System Software 2

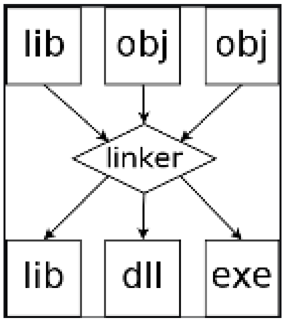
INTRODUCTION TO LINKER LECTURE03 -2023

Dr. Azam E. Al-Rawachy
COLLEGE OF COMPUTER SCIENCES AND MATHEMATICS

1 Introduction

Linker is a computer program that links and merges various object files together in order to make an ***executable file***. All these files might have been compiled by separate assemblers. The major task of a linker is to search and locate referenced module/routines in a program and to determine the memory location where these codes will be loaded, making the program instruction to have absolute references.

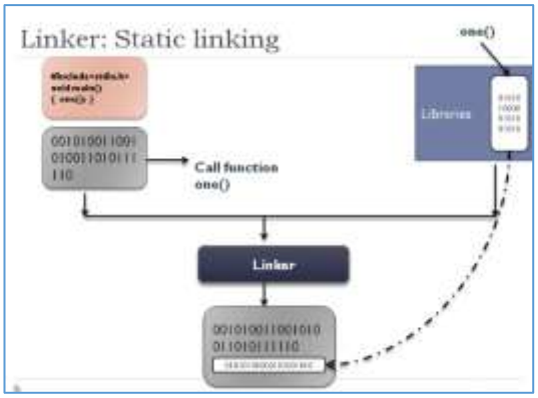
Furthermore, it combines the object codes with libraries. For example, in a C program, if there is sqrt () function to calculate the square root of a number, the linker links the program with the math library. Finally, the CPU can read and understand the generated executable file. Therefore, the CPU can execute that file to perform the task defined in the program. The above process can be summarized as program life cycle (write -> compile -> link -> load -> execute).



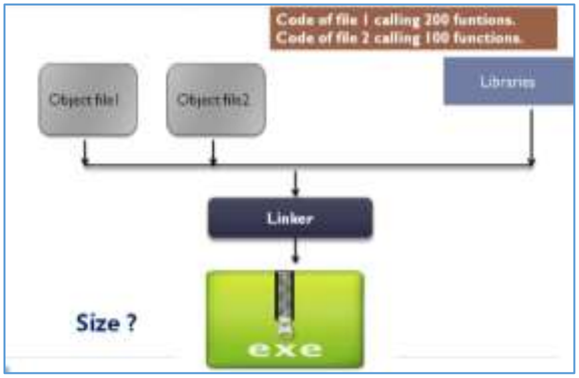
1.1 Linking is of two types:

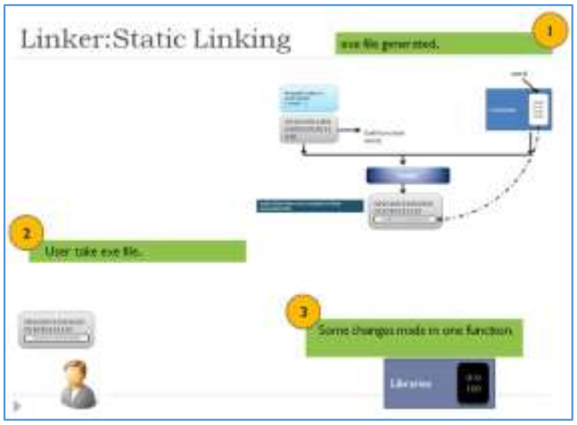
1. Static Linking

Static linking is the process of copying all library modules used in the program into the final executable image. This is performed by the linker (link editors) and it is done as the last step of the compilation process. The linker combines library routines with the program code in order to resolve ***external references***, and to generate an executable image suitable for loading into memory. When the program is loaded, the operating system places into memory a single file that contains the executable code and data.



