CurveFs 用户权限系统调研（已实现）

一、Curvefs测试

代码：https://github.com/cwl23/curve/tree/fs_s3_joint_debugging

环境：test2

1. 启动curvefs

手动创建curve卷，/etc/curve/client.conf中配置卷所在集群信息。

启动服务&client挂载卷：bash startfs.sh start volume（挂载目录为/tmp/fsmount）

```
#
wanghai01@pubbeta1-nostest2:~/curvefs/curve$ ps -ef | grep curvefs | grep wanghai
wanghai+ 2641513       1  0 13:44 pts/213  00:00:00
./bazel-bin/curvefs/src/space_allocator/curve_fs_space_allocator_main
wanghai+ 2641514       1  0 13:44 pts/213  00:00:00 ./bazel-bin/curvefs/src/metaserver/curvefs_metaserver
wanghai+ 2641515       1  0 13:44 pts/213  00:00:00 ./bazel-bin/curvefs/src/mds/curvefs_mds
wanghai+ 2642837 2589230  0 13:47 pts/156  00:00:00 ./bazel-bin/curvefs/src/client/fuse_client -f -o volume=/fs
-o user=test -o conf=./curvefs/conf/curvefs_client.conf /tmp/fsmount
```

问题1：root用户无法访问挂载目录

测试发现client mount进程是哪个用户启动的就只有该用户(filesystem owner)可以访问该目录，即使挂载点mode是777。

```
# filesystem owner
wanghai01@pubbeta1-nostest2:/tmp$ ls -l | grep fsmount
drwxrwxrwx  0 root         root                 0 Nov  9  2078 fsmount
wanghai01@pubbeta1-nostest2:/tmp$ cd fsmount/
wanghai01@pubbeta1-nostest2:/tmp/fsmount$ ls
wanghai01@pubbeta1-nostest2:/tmp/fsmount$

# nbs user
nbs@pubbeta1-nostest2:/tmp$ ls -l | grep fsmount
ls: cannot access 'fsmount': Permission denied
d?????????  ? ?            ?                    ?            ? fsmount
nbs@pubbeta1-nostest2:/tmp$ cd fsmount
-bash: cd: fsmount: Permission denied

# root user
root@pubbeta1-nostest2:/tmp# ls -l | grep fsmount
ls: cannot access 'fsmount': Permission denied
d?????????  ? ?            ?                    ?            ? fsmount
root@pubbeta1-nostest2:/tmp# cd fsmount
bash: cd: fsmount: Permission denied
```

查阅资料发现这是fuse的一种安全策略，默认是只有filesystem owner拥有该文件系统的访问权限，如果想要其他用户有权访问，需要在挂载参数中指定‘-o allow-root’ 或’-o allow-other’以允许相应用户有权访问该文件系统，如果挂载者不是root还需要在/etc/fuse.conf（/usr/local/etc/fuse.conf）中增加配置项“user_allow_other”（该配置项是无值的）。详见libfuse官方文档：https://github.com/libfuse/libfuse#security-implications

```
# The file /etc/fuse.conf allows for the following parameters:
#
# user_allow_other - Using the allow_other mount option works fine as root, in
# order to have it work as user you need user_allow_other in /etc/fuse.conf as
# well. (This option allows users to use the allow_other option.) You need
# allow_other if you want users other than the owner to access a mounted fuse.
# This option must appear on a line by itself. There is no value, just the
# presence of the option.

user_allow_other


# mount_max = n - this option sets the maximum number of mounts.
# Currently (2014) it must be typed exactly as shown
# (with a single space before and after the equals sign).

#mount_max = 100


# man fuse
user_allow_other
              Allow non-root users to specify the allow_other or
              allow_root mount options (see below).
allow_other
              This option overrides the security measure restricting
              file access to the filesystem owner, so that all users
              (including root) can access the files.
allow_root
              This option is similar to allow_other but file access is
              limited to the filesystem owner and root.  This option and
              allow_other are mutually exclusive.
```

测试 allow_root

```
wanghai+ 2641513       1  0 13:44 ?        00:00:00 ./bazel-bin/curvefs/src/space_allocator/curve_fs_space_allocator_main
wanghai+ 2641514       1  0 13:44 ?        00:00:01 ./bazel-bin/curvefs/src/metaserver/curvefs_metaserver
wanghai+ 2641515       1  0 13:44 ?        00:00:00 ./bazel-bin/curvefs/src/mds/curvefs_mds
wanghai+ 2650159 2589230  0 14:01 pts/156  00:00:00 ./bazel-bin/curvefs/src/client/fuse_client -f -o volume=/fs -o allow_root -o user=test -o conf=./curvefs/conf/curvefs_client.conf /tmp/fsmount
wanghai+ 2650638 2589230  0 14:01 pts/156  00:00:00 grep curvefs
```

```
# filesystem owner
wanghai01@pubbeta1-nostest2:/tmp$ ls -l | grep fsmount
drwxrwxrwx  0 root         root                  0 Nov 27  2078 fsmount
wanghai01@pubbeta1-nostest2:/tmp$ cd fsmount/
wanghai01@pubbeta1-nostest2:/tmp/fsmount$ ls
wanghai01@pubbeta1-nostest2:/tmp/fsmount$

# nbs user
nbs@pubbeta1-nostest2:/tmp$ ls -l | grep fsmount
ls: cannot access 'fsmount': Permission denied
d?????????  ? ?            ?                     ?            ? fsmount
nbs@pubbeta1-nostest2:/tmp$ cd fsmount
nbs@pubbeta1-nostest2:/tmp/fsmount$ ls
ls: cannot open directory '.': Permission denied

# root user
root@pubbeta1-nostest2:/tmp# ls -l | grep fsmount
drwxrwxrwx  0 root         root                  0 Nov 27  2078 fsmount
root@pubbeta1-nostest2:/tmp# cd fsmount/
root@pubbeta1-nostest2:/tmp/fsmount# ls
root@pubbeta1-nostest2:/tmp/fsmount#
```

