NAME a.out -- assembler and link editor output

SYNOPSIS

DESCRIPTION

a.out is the output file of the assembler as and the link editor ld. In both cases, a.out is executable provided there were no errors and no unresolved external references.

This file has four sections: a header, the program text, a symbol table, and relocation bits. The last two may be empty if the program was loaded with the —s option of ld or if the symbols and relocation have been removed by strip.

The header always contains 6 words:

1 a "br .+14" instruction (205(8))
2 The size of the program text
3 The size of the symbol table
4 The size of the relocation bits area
5 The size of a data area
6 A zero word (unused at present)

The sizes of the program, symbol table, and relocation area are in bytes but are always even. The branch instruction serves both to identify the file and to jump to the text entry point. The program text size includes the 6—word header.

The data area is used when the file is executed; the exec system call sets the program break to the sum of the text size and this data size. The data area is generated by the assembler when the location counter "." lies beyond the last assembled data, for example when the program ends with one or more constructions of the form .=.+n ; it is preserved by the loader for the last program in a load. (Routines other than the last have the appropriate number of 0 words inserted, since there is no other provision for zero—suppression in an a.out file.)

The symbol table consists of 6—word entries. The first four contain the ASCII name of the symbol, null—padded. (In fact, the assembler generates symbols of at most 7 bytes.) The next word is a flag indicating the type of symbol. The following values are possible:

00 undefined symbol
01 absolute symbol
02 register symbol
03 relocatable symbol
40 undefined global symbol
41 absolute global symbol

43 relocatable global symbol

An undefined global corresponds to a GMAP "symref" and an absolute or relocatable global to a "symdef" or absolute or relocatable value respectively. Values other than those given above may occur if the user has defined some of his own instructions.

The last word of a symbol table entry contains the value of the symbol. Its contents are not specified if the symbol is undefined.

If a.out contains no unresolved global references, header and text portions are exactly as they will appear in core when the file is executed. If the value of a word in the text portion involves a reference to an undefined global, the word is replaced by the offset in the symbol table of the appropriate symbol. (That is, possible offsets are 0, 12(10), 24(10), ...) Such a word will have appropriate relocation bits.

The relocation bits portion uses a variable–length encoding. There is a string of bits for each word in the text portion. The scheme has at least two bits for. each word, plus possibly two more to extend the codes available; in either case the bits may be followed by a 16–bit string to represent an offset to an external symbol. The bits are packed together without regard to word boundaries. The last word is filled out with 0's on the right.

The possible relocation bit configurations are:

00
word is absolute

01
word is relocatable

10
word is a relative reference to an undefined global symbol with no offset. Currently, the word contains the offset in the symbol table of the symbol. When the symbol becomes defined, say with x, this location will contain x–.–2, where "." is the location of the word.

1100xxxxxxxxxxxxxxxx
word is a relative reference to an external symbol with an offset. It is the same as the previous relocation type, except that the 16–bit offset is added in when the symbol

becomes defined.

1101
word is a reference to an undefined external symbol with no offset. At present the word contains the symbol table offset of the symbol. When the symbol becomes defined, the word will contain the value of the symbol.

1110xxxxxxxxxxxxxxxx
word is a reference to an undefined external symbol with an offset. At present, the word contains the symbol table offset of the symbol. When the symbol becomes defined, the word will contain the value of the symbol plus the given 16—bit offset.

FILES

SEE ALSO as ld, strip, nm, un

DIAGNOSTICS

BUGS Soon, there will be a new type of symbol: the data area symbol. In the text, it will appear as an ordinary external reference. However, it need not be defined; this will be done by the loader. Watch this space for more details.

OWNER dmr

NAME archive (library) file format

SYNOPSIS

DESCRIPTION The archive command ar is used to combine several files into one. Its use has three benefits: when files are combined, the file space consumed by the breakage at the end of each file (256 bytes on the average) is saved; directories are smaller and less confusing; archive files of object programs may be searched as libraries by the loader ld.

A file produced by ar has a "magic number" at the start, followed by the constituent files, each preceded by a file header. The magic number is —147(10), or 177555(8) (it was chosen to be unlikely to occur anywhere else). The header of each file is 16 bytes long:

0—7
file name, null padded on the right

8—Il
Modification time of the file

12
User ID of file owner

13
file mode

14—15
file size

If the file is an odd number of bytes long, it is padded with a null byte, but the size in the header is correct.

Notice there is no provision for empty areas in an archive file.

