CS 354 - Machine Organization & Programming
Tuesday December 10th, 2024

Course Evals

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

Final Exam - Thursday Dec 19th, 10:05 AM - 12:05 PM

Your final exam room has been sent to you via email (or will be shortly).
You must attend the exam room as assigned in the email you receive.
Arrive early if possible with UW ID and #2 pencils. See additional exam info on course web site.

All office hours, TA consulting, and Peer Mentoring end on Wed December 11th

Homework hw8: DUE on or before Monday Dec 9 (late day Tuesday)

Homework hw9: DUE on or before Wednesday Dec 11 (NO LATE DAY)

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

Learning Objectives

- understand and describe how compiler resolves symbols across multiple source files
- understand and describe why relocation is necessary and how it occurs
- understand and describe what the Loader is and does

This Week

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 Watch your email for your exam room assignment. All students must take the final exam in their assigned final exam room. **Students with accommodations should have or will receive email with their exam date/time/venue.**	

Resolving Globals

Confusing Globals

main.c

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

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

//code continues...
```

fun1.c

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

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

//code continues...
```

fun2.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 an identifier exist?*
- *Use `extern` to clearly indicate when*
- *Use `static` to clearly indicate when*

TEXTBOOK and OLD NOTES describe old rules for resolving global variables.

~~Strong and Weak Symbols~~ (no such thing any more, use extern when defined elsewhere)

~~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
Note: `extern` vars must be defined in another file, otherwise undefined symbol linker error.

1. Multiple symbol defns in public gobal scope are not allowed
   linker error -> mult defined symbol
   Recall: `static` makes a global private, i.e., only visible within its source file)

2. Define one symbol in one file, and declare other with `extern`
   Use **`gcc -z muldefs`** to ignore uninitialized symbols defined in multiple files.

Symbol Relocation

What? *Symbol relocation*

How?

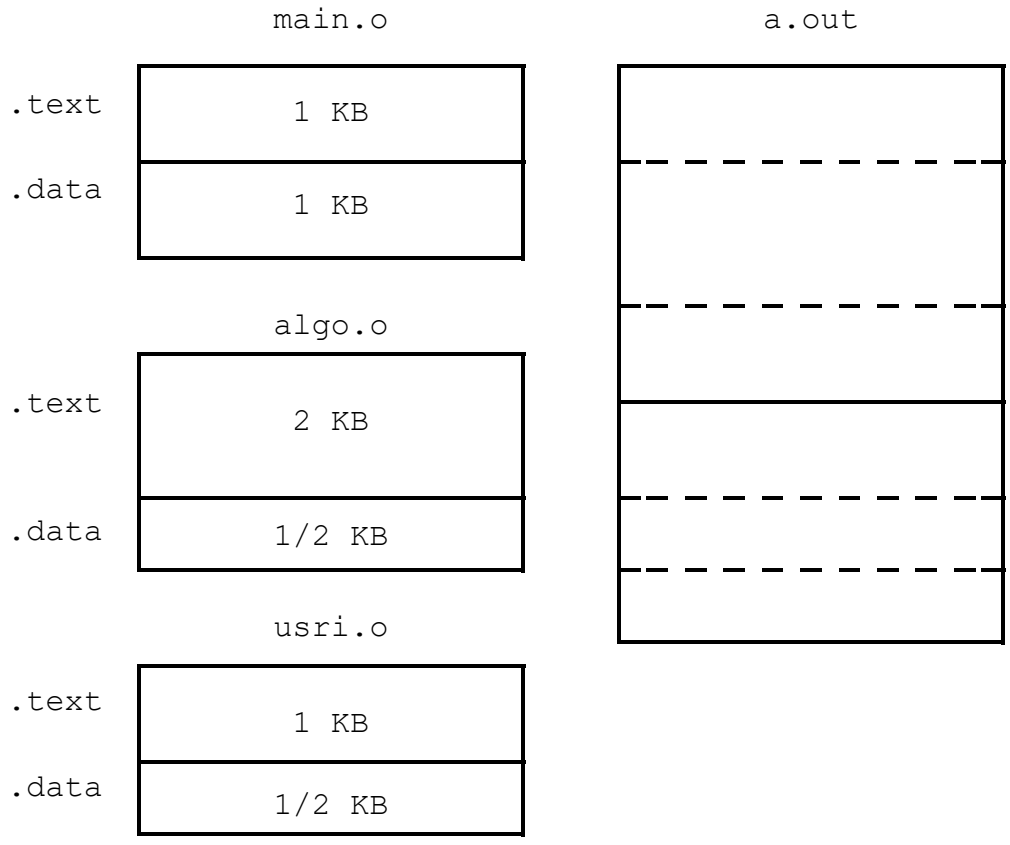
1. Merge the same sections of ROFs into one aggregate for each section type.