测试allow_other

```
wanghai+ 2661031       1  0 14:13 pts/156  00:00:00 ./bazel-bin/curvefs/src/space_allocator/curve_fs_space_allocator_main
wanghai+ 2661032       1  0 14:13 pts/156  00:00:00 ./bazel-bin/curvefs/src/metaserver/curvefs_metaserver
wanghai+ 2661033       1  0 14:13 pts/156  00:00:00 ./bazel-bin/curvefs/src/mds/curvefs_mds
wanghai+ 2664541 2589230  0 14:21 pts/156  00:00:00 ./bazel-bin/curvefs/src/client/fuse_client -f -o volume=/fs -o allow_other -o user=test -o conf=./curvefs/conf/curvefs_client.conf /tmp/fsmount
wanghai+ 2664949 2589230  0 14:21 pts/156  00:00:00 grep curvefs
```

```
# filesystem owner
wanghai01@pubbeta1-nostest2:/tmp$ ls -l | grep fsmount
drwxrwxrwx  0 root         root                 0 Dec  4  2078 fsmount
wanghai01@pubbeta1-nostest2:/tmp$ cd fsmount/
wanghai01@pubbeta1-nostest2:/tmp/fsmount$ ls
wanghai01@pubbeta1-nostest2:/tmp/fsmount$

# nbs user
nbs@pubbeta1-nostest2:/tmp$ ls -l | grep fsmount
drwxrwxrwx  0 root         root                 0 Dec  4  2078 fsmount
nbs@pubbeta1-nostest2:/tmp$ cd fsmount/
nbs@pubbeta1-nostest2:/tmp/fsmount$ ls
nbs@pubbeta1-nostest2:/tmp/fsmount$

# root user
root@pubbeta1-nostest2:/tmp# ls -l | grep fsmount
drwxrwxrwx  0 root         root                 0 Dec  4  2078 fsmount
root@pubbeta1-nostest2:/tmp# cd fsmount/
root@pubbeta1-nostest2:/tmp/fsmount# ls
root@pubbeta1-nostest2:/tmp/fsmount#
```

参考文献

Why does root get Permission denied when accessing FUSE directory?

man fuse

问题2：本地文件系统挂载默认是共享的？

目前没有查到相关确切的资料说明，但是从现象上看本地文件系统默认是多用户共享的，但是fuse作为用户态文件系统默认访问权限是文件系统的拥有者，可以通过allow_other实现共享。

```
strace ./bazel-bin/curvefs/src/client/fuse_client -f -o volume=/fs -o allow_other -o user=test -o
conf=./curvefs/conf/curvefs_client.conf /tmp/fsmount
...
mount("fuse_client", "/tmp/fsmount", "fuse.fuse_client", MS_NOSUID|MS_NODEV,
"allow_other,fd=9,rootmode=40000,"...) = 0
```

问题3：文件系统访问控制是在哪一层实现的？

测试curvefs，发现文件系统链路默认是没有做权限控制。（挂载点mode 777）

```
# mountpoint
wanghai01@pubbeta1-nostest2:/tmp$ ls -l | grep fsmount
drwxrwxrwx  0 root         root                 0 Jan  6  2079 fsmount

wanghai01@pubbeta1-nostest2:/tmp/fsmount$ touch file1
wanghai01@pubbeta1-nostest2:/tmp/fsmount$ ls -l
total 0
-rw-r--r-- 0 wanghai01 neteaseusers 0 Jan  7  2079 file1
wanghai01@pubbeta1-nostest2:/tmp/fsmount$ echo "hello" > file1
wanghai01@pubbeta1-nostest2:/tmp/fsmount$ cat file1
hello
wanghai01@pubbeta1-nostest2:/tmp/fsmount$ sudo -iu nbs
nbs@pubbeta1-nostest2:~$ cd /tmp/fsmount/
nbs@pubbeta1-nostest2:/tmp/fsmount$ ls -l
total 0
-rw-r--r-- 0 wanghai01 neteaseusers 6 Jan  7  2079 file1
nbs@pubbeta1-nostest2:/tmp/fsmount$ echo "world" >> file1
nbs@pubbeta1-nostest2:/tmp/fsmount$ cat file1
hello
world
```

测试curvefs，发现文件系统链路默认是没有做权限控制。（挂载点mode 755）

```
wanghai01@pubbeta1-nostest2:/tmp$ ls -l | grep fsmount
drwxr-xr-x  0 root         root                 0 Apr  9  2081 fsmount
wanghai01@pubbeta1-nostest2:/tmp$ cd fsmount/
wanghai01@pubbeta1-nostest2:/tmp/fsmount$ ls
wanghai01@pubbeta1-nostest2:/tmp/fsmount$ touch file1
wanghai01@pubbeta1-nostest2:/tmp/fsmount$ ls -l
total 0
-rw-r--r-- 0 wanghai01 neteaseusers 0 Apr 10  2081 file1
wanghai01@pubbeta1-nostest2:/tmp/fsmount$ mkdir folder
wanghai01@pubbeta1-nostest2:/tmp/fsmount$ ls -l
total 0
-rw-r--r-- 0 wanghai01 neteaseusers 0 Apr 10  2081 file1
drwxr-xr-x 0 wanghai01 neteaseusers 0 Apr 10  2081 folder
```

fuse默认是不会检查文件访问权限的，它允许在文件系统中自由的实现访问控制策略或将其下放到底层文件访问机制(e.g. in case of network filesystems)。挂载参数‘default_permissions’用于启用内核自己的权限检查，而不是将权限检查推迟到文件系统，除了文件系统的任何权限检查之外，内核还会进行检查，并且两者都必须成功才能允许操作。 内核执行标准的 UNIX 权限检查。如果文件系统在打开设备 fd 时的初始功能协商期间启用了 ACL 支持，则此挂载选项将被隐式激活。 在这种情况下，内核执行 ACL 和标准的 unix 权限检查。

疑问：协商期间do_init()中的启用ACL的flags如何设置？

◆ FUSE_CAP_POSIX_ACL

#define FUSE_CAP_POSIX_ACL (1 << 19)

Indicates support for POSIX ACLs.

If this feature is enabled, the kernel will cache and have responsibility for enforcing ACLs. ACL will be stored as xattrs and passed to userspace, which is responsible for updating the ACLs in the filesystem, keeping the file mode in sync with the ACL, and ensuring inheritance of default ACLs when new filesystem nodes are created. Note that this requires that the file system is able to parse and interpret the xattr representation of ACLs.

