Operazione Rif. PA 2023-19410/RER approvata con DGR 1317/2023 del 31/07/2023 finanziata con risorse del Programma Fondo sociale europeo Plus 2021-2027 della Regione Emilia –Romagna.

Progetto n. 1 - Edizione n. 1

TECNICO PER LA PROGETTAZIONE E LO SVILUPPO DI APPLICAZIONI INFORMATICHE

MODULO: N. 5 Titolo: SICUREZZA DEI SISTEMI INFORMATICI E DISPIEGO DELLE APPLICAZIONI
DURATA: 21 ORE DOCENTE: MARCO PRANDINI

AVVIO DEL SISTEMA SOFTWARE E PROCESSI

Operare un sistema sicuro

- Rifacciamoci alle definizioni base di sistema sicuro
  - assurance: certezza che operi come previsto
- Ipotizziamo (per ora) di disporre di componenti software tutti sicuri e correttamente configurati.
- Che rischi rimangono?
  - Garantire che vengano utilizzati quelli e non altri
  - Garantire che non vengano scavalcati da attacchi non software

On premises - messa in sicurezza fisica

- Un server è prima di tutto un sistema di calcolo, collocato in un ambiente e connesso a una varietà di dispositivi
  - Normalmente si concentrano le difese sul fronte degli attacchi via rete, a componenti software come applicazioni e sistema operativo
  - Le corrispondenti contromisure possono facilmente essere scavalcate da un attaccante con accesso fisico al sistema!
  - Le minacce principali sono:
    - Furto dello storage o dell'intero calcolatore
    - Connessione di sistemi di raccolta dati alle interfacce
    - Avvio del sistema con un sistema operativo arbitrario
  - La gravità di queste minacce dipende fortemante dallo specifico ambiente
- Molti di questi problemi sono cambiati nello scenario sempre più comune di virtualizzazione sul Cloud, ma altri concettualmente simili sono apparsi, e la logica delle stesse contromisure si può adattare

Fattori non informatici

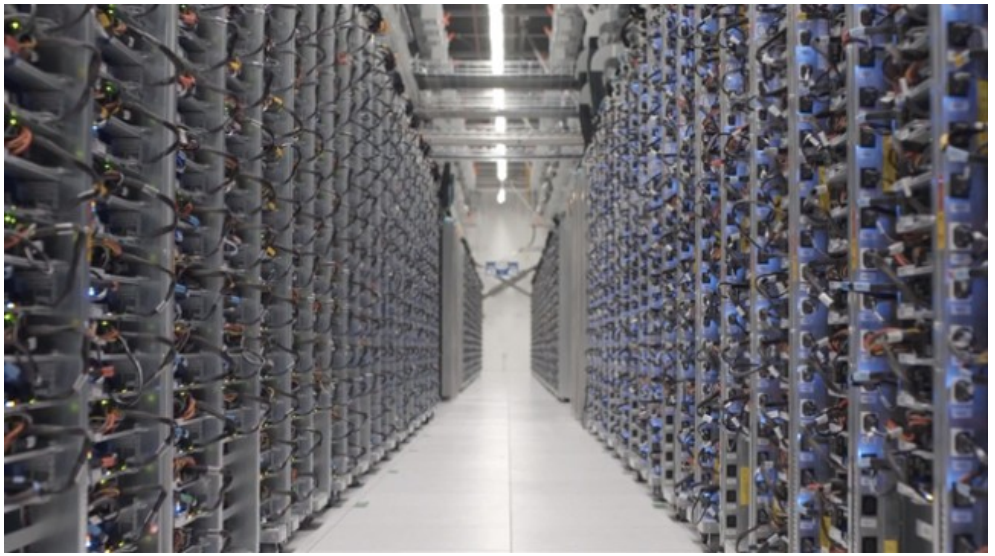
- Vedremo che l'accesso a sistema permette attacchi specifici
- Come si ottiene l'accesso?
  - Insider
  - Tailgating
  - Errata identificazione di visitatori
  - Social engineering
  - Effrazione
- La sicurezza fisica "tradizionale" è essenziale!
  - Regolamenti chiari e condivisi
  - Perimetro robusto
  - Sorveglianza e procedure
    https://docs.microsoft.com/it-it/azure/security/fundamentals/physical-security
- La disponibilità è il terzo vertice della triade CIA
  - Alimentazione
  - Connettività
  - Condizionamento
  - Incidenti, disastri, attentati
    https://goo.gl/maps/5Ukzcorg5pZNmbzW7

Alcune vulnerabilità sfruttabili in presenza

- BadUSB e simili
  https://threatpost.com/badusb-attack-code-publicly-disclosed/108663/
- Thunderspy
  https://thunderspy.io/
- Keylogging e videoghosting
  https://www.keelog.com/
- Key injection
  https://www.blackhillsinfosec.com/executing-keyboard-injection-attacks/
- Disk un/plugging
  https://www.blackhat.com/docs/eu-15/materials/eu-15-Boteanu-Bypassing-Self-Encrypting-Drives-SED-In-Enterprise-Environments-wp.pdf
- Power glitching
  https://www.darkreading.com/edge/theedge/glitching-the-hardware-attack-that-can-disrupt-secure-software-/b/d-id/1336119

- Il salto dell'Air Gap
  - Research on Air Gap Jumping (key points)
  - Unsafe at home: a tale of security vs convenience

In remoto



- Se la collocazione è fuori dalla possibilità di controllo diretto, si può considerare di:
  - Scegliere un case che possa essere chiuso e fissato al rack
  - Installare dispositivi di rilevazione delle intrusioni
  - Adottare misure di protezione dei dati che rendano inutile il furto
    - L'accesso ai dati va però abilitato manualmente
  - Disabilitare le periferiche non utilizzate
    - Salvo poi averne bisogno per esigenze nuove

Attacchi fisici alle risorse logiche

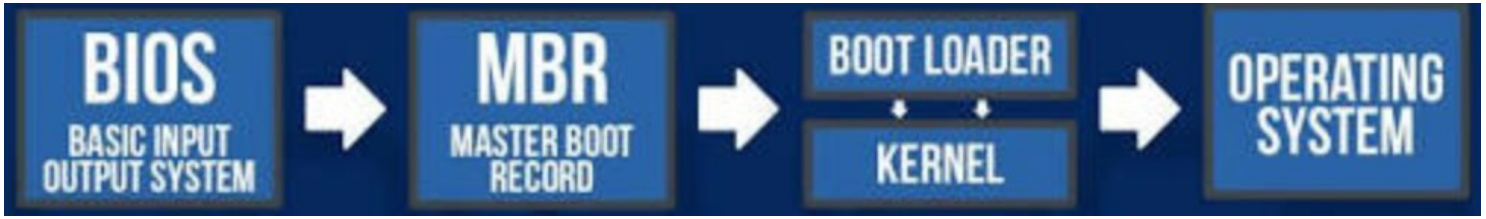
- Per andare a regime il sistema attraversa un processo di boot, che può essere diviso in queste fasi:
  (1) BIOS – Individua i dispositivi di possibile caricamento del boot loader e l'ordine per esaminarli
    - Molti BIOS prevedono la possibilità di proteggere con password l'avvio o la modifica della configurazione
  (2) Boot Loader – Sceglie il sistema operativo e gli passa eventuali parametri
    - Gestione della "maintenance mode"
    - Stesso tipo di protezione con password come descritto per BIOS
  (3) Sistema opeativo – carica i device driver (da non sottostimare) e avvia il processo *init*
  (4) *init* – gestisce i *runlevel* o i *target* per coordinare l'inizializzazione del sistema, cioè avviare i servizi nell'ordine corretto
- Ognuna di queste fasi potrebbe essere dirottata da un attaccante con accesso fisico, per far caricare software malevolo

Il processo di boot

- La maggior parte dei computer si avvia con la sequenza
  1. un firmware controlla l'hardware e avvia un boot loader
  2. il boot loader individua e carica un kernel di SO
  3. il SO avvia i processi di base
- I dettagli possono variare molto tra diversi hardware e SO
- Per il firmware ci sono due standards fondamentali:
  - BIOS - Basic Input/Output System
  - UEFI - Unified Extensible Firmware Interface

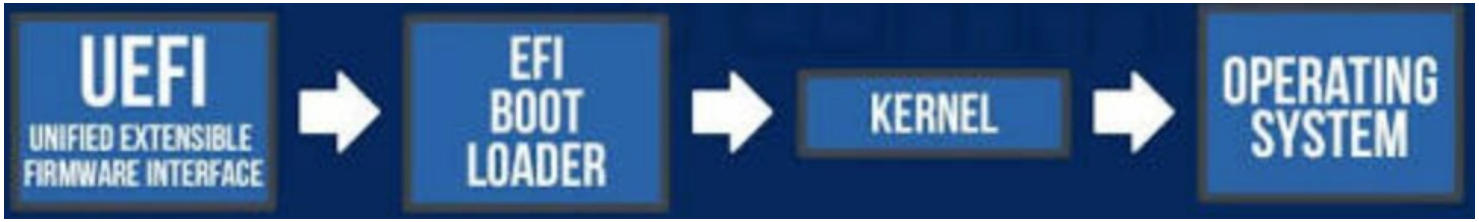
BIOS e MBR

- Il BIOS (Basic Input/Output System) firmware fa due operazioni essenziali:
  - controlla il corretto funzionamento di base (Power-On-Self-Test)
  - trova un metodo di avvio e gli passa il controllo
- Il BIOS è strettamente legato allo standard MBR (Master Boot Record), un record di 512 byte su disco che contiene
  - la tabella delle partizioni
  - il codice di bootstrap: ricerca un file sul disco e lo avvia



UEFI

- BIOS è ormai rimpiazzato da UEFI (Unified Extensible Firmware Interface) che è molto più flessibile
  - https://www.uefi.org/
- La maggior parte dei sistemi UEFI può essere configurata per utilizzare il BIOS vecchia maniera
- UEFI è basato su GPT (GUI Partition Table) per le partizioni e dispone di molto più spazio rispetto a MBR
- Monta un filesystem dedicato (EFI System Partition, ESP) per caricare il boot loader



UEFI vs Legacy BIOS

	Legacy BIOS	UEFI
Memory/ Space	• Uses MBR (32-bits) • 400b in boot sector • Max 4 disk partitions • Max 2Tb disk size	• Uses GUID Partition Table (64-bits) • Dedicated filesystem for boot loader (.efi) • Unlimited disk partitions • Max 9 Zb disk size
Performance	Limited	Much faster due to more space
Security	No specific mechanisms	Secure Boot, in which only trusted software is executed

BIOS/UEFI Passwords

- BIOS or UEFI firmware offers the ability to set lower-level passwords prompted before bootstrapping, so as to restrict people from:
  - booting the computer
  - booting from removable device
  - changing settings without your permission
- Any drawback?
  - automatic reboot (ex. after electricity dropdown) is not possible

BIOS/UEFI Passwords (2)

- No fixed rule here:
  - Always require a password to change the configuration
  - Not always to start the boot process: for instance, when the workstation can only be physically accessed by authorized people and automatic reboot is required
- Another issue: BIOS passwords are easy to reset, es. working on motherboard jumpers or CMOS batteries. Vary between systems but instructions are easily available

  →Lock the computer case when possible
- Golden rule: do not rely on a single layer of protection

Bootloader – configurazione (runtime)

- LILO, the Linux Loader
  - Usato fin dagli albori di Linux
- GRUB, the Grand Unified Bootloader
  - GRUB è più potente e flessibile di LILO, è dotato di una shell che permette di eseguire vari comandi per modificare al volo la procedura di avvio: naturalmente questo permette molti abusi
- Entrambi permettono di passare parametri al kernel, i più importanti ai fini della sicurezza sono
  - single
  - Init=...
- Alcune distribuzioni hanno default rischiosi, ad esempio se si riesce a innescare un *maintenance mode* aprono una shell di root senza chiedere password

Boot Loader passwords

- LILO

  `password=YourPasswordHere`

  Imposta una password richiesta al boot, a meno che

  `restricted`

  non sia is specificato, in tal caso la password è richoesta solo per modificare I parametri durante il boot.

