NAME

open - open a file

SYNOPSIS

```
[OH] #include <sys/stat.h>
```

```
#include <fcntl.h>
```

```
fh_t *openg(const char *path, int oflag, fh_t *handle, ... );
```

DESCRIPTION

The *openg*() function shall establish the connection between a file and a file handle. It shall return a file handle that refers to a file. The file handle is used by the *sutoc()* function to refer to that file and complete the task of acquiring an open file descriptor. The *path* argument points to a pathname naming the file.

The file status flags and file access modes of the file handle shall be set according to the value of *oflag*.

Values for *oflag* are constructed by a bitwise-inclusive OR of flags from the following list, defined in *<fcntl.h>*. Applications shall specify exactly one of the first three values (file access modes) below in the value of *oflag*:

O_RDONLY
: Open for reading only.

O_WRONLY
: Open for writing only.

O_RDWR
: Open for reading and writing. The result is undefined if this flag is applied to a FIFO.

Any combination of the following may be used:

O_APPEND
: If set, the file offset shall be set to the end of the file prior to each write.

O_CREAT

If the file exists, this flag has no effect except as noted under O_EXCL below. Otherwise, the file shall be created; the user ID of the file shall be set to the effective user ID of the process; the group ID of the file shall be set to the group ID of the file's parent directory or to the effective group ID of the process; and the access permission bits (see *<sys/stat.h>*) of the file mode shall be set to the value of the third argument taken as type **mode_t** modified as follows: a bitwise AND is performed on the file-mode bits and the corresponding bits in the complement of the process' file mode creation mask. Thus, all bits in the file mode whose corresponding bit in the file mode creation mask is set are cleared. When bits other than the file permission bits are set, the effect is unspecified. The third argument does not affect whether the file is open for reading, writing, or for both. Implementations shall provide a way to initialize the file's group ID to the group ID of the parent directory. Implementations may, but need not, provide an implementation-defined way to initialize the file's group ID to the effective group ID of the calling process.

O_DSYNC

[SIO] Write I/O operations on the final file descriptor shall complete as defined by synchronized I/O data integrity completion.

O_EXCL

If O_CREAT and O_EXCL are set, *openg*() shall fail if the file exists. The check for the existence of the file and the creation of the file if it does not exist shall be atomic with respect to other threads executing *openg()* naming the same filename in the same directory with O_EXCL and O_CREAT set. If O_EXCL and O_CREAT are set, and *path* names a symbolic link, *openg*() shall fail and set *errno* to [EEXIST], regardless of the contents of the symbolic link. If O_EXCL is set and O_CREAT is not set, the result is undefined.

O_RSYNC

[SIO] Read I/O operations on the file descriptor shall complete at the same level of integrity as specified by the O_DSYNC and O_SYNC flags. If both O_DSYNC and O_RSYNC are set in *oflag*, all I/O operations on the file descriptor shall complete as defined by synchronized I/O data integrity completion. If both O_SYNC and O_RSYNC are set in flags, all I/O operations on the file descriptor shall complete as defined by synchronized I/O file integrity completion.

O_SYNC

[SIO] Write I/O operations on the file descriptor shall complete as defined by synchronized I/O file integrity completion.

O_TRUNC
: If the file exists and is a regular file, and the file is successfully opened O_RDWR or O_WRONLY, its length shall be truncated to 0, and the mode and owner shall be unchanged. It shall have no effect on FIFO special files or terminal device files. Its effect on other file types is implementation-defined. The result of using O_TRUNC with O_RDONLY is undefined.

If O_CREAT is set and the file did not previously exist, upon successful completion, *openg()* shall mark for update the *st_atime, st_ctime*, and *st_mtime* fields of the file and the *st_ctime* and *st_mtime* fields of the parent directory.

If O_TRUNC is set and the file did previously exist, upon successful completion, *openg()* shall mark for update the *st_ctime* and *st_mtime* fields of the file.

[SIO] If both the O_SYNC and O_DSYNC flags are set, the effect is as if only the O_SYNC flag was set.

[XSI] If *path* names the master side of a pseudo-terminal device, then it is unspecified whether *openg()* locks the slave side so that it cannot be opened. Conforming applications shall call *unlockpt()* before opening the slave side.

A pointer to an application opaque buffer must be provided by *handle*. The buffer content is updated always, even on error.

The largest value that can be represented correctly in an object of type **off_t** shall be established as the offset maximum in the open file description.

RETURN VALUE

Upon successful completion, the function shall return a non-**NULL** pointer to the update application opaque buffer passed in. Otherwise, **NULL** shall be returned and *errno* set to indicate the error. No files shall be created or modified if the function returns **NULL**.

ERRORS

The *openg()* function shall fail if:

[EACCES]
: Search permission is denied on a component of the path prefix, or the file exists and the permissions specified by *oflag* are denied, or the file does not exist and write permission is denied for the parent directory of the file to be created, or O_TRUNC is specified and write permission is denied, or the file is a device or pseudo-device file.

[EEXIST]
: O_CREAT and O_EXCL are set, and the named file exists.

[EINTR]
: A signal was caught during *openg()*.

[EINVAL]
: [SIO] The implementation does not support synchronized I/O for this file.

[EISDIR]
: The named file is a directory and *oflag* includes O_WRONLY or O_RDWR.

[ELOOP]
: A loop exists in symbolic links encountered during resolution of the *path* argument.

