# Loaders and Linkers

Chapter 3

System Software

An introduction to systems programming

Leland L. Beck

# Introduction

- To execute an object program, we needs
  - » **Relocation**, which modifies the object program so that it can be loaded at an address different from the location originally specified
  - » **Linking**, which combines two or more separate object programs and supplies the information needed to allow references between them (Section 2.2.2)
  - » **Loading and Allocation**, which allocates memory location and brings the object program into memory for execution (Section 2.3.5)

# Overview of Chapter 3

- Type of loaders
  - assemble-and-go loader
  - absolute loader (bootstrap loader)
  - relocating loader (relative loader)
  - direct linking loader
- Design options
  - linkage editors
  - dynamic linking
  - bootstrap loaders

# Assemble-and-go Loader

- Characteristic
  - the object code is stored in memory after assembly
  - single JUMP instruction
- Advantage
  - simple, developing environment
- Disadvantage
  - whenever the assembly program is to be executed, it has to be assembled again
  - programs have to be coded in the same language

# Design of an Absolute Loader

- Absolute Loader
  - Advantage
    - Simple and efficient
  - Disadvantage
    - the need for programmer to specify the actual address
    - difficult to use subroutine libraries
- Program Logic
  - Next slice

# Fig. 3.2 Algorithm for an absolute loader

```
Begin
read Header record
verify program name and length
read first Text record
while record type is not ‘E’ do
    begin
    {if object code is in character form, convert into internal representation}
    move object code to specified location in memory
    read next object program record
    end
jump to address specified in End record
end
```

# Object Code Representation

- Figure 3.1 (a)
  - » each byte of assembled code is given using its hexadecimal representation in character form
  - » easy to read by human beings
- In general
  - » each byte of object code is stored as a single byte
  - » most machine store object programs in a binary form
  - » we must be sure that our file and device conventions do not cause some of the program bytes to be interpreted as control characters

# A Simple Bootstrap Loader

- Bootstrap Loader
  - When a computer is first tuned on or restarted, a special type of absolute loader, called *bootstrap loader* is executed
  - This bootstrap loads the first program to be run by the computer -- usually an operating system
- Example (SIC bootstrap loader)
  - The bootstrap itself begins at address 0
  - It loads the OS starting address 0x80
  - No header record or control information, the object code is consecutive bytes of memory

# Fig. 3.3 SIC Bootstrap Loader Logic

```
Begin
X=0x80 (the address of the next memory location to be loaded
Loop
    A←GETC (and convert it from the ASCII character code to the value
    of the hexadecimal digit)
    save the value in the high-order 4 bits of S
    A←GETC
    combine the value to form one byte A← (A+S)
    store the value (in A) to the address in register X
    X←X+1
End
```

0~9 : 48
A~F : 65

```
GETC   A←read one character
       if A=0x04 then jump to 0x80
       if A<48 then GETC
       A ← A-48 (0x30)
       if A<10 then return
       A ← A-7 (48+7=55)
       return
```

# Relocating Loaders

- Motivation
  - efficient sharing of the machine with larger memory and when several independent programs are to be run together
  - support the use of subroutine libraries efficiently
- Two methods for specifying relocation
  - modification record (Fig. 3.4, 3.5)
  - relocation bit (Fig. 3.6, 3.7)
    - each instruction is associated with one relocation bit
    - these relocation bits in a Text record is gathered into bit masks

# Modification Record

- For complex machines
- Also called RLD specification
  - » Relocation and Linkage Directory

Modification record
    col 1: M
    col 2-7: relocation address
    col 8-9: length (halfbyte)
    col 10: flag (+/-)
    col 11-17: segment name

# Relocation Bit

- For simple machines
- Relocation bit
  - » 0: no modification is necessary
  - » 1: modification is needed

Text record
  col 1: T
  col 2-7: starting address
  col 8-9: length (byte)
  col 10-12: relocation bits
  col 13-72: object code

- Twelve-bit mask is used in each Text record
  - » since each text record contains less than 12 words
  - » unused words are set to 0
  - » any value that is to be modified during relocation must coincide with one of these 3-byte segments
    - – e.g. line 210