- Global vs. Single-entry
  - `password` e `restricted` nella “global section”: chiede la password prima di consentire l’aggiunta di parametri – attenzione alle entry non sicure (cd)
  - `password` e `restricted` in una “image section”: chiede la password prima di consentire l’aggiunta di parametri, solo per l’immagine specificata
  - `password` nella “global section” e `restricted` in una “image section”: chiede la password prima di consentire l’aggiunta di parametri, solo per l’immagine specificata, mentre chiede sempre la password per avviare altre immagini

GRUB2

- GRUB 2 è il boot loader package da GNU Project
  - multi-boot e multi hardware supportato
  - riga di comando interattiva per modificare/testare opzioni all'avvio
  - configurazione flessibile
- Singolo file di configurazione: `/boot/grub/grub.cfg`
- Sovrascritto ogni volta che un nuovo kernel viene installato, o lanciando manualmente
  - `update-grub`
    o
  - `grub-mkconfig -o /boot/grub/grub.cfg`

Boot Loader passwords

- **GRUB**

  `password [--md5] passwd [new-config-file]`

  Se specificato nella "global section", imposta una password che sarà richiesta per attivare l'*interactive operation* del bootloader. Opzionalmente può innescare il caricamento di un file di configurazione alternativo

  Se specificata per un item specifico del menu, imposta una password che sarà richiesta per avviare quell'item

  `lock`

  Se specificata per un item specifico del menu, subito dopo `title`, contrassegna quell'item come password-protected.

  Funziona solo se esiste una direttiva password nella "global section"

  `md5crypt`

  Comando utilizzabile al grub prompt per calcolare il password hash da usare con `--md5`

GRUB2 configuration

- Il file di configurazione viene prodotto analizzando il sistema e i file contenuti nella directory `/etc/grub.d/`
- Contiene una serie di script eseguiti in ordine alfabetico
- Esempio:
  - `00_header`: loads GRUB 2 settings from `/etc/default/grub` file
  - `10_linux`: locates kernels in the default partition
  - `30_os-prober`: entries for operating systems found on other partitions
  - `30_uefi-firmware`: entries for UEFI-based boot process
  - `40_custom`: used to create additional menu entries

Come impostare una password in GRUB2

- Tool per la creazione:

  `grub-mkpasswd-pbkdf2`

  - genera e stampa una password in forma cifrata

    `grub.pbkdf2.sha512.10000.97AB1234...`

- Editare lo script di inizializzazione:

  `/etc/grub.d/00_header`

- Impostare username e password creata:

```
cat << EOF
set superusers="username"
password_pbkdf2 username grub.pbkdf2.sha512.10000.97AB1234
  … …
EOF
```

- Aggiornare la configurazione: `update-grub`

Boot protection

- Password pros and cons:
  - If a password is needed for booting the system, unattended operation can be problematic: a simple power outage can make the system unavailable
    - For systems where privacy and integrity considerations override availability issues, this is a minor problem, since probably there will also be specific services refusing to start if a password is not manually entered (for example to decrypt private keys they use)
  - Password protection against system configuration alterations is always advisable
- NEVER rely on a single protection layer
  - BIOS passwords can often be overridden by manufacturer's defaults
  - Any password can be guessed
  - Risky defaults on some distributions (e.g.: if a means of requesting maintenance boot to the boot loader is found, *init* provides a root shell)

This is a very useful feature to legitimately gain control of a corrupted system which will not boot, the other being booting from an external media (→ BIOS)
Bottom line: lock-down = increased integrity, lower availability!

Sicurezza del processo di boot

- Problema: come assicurarsi che ogni componente software eseguito da un computer sia autentico, integro e benevolo?
  - Anti-malware verificano le applicazioni
  - Chi verifica gli anti-malware?? Il S.O. (idealmente rendendo AM inutile)
  - Chi verifica il S.O.? Il boot loader potrebbe
  - Chi verifica il boot loader? Il BIOS potrebbe, specialmente se assistito da HW speciale, che non possa essere modificato dal S.O., e quindi sia immune da infezioni

    → *hardware root of (a chain of) trust*

    https://medium.com/@martin_24447/trusted-boot-b1ae7e6d2890

    https://dl.acm.org/doi/pdf/10.1145/3380774.3382016

Secure systems on top of unsecure firmware?

- From the UEFI white paper: “Imagine a multi-million-dollar mansion with a sturdy fence surrounding the grounds, full-time security guards, cameras and alarms systems [...] and a tunnel for the owner and security team. [...]
  Above-ground security measures are easily defeated if a burglar discovers the hidden tunnel and a way into the mansion [...] The tunnel [is the] unsecured firmware that runs when computing devices start up, completely bypassing perimeter defenses like firewalls and anti-virus software

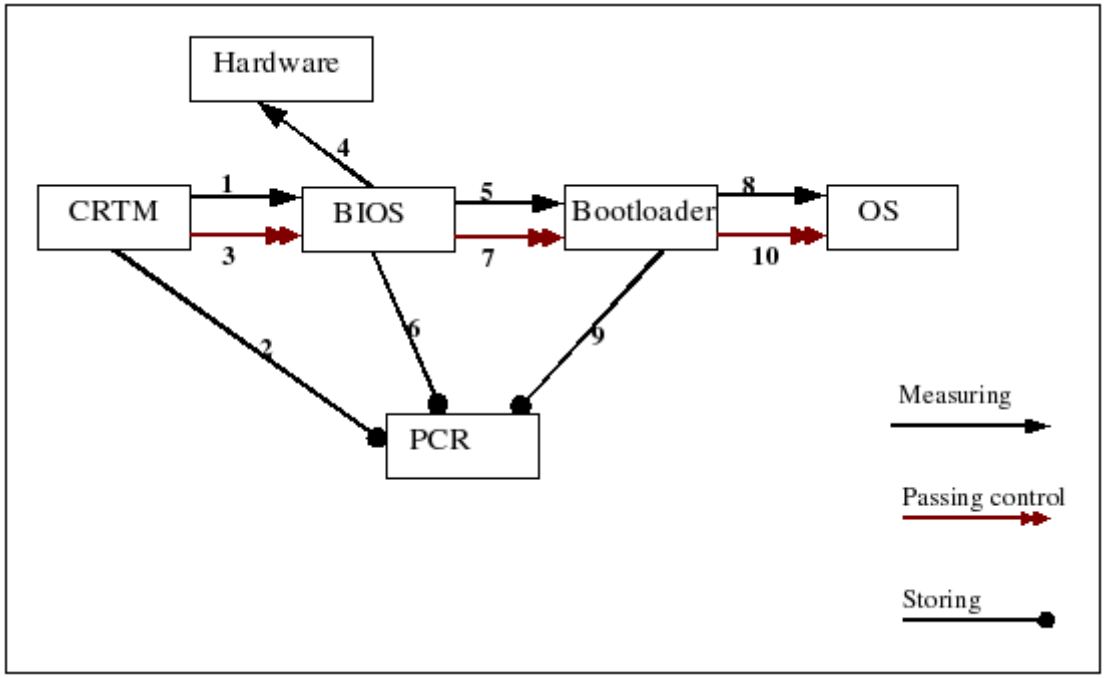
  Wilkins, R., & Nixon, T. (2016). The Chain of Trust.

Measured / Trusted / Secure Boot

- *Measured Boot* si riferisce a un processo generale, che tipicamente usa un TPM come hardware root of trust
  - TPM = Trusted Platform Module: chip con funzionalità crittografiche
    - fa parte delle specifiche del *Trusted Computing Group* https://trustedcomputinggroup.org/
  - M.B. non definisce come prevenire un avvio malevolo
- *Trusted Boot* è un processo che usa gli strumenti del M. B. e riesce a bloccare il boot non appena individua un componente non fidato
- *Secure Boot* è il nome specifico dato all'implementazione di trusted boot basata su UEFI
  - UEFI = Unified Extensible Firmware Interface http://www.uefi.org/
  - Implementazione Software + chiavi in firmware
  - Serve un BIOS standard per la fase di POST
  - Può avvalersi del TPM per velocizzare e migliorare i controlli di integrità

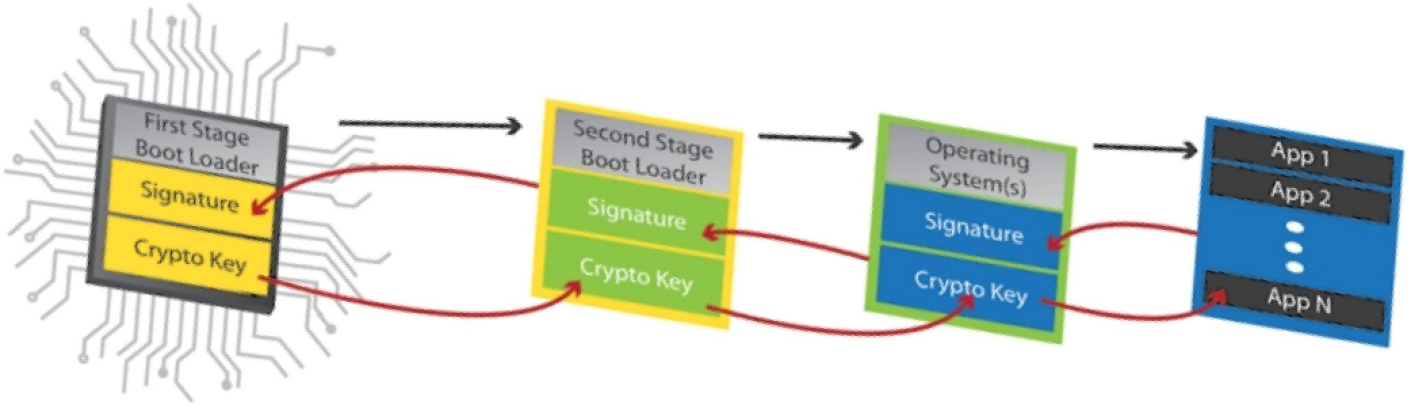
Measured boot

- Basato sul TPM
  - Core Root of Trust for Measurement (CRTM)
  - Registers (PCR)
- Raccoglie hash di ogni componente caricato
  - nei PCR che sono fisicamente non modificabili una volta scritti
- Pospone i controlli fintanto che non dispone
  - Delle crypto keys
  - Di abbastanza memoria per fare i calcoli necessari
- Si può decidere chi fa i controlli e quando
  - per esempio dall'esterno (sistema fidato) per abilitare funzioni critiche
  - remote attestation!



