Next: Access Permission, Previous: File Owner, Up: File Attributes [Contents][Index]

14.9.5 The Mode Bits for Access Permission

The *file mode*, stored in the `st_mode` field of the file attributes, contains two kinds of information: the file type code, and the access permission bits. This section discusses only the access permission bits, which control who can read or write the file. See Testing File Type, for information about the file type code.

All of the symbols listed in this section are defined in the header file `sys/stat.h`.

These symbolic constants are defined for the file mode bits that control access permission for the file:

`S_IRUSR`
`S_IREAD`

> Read permission bit for the owner of the file. On many systems this bit is 0400. `S_IREAD` is an obsolete synonym provided for BSD compatibility.

`S_IWUSR`
`S_IWRITE`

> Write permission bit for the owner of the file. Usually 0200. `S_IWRITE` is an obsolete synonym provided for BSD compatibility.

`S_IXUSR`
`S_IEXEC`

> Execute (for ordinary files) or search (for directories) permission bit for the owner of the file. Usually 0100. `S_IEXEC` is an obsolete synonym provided for BSD compatibility.

`S_IRWXU`

> This is equivalent to '`(S_IRUSR | S_IWUSR | S_IXUSR)`'.

`S_IRGRP`

> Read permission bit for the group owner of the file. Usually 040.

`S_IWGRP`

> Write permission bit for the group owner of the file. Usually 020.

`S_IXGRP`

> Execute or search permission bit for the group owner of the file. Usually 010.

`S_IRWXG`

> This is equivalent to '`(S_IRGRP | S_IWGRP | S_IXGRP)`'.

`S_IROTH`

Read permission bit for other users. Usually 04.

`S_IWOTH`

Write permission bit for other users. Usually 02.

`S_IXOTH`

Execute or search permission bit for other users. Usually 01.

`S_IRWXO`

This is equivalent to '`(S_IROTH | S_IWOTH | S_IXOTH)`'.

`S_ISUID`

This is the set-user-ID on execute bit, usually 04000. See How Change Persona.

`S_ISGID`

This is the set-group-ID on execute bit, usually 02000. See How Change Persona.

`S_ISVTX`

This is the *sticky* bit, usually 01000.

For a directory it gives permission to delete a file in that directory only if you own that file. Ordinarily, a user can either delete all the files in a directory or cannot delete any of them (based on whether the user has write permission for the directory). The same restriction applies—you must have both write permission for the directory and own the file you want to delete. The one exception is that the owner of the directory can delete any file in the directory, no matter who owns it (provided the owner has given himself write permission for the directory). This is commonly used for the `/tmp` directory, where anyone may create files but not delete files created by other users.

Originally the sticky bit on an executable file modified the swapping policies of the system. Normally, when a program terminated, its pages in core were immediately freed and reused. If the sticky bit was set on the executable file, the system kept the pages in core for a while as if the program were still running. This was advantageous for a program likely to be run many times in succession. This usage is obsolete in modern systems. When a program terminates, its pages always remain in core as long as there is no shortage of memory in the system. When the program is next run, its pages will still be in core if no shortage arose since the last run.

On some modern systems where the sticky bit has no useful meaning for an executable file, you cannot set the bit at all for a non-directory. If you try, `chmod` fails with `EFTYPE`; see Setting Permissions.

Some systems (particularly SunOS) have yet another use for the sticky bit. If the sticky bit is set on a file that is *not* executable, it means the opposite: never cache the pages of this file at all. The main use of this is for the files on an NFS server machine which are used as the swap area of diskless client machines. The idea is that the pages of the file will be cached in the client's memory, so it is a waste of the server's memory to cache them a second time. With this usage the sticky bit also implies that the filesystem may fail to record the file's modification time onto disk reliably (the idea being that no-one cares for a swap file).

This bit is only available on BSD systems (and those derived from them). Therefore one has to use the `_GNU_SOURCE` feature select macro, or not define any feature test macros, to get the definition (see Feature Test Macros).

The actual bit values of the symbols are listed in the table above so you can decode file mode values when debugging your programs. These bit values are correct for most systems, but they are not guaranteed.

Warning: Writing explicit numbers for file permissions is bad practice. Not only is it not portable, it also requires everyone who reads your program to remember what the bits mean. To make your program clean use the symbolic names.

Next: Access Permission, Previous: File Owner, Up: File Attributes [Contents][Index]