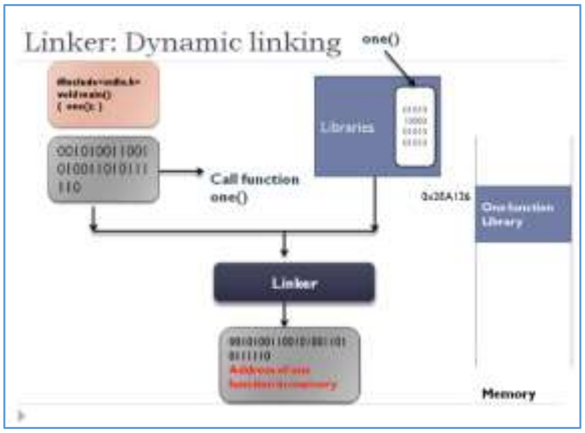
Statically linked files are significantly larger in size because external programs are built into the executable files. In static linking if any of the external programs has changed then they have to be recompiled and re-linked again, the changes won't reflect in existing executable file. Statically, linked files are significantly larger in size because external programs are built into the executable files and any changes won't reflect in existing executable file.


Code of file 1 calling 200 funtions.
Code of file 2 calling 100 functions.
Object file1
Object file2
Libraries
Linker
Size ?
exe