2. Assign virtual addresses to each aggregate section and each symbol definition.

3. Update symbol references listed in ROF relocation sections (.rel.text, .rel.data).

Example

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



address = 1 start of section + 2 offset to subsection + 3 offset within subsection

calc

cnt

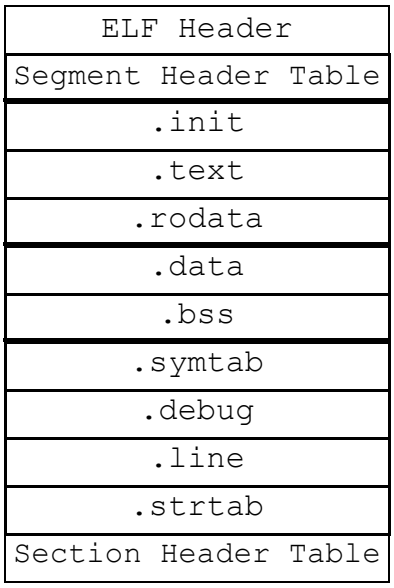
Excutable Object File (EOF)

What? An EOF, like an ROF, is

Executable and Linkable Format

ELF Header
+ entry point = addr of 1st instr

+ Segment Header Table
+ info for each segment to be loaded into mem during execution
offset in file
alignment
page size
size in file and size in mem
run-time permissions



→ Why aren't there relocation sections (.rel.text or .rel.data) in EOF?
since we've assumed static linking, all symbol relocations are done

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

Loader

What? The *loader*

- is kernel code that
- can be invoked by

Loading

1. "copies" code and data segments from EOF into memory
2. starts program executing by jumping to its entry point

Execution - the final story

1. shell creates a child process with `fork()`
2. child process invokes loader with `execve()`
3. loader creates the new runtime memory image
   a. deletes curr segments code, data, heap, stack
   b. creates new segments
   c. heap and stack initialized to size 0
   d. EOF's code and data segments are mapped in page table into page-sized chunks based on Segment Hdr Table BUT THESE ARE NOT COPIED INTO MEM except some header info
4. loader

_start:

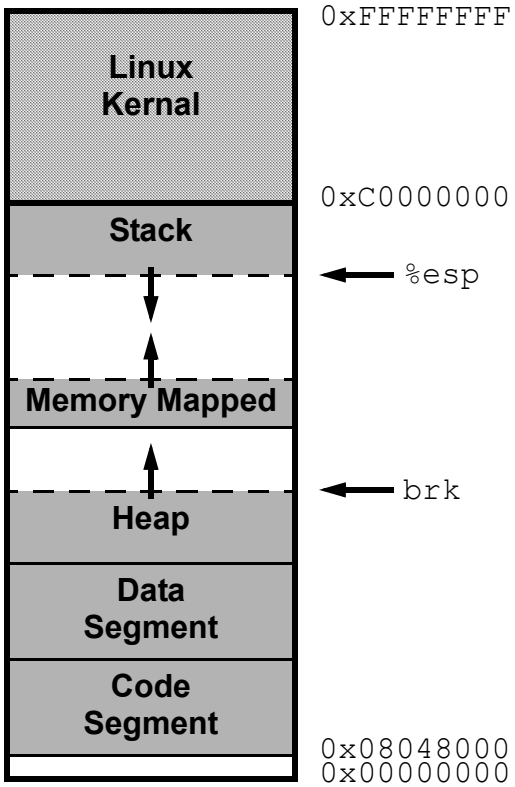
```
call __libc_init_first

call _init

call atexit

call main

call _exit
```



CS354 Project Reminders

p1 Building and running an executable from C source code
p2A reading a file with 2D array data and checking the contents meet requirements
p2B implementing algorithm to fill 2D array and writing 2D array to a file
p3A implementing alloc for a dynamically allocated memory space (heap)
p3B implementing free with immediate coalescing (heap)
p4A analyzing cache performance for large 2D array access in various sequences (strides)
p4B implementing a cache simulator for any sequence of memory access and cache config.
p5 disassembling Linux executable and tracing ASM to find input to open a safe.
p6 handle SIGINT SIGUSR1 SIGFPE signals, send signals via command line and syscalls

Exam 3 Notice: https://canvas.wisc.edu/courses/412449/pages/exam-3-notice

Expected Final Exam Format:

- (28) 3 pt Multiple Choice questions
- 84 pts total, 25% of overall weighted percentage.
- Able to manually tracing C and IA-32 x86 ASM code
- multiply and divide using powers of two,
- converting hex to/from binary and decimal to/from binary.
- understand all memory and other diagrams to understand info provided via diagram.
- plan for writing your work on exam without type code and run.
- no code writing is expected or planned on final exam

What's Next? For best recall, take these courses soon after cs354 as possible.

