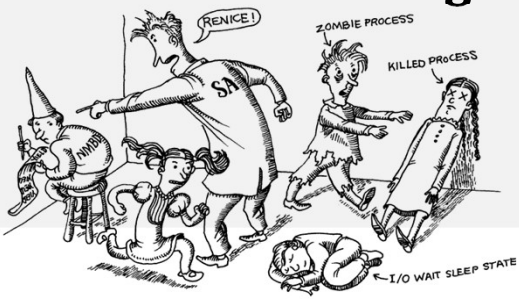
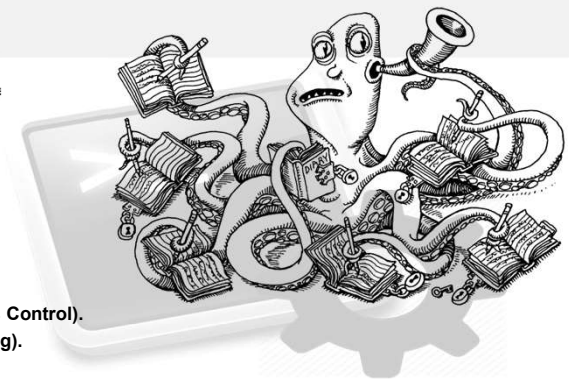
Resource Management & Logging

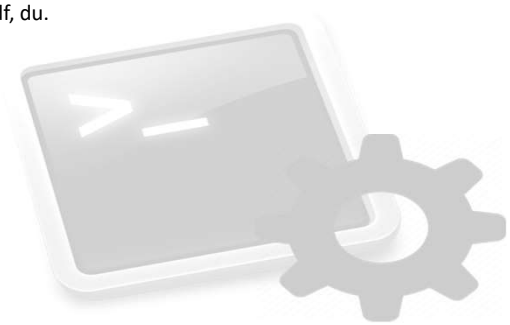




Reference Material:
[1] UNIX and Linux System Administration Handbook, Chapter 4 (Process Control).
[2] UNIX and Linux System Administration Handbook, Chapter 10 (Logging).
[3] Systems Performance: https://www.brendangregg.com

Index

- **Introduction: System Performance**
  - Basic aspects
- **Linux Processes (resource consumers)**
  - The /proc FileSystem
  - Runaway Processes
- **Resource Monitoring**
  - Command-line utilities: ps, top, vmstat, uptime, strace, free, df, du.
  - Event gathering (logging) through systemd journal.
- **Resource Management**
  - Process Management: signals and priority
  - Controlling Shell process
  - Memory Management: Swapping
  - Disk Management: Quotas
- **Periodic Processes**
  - cron/at
  - systemd timers



System Performance

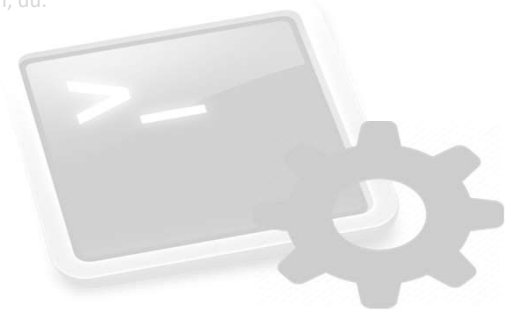
- Study of application, operating system, kernel and hardware performance.
- It is an important skill for any computer user:
  - Why is my laptop slow?
  - How can I optimize the performance of a large-scale compute environment (your company's cloud infrastructure).

- As an Administrator, why should I care about performance?
  - **Financial perspective:** improving price/performance is a key aspect in cloud environments.
  - **Future planning**: capacity planning, bottleneck elimination, scalability analysis, etc.
  - **Multi-user management**: avoid abusive behaviors of certain users concerning shared resources.

- This chapter, focused on basic supervision aspects to ensure a fair distribution among users.

