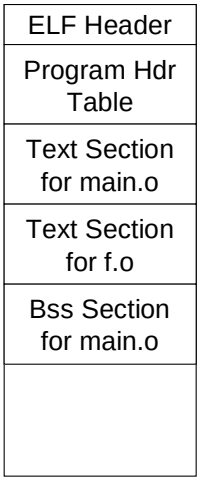
Linker

CS 217

Linker

- Combines multiple object files into a single executable file (**`a.out`**)
  - Also called a “link editor” or “linker/loader”
  - Object files (*.o) are ELF files
  - Executable files (a.out) are also ELF files



Unix `ld` Command

- Implicit (called by compiler)

```
gcc main.c foo.c
```

- Explicit (supports separate compilation)

```
gcc –c main.c
gcc –c foo.c
gcc main.o foo.o
```

alternatively

```
gcc –c main.c
gcc –c foo.c
ld /usr/lib/crt0.o main.o foo.o –lc –lm
```

(do "**`gcc –v main.o foo.o`**" to see full ugliness)

Some Details ...

- Linkers can build object files incrementally

```
% ld –o ab.o a.o b.o
% ld –o a.out ab.o c.o
```

- Linkers combine all modules from .o files, even if they are not referenced

```
gcc –o example main.o foo.o other.o
```

- Linkers combine modules from .a files, only if they are referenced

```
ar libfoo.a foo.o other.o
gcc –o example main.o libfoo.a
```

- Linkers do not type check across modules

C Language Example

main.c

```
static int a = 3;
extern int b;
static int c;

static int f1(int x) {
   return x + b;
}

int f2(int x) {
   return x + c;
}

int main() {
  f1(a);
  f2(a);
  f3(a);
  f4(a);
}
```

foo.c

```
int b = 1;

int f3(int x)
{
   int d = 0;
   return x + d;
}

int f4(int x)
{
   return x - b;
}
```

Assembly Language

main.s

```
        .section ".data"
a:      .word 3

        .section ".bss"
c:      .skip 4

        .section ".text"
f1:     save    %sp,-96,%sp
        mov     %i0,%o1
        sethi   %hi(b),%o0
        ld      [%o0+%lo(b)],%o0
        add     %o1,%o0,%i0
        ret; restore

        .global f2
f2:     save    %sp,-96,%sp
        mov     %i0,%o1
        sethi   %hi(c),%o0
        ld      [%o0+%lo(c)],%o0
        add     %o1,%o0,%i0
        ret; restore

        .global main
main:   save    %sp,-96,%sp
        sethi   %hi(a),%l0
        or      %l0,%lo(a),%l0
        ld      [%l0+0],%l1
        mov %l1, %o0; call f1; nop
        mov %l1, %o0; call f2; nop
        mov %l1, %o0; call f3; nop
        mov %l1, %o0; call f4; nop
        ret; restore
```

foo.s

```
            .section ".data"
        .global b
b:      .word 1

        .section ".text"
        .global f3
f3:     save    %sp,-104,%sp
        st      %g0,[%fp-8]
        ld      [%fp-8], %l0
        add     %i0,%l0,%i0
        ret; restore

        .global f4
f4:     save    %sp,-104,%sp
        sethi   %hi(b),%l1
        ld      [%l1+%lo(b)],%l0
        sub     %i0,%l0,%i0
        ret; restore
```

Machine Language

main.o

ELF Header
Section Hdr Table
Data Section for main.o
Bss Section for main.o
Text Section for main.o
etc

foo.o

ELF Header
Section Hdr Table
Data Section for foo.o
Text Section for foo.o
etc

Linking

- Concatenate .o files to create executable
  - Resolve undefined symbols
  - Fixup references to relocated data

ELF Header
Program Hdr Table
Data Section for main.o
Bss Section for main.o
Text Section for main.o
Data Section for foo.o
Text Section for foo.o