FILES

SEE ALSO

DIAGNOSTICS

BUGS

OWNER ken, dmr

NAME binary punched paper tape format

SYNOPSIS

DESCRIPTION Binary paper tape. is used to pass and store arbitrary information on paper tape. The format chosen has the following features: a) no format of the data is assumed. b) check summing c) zero suppress ion

The format is as follows:

Between records, NULL characters are ignored. The beginning of the tape is considered between records, thus the leader is ignored.

The first non—null character specifies the type and size of the record. If the character is positive (1 to 177), the record is a data record consisting of that many characters. All but the last of these characters are data, the last being a checksum. The checksum is calculated such that the sum of the entire record is zero mod 256.

If the first character is negative (200—376) the record is a zero suppression record. It is identical to minus that number of zeros of data. One character of checksum follows this negative character. It is the positive of the negative character.

The special case of a record looking like a single zero character suppressed (377;1) causes no data transfer, but is an end—of—file indication.

FILES

SEE ALSO lbppt, dbppt

DIAGNOSTICS

BUGS

OWNER ken, dmr

NAME format of core image

SYNOPSIS

DESCRIPTION

Three conditions cause UNIX to write out the core image of an executing program: the program generates an unexpected trap (by a bus error or illegal instruction); the user sends a quit signal (which has not been turned off by the program); a trap is simulated by the floating point simulator. The core image is called "core" and is written in the current working directory (provided it can be; normal access controls apply). It is exactly 8192+64 bytes long. The first 8192 represent the actual contents of memory at the time of the fault; the last 64 are the contents of the system's per—user data area for this process. Only the first word of this area will be described.

When any trap which is not an I/O interrupt occurs, all the useful registers are stored on the stack. After all the registers have been stored, the contents of are placed in the first cell of the user area; this cell is called <u>u.sp</u>. Therefore, within the core image proper, there is an area which contains the following registers in the following order (increasing addresses):

```
      (u.sp)—>sc
mq
ac
r5
r4
r3
r2
ri
r0
pc (at time of fault)
processor status (at time of fault)
```

The last two are stored by the hardware. It follows that the contents of at the time of the fault were (u.sp) plus 22(10).

The t—bit (trap bit) in the stored status will be on when a quit caused the generation of the core image, since this bit is used in the implementation of quits.

FILES

SEE ALSO

DIAGNOSTICS

BUGS

OWNER ken, dmr

NAME format of directories

SYNOPSIS

DESCRIPTION A directory behaves exactly like an ordinary file, save that no user may write into a directory. The fact that a file is a directory is indicated by a bit in the flag word of its i—node entry.

Directory entries are 10 bytes long. The first word is the i—node of the file represented by the entry, if non—zero; if zero, the entry is empty.

Bytes 2—9 represent the (8—character) file name, null padded on the right. These bytes are not necessarily cleared for empty slots.

By convention, the first two entries in each directory are for "." and "..". The first is an entry for the directory itself. The second is for the parent directory. The meaning of ".." is modified for the root directory of the master file system and for the root directories of removable file systems. In the first case, there is no parent, and in the second, the system does not permit off—device references without a <u>mount</u> system call. Therefore in both cases ".." has the same meaning as ".".

FILES

SEE ALSO file system format

DIAGNOSTICS

BUGS

OWNER ken, dmr

NAME format of file system

SYNOPSIS

DESCRIPTION Every file system storage volume (e.g. RF disk, RK disk, DECtape reel) has a common format for certain vital information.

Every such volume is divided into a certain number of 256 word (512 byte) blocks. Blocks 0 and 1 are collectively known as the super—block for the device; they define its extent and contain an i—node map and a free—storage map. The first word contains the number of bytes in the free—storage map; it is always even. It is followed by the map. There is one bit for each block on the device; the bit is 1 if the block is free. Thus if the number of free—map bytes is n, the blocks on the device are numbered 0 through 8n—1. The free—map count is followed by the free map itself. The bit for block k of the device is in byte k/8 of the map; it is offset k(mod 8) bits from the right. Notice that bits exist for the superblock and the i—list, even though they are never allocated or freed.

After the free map is a word containing the byte count for the i—node map. It too is always even. I—numbers below 41(10) are reserved for special files , and are never allocated; the first bit in the i—node free map refers to i—number 41. Therefore the byte number in the i—node map for i—node i is (i—41)/8. It is offset (i—41) (mod 8) bits from the right; unlike the free map, a "0" bit indicates an available i—node.

