CS 354 - Machine Organization & Programming
Tuesday May 2, and Thursday May 4, 2023

Course Evals

https://aefis.wisc.edu Course: CS354 Instructor: DEPPELER

Final Exam -Wednesday May 10th, 2:45 PM - 4:45 PM

See Exams Page for more information -- cummulative with focus on material since E2.
Exam Room information sent via email -- Bring to lecture and fill out scantron correctly
Arrive early if possible with UW ID and #2 pencils. See exam info on course web site.

Homework hw8: DUE on Monday May 1st

Homework hw9: DUE on Wednesday May 3rd

Project p6: Due on last day of classes (NO LATE PERIOD or OOPS). If you plan on getting help in labs, be sure to bring your own laptop in case there is no workstation available.

Last Week

Meet Signals Three Phases of Signaling Processes IDs and Groups Sending Signals Receiving Signals	Issues with Multiple Signals Forward Declaration Multifile Coding Multifile Compilation Makefiles

This Week

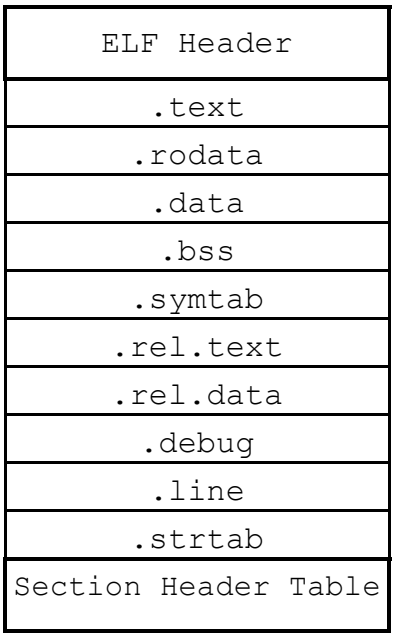
Makefiles (from last week) Relocatable Object Files Static Linking Linker Symbols Linker Symbol Table Symbol Resolution	Resolving Globals Symbol Relocation Executable Object File Loader What's next? take OS cs537 as soon as possible and Compilers cs536, too!
Next Week: FINAL EXAM	

Relocatable Object Files (ROFs)

What? A *relocatable object* file is

-
-

Executable and Linkable Format (ELF)



ELF Header

Section Header Table (SHT)

Static Linking

What? *Static linking*

static vs. dynamic

executable size:

library code:

How?

→ What issues arise from combining ROFs?

1. variable and function identifiers

2. variable and function identifiers

Making Things Private

→ Are functions and global variables only in a source file actually private if they're not in the correspondng header file?

→ How do you make them truly private?

Linker Symbols

What?

Symbols

Linker Symbols

→ Which kinds of variables need linker symbols?

1. local variables
2. `static` local variables
3. parameter variables
4. global variables
5. `static` global variables
6. `extern` global variables

→ Which kinds of functions need linker symbols?

1.

2.

3.

Linker Symbol Table

What? The linker ***symbol table*** is

-
-

`ELF_Symbol` Data Members and their Use

`int name`

`int value`

if ROF

if EOF

`int size`

`char type:4`

`binding:4`

`char section`

pseudo sections:
ABS:olute

UND:efined

COM:mon

`value`
`size`

Example

```
Num:  Value Size  Type    Bind    Ot Ndx   Name
1 - 7 not shown
8:    0     4     OBJECT  GLOBAL  0  3     bufp0
9:    0     0     NOTYPE  GLOBAL  0  UND   buf
10:   0     39    FUNC    GLOBAL  0  1     swap
11:   4     4     OBJECT  GLOBAL  0  COM   bufp1
```

→ Is `bufp0` initialized?

→ Was `buf` defined in the source file or declared `extern`?

→ What is the function's name?

→ What is the alignment and size of `bufp1`?

Symbol Resolution

What? *Symbol resolution*

-
-

Compiler's Resolution Work

- locals

  `static` locals

- globals

  `static` globals

❋ *If a global symbol is only declared in this source file*

Linker's Resolution Work

- `static` locals

- globals

❋ *If a global symbol is not defined or is multiply defined*

Resolving Globals

Globals - ODR=One Definition Rule

main.c

```
int m;
int n = 11;
short o;

extern int x;
int y;
static int z = 66;

//code continues...
```

funit1.c

```
int m = 22;
int n;
int o;

int x;
static int y = 33;
static int z = 77;

//code continues...
```

funit2.c

```
int m;
extern int n;
char o;

static int x = 33;
static int y;
int z;

//code continues...
```

