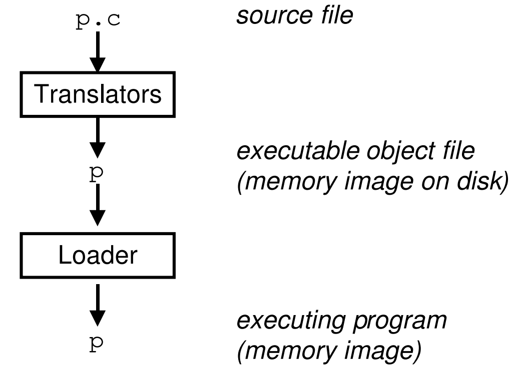
15-213

Introduction to Computer Systems

Program Translation and Execution I: Linking
Sept. 29, 1998

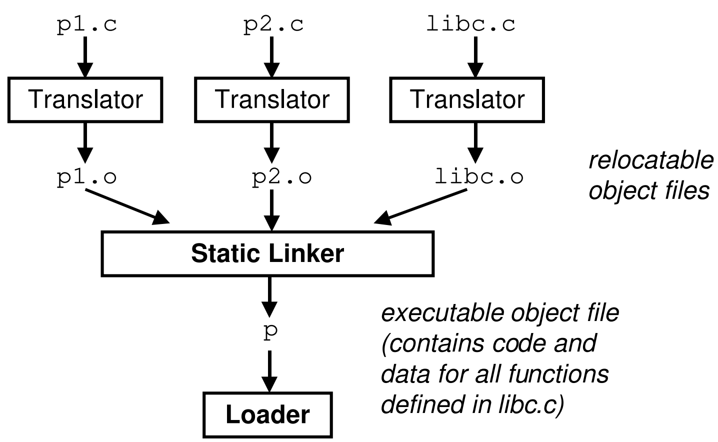
Topics
- **object files**
- **linkers**

A simplistic program translation scheme



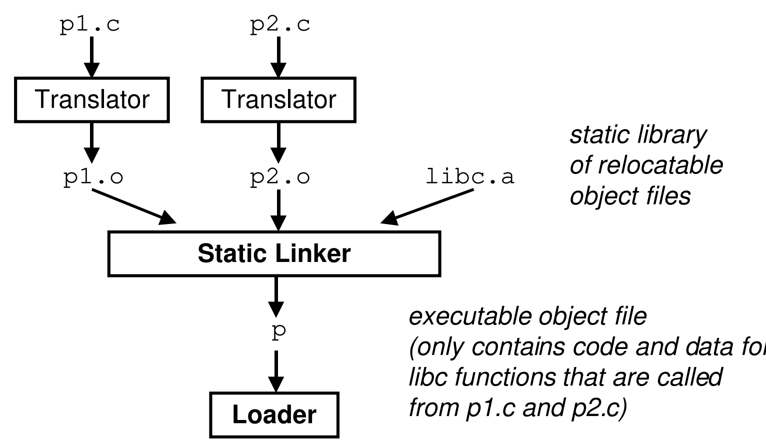
Problems:
- efficiency: small change requires complete recompilation
- modularity: hard to share common functionality (e.g. printf)

Separate compilation



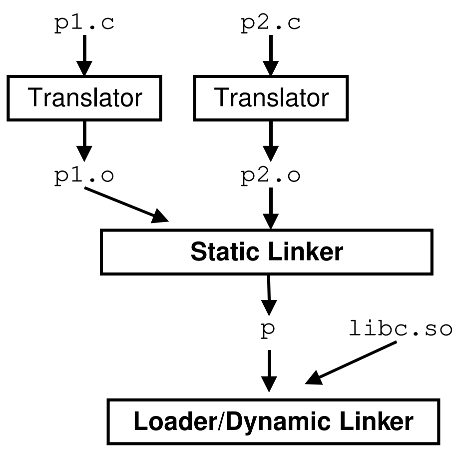
Improves modularity and efficiency, but still hard to package common functions

