IEEE Standards Interpretation for IEEE Std 1003.1™-1990 IEEE Standard for Information Technology--Portable Operating System Interfaces (POSIX®)

Interpretation Request #71
Topic: fcntl **Relevant Sections:** 6.5.2

We hereby request an official binding interpretation concerning the following items pertaining the POSIX.1-1990 standard. Section 6.5.2, page 122, lines 341-345 Table 6-5 page 123, lines 381-388 F_GETFL page 124, lines 389-295 F_SETFL 6.5.2.2 fcntl() Description F_GETFL and F_SETFL talk about setting and getting the flags associated with O_APPEND and O_NONBLOCK, but for many file types no additional semantics for these flags are otherwise required to be associated with the file types anywhere else in the standard.

The semantics of O_APPEND appear to only really apply to the write() function. Under write() it says “the file offset shall be set”, but just before that it describes files “not capable of seeking” “shall start from the current position”. The implication appears to be O_APPEND can only have effect on seek capable files, since on files not capable of seeking the file offset can’t be set. POSIX.1 defines regular files as randomly accessible sequence of bytes, but it does not specifically say that lseek() is the method of randomly accessing those bytes, nor explicitly that a regular file is “capable of seeking”. However, the common inference is that regular files are capable of seeking by being defined as randomly accessible and thus O_APPEND must apply to them. The standard only appears to require that regular files support O_APPEND.

If an implementation defines the other file types as not being capable of seeking and thus O_APPEND has no effect, is it permissible for the implementation to ignore the fcntl(fd, F_SETFL, O_APPEND) for such files and thus have fcntl(fd, F_GETFL) never return the O_APPEND flag since it has no meaning? Similary, the standard is very explicit about the meaning of O_NONBLOCK for FIFO special files and allows the support of non-blocking opens, reads and writes for other file types.

If an implementation defines the other file types as not supporting nonblocking opens, reads or writes, is it permissible for the implementation to ignore the open(path, O_NONBLOCK) and fcntl(fd, F_SETFL, O_NONBLOCK) for such files and thus have fcntl(fd, F_GETFL) never return the O_NONBLOCK flag since it has no meaning? In particular, could the following be conformant behavior: regular_fd = open(regular_path, O_RDONLY | O_APPEND | O_NONBLOCK). fcntl(regular_fd, F_GETFL) & ~O_ACCMODE yields O_APPEND. char_fd = open(char_path, O_RDONLY | O_APPEND | O_NONBLOCK). fcntl(char_fd, F_GETFL) & ~O_ACCMODE yields O_NONBLOCK. block_fd = open(block_path, O_RDONLY | O_APPEND | O_NONBLOCK). fcntl(block_fd, F_GETFL) & ~O_ACCMODE yields 0. fifo_fd = open(fifo_path, O_RDONLY | O_APPEND | O_NONBLOCK). fcntl(fifo_fd, F_GETFL) & ~O_ACCMODE yields O_NONBLOCK. dir_fd = open(dir_path, O_RDONLY | O_APPEND | O_NONBLOCK). fcntl(dir_fd, F_GETFL) & ~O_ACCMODE yields 0. fcntl(regular_fd, F_SETFL, O_APPEND | O_NONBLOCK) returns 0. fcntl(regular_fd, F_GETFL) & ~O_ACCMODE yields O_APPEND. fcntl(char_fd, F_SETFL, O_APPEND | O_NONBLOCK) returns 0. fcntl(char_fd, F_GETFL) & ~O_ACCMODE yields O_NONBLOCK. fcntl(block_fd, F_SETFL, O_APPEND | O_NONBLOCK) returns 0. fcntl(block_fd, F_GETFL) & ~O_ACCMODE yields 0. fcntl(fifo_fd, F_SETFL, O_APPEND | O_NONBLOCK) returns 0. fcntl(fifo_fd, F_GETFL) & ~O_ACCMODE yields O_NONBLOCK. fcntl(dir_fd, F_SETFL, O_APPEND | O_NONBLOCK) returns 0. fcntl(dir_fd, F_GETFL) & ~O_ACCMODE yields 0.

Interpretation Response
The standard clearly states the requirements for the fcntl() function , and such an implementation as described in the interpretation request is not conforming.

Rationale for Interpretation
The standard has no provision for failing to set or clear the file status flags via fcntl based on the type of the file.