Enabling this feature implicitly turns on the `default_permissions` mount option (even if it was not passed to mount(2)).

This feature is disabled by default.

Definition at line 338 of file fuse_common.h.

初始化时的功能协商通过init()函数实现：

- The treatment of low-level options has been made more consistent:

  Options that can be set in the init() handler (via the fuse_conn_info parameter) can now be set only here, i.e. fuse_session_new() no longer accepts arguments that change the fuse_conn_info object before or after the call do init(). As a side effect, this removes the ambiguity where some options can be overwritten by init(), while others overwrite the choices made by init().

  For file systems that wish to offer command line options for these settings, the new fuse_parse_conn_info_opts() and fuse_apply_conn_info_opts() functions are available.

  Consequently, the fuse_lowlevel_help() method has been dropped.

```
// libfuse lib/fuse_lowlevel.c
static void do_init(fuse_req_t req, fuse_ino_t nodeid, const void *inarg)
{
 struct fuse_init_in *arg = (struct fuse_init_in *) inarg;
 struct fuse_session *se = req->se;
 ...
 if (arg->flags & FUSE_POSIX_ACL)
   se->conn.capable |= FUSE_CAP_POSIX_ACL;
 ...
 se->op.init(se->userdata, &se->conn);
}
// libfuse include/fuse_kernel.h
// This is the first request sent by the kernel to the daemon. It is used to negotiate the protocol version and
other filesystem parameters.
struct fuse_init_in {
 uint32_t major;
 uint32_t minor;
 uint32_t max_readahead;
 uint32_t flags;
};

// libfuse include/fuse.h
/**
```

```
     * Initialize filesystem
     *
     * The return value will passed in the private_data field of
     * fuse_context to all file operations and as a parameter to the
     * destroy() method.
     *
     * Introduced in version 2.3
     * Changed in version 2.6
     */
    void *(*init) (struct fuse_conn_info *conn);


// libfuse  include/fuse_common.h
// Capability flags that the filesystem wants to enable.
// unsigned want;
struct fuse_conn_info {
    /**
     * Major version of the protocol (read-only)
     */
    unsigned proto_major;

    /**
     * Minor version of the protocol (read-only)
     */
    unsigned proto_minor;

    /**
     * Maximum size of the write buffer
     */
    unsigned max_write;

    /**
     * Maximum size of read requests. A value of zero indicates no
     * limit. However, even if the filesystem does not specify a
     * limit, the maximum size of read requests will still be
     * limited by the kernel.
     *
     * NOTE: For the time being, the maximum size of read requests
```

```
 * must be set both here *and* passed to fuse_session_new()
 * using the ``-o max_read=<n>`` mount option. At some point
 * in the future, specifying the mount option will no longer
 * be necessary.
 */
unsigned max_read;

/**
 * Maximum readahead
 */
unsigned max_readahead;

/**
 * Capability flags that the kernel supports (read-only)
 */
unsigned capable;

/**
 * Capability flags that the filesystem wants to enable.
 *
 * libfuse attempts to initialize this field with
 * reasonable default values before calling the init() handler.
 */
unsigned want;

/**
 * Maximum number of pending "background" requests. A
 * background request is any type of request for which the
 * total number is not limited by other means. As of kernel
 * 4.8, only two types of requests fall into this category:
 *
 *   1. Read-ahead requests
 *   2. Asynchronous direct I/O requests
 *
 * Read-ahead requests are generated (if max_readahead is
 * non-zero) by the kernel to preemptively fill its caches
 * when it anticipates that userspace will soon read more
 * data.
```

```
 *
 * Asynchronous direct I/O requests are generated if
 * FUSE_CAP_ASYNC_DIO is enabled and userspace submits a large
 * direct I/O request. In this case the kernel will internally
 * split it up into multiple smaller requests and submit them
 * to the filesystem concurrently.
 *
 * Note that the following requests are *not* background
 * requests: writeback requests (limited by the kernel's
 * flusher algorithm), regular (i.e., synchronous and
 * buffered) userspace read/write requests (limited to one per
 * thread), asynchronous read requests (Linux's io_submit(2)
 * call actually blocks, so these are also limited to one per
 * thread).
 */
unsigned max_background;

/**
 * Kernel congestion threshold parameter. If the number of pending
 * background requests exceeds this number, the FUSE kernel module will
 * mark the filesystem as "congested". This instructs the kernel to
 * expect that queued requests will take some time to complete, and to
 * adjust its algorithms accordingly (e.g. by putting a waiting thread
 * to sleep instead of using a busy-loop).
 */
unsigned congestion_threshold;

/**
 * When FUSE_CAP_WRITEBACK_CACHE is enabled, the kernel is responsible
 * for updating mtime and ctime when write requests are received. The
 * updated values are passed to the filesystem with setattr() requests.
 * However, if the filesystem does not support the full resolution of
 * the kernel timestamps (nanoseconds), the mtime and ctime values used
 * by kernel and filesystem will differ (and result in an apparent
 * change of times after a cache flush).
 *
 * To prevent this problem, this variable can be used to inform the
 * kernel about the timestamp granularity supported by the file-system.
```

```
     * The value should be power of 10.  The default is 1, i.e. full
     * nano-second resolution. Filesystems supporting only second resolution
     * should set this to 1000000000.
     */
    unsigned time_gran;

    /**
     * For future use.
```

```
    */
   unsigned reserved[22];
  };
```

