Modernizing local storage management for systemd services

Andres Beltran

Linux Systems Group

Azure Boost, Microsoft

Agenda

1. **systemd overview**
    1. Dynamic Users in systemd
    2. Local storage directories
2. **ID-mapped mounts**
    1. Use cases
    2. ID mapping for systemd storage directories
    3. ID mapping mechanism
    4. Demo
3. **Disk Quotas**
    1. Background
    2. Disk quotas for systemd storage directories
    3. Quota Mechanism
    4. Demo

What is systemd?

- **PID 1:** init system and service manager used by almost all major Linux distributions
  - Fedora, CentOS, RedHat, Ubuntu, Debian, Arch, etc.
- Starts/monitors **userspace services** and manages their dependencies
- Provides **parallelization** capabilities, **on-demand** activation of services, simple network configuration, and cgroup-based **resource control**
- Exposes different configuration files: .service, .socket, .timer, .mount, and .target
- **Primary tools**: *systemctl* (control/status) and *journalctl* (centralized logs)

Dynamic Users with systemd

- Enabled by setting ***DynamicUser=yes*** in a service file

- A UNIX dynamic user/group ID pair is **allocated when the service starts** and **released when it terminates**
  - systemd picks the first available UID from a pre-defined range (61184–65519)
  - glibc nss-systemd module used to leave /etc/passwd untouched

- **Benefit:** running a privilege-separated service leaves no artifacts in the system
  - A system user is allocated but it is discarded automatically in a safe and secure way after use

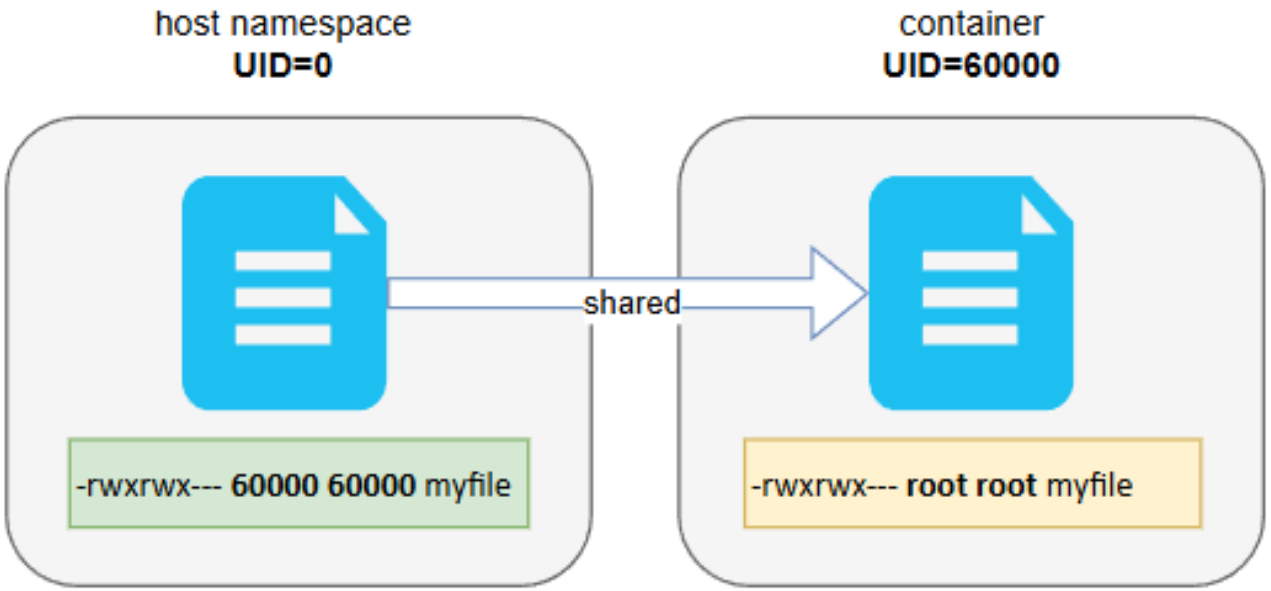
```
[Service]
ExecStart=-/usr/bin/myservicebinary
DynamicUser=yes
```

systemd storage directories

- Help define where systemd services **store their data**
  - Data from persistent directories survive from one service invocation to the next
  - Namespaced bind mounts are created for storage directories
- Avoid **sticky file ownership problem** when combined with *DynamicUser=yes*
  - Create boundary directories: host directories made inaccessible to unprivileged users
  - Ensures that access cannot be gained through dynamic **UID recycling**
- Two important **features** have been implemented to enhance storage directories:
  - ID-mapped mounts
  - Disk quotas