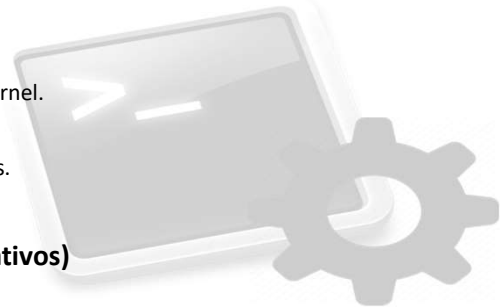
Index

- Introduction: System Performance
  - Basic aspects
- **Linux Processes (resource consumers)**
  - The /proc FileSystem
  - Runaway Processes
- Resource Monitoring
  - Command-line utilities: ps, top, vmstat, uptime, strace, free, df, du.
  - Event gathering (logging) through systemd journal.
- Resource Management
  - Process Management: signals and priority
  - Controlling Shell process
  - Memory Management: Swapping
  - Disk Management: Quotas
- Periodic Processes
  - cron/at
  - systemd timers



Linux Processes

- Process = Running Program ("**unit**" for memory, processor time and I/O resources **monitoring** and **management**).
- Process Components:
  - **Address Space:** addresses in the virtual memory that the process is allowed to use.
  - **Kernel Data Structures**: address space map, status of the process, priority, resources, files & network ports opened, etc.
- Process Identification:
  - **PID**: unique ID number assigned to every process by the kernel.
  - **PPID**: parent PID
  - **UID/GID**: user who created the process/ group of a process.



- **You should already know all these (Sistemas Operativos)**

The /proc FileSystem

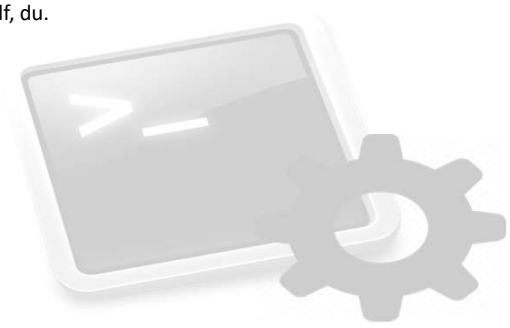
- Is process information available in Linux? Yes, through the /proc filesystem.

- **/proc**: pseudo File System representing current kernel status.
  - Details about system hardware (/proc/cpuinfo or /proc/devices)
  - Information about any process currently running.
    - cmdline: command line arguments
    - environ: env variables
    - fd: subdirectory with links to each open file descriptor
    - stat: process status information (decoded with ps)
    - maps: memory maps to executable & library files
    - mem: memory held by process
    - … (see man proc)
  - Usually, not read directly, but through specific commands (ps, vmstat, top…)

Runaway Processes

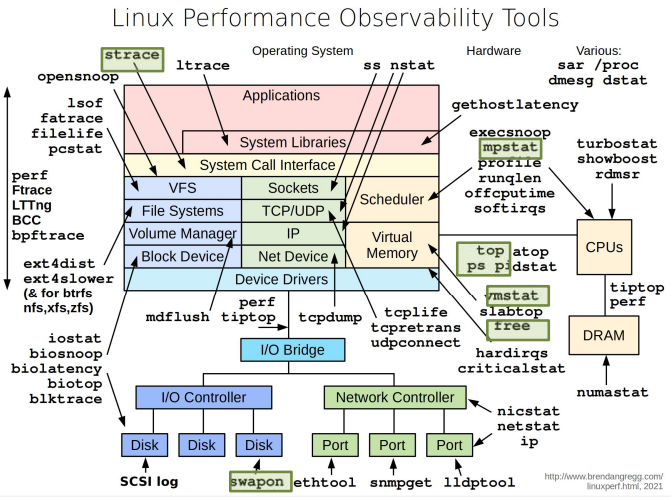
- Monitoring and controlling system and user processes is a critical aspect for correct system status.
- Runaway process: those that can exhaust system's CPU, memory or disk resources.
  - program with bugs.
  - Inefficient behavior (uncontrolled logging)
  - Malicious/fool users (while 1; echo "disk consuming" >> file.txt)
- How to deal with these processes?
  - **Prevention**: Limit the amount of resources available for user processes.
  - **Monitoring** & **correction**: periodic supervision and problem handling.