Linker's Two Main Functions

- Symbol Resolution:
  - Use symbol table to resolve all external references; this also involves searching libraries to find any undefined symbols
- Symbol Relocation:
  - Fixup references to addresses in other segments. Assembler assumes each .o file begins at address 0, but when multiple object files are linked together some addresses must be relocated

Symbol Resolution

main.c

```
static int a = 3;
extern int b;
static int c;

static int f1(int x) {
   return x + b;
}

int f2(int x) {
   return x + c;
}

int main() {
  f1(a);
  f2(a);
  f3(a);
  f4(a);
}
```

foo.c

```
int b = 1;

int f3(int x)
{
   int d = 0;
   return x + d;
}

int f4(int x)
{
   return x - b;
}
```

Symbol Resolution

main.c

```
static int a = 3;
extern int b;
static int c;

static int f1(int x) {
   return x + b;
}

int f2(int x) {
   return x + c;
}

int main() {
  f1(a);
  f2(a);
  f3(a);
  f4(a);
}
```

foo.c

```
int b = 1;

int f3(int x)
{
   int d = 0;
   return x + d;
}

int f4(int x)
{
   return x - b;
}
```

Symbol Resolution

main.c

```
static int a = 3;
extern int b;
static int c;

static int f1(int x) {
   return x + b;
}

int f2(int x) {
   return x + c;
}

int main() {
  f1(a);
  f2(a);
  f3(a);
  f4(a);
}
```

foo.c

```
int b = 1;

int f3(int x)
{
   int d = 0;
   return x + d;
}

int f4(int x)
{
   return x - b;
}
```

Symbol Table Before Linking

```
% gcc –c main.c
% nm main.o

Index Value  Size Type  Bind        Shndx   Name
[4]   |    0|    4|OBJT |LOCL |0    |4      |a
[11]  |    0|    0|NOTY |GLOB |0    |UNDEF  |b
[6]   |    0|    4|OBJT |LOCL |0    |3      |c
[5]   |   16|   40|FUNC |LOCL |0    |2      |f1
[10]  |   72|   40|FUNC |GLOB |0    |2      |f2
[8]   |    0|    0|NOTY |GLOB |0    |UNDEF  |f3
[7]   |    0|    0|NOTY |GLOB |0    |UNDEF  |f4
[9]   |  128|   60|FUNC |GLOB |0    |2      |main
```

```
2 = text section
3 = bss section
4 = data section
UNDEF = undefined
```

Symbol Table Before Linking

```
% gcc –c foo.c
% nm foo.o

Index Value  Size Type  Bind        Shndx   Name
[4]   |    0|    4|OBJT |GLOB |0    |3      |b
[3]   |   16|   36|FUNC |GLOB |0    |2      |f3
[2]   |   72|   40|FUNC |GLOB |0    |2      |f4
```

```
2 = text section
3 = bss section
4 = data section
UNDEF = undefined
```

Symbol Table After Linking

```
% gcc -o example main.o foo.o
% nm example

Index       Value     Size Type Bind       Shndx  Name
[36]    |    133472|       4|OBJT |LOCL |0    |17     |a
[60]    |    133476|       4|OBJT |GLOB |0    |17     |b
[38]    |    133480|       4|OBJT |LOCL |0    |18     |c
[37]    |     67120|      40|FUNC |LOCL |0    |8      |f1
[44]    |     67176|      40|FUNC |GLOB |0    |8      |f2
[46]    |     67312|      36|FUNC |GLOB |0    |8      |f3
[48]    |     67368|      40|FUNC |GLOB |0    |8      |f4
[56]    |     67232|      60|FUNC |GLOB |0    |8      |main
...
```