Directory	Path	Boundary directories
StateDirectory=	/var/lib	/var/lib/private
CacheDirectory=	/var/cache	/var/cache/private
LogsDirectory=	/var/log	/var/log/private
RuntimeDirectory=	/run	-

ID-mapped mounts

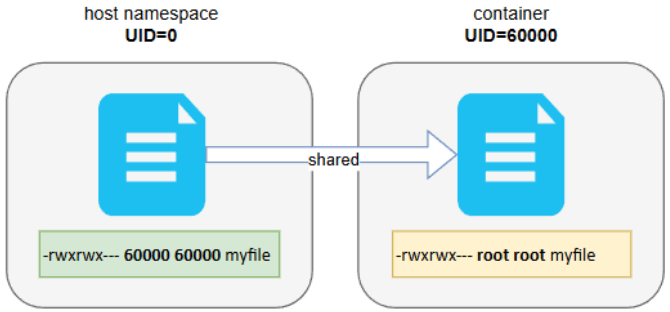


ID mapping for mounted filesystems

- There are several use cases for making file ownership **relative to the environment** a given process is running in

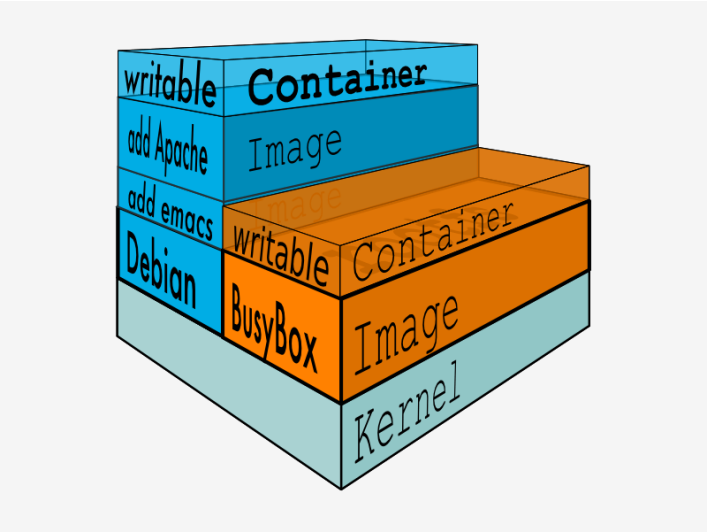
- In many instances, UID/GIDs for files on disk **do not match the current user** (or process) of those files
  - ID-mapped mounts provide a way to resolve that problem, without changing the files on disk

- ID-mapping is a filesystem feature that allows a mount namespace to show a **different UID/GID than what is stored on a file**
  - Available in Linux 5.12+
  - Offers per-mount mapping of UIDs and GIDs



ID mapping use cases: containers

- Container runtime systems may want to provide a **common root image** to each container
  - User namespaces may be used to ensure that each container runs with non-privileged IDs on the host system
  - However, containers should still be able to access their root images with **root privileges**
  - Mounting that image with ID mapping enables **non-privileged containers** to have **privileged UIDs**

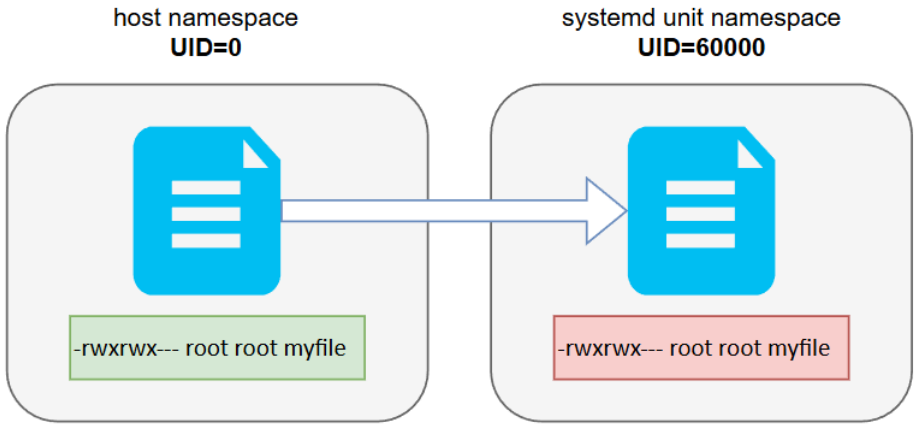


- ID-mapping makes it easier to **share filesystems** between containers regardless of the UIDs used within each container

