A.OUT(6) A.OUT(6)

NAME

a.out - object file format

SYNOPSIS

```
#include <a.out.h>
```

DESCRIPTION

An executable Plan 9 binary file has up to six sections: a header, the program text, the data, a symbol table, a PC/SP offset table (MC68020 only), and finally a PC/line number table. The header, given by a structure in <a.out.h>, contains 4-byte integers in big-endian order:

```
typedef struct Exec {
        long       magic;      /* magic number */
        long       text;       /* size of text segment */
        long       data;       /* size of initialized data */
        long       bss;        /* size of uninitialized data */
        long       syms;       /* size of symbol table */
        long       entry;      /* entry point */
        long       spsz;       /* size of pc/sp offset table */
        long       pcsz;       /* size of pc/line number table */
} Exec;
#define  _MAGIC(b)  ((((4*b)+0)*b)+7)
#define  A_MAGIC    _MAGIC(8)   /* 68020 */
#define  I_MAGIC    _MAGIC(11)  /* intel 386 */
#define  J_MAGIC    _MAGIC(12)  /* intel 960 */
#define  K_MAGIC    _MAGIC(13)  /* sparc */
#define  V_MAGIC    _MAGIC(16)  /* mips 3000 */
#define  X_MAGIC    _MAGIC(17)  /* att dsp 3210 */
#define  M_MAGIC    _MAGIC(18)  /* mips 4000 */
#define  D_MAGIC    _MAGIC(19)  /* amd 29000 */
#define  E_MAGIC    _MAGIC(20)  /* arm 7-something */
#define  Q_MAGIC    _MAGIC(21)  /* powerpc */
#define  N_MAGIC    _MAGIC(22)  /* mips 4000 LE */
#define  L_MAGIC    _MAGIC(23)  /* dec alpha */
```

Sizes are expressed in bytes. The size of the header is not included in any of the other sizes.

When a Plan 9 binary file is executed, a memory image of three segments is set up: the text segment, the data segment, and the stack. The text segment begins at a virtual address which is a multiple of the machine-dependent page size. The text segment consists of the header and the first text bytes of the binary file. The entry field gives the virtual address of the entry point of the program. The data segment starts at the first page-rounded virtual address after the text segment. It consists of the next data bytes of the binary file, followed by bss bytes initialized to

A.OUT(6) A.OUT(6)

zero. The stack occupies the highest possible locations in the core image, automatically growing downwards. The bss segment may be extended by brk(2).

The next syms (possibly zero) bytes of the file contain symbol table entries, each laid out as:

```
uchar value[4];
char  type;
char  name[n];   /* NUL-terminated */
```

The value is in big-endian order and the size of the name field is not pre-defined: it is a zero-terminated array of variable length.

The type field is one of the following characters with the high bit set:

T	text segment symbol
t	static text segment symbol
L	leaf function text segment symbol
l	static leaf function text segment symbol
D	data segment symbol
d	static data segment symbol
B	bss segment symbol
b	static bss segment symbol
a	automatic (local) variable symbol
p	function parameter symbol

A few others are described below. The symbols in the symbol table appear in the same order as the program components they describe.

The Plan 9 compilers implement a virtual stack frame pointer rather than dedicating a register; moreover, on the MC680X0 architectures there is a variable offset between the stack pointer and the frame pointer. Following the symbol table, MC680X0 executable files contain a spsz-byte table encoding the offset of the stack frame pointer as a function of program location; this section is not present for other architectures. The PC/SP table is encoded as a byte stream. By setting the PC to the base of the text segment and the offset to zero and interpreting the stream, the offset can be computed for any PC. A byte value of 0 is followed by four bytes that hold, in big-endian order, a constant to be added to the offset. A byte value of 1 to 64 is multiplied by four and added, without sign extension, to the offset. A byte value of 65 to 128 is reduced by 64, multiplied by four, and subtracted from the offset. A byte value of 129 to 255 is reduced by 129, multiplied by the quantum of instruction size (e.g. two on the MC680X0), and added to the current PC without changing the offset. After any of these

operations, the instruction quantum is added to the PC.

A similar table, occupying pcsz-bytes, is the next section in an executable; it is present for all architectures. The same algorithm may be run using this table to recover the absolute source line number from a given program location. The absolute line number (starting from zero) counts the newlines in the C-preprocessed source seen by the compiler. Three symbol types in the main symbol table facilitate conversion of the absolute number to source file and line number:

f	source file name components
z	source file name
Z	source file line offset

The f symbol associates an integer (the value field of the `symbol') with a unique file path name component (the name of the `symbol'). These path components are used by the z symbol to represent a file name: the first byte of the name field is always 0; the remaining bytes hold a zero-terminated array of 16-bit values (in big-endian order) that represent file name components from f symbols. These components, when separated by slashes, form a file name. The initial slash of a file name is recorded in the symbol table by an f symbol; when forming file names from z symbols an initial slash is not to be assumed. The z symbols are clustered, one set for each object file in the program, before

any text symbols from that object file. The set of z symbols for an object file form a history stack of the included source files from which the object file was compiled. The value associated with each z symbol is the absolute line number at which that file was included in the source; if the name associated with the z symbol is null, the symbol represents the end of an included file, that is, a pop of the history stack. If the value of the z symbol is 1 (one), it represents the start of a new history stack. To recover the source file and line number for a program location, find the text symbol containing the location and then the first history stack preceding the text symbol in the symbol table. Next, interpret the PC/line offset table to discover the absolute line number for the program location. Using the line number, scan the history stack to find the set of source files open at that location. The line number within the file can be found using the line numbers in the history stack. The Z symbols correspond to #line directives in the source; they specify an adjustment to the line number to be printed by the above algorithm. The offset is associated with the first previous z symbol in the symbol table.

A.OUT(6) A.OUT(6)

SEE ALSO

db(1), acid(1), 2a(1), 2l(1), nm(1), strip(1), mach(2), symbol(2)

BUGS

There is no type information in the symbol table; however, the -a flags on the compilers will produce symbols for acid(1).