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

Interpretation Request #37
Topic: off_t error messages **Relevant Sections:** not specified **Classification:** No change

off_t has a finite upper bound. No error conditions are specifically identified for functions that attempt to exceed the inherent limit of off_t. Take for example lseek(fildes, 2**31-2,SEEK_SET);write(filedes,'abcd',4). Do any characters get written? Page 119, lines 204-205 imply 2 bytes would get written and subsequent call would get EFBIG. When more than one binding is supported, is EFBIG set to a size that all binds on the implementation can cope with? What is returned from lseek() and fcntl() when the resulting offset exceeds the size of off_t? I assume, EINVAL.

What must be documented in the Conformance Document when underlying file file systems and other bindings permit different limits than the 'C' binds? (Paul Wanish IBM)

Interpretation Response
There is an error for lseek() that applies to this situation:

[EINVAL] The whence argument is not a proper value, or the resulting file offset would be invalid.

The result of attempting a write() that would cause the file offset to exceed the maximum value that can be stored in an object of type off_t is unspecified. Although IEEE Std 1003.1-1990 states (page 119, lines 195-196) that:

Before successful return from write(), the file offset shall be incremented by the number of bytes actually written.

There is no error specified for the case where advancing the file pointer would produce an offset with no well-defined value. Since write() does not return this offset and need not examine it in this case, there is no requirement that an error condition be detected. This also applies to the interfaces from the C Standard that can extend the size of a file (fwrite(), fprintf(), etc.).

The description of the [EINVAL] error condition for fcntl() with the F_GETLK, F_SETLK, or F_SETLKW flag refers only to invalid data in the structure passed to fcntl(). The only way to set a lock on the portion of a file beyond the size that can be represented in type off_t is to set l_len to 0 to lock to the end of the file, and that is the only way that information that refers to that portion of the file can be returned by fcntl(). Issues related to harmonizing semantics with standards other than the C Standard are beyond the scope of IEEE Std 1003.1-1990. There is no requirement in IEEE Std 1003.1 that a mismatch in the ability to handle file sizes between POSIX.1 and the C Standard be documented. Note that the C Standard provides interfaces to be used in manipulating the file offsets for very large files (fgetpos(), fsetpos()).

Rationale for Interpretation

POSIX.1 does not specify a specific relationship among the maximum file size, {SSIZE_MAX}, the maximum value that can be stored in an object of type off_t, and the storage capacity of a particular medium or filesystem. Page 119, lines 204-205 refer to the case where there is no more room for data, which is not necessarily the same as the case where a write would cause the offset of the file pointer to exceed the maximum value that can be stored in an object of type off_t.

An application that needs to use file offsets that are larger than can be represented in type off_t should, if possible, use the fgetpos() and fsetpos() interfaces from the C Standard rather than using lseek().

It is suggested that a future revision of IEEE Std 1003.1 specify the behavior of fcntl() when used on files whose sizes cannot be represented in variables of type off_t. This condition can arise when file systems are mounted from a remote POSIX.1 system on which off_t is a larger type than on the local system.