CS 354 - Machine Organization & Programming
Tuesday, December 10, 2019

Final Exam - Monday December 16th, 5:05 - 7:05 PM

- **A - H Last/Family Name:** room **2**650 Humanities
- **I - R Last/Family Name:** room 105 Psychology
- **S - Z Last/Family Name:** room B130 Van Vleck
- UW ID required
- #2 pencils required
- closed book, no notes, no electronic devices (e.g., calculators, phones, watches)
- see "Final Exam" on course site Assignments for topics

Project p6 (4.5%): DUE at 10 pm on Saturday, December 14th

Homework hw9 (1.5%): Due at 10 pm on Friday, December 13th

Homework hw10 (1.5%): DUE at 10 pm on Sunday, December 15th

sixteen 1/2pt questions, 2 attempts, 20 mins each

Last Time

Makefiles
Relocatable Object Files
Static Linking
Linker Symbols
Linker Symbol Table
Symbol Resolution

Today the last time!

Final Exam Information
Resolving Globals
Symbol Relocation
Executable Object File
Loading

Final Exam

The final is cumulative. Review your midterm exams.

Part I: 52 1-pt questions
Part II: 16 3-pt questions

Reference Page include in exam:

Powers of 2

$2^5 = 32$, $2^6 = 64$, $2^7 = 128$, $2^8 = 256$, $2^9 = 512$, $2^{10} = 1024$
$2^{10} = K$, $2^{20} = M$, $2^{30} = G$
$2^A \times 2^B = 2^{A+B}$
$2^A / 2^B = 2^{A-B}$

Hexadecimal Digits

$9_{16} = 9_{10} = 1001_2$
$A_{16} = 10_{10} = 1010_2$
$B_{16} = 11_{10} = 1011_2$
$C_{16} = 12_{10} = 1100_2$
$D_{16} = 13_{10} = 1101_2$
$E_{16} = 14_{10} = 1110_2$
$F_{16} = 15_{10} = 1111_2$

Registers

32 bit	16 bit	8 bit
%eax	%ax	%ah, %al
%ecx	%cx	%ch, %cl
%edx	%dx	%bh, %bl
%ebx	%bx	%dh, %dl
%edi	%di	
%esi	%si	
%ebp	%bp	
%esp	%sp	

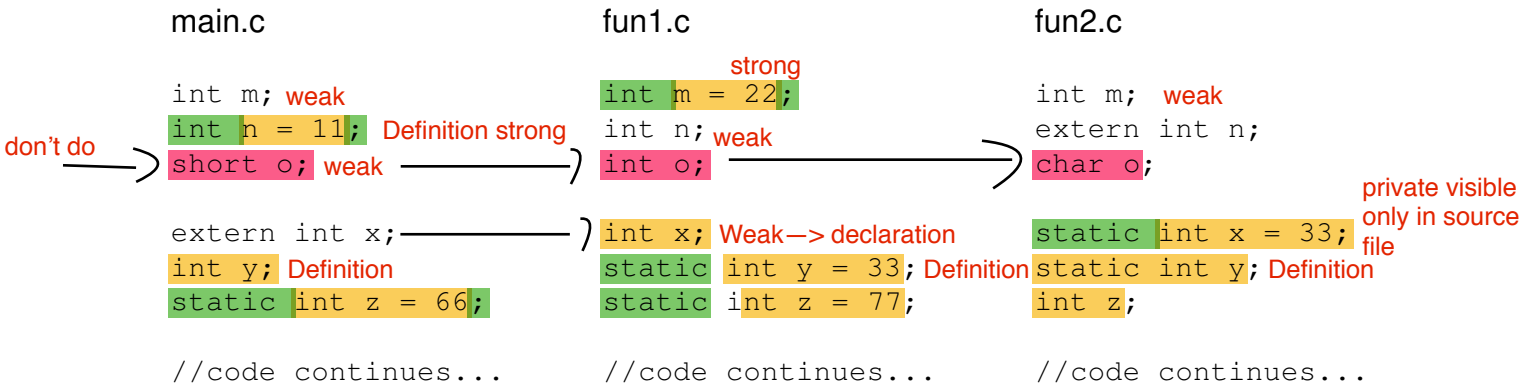
32 bit registers: underlined are caller saved, others are callee saved

