WIKIPEDIA

Data segment

In computing, a **data segment** (often denoted **.data**) is a portion of an object file or the corresponding virtual address space of a program that contains initialized static variables, that is, global variables and static local variables. The size of this segment is determined by the size of the values in the program's source code, and does not change at run time.

The data segment is read-write, since the values of variables can be altered at run time. This is in contrast to the *read-only data segment* (*rodata segment* or *.rodata*), which contains static constants rather than variables; it also contrasts to the code segment, also known as the text segment, which is read-only on many architectures. Zero-initialized data, both variables and constants, is instead in the BSS segment.

Historically, to be able to support memory address spaces larger than the native size of the internal address register would allow, early CPUs implemented a system of segmentation whereby they would store a small set of indexes to use as offsets to certain areas. The Intel 8086 family of CPUs provided four segments: the code segment, the data segment, the stack segment and the extra segment. Each segment was placed at a specific location in memory by the software being executed and all instructions that operated on the data within those segments were performed relative to the start of that segment. This allowed a 16-bit address register, which would normally be able to access 64 KB of memory space, to access 1 MB of memory space.

This segmenting of the memory space into discrete blocks with specific tasks carried over into the programming languages of the day and the concept is still widely in use within modern programming languages.

Contents

Program memory

A computer program memory can be largely categorized into two sections: read-only and read-write. This distinction

grew from early systems holding their main program in read-only memory such as Mask ROM, PROM or EEPROM. As systems became more complex and programs were loaded from other media into RAM instead of executing from ROM the idea that some portions of the program's memory should not be modified was retained. These became the *.text* and *.rodata* segments of the program, and the remainder which could be written to divided into a number of other segments for specific tasks.

Text

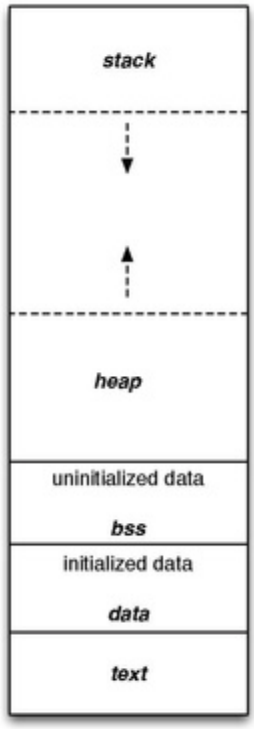
The **code segment**, also known as a **text segment** or simply as **text**, is where a portion of an object file or the corresponding section of the program's virtual address space that contains executable instructions is stored and is generally read-only and fixed size.

Data

The *.data* segment contains any global or static variables which have a pre-defined value and can be modified. That is any variables that are not defined within a function (and thus can be accessed from anywhere) or are defined in a function but are defined as *static* so they retain their address across subsequent calls. Examples, in C, include:

```
int val = 3;
char string[] = "Hello World";
```

The values for these variables are initially stored within the read-only memory (typically within *.text*) and are copied into the *.data* segment during the start-up routine of the program.



This shows the typical layout of a simple computer's program memory with the text, various data, and stack and heap sections.

BSS

The BSS segment, also known as *uninitialized data*, is usually adjacent to the data segment. The BSS segment contains all global variables and static variables that are initialized to zero or do not have explicit initialization in source code. For instance, a variable defined as `static int i;` would be contained in the BSS segment.

Heap

The heap area commonly begins at the end of the .bss and .data segments and grows to larger addresses from there. The heap area is managed by malloc, calloc, realloc, and free, which may use the brk and sbrk system calls to adjust its size (note that the use of brk/sbrk and a single "heap area" is not required to fulfill the contract of malloc/calloc /realloc/free; they may also be implemented using mmap/munmap to reserve/unreserve potentially non-contiguous regions of virtual memory into the process' virtual address space). The heap area is shared by all threads, shared libraries, and dynamically loaded modules in a process.

Stack

The stack area contains the program stack, a LIFO structure, typically located in the higher parts of memory. A "stack pointer" register tracks the top of the stack; it is adjusted each time a value is "pushed" onto the stack. The set of values pushed for one function call is termed a "stack frame". A stack frame consists at minimum of a return address. Automatic variables are also allocated on the stack.

The stack area traditionally adjoined the heap area and they grew towards each other; when the stack pointer met the heap pointer, free memory was exhausted. With large address spaces and virtual memory techniques they tend to be placed more freely, but they still typically grow in a converging direction. On the standard PC x86 architecture the stack grows toward address zero, meaning that more recent items, deeper in the call chain, are at numerically lower addresses and closer to the heap. On some other architectures it grows the opposite direction.

Interpreted languages

Some interpreted languages offer a similar facility to the data segment, notably Perl[1] and Ruby.[2] In these languages, including the line `__DATA__` (Perl) or `__END__` (Ruby, old Perl) marks the end of the code segment and the start of the data segment. Only the contents prior to this line are executed, and the contents of the source file after this line are available as a file object: `PACKAGE::DATA` in Perl (e.g., `main::DATA`) and `DATA` in Ruby. This can be considered a form of here document (a file literal).

See also

- Segmentation (memory)
- Segmentation fault
- Linker (computing)
- Code segment
- .bss
- Uninitialized variable
- Stack (abstract data type)

References

1. perldata: Special Literals (http://perldoc.perl.org/perldata.html#Special-Literals)
2. Ruby: Object: __END__ (http://ruby-doc.org/docs/keywords/1.9/Object.html#method-i-__END__)

- "BraveGNU.org" (http://www.bravegnu.org/gnu-eprog/c-startup.html).

External links

- mem_sequence.c - sequentially lists memory regions in a process (http://blog.ooz.ie/2008/09/0x03-notes-on-assembly-memory-from.html)

- Expert C Programming: Deep C Secrets, Peter van der Linden, Prentice Hall 1997, p. 119ff (http://www.comp.nus.edu.sg/~xujia/Expert.C.Programming.pdf)
