Microsoft

Idmapped Mounts

per vfsmount ownership changes

VFS Ownership

- uids and gids express ownership
- VFS uses them for permission checking (DAC, POSIX ACLs, fscaps)
- 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

Idmappings

- 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`

Idmappings

- `make_kuid(u0:k10000:r10000, u1000)`
  What does `u1000` map down to?
  ```
  id     - u  + k       = n
  u1000 - u0 + k10000 = k11000
  ```

- `from_kuid(u0:k10000:r10000, k11000)`
  What does `k11000` map up to?
  ```
  id      - k       + u      = 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`

```
i_uid_write(u0:k0:r4294967295, u1000) = k1000
i_uid_read(u0:k0:r4294967295,  k1000) = u1000
```

  - fs mounted with idmapping

```
i_uid_write(u0:k10000:r10000, u1000) = k1100
i_uid_read(u0:k10000:r10000, k11000) = 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 idmapping: u0:k10000:r10000
  fs idmapping: u20000:k10000:r10000

  /* fsuidgid_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

Filesystem Use-Cases

home directories, containers, and service isolation

Portable Home Directories

- aims to make it trivial to transport home directories between different machines
- all files are owned by uid and gid `nobody/65534` on-disk
- assign first free uid and gid in the range `60001...60513` at login
- recursively `chown()` to login uid and gid in case login uid and gid has changed :/

Containers

- using unprivileged containers makes filesystem interactions difficult
- on-disk ownership of the container's rootfs needs to correspond to container's idmapping
- cannot share layers between unprivileged containers with different idmappings or between privileged and unprivileged containers
- recursive ownership changes waste space and make starting containers expensive

Idmapped Mounts

temporary and localized ownership changes

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

Remapping Helpers

- `i_uid_into_mnt()`
  - Remap inode kernel ids from the filesystem into the mount 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()`
  - Remap caller kernel fsids according to the mount 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
```

Filesystem Use-Cases revisited

home directories, containers, and service isolation with idmapped mounts

Portable Home Directories

```
vfs_mkdir()
```

- ```
  caller id:             u60001
  caller idmapping:      u0:k0:r4294967295
  filesystem idmapping: u0:k0:r4294967295
  mount idmapping:       u65534:k60001:r1 /* Of course, systemd will map way more IDs than that */
  ```

- Map the caller's userspace ids into kernel ids in the caller's idmapping
  ```
  make_kuid(u0:k0:r4294967295, u60001) = k60001 /* current_fsuid() */
  ```

- Translate caller's kernel id into a kernel id in the filesystem's idmapping
  ```
  mapped_fsuid(k60001)
          /* Map the kernel id up into a userspace id in the mount's idmapping. */
          from_kuid(u65534:k60001:r1, k60001) = u65534

          /* Map the userspace id down into a kernel id in the filesystem's idmapping. */
          make_kuid(u0:k0:r4294967295, u65534) = k65534
  ```

- Verify that the caller's kernel ids can be mapped to userspace ids in the filesystem's idmapping
  ```
  from_kuid(u0:k0:r4294967295, k65534) = u65534 /* VFS to Disk */
  ```
- So ultimately the file will be created with raw uid `65534` on disk.
  ```

Portable Home Directories

```
vfs_getattr() + cp_statx()
```

- ```
  caller id:            u60001
  caller idmapping:     u0:k0:r4294967295
  filesystem idmapping: u0:k0:r4294967295
  mount idmapping:      u65534:k60001:r1 /* Of course, systemd will map way more IDs than that */
  ```

- Map the userspace id on disk down into a kernel id in the filesystem's idmapping
  ```
  make_kuid(u0:k0:r4294967295, u65534) = k65534 /* i_uid_write() */
  ```

- Translate the kernel id into a kernel id in the mount's idmapping
  ```
  i_uid_into_mnt(k65534)
          /* Map the kernel id up into a userspace id in the filesystem's idmapping. */
          from_kuid(u0:k0:r4294967295, k65534) = u65534

          /* Map the userspace id down into a kernel id in the mounts's idmapping. */
          make_kuid(u65534:k60001:r1, u65534) = k60001
  ```