Static libraries



Further improves modularity and efficiency, but duplicates common functions in different processes

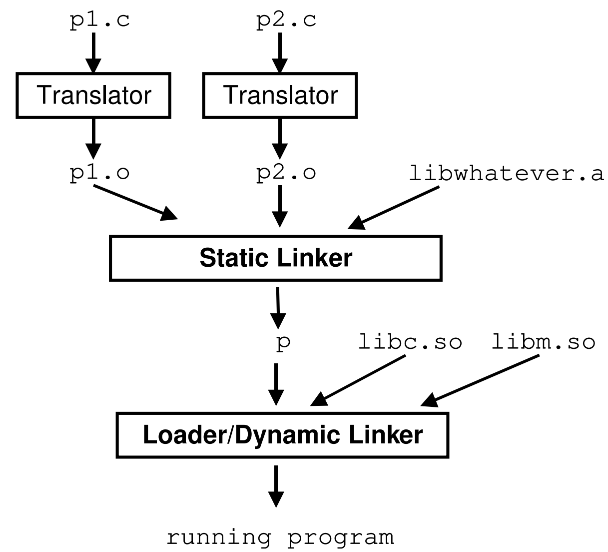
Shared libraries



shared library of dynamically relocatable object files

libc functions called by p1.c and p2.c are loaded at run-time and shared among processes.

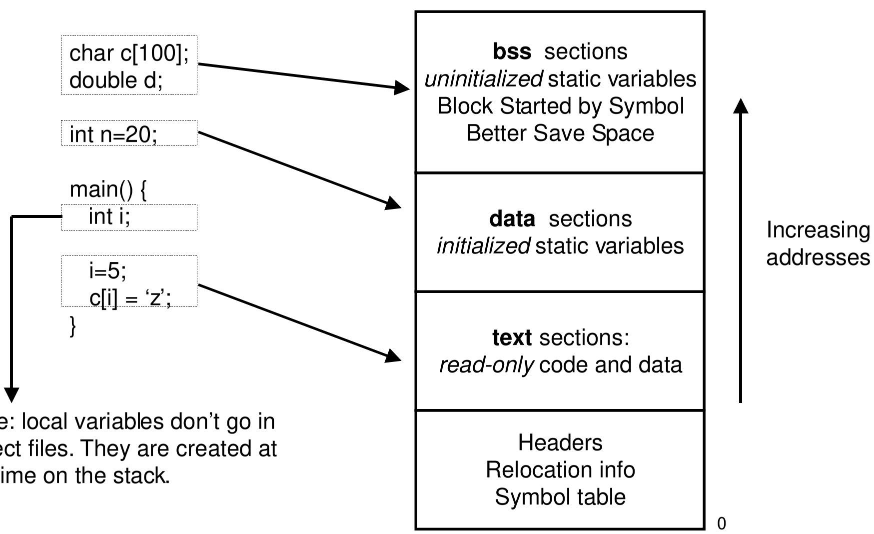
The complete picture



Object files

C Program

Object file



Note: local variables don't go in object files. They are created at runtime on the stack.

Example C program

main.c:

```
main() {
  zip();
}
```

zip.c:

```
extern int x;

int *p = &x;

zip() {
  code(p);
}
```

code.c:

```
int x=0xfe;

code(int *p) {
  srand(*p);
}
```

Compiling and linking the example program

% cc -v -c main.c
cfe main.c > /tmp/ctmfaacgja
ugen -O1 /tmp/ctmfaacgja -o /tmp/ctmcaacgja
as1 -O1 /tmp/ctmcaacgja -o main.o