Index



Resource Monitoring (commands)

- Being a critical aspect, the amount of available shell tools is huge…
- We limit to the most basic ones (highlighted in green boxes)



Resource Monitoring (commands)

- **Command ps**: main tool for monitoring processes (per-process information).
  - **ps aux**: useful overview of all processes running on the system

```
[ (SI) root core ~] ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1    7.2  0.3  56928  6652 ?        Ss   13:10   0:00 /sbin/init
…
root       428  1.0  0.2  20984  4756 tty1     S    13:10   0:00 -bash
```

Virtual size of the process

Resident set size (pages in memory)

CPU time consumed by the process

Process Status: Runnable (R), Sleeping (S), Zombie (Z),…

  - **ps lax**: extended information such as parent process ID (PPID), Niceness (NI), resource on which the process is waiting (WCHAN).
  - pstree: tree-like version of ps.
- **Command top (htop)**: real-time version of ps (regularly updated).
  - Interactive “shell” , can kill processes, modify priorities, etc. (h for extended help).

Resource Monitoring (commands)

- Command **vmstat** (mpstat): **global** instantaneous utilization.
  - Syntax: #vmstat <interval> <samples>
  - **man vmstat** for a detailed description of each column.

```
[ (SI) root core ~] vmstat 5 3
procs -----------memory----------   ---swap-- -----io---- --system-- ----cpu----
 r  b   swpd   free   buff  cache     si   so    bi    bo   in    cs us sy id wa
 0  1 132908 2106268  22712 1609316    0    0    15     1   11    17  3  1 92  4
 0  1 132908 2106028  22832 1609468    0    0     0  1254 1730   983  0  1 80 19
 0  1 132908 2105980  22952 1609620    0    0     2   302 1599   553  0  1 84 15
```

- Command **uptime**: global average utilization.
  - Reports the following information: Current time, how long running, users logged on and system load (Average load values in 1, 5 and 15 minutes).
  - Load: num of processes in runnable (running or waiting to run) or I/O Waiting state
    - Not a normalized value, Load=1 does not mean the same for 1CPU or 4CPU systems.

Resource Monitoring (commands)

- Command **mpstat**: CPU utilization report (information available per core)

```
[ (SI) root core ~] mpstat -P ALL
15:27:20     CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
15:27:20     all    0,05    0,00    0,11    0,03    0,00    0,17    0,00    0,00    0,00   99,65
15:27:20       0    0,07    0,00    0,08    0,03    0,00    0,29    0,00    0,00    0,00   99,52
15:27:20       1    0,03    0,00    0,14    0,02    0,00    0,05    0,00    0,00    0,00   99,77
```

- Command **free**: information about Main Memory availability

```
[ (SI) root core ~] free
              total        used        free      shared  buff/cache   available
Mem:        2039660       42912     1434536        2976      562212     1839008
Swap:       2095100           0     2095100
```

- Command **iostat**: input/output statistics for devices and partitions

```
[ (SI) root core ~] iostat -p ALL
Device             tps    kB_read/s    kB_wrtn/s    kB_dscd/s    kB_read    kB_wrtn    kB_dscd
sda               0,56         9,25        24,88         0,00     124935     336188          0
sda1              0,01         0,16         0,00         0,00       2117          0          0
sda2              0,53         8,76        24,88         0,00     118365     336188          0
sda3              0,01         0,17         0,00         0,00       2296          0          0
sr0               0,00         0,00         0,00         0,00          1          0          0
```