ID mapping use cases: DAC permissions with file sharing

Without ID-mapping, **world-writeable permissions** may be needed for shared files:

- Consider a running systemd service that defines a storage directory combined with *DynamicUsers=yes*
- Next, a file owned by *root* is created in the host under the shared storage directory with *770* permissions
- The unprivileged service **will not be able to R/W** the file since it is still owned by *root* within its mount namespace
- For the systemd unit to access the shared file, it must be created with **world-writable permissions**, which can be a DAC **security concern**



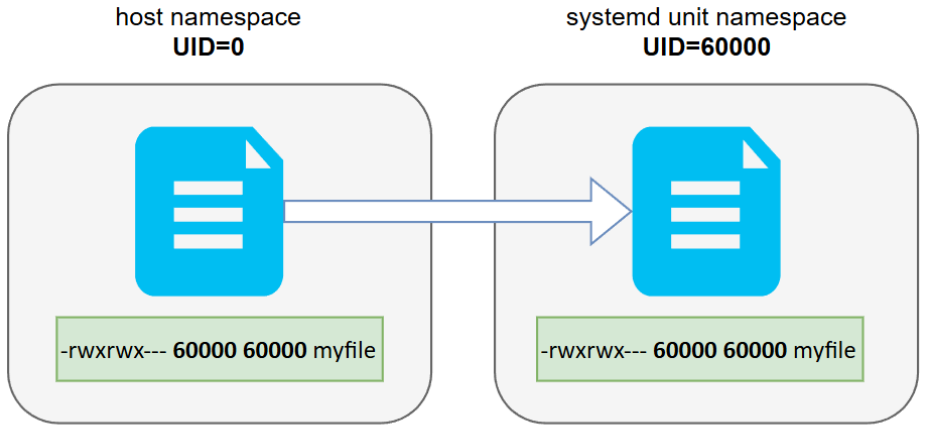
ID mapping use cases: avoiding recursive chown()

Another solution is to **change the ownership** of the shared file so that the unprivileged service can access it

- Without ID-mapped mounts, these cases are handled with **recursive chown()** calls

Example: systemd-homed

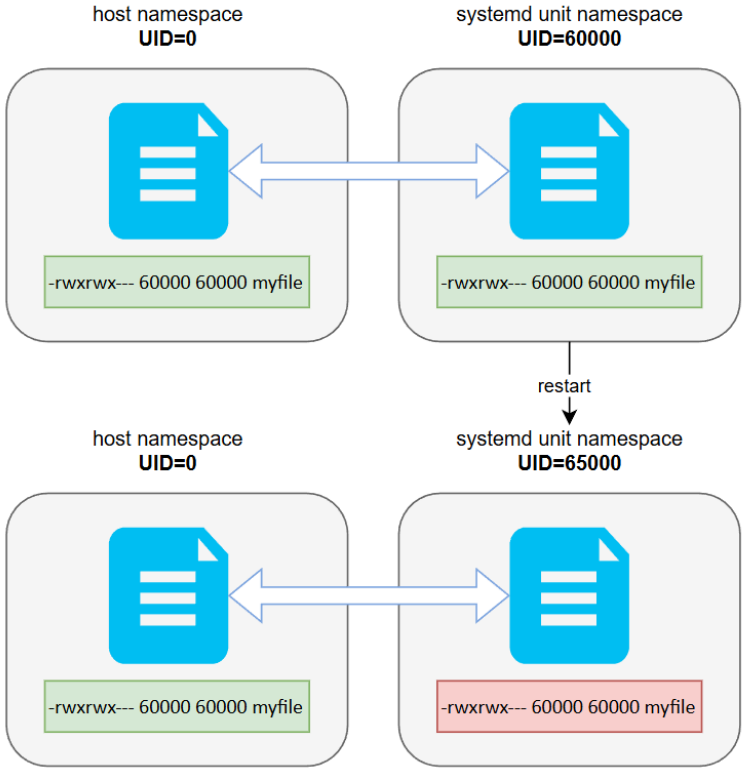
- Must chown **entire home directories** when the user ID does not match the ownership of its home directory
  - Slow and inefficient
  - ID mapping fixes this: files in the home directory are mapped from a **fixed UID** to the **user ID**
  - Extend this approach to **systemd storage directories**



