Loaders and Linkers

Dr. Arka Prokash Mazumdar

Introduction

- To execute an object program -
  - **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
  - **Loading and Allocation**, which allocates memory location and brings the object program into memory for execution

Introduction

- Type of loaders
  - assemble-and-go loader
  - absolute loader (bootstrap loader)
  - relocating loader (relative loader)
  - direct linking loader

Assemble-and-go Loader

- Characteristic
  - 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

Absolute Loader

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

Algorithm: 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

- 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

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
A← (A+S) (combine the value to form one byte)
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

Object File Format (SIC)

Header record:

Col. 1	H
Col. 2-7	Program name
Col. 8-13	**Starting address of object program** (hexadecimal)
Col. 14-19	**Length** of object **program** in bytes (hexadecimal)

Text record:

Col. 1	T
Col. 2-7	**Starting address** for object code in this record (hexadecimal)
Col. 8-9	**Length** of **object** code in this record in bytes (hexadecimal)
Col. 10 – 69	**Object code**, represented in hexadecimal (2 columns per byte of object code)

End record:

Col. 1	E
Col. 2-7	**Address of first executable instruction** in object program (hexadecimal)

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**
  - **Relocation bit**
    - 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 (SIC?)**
  - 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

Example

```
T    000000    1E   FFC  140033 481039 000036 280030
                        300015 481061 3C0003 00002A
                        0C0039 00002D
```

- **FFC=111111111100**

Program Linking

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

Control Section 1

```
5     0000  COPY      START   0 COPY FILE FROM INPUT TO
            OUTPUT
6                     EXTDEF  BUFFER, BUFEND, LENGTH
7                     EXTREF  RDREC, WRREC
10    0000  FIRST     STL     RETADR
15    0003  CLOOP     +JSUB   RDREC
20    0007            LDA     LENGTH
25    000A            COMP    #0
30    000D            JEQ     ENDFIL
35    0010            +JSUB   WRREC
40    0014  J         CLOOP
45    0017  ENDFIL    LDA     =C'EOF'
50    001A            STA     BUFFER
55    001D            LDA     =C'EOF'
60    0020            STA     LENGTH
65    0023            +JSUB   WRREC
70    0027            J       @RETADR
95    002A            RETARD  RESW    1
100   002D            LENGTH  RESW    1
103                   LTORG
      0030  *         =C'EOF'
•     0033  BUFFER    RESB    4096
•     1033  BUFEND    EQU     *
105   1000  MAXLEN    EQU     BUFEND-BUFFER
```

Control Section 2

```
109  0000        RDREC   CSECT
122              EXTREF  BUFFER, LENGTH, BUFEND
125  0000        CLEAR   X
130  0002        CLEAR   A
132  0004        CLEAR   S
133  0006        LDT     MAXLEN
135  0009 RLOOP  TD      INPUT
140  000C        JEQ     RLOOP
145  000F        RD      INPUT
150  0012        COMPR   A,S
155  0014        JEQ     EXIT
160  0017        +STCH   BUFFER,X
165  001B        TIXR    T
170  001D        JLT     RLOOP
175  0020 EXIT   +STX    LENGTH
180  0024        RSUB
185  0027 INPUT  BYTE    X'F1'
186  0028 MAXLEN WORD    BUFEND-BUFFER
```

Control Section 3

```
193   0000  WRREC    CSECT
195
                     EXTREF   LENGTH,BUFFER
212   0000           CLEAR    X
215   0002           +LDT     LENGTH
220   0006  WLOOP     TD       =X'05'
225   0009           JEQ      WLOOP
230   000C           +LDCH    BUFFER,X
235   0010           WD       =X'05'
240   0013           TXR      T
245   0015           JLT      WLOOP
•     0018           RSUB
255                  END      FIRST
      001B *         =X'05'
```

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

- 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
  - T 000054 0F 000014 FFFFF6 00003F 000014 FFFFC0
  - M 000054 **06** + 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 1:
  - 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

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	40E2
2	LISTA	4040
3	ENDA	4054
4	LISTB	40C3
5	ENDB	40D3

PROGC