% cc -c zip.c
% cc -c code.c

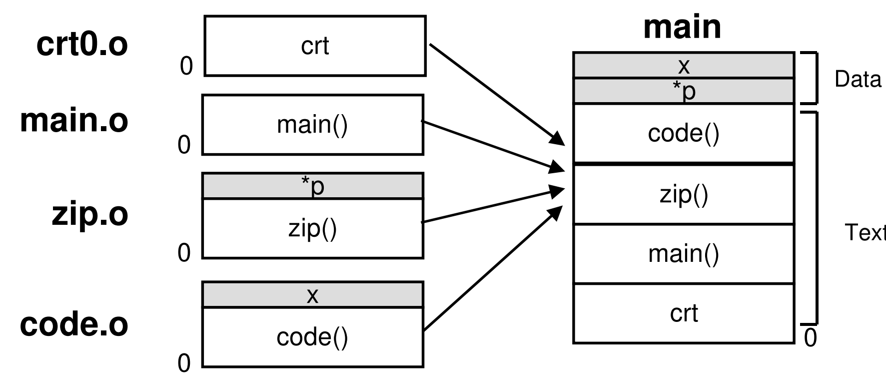
% cc -v -Wl,-O0 -o main main.o zip.o code.o
ld -o main -O0 -call_shared /usr/lib/cmplrs/cc/crt0.o main.o zip.o code.o -lc

Unix linker

C runtime system

Linker functions

1. *Resolving* external references
2. *Relocating* symbol definitions and references



Symbols

- **Lexical entities that correspond to functions and global variables**
- **Each symbol has a *value* (address)**
- **Code consists of symbol *definitions* and *references***
- **References can be either *local* or *external***

main.c:

```
main() {
  zip();
}
```

Ref to external symbol zip

zip.c:

```
extern int x;

int *p = &x;

zip() {
  code(p);
}
```

Def of local symbol p

Def of local symbol zip

Ref to local symbol p

Ref to external symbol x

code.c:

```
int x=0xfe;

code(int *p) {
  srand(*p);
}
```

Def of local symbol x

Ref to external symbol srand (defined in libc.so)

Strong and weak symbols

Program symbols are either *strong* or *weak*

- **strong: procedures and ininitialized globals**
- **weak: uninitialized globals**

p1.c:

strong → `int foo=5;`

strong → `p1() {`

```
int foo=5;

p1() {
}
```

p2.c:

```
int foo;

p2() {
}
```

weak → `int foo;`

strong → `p2() {`

Linker's symbol rules

1. A strong symbol can only appear once.

2. A weak symbol can be overridden by a strong symbol of the same name.

3. If multiple weak symbols, the linker can pick either one.

Linker puzzles

int foo; p1() {}	p1() {}
int foo; p1() {}	int foo; p2() {}
int foo; p1() {}	double foo; p2() {}
int foo=7; p1() {}	double foo; p2() {}
int foo=7; p1() {}	int foo; p2() {}

Alpha ECOFF object files

Symbol table
Section relocation info
Section data
Section headers
Optional header
File header

0

File and optional headers

- **magic number, size and number of sections, flags**
  - byte ordering, executable or relocatable?, shared?, calls shared library functions?
- **entry point, gp value, start and sizes of text, data, and bss segments**

Section headers

- **for each section in "Section data": address, size, number of relocation entries**

Section data

- **program code and data**

Section relocation info

- **which instructions and data items will need to be relocated?**

Symbol table

- **type and value of relevant symbols**

Section data (relocatable object files)

(Runtime memory image)	(File format)	
bss segment uninitialized variables (block started by symbol) (better save space!)	.bss	uninitialized data
	.sbss	small (4-byte) bss
data segment initialized variables	.pdata/.xdata	support for exceptions
	.lit4	4-byte literal pool
	.lit8	8-byte literal pool
	.lita	8-byte literal address pool
	.rconst	read only constants
	.rdata	read only data
	.sdata	small data
	.data	large data
text segment machine code	.init/.fini	process init/deinit code
	.text	machine instructions

