WIKIPEDIA

ตัวระบุผู้ใช้

Unix เหมือนระบบปฏิบัติการระบุผู้ใช้โดยค่าที่เรียกว่า**ตัวระบุผู้ใช้**มักจะยากที่จะเป็น**ID ผู้ใช้**หรือโพสต์ UID ร่วมกับตัวระบุกลุ่ม (GID) และเกณฑ์การควบคุมการเข้าถึงอื่นๆ ใช้เพื่อกำหนดว่าผู้ใช้สามารถเข้าถึงทรัพยากรระบบใด ไฟล์รหัสผ่านแผนที่ต้นฉบับเดิมชื่อผู้ใช้เพื่อ UIDs UIDs ถูกเก็บไว้ในinodesของยูนิกซ์ ระบบแฟ้มวิ่งกระบวนการ , tarจดหมายเหตุและตอนนี้ล้าสมัยข้อมูลการให้บริการเครือข่าย ในสภาพแวดล้อมที่สอดคล้องกับPOSIXคำสั่งบรรทัดคำสั่ง `id` ให้ UID ของผู้ใช้ปัจจุบัน รวมถึงข้อมูลเพิ่มเติม เช่น ชื่อผู้ใช้ กลุ่มผู้ใช้หลัก และตัวระบุกลุ่ม (GID)

คุณสมบัติกระบวนการ

The POSIX standard introduced three different UID fields into the process descriptor table, to allow privileged processes to take on different roles dynamically:

Effective user ID

The effective UID (`euid`) of a process is used for most access checks. It is also used as the owner for files created by that process. The effective GID (`egid`) of a process also affects access control and may also affect file creation, depending on the semantics of the specific kernel implementation in use and possibly the mount options used. According to BSD Unix semantics, the group ownership given to a newly created file is unconditionally inherited from the group ownership of the directory in which it is created. According to AT&T UNIX System V semantics (also adopted by Linux variants), a newly created file is normally given the group ownership specified by the `egid` of the process that creates the file. Most filesystems implement a method to select whether BSD or AT&T semantics should be used regarding group ownership of a newly created file; BSD semantics are selected for specific directories when the S_ISGID (s-gid) permission is set.[1]

File system user ID

Linux also has a file system user ID (`fsuid`) which is used explicitly for access control to the file system. It matches the `euid` unless explicitly set otherwise. It may be root's user ID only if `ruid`, `suid`, or `euid` is root. Whenever the `euid` is changed, the change is propagated to the `fsuid`.

The intent of `fsuid` is to permit programs (e.g., the NFS server) to limit themselves to the file system rights of some given `uid` without giving that `uid` permission to send them signals. Since kernel 2.0, the existence of `fsuid` is no longer necessary because Linux adheres to SUSv3 rules for sending signals, but `fsuid` remains for compatibility reasons.[2]

Saved user ID

The saved user ID (`suid`) is used when a program running with elevated privileges needs to do some unprivileged work temporarily; changing `euid` from a privileged value (typically `0`) to some unprivileged value (anything other than the privileged value) causes the privileged value to be stored in `suid`.[3] Later, a program's `euid` can be set back to the value stored in `suid`, so that elevated privileges can be restored; an unprivileged process may set its `euid` to one of only three values: the value of `ruid`, the value of `suid`, or the value of `euid`.

Real user ID

The real UID (`ruid`) and real GID (`rgid`) identify the real owner of the process and affect the permissions for sending signals. A process without superuser privileges may signal another process only if the sender's `ruid` or `euid` matches receiver's `ruid` or `suid`. Because a child process inherits its credentials from its parent, a child and parent may signal each other.

อนุสัญญา

Type

POSIX requires the UID to be an integer type. Most Unix-like operating systems represent the UID as an unsigned integer. The size of UID values varies amongst different systems; some

UNIX OS's used 15-bit values, allowing values up to 32767, while others such as Linux (before version 2.4) supported 16-bit UIDs, making 65536 unique IDs possible. The majority of modern Unix-like systems (e.g., Solaris-2.0 in 1990, Linux 2.4 in 2001) have switched to 32-bit UIDs, allowing 4,294,967,296 (2^{32}) unique IDs.

Reserved ranges

The Linux Standard Base Core Specification specifies that UID values in the range 0 to 99 should be statically allocated by the system, and shall not be created by applications, while UIDs from 100 to 499 should be reserved for dynamic allocation by system administrators and post install scripts.[4]

Debian Linux not only reserves the range 100–999 for dynamically allocated system users and groups, but also centrally and statically allocates users and groups in the range 60000-64999 and further reserves the range 65000–65533.[5]

Systemd defines a number of special UID ranges, including[6]

- 60001-60513: UIDs for home directories managed by systemd-homed
- 61184-65519 (0xef00-0xffef): UIDs for dynamic users

On FreeBSD, porters who need a UID for their package can pick a free one from the range 50 to 999 and then register the static allocation.[7][8]

Some POSIX systems allocate UIDs for new users starting from 500 (macOS, Red Hat Enterprise Linux till version 6), others start at 1000 (Red Hat Enterprise Linux since version 7,[9] openSUSE, Debian[5]). On many Linux systems, these ranges are specified in `/etc/login.defs`, for `useradd` and similar tools.