- Map the kernel id up into a userspace id in the caller's idmapping
  ```
  from_kuid(u0:k0:r4294967295, k60001) = u60001 /* VFS to Userspace */
  ```
- So ultimately the caller will be reported that the file belongs to raw uid `60001` which is the caller's userspace id in our example.
  ```

UAPI

How to create idmapped mounts

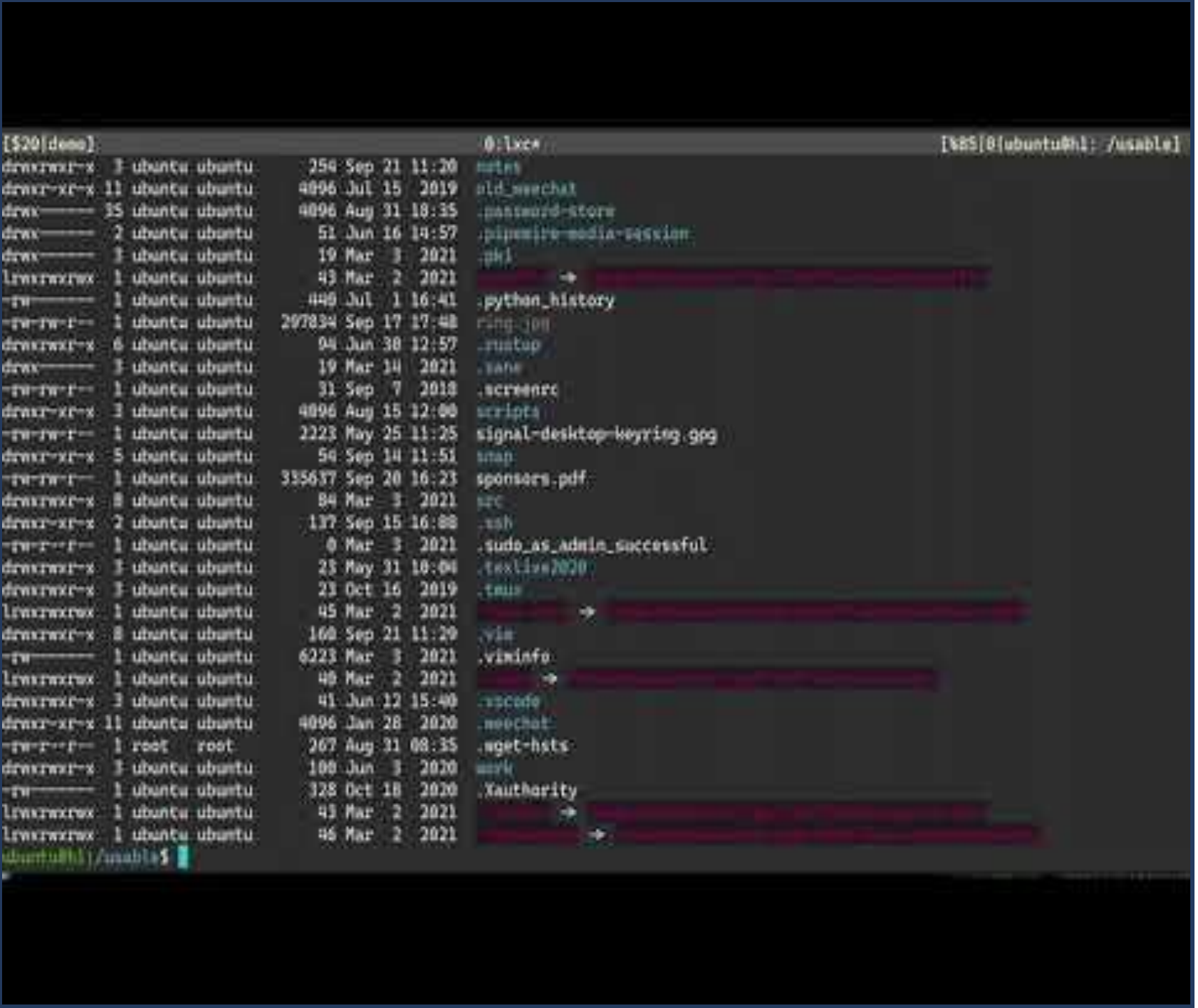
mount_setattr()

```
struct mount_attr *attr = &(struct mount_attr){};

/* create private, detached (not reachable anywhere in the filesystem) mount */
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));
```



Demo

A few simple examples

Support & adoption

Filesystem support and userspace adoption

Filesystem support

v5.12

- ext4
- fat (msdos, vfat)
- xfs

v5.15

- btrfs
- ntfs3

v5.18

- f2fs

v5.19

- erofs
- overlayfs (mounted on top of idmapped lower- and upper layers)

Userspace support

- systemd
- containerd
- crun
- runC
- LXC
- LXD
- Podman
- Open Container Initiative (OCI) runtime spec
- **mount(2)** in util-linux

Microsoft

Thank you