UEFI Secure Boot

- Each step checks a cryptographic signature on the executable of the next step before launching it
  - the BIOS will check a signature on the loader
  - the loader will check signatures on all the kernel objects that it loads
- Signed components are verified against a certificates database stored in the firmware
- If any of the components have been tampered, then the signatures does not match, and the device stops the boot process



UEFI and the secure boot

- EFI (from Intel) was born as a more-flexible-than-BIOS interface between the OS and the firmware
- UEFI forum standardized and updated it
  - http://www.uefi.org/
- UEFI is a “mini OS”
  - BIOS boot via MBR:
    - 400 bytes of ASM in boot sector
    - 4 primary partitions or 3 primary parts + 11 logical units
  - EFI with GPT
    - its own filesystem (100-250MB) for boot loaders
    - nearly unlimited partitions of up to 9ZB
    - downsides too!
      - Complex → hard to secure
      - Standard → high impact if broken

    https://media.kaspersky.com/en/business-security/Threats_to_UEFI.pdf

    https://www.csoonline.com/article/3599908/trickbot-gets-new-uefi-attack-capability-that-makes-recovery-incredibly-hard.html

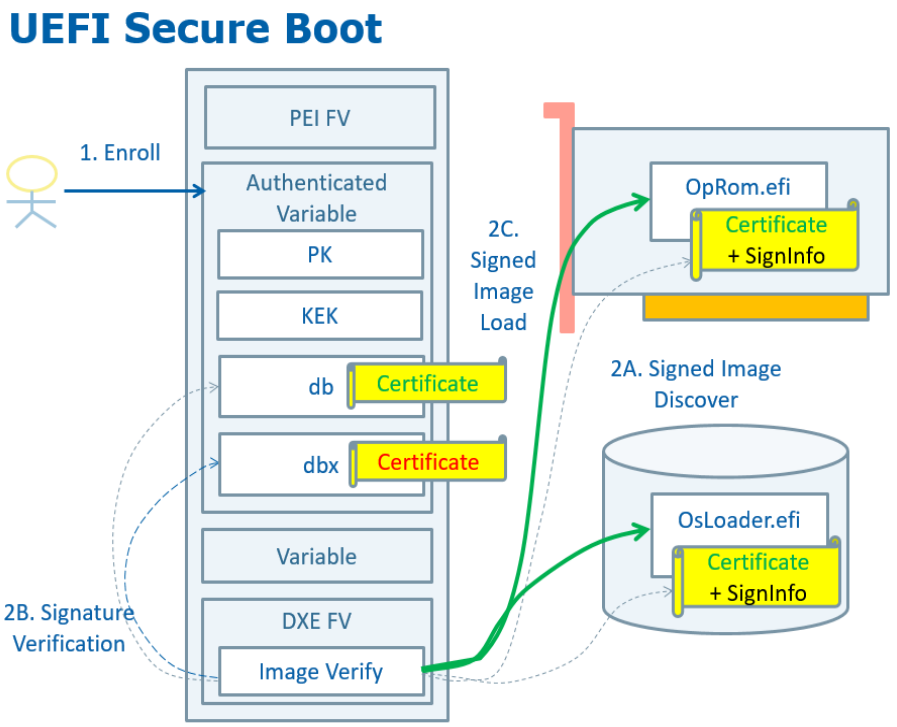
    https://www.debian.org/security/2021-GRUB-UEFI-SecureBoot/
- UEFI verifies each piece of software before yielding control
  - It needs a key database to be always available
  - As soon as a verification fails, the boot process stops

Le chiavi di UEFI Secure Boot

https://edk2-docs.gitbook.io/understanding-the-uefi-secure-boot-chain/secure_boot_chain_in_uefi/uefi_secure_boot

- UEFI Secure Boot definisce due processi di sicurezza:
  - verifica dell'immagine di boot
  - verifica degli aggiornamenti al database della sicurezza delle immagini
- Per fare questo si avvale di differenti database e set di chiavi

Key	Verifies	Update is verified by	NOTES
PK	New PK New KEK New db/dbx/dbt/dbr New OsRecoveryOrder New OsRecovery####	PK	Platform Key
KEK	New db/dbx/dbt/dbr New OsRecoveryOrder New OsRecovery####	PK	Key Exchange Key
db	UEFI Image	PK/KEK	Authorized Image Database
dbx	UEFI Image	PK/KEK	Forbidden Image Database
dbt	UEFI Image + dbx	PK/KEK	Timestamp Database
dbr	New OsRecoveryOrder New OsRecovery####	PK/KEK	Recovery Database



UEFI Secure Boot Image Verification

https://edk2-docs.gitbook.io/understanding-the-uefi-secure-boot-chain/secure_boot_chain_in_uefi/uefi_secure_boot

- Entità coinvolte nel processo di verifica delle immagini al boot
  - TP = trusted platform, procedura di verifica
  - CDI = UEFI Secure Boot Image Security Database
  - UDI = qualsiasi firmware di terze parti, inclusi boot loader, PCI option ROMs, o UEFI shell tool.
- Al boot, TP verifica l'integrità di UDI utilizzando le policy CDI
  - se ok, UDI entra a far parte di CDI e il firmware di terze parti viene eseguito
- Il CDI, cioè il database delle politiche di sicurezza da applicare alle immagini software da caricare, è quindi aggiornabile.
  - Il fornitore del componente deve firmarlo con la propria chiave privata e rendere disponibile la chiave pubblica.
  - la chiave pubblica deve essere iscritta (enrolled) nel firmware del sistema
  - normalmente questo passaggio richiede un reboot in una modalità speciale e l'intervento sulla console, bloccando quindi l'azione di utenti malevoli ma senza accesso fisico al sistema

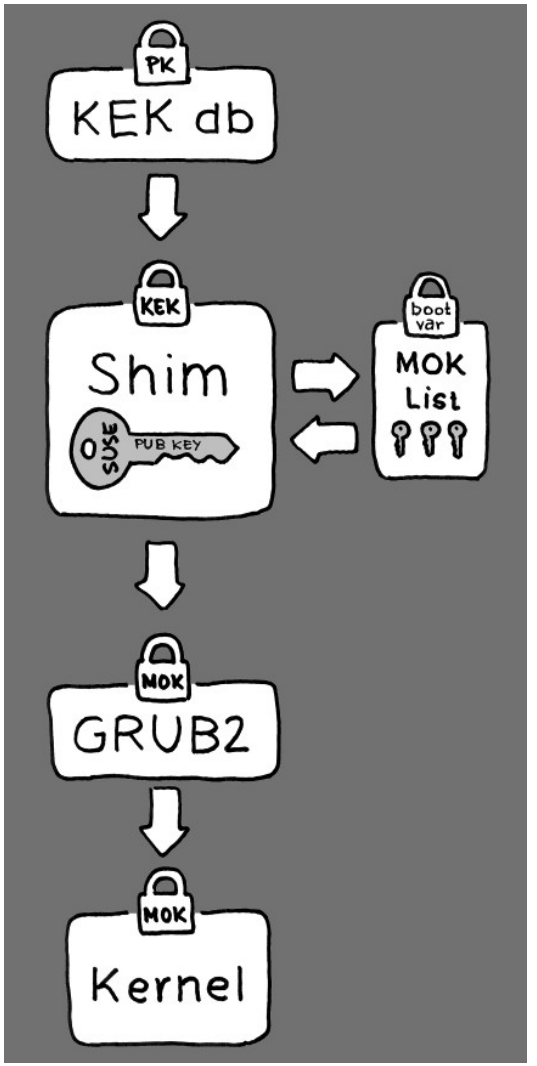
Item	Entity	Provider	Location
TP	UEFI Secure Boot Image Verification	OEM	Originally on flash, loaded into DRAM
CDI	Manufacture Firmware Code	OEM	Originally on flash, loaded into DRAM
	UEFI Secure Boot Image Security Database (Policy)	End user (or OEM default)	Originally on flash, authenticated variable region, loaded into DRAM
UDI	3rd party Firmware Code, (OS boot loader)	OSV	Originally on external storage (e.g. Hard drive, USB), loaded into DRAM
	3rd party Firmware Code, (PCI Option ROM)	IHV	Originally on PCI card, loaded into DRAM
	3rd party Firmware Code, (UEFI Shell Tool)	Any	External Storage (e.g. hard drive, USB), loaded into DRAM

UEFI e secure boot in Linux

https://www.suse.com/media/presentation/uefi_secure_boot_webinar.pdf
https://wiki.ubuntu.com/UEFI/SecureBoot

1) La Platform Key ufficiale verifica un piccolo pre-boot-loader, **shim**
   - La chiave key usata per "firmare" shim deve essere fornita dal costruttore HW
   - È una chiave Microsoft!
2) Shim può usare o trasferire MOKs (Machine Owner Keys)
   - Per validare il bootloader
   - Per validare moduli custom del kernel

- Componenti aggiuntivi del kernel devono essere firmati per poterli caricare
  - L'utente genera le MOKs
  - L'utente deposita le MOKs in shim
  - Al boot successivo, shim trova le chiavi nella fase di setup, e chiede conferma per salvarle in firmware → **consenso esplicito e basato su password sempre richiesto!**



https://www.suse.com/communities/blog/uefi-secure-boot-details/

UEFI / Secure boot links

- E-book con schemi chiari e riferimenti a proprietà formali di sicurezza secondo il modello di Clark-Wilson garantite dal procedimento
  - https://edk2-docs.gitbook.io/understanding-the-uefi-secure-boot-chain/
- Documentazione originale
  - https://uefi.org/sites/default/files/resources/UEFI_Secure_Boot_in_Modern_Computer_Security_Solutions_2013.pdf
- Articoli datati ma illustrano i problemi incontrati all'introduzione di UEFI
  - http://www.rodsbooks.com/linux-uefi/
  - http://www.linux-magazine.com/Online/Features/Coping-with-the-UEFI-Boot-Process
  - https://help.ubuntu.com/community/UEFI
  - http://askubuntu.com/questions/760671/could-not-load-vboxdrv-after-upgrade-to-ubuntu-16-04-and-i-want-to-keep-secur
  - https://www.suse.com/communities/blog/uefi-secure-boot-details/
  - https://lwn.net/Articles/519618/
- Guide alla personalizzazione
  - https://media.defense.gov/2020/Sep/15/2002497594/-1/-1/0/CTR-UEFI-Secure-Boot-Customization-UOO168873-20.PDF
  - https://www.linuxjournal.com/content/take-control-your-pc-uefi-secure-boot

Dopo il boot

- Il sistema operativo è modulare
  - kernel, caricato tramite il processo di avvio
  - moduli, caricati e agganciati al kernel per fornire funzionalità aggiuntive (ad esempio driver di dispositivo, stack di protocollo, modelli di controllo degli accessi, ecc.)
- I moduli sono caricati
  - Manualmente (poco frequente)
  - Automaticamente (più comune)
    - seguendo indicazioni di configurazione manuale
    - A seguito di eventi di rilevamento hardware
- Guide utili
  - https://wiki.archlinux.org/title/Kernel_module
  - https://docs.fedoraproject.org/en-US/fedora/latest/system-administrators-guide/kernel-module-driver-configuration/Working_with_Kernel_Modules/
  - https://www.cyberciti.biz/faq/linux-how-to-load-a-kernel-module-automatically-at-boot-time/

Integrità dei moduli

- Viene verificata con una firma digitale

```
# modinfo psmouse
filename:
/lib/modules/4.13.0-37-generic/kernel/drivers/input/mouse/psmouse.ko
license:        GPL
description:    PS/2 mouse driver
author:         Vojtech Pavlik <vojtech@suse.cz>
srcversion:     16F6FEC23F72FA71FF21E33
alias:          serio:ty05pr*id*ex*
alias:          serio:ty01pr*id*ex*
depends:
intree:         Y
name:           psmouse
vermagic:       4.13.0-37-generic SMP mod_unload
signat:         PKCS#7
```

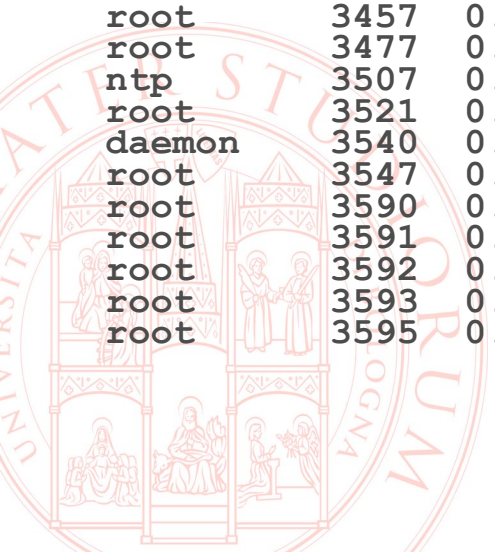
Aggiornamenti software e Secure Boot

- Come al solito: compromesso tra sicurezza e usabilità
  - massima sicurezza: kernel statico senza il supporto dei modulo
  - sistema tipico (distribuzioni comuni): tutto è un modulo
- Secure Boot può intralciare l'aggiornamento di software non gestito dalla distribuzione
  - per esempio. moduli del kernel per VirtualBox
- Come firmare manualmente il software per Secure Boot
  1) genera la tua coppia di chiavi pubblica-privata
  2) registra il tuo certificato di chiave pubblica tramite shim
  3) reboot – in questo passaggio viene richiesta conferma della registrazione della chiave pubblica, possibile solo avendo accesso fisico e conoscendo una password scelta al passo (3)
  4) ora puoi firmare il software con la tua chiave privata e la firma sarà convalidata da shim e dal kernel