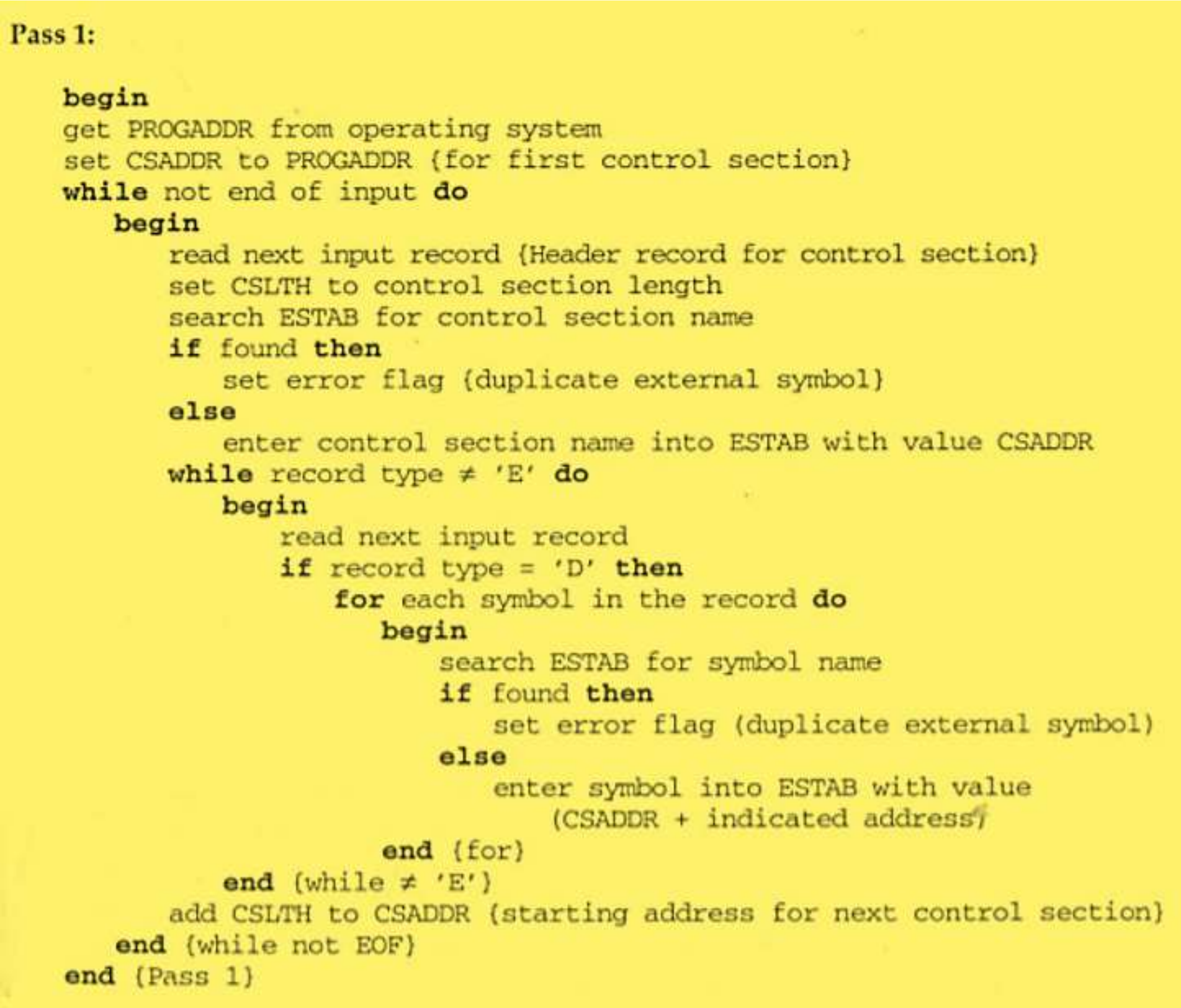
2. **Dynamic linking**

In dynamic linking the names of the external libraries (shared libraries) are placed in the final executable file while the actual linking takes place at **run time** when both executable file and libraries are placed in the memory. Dynamic linking lets several programs use a single copy of an executable module. It is performed at run time by the operating system.



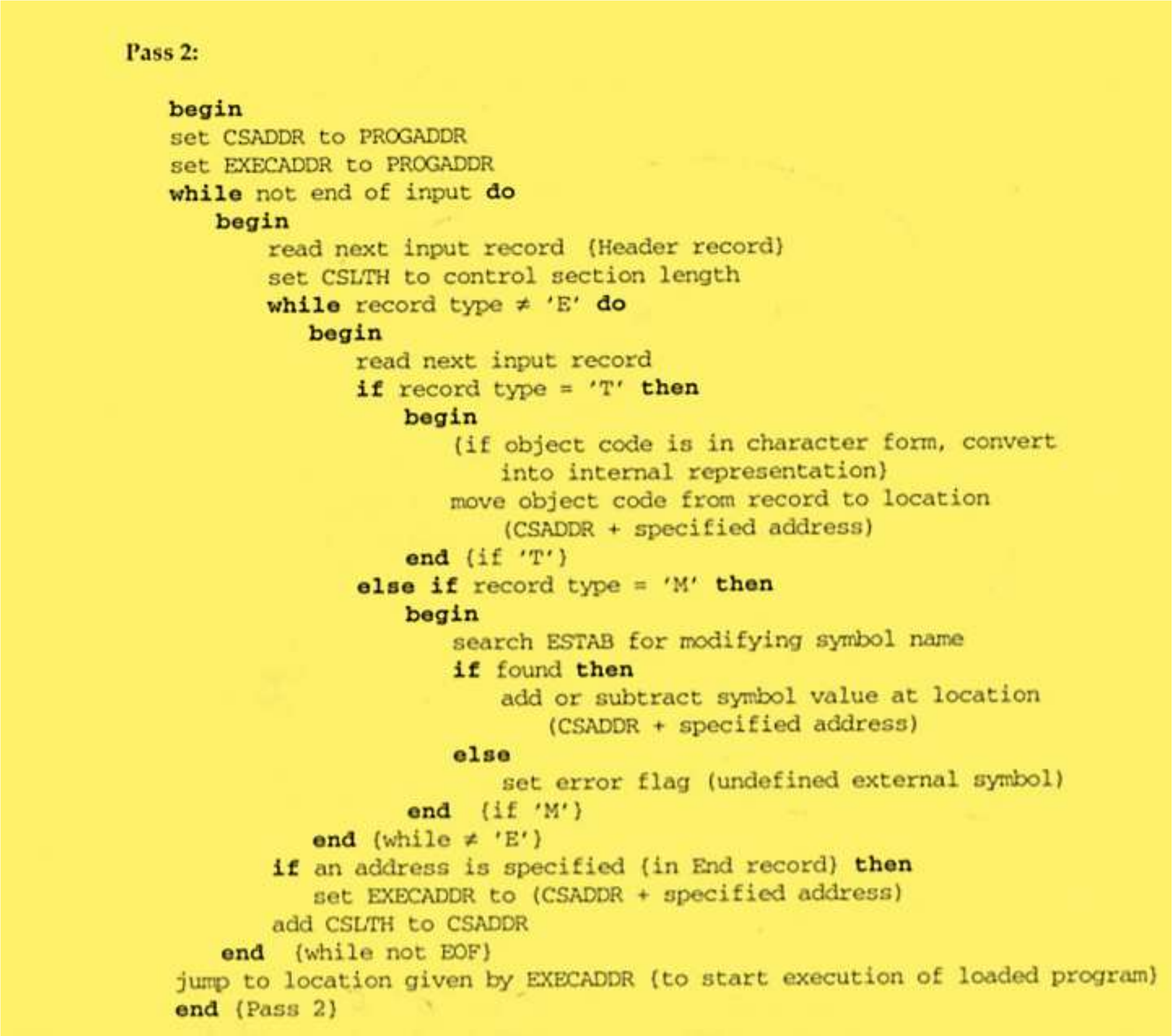
In dynamic linking, only one copy of shared library is kept in memory. This significantly reduces the size of executable programs, thereby saving memory and disk space. Individual shared modules can be updated and recompiled. This is one of the greatest advantages dynamic linking offers. Dynamically linked programs are dependent on having a compatible library. If a library is changed (for example, a new compiler release may change a library), applications might have to be reworked to be made compatible with the new version of the library. If a library is removed from the system, programs using that library will no longer work.