- cs536 Intro to Compilers
- cs537 Intro to Operating Systems

CS 354 - Machine Organization & Programming
Tuesday December 10th, 2024

Course Evals

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

Final Exam - Thursday Dec 19th, 10:05 AM - 12:05 PM

Your final exam room has been sent to you via email (or will be shortly).
You must attend the exam room as assigned in the email you receive.
Arrive early if possible with UW ID and #2 pencils. See additional exam info on course web site.

All office hours, TA consulting, and Peer Mentoring end on Wed December 11th

Homework hw8: DUE on or before Monday Dec 9 (late day Tuesday)

Homework hw9: DUE on or before Wednesday Dec 11 (NO LATE DAY)

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

Learning Objectives

- understand and describe how compiler resolves symbols across multiple source files
- understand and describe why relocation is necessary and how it occurs
- understand and describe what the Loader is and does

This Week

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 Watch your email for your exam room assignment. All students must take the final exam in their assigned final exam room. **Students with accommodations should have or will receive email with their exam date/time/venue.**	

Resolving Globals

Confusing Globals

main.c

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

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

//code continues...
```

fun1.c

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

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

//code continues...
```

fun2.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 an identifier exist?*
- *Use `extern` to clearly indicate when*
- *Use `static` to clearly indicate when*

TEXTBOOK and OLD NOTES describe old rules for resolving global variables.

~~Strong and Weak Symbols~~ (no such thing any more, use extern when defined elsewhere)

~~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
Note: `extern` vars must be defined in another file, otherwise undefined symbol linker error.

1. Multiple symbol defns in public gobal scope are not allowed
   linker error -> mult defined symbol
   Recall: `static` makes a global private, i.e., only visible within its source file)

2. Define one symbol in one file, and declare other with `extern`
   Use **`gcc -z muldefs`** to ignore uninitialized symbols defined in multiple files.

Symbol Relocation

What? *Symbol relocation*

How?

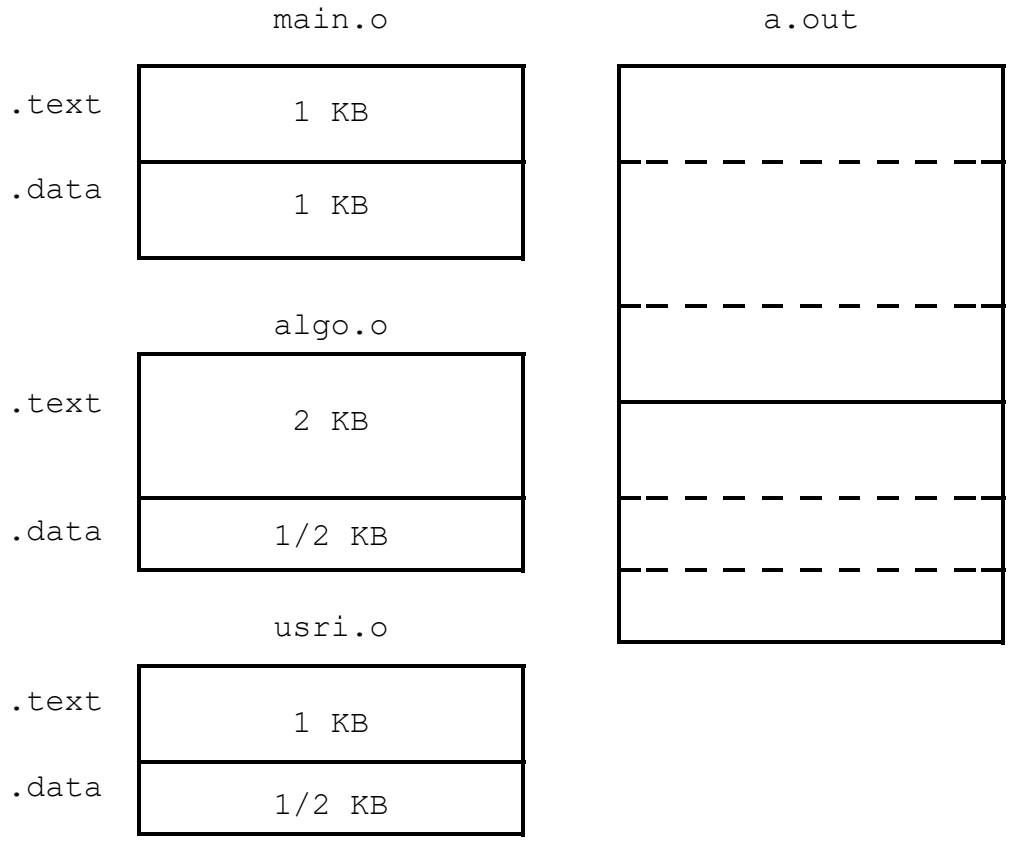
1. Merge the same sections of ROFs into one aggregate for each section type.