I—numbers begin at 1, and the storage for i—nodes begins at block 2. Also, i—nodes are 32 bytes long, so 16 of them fit into a block. Therefore, i—node i is located in block (i+31)/16 of the file system, and begins 32*(i+31)(mod 16)) bytes from its start.

There is always one file system which is always mounted; in standard UNIX it resides on the RF disk. This device is also used for swapping. The swap areas are at the high addresses on the device. It would be convenient if these addresses did not appear in the free list, but in fact this is not so. Therefore a certain number of blocks at the top of the device appear in the free map, are not marked free, yet do not appear within any file. These are the blocks that show up missing in a check of the RE’ disk.

Again on the primary file system device, there are several pieces of information following that previously discussed. They contain basically the information typed by the tm command; namely, the times spent since a cold boot in various categories, and a count of I/O errors. In particular, there are two words with the calendar time (measured since 00:00 Jan 1, 1971); two words with the time spent executing in the system; two words with the time spent waiting for I/O on the RF and RK disks; two words with the time spent executing in a user's core; one byte with the count of errors on the RF disk; and one byte with the count of errors on the RK disk. All the times are measured in sixtieths of a second.

I—node 41(10) is reserved for the root directory of the file system. No i—numbers other than this one and those from I to 40 (which represent special files) have a built—in meaning. Each i—node represents one file. The format of an i—node is as follows, where the left column represents the offset from the beginning of the i—node:

0—1	flags (see below)
2	number of links
3	user ID of owner
4—5	size in bytes
6—7	first indirect block or contents block
...	
20—21	eighth indirect block or contents block
22—25	creation time
26—29	modification time
30—31	unused

The flags are as follows:

100000	i—node is allocated
040000	directory
020000	file has been modified (always on)
010000	large file
000040	set user ID on execution
000020	executable
000010	read, owner
000004	write, owner
000002	read, non—owner
000001	write, non—owner

The allocated bit (flag 100000) is believed even if the i-node map says the i—node is free; thus corruption of the map may cause i—nodes to become unallocatable, but will not cause active nodes to be reused.

Byte number n of a file is accessed as follows: n is divided by 512 to find its logical block number (say b) in the file. If the file is small (flag 010000 is 0), then b must be less than 8, and the physical block number corresponding to b is the bth entry in the address portion of the i—node.

If the file is large, b is divided by 256 to yield a number which must be less than 8 (or the file is too large for UNIX to handle). The corresponding slot in the i—node address portion gives the physical block number of an indirect block. The residue mod 256 of b is multiplied by two (to give a byte offset in the indirect block) and the word found there is the physical address of the block corresponding to b.

If block b in a file exists, it is not necessary that all blocks less than b exist. A zero block number either in the address words of the i—node or in an indirect block indicates that the corresponding block has never been allocated. Such a missing block reads as if it contained all zero words.

FILES

SEE ALSO format of directories

DIAGNOSTICS

BUGS Two blocks are not enough to handle the i— and free—storage maps for an RP02 disk pack, which contains around 10 million words.

OWNER

NAME passwd -- password file

SYNOPSIS

DESCRIPTION _passwd_ contains for each user the following information:

name (login name)
password
numerical user ID
default working directory
program to use as Shell

This is an ASCII file. Each field within each a user's entry is separated from the next by a colon. Each user is separated from the next by a new—line. If the password field is null, no password is demanded; if the Shell field is null, the Shell itself is used.

This file, naturally, is inaccessible to anyone but the super—user.

This file resides in directory /etc.

FILES

SEE ALSO /etc/init

DIAGNOSTICS

BUGS

OWNER super—user

NAME /etc/uids -- map user names to user IDs SYNOPSIS

DESCRIPTION This file allows programs to map user names into user numbers and vice versa. Anyone can read it. It resides in directory /etc, and should be updated along with the password file when a user is added or deleted.

The format is an ASCII name, followed by a colon, followed by a decimal ASCII user ID number.

FILES

SEE ALSO

DIAGNOST ICS

BUGS

OWNER dmr, ken

NAME /tmp/utmp -- user information

SYNOPSIS

DESCRIPTION This file allows one to discover information about who is currently using UNIX. The file, is binary; each entry is 16(10) bytes long. The first eight bytes contain a user's login name or are null if the table slot is unused. The low order byte of the next word contains the last character of a typewriter name (currently. '0' to '5' for /dev/tty0 to /dev/tty5). The next two words contain the the user's login time. The last word is unused.

This file resides in directory /tmp.

FILES

SEE ALSO /etc/init, which maintains the file.

DIAGNOSTICS

BUGS

OWNER ken, dmr