GAS Tutorial - 4. Sections & Relocation

Young W. Lim

2016-03-01 Tue

Outline

Based on

"Using as", Dean Elsner, Jay Fenlason & friends

Section

- a section is a range of addresses, with no gaps
- the linker ld
  - reads many object files (initial address 0)
  - combines them to an executable
  - moves blocks of bytes (i.e, sections) of your program to their run-time addresses
- relocation: assigning run-time addresses to sections

Section Type

- an object file written by as has at least three sections
  - text section
  - data section
  - bss section
- these sections can be empty
- in an object file: the text section starts at address 0, the data section follows, and

finally the bss section.

- COFF or ELF output
  - named section (.section directives)

Relocation

- when the sections are relocated, the ld should know
  - which data changes
  - how to change that data
- whenever an address in the object file is referenced,
  - the beginning of this reference to an address?
  - the length (in bytes) of this reference?
  - which section does the address refer to?
  - (address) (start-address of section)?
  - "Program-Counter relative"?

Section Relative

every address is expressed as

- (section) + (offset into section)
- every expression has this section-relative nature
- {secname N } notation: “offset N into section sec-name.”

Absolute Section

- addresses in the absolute section remain unchanged
- address {absolute 0} is “relocated” to run-time address 0 by ld
- generally, linker never use overlapping addresses
- address in absolute sections must overlap

Undefined Section

{undefined U }

- any address whose section is unknown at assembly time
- U is to be filled
- to generate an undefined address
  - using an undefined symbol.
  - using a named common block

Linker

ld puts

- all partial programs' text sections
- in contiguous addresses in the linked program.

- all partial programs' data sections
- in contiguous addresses in the linked program.

- all partial programs' bss sections
- in contiguous addresses in the linked program.

Linker Section

Linker's view of section types

1. text section, data section, named section
2. bss section
3. absolute section
4. undefind section

Linker Section - text/data/named sections

- these sections hold your program
- as and ld treat them as separate but equal sections
- these sections are differentiated when the program is running
  - the text section : unalterable
    - often shared among processes
    - contains instructions, constants
  - the data section : alterable:
    - C variables

Linker Section - bss sections

- contains zeroed bytes when your program begins running
- used to hold uninitialized variables or common storage
- the length of each partial program's bss section is important
- there is no need to store explicit zero bytes in the object file
- because the program starts out containing zeroed bytes
- bss section was invented to eliminate those explicit zeros from object files.

Linker Section - absolute sections

- Address 0 of this section is always “relocated” to runtime address 0
- useful when refering to an address that ld must not change
- being “unrelocatable”: addresses do not change during relocation.

Linker Section - undefined sections

- This “section” is a catch-all for address references to objects not in the preceding sections.

Subsections

used to locate separate groups of data in named sections close to each other in the object file

- a section can be divided into numbered subsections
- subsection number ranging from 0 to 8192
- default subsection number 0
- bytes with the same subsection number are assembled together

Subsection Usage

- .text expression
- .data expression
- .section name , expression [COFF]
- .subsection expression [ELF]

- expression respresents subsection number
- expression should use absolute address

- .text (.text 0, equivalently)
- .data (.data 0, equivalently)

Subsection Examples (1)

For example, to store constants in the text section not interspersed

- '.text 0' before each section of code being output
- '.text 1' before each group of constants being output.

Subsection Examples (2)

- .text 0
- .ascii “1st text subsection - (1)”
- .text 1
- .ascii “2nd text subsection.”
- .data 0
- .ascii “1st data subsection,”
- .text 0
- .ascii “1st text subsection - (2)”

Location Counter

- each section has a location counter
- incremented by one for every byte assembled into that section
- subsection do not have its own location counter
- by using .align, a location counter is manipulated indirectly
- by using label, the value of a location counter can be captured
- the location counter of a section which are being assembled is said to be active

BSS Section

- used for local common variable storage
- allocate address space without data loading
- when the programming starts running,

all the contents of the bss section are zeroed

- .lcomm defines a local common symbol in the bss section
- .comm can be used to declare a common symbol
- .section name, “b” [COFF]
- .section name, “a” [ELF]
- .skip size, 0

TTTT