BAHIRDAR INSTITUTE OF TECHNOLOGY (BIT)

DEPARTMENT OF SOFTWARE ENGINEERING

OPERATING SYSTEM AND SYSTEM PROGRAMING INDIVIDUAL ASSIGNMENT

NAME: EDMEALEM KASSAHUN

ID NUMBER: BDU1306115

Submitted to : Mr Wondimu B.(Msc)

Submission date : 17/11/2014 E.C

WHAT IS FANOTIFY_MARK(2) SYSTEM CALL , HOW DOES IT WORK , WHY WE USE IT?

The fanotify API provides notification and interception (monitoring) of filesystem events. Use cases include virus scanning and hierarchical storage management. It introduces a set of system calls to obtain a file descriptor and "marks" filesystem objects as being watched. Fanotify is a file access notification system built in on many common Linux kernels. This kernel feature allows Sophos Anti-Virus to scan files on access and, if necessary, block access to threats.

- The following system calls are used with this API: fanotify_init(2), fanotify_mark(2), read(2), write(2), and close(2).

❖ The fanotify_mark (2) system call adds a file, directory, or mount to a notification group and specifies which events shall be reported (or ignored), or removes or modifies such an entry. The caller must have read permission on the filesystem object that is to be marked.

The fanotify_mark() system call multiplexes adding, removing, and flushing marks.

BRIEFLY DESCRIBE ABOUT THE LIST OF PARAMETERS AND FALGS?

❖ The sysnopsis of fanotify mark is :

```
#include <sys/fanotify.h>

int fanotify_mark(int fanotify_fd,

                  unsigned int flags,

                  uint64_t mask,

                  int dirfd,

                  const char *pathname);
```

parameters :

➢ The **fanotify_fd** argument is a file descriptor returned by fanotify_init(2).

➢ **flags** is a bit mask describing the modification to perform.

➢ **mask** defines which events shall be listened for (or which shall be ignored).

- The filesystem object to be marked is determined by the file descriptor **dirfd** and the **pathname** specified in pathname.

BRIEFE DESCRITION

- **A FLAG must include exactly one of the following values:**

FAN_MARK_ADD : The events in mask will be added to the mark mask (or to the ignore mask). mask must be nonempty or the error EINVAL will occur.

FAN_MARK_REMOVE : The events in argument mask will be removed from the mark mask (or from the ignore mask). mask must be nonempty or the error EINVAL will occur.

FAN_MARK_FLUSH : Remove either all marks for filesystems, all marks for mounts, or all marks for directories and files from the fanotify group. If flags contains FAN_MARK_MOUNT, all marks for mounts are removed from the group. If flags contains FAN_MARK_FILESYSTEM, all marks for filesystems are removed from the group. Otherwise, all marks for directories and files are removed.

- **MASK defines which events shall be listened for (or which shall be ignored). It is a bit mask composed of the following values:**

FAN_ACCESS Create an event when a file or directory (but see BUGS) is accessed (read).

FAN_MODIFY Create an event when a file is modified (write).

FAN_CLOSE_WRITE Create an event when a read-only file or directory is closed

FAN_OPEN Create an event when a file or directory is opened.

FAN_OPEN_EXEC (since Linux 5.0) Create an event when a file is opened with the intent to be executed.

FAN_ATTRIB (since Linux 5.1) Create an event when the metadata for a file or directory has changed.

FAN_CREATE (since Linux 5.1) Create an event when a file or directory has been created in a marked parent directory.

FAN_DELETE (since Linux 5.1) Create an event when a file or directory has been deleted in a marked parent directory.

FAN_DELETE_SELF (since Linux 5.1) Create an event when a marked file or directory itself is deleted.

FAN_MOVED_FROM (since Linux 5.1)Create an event when a file or directory has been moved from a marked parent directory.

FAN_MOVED_TO (since Linux 5.1)Create an event when a file or directory has been moved toa marked parent directory.

FAN_MOVE_SELF (since Linux 5.1)Create an event when a marked file or directory itself has been moved.

FAN_OPEN_PERM Create an event when a permission to open a file or directory is requested. An fanotify file descriptor created with FAN_CLASS_PRE_CONTENT or FAN_CLASS_CONTENT is required.

FAN_OPEN_EXEC_PERM (since Linux 5.0) Create an event when a permission to open a file forexecution is requested. An fanotify file descriptorcreated with FAN_CLASS_PRE_CONTENT or FAN_CLASS_CONTENT is required.