.text section (main.o)

```
main() {
  zip();
}
```

	start	size
.text	0x00 (0)	0x30 (48)
.lita	0x50 (80)	0x10 (16)

```
      main:
 0x0:  lda   sp, -16(sp)
 0x4:  ldah  gp, 1(t12)
 0x8:  lda   gp, -32704(gp)
 0xc:  stq   ra, 0(sp)
 0x10: ldq   t12, -32752(gp)
 0x14: jsr   ra, (t12), zip
 0x18: ldah  gp, 1(ra)
 0x1c: ldq   ra, 0(sp)
 0x20: bis   zero, zero, v0
 0x24: lda   gp, -32728(gp)
 0x28: lda   sp, 16(sp)
 0x2c: ret   zero, (ra), 1
```

Assembler assumes that gp=32,832=(64K+128)/2 (Note that t12 = pc-4)

t12 = address stored at .lita+0

resets gp to 32,832 (Note: ra = pc)

Question: Why use gp-relative referencing of data and functions?

Symbol table (main.o)

```
main() {
  zip();
}
```

	start	size
.text	0x00 (0)	0x30 (48)
.lita	0x50 (80)	0x10 (16)

[Index]	Name	Value	Sclass	Symtype
[0]	main	0x0	Text	Proc
[1]	zip	0x0	Undefined	Proc

Relocation entries (main.o)

```
main() {
  zip();
}
```

	start	size
.text	0x00 (0)	0x30 (48)
.lita	0x50 (80)	0x10 (16)

Vaddr	Symndx	Type	Extern	Name
.text:				
0x04	4	GPDISP	local	
0x10	13	LITERAL	local	.lita
0x14	3	LITUSE	local	R_LU_JSR
0x14	1	HINT	extern	zip
0x18	12	GPDISP	local	
.lita:				
0x50	1	REFQUAD	extern	zip

GPDISP : instruction pair that sets gp value

LITERAL: gp-relative reference to literal in .lita

LITUSE: identifies instance of a literal address previously loaded into a register

HINT: 14 bit jsr hint reference

REFQUAD: 64-bit reference to the symbol's virtual address

Relocations (main.o)

```
main() {
  zip();
}
```

	start	size
.text	0x00 (0)	0x30 (48)
.lita	0x50 (80)	0x10 (16)

```
      main:
 0x0:  lda   sp, -16(sp)
 0x4:  ldah  gp, 1(t12)
 0x8:  lda   gp, -32704(gp)
 0xc:  stq   ra, 0(sp)
 0x10: ldq   t12, -32752(gp)
 0x14: jsr   ra, (t12), zip
 0x18: ldah  gp, 1(ra)
 0x1c: ldq   ra, 0(sp)
 0x20: bis   zero, zero, v0
 0x24: lda   gp, -32728(gp)
 0x28: lda   sp, 16(sp)
 0x2c: ret   zero, (ra), 1
```

Vaddr	Symndx	Type	Extern	Name
.text:				
0x04	4	GPDISP	local	
0x10	13	LITERAL	local	.lita
0x14	3	LITUSE	local	R_LU_JSR
0x14	1	HINT	extern	zip
0x18	12	GPDISP	local	
.lita:				
0x50	1	REFQUAD	extern	zip

Notes:
1. LITUSE at 0x14 allows jsr to be replaced by bsr

.text section (zip.o)

```
extern int x;
int *p = &x;

zip() {
  code(p);
}
```

	start	size
.text	0x00 (0)	0x40 (64)
.lita	0x60 (96)	0x10 (16)
.sdata	0x70 (112)	0x10 (16)

```
0x0:  lda   sp, -16(sp)
0x4:  ldah  gp, 1(t12)          <- Assembler assumes
0x8:  lda   gp, -32688(gp)         that gp=32,848
0xc:  stq   ra, 0(sp)
0x10: ldq   t12, -32744(gp)     <- t12 = addr(code) = cont(.lita+8)
0x14: ldq   a0, -32752(gp)      <- a0 = addr(p) = cont(.lita+0)
0x18: ldq   a0, 0(a0)           <- a0 = p
0x1c: jsr   ra, (t12), code
0x20: ldah  gp, 1(ra)
0x24: ldq   ra, 0(sp)
0x28: lda   gp, -32720(gp)
0x2c: lda   sp, 16(sp)
0x30: ret   zero, (ra), 1
```