未实现任何权限检查的文件系统通常应在内部添加此选项，可与参数‘allow_other’一起达到共享文件访问控制。

```
# LD_PRELOAD=/usr/lib/x86_64-linux-gnu/libjemalloc.so.1 ./bazel-bin/curvefs/src/client/fuse_client -f -o
volume=/fs -o allow_other -o default_permissions -o user=test -o conf=./curvefs/conf/curvefs_client.conf
/tmp/fsmount
root@pubbeta1-nostest2:/tmp# ls -l | grep fsmount
drwxr-xr-x  0 root         root                 0 Feb 17  2090 fsmount
root@pubbeta1-nostest2:/tmp# cd fsmount/
root@pubbeta1-nostest2:/tmp/fsmount# ls
root@pubbeta1-nostest2:/tmp/fsmount# touch hello.txt
root@pubbeta1-nostest2:/tmp/fsmount# ls
hello.txt
root@pubbeta1-nostest2:/tmp/fsmount# echo "hello" >> hello.txt
root@pubbeta1-nostest2:/tmp/fsmount# cat hello.txt
hello

# other user
wanghai01@pubbeta1-nostest2:/tmp/fsmount$ touch file1
touch: cannot touch 'file1': Permission denied
wanghai01@pubbeta1-nostest2:/tmp/fsmount$ echo "world" >> hello.txt
-bash: hello.txt: Permission denied

# chmod 777
root@pubbeta1-nostest2:/tmp/fsmount# chmod 777 hello.txt
root@pubbeta1-nostest2:/tmp/fsmount# ls -l
total 0
-rwxrwxrwx 0 root root 6 Feb 17  2090 hello.txt

# after chmod
wanghai01@pubbeta1-nostest2:/tmp/fsmount$ echo "world" >> hello.txt
wanghai01@pubbeta1-nostest2:/tmp/fsmount$ cat hello.txt
hello
world
```

结论：fuse挂载时使用’default_permissions’ 和 ‘allow_other’可以达到共享文件系统下的基于内核权限检查的文件访问控制；或者可以在用户态文件系统中自由的实现访问控制策略。

二、文件系统权限管理

```
// example in linux
// ---------
drwxr-xr-x  3 wanghai01 neteaseusers 4096 Jul 16 10:41 .
drwx------ 17 wanghai01 neteaseusers 4096 Jul 16 10:38 ..
drwxr-xr-x  2 wanghai01 neteaseusers 4096 Jul 16 10:39 directory
-rw-r--r--  2 wanghai01 neteaseusers    0 Jul 16 10:40 file
-rw-r--r--  2 wanghai01 neteaseusers    0 Jul 16 10:40 hardlink
lrwxrwxrwx  1 wanghai01 neteaseusers    4 Jul 16 10:41 softlink -> file
```

文件类型

文件类型标识	文件类型
-	普通文件
d	目录文件
l	符号链接
s	套接字（伪文件）
b	块设备（伪文件）
c	字符设备（伪文件）
p	管道（伪文件）

文件权限

文件权限分为三段：分别对应文件“属主权限”、“属组权限”、“其他用户权限”

权限标识	权限类型
-	无权限
r	读权限4
w	写权限2
x	执行权限1

特殊权限(SUID, SGID, STICKY)

SUID: 仅设置在可执行的文件上。默认情况下，当用户执行此类可执行文件时，被发起的进程的所有者不是进程发起者，而是可执行文件的所有者；换句话说，进程以所有者的身份运行。权限所显示的位置在文件的属主的权限位中的执行权限位上，如果属主本来就具有执行权限，则显示为“s”，如果本来没有执行权限，则显示为“S”。

```
# SUID
# chmod u+s FILE
# chmod 4755 FILE
wanghai01@pubbeta1-nostest2:~/tmp$ ls -l
total 0
-rwxr-xr-x 1 wanghai01 neteaseusers 0 Jul 29 10:37 file1
-rw-r-xr-x 1 wanghai01 neteaseusers 0 Jul 29 10:37 file2
wanghai01@pubbeta1-nostest2:~/tmp$ chmod u+s file1
wanghai01@pubbeta1-nostest2:~/tmp$ chmod 4655 file2
wanghai01@pubbeta1-nostest2:~/tmp$ ls -l
total 0
-rwsr-xr-x 1 wanghai01 neteaseusers 0 Jul 29 10:37 file1
-rwSr-xr-x 1 wanghai01 neteaseusers 0 Jul 29 10:37 file2
```

SGID: SGID可设置在可执行文件或目录的属组权限位的执行权限上。如果某个目录设置了SGID权限，并且对于某些用户有写权限，则所有在此目录创建的新文件和目录的所属组均为其父目录的所属组，而并非进程发起者的主要组。SGID权限的显示位置在文件的属组权限位上的执行权限上；如果属组本来就有执行权限。则显示为″s″，否则，就显示为“S”；

```
# SGID
# chmod g+s DIR
# chmod 2770 DIR
wanghai01@pubbeta1-nostest2:~$ ls -l | grep tmp
drwxr-xr-x  2 wanghai01 neteaseusers   4096 Jul 29 10:44 tmp
wanghai01@pubbeta1-nostest2:~$ chmod g+s tmp
wanghai01@pubbeta1-nostest2:~$ ls -l | grep tmp
drwxr-sr-x  2 wanghai01 neteaseusers   4096 Jul 29 10:44 tmp
wanghai01@pubbeta1-nostest2:~$ touch ./tmp/file1

root@pubbeta1-nostest2:/home/wanghai01# touch ./tmp/file2
wanghai01@pubbeta1-nostest2:~/tmp$ ls -l
total 0
-rw-r--r-- 1 wanghai01 neteaseusers 0 Jul 29 10:47 file1
-rw-r--r-- 1 root      neteaseusers 0 Jul 29 10:47 file2
```

STICKY: 仅设置在目录的其他用户权限位的执行权限上。如果在某个目录上的权限设置为多个用户都拥有写权限，那就意味着凡是拥有写权限的用户都能直接管理该目录中的所有文件名，包括改名文件及删除文件名等操作；因此需要在这样的目录上设置STICKY特殊权限；如果此类目录设置了STICKY，则所有用户即便拥有写权限，也仅能删除或改名所有者为其自身的文件；

```
# STICKY
# chmod o+t DIR
# chmod 1777 DIR
nbs@pubbeta1-nostest2:/tmp$ ls -l | grep stmp
drwxrwxrwx  2 nbs          netease           4096 Jul 29 11:02 stmp
nbs@pubbeta1-nostest2:/tmp$ chmod o+t stmp
nbs@pubbeta1-nostest2:/tmp/stmp$ ls -l
total 0
-rw-r--r-- 1 nbs netease 0 Jul 29 11:03 file1

wanghai01@pubbeta1-nostest2:/tmp/stmp$ rm -f file1
rm: cannot remove 'file1': Operation not permitted
```