FAN_ACCESS_PERM Create an event when a permission to read a file or directory is requested. An fanotify file descriptor created with FAN_CLASS_PRE_CONTENT or FAN_CLASS_CONTENT is required.

FAN_ONDIR Create events for directories

FAN_EVENT_ON_CHILD Events for the immediate children of marked directories shall be created. The flag has no effect when marking mounts and filesystems.

- ❖ **The filesystem object to be marked is determined by the file descriptor dirfd and the pathname specified in pathname:**

If pathname is NULL, dirfd defines the filesystem object to be marked.

If pathname is NULL, and dirfd takes the special value AT_FDCWD, the current working directory is to be marked.

If pathname is absolute, it defines the filesystem object to be marked, and dirfd is ignored.

If pathname is relative, and dirfd does not have the value AT_FDCWD, then the filesystem object to be marked isdetermined by interpreting pathname relative the directory referred to by dirfd.

If pathname is relative, and dirfd has the value AT_FDCWD,then the filesystem object to be marked is determined by interpreting pathname relative to the current working directory.

- ❖ **RETURN VALUE On success, fanotify_mark() returns 0. On error, -1 is returned, and errno is set to indicate the error.**
- ❖ **ERRORS**

EBADF An invalid file descriptor was passed in fanotify_fd.

EBADF pathname is relative but dirfd is neither AT_FDCWD nor a valid file descriptor.

EINVAL An invalid value was passed in flags or mask, or fanotify_fd was not an fanotify file descriptor.

EINVAL The fanotify file descriptor was opened with FAN_CLASS_NOTIF or the fanotify group identifies filesystem objects by file handles and mask contains a flag for permission events (FAN_OPEN_PERM or FAN_ACCESS_PERM).

ENODEV The filesystem object indicated by pathname is not associated with a filesystem that supports fsid (e.g., tmpfs(5)).This error can be returned only with an fanotify group that identifies filesystem objects by file handles.

ENOENT The filesystem object indicated by dirfd and pathname does not exist.This error also occurs when trying to remove a mark from an object which is not marked.

ENOMEM The necessary memory could not be allocated.

ENOSPC The number of marks exceeds the limit of 8192 and the FAN_UNLIMITED_MARKS flag was not specified when the fanotify file descriptor was created with fanotify_init(2).

ENOSYS This kernel does not implement fanotify_mark(). The fanotify API is available only if the kernel was configured with CONFIG_FANOTIFY.

ENOTDIR flags contains FAN_MARK_ONLYDIR, and dirfd and pathname do not specify a directory.

EOPNOTSUPP The object indicated by pathname is associated with a filesystem that does not support the encoding of file handles. This error can be returned only with an fanotify group that identifies filesystem objects by file handles.

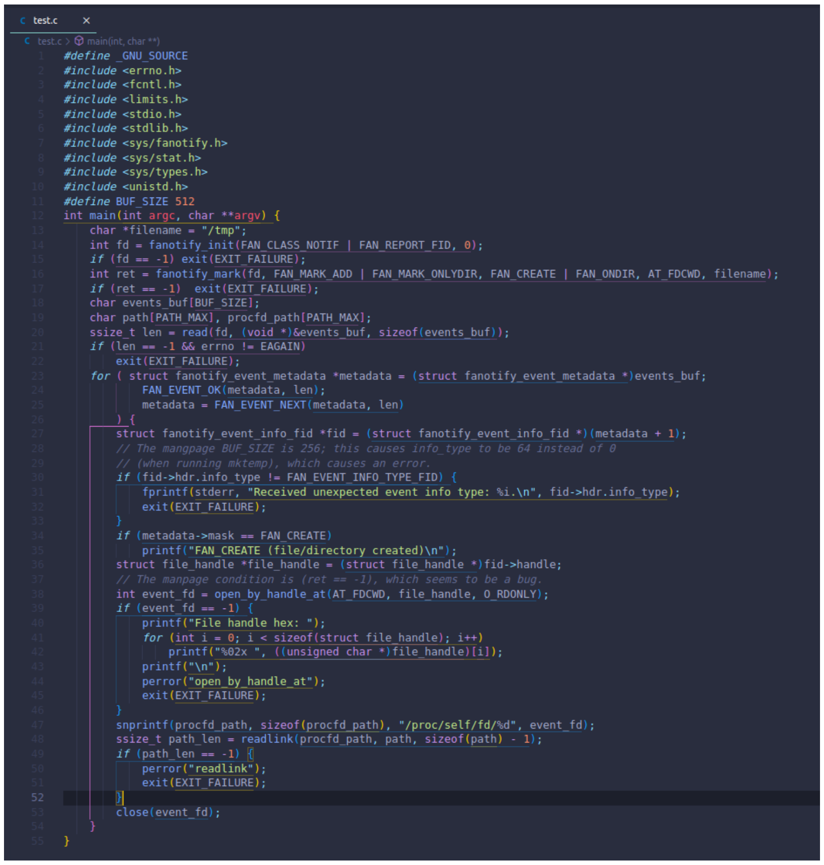
EXDEV The filesystem object indicated by pathname resides within a filesystem subvolume (e.g., btrfs(5)) which uses a different fsid than its root superblock. This error can be returned only with an fanotify group that identifies filesystem objects by file handles.

