Architecture et Systèmes

Stefan Schwoon

Cours L3, 2014/15, ENS Cachan

Users and groups in Unix

Each user has a numerical identifier (user id, or uid).

Each user also belongs to one *primary* group and possibly multiple other groups.
Each group also has a numerical identifier (gid).

(Group memberships are usually determined by the administrator.)

Users/groups and processes (Part I)

Each process has several attributes relating to users and groups:

- its effective user ID;
- its effective group ID;
- a list of supplementary group IDs.

These determine most of the privileges that a user has with respect to accessing files etc.

C functions: `getuid`, `getgid`, `getgroups`
Shell: `id`

A process normally inherits its IDs from its parent process.
Execeptions: login, setuid (see later)

Users/groups and files

Each file (more precisely: each inode) belongs to some user and some group.

By default, the process that created the file imparts its effective uid and gid on the file.

Access rights on a file are governed by the uid and gid of a file:

- 9 “ugo” bits: (user,group,others) $\times$ (read,write,execute)
- 3 other bits: setuid, setgid, sticky (see later)

Shell commands: `chmod`, `chown`, `chgrp`, `umask`

Access rights: example

Suppose a process opens a file for reading. When does it have the permission?

If the process uid equals the file owner, the file must be readable for its owner.

Else, if one of the process gids equals the file's group, the file must be readable for its group.

Else, the file must be readable for "others".

For write: analogous; execute bit: file may be `exec`'d.

Access rights on directories

rwx bits have a different meaning on directories:

- read: can read the list of files in that directory
- write: can create, rename, delete files in that directory
- execute: can recover the inode data for the files in that directory (`stat`)

Setuid and setgid

Function of the setuid/setgid bits on *files*:

When file is executed, set the effective user/group id of the child to that of the file owner and group.

The original uid/gid of the parent process that created the child is saved in the *real uid/gid*.

C functions: `getuid`, `geteuid`

Function of setgid bit on *directories*:

Files created in the directory will carry the gid of the directory (and not of the process creating the file).

Networks

Networks = in general: exchange of information between computers

Applications:

- Communication between users (e-mail, web)
- Network file system (NFS)
- Remote procedure call (RPC)
- Automatic maintenance and backup
- . . .

Topics today

Vast subject, we can only discuss a few points

$\rightarrow$ specialized course in the M1

OSI layer model

Realizing a reliable protocol on top of an unreliable connection

OS functionality

OSI model

Network communication requires standards and protocols

$\rightarrow$ actually, on many levels

OSI = Open Systems Interconnection, standardized model by ISO

Goal:

separate the various issues in network communication

allow modular definition of protocols

OSI layers

OSI defines seven layers of communication:

physical / data link / network / transport / session / presentation / application

Theoretically, data flow between two users/applications on different computers traverses the seven layers on both sides.

In practice, certain applications do not require all levels and use only some of them; certain protocols may realize multiple layers at once.

1. Physical layer

Defines physical standards for transferring bits from one computer to the other (voltage, speed, other physical specifications).

Examples: Ethernet, ISDN, DSL, . . .

Hardware: repeater, hub

2. Data link layer

Still concerned with point-to-point communication in a local network:

Defines protocols for:

- Packaging sequences of bits into *frames*
- Error detection (and correction)
- Local addressing (MAC address)
- Regulate access to the transfer medium

Two sublayers: Logical link control, media access control

Example: Ethernet, IEEE 802 varieties

Hardware: bridge, switch

3. Network layer

Defines communication beyond local networks, across multiple hops (e.g., Internet)

Each participant (host) must have a unique address in the network

Data arranged into *packets*

Examples: IP, ARP

Hardware: Router

4. Transport layer

Provide reliable communication across a network layer

$\rightarrow$ Network layer ensures that one packet is sent to the next ‘hop’ along the way

$\rightarrow$ if one node along the way is unreliable, packets may still be lost

Transport layer cuts data into packets at the sender, assembles them at the receiver.

Ensures that all packets are eventually received and assembled in the right order.

Examples: TCP, UDP

5. Session layer

Preserve a ‘logical connection’ (session) across multiple connections on the transport level

Example: Occasional communication including multiple TCP connections

Other examples: Authentication services, transfer of large files, remote procedure call

6. Representation layer

Concerns the representation of certain data types:

Character encoding (ASCII, ISO 8859-1, Unicode, . . .)

Integer encoding (big-endian, little-endian, width, decimal encoding, . . .)

7. Application layer

Actual communication between application

For instance, in the HTTP protocol, one defines the data that a web browser must send to a web server to obtain a certain web page/image etc.

Examples: SSH, HTTP, FTP, . . .

Telnet: general-purpose utility

Adressing in IP: ports

In the IP protocol, an Internet address consists of a *machine address* and a *port*.

A port can be thought of as fine-grained addressing:
each machine can have 2^{16} of them.

Ports 0..1023 reserved for special use, certain ports pre-defined.

E.g., 80 for HTTP protocol.

IP sockets and OS functionality

Communication in the IP protocol happens through so-called sockets.

In POSIX, *sockets* are realized through file descriptors.

A socket is created using `socket`(2).

A socket, once created, can be connected to an Internet address, see `bind`(2) and `connect`(2).

Further: `listen` / `accept`.