University of Arizona

CSc 340

Foundations of Computer Systems

Christian Collberg

February 13, 2001

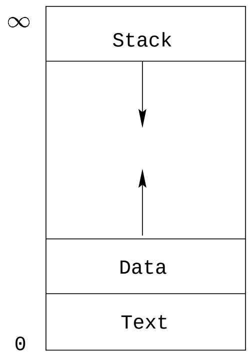
Static Memory Allocation

Memory Layout

- Static memory allocation refers to memory that is allocated to your program before it runs.
- Dynamic memory allocation refers to memory that your program allocates while it runs.
- Memory is allocated statically using the directives `.data`, `.word`, `.byte`, etc.

Segments

- When your program is running (under UNIX) the memory it is using looks like this:



Segments

- There are three segments:

  text The text segment contains the program's instructions. It is called text, instead of code, for historical reasons.

  data The data segment contains the program's static and dynamic memory. It is often called the heap.

  stack The stack segment contains the program's stack. It is used to implement subroutine calls and holds each subroutine invocation's automatic local variables and parameters.

Slide 9–4

Segments. . .

- The heap grows upwards towards high addresses, and the stack grows downwards towards low addresses. If they collide your program crashes.
- The above picture is very simplistic in that it ignores other programs that might be running in the computer, the operating system, shared libraries (DLLs), etc.
- The assembler, in addition to converting assembly instructions into machine code, decides the initial layout of the text and data segments.

Slide 9–5

Segments. . .

- The assembler ensures that,
  1. all instructions (denoted by `.text` directives) are assigned to the text segment;
  2. all statically-allocated data (denoted by `.data` directives) are assigned to the data segment;
  3. the data segment has two sub-regions: initialized data and uninitialized data. If you specify an initial value for a memory location it is assigned to the former, otherwise it goes into the latter (and has an initial value of 0).

Slide 9–6

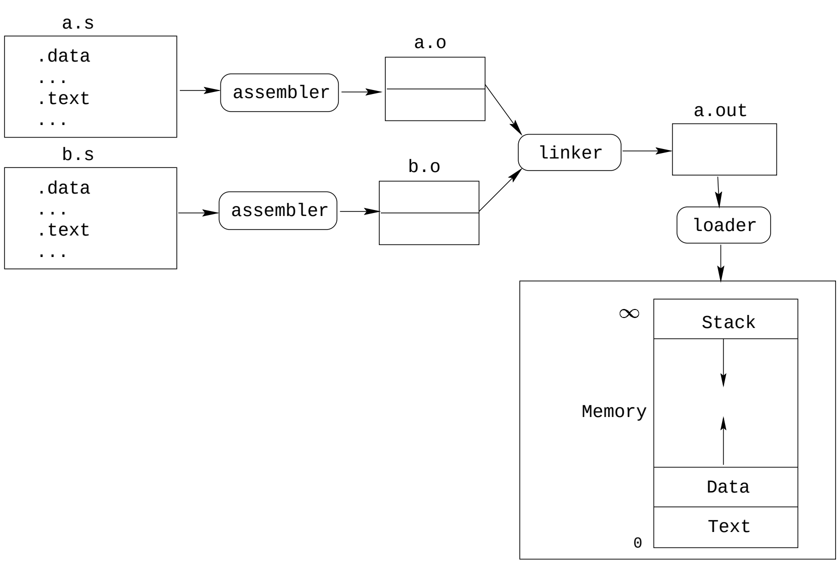
- To run a program P we need three other programs: an **assembler**, a **linker**, and a **loader**:

The assembler produces one object file for each assembly file, containing the starting addresses and sizes of the segments, the machine code for the text segment, and the values for the initialized data in the data segment. Object files usually end with `.o`.

The linker links several object files into a complete executable file.

The loader puts the program's segments into the computer's memory at the correct addresses and with the correct contents, and starts running the program.

Slide 9–7



Slide 9–8

```
        .data
foo:    .word 10
bar:    .space 4
baz:    .word 20
        .text
main:   lw    $s0, foo
        lw    $s1, bar
```

Symbol	Segment	Address	Comments
main	text	0x0	First instruction is at address 0
foo	data (i)	0x8	Follows text segment (2 insts.)
bar	data (u)	0x10	Follows initialized data
baz	data (i)	0xC	Second word in initialized data

Slide 9–9

Static Allocation

- Benefits of static allocation:
  - Easy to use and implement
- Problems with static allocation:
  - Must allocate memory before the program runs; no way to get more once it is running.

Slide 9–10

SPIM caveats

1. Text segment starts at address 0x400000 (not zero)
2. Top of stack is address 0x7FFFFFFF (not infinity)
3. Initialized and uninitialized data are not segregated in data segment.
4. Segments start at 0x1000 (4096) boundaries.
5. SPIM has an assembler, linker, and loader built-in, so you don't have to worry about all this when doing your programming assignments.

Slide 9–11

Readings and References

- Read Maccabe, section 5.1.
- SPIM can be found at `http://www.cs.wisc.edu/~larus/spim.html`.