1.2 Linking Loader Pass 1 & 2 Algorithms



```
Pass 1:

    begin
    get PROGADDR from operating system
    set CSADDR to PROGADDR {for first control section}
    while not end of input do
        begin
            read next input record {Header record for control section}
            set CSLTH to control section length
            search ESTAB for control section name
            if found then
                set error flag {duplicate external symbol}
            else
                enter control section name into ESTAB with value CSADDR
            while record type ≠ 'E' do
                begin
                    read next input record
                    if record type = 'D' then
                        for each symbol in the record do
                            begin
                                search ESTAB for symbol name
                                if found then
                                    set error flag (duplicate external symbol)
                                else
                                    enter symbol into ESTAB with value
                                        (CSADDR + indicated address)
                            end {for}
                end {while ≠ 'E'}
            add CSLTH to CSADDR {starting address for next control section}
        end {while not EOF}
    end {Pass 1}
```



Pass 2:

```
begin
set CSADDR to PROGADDR
set EXECADDR to PROGADDR
while not end of input do
    begin
        read next input record  {Header record}
        set CSLTH to control section length
        while record type ≠ 'E' do
            begin
                read next input record
                if record type = 'T' then
                    begin
                        {if object code is in character form, convert
                            into internal representation}
                        move object code from record to location
                            (CSADDR + specified address)
                    end {if 'T'}
                else if record type = 'M' then
                    begin
                        search ESTAB for modifying symbol name
                        if found then
                            add or subtract symbol value at location
                                (CSADDR + specified address)
                        else
                            set error flag (undefined external symbol)
                    end  {if 'M'}
            end {while ≠ 'E'}
        if an address is specified {in End record} then
            set EXECADDR to (CSADDR + specified address)
        add CSLTH to CSADDR
    end  {while not EOF}
jump to location given by EXECADDR {to start execution of loaded program}
end {Pass 2}
```

Introduction to program linking

A program made up of more than one subroutine (functions, modules). After assembling has been completed, these subroutines appear as separate sections of object code and then link together.

It may be happening that instructions in one subroutine might need to refer to instructions or data located in another. Such references between subroutine are called external reference. The linker performs required linking process between these different subroutines. In any subroutines, we may have the following two types of symbols:

- Defined "External" symbols, sometimes called "public" or "entry" symbols, which allow it to be called by other modules.
- Undefined "External" symbols, which reference other modules where these symbols are defined.

The goal of program linking is to resolve the problems with external references from different subroutines. The assembler has no idea where the different subroutines will be loaded. So, it performs the following:

1. Inserts an address of 0 for any external reference and passes information to linking loader.

2. Includes information in the object program that will cause the linking loader to insert the proper values where they are required. To perform this, the assembler needs to **add two new record types** in the object program: ***Define record*** and ***Refer record***. The other information needed for program linking is added to the Modification record.