Gestione dei processi (prequel)

- Dopo un'installazione "minimale"...

```
milk:~# ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0   1948   468 ?        Ss   May15   0:02 init [2]
[... kernel processes ...]
root      1753  0.0  0.0   2704   392 ?        S<s  May15   0:00 udevd --daemon
daemon    2953  0.0  0.0   1688   408 ?        Ss   May15   0:00 /sbin/portmap
root      3231  0.0  0.0   1624   568 ?        Ss   May15   0:26 /sbin/syslogd
root      3237  0.0  0.0   1576   340 ?        Ss   May15   0:00 /sbin/klogd -x
bind      3251  0.0  0.1  39732  1964 ?        Ssl  May15   0:00 /usr/sbin/named
root      3266  0.0  0.0  39500   944 ?        Ssl  May15   0:00 /usr/sbin/lwres
root      3339  0.0  0.0   1572   444 ?        Ss   May15   0:00 /usr/sbin/acpid
103       3344  0.0  0.0   2376   760 ?        Ss   May15   0:00 /usr/bin/dbus-d
106       3352  0.0  0.1   6116  1972 ?        Ss   May15   0:03 /usr/sbin/hald
root      3353  0.0  0.0   2896   716 ?        S    May15   0:00 hald-runner
106       3359  0.0  0.0   2016   472 ?        S    May15   0:00 hald-addon-acpi
106       3367  0.0  0.0   2020   480 ?        S    May15   0:00 hald-addon-keyb
root      3387  0.0  0.0   1808   360 ?        S    May15  14:15 hald-addon-stor
root      3414  0.0  0.0   1864   396 ?        Ss   May15   0:00 /usr/sbin/dhcdb
root      3421  0.0  0.1   3984  1164 ?        Ss   May15   0:00 /usr/sbin/Netwo
avahi     3433  0.0  0.1   2936  1424 ?        Ss   May15   4:14 avahi-daemon: r
avahi     3434  0.0  0.0   2552   180 ?        Ss   May15   0:00 avahi-daemon: c
root      3441  0.0  0.0   2908   536 ?        Ss   May15   0:00 /usr/sbin/Netwo
root      3457  0.0  0.0   1752   452 ?        Ss   May15   0:02 /usr/sbin/inetd
root      3477  0.0  0.0   4924   512 ?        Ss   May15   0:02 /usr/sbin/sshd
ntp       3507  0.0  0.0   4144   764 ?        Ss   May15   0:00 /usr/sbin/ntpd
root      3521  0.0  0.0   1976   724 ?        Ss   May15   0:02 /sbin/mdadm --m
daemon    3540  0.0  0.0   1828   280 ?        Ss   May15   0:00 /usr/sbin/atd
root      3547  0.0  0.0   2196   720 ?        Ss   May15   0:00 /usr/sbin/cron
root      3590  0.0  0.0   1572   372 tty2     Ss+  May15   0:00 /sbin/getty 384
root      3591  0.0  0.0   1576   372 tty3     Ss+  May15   0:00 /sbin/getty 384
root      3592  0.0  0.0   1572   372 tty4     Ss+  May15   0:00 /sbin/getty 384
root      3593  0.0  0.0   1572   372 tty5     Ss+  May15   0:00 /sbin/getty 384
root      3595  0.0  0.0   1576   372 tty6     Ss+  May15   0:00 /sbin/getty 384
```



Systemd – avvio e arresto dei servizi

- La maggior parte dei processi visibili sono *demoni* (processi non collegati ad alcun terminale) avviati da *systemd* (il demone con PID 1)
- Controllo a run time dei servizi
  - `systemctl {start|stop|status|restart|reload} servicename`
    - ...intuitivo
    - output molto descrittivo dello stato
      - stato corrente ed elenco dei passi fatti per raggiungerlo
      - process tree
      - righe di log rilevanti
    - con `-H [hostname]` si connette a un host remoto via ssh
- Cosa fanno in realtà? Vedremo in pratica i dettagli, ma in sintesi
  - Nelle unit sono definiti i comandi da eseguire attraverso parametri di configurazione (ExecStart, ExecReload, ExecStop)
    - **restart** è semplicemente **stop** seguito da **start**
  - systemd tiene traccia dei processi avviati con **start**, in modo che i loro PID possano essere usati come parametri nei comandi **reload**/**stop**
  - **stop**, oltre a eseguire (l'eventuale) comando specificato, di default **manda SIGTERM al processo, seguito da SIGKILL dopo un timeout**. Moltissime varianti configurabili: **man 5 systemd.kill**

Systemd – boot e shutdown

- Le operazioni descritte alla slide precedente sono **volatili**
  - si impartisce il comando
  - si ottiene l'effetto
  - nulla cambia nella **configurazione** del sistema
- Per automatizzare avvio al boot e arresto allo shutdown si utilizzano invece
  - `systemctl {enable|disable|mask|unmask} servicename`
    - `disable` lascia disponibile la possibilità di usare manualmente `start`
    - `mask` "neutralizza" l'intera definizione della unit, impedendo anche il controllo manuale
  - questi comandi non hanno alcun effetto immediato
  - **l'effetto sulla configurazione del sistema è persistente**

Systemd – verifica della configurazione

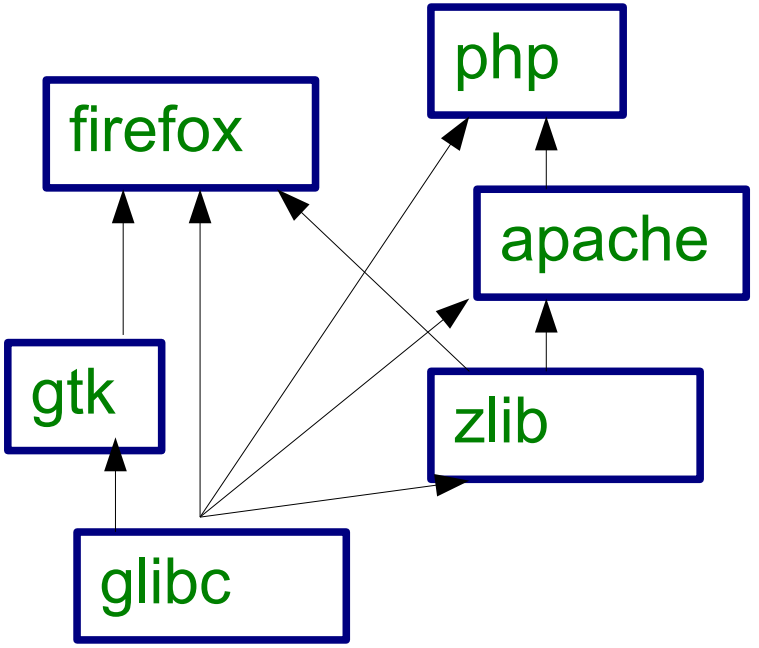
- Solo qualche esempio
  - `systemctl list-units`
    - mostra tutte le *unit* gestite (di tutti i tipi elencati!)
  - `systemctl -t type`
    - es.: `systemctl -t timers`
    - mostra tutte le unit attive del tipo specificato
  - `systemctl list-unit-files [-t type]`
    - es.: `systemctl list-unit-files -t services`
    - mostra tutte le unit installate del tipo specificato
  - `systemctl --state state`
    - es.: `systemctl --state failed`
    - mostra tutte le unit che si trovano nello stato specificato

Gestione dei pacchetti software

Installazione assistita

- Comunemente effettuata per mezzo di software ausiliari
  - package manager specifico della distribuzione Linux (rpm/yum, dpkg/apt, ...)
  - installer per Windows
- Un tool di installazione
  - può farsi carico delle verifiche relative alle dipendenze
  - non può configurare ogni dettaglio del sistema in modo specifico
  - può generare dinamicamente dati specifici

Esempio di *grafo delle dipendenze*:



A → B significa che A “serve” per B; “serve” può essere una dipendenza tra funzionalità logiche (non ha senso avere un linguaggio di generazione pagine web senza un web server) o fisiche (un binario linkato dinamicamente non gira senza tutte le librerie di cui importa i simboli)

Pacchetti

- Le *distribuzioni* di Linux organizzano il software in *pacchetti* e dispongono di un *package manager* per la loro gestione
- Un pacchetto si presenta sotto forma di singolo file che contiene in forma compatta l'insieme di
  - software precompilato
  - criteri per la verifica della compatibilità e dei prerequisiti
  - procedure di pre/post-installazione
- La garanzia della compatibilità con un determinato sistema può essere data solo a patto di vincolare con precisione alcuni parametri:
  - architettura
  - versione della distribuzione
  - versione del software contenuto nel pacchetto

Distribuzioni: criteri per la scelta

Architetture supportate

- Tutte le distribuzioni supportano i processori Intel 32bit, la maggior parte quelli a 64bit, alcune sono disponibili per tutte le varietà di processori su cui è stato portato il kernel
- È bene ricordare che i pacchetti di terze parti potrebbero non essere disponibili per tutte le architetture supportate

Stabilità vs. Aggiornamento

- Il processo di rilascio frequente e continuo del software nel mondo GNU/Linux ha come conseguenza inevitabile che le versioni più aggiornate possano essere meno stabili
- Vi sono distribuzioni che hanno come filosofia l'inclusione dei pacchetti più recenti (e quindi con funzionalità maggiori) anche a costo di una minor robustezza, ed altre che garantiscono l'inclusione solo di software ben collaudato

Distribuzioni: criteri per la scelta

Version vs rolling

- Alcune distribuzioni sono “versionate”: durante il ciclo di vita di una versione vengono forniti solo aggiornamenti correttivi, tutte le novità vengono testate e accumulate per la pubblicazione in una nuova versione (che va installata sovrascrivendo la precedente)
- Altre sono “rolling”: ogni volta che c’è una novità viene testata e distribuita, quindi in ogni momento il sistema è alla versione più recente

Supporto e durata

- La disponibilità di supporto garantito è tipica delle distribuzioni commerciali, ma anche con le distribuzioni gratuite più diffuse, in virtù della dimensione della relativa comunità di utenti, è semplice risolvere eventuali problemi
- Per installazioni di tipo server esistono varianti denominate LTS (Long Term Support): per 5/7 anni chi cura la distro garantisce che gli aggiornamenti non modifichino le API (tipicamente viene garantito solo il backporting dei security fix, non quello di tutti i bug fix)

Ampiezza del set di pacchetti

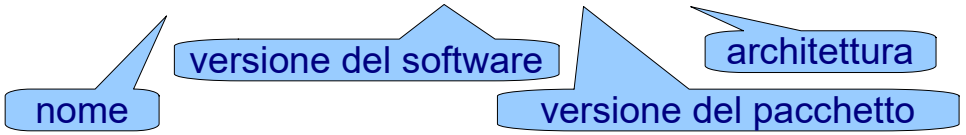
- Si va dai 1500 delle distro minimali ai 26000 di Debian
- Una scelta intelligente mette tutto l'essenziale in 1 CD

Debian e Red Hat

- Due distribuzioni capostipite da cui sono state derivate quasi tutte le varianti più diffuse
  http://upload.wikimedia.org/wikipedia/commons/9/9a/Gldt1009.svg
- Due sistemi di gestione dei pacchetti con molte somiglianze
  - Tool di basso livello per la gestione dei singoli pacchetti
  - Tool intermedi per la gestione coordinata di pacchetti e dipendenze
  - Tool per il reperimento automatico da *repository* dei pacchetti necessari