Symbol Table After Linking

```
% gcc -o example main.o foo.o
% nm example

Index       Value     Size Type Bind       Shndx  Name
[36]    |    133472|       4|OBJT |LOCL |0    |17     |a
[60]    |    133476|       4|OBJT |GLOB |0    |17     |b
[38]    |    133480|       4|OBJT |LOCL |0    |18     |c
[37]    |     67120|      40|FUNC |LOCL |0    |8      |f1
[44]    |     67176|      40|FUNC |GLOB |0    |8      |f2
[46]    |     67312|      36|FUNC |GLOB |0    |8      |f3
[48]    |     67368|      40|FUNC |GLOB |0    |8      |f4
[56]    |     67232|      60|FUNC |GLOB |0    |8      |main
...
```

Symbol Relocation

main.c

```
static int a = 3;
extern int b;
static int c;

static int f1(int x) {
   return x + b;
}

int f2(int x) {
   return x + c;
}

int main() {
  f1(a);
  f2(a);
  f3(a);
  f4(a);
}
```

foo.c

```
int b = 1;

int f3(int x)
{
   int d = 0;
   return x + d;
}

int f4(int x)
{
   return x - b;
}
```

Symbol Relocation

main.c

```
static int a = 3;
extern int b;
static int c;

static int f1(int x) {
   return x + b;
}

int f2(int x) {
   return x + c;
}

int main() {
  f1(a);
  f2(a);
  f3(a);
  f4(a);
}
```

foo.c

```
int b = 1;

int f3(int x)
{
   int d = 0;
   return x + d;
}

int f4(int x)
{
   return x - b;
}
```

Symbol Relocation

main.c

```
static int a = 3;
extern int b;
static int c;

static int f1(int x) {
   return x + b;
}

int f2(int x) {
   return x + c;
}

int main() {
  f1(a);
  f2(a);
  f3(a);
  f4(a);
}
```

foo.c

```
int b = 1;

int f3(int x)
{
   int d = 0;
   return x + d;
}

int f4(int x)
{
   return x - b;
}
```

Symbol Relocation

main.c

```
static int a = 3;
extern int b;
static int c;

static int f1(int x) {
   return x + b;
}

int f2(int x) {
   return x + c;
}

int main() {
  f1(a);
  f2(a);
  f3(a);
  f4(a);
}
```

foo.c

```
int b = 1;

int f3(int x)
{
   int d = 0;
   return x + d;
}

int f4(int x)
{
   return x - b;
}
```

Symbol Relocation

main.c

```
static int a = 3;
extern int b;
static int c;

static int f1(int x) {
   return x + b;
}

int f2(int x) {
   return x + c;
}

int main() {
  f1(a);
  f2(a);
  f3(a);
  f4(a);
}
```

foo.c

```
int b = 1;

int f3(int x)
{
   int d = 0;
   return x + d;
}

int f4(int x)
{
   return x - b;
}
```

Symbol Relocation

main.c

```
static int a = 3;
extern int b;
static int c;

static int f1(int x) {
   return x + b;
}

int f2(int x) {
   return x + c;
}

int main() {
  f1(a);
  f2(a);
  f3(a);
  f4(a);
}
```

foo.c

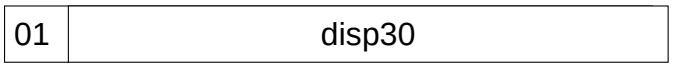
```
int b = 1;

int f3(int x)
{
   int d = 0;
   return x + d;
}

int f4(int x)
{
   return x - b;
}
```

Relocation

- Assembler adds relocation entries to object file
  - Stored in a relocation table
- Each relocation entry:
  - Identifies address of instruction that references relocatable address, and
  - Tells how to fixup the reference

Relocation Entries

- Example 1: external procedure call



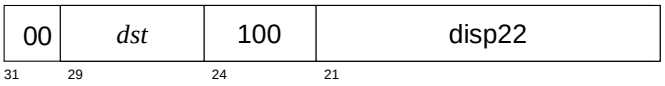

```
int main() {
  f1(a);
  f2(a);
  f3(a);
  f4(a);
}
```

```
...
main: save %sp, -96, %sp
      set  a, %l0
      ld [%l0] %o0
      call f1
      nop
      call f2
      nop
      call f3
      nop
      call f4
      nop
      ...
```