# Program Linking

- Goal
  - Resolve the problems with EXTREF and EXTDEF from different control sections
- Linking
  - 1. User, 2. Assembler, 3. Linking loader
- Example
  - Program in Fig. 3.8 and object code in Fig. 3.9
  - Use modification records for both relocation and linking
    - address constant
    - external reference

# Program Linking Example

| | | Program A | Program B | Program C |
|---|---|---|---|---|
| Label | Expression | LISTA, ENDA | LISTB, ENDB | LISTC, ENDC |
| REF1 | LISTA | local, R, PC | external | external |
| REF2 | LISTB+4 | external | local, R, PC | external |
| REF3 | ENDA-LISTA | local, A | external | external |
| REF4 | ENDA-LISTA+LISTC | local, A | external | local, R |
| REF5 | ENDC-LISTC-10 | external | external | local, A |
| REF6 | ENDC-LISTC+LISTA-1 | local, R | external | local, A |
| REF7 | ENDA-LISTA-(ENDB-LISTB) | local, A | local, A | external |
| REF8 | LISTB-LISTA | local, R | local, R | external |

# Program Linking Example

- Fig. 3.10
- Load address for control sections

| | | |
|---|---|---|
| PROGA | 004000 | 63 |
| PROGB | 004063 | 7F |
| PROGC | 0040E2 | 51 |

- Load address for symbols
  - LISTA: PROGA+0040=4040
  - LISTB: PROGB+0060=40C3
  - LISTC: PROGC+0030=4112
- REF4 in PROGA
  - ENDA-LISTA+LISTC=14+4112=4126
  - T0000540F000014FFFFF600003F000014FFFFC0
  - M00005406+LISTC

# Program Logic and Data Structure

- Two Passes Logic
  - Pass 1: assign addresses to all external symbols
  - Pass 2: perform the actual loading, relocation, and linking
- ESTAB (external symbol table)

| Control section | Symbol | Address | Length |
|---|---|---|---|
| Progam A | | 4000 | 63 |
| | LISTA | 4040 | |
| | ENDA | 4054 | |
| Program B | | 4063 | 7F |
| | LISTB | 40C3 | |
| | ENDB | 40D3 | |
| Program C | | 40E2 | 51 |
| | LISTC | 4112 | |
| | ENDC | 4124 | |

# Pass 1 Program Logic

- Pass 1:
  - » assign addresses to all external symbols
- Variables
  - » PROGADDR (program load address) from OS
  - » CSADDR (control section address)
  - » CSLTH (control section length)
  - » ESTAB
- Fig. 3.11(a)
  - » Process Define Record

# Pass 2 Program Logic

- Pass 2:
  - perform the actual loading, relocation, and linking
- Modification record
  - lookup the symbol in ESTAB
- End record for a main program
  - transfer address
- Fig. 3.11(b)
  - Process Text record and Modification record

# Improve Efficiency

- Use local searching instead of multiple searches of ESTAB for the same symbol
  - » assign a reference number to each external symbol
  - » the reference number is used in Modification records
- Implementation
  - » 01: control section name
  - » other: external reference symbols
- Example
  - » Fig. 3.12

# Figure 3.12

| Ref No. | Symbol | Address |
|---|---|---|
| 1 | PROGA | 4000 |
| 2 | LISTB | 40C3 |
| 3 | ENDB | 40D3 |
| 4 | LISTC | 4112 |
| 5 | ENDC | 4124 |

PROGA

| Ref No. | Symbol | Address |
|---|---|---|
| 1 | PROGB | 4063 |
| 2 | LISTA | 4040 |
| 3 | ENDA | 4054 |
| 4 | LISTC | 4112 |
| 5 | ENDC | 4124 |

PROGB

| Ref No. | Symbol | Address |
|---|---|---|
| 1 | PROGC | 4063 |
| 2 | LISTA | 4040 |
| 3 | ENDA | 4054 |
| 4 | LISTB | 40C3 |
| 5 | ENDB | 40D3 |

PROGC

# Machine-Independent Loader Features

- Automatic Library Search
  - » Many linking loaders can automatically incorporate routines from a subprogram library into the program being loaded
    - – A standard library
    - – Other libraries may be specified by control statements or by parameters to the loader
  - » Also called *automatic library call* in some systems

# Automatic Library Search

- Implementation
  - Linking loaders that support automatic library search must keep track of external symbols that are referred to , but not defined, in the primary input to the loader
  - At the end of Pass 1, the symbols in ESTAB that remain undefined represented *unresolved* external references
  - Then, the loader searches the library or libraries specified for routines that contain the definitions of these symbols
  - Note that the subroutines fetched from a library in this way may themselves contain external references.
    - It is therefore necessary to repeat the library search process until all reference are resolved.