Symbol table (zip.o)

```
extern int x;
int *p = &x;

zip() {
  code(p);
}
```

	start	size
.text	0x00 (0)	0x40 (64)
.lita	0x60 (96)	0x10 (16)
.sdata	0x70 (112)	0x10 (16)

[Index]	Name	Value	Sclass	Symtype
[0]	p	0x70	SData	Global
[1]	x	0x04	SUndefined	Global
[2]	zip	0x00	Text	Proc
[3]	code	0x00	Undefined	Proc

Relocation entries (zip.o)

```
extern int x;
int *p = &x;

zip() {
  code(p);
}
```

	start	size
.text	0x00 (0)	0x40 (64)
.lita	0x60 (96)	0x10 (16)
.sdata	0x70 (112)	0x10 (16)

```
0x0:  lda   sp, -16(sp)
0x4:  ldah  gp, 1(t12)
0x8:  lda   gp, -32688(gp)
0xc:  stq   ra, 0(sp)
0x10: ldq   t12, -32744(gp)
0x14: ldq   a0, -32752(gp)
0x18: ldq   a0, 0(a0)
0x1c: jsr   ra, (t12), code
0x20: ldah  gp, 1(ra)
0x24: ldq   ra, 0(sp)
0x28: lda   gp, -32720(gp)
0x2c: lda   sp, 16(sp)
0x30: ret   zero, (ra), 1
```

Vaddr	Symndx	Type	Extern	Name
.text:				
0x04	4	GPDISP	local	
0x14	13	LITERAL	local	.lita
0x18	1	LITUSE	local	R_LU_BASE
0x10	13	LITERAL	local	.lita
0x1c	3	LITUSE	local	R_LU_JSR
0x1c	3	HINT	extern	code
0x20	8	GPDISP	local	
.lita:				
0x60	0	REFQUAD	extern	p
0x68	3	REFQUAD	extern	code
.sdata:				
0x70	1	REFQUAD	extern	x

.text section (code.o)

```
int x=0xfe;
code(int *p) {
  srand(*p);
}
```

	start	size
.text	0x00 (0)	0x40 (64)
.lita	0x60 (96)	0x10 (16)
.sdata	0x70 (112)	0x10 (16)

```
0x0:  lda   sp, -64(sp)
0x4:  ldah  gp, 1(t12)          <- Assembler assumes
0x8:  lda   gp, -32688(gp)         that gp=32,848
0xc:  stq   ra, 0(sp)
0x10: stq   a0, 16(sp)
0x14: ldq   t12, -32752(gp)     <- t12 = addr(srand) = cont(.lita+0)
0x18: bis   a0, a0, t0
0x1c: ldl   a0, 0(t0)
0x20: jsr   ra, (t12), srand    <- srand is unresolved
0x24: ldah  gp, 1(ra)
0x28: ldq   ra, 0(sp)
0x2c: lda   gp, -32724(gp)
0x30: lda   sp, 64(sp)
0x34: ret   zero, (ra), 1
```

Symbol table (code.o)

```
int x=0xfe;
code(int *p) {
  srand(*p);
}
```

	start	size
.text	0x00 (0)	0x40 (64)
.lita	0x60 (96)	0x10 (16)
.sdata	0x70 (112)	0x10 (16)

[Index]	Name	Value	Sclass	Symtype
[0]	x	0x70	SData	Global
[1]	code	0x00	Text	Proc
[2]	srand	0x00	Undefined	Proc

