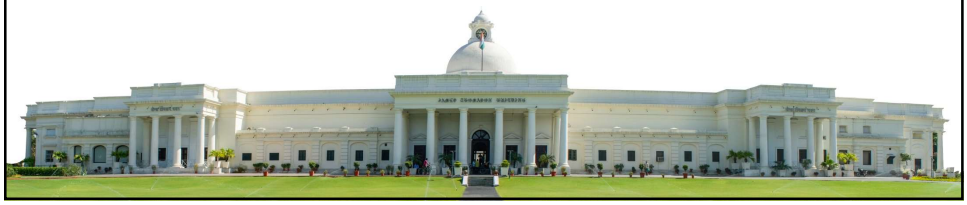
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE

System Software
CSN-252
Linkers

R. E. Bryant Chap. On Linkers



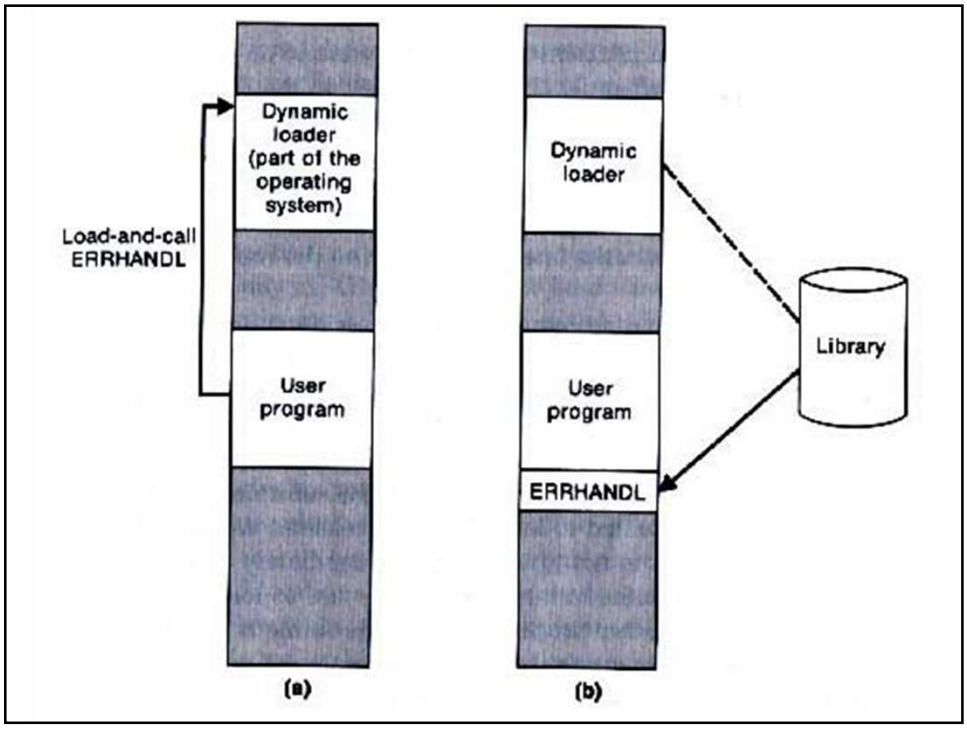
Dynamic linking, dynamic loading, or load on call