[ENOENT]
: O_CREAT is not set and the named file does not exist; or O_CREAT is set and either the path prefix does not exist or the *path* argument points to an empty string.

[ENOSPC]
: The directory or file system that would contain the new file cannot be expanded, the file does not exist, and O_CREAT is specified.

[ENOTDIR]
: A component of the path prefix is not a directory.

[ENXIO]
: O_NONBLOCK is set, the named file is a FIFO, O_WRONLY is set, and no process has the file open for reading.

[ENXIO]
: The named file is a character special or block special file, and the device associated with this special file does not exist.

[EOVERFLOW]
: The named file is a regular file and the size of the file cannot be represented correctly in an object of type **off_t**.

[EROFS]
: The named file resides on a read-only file system and either O_WRONLY, O_RDWR, O_CREAT (if the file does not exist), or O_TRUNC is set in the *oflag* argument.

The *openg()* function may fail if:

[EAGAIN]
: [XSI] The *path* argument names the slave side of a pseudo-terminal device that is locked.

[EINVAL]
: The value of the *oflag* argument is not valid.

[ELOOP]
: More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

[ENAMETOOLONG]
: As a result of encountering a symbolic link in resolution of the *path* argument, the length of the substituted pathname string exceeded {PATH_MAX}.

[ETXTBSY]
: The file is a pure procedure (shared text) file that is being executed and *oflag* is O_WRONLY or O_RDWR.

The following sections are informative.

EXAMPLES

Opening a File for Writing by the Owner

The following example opens the file **/tmp/file**, either by creating it (if it does not already exist), or by truncating its length to 0 (if it does exist). In the former case, if the call creates a new file, the access permission bits in the file mode of the file are set to permit reading and writing by the owner, and to permit reading only by group members and others.

If the call to *openg()* is successful, the file is opened for writing.

```
#include <fcntl.h>
...
fh_t fhandle, *fh;
mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
char *filename = "/tmp/file";
...
fh = openg(filename, O_WRONLY | O_CREAT | O_TRUNC, &fhandle, mode);
...
```

Opening a File Using an Existence Check

The following example uses the *openg()* function to try to create the **LOCKFILE** file and open it for writing. Since the *openg()* function specifies the O_EXCL flag, the call fails if the file already exists. In that case, the program assumes that someone else is updating the password file and exits.

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>


#define LOCKFILE "/etc/ptmp"
...
fh_t pfhandle, *pfh; /* Integer for file descriptor returned by openg()
call. */
...
if ((pfh = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL,
    S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH), &pfhandle) == -1)
{
    fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
    exit(1);
}
...
```

Opening a File for Writing

The following example opens a file for writing, creating the file if it does not already exist. If the file does exist, the system truncates the file to zero bytes.

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>


#define LOCKFILE "/etc/ptmp"
...
fh_t pfhandle, *pfh;
char filename[PATH_MAX+1];
...
if ((pfh = open(filename, O_WRONLY | O_CREAT | O_TRUNC,
    S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH), &pfhandle) == -1)
{
    perror("Cannot open output file\n"); exit(1);
}
...
```

APPLICATION USAGE

None.

RATIONALE

Except as specified in this volume of IEEE Std 1003.1-2001, the flags allowed in *oflag* are not mutually-exclusive and any number of them may be used simultaneously.

Devices and pseudo-devices may not be specified by the *path* argument.

See *getgroups()* about the group of a newly created file.

IEEE Std 1003.1-2001 permits [EACCES] to be returned for conditions other than those explicitly listed.

In historical implementations the value of O_RDONLY is zero. Because of that, it is not possible to detect the presence of O_RDONLY and another option. Future implementations should encode O_RDONLY and O_WRONLY as bit flags so that:

```
O_RDONLY | O_WRONLY == O_RDWR
```

In general, the *openg()* function follows the symbolic link if *path* names a symbolic link. However, the *openg()* function, when called with O_CREAT and O_EXCL, is required to fail with [EEXIST] if *path* names an existing symbolic link, even if the symbolic link refers to a nonexistent file. This behavior is required so that privileged applications can create a new file in a known location without the possibility that a symbolic link might cause the file to be created in a different location.

For example, a privileged application that must create a file with a predictable name in a user-writable directory, such as the user's home directory, could be compromised if the user creates a symbolic link with that name that refers to a nonexistent file in a system directory. If the user can influence the contents of a file, the user could compromise the system by creating a new system configuration or spool file that would then be interpreted by the system. The test for a symbolic link which refers to a nonexisting file must be atomic with the creation of a new file.

The POSIX.1-1990 standard required that the group ID of a newly created file be set to the group ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2 required that implementations provide a way to have the group ID be set to the group ID of the containing directory, but did not prohibit implementations also supporting a way to set the group ID to the effective group ID of the creating process. Conforming applications should not assume which group ID will be used. If it matters, an application can use *chown()* to set the group ID after the file is created, or determine under what conditions the implementation will set the desired group ID.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod(), *close()*, *creat()*, *dup()*, *fcntl()*, *lseek()*, *read()*, *sutoc()*, *umask()*, *unlockpt()*, *write()*, the Base Definitions volume of IEEE Std 1003.1-2001, *<fcntl.h>*, *<sys/stat.h>*, *<sys/types.h>*

CHANGE HISTORY

Proposed.

End of informative text.