems:
  - Long files are harder to manage
    (for both programmers and machines)
  - Every change requires long compilation
  - Many programmers can not modify the same file simultaneously
  - Division to components is desired

Motivation – continued

- Solution : divide project to multiple files
- Targets:
  - Good division to components
  - Minimum compilation when something is changed
  - Easy maintenance of project structure, dependencies and creation

Project maintenance

- Done in Unix by the Makefile mechanism
- A makefile is a file (script) containing :
  - Project structure (files, dependencies)
  - Instructions for files creation
- The make command reads a makefile, understands the project structure and makes up the executable
- Makefile mechanism is
  - not limited to C or Fortran programs

How Does it Work?

- When you type the command “make” the OS looks for a file called either “makefile” or “Makefile”.
- This file contains a series of directives that tell the “make” utility
  - How to compile your program and in what order.
- Each file will be associated with a list of other files by which it is dependent.
  - This is called a dependency line.
- If any of the associated files have been recently modified,
  - The make utility will execute a directive command just below the dependency line.

How Does it Work?

- The “make” utility is recursive.
  - For instance, if a very low level utility is the only thing changed,
  - it could cause all of the modules within a program to be re-compiled.
- After the utility finishes the file,
  - it goes through and checks all of the dependencies again to make sure all are up to date.

Thanks