Central UID allocations in enterprise networks (e.g., via LDAP and NFS servers) may limit themselves to using only UID numbers well above 1000, and outside the range 60000–65535, to avoid potential conflicts with UIDs locally allocated on client computers.

OS-level virtualization can remap user identifiers, e.g. using Linux namespaces, and therefore need to allocate ranges into which remapped UIDs and GIDs are mapped:

- snapd maps UIDs and GIDs into the range 524288-589823 (https://forum.snapcraft.io/t/system-usernames/13386) (0x80000-0x8ffff)
- systemd-nspawn automatic allocates of per-container UID ranges uses the range 524288-1879048191 (0x80000-0x6fffffff)[6]

The systemd authors recommend that OS-level virtualization systems should allocate 65536 (2^{16}) UIDs per container, and map them by adding an integer multiple of 2^{16}.[6]

Special values

- 0: The superuser normally has a UID of zero (0).[10]
- −1: The value `(uid_t) -1` is reserved by POSIX to identify an omitted argument.[11]
- 65535: This value is still avoided because it was the API error return value when uid_t was 16 bits.
- Nobody: Historically, the user "nobody" was assigned UID `-2` by several operating systems, although other values such as $2^{15}-1$ = 32,767 are also in use, such as by OpenBSD.[12] For compatibility between 16-bit and 32-bit UIDs, many Linux distributions now set it to be $2^{16}-2$ = 65,534; the Linux kernel defaults to returning this value when a 32-bit UID does not fit into the return value of the 16-bit system calls.[13] Fedora Linux assigns the last UID of the range statically allocated for system use (0–99) to nobody: 99, and calls 65534 instead `nfsnobody`.

ทางเลือก

NFSv4 was intended to help avoid numeric identifier collisions by identifying users (and groups) in protocol packets using textual “user@domain” names rather than integer numbers. However, as long as operating-system kernels and local file systems continue to use integer user identifiers, this comes at the expense of additional translation steps (using idmap daemon processes), which can introduce additional failure points if local UID mapping mechanisms or databases get configured incorrectly, lost, or out of sync. The “@domain” part of the user name could be used to indicate which authority allocated a particular name, for example in form of

- a Kerberos realm name
- an Active Directory domain name
- the name of an operating-system vendor (for distribution-specific allocations)
- the name of a computer (for device-specific allocations)

But in practice many existing implementations only allow setting the NFSv4 domain to a fixed value, thereby rendering it useless.

ดูสิ่งนี้ด้วย

- setuid
- Sticky bit
- Group identifier
- Process identifier
- File system permissions
- Open (system call)
- Mount (Unix)
- FAT access rights
- Security Identifier (SID) – the Windows NT equivalent

อ้างอิง

1. `chmod(1) (https://docs.oracle.com/cd/E26505_01/html/816-5165/chmod-1.html)` – *Solaris 10 User Commands Reference Manual*
2. *Kerrisk, Michael. The Linux Programming Interface. No Starch Press, 2010, p. 171.*
3. *"Setuid Demystified" (http://www.cs.berkeley.edu/~daw/papers/setuid-usenix02.pdf) (PDF). Cs.berkeley.edu. Retrieved 2016-09-24.*
4. *"9.3. UID Ranges" (https://refspecs.linuxfoundation.org/LSB_3.0.0/LSB-Core-generic/LSB-Core-generic/uidrange.html) . Refspecs.linuxfoundation.org. Retrieved 2016-09-24.*
5. *"Debian Policy Manual – Section 9.2.2: UID and GID classes" (https://www.debian.org/doc/debian-policy/ch-opersys.html#uid-and-gid-classes) . Debian.org. 2019-07-18. Retrieved 2019-07-26.*
6. *"Users, groups, UIDs and GIDs on systemd systems" (https://github.com/systemd/systemd/blob/master/docs/UIDS-GIDS.md) . GitHub. Retrieved 2020-09-26.*
7. *"FreeBSD Porter's Handbook" (http://www.freebsd.org/doc/en/books/porters-handbook/) . Freebsd.org. Retrieved 2016-09-24.*
8. *http://www.freebsd.org/doc/en/books/porters-handbook/users-and-groups.html*
9. *"RHEL7 System changes" (https://www.certdepot.net/rhel7-system-changes/) . Certdepot.net. 2016-01-17. Retrieved 2017-03-22.*
10. *"Getpwuid" (http://pubs.opengroup.org/onlinepubs/009695399/functions/getpwuid.html) . Pubs.opengroup.org. Retrieved 2016-09-24.*
11. *"Chown" (http://pubs.opengroup.org/onlinepubs/009695399/functions/chown.html) . Pubs.opengroup.org. Retrieved 2016-09-24.*

12. *"NetBSD Problem Report #6594: the default "nobody" credentials (32767:9999) do not match mountd's default (-2:-2)" (http://gnats.netbsd.org/6594) . GnaNFSv4ts.netbsd.org. Retrieved 2016-09-24.*

13. *"Namespaces in operation, part 5: User namespaces" (https://lwn.net/Articles/532593/) . Lwn.net. Retrieved 2016-09-24.*