文件默认权限umask

为什么默认创建的目录权限为755，文件为644？ 在linux系统中，创建一个新的文件或者目录时，这些新的文件或目录都会有默认的访问权限
umask命令与文件和目录的默认访问权限有关，umask值则表明了需要从默认权限中去掉哪些权限来成 为最终的默认权限值。
root@pubbeta2-curve5:~# umask 0022
默认目录权限为777，文件为666，经过与umask作用，最终权限为755、644

```
Windows  Linux  umask
/etc/login.defs

# UMASK is the default umask value for pam_umask and is used by
# useradd and newusers to set the mode of the new home directories.
# 022 is the "historical" value in Debian for UMASK
# 027, or even 077, could be considered better for privacy
# There is no One True Answer here : each sysadmin must make up his/her
# mind.
...
UMASK           022
...
```

用户&用户组

用户的角色是通过UID和GID在系统内进行识别的，username 和 group name是便于人工记忆，它们和uid、gid是一一对应的关系。
UID（User Identify） GID（Group Identity）
超级用户： UID:0 默认是root用户，UID为0的用户为超级用户，
虚拟用户： UID:1~499 与真实普通用户区分开来，这类用户最大的特点是安装系统后默认就会存在，且默认情况大多数不能登录系统
普通用户： UID:500~65535 具备系统管理员root的权限的运维人员添加的，权限很小，一般用sudo管理提权
用户和用户组的关系： 一对一、一对多、多对一、多对多

文件系统用户权限管理

对mode的管理

```
uidgidmode
message Inode {
    required uint64 inodeId = 1;
    required uint32 fsId = 2;
    required uint64 length = 3;
    required uint32 ctime = 4;
    required uint32 mtime = 5;
    required uint32 atime = 6;
    required uint32 uid = 7;
    required uint32 gid = 8;
    required uint32 mode = 9;
    required sint32 nlink = 10;
    required FsFileType type = 11;
    optional string symlink = 12;   // TYPE_SYM_LINK only
    optional VolumeExtentList volumeExtentList = 13;  // TYPE_FILE only
}
```

1. 创建目录、文件时设置uid、gid、mode信息
2. client对文件操作前进行鉴权
3. 实现chown chmod进行权限更改

```
fuseuid gid
struct fuse_ctx {
        uid_t uid;
        gid_t gid;
        pid_t pid;
        mode_t umask;
 };

// in fuse_lowlevel.h is
const struct fuse_ctx *fuse_req_ctx(fuse_req_t req)
```

对ACL（Access Control Lists）的管理

访问控制列表（ACL 或 POSIX ACL）是多用户系统的附加安全控制功能。 与基本的 POSIX RWX 位相比，POSIX ACL 有助于对文件系统权限进行更灵活、更细粒度的控制。可以针对用户（User）、群组（Group）、默认属性掩码（umask）进行设置。

ACL是Linux系统权限额外支持的一项功能，需要文件系统的支持，例如：ReiserFS ， EXT2 ， EXT3 ， EXT4 ， JFS ， XFS等都支持ACL功能。使用‘dumpe2fs’命令查看你的ACL功能是否启用：

```
# acl
root@pubbeta2-curve5:/home/nbs# dumpe2fs -h  /dev/sdk
dumpe2fs 1.43.4 (31-Jan-2017)
Filesystem volume name:   <none>
Last mounted on:          /data/chunkserver16
Filesystem UUID:          5ba783e9-44bd-49ce-b8bc-b7ba0ef33531
Filesystem magic number:  0xEF53
Filesystem revision #:    1 (dynamic)
Filesystem features:      has_journal ext_attr resize_inode dir_index filetype needs_recovery extent 64bit
flex_bg sparse_super large_file huge_file dir_nlink extra_isize metadata_csum
Filesystem flags:         signed_directory_hash
Default mount options:    user_xattr acl
...

# aclacl
mount -o remount, acl [mount point]

# ACL , getfacl --help
```

```
# getfacl: ACL
# setfacl: ACL
# chacl: Access ACL and Default ACL
root@pubbeta2-curve5:/home/nbs# getfacl --help
bash: getfacl: command not found
root@pubbeta2-curve5:/home/nbs# apt policy acl
acl:
  Installed: (none)
  Candidate: 2.2.52-3+b1
  Version table:
     2.2.52-3+b1 500
        500 http://debian.hz.netease.com/debian-current stretch/main amd64 Packages
root@pubbeta2-curve5:/home/nbs# apt-get install acl
root@pubbeta2-curve5:/home/nbs# getfacl --help
getfacl 2.2.52 -- get file access control lists
Usage: getfacl [-aceEsRLPtpndvh] file ...
  -a,  --access           display the file access control list only
  -d, --default           display the default access control list only
  -c, --omit-header       do not display the comment header
  -e, --all-effective     print all effective rights
  -E, --no-effective      print no effective rights
  -s, --skip-base         skip files that only have the base entries
  -R, --recursive         recurse into subdirectories
  -L, --logical           logical walk, follow symbolic links
  -P, --physical          physical walk, do not follow symbolic links
  -t, --tabular           use tabular output format
  -n, --numeric           print numeric user/group identifiers
```

```
  -p, --absolute-names    don't strip leading '/' in pathnames
  -v, --version           print version and exit
  -h, --help              this help text
```