Pacchetti

- I pacchetti per le distribuzioni Debian e derivate (es. Ubuntu) sono in formato *.deb*
  - aptitude-0.2.15.9-2_i386.deb



- I pacchetti per le distribuzioni RedHat e derivate (es. CentOS, Fedora) sono in formato *.rpm*
  - httpd-2.4.6-45.el7.centos.x86_64.rpm

Repository

- I pacchetti possono essere scaricati e gestiti singolarmente
- Normalmente però si usano i repository (repo)
  - raccolte indicizzate di pacchetti
  - possono essere online o su filesystem locali
- I *package manager* leggono per ogni repo l'indice e i metadati dei pacchetti
  - conoscono quali versioni sono disponibili per ogni pacchetto
  - conoscono le dipendenze tra pacchetti (e quindi come risolverle)
- Collocazioni delle liste di repo ed esempi:
  - `/etc/apt/sources.list` `/etc/apt/sources.list.d/*`

```
deb http://archive.ubuntu.com/ubuntu bionic-updates universe
```

  - `/etc/yum.conf` `/etc/yum.repos.d/*.repo`

```
[base]
name=CentOS-$releasever - Base
mirrorlist=http://mirrorlist.centos.org/?
release=$releasever&arch=$basearch&repo=os&infra=$infra
#baseurl=http://mirror.centos.org/centos/$releasever/os/$basearch/
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7
```

Gestione dei pacchetti .deb

database location:	`/var/lib/dpkg, /var/lib/apt`
sources file:	`/etc/apt/sources.list`
update sources:	`apt update`
key management:	`apt-key`
search:	`apt search keywords`
install:	`dpkg -i filename.deb` `apt install packagenames`
upgrade	`apt upgrade [packagenames]`
remove	`dpkg -r packagename` `apt remove packagenames`

Gestione dei pacchetti .rpm

database location:	`/var/lib/rpm`
sources file:	`/etc/yum.conf`
update sources:	`yum update`
key management:	`rpm --import keyfile`
search:	`yum search keywords`
install:	`rpm -i filename.rpm` `yum install packagenames`
upgrade:	`yum upgrade [packagenames]`
verify integrity:	`rpm -V [packagenames|a]`
remove:	`rpm -e packagenames` `yum remove packagenames`

Verifica dell'autenticità

- La firma dei pacchetti è gestita centralmente
- I *mantainer* di una distribuzione forniscono le chiavi di verifica nei media di installazione ufficiali o sui *repository* online
- I set di chiavi possono essere gestiti in modo standard con GnuPG

  es. .deb-based mettono gpg keyrings in `/etc/apt/trusted.gpg.d/`

- ... ma è più comune usare strumenti forniti dalla distribuzione

.deb `apt-key {add file | list | del keyid | adv --recv-key keyid | ... }`

.rpm `rpm {--import | -e | -q[ai] | ...}`

rpm tratta le chiavi come se fossero pacchetti
→ si possono usare gli stessi
comandi per interrogarli, eliminarli, ecc.

Lavorare coi repository

- Un'esigenza molto comune è quella di installare software ben supportato ma non incluso per qualsiasi motivo nei canali ufficiali della distribuzione
- Si aggiunge semplicemente il repository all'elenco
  - Apt (deb):

    ```
    /etc/apt/sources.list.d/virtualbox.list :
    deb http://download.virtualbox.org/virtualbox/debian xenial contrib
    ```

  - Yum (rpm):

    ```
    /etc/yum.repos.d/epel.repo :
    [epel]
    name=Epel Linux -
    baseurl=http://mirror.example.com/repo/epel5_x86_64
    enabled=1
    gpgcheck=0
    ```

Gestire la provenienza dei pacchetti

- Si può generare confusione se un pacchetto con lo stesso nome è presente in versioni diverse in repository differenti
  - I package manager, di default, scelgono sempre la versione più avanzata
  - supponiamo di aver aggiunto un repo semisconosciuto per installare un'applicazione innocua
  - se a tale repo viene aggiunto un pacchetto "core" dichiarato più recente della versione ufficiale → software injection!
- In alcuni casi anche aggiornamenti nello stesso repo sono indesiderabili
  - situazioni legacy
- La situazione va controllata e gestita
  - Controllo della provenienza di un pacchetto
    - Yum: `repoquery -i [package name]`
    - Apt: `apt-cache showpkg [package name]`
  - Elenco dei pacchetti provenienti da un repo
    - Yum: `yum list installed | grep [repo name]`
    - Apt: vari comandi per estrarre manualmente info dai file della cache

Limitare le modifiche automatiche

- Per evitare a priori problemi in sistemi con dipendenze complesse (ad esempio mix di pacchetti installati manualmente e via package manager)
  - Version locking/pinning
    - Apt

      editare **`/etc/apt/preferences.d/*`**
      - https://wiki.debian.org/AptPreferences
    - Yum

      `yum install yum-plugin-versionlock`

      poi

      `yum versionlock [package name]`

      o editare a mano

      `/etc/yum/pluginconf.d/versionlock.list`

deb e rpm

- Link per deb
  http://www.debian.org/doc/manuals/debian-reference/ch02.en.html
  https://guide.debianizzati.org/index.php/Introduzione_all%27APT_System
- Link per rpm
  http://yum.baseurl.org/wiki/YumCommands
  http://yum.baseurl.org/wiki/RpmCommands

Gestione dei processi

Convenzioni

- Il font `courier` è usato per mostrare ciò che accade sul sistema; i colori rappresentano diversi elementi:

  `rosso per comandi da impartire o nomi di file`

  `blu per l'output dei comandi`

  `verde per l'input (incluse righe nei file di configurazione)`

- Altri colori possono essere usati in modo meno formale per evidenziare parti da distinguere nei comandi o indicazioni importanti nel testo

- I parametri formali sono normalmente scritti in maiuscolo e riportati nello stesso colore nel testo che ne descrive l'utilizzo

Principi di shell scripting

- Bash può essere usata per programmare task da eseguire automaticamente anziché dover impartire comandi a mano
- Ci sono due aspetti importanti da tenere a mente rispetto a un linguaggio di programmazione come C o Java

1) Gli elementi di base gestiti da bash sono file e processi

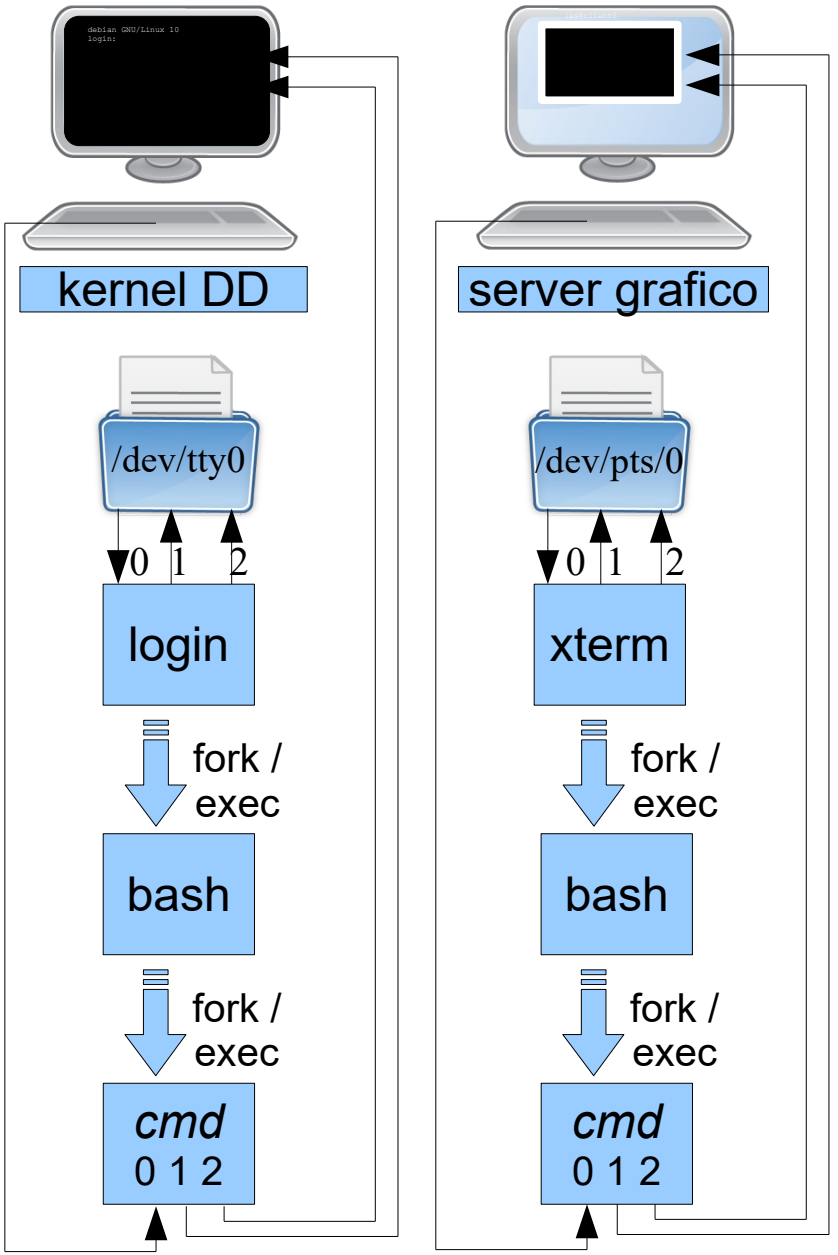
bash ha come scopo fondamentale l'avvio di processi, la predisposizione delle comunicazioni tra loro e col filesystem, il controllo dello stato in uscita. È fondamentale pensare sempre, quando si scrive o si analizza una riga di comando, a quali processi verranno eseguiti e a quali file possono essere coinvolti

2) Il linguaggio di bash è interpretato, non compilato

Il significato dato a molti caratteri è sintattico, non letterale, e la riga di comando effettivamente eseguita risulta da un procedimento, detto espansione, che individua sottostringhe speciali contrassegnate da caratteri speciali, e le sostituisce col risultato di una corrispondente elaborazione

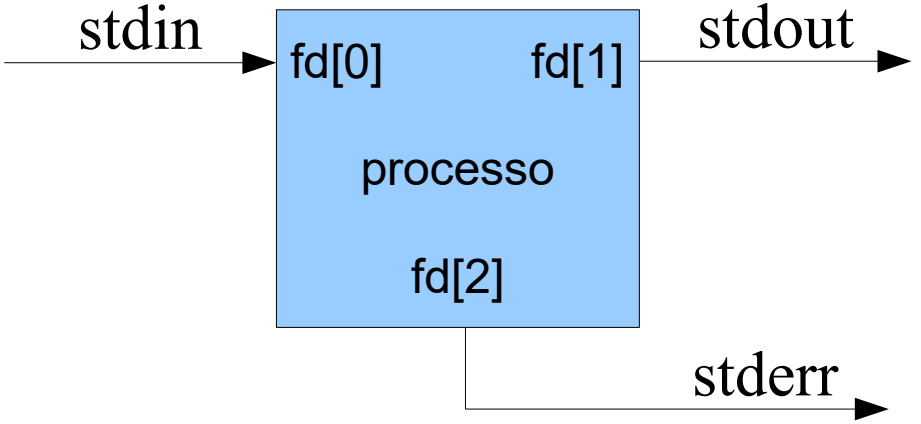
Stream, shell, terminale e lancio di programmi

- All'avvio il kernel inizializza i dispositivi HW e li espone come
  - /dev/tty* (terminali virtuali che accedono direttamente a console)
  - /dev/pts/* (terminali che accedono a finestre grafiche)
- Il device driver che gestisce tali file
  - vi rende disponibili per la lettura i caratteri digitati da tastiera
  - preleva i caratteri che vi vengono scritti e li visualizza a schermo
- Viene avviato un processo di gestione del terminale
  - apre in lettura il file speciale
    → assegnato file descriptor 0
  - apre in scrittura due volte il file speciale
    → assegnati file descriptor 1 e 2
- "qualcuno" istruisce l'avvio di bash
  - eredita i f.d. quindi comunica col terminale
- si lancia un comando da bash
  - (di default) eredita i f.d. quindi comunica col terminale



