程序 14-1 linux/include/a.out.h

```
1  #ifndef _A_OUT_H
2  #define _A_OUT_H
3
4  #define __GNU_EXEC_MACROS__
5
   // 第 6--108 行是该文件第 1 部分。定义目标文件执行结构以及相关操作的宏定义。
   // 目标文件头结构。参见程序后的详细说明。
   // ============================
   // unsigned long a_magic        // 执行文件魔数。使用 N_MAGIC 等宏访问。
   // unsigned a_text              // 代码长度，字节数。
   // unsigned a_data              // 数据长度，字节数。
   // unsigned a_bss               // 文件中的未初始化数据区长度，字节数。
   // unsigned a_syms              // 文件中的符号表长度，字节数。
   // unsigned a_entry             // 执行开始地址。
   // unsigned a_trsize            // 代码重定位信息长度，字节数。
   // unsigned a_drsize            // 数据重定位信息长度，字节数。
   // ----------------------------
6  struct exec {
7    unsigned long a_magic;        /* Use macros N_MAGIC, etc for access */
8    unsigned a_text;              /* length of text, in bytes */
9    unsigned a_data;              /* length of data, in bytes */
10   unsigned a_bss;               /* length of uninitialized data area for file, in bytes */
11   unsigned a_syms;              /* length of symbol table data in file, in bytes */
12   unsigned a_entry;             /* start address */
13   unsigned a_trsize;            /* length of relocation info for text, in bytes */
14   unsigned a_drsize;            /* length of relocation info for data, in bytes */
15 };
16
   // 用于取上述 exec 结构中的魔数。
17 #ifndef N_MAGIC
18 #define N_MAGIC(exec) ((exec).a_magic)
19 #endif
20
21 #ifndef OMAGIC
22 /* Code indicating object file or impure executable.  */
   /* 指明为目标文件或者不纯的可执行文件的代号 */
   // 历史上最早在 PDP-11 计算机上，魔数（幻数）是八进制数 0407（0x107）。它位于执行程序
   // 头结构的开始处。原本是 PDP-11 的一条跳转指令，表示跳转到随后 7 个字后的代码开始处。
   // 这样加载程序（loader）就可以在把执行文件放入内存后直接跳转到指令开始处运行。 现在
   // 已没有程序使用这种方法，但这个八进制数却作为识别文件类型的标志（魔数）保留了下来。
   // OMAGIC 可以认为是 Old Magic 的意思。
23 #define OMAGIC 0407
24 /* Code indicating pure executable.  */
   /* 指明为纯可执行文件的代号 */       // New Magic，1975 年以后开始使用。涉及虚存机制。
25 #define NMAGIC 0410                 // 0410 == 0x108
26 /* Code indicating demand-paged executable.  */
   /* 指明为需求分页处理的可执行文件 */ // 其头结构占用文件开始处 1K 空间。
27 #define ZMAGIC 0413                 // 0413 == 0x10b
28 #endif /* not OMAGIC */
29 // 另外还有一个 QMAGIC，是为了节约磁盘容量，把盘上执行文件的头结构与代码紧凑存放。
   // 下面宏用于判断魔数字段的正确性。如果魔数不能被识别，则返回真。
30 #ifndef N_BADMAG
```

```
31 #define N_BADMAG(x)                                     \
32  (N_MAGIC(x) != OMAGIC && N_MAGIC(x) != NMAGIC          \
33   && N_MAGIC(x) != ZMAGIC)
34 #endif
35
36 #define _N_BADMAG(x)                                    \
37  (N_MAGIC(x) != OMAGIC && N_MAGIC(x) != NMAGIC          \
38   && N_MAGIC(x) != ZMAGIC)
39
   // 目标文件头结构末端到 1024 字节之间的长度。
40 #define _N_HDROFF(x) (SEGMENT_SIZE - sizeof (struct exec))
41
   // 下面宏用于操作目标文件的内容，包括.o 模块文件和可执行文件。

   // 代码部分起始偏移值。
   // 如果文件是 ZMAGIC 类型的，即是执行文件，那么代码部分是从执行文件的 1024 字节偏移处
   // 开始；否则执行代码部分紧随执行头结构末端（32 字节）开始，即文件是模块文件（OMAGIC
   // 类型）。
42 #ifndef N_TXTOFF
43 #define N_TXTOFF(x) \
44  (N_MAGIC(x) == ZMAGIC ? _N_HDROFF((x)) + sizeof (struct exec) : sizeof (struct exec))
45 #endif
46
   // 数据部分起始偏移值。从代码部分末端开始。
47 #ifndef N_DATOFF
48 #define N_DATOFF(x) (N_TXTOFF(x) + (x).a_text)
49 #endif
50
   // 代码重定位信息偏移值。从数据部分末端开始。
51 #ifndef N_TRELOFF
52 #define N_TRELOFF(x) (N_DATOFF(x) + (x).a_data)
53 #endif
54
   // 数据重定位信息偏移值。从代码重定位信息末端开始。
55 #ifndef N_DRELOFF
56 #define N_DRELOFF(x) (N_TRELOFF(x) + (x).a_trsize)
57 #endif
58
   // 符号表偏移值。从上面数据段重定位表末端开始。
59 #ifndef N_SYMOFF
60 #define N_SYMOFF(x) (N_DRELOFF(x) + (x).a_drsize)
61 #endif
62
   // 字符串信息偏移值。在符号表之后。
63 #ifndef N_STROFF
64 #define N_STROFF(x) (N_SYMOFF(x) + (x).a_syms)
65 #endif
66
   // 下面对可执行文件被加载到内存（逻辑空间）中的位置情况进行操作。
67 /* Address of text segment in memory after it is loaded.  */
   /* 代码段加载后在内存中的地址 */
68 #ifndef N_TXTADDR
69 #define N_TXTADDR(x) 0                          // 可见，代码段从地址 0 开始执行。
```

```
70 #endif
71
72 /* Address of data segment in memory after it is loaded.