Here is a simple example

```
#define _GNU_SOURCE

#include <errno.h>

#include <fcntl.h>

#include <limits.h>

#include <stdio.h>
```

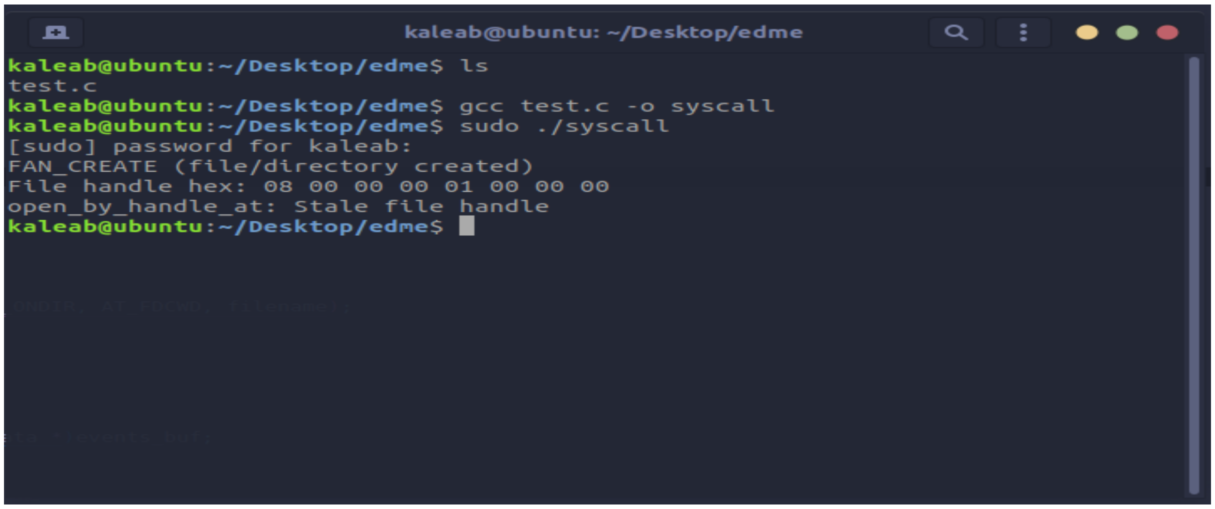
```
#include <stdlib.h>

#include <sys/fanotify.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <unistd.h>

#define BUF_SIZE 512

int main(int argc, char **argv) {

 char *filename = "/tmp";

  int fd = fanotify_init(FAN_CLASS_NOTIF | FAN_REPORT_FID, 0);

  if (fd == -1) exit(EXIT_FAILURE);

  int ret = fanotify_mark(fd, FAN_MARK_ADD | FAN_MARK_ONLYDIR, FAN_CREATE |
FAN_ONDIR, AT_FDCWD, filename);

  if (ret == -1)  exit(EXIT_FAILURE);

  char events_buf[BUF_SIZE];

  char path[PATH_MAX], procfd_path[PATH_MAX];

  ssize_t len = read(fd, (void *)&events_buf, sizeof(events_buf));

  if (len == -1 && errno != EAGAIN)

    exit(EXIT_FAILURE);

  for ( struct fanotify_event_metadata *metadata = (struct fanotify_event_metadata
*)events_buf;

    FAN_EVENT_OK(metadata, len);

    metadata = FAN_EVENT_NEXT(metadata, len)

  ) {

    struct fanotify_event_info_fid *fid = (struct fanotify_event_info_fid *)(metadata + 1);

    // The mangpage `BUF_SIZE` is 256; this causes `info_type` to be 64 instead of 0
```

```
   // (when running `mktemp`), which causes an error.
   if (fid->hdr.info_type != FAN_EVENT_INFO_TYPE_FID) {
     fprintf(stderr, "Received unexpected event info type: %i.\n", fid->hdr.info_type);
     exit(EXIT_FAILURE);   }
   if (metadata->mask == FAN_CREATE)
     printf("FAN_CREATE (file/directory created)\n");
   struct file_handle *file_handle = (struct file_handle *)fid->handle;
   // The manpage condition is `(ret == -1)`, which seems to be a bug.
   int event_fd = open_by_handle_at(AT_FDCWD, file_handle, O_RDONLY);
   if (event_fd == -1) {
     printf("File handle hex: ");
     for (int i = 0; i < sizeof(struct file_handle); i++)
       printf("%02x ", ((unsigned char *)file_handle)[i]);
     printf("\n");
    perror("open_by_handle_at");
     exit(EXIT_FAILURE);
   }
   snprintf(procfd_path, sizeof(procfd_path), "/proc/self/fd/%d", event_fd);
   ssize_t path_len = readlink(procfd_path, path, sizeof(path) - 1);
   if (path_len == -1) {
       perror("readlink");
       exit(EXIT_FAILURE);
 }   close(event_fd);}}
```



```c
#define _GNU_SOURCE
#include <errno.h>
#include <fcntl.h>
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/fanotify.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#define BUF_SIZE 512
int main(int argc, char **argv) {
    char *filename = "/tmp";
    int fd = fanotify_init(FAN_CLASS_NOTIF | FAN_REPORT_FID, 0);