ACL的使用规则和原理：

ACL是由一系列的Access Entry所组成的，每一条Access Entry定义了特定的类别可以对文件拥有的操作权限。Access Entry有三个组成部分：Entry tag type, qualifier (optional), permission。

```
# example an Access Entry
user:john:rw-
group:dev:r--

1. Entry tag type:
ACL_USER_OBJLinuxfile_ownerpermission
ACL_USERpermission
ACL_GROUP_OBJLinuxgrouppermission
ACL_GROUPpermission
ACL_MASKACL_USER, ACL_GROUP_OBJACL_GROUP
ACL_OTHERLinuxotherpermission
2. Qualifier

3. Permission
RWX

# test set acl
# acl.test
root@pubbeta2-curve5:/tmp# ls -l | grep acl.test
-rw-r--r-- 1 root        root            0 Jul 25 12:13 acl.test
root@pubbeta2-curve5:/tmp# getfacl acl.test
# file: acl.test
# owner: root
# group: root
user::rw-
group::r--
other::r--

# wanghai01w
```

```
# before set
wanghai01@pubbeta2-curve5:/tmp$ echo "hello" >> acl.test
-bash: acl.test: Permission denied
# set acl
root@pubbeta2-curve5:/tmp# setfacl -m u:wanghai01:rw- acl.test
root@pubbeta2-curve5:/tmp# ls -l | grep acl.test
-rw-rw-r--+ 1 root        root            0 Jul 25 12:13 acl.test
root@pubbeta2-curve5:/tmp# getfacl acl.test
# file: acl.test
# owner: root
# group: root
user::rw-
user:wanghai01:rw-
group::r--
mask::rw-
other::r--
# after set
wanghai01@pubbeta2-curve5:/tmp$ echo "hello" >> acl.test
wanghai01@pubbeta2-curve5:/tmp$ cat acl.test
hello

ls -l ‘+’;
getaclaccess entry 'user:wanghai01:rw-',Posix u/g/o
mask::rw-: ACL_USERACL_GROUP_OBJACL_GROUP
root@pubbeta2-curve5:/tmp# getfacl -e acl.test
# file: acl.test
# owner: root
# group: root
user::rw-
user:wanghai01:rw-  #effective:rw-
group::r--          #effective:r--
mask::rw-
other::r--
root@pubbeta2-curve5:/tmp# setfacl -m mask::r-- acl.test
root@pubbeta2-curve5:/tmp# getfacl -e acl.test
# file: acl.test
# owner: root
# group: root
```

```
user::rw-
user:wanghai01:rw-  #effective:r--
group::r--       #effective:r--
mask::r--
other::r--
wanghai01@pubbeta2-curve5:/tmp$ echo "hello" >> acl.test
-bash: acl.test: Permission denied
# aclls -lgroupmaskgetfacl -e
# maskrwxr--
root@pubbeta2-curve5:/tmp# setfacl -m mask::rwx acl.test
root@pubbeta2-curve5:/tmp# ls -l | grep acl.test
-rw-rwxr--+ 1 root        root            6 Jul 25 12:20 acl.test
root@pubbeta2-curve5:/tmp# getfacl -e acl.test
# file: acl.test
# owner: root
# group: root
user::rw-
user:wanghai01:rw-  #effective:rw-
group::r--       #effective:r--
mask::rwx
other::r--

# Default ACL
# Default ACLDefault ACLACL
root@pubbeta2-curve5:/tmp# mkdir acl
root@pubbeta2-curve5:/tmp# ls -l | grep acl
drwxr-xr-x  2 root        root         4096 Jul 25 12:59 acl
root@pubbeta2-curve5:/tmp# getfacl -e acl
# file: acl
# owner: root
# group: root
user::rwx
group::r-x
other::r-x
# Default acl
root@pubbeta2-curve5:/tmp# setfacl -d -m u:wanghai01:rwx acl
root@pubbeta2-curve5:/tmp# ls -l | grep acl
drwxr-xr-x+ 2 root        root          4096 Jul 25 12:59 acl
```

```
root@pubbeta2-curve5:/tmp# getfacl -e acl
# file: acl
# owner: root
# group: root
user::rwx
group::r-x
other::r-x
default:user::rwx
default:user:wanghai01:rwx #effective:rwx
default:group::r-x      #effective:r-x
default:mask::rwx
default:other::r-x
root@pubbeta2-curve5:/tmp# cd acl
root@pubbeta2-curve5:/tmp/acl# touch file
root@pubbeta2-curve5:/tmp/acl# getfacl -e file
# file: file
# owner: root
# group: root
user::rw-
user:wanghai01:rwx  #effective:rw-
```

```
group::r-x      #effective:r--
mask::rw-
other::r--
```

Table: Types of ACL Entries

Entry type	Text form
Owner	`user::rwx`
Named user	`user:name:rwx`
Owning group	`group::rwx`
Named group	`group:name:rwx`
Mask	`mask::rwx`
Others	`other::rwx`

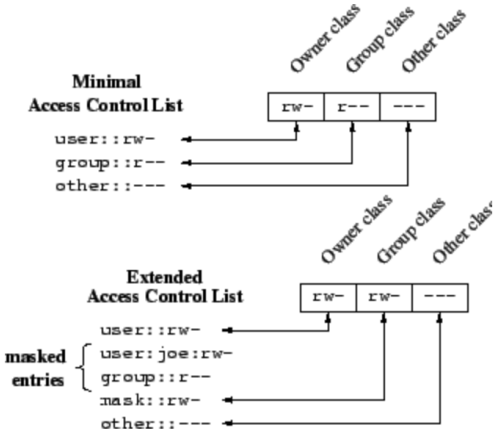


Figure: Mapping between ACL Entries and File Mode Permission Bits

Table: Masking of Permissions

Entry type	Text form	Permissions
Named user	`user:joe:r-x`	`r-x`
Mask	`mask::rw-`	`rw-`
Effective permissions		`r-`

ACL Access Entry保存在哪？

ACL的表示

在Linux 中， ACL 是按照 Posix 标准来实现，其数据结构和 Posix 规定的 ACL 的数据是一致的。其定义在 include/linux/posix_acl.h ，实现在 fs/posix_acl.c 中

```
struct posix_acl_entry {
 short           e_tag;
 unsigned short  e_perm;
 union {
  kuid_t        e_uid;
  kgid_t        e_gid;
 };
};

struct posix_acl {
 refcount_t            a_refcount;
 struct rcu_head        a_rcu;
 unsigned int          a_count;
 struct posix_acl_entry a_entries[0];
};
```

