Idmapped Mounts

Per-Mount Ownership Changes
Ownership

- uids and gids express ownership
- VFS uses them for permission checking
- persisted to disk for FS_REQUIRES_DEV filesystems
Ownership & struct inode

- `i_uid_read()`
  - read ownership information from struct inode
  - calls `from_kuid()` to translate kuids to raw uids

- `i_uid_write()`
  - write ownership information to struct inode
  - calls `make_kuid()` to translate raw uids into kuids
ID mappings

- translation of range of ids into another or same range of ids
- notational convention in this talk: u:k:r
  u == userspace-id / userspace-idmapset
  k == kernel-id / kernel-idmapset
  r == range
- associated with struct user_namespace
- init_user_ns has identity idmapping: u0:k0:r4294967295
ID mappings

- make_kuid(u0:k10000:r10000, u1000)
  What does u1000 map down to?
  \[ \text{id} - \text{u} + \text{k} = \text{n} \]
  \[ u1000 - u0 + k10000 = k11000 \]

- from_kuid(u0:k10000:r10000, k11000)
  What does k11000 map up to?
  \[ \text{id} - \text{k} + \text{u} = \text{n} \]
  \[ k11000 - k10000 + u0 = u1000 \]
Ownership: Disk to VFS

- file owned on disk by raw uid 1000

  - fs mounted in init_user_ns
    \[ i\_uid\_write(u0:k0:r4294967295, u1000) = k1000 \]

  - fs mounted with idmapping
    \[ i\_uid\_write(u0:k10000:r10000, u1000) = k11000 \]

// Examples
xfs_inode_to_disk(), ext4_do_update_inode(), fill_inode_item() // btrfs
Ownership: VFS to Disk

- file owned on disk by raw uid 1000

- fs mounted in init_user_ns
  \[
  \text{i\_uid\_write}(u0:k0:r4294967295, u1000) = k1000 \\
  \text{i\_uid\_read}(u0:k0:r4294967295, k1000) = u1000
  \]

- fs mounted with idmapping
  \[
  \text{i\_uid\_write}(u0:k10000:r10000, u1000) = k1100 \\
  \text{i\_uid\_read}(u0:k10000:r10000, k1100) = u1000
  \]

// Examples
xfs_inode_from_disk(), __ext4_iget(), btrfs_read_locked_inode()
Creating New Files (Userspace to/from VFS)

Translate between two ID-mappings via the kernel idmapset:

1. Map caller's userspace ids down into kernel ids in the caller's idmapping.
   // current_fsuid()

2. Verify caller's kernel ids can be mapped up to userspace ids in filesystem's idmapping.
   // fsuidgid_has_mapping()
Crossmapping

vfs_mkdir()

- caller id: u1000
  caller id mapping: u0:k10000:r10000
  fs id mapping: u20000:k10000:r10000

/* fsuid_gid_has_mapping() */
make_kuid(u0:k10000:r10000, u1000) = k11000 // current_fsuid()
from_kuid(u20000:k10000:r10000, k11000) = u21000
Filesystem-wide Idmappings

- alter ownership filesystem-wide
- relevant idmapping is represented in the filesystem's superblock
- determined at mount time
Selected Modern Filesystem Use-Cases
Selected Modern Filesystem Use-Cases

- Portable Home Directories
  - make login uid and gid random
  - take home directory between computers
Selected Modern Filesystem Use-Cases

- Portable Home Directories
  - make login uid and gid random
  - take home directory between computers

- Containers
  - rootfs
  - data sharing host <> container
  - data sharing container <> container
Idmapped Mounts

File ownership should be changeable on a per-mount basis instead of a filesystem wide basis.