    if (fd == -1) exit(EXIT_FAILURE);
    int ret = fanotify_mark(fd, FAN_MARK_ADD | FAN_MARK_ONLYDIR, FAN_CREATE | FAN_ONDIR, AT_FDCWD, filename);
    if (ret == -1)  exit(EXIT_FAILURE);
    char events_buf[BUF_SIZE];
    char path[PATH_MAX], procfd_path[PATH_MAX];
    ssize_t len = read(fd, (void *)&events_buf, sizeof(events_buf));
    if (len == -1 && errno != EAGAIN)
        exit(EXIT_FAILURE);
    for ( struct fanotify_event_metadata *metadata = (struct fanotify_event_metadata *)events_buf;
            FAN_EVENT_OK(metadata, len);
            metadata = FAN_EVENT_NEXT(metadata, len)
        ) {
        struct fanotify_event_info_fid *fid = (struct fanotify_event_info_fid *)(metadata + 1);
        // The mangpage BUF_SIZE is 256; this causes info_type to be 64 instead of 0
        // (when running mktemp), which causes an error.
        if (fid->hdr.info_type != FAN_EVENT_INFO_TYPE_FID) {
            fprintf(stderr, "Received unexpected event info type: %i.\n", fid->hdr.info_type);
            exit(EXIT_FAILURE);
        }
        if (metadata->mask == FAN_CREATE)
            printf("FAN_CREATE (file/directory created)\n");
        struct file_handle *file_handle = (struct file_handle *)fid->handle;
        // The manpage condition is (ret == -1), which seems to be a bug.
        int event_fd = open_by_handle_at(AT_FDCWD, file_handle, O_RDONLY);
        if (event_fd == -1) {
            printf("File handle hex: ");
            for (int i = 0; i < sizeof(struct file_handle); i++)
                printf("%02x ", ((unsigned char *)file_handle)[i]);
            printf("\n");
            perror("open_by_handle_at");
            exit(EXIT_FAILURE);
        }
        snprintf(procfd_path, sizeof(procfd_path), "/proc/self/fd/%d", event_fd);
        ssize_t path_len = readlink(procfd_path, path, sizeof(path) - 1);
        if (path_len == -1) {
            perror("readlink");
            exit(EXIT_FAILURE);
        }
        close(event_fd);
    }
}
```

Output



```
kaleab@ubuntu:~/Desktop/edme$ ls
test.c
kaleab@ubuntu:~/Desktop/edme$ gcc test.c -o syscall
kaleab@ubuntu:~/Desktop/edme$ sudo ./syscall
[sudo] password for kaleab:
FAN_CREATE (file/directory created)
File handle hex: 08 00 00 00 01 00 00 00
open_by_handle_at: Stale file handle
kaleab@ubuntu:~/Desktop/edme$
```

REFERENCE

- inotify - How do I program for Linux's new fanotify file system monitoring feature? - Stack Overflow
- https://man7.org/linux/man-pages/man2/fanotify_mark.2.html
- http://www.bricktou.com/fs/notify/fanotify/fanotify_userfanotify_remove_inode_mark_src.html