内存中的ACL 是如何与具体的 Inode 相关联

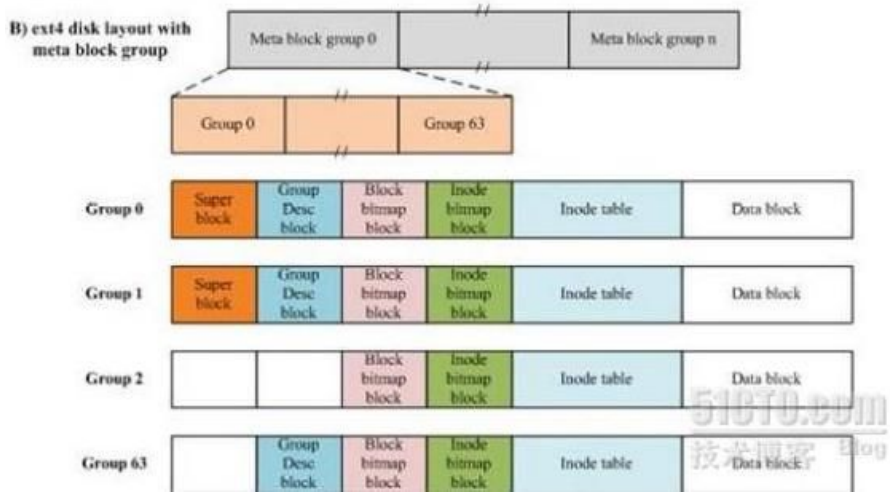
acl 属性是用于访问控制的，对一个文件读写执行都要通过这个 acl 属性来控制。 default_acl 属性是目录特有的 ACL 属性，在此目录中创建的文件和目录都将继承这个 default_acl 属性（对于普通文件来说，该指针为空）。Linux 中使用简单的 Posix_acl_xattr 来对其操作，setfacl 和 getfacl 都是通过 getxattr 和 setxattr 来实现的。

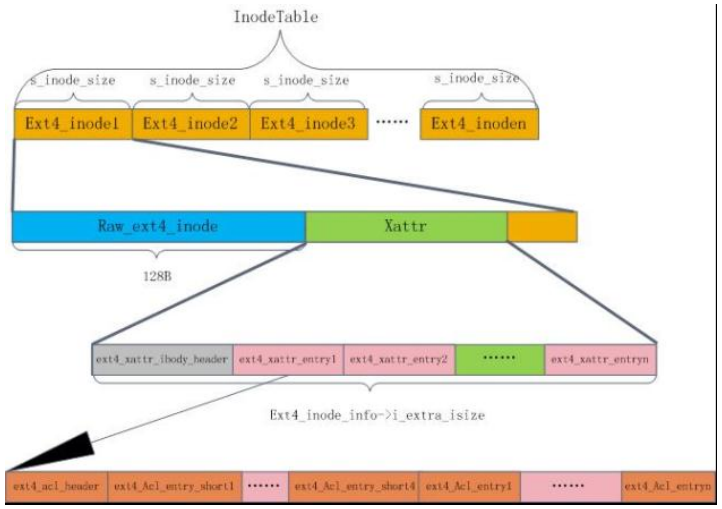
```
// include/linux/fs.h
struct inode {
 ...
 #ifdef CONFIG_FS_POSIX_ACL
  struct posix_acl *i_acl;
  struct posix_acl *i_default_acl;
 #endif
...
}
```

如何存储和获取ACL信息

ACL 是物理文件系统的一个属性，需要永久保存。如何将 ACL 保存在外存中，包括 ACL 在外存中具体存放的位置，以及如何从外存中读取和写入原始 ACL 内容。涉及到 VFS 和具体的物理文件系统，这里以Ext4文件系统为例说明。

在 Linux 操作系统中，如果libattr 功能在内核设置中被打开， ext2 、 ext3 、 ext4 、 JFS 、 ReiserFS 以及 XFS 文件系统都支持扩展属性（简写为xattr, 详见man xattr ）。任何一个普通文件都可能包含有一系列的扩展属性。每一个属性由一个名字以及与之相关联的数据所表示。其中名字必须为一个 字符串 ，并且必须有一个 命名空间
前缀标识符与一个点字符。目前存在有四种命名空间：用户命名空间、信任命名空间、安全命名空间以及系统命名空间。用户命名空间在命名或者内容上没有任何限制。系统命名空间主要被内核用于访问控制表上。目前Linux 的 ACL 存储实现就是基于这种扩展属性的。 Inode Table中保存有若干个 Ext4_inode ，每个 Inode 大小为 ext4_super_block 中指定的 s_inode_size，然而一个 Inode
不一定用到这么多的大小，节点信息只用到 128 个字节的空间。剩下的部分作为扩展文件属性（Xattr），扩展属性内部是由一个扩展属性头和若干个扩展属性实体项构成的。





```
// fs/ext4/xattr.h
struct ext4_xattr_header {
 __le32 h_magic; /* magic number for identification */
 __le32 h_refcount; /* reference count */
 __le32 h_blocks; /* number of disk blocks used */
 __le32 h_hash;  /* hash value of all attributes */
 __le32 h_checksum; /* crc32c(uuid+id+xattrblock) */
    /* id = inum if refcount=1, blknum otherwise */
 __u32 h_reserved[3]; /* zero right now */
};

struct ext4_xattr_entry {
 __u8 e_name_len; /* length of name */
 __u8 e_name_index; /* attribute name index */
 __le16 e_value_offs; /* offset in disk block of value */
 __le32 e_value_inum; /* inode in which the value is stored */
 __le32 e_value_size; /* size of attribute value */
 __le32 e_hash;  /* hash value of name and value */
 char e_name[0]; /* attribute name */
};

// fs/ext4/acl.h
typedef struct {
 __le16  e_tag;
 __le16  e_perm;
 __le32  e_id;
} ext4_acl_entry;

typedef struct {
 __le16  e_tag;
 __le16  e_perm;
} ext4_acl_entry_short;

typedef struct {
 __le32  a_version;
} ext4_acl_header;
```

libfuse FUSE_CAP_POSIX_ACL

◆ FUSE_CAP_POSIX_ACL

#define FUSE_CAP_POSIX_ACL (1 << 19)

Indicates support for POSIX ACLs.