73    Note that it is up to you to define SEGMENT_SIZE
74    on machines not listed here.  */
   /* 数据段加载后在内存中的地址。
      注意，对于下面没有列出名称的机器，需要你自己来定义
      对应的 SEGMENT_SIZE */
75 #if defined(vax) || defined(hp300) || defined(pyr)
76 #define SEGMENT_SIZE PAGE_SIZE
77 #endif
78 #ifdef  hp300
79 #define PAGE_SIZE       4096
80 #endif
81 #ifdef  sony
82 #define SEGMENT_SIZE    0x2000
83 #endif  /* Sony.  */
84 #ifdef is68k
85 #define SEGMENT_SIZE 0x20000
86 #endif
87 #if defined(m68k) && defined(PORTAR)
88 #define PAGE_SIZE 0x400
89 #define SEGMENT_SIZE PAGE_SIZE
90 #endif
91
   // 这里，Linux 0.12 内核把内存页定义为 4KB，段大小定义为 1KB。因此没有使用上面的定义。
92 #define PAGE_SIZE 4096
93 #define SEGMENT_SIZE 1024
94
   // 以段为界的大小（进位方式）。
95 #define _N_SEGMENT_ROUND(x) (((x) + SEGMENT_SIZE - 1) & ~(SEGMENT_SIZE - 1))
96
   // 代码段尾地址。
97 #define _N_TXTENDADDR(x) (N_TXTADDR(x)+(x).a_text)
98
   // 数据段开始地址。
   // 如果文件是 OMAGIC 类型的，那么数据段就直接紧随代码段后面。否则的话数据段地址从代码
   // 段后面段边界开始（1KB 边界对齐）。例如 ZMAGIC 类型的文件。
99 #ifndef N_DATADDR
100 #define N_DATADDR(x) \
101     (N_MAGIC(x)==OMAGIC? (_N_TXTENDADDR(x)) \
102      : (_N_SEGMENT_ROUND (_N_TXTENDADDR(x))))
103 #endif
104
105 /* Address of bss segment in memory after it is loaded.  */
   /* bss 段加载到内存以后的地址 */
   // 未初始化数据段 bbs 位于数据段后面，紧跟数据段。
106 #ifndef N_BSSADDR
107 #define N_BSSADDR(x) (N_DATADDR(x) + (x).a_data)
108 #endif
109
   // 第 110—185 行是第 2 部分。对目标文件中的符号表项和相关操作宏进行定义和说明。
   // a.out 目标文件中符号表项结构（符号表记录结构）。参见程序后的详细说明。
```

```
110 #ifndef N_NLIST_DECLARED
111 struct nlist {
112   union {
113     char *n_name;
114     struct nlist *n_next;
115     long n_strx;
116   } n_un;
117   unsigned char n_type;        // 该字节分成 3 个字段，146--154 行是相应字段的屏蔽码。
118   char n_other;
119   short n_desc;
120   unsigned long n_value;
121 };
122 #endif
123
    // 下面定义 nlist 结构中 n_type 字段值的常量符号。
124 #ifndef N_UNDF
125 #define N_UNDF 0
126 #endif
127 #ifndef N_ABS
128 #define N_ABS 2
129 #endif
130 #ifndef N_TEXT
131 #define N_TEXT 4
132 #endif
133 #ifndef N_DATA
134 #define N_DATA 6
135 #endif
136 #ifndef N_BSS
137 #define N_BSS 8
138 #endif
139 #ifndef N_COMM
140 #define N_COMM 18
141 #endif
142 #ifndef N_FN
143 #define N_FN 15
144 #endif
145
    // 以下 3 个常量定义是 nlist 结构中 n_type 字段的屏蔽码（八进程表示）。
146 #ifndef N_EXT
147 #define N_EXT 1                  // 0x01（0b0000,0001）符号是否是外部的（全局的）。
148 #endif
149 #ifndef N_TYPE
150 #define N_TYPE 036               // 0x1e（0b0001,1110）符号的类型位。
151 #endif
152 #ifndef N_STAB                   // STAB -- 符号表类型（Symbol table types）。
153 #define N_STAB 0340              // 0xe0（0b1110,0000）这几个比特用于符号调试器。
154 #endif
155
156 /* The following type indicates the definition of a symbol as being
157    an indirect reference to another symbol.  The other symbol
158    appears as an undefined reference, immediately following this symbol.