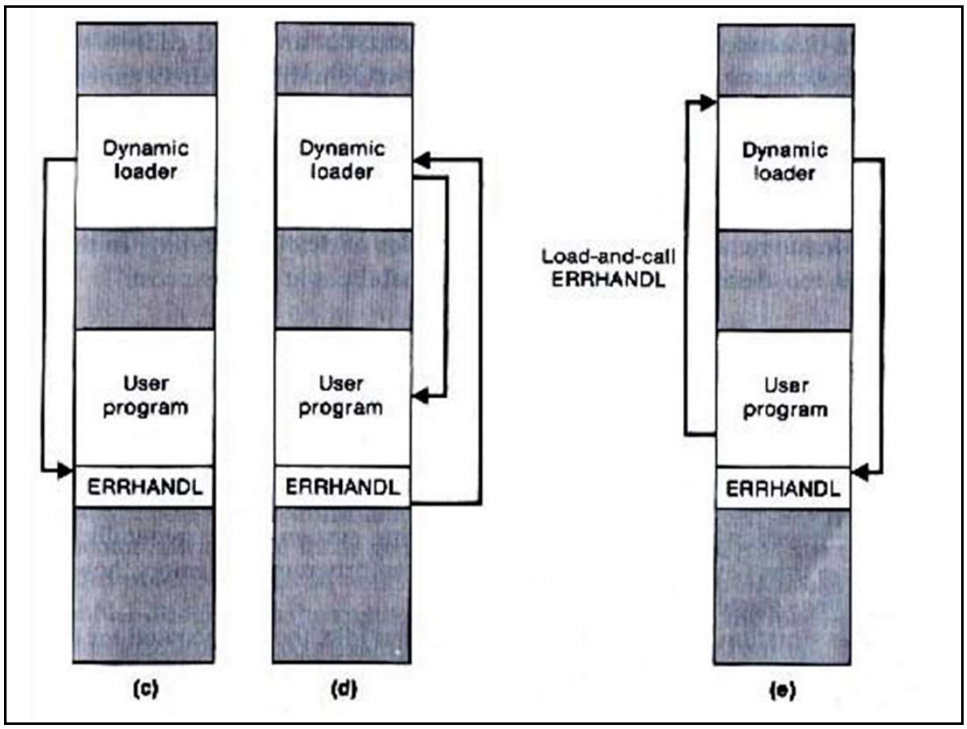
- Postpones the linking function until execution time

  => a subroutine is loaded and linked to the rest of the program when it is first called.

- Allows several executing programs to share one copy of a subroutine or library

- **Scenario:** a program that allows its user to interactively call any of the subroutines of a large mathematical and statistical library
  - all of the library subroutines could potentially be needed, but only a few will actually be used in any one execution.
  - Dynamic linking can avoid the necessity of loading the entire library for each execution except those necessary subroutines.


Dynamic loader (part of the operating system)
Load-and-call ERRHANDL
User program
(a)
Dynamic loader
Library
User program
ERRHANDL
(b)


Dynamic loader
User program
ERRHANDL
(c)
Dynamic loader
User program
ERRHANDL
(d)
Load-and-call ERRHANDL
Dynamic loader
User program
ERRHANDL
(e)

Example C Program (Tut 1)

```
#include <stdio.h>
int x=7;
int y=5;
int p2();
int main() {
p2();
printf("%d %d\n", x, y);
}
```

first.c

```
double x;
int p2() {
x = 2.5;
}
```

second.c

Example C Program

```
manoj@manoj-VirtualBox:~/CSN-252/linker$ ls
a.out  first.c  second.c
manoj@manoj-VirtualBox:~/CSN-252/linker$ gcc first.c second.c
/usr/bin/ld: warning: alignment 4 of symbol `x' in /tmp/ccSlYpTb.o is smaller t
han 8 in /tmp/ccuh7Ibe.o
manoj@manoj-VirtualBox:~/CSN-252/linker$ ./a.out
0 1074003968
manoj@manoj-VirtualBox:~/CSN-252/linker$ vi first.c
manoj@manoj-VirtualBox:~/CSN-252/linker$ gcc first.c second.c
/usr/bin/ld: warning: alignment 4 of symbol `x' in /tmp/ccpHApsX.o is smaller t
han 8 in /tmp/ccPITFsW.o
manoj@manoj-VirtualBox:~/CSN-252/linker$ ./a.out
0 40040000
manoj@manoj-VirtualBox:~/CSN-252/linker$
```

IIT ROORKEE

Three Kinds of Object Files

Relocatable object file (`.o` file)

Contains code and data in a form that can be combined with other relocatable object files to form executable object file.

Executable object file (`a.out` file)

Contains code and data in a form that can be copied directly into memory and then executed.

Shared object file (`.so` file)

Special type of relocatable object file that can be loaded into memory and linked dynamically, at either load time or run-time.