If this feature is enabled, the kernel will cache and have responsibility for enforcing ACLs. ACL will be stored as xattrs and passed to userspace, which is responsible for updating the ACLs in the filesystem, keeping the file mode in sync with the ACL, and ensuring inheritance of default ACLs when new filesystem nodes are created. Note that this requires that the file system is able to parse and interpret the xattr representation of ACLs.

Enabling this feature implicitly turns on the `default_permissions` mount option (even if it was not passed to mount(2)).

This feature is disabled by default.

Definition at line 338 of file fuse_common.h.

CephFs Client's Capbilities

ABILITIES GRANTED BY EACH CAP

While that is how capabilities are granted (and communicated), the important bit is what they actually allow the client to do:

- PIN: this just pins the inode into memory. This is sufficient to allow the client to get to the inode number, as well as other immutable things like major or minor numbers in a device inode, or symlink contents.
- AUTH: this grants the ability to get to the authentication-related metadata. In particular, the owner, group and mode. **Note that doing a full permission check may require getting at ACLs as well, which are stored in xattrs**.
- LINK: the link count of the inode
- XATTR: ability to access or manipulate xattrs. Note that since ACLs are stored in xattrs, it's also sometimes necessary to access them when checking permissions.
- FILE: this is the big one. These allow the client to access and manipulate file data. It also covers certain metadata relating to file data – the size, mtime, atime and ctime, in particular.

Inode权限校验

对Inode访问权限检查（inode_permission()）包括对U/G/O 9bits的校验和对ACL属性的校验（posix_acl_permission()）。

chmod、chown、setfacl、getfacl接口文件系统自己如何实现

chmod、chown 可以通过setattr接口实现
setfacl、getfacl通过setxattr、getxattr接口实现，需要在Inode中增加扩展属性

```
root@pubbeta2-curve5:/tmp# strace getfacl acl.test
execve("/usr/bin/getfacl", ["getfacl", "acl.test"], [/* 25 vars */]) = 0
```

```
...
ioctl(1, TCGETS, {B38400 opost isig icanon echo ...}) = 0
getrlimit(RLIMIT_NOFILE, {rlim_cur=256*1024, rlim_max=256*1024}) = 0
lstat("acl.test", {st_mode=S_IFREG|0674, st_size=6, ...}) = 0
getxattr("acl.test", "system.posix_acl_access",
"\2\0\0\0\1\0\6\0\377\377\377\377\2\0\6\0\3362\0\0\4\0\4\0\377\377\377\377\20\0\7\0\377\377\377\377
\0\4\0\377\377\377\377", 132) = 44
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 3), ...}) = 0
write(1, "# file: acl.test\n", 17# file: acl.test
)      = 17
...

root@pubbeta2-curve5:/tmp# strace setfacl -m u:nbs:rw- acl.test
execve("/usr/bin/setfacl", ["setfacl", "-m", "u:nbs:rw-", "acl.test"], [/* 25 vars */]) = 0
...
lstat("acl.test", {st_mode=S_IFREG|0674, st_size=6, ...}) = 0
getxattr("acl.test", "system.posix_acl_access",
"\2\0\0\0\1\0\6\0\377\377\377\377\2\0\6\0\3362\0\0\4\0\4\0\377\377\377\377\20\0\7\0\377\377\377\377
\0\4\0\377\377\377\377", 132) = 44
setxattr("acl.test", "system.posix_acl_access",
"\2\0\0\0\1\0\6\0\377\377\377\377\2\0\6\0\332\7\0\0\2\0\6\0\3362\0\0\4\0\4\0\377\377\377\377\20\0\6\0\377\377\37
7\377 \0\4\0\377\377\377\377", 52, 0) = 0
...

# cephfs src/client/Inode.h
struct Inode : RefCountedObject {
 ...
 map<string,bufferptr> xattrs
 ...
}

# test setfacl in curvefs
# ./bazel-bin/curvefs/src/client/fuse_client -f -d -o volume=/fs -o allow_other -o default_permissions -o
user=test -o conf=./curvefs/conf/curvefs_client.conf /tmp/fsmount
root@pubbeta1-nostest2:/tmp/fsmount# setfacl -m u:wanghai01:rw- file
setfacl: file: Operation not supported
# mount debug info
...
```

```
unique: 69, opcode: GETXATTR (22), nodeid: 2, insize: 72, pid: 2081159
```

```
    unique: 69, error: -38 (Function not implemented), outsize: 16
unique: 70, opcode: SETXATTR (21), nodeid: 2, insize: 116, pid: 2081159
    unique: 70, error: -38 (Function not implemented), outsize: 16
```

结论：

1：前期可以先不自己实现权限管理，使用‘default_permissions’ 和 ‘allow_other’的mount option（如果是非root用户进行挂载还需要在/etc/fuse.conf中增加配置项‘user_allow_other’）启用内核基于mode的权限控制。

2：新建rootinode mode = 1777（原因是设置STICKY，避免普通用户对非自己所属文件的删除）

3：这样达到的效果除了不支持ACL外与正常本地文件系统权限管理一致（一般情况下使用ACL极少，且从抓取的传媒接口调用发现并未涉及相关接口的调用）。

参考文献：

https://www.huaweicloud.com/articles/0fe3750d1a5352b42911fdb96c6a8a47.html

https://www.jianshu.com/p/eb8b2a679537

https://zhuanlan.zhihu.com/p/44267768

https://sourceforge.net/p/fuse/mailman/fuse-devel/thread/CAGRbiNS5YL5vjV_XNhv3RL-ub3VbwNTwmbHGMOgcpvnnnUpmng%40mail.gmail.com/

https://unix.stackexchange.com/questions/325473/in-fuse-how-do-i-get-the-information-about-the-user-and-the-process-that-is-try

https://wiki.gentoo.org/wiki/Filesystem/Access_Control_List_Guide

https://www.usenix.org/legacy/publications/library/proceedings/usenix03/tech/freenix03/full_papers/gruenbacher/gruenbacher_html/main.html

https://blog.csdn.net/qq_34805255/article/details/89348483

https://blog.51cto.com/denglz/1341873

https://blog.csdn.net/kyosanma/article/details/5722271

https://blog.51cto.com/liujingyu/1980457

https://unix.stackexchange.com/questions/443318/file-permissions-kernel-or-file-system