Relocation entries (code.o)

```
int x=0xfe;
code(int *p) {
  srand(*p);
}
```

	start	size
.text	0x00 (0)	0x40 (64)
.lita	0x60 (96)	0x10 (16)
.sdata	0x70 (112)	0x10 (16)

```
0x0:  lda   sp, -64(sp)
0x4:  ldah  gp, 1(t12)
0x8:  lda   gp, -32688(gp)
0xc:  stq   ra, 0(sp)
0x10: stq   a0, 16(sp)
0x14: ldq   t12, -32752(gp)
0x18: bis   a0, a0, t0
0x1c: ldl   a0, 0(t0)
0x20: jsr   ra, (t12), srand
0x24: ldah  gp, 1(ra)
0x28: ldq   ra, 0(sp)
0x2c: lda   gp, -32724(gp)
0x30: lda   sp, 64(sp)
0x34: ret   zero, (ra), 1
```

Vaddr	Symndx	Type	Extern	Name
.text:				
0x04	4	GPDISP	local	
0x14	13	LITERAL	local	.lita
0x20	3	LITUSE	local	R_LU_JSR
0x20	2	HINT	extern	srand
0x24	8	GPDISP	local	
.lita:				
0x60	2	REFQUAD	extern	srand

Why no relocation entry for x?

Executable object file (main)

.rconst	0x120001080 (4224)	0x60 (96)
.text	0x120001140 (4416)	0x450 (1104)
.data	0x140000000 (0)	0x10 (16)
.lit8	0x1400000a0 (160)	0x10 (16)
.sdata	0x140000b0 (176)	0x20 (32)
.got	0x140000d0 (208)	0x120 (288)
.sbss	0x1400001f0 (496)	0x40 (64)

```
   main:
0x120001320:  lda   sp, -16(sp)
0x120001324:  ldah  gp, 8192(t12)     <-- Linker sets gp at
0x120001328:  lda   gp, 28064(gp)         .data+32960
0x12000132c:  stq   ra, 0(sp)
0x120001330:  ldq   t12, -32672(gp)   <-- addr(code) at .data+288
0x120001334:  ldah  a0, -1(gp)
0x120001338:  ldq   a0, 32752(a0)
0x12000133c:  bsr   ra, code          <-- changed by linker!
0x120001340:  ldah  gp, 8192(ra)
0x120001344:  ldq   ra, 0(sp)
0x120001348:  lda   gp, 28032(gp)
0x12000134c:  lda   sp, 16(sp)
0x120001350:  ret   zero, (ra), 1
```


Symbol table (main)

```
[0] .rconst   0x120001080   scRConst   Local
[1] .pdata    0x1200010e0   PData      Local
[2] .text     0x120001140   Text       Local
[3] .init     0x120001590   Init       Local
[4] .fini     0x1200015d0   Fini       Local
[5] .data     0x140000000   Data       Local
[6] .xdata    0x140000010   XData      Local
[7] .rdata    0x140000010   RData      Local
[8] .lit8     0x1400000b0   SData      Local
[9] .lit4     0x1400000b0   SData      Local
[10] .sdata   0x1400000b0   SData      Local
[11 .sbss     0x1400001f0   SBss       Local
[12] .bss     0x140000230   Bss        Local
```

Symbol table (main)

```
[13]    __start   0x1200011a0   Text       Proc
[24]    x         0x1400000c0   SData      Global
[25]    __environ 0x000000000   Common     Global
[26]    errno     0x000000004   Common     Global
[29]    exit      0x120001140   Undefined  Proc
[31]    main      0x1200012f0   Text       Proc
[32]    zip       0x120001320   Text       Proc
[33]    code      0x120001360   Text       Proc
[42]    p         0x1400000b0   SData      Global
[43]    __Argc    0x1400001f0   SBss       Global
[44]    __Argv    0x1400001f8   SBss       Global
[56]    srand     0x000000000   Undefined  Proc
```