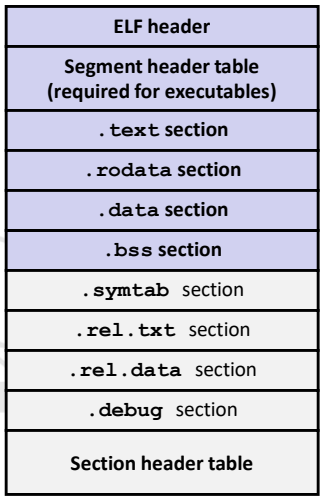
Called *Dynamic Link Libraries* (DLLs) by Windows

Generic name: ELF binaries

IIT ROORKEE

ELF Object File Format

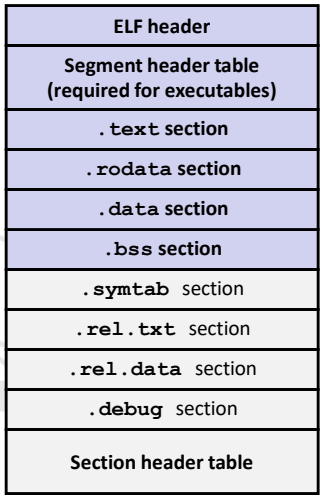
- Elf header
  - Word size, byte ordering, file type (.o, exec, .so), machine type, etc.
  - First four bytes (7F 45 4c 46)
- Segment header table
  - Page size, virtual addresses memory segments (sections), segment sizes.
- `.text` section
  - Code
- `.rodata` section
  - Read only data: jump tables, ...
- `.data` section
  - Initialized global variables



IIT ROORKEE

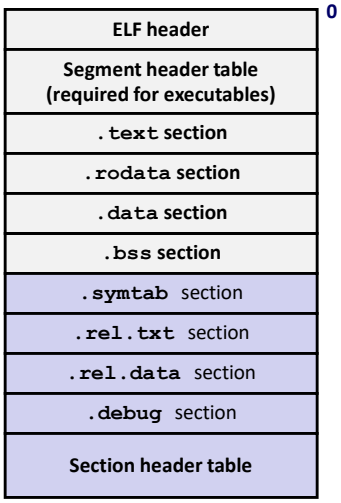
ELF Object File Format

- `.bss` section
  - Uninitialized global variables
  - "Block Started by Symbol" / "Better Save Space"
  - Has section header but occupies no space
- `.symtab` section
  - Symbol table
  - Procedure and static variable names
  - Section names and locations
  - Every relocatable file has this



ELF Object File Format (cont.)

- `.rel.text` section
  - Relocation info for **`.text`** section
  - Addresses of instructions that will need to be modified in the executable
  - Instructions for modifying
- `.rel.data` section
  - Relocation info for **`.data`** section
  - Addresses of pointer data that will need to be modified in the merged executable
- `.debug` section
  - Info for symbolic debugging (**`gcc -g`**)
- Section header table
  - Offsets and sizes of each section



Relocatable Object Module Format (OMF)

- Used primarily for software intended to run on x86 microprocessors
- Originally developed during 1981
- Important record types
  - THEADR - Header
  - EXTDEF - Defines external references
  - PUBDEF - Identifies external symbols in this module
  - SEGDEF - Identifies segments (name, length, alignment etc.)
  - GRPDEF - Identifies groups of segments
  - FIXUPP - relocation and linking information
  - LEDATA - Contains text of a code or data section
  - MODEND - Indicates end of module

Example C Program (Tut 1)

```
#include <stdio.h>
int x=7;
int y=5;
int p2();
int main() {
p2();
printf("%d %d\n", x, y);
}
```

first.c

```
double x;
int p2() {
x = 2.5;
}
```

second.c

>objdump –D a.out

Why Linkers?

- Reason 1: Modularity
  - Program can be written as a collection of smaller source files, rather than one monolithic mass.
  - Can build libraries of common functions
    - e.g., Math library, standard C library

Why Linkers?

- Reason 2: Efficiency
  - Time: Separate compilation
    - Change one source file, compile, and then relink
    - No need to recompile other source files.
  - Space: Libraries
    - Common functions can be aggregated into a single file...
    - Yet executable files and running memory images contain only code for the functions they actually use.