Assembly

Most instructions with two operands have the order: Source, Destination
e.g., `subl s,d` means d = d - s; `imull s,d` means d = d * s
Comparison (cmp) and test instructions have operand order: Source2, Source1
e.g., `cmpl s2,s1` means s1 - s2; `test s2,s1` means s1 & s2
Suffixes for set, jump, and conditional move instruction are:
e.g., `setns` means set not signed, `jl` means jump if less
general - `e`: equal, `ne`: not equal, `s`: signed, `ns`: not signed
unsigned - `b`: below, `be`: below or equal, `a`: above, `ae`: above or equal
signed - `l`: less, `le`: less or equal, `g`: greater, `ge`: greater or equal

Resolving Globals

Confusing Globals



Strong and Weak Symbols

strong: Function definitions and initialized global variables

weak: Functions declarations and un initializations global variables

→ Which code statements above correspond to strong symbols?

Rules for Resolving Globals

→ Which code statements above correspond to definitions?

1. Multiple strong symbols In "Public", global scope are not allowed

   Result in a linker error

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

3. Given only weak symbols, Linker can choose any one

   Dangerous with different types with the same global variable.

   To avoid to use: gcc -fno-common

❋ *Use `extern` to clearly indicate when* a global variable is only a declarations

❋ *Use `static` to clearly indicate when* A global variable or function is meant to be private to its source file

Symbol Relocation

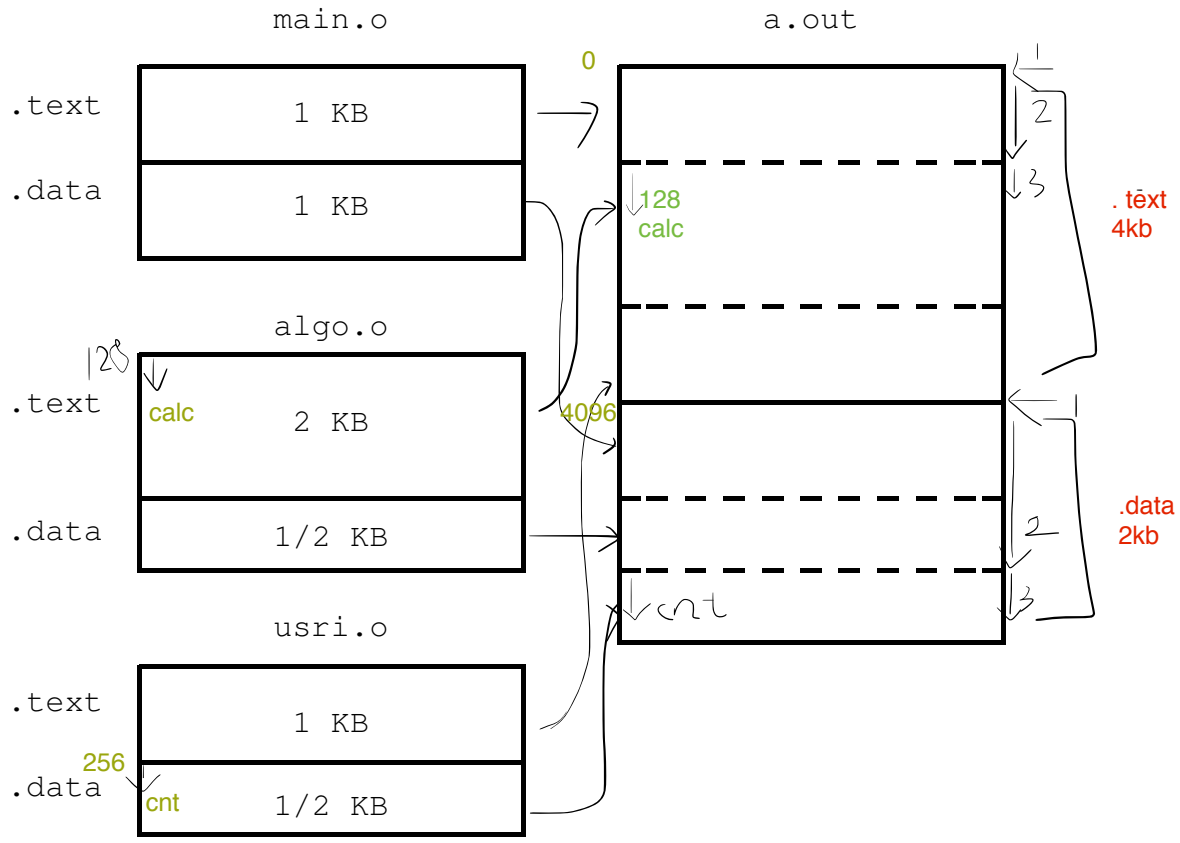
What? *Symbol relocation* Combines ROFs / SOFs so that addresses can be determined for linker symbols.