- *What happens if multiple definitions of a variable identifier?*
- *Use `extern` to indicate when*
- *Use `static` to indicate when*

TEXTBOOK and OLD NOTES may describe old rules for resolving globals

~~**Strong and Weak Symbols**~~

~~strong: function definitions and initialized global variables~~

~~weak: function declarations and uninitialized global variables~~

→ ~~Which code statements above correspond to strong symbols?~~

~~**Rules for Resolving Globals**~~

→ ~~Which code statements above correspond to definitions?~~
~~Recall: `extern` is only a declaration~~

~~1. Multiple strong symbols~~
~~linker errorRecall: `static` makes a global private, i.e., only visible within its source file)~~

~~2. Given one strong symbol and one or more weak symbols,~~

~~3.Given only weak symbols,~~
~~dangerous with different types,to avoid use gcc -fno-common~~

Symbol Relocation

What? *Symbol relocation*

How?

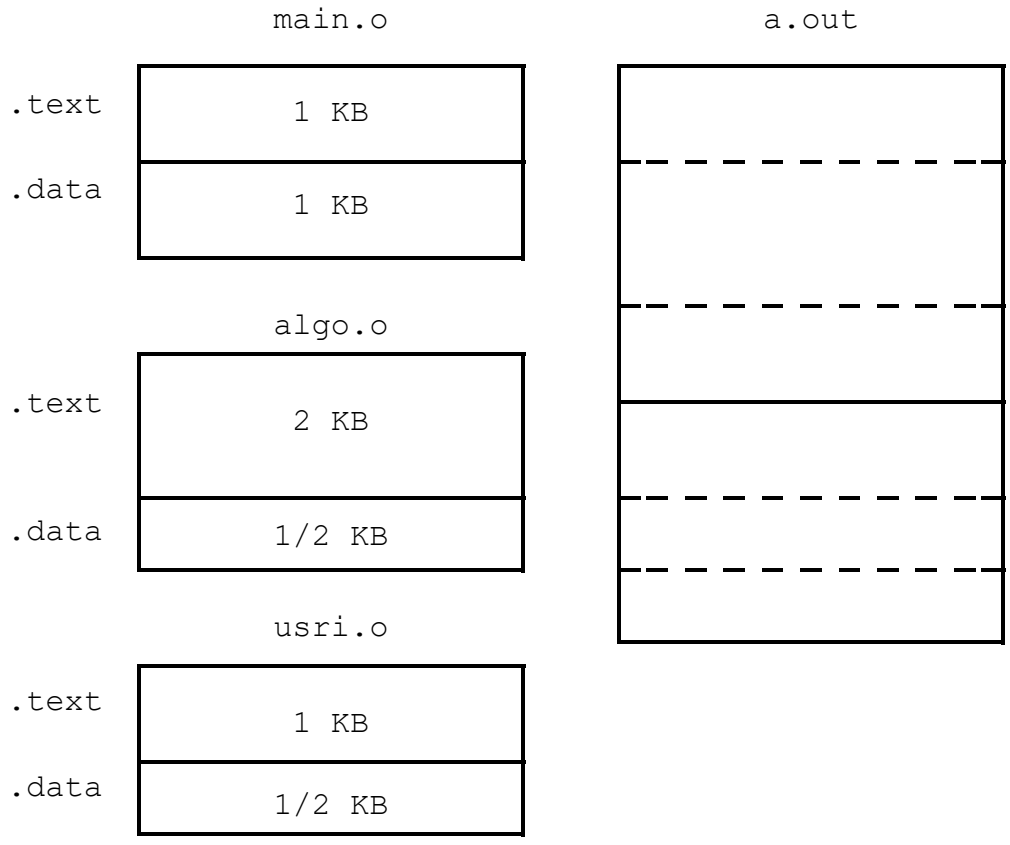
1. Merges the same sections

2. Assigns virtual addresses

3. Updates symbol references

Example

Consider the .text and .data sections of 3 object files below combined into an executable:



address =

Excutable Object File (EOF)

What? An EOF, like an ROF, is

Executable and Linkable Format

ELF Header

+ Segment Header Table

ELF Header
Segment Header Table
.init
.text
.rodata
.data
.bss
.symtab
.debug
.line
.strtab
Section Header Table

→ Why aren't there relocation sections (.rel.text or .rel.data) in EOF?

➢ Why is the data segment's size in memory larger than its size in the EOF?

Loader

What? The *loader*

-
-

Loading

1.
2.

Execution - the final story

1. shell
2. child process
3. loader
   a.
   b.
   c.
   d.
4. loader

```
call __libc_init_first
call _init
call atexit
call main
call _exit
```