# Automatic Library Search

- Implementation
  - The process allows the programmer to override the standard subroutines in the library by supplying his or her own routines
- The libraries to be searched by the loader ordinarily contain assembled or compiled versions of the subroutines (i.e., object programs)
  - For efficient searching
    - Directory
  - Some operating systems can keep the directory for commonly used libraries permanently in memory
- The same technique applies equally well to the resolution of external references to data items

# Loader Options

- Many loaders allow the user to specify options that modify the standard processing
- Many loaders have a special command language
  - A separate input file to loader
  - Embedded in the primary input stream
  - In source program

# Loader Options

- Examples of command language

1. *INCLUDE program-name(library-name)*

   Direct the loader to read the designated object program from a library and treat it as if it were part of the primary loader input

2. *DELETE csdect-name*

   Instruct the loader to delete the named control section(s) from the set of programs being loaded

3. *CHANGE name1, name2*

   Cause the external symbol *name1* to be changed to *name2* wherever it appears in the object programs

```
INCLUDE READ(UTLIB)
INCLUDE WRITE(UTLIB)
DELETE RDREC, WRREC
CHANGE RDREC, READ
CHANGE WRREC, WRITE
```

# Loader Options

- Examples of command language

4. *LIBRARY MYLIB*

   Automatic inclusion of library routines to satisfy external references

   Searched before the standard libraries

5. *NOCALL STDDEV, PLOT, CORREL*

   To instruct the loader that these external references are to remain unsolved

6. *Others*

   Output from the load, e.g., the map which includes control section names and adddresses

   The ability to specify the location at which execution is to begin

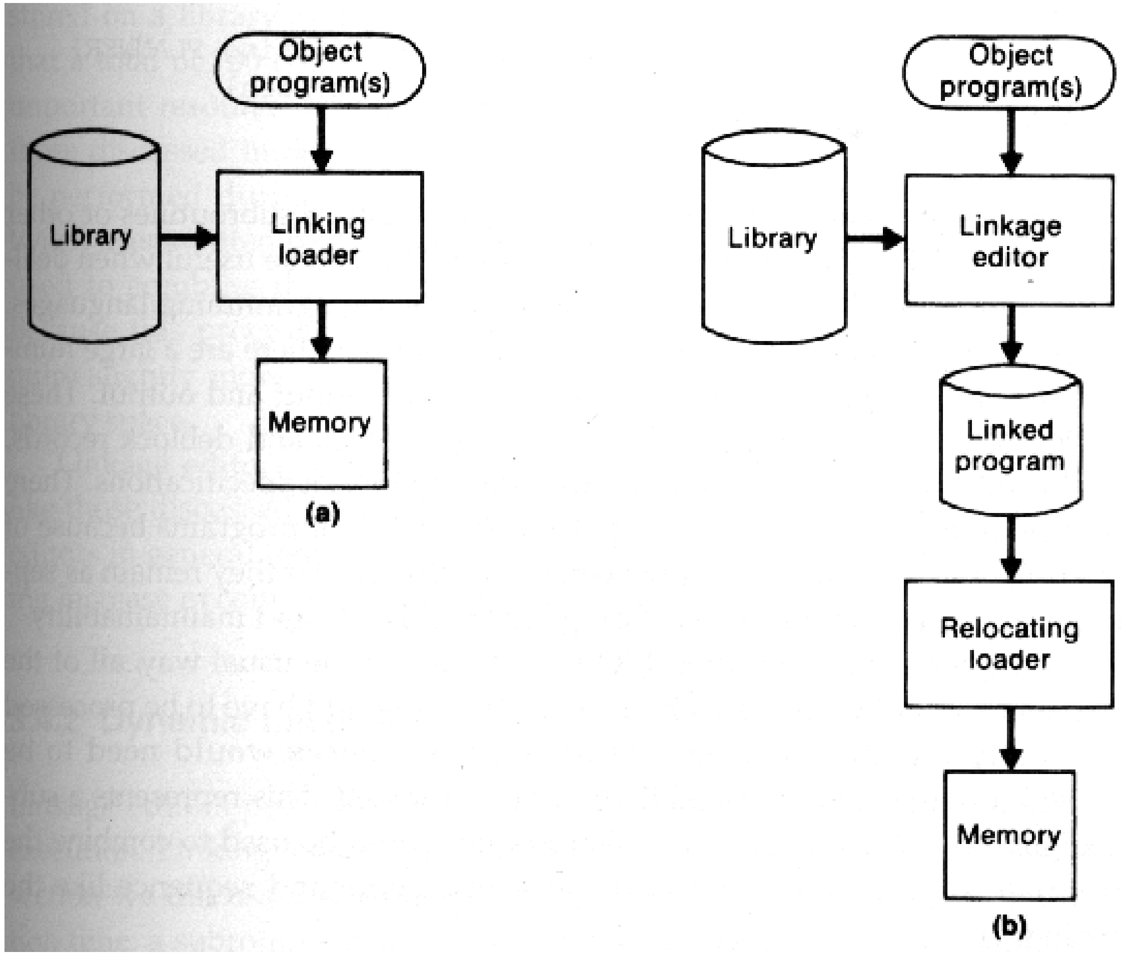
   Control whether or not the loader should attempt to execute the program if errors are detected during the load