ID mapping use cases: ephemeral UIDs

systemd uses **ephemeral UIDs** when *DynamicUsers=yes*

- Suppose a systemd service runs with UID *'x'* and owns a shared file on a bind-mounted directory with *770* permissions
- Next, the service gets **restarted**
- A new UID *'y'* could be assigned to the service. The file permissions are still the same
  - The service with UID *'y'* will **no longer have access** to its file
  - As shown before, shared files would have to be created with world-writable permissions or chowned() recursively
  - ID-mapped mounts take care of **systemd service restarts**



ID mapping for systemd storage directories

- When ***DynamicUser=yes*** is combined with a storage directory, and ID-mapped mounts are available on the referenced path:
  - ID mapping is defined from the **"nobody" user (65534)** to the **dynamic UID** for each mounted directory
  - Available in systemd **v257+**

Security considerations:

- *"nobody"* user is the fixed UID on the host to minimize the possibility of a root exploit
  - Using *"root"* instead of *"nobody"* would mean that unprivileged users get access to root-owned inodes
- Turn off **SUID** on the ID-mapped mount and mount it as ***noexec***

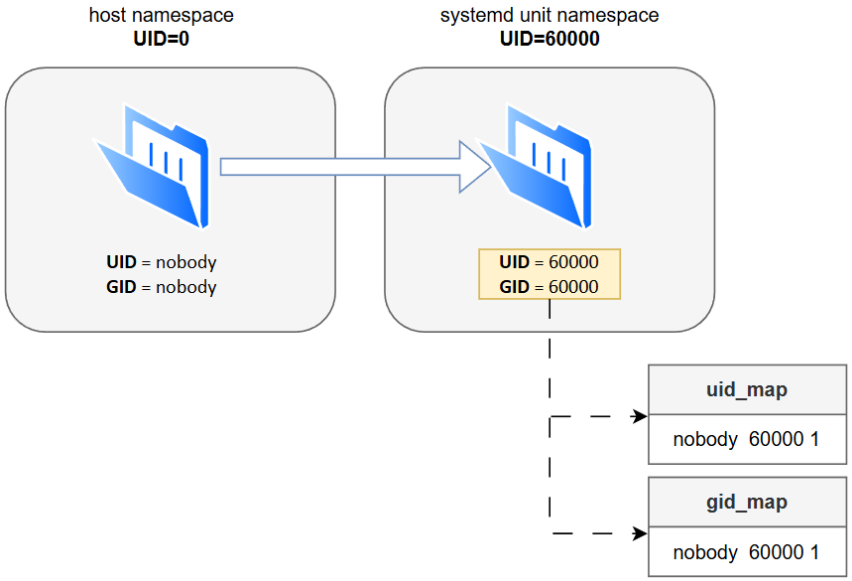
Backward compatibility:

- Continue doing *chown()* for pre-existing storage directories

ID mapping mechanism

Setting up ID-mapped mounts involves the creation of **user namespaces** to contain the ID-mapping tables

1. Create a new user namespace with the UID/GID mapping **for each storage directory mount**
2. Establish the ID mapping through the new user namespace by writing to the ***uid_map*** and ***gid_map*** files
3. Associate the user namespace with the mount via the ***mount_setattr*** **syscall:**
   - The *attr_set* and *attr_clr* fields of the *mount_attr* struct describe the attributes to be set and cleared
   - systemd adds ***MOUNT_ATTR_IDMAP*** to *attr_set* and sets *userns_fd* to the file descriptor of the created user namespace



Demo: StateDirectory with ID mapping

Disk Quotas


Disk quota exceeded'
quota -s
Disk quotas for user user (uid 1000):
Filesystem space quota limit Grace files hard
/dev/sda1 200 MB 500MB 500MB – 51 % 1000
/dev/sda1 20% B 500 100 1000 1000

