NAME

a.out – Executable and Linking Format (ELF) files

SYNOPSIS

#include <elf.h>

DESCRIPTION

The file name **a.out** is the default output file name from the link editor, **ld**(1). The link editor will make an **a.out** executable if there were no errors in linking. The output file of the assembler, **as**(1), also follows the format of the **a.out** file although its default file name is different.

Programs that manipulate ELF files may use the library that **elf**(3E) describes. An overview of the file format follows. For more complete information, see the references given below.

<table>
<tr><th>Linking View</th><th>Execution View</th></tr>
<tr><td>ELF header</td><td>ELF header</td></tr>
<tr><td>Program header table
optional</td><td>Program header table</td></tr>
<tr><td>Section 1</td><td rowspan="2">Segment 1</td></tr>
<tr><td>. . .</td></tr>
<tr><td>Section n</td><td rowspan="2">Segment 2</td></tr>
<tr><td>. . .</td></tr>
<tr><td>. . .</td><td>. . .</td></tr>
<tr><td>Section header table</td><td>Section header table
optional</td></tr>
</table>

An ELF header resides at the beginning and holds a ‘‘road map’’ describing the file’s organization. Sections hold the bulk of object file information for the linking view: instructions, data, symbol table, relocation information, and so on. Segments hold the object file information for the program execution view. As shown, a segment may contain one or more sections.

A program header table, if present, tells the system how to create a process image. Files used to build a process image (execute a program) must have a program header table; relocatable files do not need one. A section header table contains information describing the file’s sections. Every section has an entry in the table; each entry gives information such as the section name, the section size, etc. Files used during linking must have a section header table; other object files may or may not have one.

Although the figure shows the program header table immediately after the ELF header, and the section header table following the sections, actual files may differ. Moreover, sections and segments have no specified order. Only the ELF header has a fixed position in the file.

When an **a.out** file is loaded into memory for execution, three logical segments are set up: the text segment, the data segment (initialized data followed by uninitialized, the latter actually being initialized to all 0’s), and a stack. The text segment is not writable by the program; if other processes are executing the same **a.out** file, the processes will share a single text segment.

The data segment starts at the next maximal page boundary past the last text address. If the system supports more than one page size, the ''maximal page'' is the largest supported size. When the process image is created, the part of the file holding the end of text and the beginning of data may appear twice. The duplicated chunk of text that appears at the beginning of data is never executed; it is duplicated so that the operating system may bring in pieces of the file in multiples of the actual page size without having to realign the beginning of the data section to a page boundary. Therefore, the first data address is the sum of the next maximal page boundary past the end of text plus the remainder of the last text address divided by the maximal page size. If the last text address is a multiple of the maximal page size, no duplication is necessary. The stack is automatically extended as required. The data segment is extended as requested by the **brk**(2) system call.

SEE ALSO

as(1), **cc**(1B), **ld**(1), **brk**(2), **elf**(3E)

ANSI C Programmer's Guide