Need to fixup addresses of **`f3`** and **`f4`** when link

Relocation Entries

- Example 2: external global variable



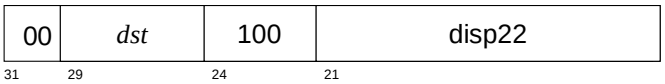

```
static int f1(int x)
{
  return x + b;
}
```

```
...
f1: save %sp, -96, %sp

    sethi %hi(b),%l0
    or dst,%lo(b),%l0
    ld [%l0], %l1

    add %i0, %l1, %i0

    ret
    restore
```

Need to fixup address of **b** in **sethi** and **or** instructions when link

Relocation Entries

- Example 3: internal global variable




```
static int f2(int x)
{
  return x + c;
}
```

```
...
f2: save %sp, -96, %sp

    sethi %hi(c),%l0
    or dst,%lo(c),%l0
    ld [%l0], %l1

    add %i0, %l1, %i0

    ret
    restore
```

Need to fixup address of **c** in **sethi** and **or** instructions when link

Relocation Entries

```
elfdump -r main.o
type                 offset   addend  section        name
R_SPARC_HI22         0x1c          0  .rela.text     b
R_SPARC_LO10         0x20          0  .rela.text     b
R_SPARC_HI22         0x54          0  .rela.text     .bss
R_SPARC_LO10         0x58          0  .rela.text     .bss
R_SPARC_HI22         0x84          0  .rela.text     .data
R_SPARC_LO10         0x88          0  .rela.text     .data
R_SPARC_WDISP30      0x94          0  .rela.text     f2
R_SPARC_WDISP30      0x9c          0  .rela.text     f3
R_SPARC_WDISP30      0xa4          0  .rela.text     f4

elfdump -r foo.o
type                 offset   addend  section        name
R_SPARC_HI22         0x54          0  .rela.text     b
R_SPARC_LO10         0x58          0  .rela.text     b
```

Symbol Table Before Linking

```
% gcc –c main.c
% nm main.o

Index Value  Size Type  Bind        Shndx   Name
[4]   |    0|    4|OBJT |LOCL |0    |4      |a
[11]  |    0|    0|NOTY |GLOB |0    |UNDEF  |b
[6]   |    0|    4|OBJT |LOCL |0    |3      |c
[5]   |   16|   40|FUNC |LOCL |0    |2      |f1
[10]  |   72|   40|FUNC |GLOB |0    |2      |f2
[8]   |    0|    0|NOTY |GLOB |0    |UNDEF  |f3
[7]   |    0|    0|NOTY |GLOB |0    |UNDEF  |f4
[9]   |  128|   60|FUNC |GLOB |0    |2      |main
```

```
2 = text section
3 = bss section
4 = data section
UNDEF = undefined
```

Symbol Table Before Linking

```
% gcc –c foo.c
% nm foo.o

Index Value  Size Type  Bind        Shndx   Name
[4]  |    0|    4|OBJT |GLOB |0    |3      |b
[3]  |   16|   36|FUNC |GLOB |0    |2      |f3
[2]  |   72|   40|FUNC |GLOB |0    |2      |f4
```

```
2 = text section
3 = bss section
4 = data section
UNDEF = undefined
```

Symbol Table After Linking

```
% gcc -o example main.o foo.o
% nm example

Index       Value     Size Type Bind        Shndx  Name
[36]    |  133472|       4|OBJT |LOCL |0    |17     |a
[60]    |  133476|       4|OBJT |GLOB |0    |17     |b
[38]    |  133480|       4|OBJT |LOCL |0    |18     |c
[37]    |   67120|      40|FUNC |LOCL |0    |8      |f1
[44]    |   67176|      40|FUNC |GLOB |0    |8      |f2
[46]    |   67312|      36|FUNC |GLOB |0    |8      |f3
[48]    |   67368|      40|FUNC |GLOB |0    |8      |f4
[56]    |   67232|      60|FUNC |GLOB |0    |8      |main
...
```