Elementi di base – stream e ridirezione

- Per convenzione quindi tutti i comandi *nix che operano su stream di testo (filtri) sono progettati per disporre di tre stream con cui comunicare con il resto del sistema:
  - standard input in ingresso (file descriptor 0)
  - standard output in uscita (file descriptor 1)
  - standard error in uscita (file descriptor 2)



Premessa: shell expansion

La shell opera secondo un procedimento di *espansione*

- Individua sequenze speciali contrassegnate da *meta-caratteri*, che non vengono presi a valore nominale
- Interpreta il significato della sequenza speciale
- Al posto della sequenza mette il risultato dell'interpretazione, creando una riga di comando diversa da quella digitata
  - Se un'espansione fallisce (ad esempio la sequenza speciale è mal formata, o dipende dalla presenza di dati che a tempo di esecuzione mancano) la sequenza è solitamente lasciata inalterata sulla riga di comando
- Ci sono ben 12 passi che svolgono manipolazioni diverse della riga di comando, in una sequenza precisa
- Alcuni/tutti possono essere saltati per mezzo del *quoting*, cioè proteggendo i meta-caratteri da non interpretare, per mezzo di altri caratteri speciali: apici ' , doppi apici " , backslash \
  - Uso minimale del quoting: evitare che gli spazi vengano interpretati dalla shell come separatori tra comandi e argomenti

Riga di comando da espandere

- Ogni comando può essere preceduto da assegnamento di valore a variabili
  - es. `A=40 mycommand | othercommand > outfile`
  - queste parti vengono temporaneamente accantonate
- Bash passa all'espansione degli elementi della riga di comando come descritto nel seguito
- Bash predispone le ridirezioni
- Bash riprende gli assegnamenti accantonati
  - ogni parte di testo dopo '=' viene sottoposta (vedi seguito) a
    - tilde expansion
    - parameter expansion
    - command substitution
    - arithmetic expansion
    - quote removal

    e assegnata alla variabile corrispondente
- Vengono eseguiti i comandi

Ridirezione da/verso file

- Bash, nell'interpretare la riga di comando, può disconnettere gli stream predefiniti dal terminale (chiudendoli nel processo figlio dopo la fork) e far trovare gli stessi file descriptor aperti su di un file diverso (aprendolo prima della exec)
- Ridirezione dello stdout: `>` e `>>`
  - `ls > miofile` scrive lo stdout di ls nel file miofile (troncandolo)
  - `ls >> miofile` scrive lo stdout di ls nel file miofile (in append)
  - se miofile non esiste viene creato
- Ridirezione dello stderr: `2>` e `2>>`
  - come sopra ma ridirige lo sdterr
- Confluenza degli stream
  - `ls > miofile 2>&1` ridirige lo stderr dentro stdout e poi stdout su file l'ordine è importante!
- Ridirezione dello stdin `<`
  - `sort < miofile` riversa il contenuto di miofile su stdin di sort

Ridirezioni speciali

- Here documents – inviare direttamente un testo a un comando

```
comando <<MARCATORE
questo testo
va tutto
a finire
sullo stdin
di comando
MARCATORE
```

- Per una singola linea non serve il marcatore

```
comando <<< "testo da mandare a stdin di comando"
```

Ridirezioni speciali

- Se si vogliono ridirigere stream definitivamente si può usare exec [ridirezione]

  Es. `exec 2>/dev/null`
  - tutti i comandi eseguiti da qui in poi avranno stderr riversato su /dev/null
  - non pratico interattivamente (la shell usa stderr per mostrare prompt e echo di quel che scrivete!) ma utilissimo negli script
  - può essere ripristinato al settaggio originale con `exec 2>&-`
- Con exec si possono creare anche nuovi fd
  - utile perché i fd aperti vengono ereditati dai processi figli

  Es. `exec 3< filein 4> fileout 5<> filerw`
  - da ora in avanti
    - ogni lettura dal fd 3 fatta con `<&3` leggerà da filein
    - ogni scrittura fatta con `>&4` sul fd 4 scriverà su fileout
    - il fd 5 può essere usato sia per leggere che per scrivere su filerw
  - per chiudere: `exec 3>&- 4>&- 5>&-`

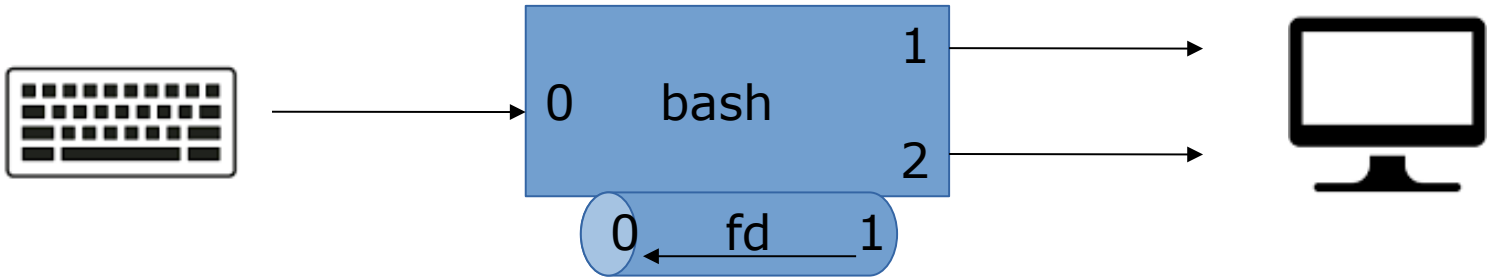
bash pipe

- Cosa succede quando si esegue

  `ls | sort`

- Bash prepara il terreno perché ciò che ls produce su stdout venga riportato su stdin di sort

  1) Call di **`pipe(fd[ ])`**



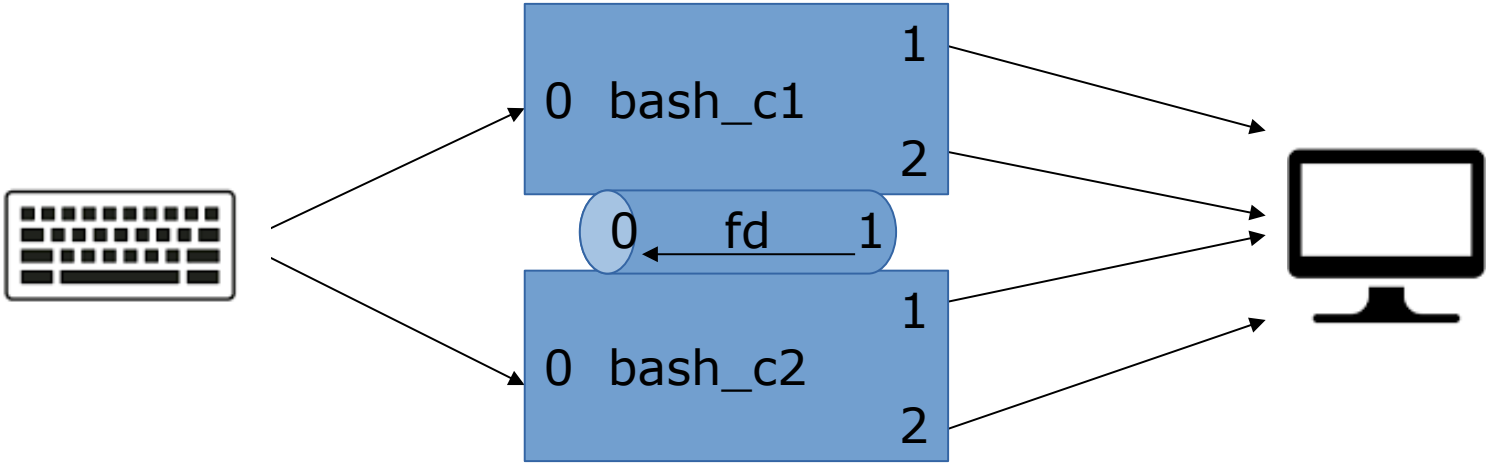
bash pipe

- Cosa succede quando si esegue

  `ls | sort`

- Bash prepara il terreno perché ciò che ls produce su stdout venga riportato su stdin di sort

  2) Call (due volte) di **`fork`**



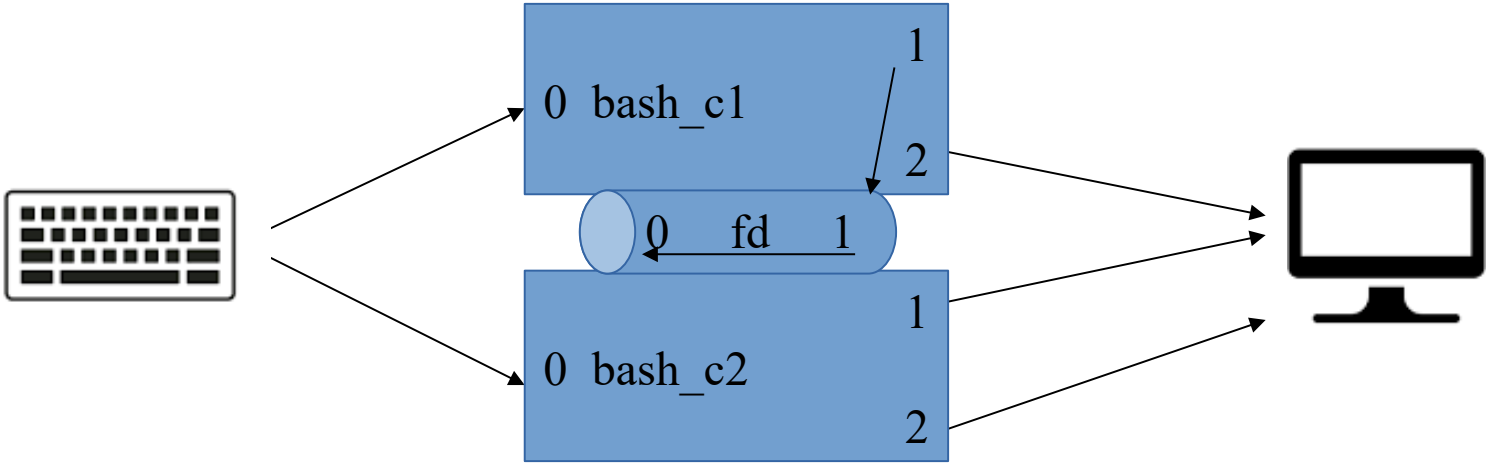
bash pipe

- Cosa succede quando si esegue

  `ls | sort`

- Bash prepara il terreno perché ciò che ls produce su stdout venga riportato su stdin di sort

  3) Child 1 chiama **`dup2(fd[1], 1)`** – taglia lo stream stdout, crea un duplicato dell'estremità scrivibile della pipe, e gli assegna il file descriptor 1 (stdout)