Example of Linking Loader:

Generate the object program for the following two code sections using Two-Pass Assembler and Linking Loader.

```
PRG:

 BEGIN:    JMP      MAIN                     EB --
 N1        DW       -3
 N2        DW        3
 MAIN:     MOV       AX, N1                  A1 -- --
           CALL     MLL                      E8 -- --
           CMP      WORD PTR RES+2, 0        83 3E -- -- -- --
           JNE      CONT                     75 --
           MOV      WORD PTR RES+2, -1       C7 06 -- -- -- --
CONT:      RET                              C3
           END
```

```
SUB:

MLL:       MUL      WORD PTR N2      F7 26 -- --
           MOV      RES, AX          A3 -- --
           MOV      RES+2, DX        89 16 -- --
           RET                       C3
           END
RES        DW       0, 0
           END
```

Inserting ENTRY and EXTURN to the PRG and SUB

PRG:

```
             ENTRY       N2
             EXTRN       MLL, RES
BEGIN:       JMP         MAIN
N1           DW          -3
N2           DW           3
MAIN         MOV         AX, N1
             CALL        MLL
             CMP         WORD PTR RES+2, 0
             JNE         CONT
             MOV         WORD PTR RES+2, -1
CONT:        RET
```

SUB:

```
             ENTRY       MLL, RES
             EXTRN       N2
MLL:         MUL         WORD PTR N2
             MOV         RES, AX
             MOV         RES+2, DX
             RET
             END
RES          DW          0, 0
             END
```

PRG:

Pass-1

Intermediate file:

Address	Mnemonic field			Error Flag
0000	BEGIN:	JMP	MAIN	0
0002	N1	DW	-3	0
0004	N2	DW	3	0
0006	MAIN:	MOV	AX, N1	0
0009		CALL	MLL	0
000C		CMP	WORD PTR RES+2, 0	0
0012		JNE	CONT	0
0014		MOV	WORD PTR RES+2, -1	0
001A	CONT:	RET		0
001B		END		0

SYMTAB:

Symbol	Value	Error Flag
BEGIN:	0000	0
N1	0002	0
N2	0004	0
MAIN	0006	0
CONT	001A	0

Pass-2 ➔ Listing file:

Line No.	Address	Mnemonic field	Machin code	Error flag
1	0000	BEGIN: JMP MAIN	EB 04	0
2	0002	N1 DW -3	FD FF	0
3	0004	N2 DW 3	03 00	0
4	0006	MAIN: MOV AX, N1	A1 02 00	0
5	0009	CALL MLL	E8 **00 00**	0
6	000C	CMP WORD PTR RES+2, 0	83 3E **00 00** 00 00	0
7	0012	JNE CONT	75 06	0
8	0014	MOV WORD PTR RES+2, -1	C7 06 **00 00** FF FF	0
9	001A	CONT: RET	C3	0
10	001B	END		

BEGIN:	0000	EB		
	0001	04		
N1	0002	FD		
	0003	FF		
N2	0004	03		
	0005	00		
MAIN:	0006	A1		
	0007	02	N1	N1
	0008	00		
	0009	E8		
	000A	00	MLL	MLL
	000B	00		
	000C	83		
	000D	3E		
	000E	00	RES	RES
	000F	00		
	0010	00		
	0011	00		
	0012	75		
	0013	06		
	0014	C7		
	0015	06		
	0016	00	RES+2	RES+2
	0017	00		
	0018	FF		
	0019	FF		
CONT:	001A	C3		

Object Program:

H PRG - - - 0000 001B
T 0000 08 EB 04 FD FF 03 00 A1 02
T 0008 08 00 E8 00 00 83 3E 00 00
T 0010 08 00 00 75 06 C7 06 00 00
T 0018 03 FF FF C3
M 0007 + PRG
M 000A + SUB
M 000E + SUB
M 0016 + SUB
D N2- - - - 0004 ➔ For ENTRY
R MLL - - - + SUB ➔ For EXTURN
R RES - - - + SUB ➔ For EXTURN
E 0000

SUB:

Pass-1

Intermediate file:

Address	Mnemonic field			Error Flag
0000	MLL:	MUL	WORD PTR N2	0
0004		MOV	RES, AX	0
0007		MOV	RES+2, DX	0
000B		RET		0
000C		END		0
000C	RES	DW	0, 0	0
0010		END		0

SYMTAB:

Symbol	Value	Error Flag
MLL:	0000	0
RES	000C	0

Pass-2

Listing file:

Line No.	Address	Mnemonic field			Machin code	Error flag
1	0000	MLL:	MUL	WORD PTR N2	F7 26 **00 00**	0
2	0004		MOV	RES, AX	A3 0C 00	0
3	0007		MOV	RES+2, DX	89 16 0E 00	0
4	000B		RET		C3	0
5	000C		END			0
6	000C	RES	DW	0, 0	00 00 00 00	0
7	0010		END			0

MLL:	0000	F7		
	0001	26		
	0002	00	N2	N2
	0003	00		
	0004	A3		
	0005	0C	RES	RES
	0006	00		
	0007	89		
	0008	16		
	0009	0E	RES+2	RES+2
	000A	00		
	000B	C3		
RES	000C	00		
	000D	00		
RES+2	000E	00		
	000F	00		

Object Program:
H SUB - - - 0000 0010
T 0000 08 F7 26 00 00 A3 OC 00 89
T 0008 08 16 0E 00 C3 00 00 00 00
M 0002+ PRG ➔ Modification
M 0005 + SUB ➔ Modification
M 0009 + SUB ➔ Modification
D MLL - - - 0000 ➔ For ENTRY
D RES - - - 000C ➔ For ENTRY
R N2 - - - - ➔ For EXTURN
E 0000

Loading with Linking (Linking Loader)

1. Get PROGADDR from O.S.
2. SET CSADDR TO PROGADDR for the first control section

Assume **PROGADDR = C200H**,

ESTAB (Extended SYMTAB)

Symbol	**Value**	**Error Flag**	
PRG	C200	0	0000H+CSADDR=0000H+C200H=C200H
BEGIN:	C200	0	0000H+CSADDR=0000H+C200H=C200H
N1	C202	0	0002H+CSADDR=0002H+C200H=C202H
N2	C204	0	0004H+CSADDR=0004H+C200H=C204H
MAIN:	C206	0	0006H+CSADDR=0006H+C200H=C206H
CONT	C21A	0	001AH+CSADDR=001AH+C200H=C21AH
SUB	C21B	0	0000H+CSADDR+LENGTH(PRG:1B) =C21BH
MLL:	C21B	0	0000H+CSADDR+LENGTH(PRG:1B) =C21BH
RES	C227	0	**000C**H+CSADDR+LENGTH(PRG:1B)=C227H

BEGIN:	C200	EB	
	C201	04	
N1	C202	FD	
	C203	FF	
N2	C204	03	
	C205	00	
MAIN:	C206	A1	
	C207	02	N1
	C208	C2	
	C209	E8	
	C20A	0F	MLL
	C20B	00	
	C20C	83	
	C20D	3E	
	C20E	27	RES
	C20F	C2	
	C210	00	
	C211	00	
	C212	75	
	C213	06	
	C214	C7	
	C215	06	
	C216	29	RES+2
	C217	C2	
	C218	FF	
	C219	FF	
CONT:	C21A	C3	
MLL:	C21B	F7	
	C21C	26	
	C21D	04	N2
	C21E	C2	
	C21F	A3	
	C220	27	RES
	C221	C2	
	C222	89	
	C223	16	
	C224	29	RES+2
	C225	C2	
	C226	C3	
RES	C227	00	
	C228	00	
RES+2	C229	00	
	C22A	00	

Object Program:

```
H Linked C200 002B
T C200 08 EB 04 FD FF 03 00 A1 02
T C208 08 00 EB 00 00 83 3E 00 00
T C210 08 00 00 75 06 C7 06 00 00
T C218 08 FF FF C3 F7 26 00 00 A3
T C220 08 0C 00 89 A6 0E 00 C3 00
T C228 03 00 00 00
E C200
```

Disp = Destination address- Source address -3

3: Length of (CALL) instruction

Disp = C21BH-C209H-3

Disp = 000FH