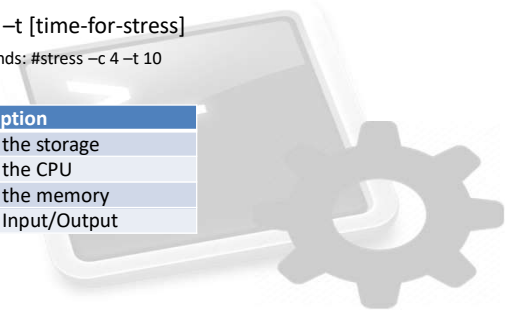
Resource Monitoring (commands)

- **Disk Monitoring**:
  - Command **df**: % of occupation of the mounted file systems.
    - Syntax: #df -<options> (option –h for human readable sizes).
  - Command **du**: size of a branch in the File System
    - Allows us to know where is the leak of disk storage and which user is responsible
    - #du –h (only directories) vs #du –a (directories and files)
    - Example: find out the top 5 directories eating space on alumno's $HOME
      - du –a /home/alumno | sort –n –r | head –n 10

Stressing System Resources

- Command **stress**: test system performance and reliability by introducing stress (heavy load)
  - Used by system administrators to test the scalability of their system.
  - Used by kernel programmers to evaluate kernel performance
  - Used by system programmers to find bugs which usually occur when system is under heavy load.
  - Syntax: #stress [Resource-to-stress] [Number-of-workers] –t [time-for-stress]
    - Example: Stressing the CPU with 4 workers (threads) during 10 seconds: #stress –c 4 –t 10

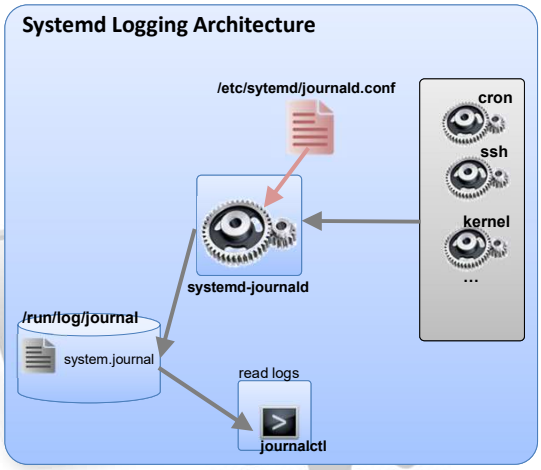
Option	Description
-d / --hdd	Stress the storage
-c / --cpu	Stress the CPU
-v / --vm	Stress the memory
-I / --io	Stress Input/Output



Resource Monitoring (Logging)

- **Once detected a problem, how can we check if some process/app/service is malfunctioning?**

- Kernel, services, apps generate/send **events** constantly.
  - Information about normal activity, **failures or other anomalies**.
  - **Failed booting** of system and services.
  - **Access** information (security).
- **Correct management of this information is essential to discover and solve the problems.**
- The events from all services have a common manager (systemd journal)



Anatomy of a Journal entry

- **journalctl** command prints messages from the journal
  - Output format can be configured through –o option
- Only the most relevant fields
  - For a complete list of fields: man systemd.journal-fields

Field	Description
MESSAGE/MESSAGE_ID	Human-readable string/128-bit identifier
PRIORITY	Priority value (0-7) How critical is your message?
SYSLOG_FACILITY	Identifier for the facility that generated the message
_PID/_UID/_GID	Info about the process that generates the entry
_BOOT_ID	In which boot was generated the message
...	Some more fields, please see systemd.journal-fields

Message classification: severity code

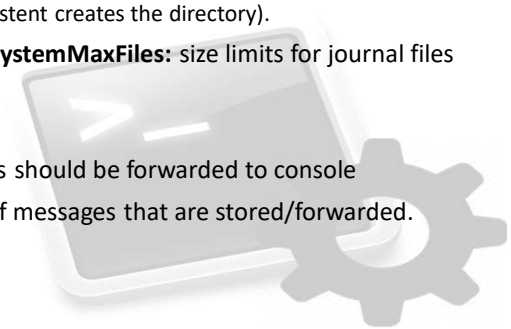
Code	Severity	Description
0	emerg	system is unusable
1	alert	should be corrected now
2	crit	critical conditions
3	err	error conditions
4	warning	error will ocurr if action not taken
5	notice	unusual events, no error
6	info	normal operation
7	debug	extra info for debugging

Message classification: who generates the entry?