# Loader Design Options

- Linkage Editor
  - » Perform linking prior to load time
- Dynamic linking
  - » Linking function is performed at execution time
- Bootstrap loader
  - » Be used to run stand-alone programs independent of the operating system or the system loader

# Linkage Editors

- The essential difference between a linkage editor and a linking loader



# Linkage Editors

- A linking loaders performs
  - » All linking and relocation operations
  - » Automatic library search
  - » Loads the linked program directly into memory for execution
- A linkage editor
  - » Produces a linked version of program (often called a load module or an executable image), which is written to a file or library for later execution
  - » A simple relocating loader can be used to load the linked version of program into memory
    - – The loading can be accomplished in one pass with no external symbol table required

# Linkage Editors

- A linkage editor
  - Resolution of external references and library searching are only performed once
  - In the linked version of programs
    - All external references are resolved, and relocation is indicated by some mechanism such as modification records or a bit mask
  - External references is often retained in the linked program
    - To allow subsequent relinking of the program to replace control sections, modify external references, etc.

# Linkage Editors

- Linkage editors can perform many useful functions besides simply preparing an object program for execution
  1. The linkage editor can be used to replace the subroutines in the linked version

```
INCLUDE     PLANNER(PROGLIB)
DELETE      PROJECT
INCLUDE     PROJECT(NEWLIB)
REPLACE     PLANNER(PROGLIB)
```

# Linkage Editors

2. Linkage editors can also be used to build packages of subroutines or other control sections that are generally used together

   It could be used to combine the appropriate subroutines into a package with a command sequence

```
INCLUDE        READR(FTNLIB)
INCLUDE        WRITER(FTNLIB)
INCLUDE        BLOCK(FTNLIB)
INCLUDE        DEBLOCK(FTNLIB)
INCLUDE        ENCODE(FTNLIB)
INCLUDE        DECODE(FTNLIB)
.
.
.
SAVE           FTNIO(SUBLIB)
```