Idmapped mounts make it possible to change ownership in a temporary and localized way:

- ownership changes are restricted to a specific mount
- ownership changes are tied to the lifetime of a mount
Idmapped Mounts

Idmapping functions were added that translate between idmappings:

- **i_uid_into_mnt()**
  - translate filesystems kernel ids into kernel ids in the mount's idmapping
    
    /* Map filesystem's kernel id up into a userspace id in the filesystem's idmapping. */
    from_kuid(filesystem-idmapping, kid) = uid

    /* Map filesystem's userspace id down into a kernel id in the mount's idmapping. */
    make_kuid(mount, uid) = kuid

- **mapped_fsuid()**
  - translate caller's kernel ids into kernel ids in the filesystem's idmapping by remapping the caller's kernel ids using the mount's idmapping
    
    /* Map the caller's kernel id up into a userspace id in the mount's idmapping. */
    from_kuid(mount-idmapping, kid) = uid

    /* Map the mount's userspace id down into a kernel id in the filesystem's idmapping. */
    make_kuid(filesystem-idmapping, uid) = kuid

*In our documentation I call it "remapping algorithm" because it undoes an existing idmapping and remaps it according to the mount's idmapping.
Idmapped Mounts: Portable Home Directories

vfs_mkdir()

- caller id: u1001
caller idmapping: u0:k0:r4294967295
filesystem idmapping: u0:k0:r4294967295
mount idmapping: u1000:k1001:r1

1. Map the caller's userspace ids into kernel ids in the caller's idmapping
make_kuid(u0:k0:r4294967295, u1001) = k1001 // current_fsuid()

2. Translate caller's kernel id into a kernel id in the filesystem's idmapping
mapped_fsuid(k1001)
   /* Map the kernel id up into a userspace id in the mount's idmapping. */
   from_kuid(u1000:k1001:r1, k1001) = u1000

   /* Map the userspace id down into a kernel id in the filesystem's idmapping. */
   make_kuid(u0:k0:r4294967295, u1000) = k1000

3. Verify that the caller's kernel ids can be mapped to userspace ids in the filesystem's idmapping
from_kuid(u0:k0:r4294967295, k1000) = u1000 // VFS to Disk

So ultimately the file will be created with raw uid 1000 on disk.
Idmapped Mounts: Portable Home Directories

vfs_getattr() + cp_statx()

- caller id: u1001
caller idmapping: u0:k0:r4294967295
filesystem idmapping: u0:k0:r4294967295
mount idmapping: u1000:k1001:r1

1. Map the userspace id on disk down into a kernel id in the filesystem's idmapping
make_kuid(u0:k0:r4294967295, u1000) = k1000 // i_uid_write()

2. Translate the kernel id into a kernel id in the mount's idmapping
i_uid_into_mnt(k1000)
    /* Map the kernel id up into a userspace id in the filesystem's idmapping. */
    from_kuid(u0:k0:r4294967295, k1000) = u1000

    /* Map the userspace id down into a kernel id in the mount's idmapping. */
    make_kuid(u1000:k1001:r1, u1000) = k1001

3. Map the kernel id up into a userspace id in the caller's idmapping
from_kuid(u0:k0:r4294967295, k1001) = u1001 // VFS to Userspace

So ultimately the caller will be reported that the file belongs to raw uid 1001 which is the caller's userspace id in our example.
Idmapped Mounts

```c
struct mount_attr *attr = &struct mount_attr{};
int fd_tree = open_tree(-EBADF, source,
    OPEN_TREE_CLONE | OPEN_TREE_CLOEXEC |
    AT_EMPTY_PATH | AT_RECURSIVE);

attr->attr_set  |= MOUNT_ATTR_IDMAP;
attr->userns_fd  = fd_userns;

mount_setattr(fd_tree, "", AT_EMPTY_PATH | AT_RECURSIVE,
    attr, sizeof(struct mount_attr));
```
Filesystem Support & Adoption

- fat, ext4, xfs, btrfs, ksmbd and more to come
- Already widely adopted in userspace with and a variety of patchsets out there LXD, containerd, systemd, …
- not a container feature!