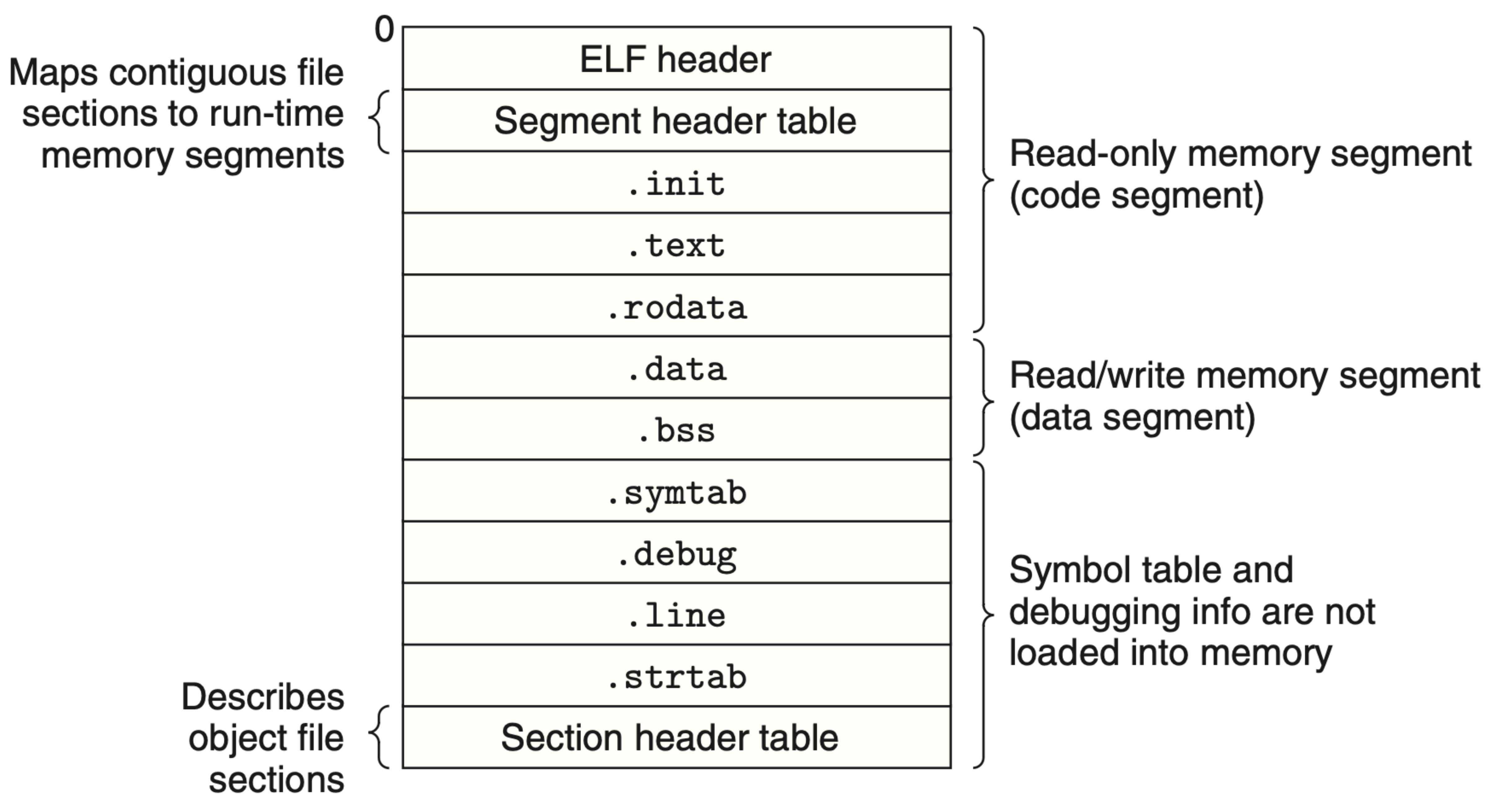
COMS 4995 Advanced Systems Programming

Introduction to ELF

ELF (Executable and Linkable Format) is the standard executable and object file format in Linux.

CSAPP Figure 7.13 depicts the layout of an executable file:


0
Maps contiguous file sections to run-time memory segments
ELF header
Segment header table
.init
.text
.rodata
.data
.bss
.symtab
.debug
.line
.strtab
Section header table
Describes object file sections
Read-only memory segment (code segment)
Read/write memory segment (data segment)
Symbol table and debugging info are not loaded into memory

ELF files can be broken down into segments and sections. Segments and sections may overlap (i.e., several sections may compose a single segment). This is because ELF offers two views of the executable files:

- Linker/Section view: As object files are linked together, the linker will have to merge multiple instances of a given section from different object files into one section in the executable.
- Loader/Segment view: Segments classify different regions of the executable or shared library for the purpose of loading into memory for execution.
    - Not all sections are to be loaded into memory for execution (e.g, debugging information)
    - A segment contains one more sections. The diagram above shows two loadable segments, one containing read-only data and one containing read/write data.

See also this other webpage with a colorful diagram.

The `readelf` command-line utility allows you to view everything about an ELF file. See `man elf` for the C programming interface for working with ELF files.

ELF Header

All ELF files begin with a header that contains metadata for the ELF file, including:

- Four "magic" bytes in the beginning of the file: `0x7f`, `'E'`, `'L'`, `'F'`
- Information about the system on which the ELF binary is meant to run
- Offsets into the file for the Program header (Phdr) table and the Section header (Shdr) table. The former lists the segments and the latter lists the sections in the ELF file.

Section Header Table

The section header table is an array of `Elf64_Shdr` structs. Each entry contains metadata about a section in the ELF file, including the offset into the ELF file at which the section is laid out and the section's name. The section's name isn't stored directly in `Elf64_Shdr`. Instead, the struct has a "name" field which is an index into a dedicated string table section. This is usually how ELF stores strings. The string table for the section names is stored in the `.shstrtab` section (not shown in the figure above). The ELF header stores the index of this section in the header table for ease of lookup.

Symbol Table

The ELF symbol table (stored at the `.symtab` section) is a list of all symbols referred to by the executable, namely functions and static variables. It also contains symbols that are referred to in the program by not actually defined by it, such as C library functions like `printf()`.

The symbol table is an array of `Elf64_Sym` structs, defined as follows:

```
typedef struct {
    uint32_t      st_name;
    unsigned char st_info;
    unsigned char st_other;
    uint16_t      st_shndx;
    Elf64_Addr    st_value;
    uint64_t      st_size;
} Elf64_Sym;
```

A few notes about some fields of interest:

- `st_name` is the index into the symbol table's string table, stored in the `.strtab` section.
- `st_info` is a packed byte containing the symbol's type and binding.
- `st_value` is the address of the symbol.
- `st_size` is the size of the symbol. For functions, `st_size` is the total byte size of all of the instructions in the function.

Last updated: 2024-03-26