How?

1. Merges the same sections of ROFs into an aggregate for each section type.

2. Assigns virtual addresses to each aggregate section and then each symbol definition.

3. Updates symbol references as listed in ROF's relocation sections
   sections. (.rel.text, .rel.data)



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

base 10 cal= 0 + 1024 + 128

cnt= 4096 + 1536 + 256

Excutable Object File (EOF)

What? An EOF, like an ROF, is

a file produced by linker containing object code and it can be loaded into memory and run

Executable and Linkable Format

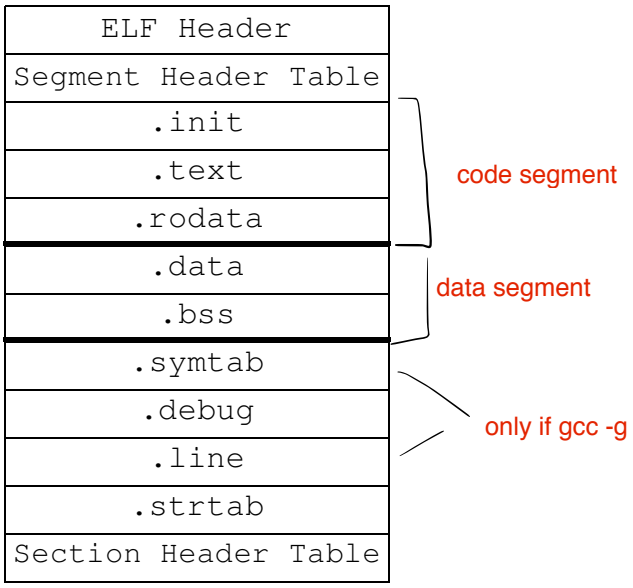
same as ROF with additions used by loader

ELF Header

+ entry point (address of program's first instruction)

+ Segment Header Table

info for each segments, loaded into memory
offset in file,
alignment,
page size (4kb)
size in file and in memory
run time permissions



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

with static linking all symbols REFs have been replaced with address

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

Loader

What? The *loader*

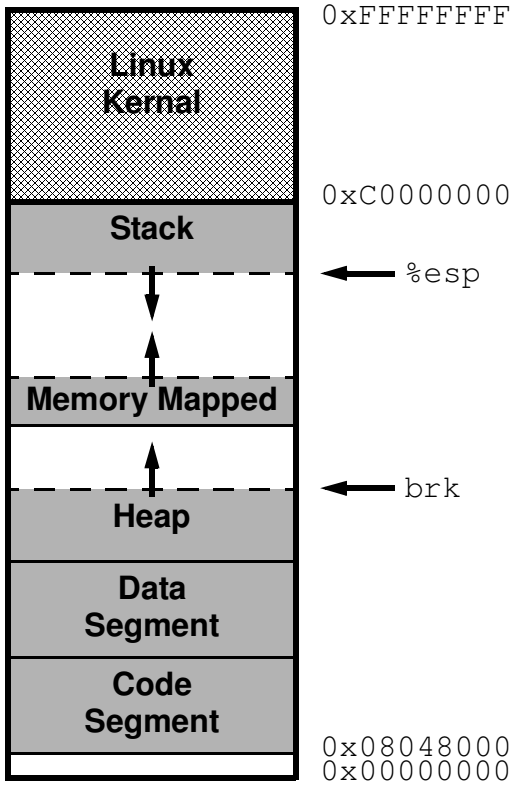
- is kernel code that starts a program executin.
- can be invoked by any linux program using exexve. —> system call

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 forks itself to create a new child process.
   fork()
2. child process invokes the loader by execve()
3. loader create new runtime memory image.
   a. delete current code, data, heap, and stack.
   b. creates new segments
   c. heap and stack initialize to 0
   d. EOF's code and data segments are mapped by loader into page size chunks
      But these are not copied into memory



4. loader JUMPs to _start
   causes a page fault which then results in first page of code segments being loaded from the file.

```
_start:
    call __libc_init_first
    call _init
    call atexit
    call main
    call _exit
```

initializations of c lib function stack with CIAs, poss. Heap.

.init SECTION

adds functions to that clean up resources used.

calling clean up functions and return to Operation system

main return statement, main}, exit()