Code	Keyword	Programs that use it
0	kern	kernel messages
1	user	user-level messages
3	daemon	system daemons
4,10	auth/authpriv	security/authorization messages
15	cron	scheduling daemon
16-23	local0-7	local use messages

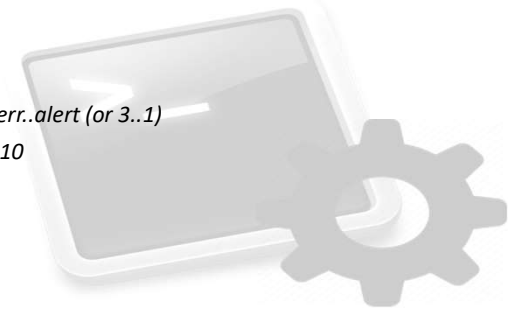
Logging Configuration

- Default configuration file is **/etc/systemd/journald.conf**
  - **This file includes a commented-out version of every option, with its default value**
- **Storage**: controls whether to save the journal to disk
  - volatile: stores the journal in memory only (none: discard all log data)
  - persistent/auto: save the journal in /var/log/journal (persistent creates the directory).
- **SystemMaxUse/SystemKeepFree/SystemMaxFileSize/SystemMaxFiles:** size limits for journal files
  - KeepFree vs. MaxUse, smaller is mandatory
  - Time limits are also available.
- **ForwardToConsole/ForwardToWall:** controls If messages should be forwarded to console
- **MaxLevelStore/MaxLevelConsole**: maximum log level of messages that are stored/forwarded.



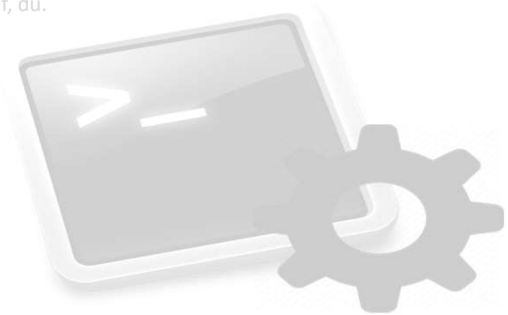
Logging Filtering

- The amount of logs is huge, How to identify precise information? FILTERING
- **Time-based** filtering:
  - Show all messages from a specific date (time optional): *journalctl –since="2020-16-01 23:00:00"*
  - Show all messages since XXX time ago: *journalctl –since "20 min ago"*.
- **Boot session** filtering:
  - Show all messages from previous boot: *journalctl –b -1*
- **Severity/Facility** filtering:
  - Show only error, critical and alert messages: *journalctl –p err..alert (or 3..1)*
  - Show only security messages*: journactl SYSLOG_FACILITY=10*
- PID/Unit filtering/…:
  - *man journactl* for the complete list of options.



Index

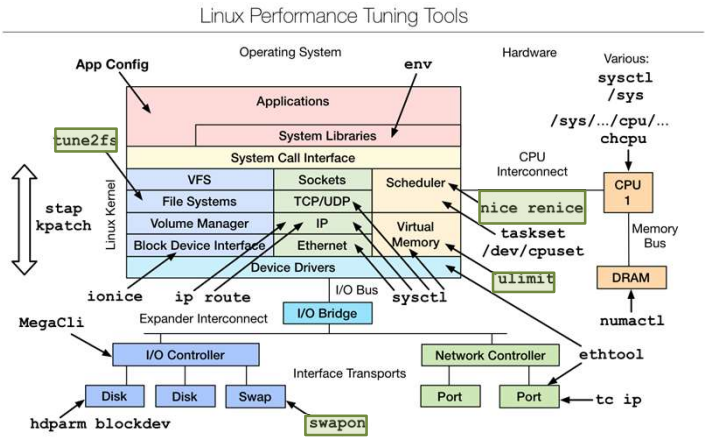
- Introduction: System Performance
  - Basic aspects
- Linux Processes (resource consumers)
  - The /proc FileSystem
  - Runaway Processes
- Resource Monitoring
  - Command-line utilities: ps, top, vmstat, uptime, strace, free, df, du.
  - Event gathering (logging) through systemd journal.
- **Resource Management**
  - Process Management: signals and priority
  - Controlling Shell process
  - Memory Management: Swapping
  - Disk Management: Quotas
- Periodic Processes
  - cron/at
  - systemd timers