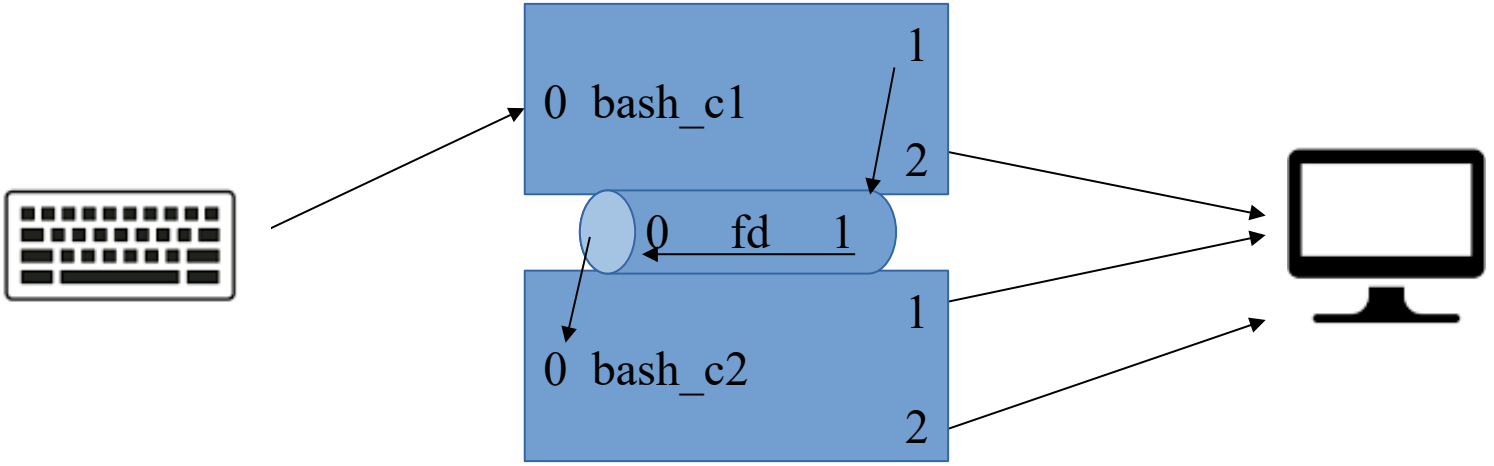
bash pipe

- Cosa succede quando si esegue

  `ls | sort`

- Bash prepara il terreno perché ciò che ls produce su stdout venga riportato su stdin di sort

  4) Child 2 chiama **`dup2(fd[0], 0)`** – taglia lo stream stdin, crea un duplicato dell'estremità leggibile della pipe e gli assegna il file descriptor 0 (stdin)



  5) Child 1 chiama **`close(fd[0])`** e child 2 chiama **`close(fd[1])`** per evitare utilizzi incoerenti della pipe

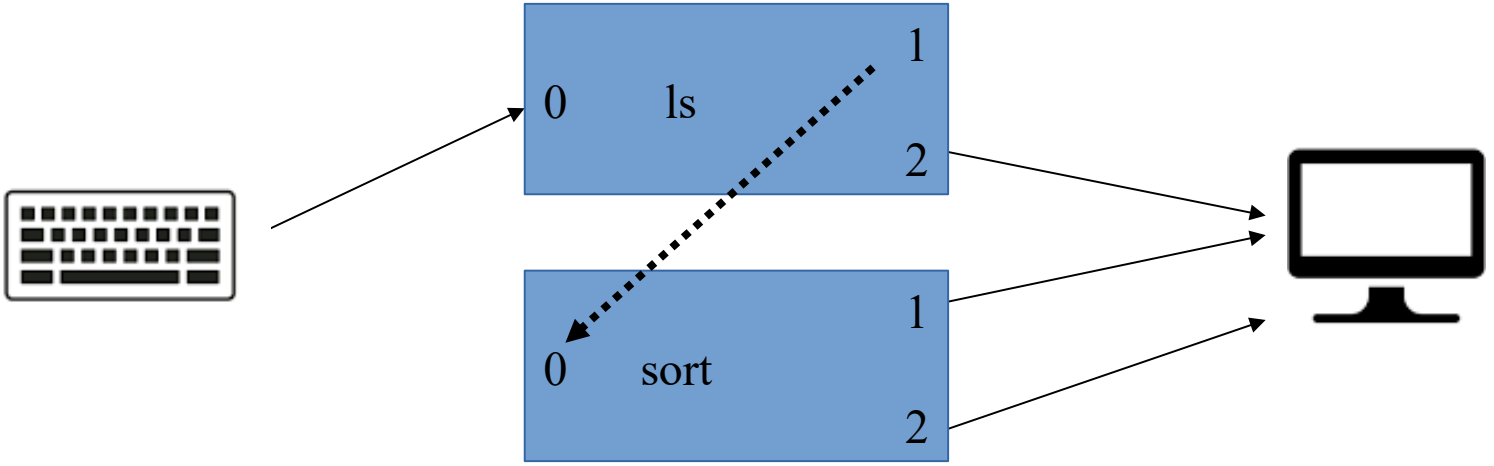
bash pipe

- Cosa succede quando si esegue

  `ls | sort`

- Bash prepara il terreno perché ciò che ls produce su stdout venga riportato su stdin di sort

  6) Child 1 chiama **`exec("ls")`** e child 2 chiama **`exec("sort")`** - I nuovi programmi prendono vita e usano I loro stream standard senza bisogno di sapere a cosa sono connessi. Il sistema operativo implementa buffering, sincronizzazione, e signali per le eccezioni



Interazione coi processi

- Ogni comando lanciato da shell o dal sistema diviene un *processo*. I processi sono identificati
  - globalmente nel sistema da un numero univoco (Process ID o PID)
  - in aggiunta, in alcuni casi, da un Job ID valido localmente alla shell
- Un processo svolge le proprie azioni a nome dell'utente che lo ha lanciato (i processi lanciati da root hanno il potere di assumere l'identità di altri utenti, così facendo si "declassano" e perdono il potere di tornare indietro)
- I processi anche non lanciati da una stessa pipeline possono comunicare tra loro
  - per mezzo di sistemi da predisporre appositamente (named pipe, socket)
  - in modo più limitato ma semplice per mezzo dei segnali

Segnali – caratteristiche di base

- I segnali sono **eventi asincroni** notificati dal kernel a un processo
  - generati dal kernel stesso (eventi hardware, eccezioni durante IPC, ecc.)
  - generati da un altro processo
    - eseguito dallo stesso utente del destinatario (o da root)
  - Il contenuto informativo è limitato a un numero
- Ricezione:
  - il controllo dei segnali ricevuti avviene ogni volta che il processo rientra in user space (es. dopo una syscall o quando schedulato sulla CPU)
  - se tra un controllo e il successivo sono stati ricevuti più segnali diversi, vengono posti in uno stato "pending"
    - l'ordine in cui verranno presi in considerazione non è specificato
    - pending non è una coda: che ne arrivi uno o più (dello stesso tipo) il flag sarà semplicemente settato
- Gestione (a livello di sistema operativo):
  - Ogni processo può "registrare" presso il sistema operativo una **routine di gestione** (**handler**) per un segnale.
  - alla rilevazione di un segnale pending, il flusso di esecuzione del processo a cui è destinato viene interrotto e viene eseguito l'handler
  - durante l'esecuzione dell'handler, i segnali dello stesso tipo sono bloccati
    - non causano esecuzioni annidate dell'handler ma settano il flag pending

Signal disposition

- Il comportamento di un processo alla ricezione di un segnale può essere
  - terminare (eventualmente con core dump)
  - ignorarlo
  - sospendersi (stato stop)
  - riprendersi da stop (cont)
- Vedere signal (7) per l'elenco delle disposition predefinite
- La disposition può essere modificata da un processo
  - tre possibili scelte
    - attuare quella di default
    - ignorare il segnale
    - eseguire un handler
  - fanno eccezione i segnali KILL e STOP, che non possono essere bloccati, ignorati, o intercettati da un handler personalizzato

Invio di segnali

- Per inviare un segnale a un processo si può usare

  `kill [options] <pid> […]`

  - PID negativi identificano l'intero process group
  - l'opzione `-l` / `-L` elenca i segnali supportati

- Il terminale trasforma la ricezione di alcune combinazioni di tasti in segnali inviati al processo che lo sta occupando:

  `Ctrl + Z` → SIGTSTP

  `Ctrl + C` → SIGINT

  `Ctrl + \` → SIGQUIT

  - osservazione a lato: il terminale genera anche altri effetti di controllo non legati ai segnali, come `eof` = `^D`; `start` = `^Q`; `stop` = `^S`;

sleep

- Il comando `sleep` innesca un timer per far "dormire" il processo
- Il parametro può essere un float
  - di default interpretato in secondi
  - sono supportati i suffissi m(inutes) h(ours) d(ays)
- Interazioni coi segnali – valgono le regole di qualsiasi altro comando lanciato dalla shell
  - sleep è un comando esterno
    → genera un processo figlio
    → mandare un segnale alla shell che lo ha lanciato non lo tocca
  - sleep invoca una system call che sospende il processo
    → fino al termine della sleep il processo non rientra in user mode
    → i segnali sono ricevuti ma non processati

Processi in background

- Si può usare un'unica shell per l'esecuzione contemporanea di più comandi che non abbiano necessità di accedere al terminale, lanciandoli in **background** (sullo sfondo).

- Questo si ottiene postponendo il carattere `&` alla command line.
  - La shell risponde comunicando un numero tra parentesi quadre (**job id**) che identifica il job **localmente** a questa shell.
    - per usarlo al posto di un PID, si utilizza `%job_id`
  - **MOLTO UTILE:** Il PID del processo viene memorizzato nella variabile `$!`

- Se si lancia una command line senza **&**, e si vuole rimediare, si può dare un segnale di **STOP** con Ctrl+Z.
  - Anche in questo caso si riceve un job id.
  - Con il comando `bg %job_id`, si invia un segnale CONT che riavvia il processo e contemporaneamente lo si mette in background.

wait

- Il builtin `wait` permette di bloccare l'esecuzione fino al completamento dei job in background
  - di default attende il completamento di tutti i job
  - si possono passare come argomento job_id specifici
- Se durante l'attesa la shell riceve un segnale per il quale è definito un handler con trap
  - wait esce immediatamente con exit code > 128
  - l'handler viene eseguito
  - l'esecuzione prosegue dopo la wait
    - si può controllare in $? l'exit code di wait per capire cosa l'ha terminata

jobs e foreground

- Un processo in background non riceve più comandi dal terminale, poiché la tastiera torna ad agire sulla shell;
  - continua però a utilizzare il terminale per STDOUT e STDERR
- se è necessario riportare in **foreground** (primo piano) un processo ricollegandolo così al terminale, si usa il comando `fg %job_id`.

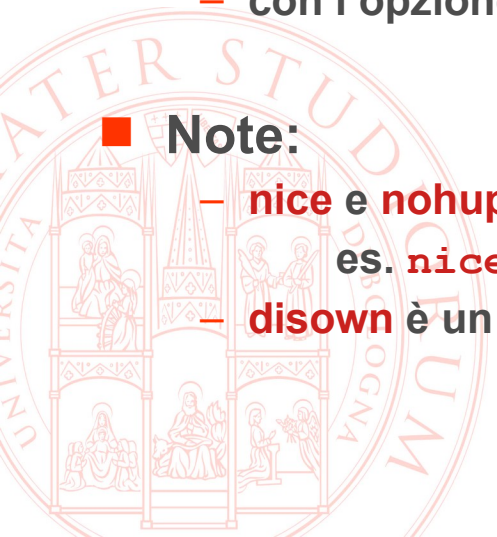
- Il comando `jobs` mostra l'elenco dei job, cioè di tutti i processi avviati dalla shell corrente, indicando il loro stato (attivo o stoppato).