Elfdump

```
% gcc –c main.c
% elfdump -s main.o

Symbol Table:  .symtab
index    value       size     type bind oth ver shndx       name
  [0]  0x00000000 0x00000000  NOTY LOCL  D    0 UNDEF
  [1]  0x00000000 0x00000000  FILE LOCL  D    0 ABS         main.c
  [2]  0x00000000 0x00000000  SECT LOCL  D    0 .bss
  [3]  0x00000000 0x00000000  SECT LOCL  D    0 .data
  [4]  0x00000000 0x00000004  OBJT LOCL  D    0 .data       a
  [5]  0x00000010 0x00000028  FUNC LOCL  D    0 .text       f1
  [6]  0x00000000 0x00000004  OBJT LOCL  D    0 .bss        c
  [7]  0x00000000 0x00000000  NOTY GLOB  D    0 UNDEF       f4
  [8]  0x00000000 0x00000000  NOTY GLOB  D    0 UNDEF       f3
  [9]  0x00000080 0x0000003c  FUNC GLOB  D    0 .text       main
 [10]  0x00000048 0x00000028  FUNC GLOB  D    0 .text       f2
 [11]  0x00000000 0x00000000  NOTY GLOB  D    0 UNDEF       b
 [12]  0x00000000 0x00000000  NOTY GLOB  D    0 ABS      __fsr_init_value
```

Advanced Topics

- Dynamic Linking
  - complete linking step at execution time
  - supports dynamic libraries
- Shared Objects
  - text segment shared by multiple processes
  - requires less physical memory

Symbol Resolution

- Linking must resolve all symbols

```
% gcc main.o
ld: Undefined symbol
  _b
  _f
  _g
```

- Linking must define each symbol once

```
% cp f.o g.o
% gcc main.o f.o g.o
ld: g.o: _f multiply defined
g.o: _g multiply defined
```

Example

main.c

```
static int a = 3;
extern int b;
static int c;

static int f1(int x) {
   return x + b;
}

int f2(int x) {
   return x + c;
}

int main() {
  f1(a);
  f2(a);
  f3(a);
  f4(a);
}
```

main.s

```
        .section ".data"
a:      .word 3

        .section ".bss"
c:      .skip 4

        .section ".text"
f1:     save    %sp,-96,%sp
        mov     %i0,%o1
        sethi   %hi(b),%o0
        ld      [%o0+%lo(b)],%o0
        add     %o1,%o0,%i0
        ret; restore

        .global f2
f2:     save    %sp,-96,%sp
        mov     %i0,%o1
        sethi   %hi(c),%o0
        ld      [%o0+%lo(c)],%o0
        add     %o1,%o0,%i0
        ret; restore

        .global main
main:   save    %sp,-96,%sp
        sethi   %hi(a),%l0
        or      %l0,%lo(a),%l0
        ld      [%l0+0],%l1
        mov %l1, %o0; call f1; nop
        mov %l1, %o0; call f2; nop
        mov %l1, %o0; call f3; nop
        mov %l1, %o0; call f4; nop
        ret; restore
```

Example

foo.c

```
int b = 1;

int f3(int x)
{
   int d = 0;
   return x + d;
}

int f4(int x)
{
   return x - b;
}
```

foo.s

```
        .section ".data"
        .global b
b:      .word 1

        .section ".text"
        .global f3
f3:     save    %sp,-104,%sp
        st      %g0,[%fp-8]
        ld      [%fp-8], %l0
        add     %i0,%l0,%i0
        ret; restore

        .global f4
f4:     save    %sp,-104,%sp
        sethi   %hi(b),%l1
        ld      [%l1+%lo(b)],%l0
        sub     %i0,%l0,%i0
        ret; restore
```