NAME

filemon - the filemon device

SYNOPSIS

#include <dev/filemon/filemon.h>

DESCRIPTION

The **filemon** device allows a process to collect file operations data of its children. The device */dev/filemon* responds to two ioctl(2) calls.

filemon is not intended to be a security auditing tool. Many system calls are not tracked and binaries using a non-native ABI may not be fully audited. It is intended for auditing of processes for the purpose of determining their dependencies using an efficient and easily parsable format. An example of this is make(1) which uses this module with **.MAKE.MODE=meta** to handle incremental builds more smartly.

System calls are denoted using the following single letters:

‘A’ openat(2). The next log entry may be lacking an absolute path or be inaccurate.
‘C’ chdir(2)
‘D’ unlink(2)
‘E’ exec(2)
‘F’ fork(2), vfork(2)
‘L’ link(2), linkat(2), symlink(2)
‘M’ rename(2)
‘R’ open(2) or openat(2) for read
‘W’ open(2) or openat(2) for write
‘X’ _exit(2)

Note that ‘R’ following ‘W’ records can represent a single open(2) for R/W, or two separate open(2) calls, one for ‘R’ and one for ‘W’. Note that only successful system calls are captured.

IOCTLS

User mode programs communicate with the **filemon** driver through a number of ioctls which are described below. Each takes a single argument.

FILEMON_SET_FD Write the internal tracing buffer to the supplied open file descriptor.

FILEMON_SET_PID Child process ID to trace. This should normally be done under the control of a parent in the child after fork(2) but before anything else. See the example below.

RETURN VALUES

The **ioctl**() function returns the value 0 if successful; otherwise the value -1 is returned and the global variable *errno* is set to indicate the error.

ERRORS

The **ioctl**() system call with FILEMON_SET_FD will fail if:

[EEXIST] The **filemon** handle is already associated with a file descriptor.

[EINVAL] The file descriptor has an invalid type and cannot be used for tracing.

[EBADF] The file descriptor is invalid or not opened for writing.

The **ioctl**() system call with FILEMON_SET_PID will fail if:

[ESRCH] No process having the specified process ID exists.

[EBUSY] The process ID specified is already being traced and was not the current process.

The **close**() system call on the filemon file descriptor may fail with the errors from write(2) if any error is encountered while writing the log. It may also fail if:

[EFAULT] An invalid address was used for a traced system call argument, resulting in no log entry for the system call.

[ENAMETOOLONG] An argument for a traced system call was too long, resulting in no log entry for the system call.

FILES

/dev/filemon

EXAMPLES

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <sys/ioctl.h>
#include <dev/filemon/filemon.h>
#include <fcntl.h>
#include <err.h>
```

```
#include <errno.h>
#include <unistd.h>

static void
open_filemon(void)
{
        pid_t child, wait_rv;
        int fm_fd, fm_log;

        if ((fm_fd = open("/dev/filemon", O_RDWR | O_CLOEXEC)) == -1)
                err(1, "open(\"/dev/filemon\", O_RDWR)");
        if ((fm_log = open("filemon.out",
           O_CREAT | O_WRONLY | O_TRUNC | O_CLOEXEC, DEFFILEMODE)) == -1)
                err(1, "open(filemon.out)");

        if (ioctl(fm_fd, FILEMON_SET_FD, &fm_log) == -1)
                err(1, "Cannot set filemon log file descriptor");

        if ((child = fork()) == 0) {
                child = getpid();
                if (ioctl(fm_fd, FILEMON_SET_PID, &child) == -1)
                        err(1, "Cannot set filemon PID");
                /* Do something here. */
        } else if (child == -1)
                err(1, "Cannot fork child");
        else {
                while ((wait_rv = wait(&child)) == -1 &&
                   errno == EINTR)
                        ;
                if (wait_rv == -1)
                        err(1, "cannot wait for child");
                close(fm_fd);
        }
}
```

Creates a file named *filemon.out* and configures the **filemon** device to write the **filemon** buffer contents to it.

SEE ALSO

dtrace(1), ktrace(1), script(1), truss(1), ioctl(2)

HISTORY

A **filemon** device appeared in FreeBSD 9.1.

BUGS

Unloading the module may panic the system, thus requires using **kldunload -f**.