- Per esempi e approfondimenti sulla propagazione di segnali a child process:

  https://linuxconfig.org/how-to-propagate-a-signal-to-child-processes-from-a-bash-script

Modificatori per processi in bg

- `nohup <comando>` evita che la shell, alla chiusura, invii il segnale SIGHUP al <comando> (il che normalmente ne causerebbe la terminazione)
  - provvede, inoltre, a scollegare l'output del processo dal terminale se non fatto esplicitamente nell'invocazione.
  - di default, nohup dirige l'output sul file 'nohup.out'
- `nice <comando>` lancia <comando> con una *niceness* diversa da zero, modificando la priorità del processo
  - di default 10
  - valori negativi (che incrementano la priorità) sono utilizzabili solo da root
- `disown` rimuove completamente un job dalla job table della shell
  - di default quello lanciato per ultimo
  - con l'opzione `-h` implementa anche l'immunità all'hangup



- Note:
  - **nice** e **nohup** sono comandi esterni e usati all'avvio di un processo, anche insieme
    es. `nice nohup long_calculation &`
  - **disown** è un builtin che agisce su PID/job_id di processi lanciati in precedenza

ps

- ps (1) supporta un numero strabiliante di opzioni, perché è compatibile con ben tre sintassi
  - Unix, singole lettere precedute da singolo trattino
  - BSD, singole lettere, senza trattino
  - estensioni GNU, parole precedute da doppio trattino
- Le opzioni delle tre famiglie possono essere mescolate nello stesso comando, a meno di non creare contraddizioni o ambiguità (...)
- Per gli usi più comuni, ci sono esempi collocati all'inizio della man page
- Suggerimenti – andiamo a leggere la man page
  - la sezione PROCESS SELECTION BY LIST mostra come ottenere una lista di processi secondo le loro proprietà (es. comando lanciato, pid, utente, ecc.)
    - è molto meglio usare queste opzioni che non "greppare" l'intera lista di processi!
  - la sezione OUTPUT FORMAT CONTROL illustra come formattare la lista prodotta
    - in particolare le opzioni (equivalenti) `-o`, `o`, `--format` seguite da una stringa di specificatori documentata nella sezione STANDARD FORMAT SPECIFIERS permettono un controllo completo sui campi che si vogliono far comparire nella lista
  - la sezione PROCESS STATE CODES spiega il significato della colonna STAT e dà un'indicazione fondamentale dello stato del processo

uptime

- uptime prende il nome dal primo elemento di output
- riporta anche il *carico* del sistema
  - ad ogni invocazione dello scheduler viene registrato il numero totale di processi in stato R (runnable) o D (uninterruptable sleep)
  - i campioni vengono accumulati e mediati su tre scale temporali diverse per ottenere un'indicazione del trend nel tempo
    - lo stato di "salute" va valutato in confronto al numero di processori disponibili

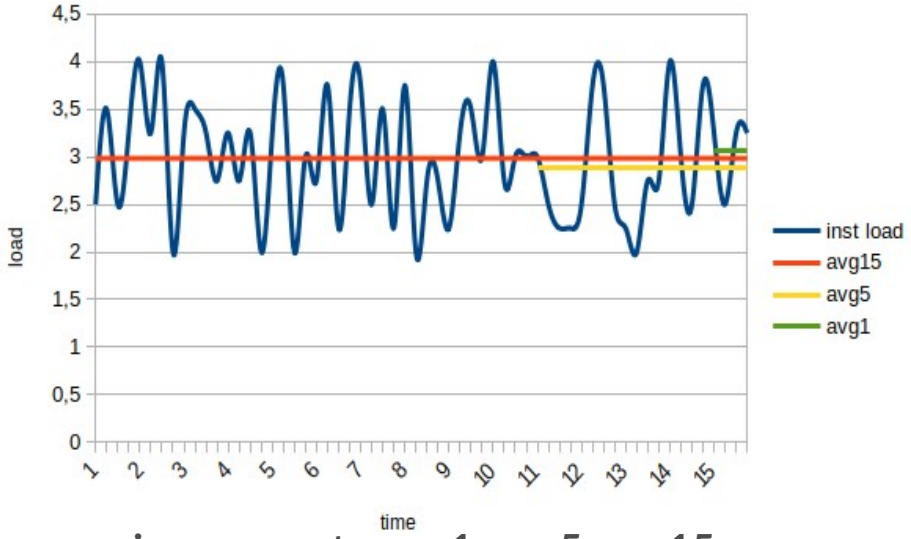
```
21:27:56 up  7:10,  2 users,  load average: 0.00, 0.00, 0.00
```

ora n. utenti connessi carico medio negli ultimi...

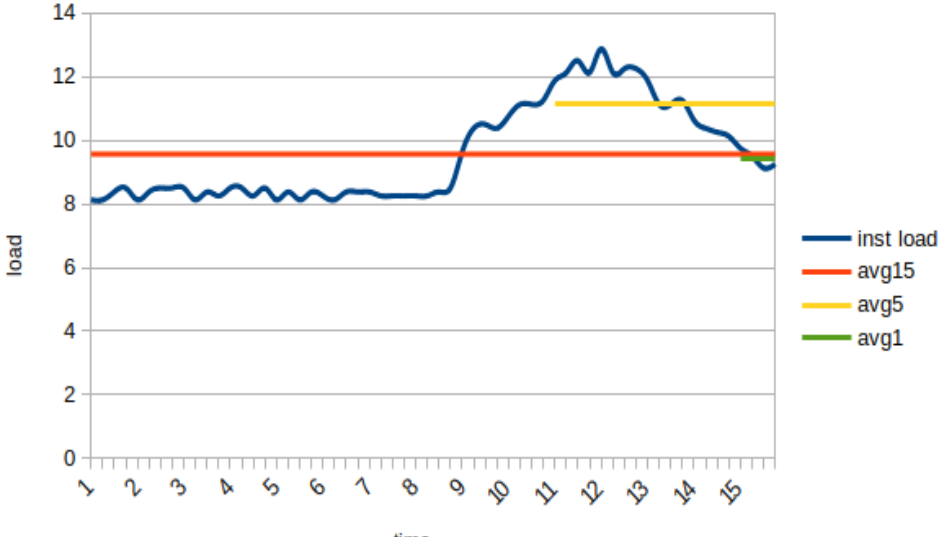
tempo trascorso dal boot 1' 5' 15'

i tre carichi di uptime: esempi di interpretazione

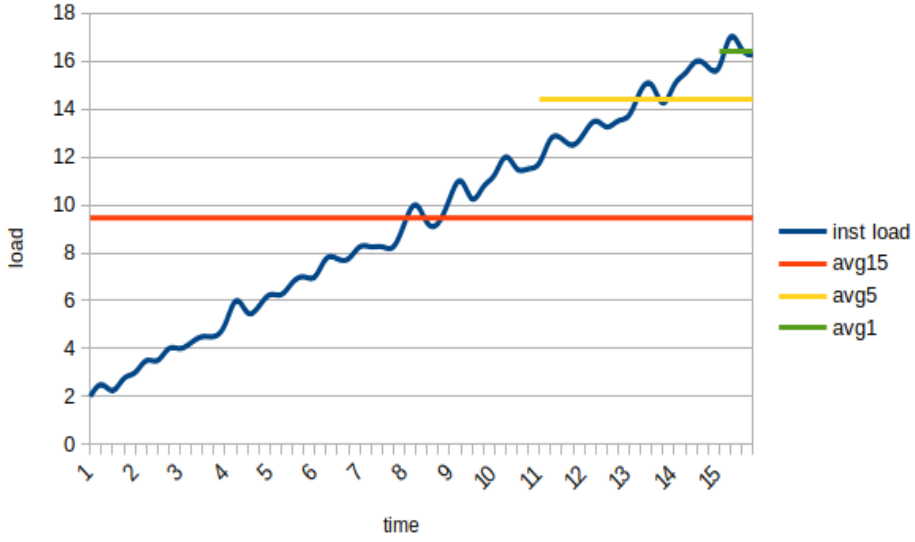
carico costante: avg1≈avg5≈avg15



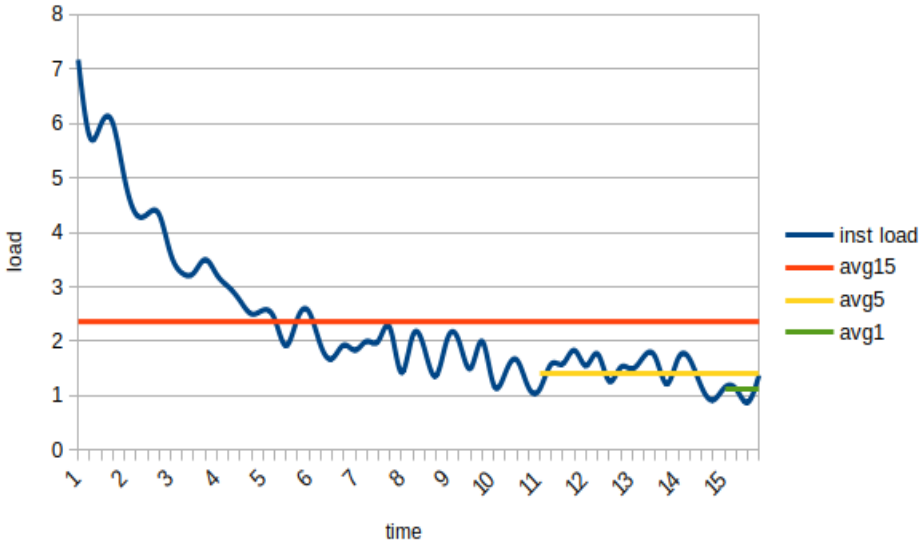
picco recente: avg5>(avg1≈avg15)



carico crescente: avg1>avg5>avg15



carico calante: avg1<avg5<avg15



free

```
las@client:~$ free
               total        used        free      shared  buff/cache   available
Mem:          237040      121248        9596        1464      106196       98720
Swap:        1045500       29572     1015928
```

- la maggior parte della memoria usata per cache può essere liberata per usi prioritari, da cui `available ≈ free + buff/cache`
  - l'impatto sulle prestazioni della rinuncia alle cache non è nullo
- `used swap > 0` significa solo che in qualche momento è servita

ps – uptime – free → top

- Comandi che scattano un'istantanea del sistema
  - ps: stato dei processi
  - uptime: carico del sistema
  - free: occupazione memoria
- Comandi di monitoraggio interattivi
  - top riassume ps, uptime, free + uso dettagliato cpu
  - aggiornato regolarmente
  - permette di interagire coi processi
  - utile per stima intuitiva dello stato di salute

top

```
9:31am  up 50 min,  2 users,  load average: 0.02, 0.02, 0.04
71 processes: 70 sleeping, 1 running, 0 zombie, 0 stopped
CPU states:  4.3% user,  5.2% system,  0.1% nice, 90.2% idle
Mem:   384480K av,  380688K used,    3792K free,    1312K shrd,   51312K buff
Swap:  128516K av,       0K used,  128516K free                  139136K cached

  PID USER     PRI  NI  SIZE  RSS SHARE STAT %CPU %MEM   TIME COMMAND
 1179 root      13   0  3092 3092  2592 S     2.8  0.8   0:50 magicdev
 9299 root      16   0  1044 1040   832 R     2.8  0.2   0:00 top
    1 root       8   0   520  520   452 S     0.0  0.1   0:03 init
    2 root       9   0     0    0     0 SW    0.0  0.0   0:00 keventd
    3 root       9   0     0    0     0 SW    0.0  0.0   0:00 kapm-idled
    4 root      19  19     0    0     0 SWN   0.0  0.0   0:00 ksoftirqd_CPU0
    5 root       9   0     0    0     0 SW    0.0  0.0   0:00 kswapd
    6 root       9   0     0    0     0 SW    0.0  0.0   0:00 kreclaimd
    7 root       9   0     0    0     0 SW    0.0  0.0   0:00 bdflush
    8 root       9   0     0    0     0 SW    0.0  0.0   0:00 kupdated
    9 root      -1 -20     0    0     0 SW<   0.0  0.0   0:00 mdrecoveryd
   71 root       9   0     0    0     0 SW    0.0  0.0   0:00 khubd
  465 root       9   0     0    0     0 SW    0.0  0.0   0:00 eth0
  546 root       9   0   592  592   496 S     0.0  0.1   0:00 syslogd
  551 root       9   0  1124 1124   448 S     0.0  0.2   0:00 klogd
  569 rpc        9   0   592  592   504 S     0.0  0.1   0:00 portmap
  597 rpcuser    9   0   788  788   688 S     0.0  0.2   0:00 rpc.statd
```