159
160    Indirection is asymmetrical.  The other symbol's value will be used
```

```
161     to satisfy requests for the indirect symbol, but not vice versa.
162     If the other symbol does not have a definition, libraries will
163     be searched to find a definition.  */
    /* 下面的类型指明对一个符号的定义是作为对另一个符号的间接引用。紧接该
     * 符号的其他的符号呈现为未定义的引用。
     *
     * 这种间接引用是不对称的。另一个符号的值将被用于满足间接符号的要求，
     * 但反之则不然。如果另一个符号没有定义，则将搜索库来寻找一个定义 */
164 #define N_INDR 0xa
165
166 /* The following symbols refer to set elements.
167     All the N_SET[ATDB] symbols with the same name form one set.
168     Space is allocated for the set in the text section, and each set
169     element's value is stored into one word of the space.
170     The first word of the space is the length of the set (number of elements).
171
172     The address of the set is made into an N_SETV symbol
173     whose name is the same as the name of the set.
174     This symbol acts like a N_DATA global symbol
175     in that it can satisfy undefined external references.  */
    /* 下面的符号与集合元素有关。所有具有相同名称 N_SET[ATDB]的符号
       形成一个集合。在代码部分中已为集合分配了空间，并且每个集合元素
       的值存放在一个字（word）的空间中。空间的第一个字存有集合的长度（集合元素数目）。

       集合的地址被放入一个 N_SETV 符号中，它的名称与集合同名。
       在满足未定义的外部引用方面，该符号的行为象一个 N_DATA 全局符号。*/
176
177 /* These appear as input to LD, in a .o file.  */
    /* 以下这些符号在 .o 文件中是作为链接程序 LD 的输入。*/
178 #define N_SETA  0x14        /* Absolute set element symbol */  /* 绝对集合元素符号 */
179 #define N_SETT  0x16        /* Text set element symbol */      /* 代码集合元素符号 */
180 #define N_SETD  0x18        /* Data set element symbol */      /* 数据集合元素符号 */
181 #define N_SETB  0x1A        /* Bss set element symbol */       /* Bss 集合元素符号 */
182
183 /* This is output from LD.  */
    /* 下面是 LD 的输出。*/
184 #define N_SETV  0x1C        /* Pointer to set vector in data area.  */
                                /* 指向数据区中集合向量。*/
185
186 #ifndef N_RELOCATION_INFO_DECLARED
187
188 /* This structure describes a single relocation to be performed.
189     The text-relocation section of the file is a vector of these structures,
190     all of which apply to the text section.
191     Likewise, the data-relocation section applies to the data section.  */
    /* 下面结构描述单个重定位操作的执行。
       文件的代码重定位部分是这些结构的一个数组，所有这些适用于代码部分。
       类似地，数据重定位部分用于数据部分。*/
192
    // a.out 目标文件中代码和数据重定位信息结构。
193 struct relocation_info
194 {
195   /* Address (within segment) to be relocated.  */
```

```
    /* 段内需要重定位的地址。*/
196 int r_address;
197 /* The meaning of r_symbolnum depends on r_extern.  */
    /* r_symbolnum 的含义与 r_extern 有关。*/
198 unsigned int r_symbolnum:24;
199 /* Nonzero means value is a pc-relative offset
200    and it should be relocated for changes in its own address
201    as well as for changes in the symbol or section specified.  */
    /* 非零意味着值是一个 pc 相关的偏移值，因而在其自己地址空间
       以及符号或指定的节改变时，需要被重定位 */
202 unsigned int r_pcrel:1;
203 /* Length (as exponent of 2) of the field to be relocated.
204    Thus, a value of 2 indicates 1<<2 bytes.  */
    /* 需要被重定位的字段长度（是 2 的次方）。
       因此，若值是 2 则表示 1<<2 字节数。*/
205 unsigned int r_length:2;
206 /* 1 => relocate with value of symbol.
207         r_symbolnum is the index of the symbol
208         in file's the symbol table.
209    0 => relocate with the address of a segment.
210         r_symbolnum is N_TEXT, N_DATA, N_BSS or N_ABS
211         (the N_EXT bit may be set also, but signifies nothing).  */
    /* 1 => 以符号的值重定位。
            r_symbolnum 是文件符号表中符号的索引。
       0 => 以段的地址进行重定位。
            r_symbolnum 是 N_TEXT、N_DATA、N_BSS 或 N_ABS
            (N_EXT 比特位也可以被设置，但是毫无意义)。*/
212 unsigned int r_extern:1;
213 /* Four bits that aren't used, but when writing an object file
214    it is desirable to clear them.  */
    /* 没有使用的 4 个比特位，但是当进行写一个目标文件时
       最好将它们复位掉。*/
215 unsigned int r_pad:4;
216 };
217 #endif /* no N_RELOCATION_INFO_DECLARED.  */
218
219
220 #endif /* __A_OUT_GNU_H__ */
221
```