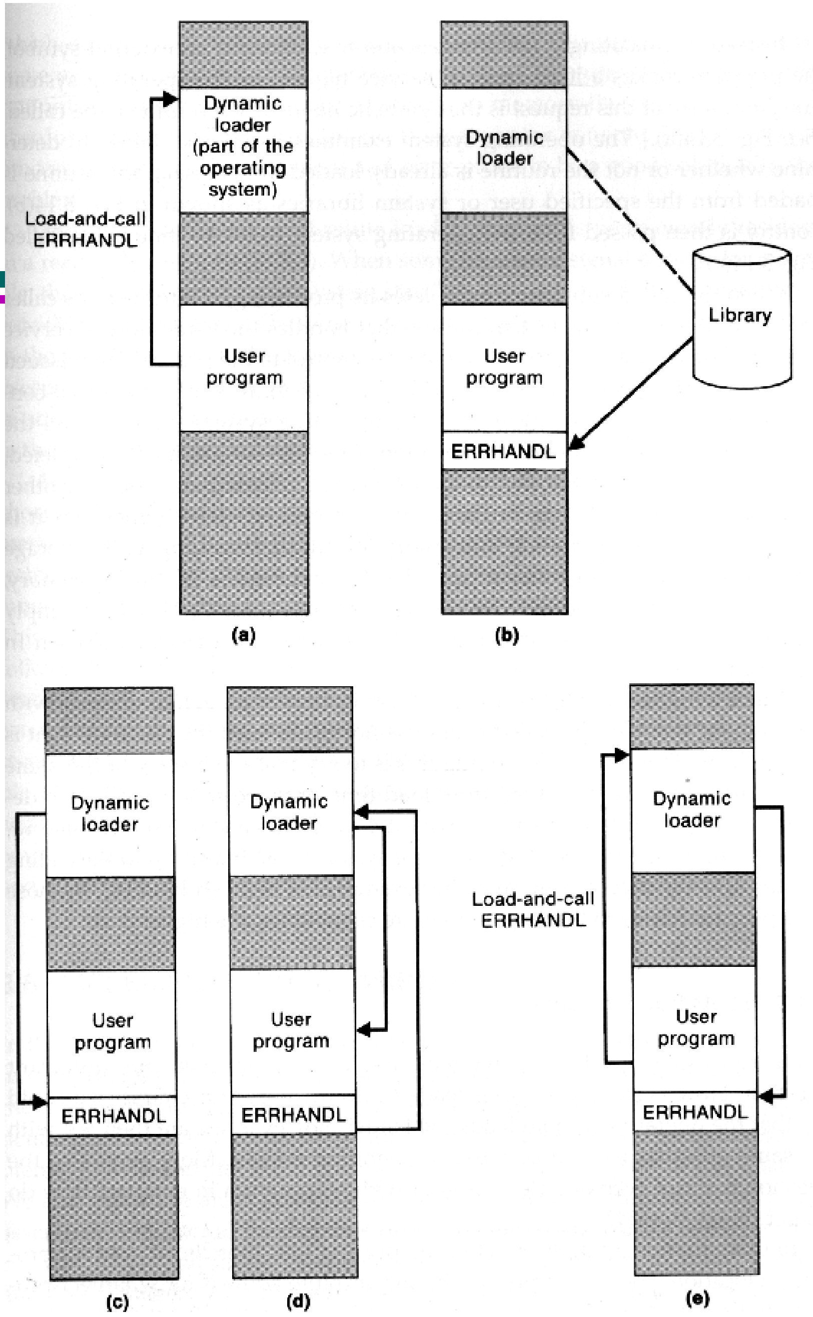
# Linkage Editors

3. Linkage editors often allow the user to specify that external references are not to be resolved by automatic library search

   Only the external references between user-written routines would be resolved

# Dynamic Linking

- Postpone the linking function until execution time
  - » A subroutine is loaded and linked to the rest of the program when it is first called
    - – Dynamic linking, dynamic loading, or load on call
- Allow several executing programs to share one copy of a subroutine or library
- In object-oriented system, it allows the implementation of the object and its methods to be determined at the time the program is run
- Dynamic linking provides the ability to load the routines only when they are needed



- Dynamically loaded must be called via an operating system service request
- Load-and-call service
  a) OS examines its inernal tables to determine whether or not the routine is already loaded
  b) Routine is loaded from library
  c) Control is passed from OS to the called subroutine
  d) Subroutine is finished
  e) Calling to a subroutine which is already in memory
- *Binding* of the name to an actuall address is delayed from load time until execution  time

# Bootstrap Loaders

- Given an idle computer with no program in memory, how do we get things started?
  - » With the machine empty and idle, there is no need for program relocation
    - – Some early computers required the operator to enter into memory the object code for an absolute loader, using switches on the computer console
    - – One some computer, an absolute loader program is permanently resident in a ROM
    - – A built-in hardware function that reads a fixed-length record form some device into memory at a fixed location