2. Assign virtual addresses to each aggregate section and each symbol definition.

3. Update symbol references listed in ROF relocation sections (.rel.text, .rel.data).

Example

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



address = 1 start of section + 2 offset to subsection + 3 offset within subsection

calc

cnt

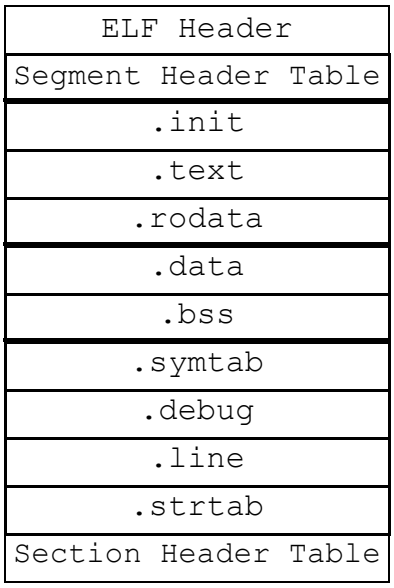
Excutable Object File (EOF)

What? An EOF, like an ROF, is

Executable and Linkable Format

ELF Header
+ entry point = addr of 1st instr

+ Segment Header Table
+ info for each segment to be loaded into mem during execution
offset in file
alignment
page size
size in file and size in mem
run-time permissions



→ Why aren't there relocation sections (.rel.text or .rel.data) in EOF?
since we've assumed static linking, all symbol relocations are done

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

Loader

What? The *loader*

- is kernel code that
- can be invoked by

Loading

1. "copies" code and data segments from EOF into memory
2. starts program executing by jumping to its entry point

Execution - the final story

1. shell creates a child process with `fork()`
2. child process invokes loader with `execve()`
3. loader creates the new runtime memory image
   a. deletes curr segments code, data, heap, stack
   b. creates new segments
   c. heap and stack initialized to size 0
   d. EOF's code and data segments are mapped in page table into page-sized chunks based on Segment Hdr Table BUT THESE ARE NOT COPIED INTO MEM except some header info
4. loader

_start:

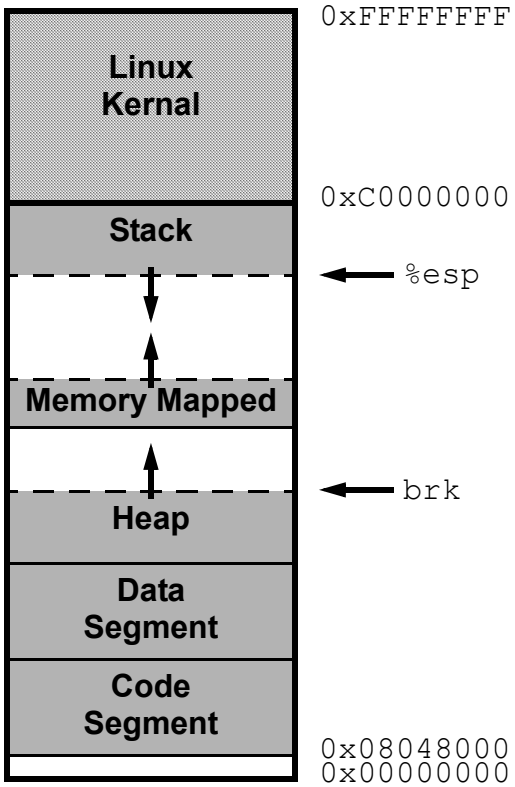
```
call __libc_init_first

call _init

call atexit

call main

call _exit
```



CS354 Project Reminders

p1 Building and running an executable from C source code
p2A reading a file with 2D array data and checking the contents meet requirements
p2B implementing algorithm to fill 2D array and writing 2D array to a file
p3A implementing alloc for a dynamically allocated memory space (heap)
p3B implementing free with immediate coalescing (heap)
p4A analyzing cache performance for large 2D array access in various sequences (strides)
p4B implementing a cache simulator for any sequence of memory access and cache config.
p5 disassembling Linux executable and tracing ASM to find input to open a safe.
p6 handle SIGINT SIGUSR1 SIGFPE signals, send signals via command line and syscalls

Exam 3 Notice: https://canvas.wisc.edu/courses/412449/pages/exam-3-notice

Expected Final Exam Format:

- (28) 3 pt Multiple Choice questions
- 84 pts total, 25% of overall weighted percentage.
- Able to manually tracing C and IA-32 x86 ASM code
- multiply and divide using powers of two,
- converting hex to/from binary and decimal to/from binary.
- understand all memory and other diagrams to understand info provided via diagram.
- plan for writing your work on exam without type code and run.
- no code writing is expected or planned on final exam

What's Next? For best recall, take these courses soon after cs354 as possible.

- cs536 Intro to Compilers
- cs537 Intro to Operating Systems