Background

- Disk quota is a mechanism **for limiting the amount of disk space** that a user or group can use on a filesystem
  - Without limits, a user could fill up the machine's disk and cause problems for other users
- Filesystems that support disk quotas: ext3/ext4, XFS, and Btrfs

Types of Disk Quotas

- *User Quotas:* apply limits to individual **users**
- *Group Quotas:* apply limits to **groups**, in systems where resources are shared among users
- *Project quotas:* directory **tree quota**, where limits are applied to a specified directory and all the files/subdirectories below it. Only supported in **XFS and ext4**

Disk quotas for systemd storage directories

- Available for filesystems that **support project quotas** (ext4 or XFS)
- Different systemd services can enforce distinct limits for their storage directories
  - Available in systemd **v258+**
- Quotas can be defined for **disk blocks** and, optionally, **number of inodes**
  - Hard limits: limit cannot be exceeded
  - Soft limits: limit can be exceeded, but warnings are issued
- Requirements:
  - Enable **prjquota** on the filesystem: *tune2fs -Q prjquota*
  - Quotas must also be turned on with the ***quotaon*** utility

Disk quota enforcement and accounting

- **StateDirectoryQuota=, CacheDirectoryQuota=, and LogsDirectoryQuota=**
  - Why not RuntimeDirectory? tmpfs does not support project IDs yet
  - If an absolute size limit is specified, only the block quota is set
  - If a percentage value is specified, same percent quota to both blocks and inodes is set
  - Only hard limits for blocks/inodes are set

- **StateDirectoryAccounting=, CacheDirectoryAccounting=, and LogsDirectoryAccounting=**
  - Tracks the unit's disk usage without enforcing limits like their Quota counterparts
  - Usage is shown with *systemctl status <unit_name>*

Quota mechanisms

Project IDs

- Unsigned, 32-bit integers. A **pre-defined range** is reserved in systemd for project IDs
- Unique per-service and per-directory-type
  - Each file and directory belonging to the same service and storage directory type (State, Cache, Logs) will have the same project ID
- Set **recursively** using the *fsx_projid* extended attribute via ***ioctl()***
- Released if neither enforcement nor accounting is needed anymore

Quotactl

- Hard limits are set for a given project ID using the *dqblk* struct fields, which is passed to the ***quotactl_fd()*** **syscall**

```
struct dqblk {      /* Definition since Linux 2.4.22 */
    uint64_t dqb_bhardlimit;  /* Absolute limit on disk
                                 quota blocks alloc */
    uint64_t dqb_bsoftlimit;  /* Preferred limit on
                                 disk quota blocks */
    uint64_t dqb_curspace;    /* Current occupied space
                                 (in bytes) */
    uint64_t dqb_ihardlimit;  /* Maximum number of
                                 allocated inodes */
    uint64_t dqb_isoftlimit;  /* Preferred inode limit */
    uint64_t dqb_curinodes;   /* Current number of
                                 allocated inodes */
    uint64_t dqb_btime;       /* Time limit for excessive
                                 disk use */
    uint64_t dqb_itime;       /* Time limit for excessive
                                 files */
    uint32_t dqb_valid;       /* Bit mask of QIF_*
                                 constants */
};
```

Demo: Defining Quotas for StateDirectory

Conclusions

- **ID-mapped mounts** provide a way for systemd storage directories to avoid recursive chown() of entire directories upon unit start/restarts, which can be a very slow and expensive operation
  - This applies if the unit has *DynamicUser* enabled

- **Project quotas** are used to enforce/track disk limits for systemd services. Different quotas can be set for different storage directory types
  - Currently available for the ext-4 and XFS filesystems

References

- ID-mapping patch: Add support for id-mapped mounts to Exec directories (#34078) · systemd/systemd@c7e818f · GitHub
- Disk Quotas patch: core: add quota support for State, Cache, and Log exec directories (#… · systemd/systemd@3ef7918 · GitHub
- Dynamic Users with systemd
- systemd storage settings: systemd.exec
- ID-mapped mounts [LWN.net]
- ID mapping for mounted filesystems [LWN.net]
- ext4(5) - Linux manual page
- xfs_quota(8) - Linux manual page
- quotactl(2) - Linux manual page