Resource Management (commands)

- Again, being a critical aspect, the amount of available shell tools is huge…

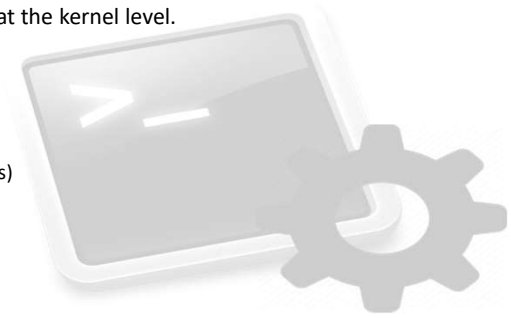
- Again, we limit to the most basic ones (highlighted in green boxes)



Process Management: Signals

Signal		Description	Default
1	HUP	Hangup	Terminate
2	INT	Interrupt	Terminate
9	KILL	Kill	Terminate
11	SEGV	Segmentation Fault	Terminate
19	STOP	Stop	Stop
18	CONT	Continue after stop	Ignore

- **Signals:** process-level interrupt requests
  - When received, two things can happen:
    - If the process has defined a handler, it is called.
    - Otherwise, the kernel takes some default action
  - 62 different signals (kill –l).
  - KILL: unblockable (no handler) and terminates a process at the kernel level.
  - INT: the Control-C combo.
- **Kill command: sending signals to a process**
  - Syntax: #kill <options> PID
    - Option –STOP: stop the process (-CONT to rerun the process)
    - Option -9: Kill the process (killall -9 user)

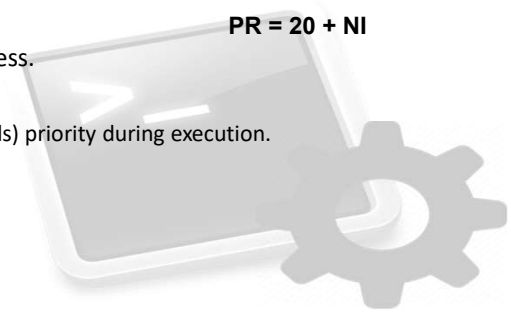


Process Management: Priority

- Process priority: scheduler of multi-process systems.
  - The scheduler allocates time intervals according to priority (column PR in top)
  - Users can “partially” regulate priority (column NI in top)
    - From -20 (max priority) to 19 (min priority)
  - If it is not modified externally, a process inherits its father priority.

PR = 20 + NI

- Command **nice**: Change the inherited priority of a process.
  - Syntax: nice –n +-value command.
  - **renice** allows to change command (or group of commands) priority during execution.
    - renice +-value [-p PID] [-u user] [-g group]



Process Management: CPU isolation & Affinity

- The OS scheduler is in charge of deciding where to place each process/thread.
  - Can we “manipulate this decision? Partially, yes.

- **CPU Isolation**: limiting the scheduling scope.
  - Isolate a CPU (or a group) so no process or interrupt can be scheduled for that CPU.
  - How to isolate a CPU?
    - In kernel boot options (grub) we can provide the kernel boot parameter “isolcpus=<num-cpu>”
    - Systemd CPU Affinity (/etc/systemd/system.conf): set affinity for systemd as well as everything it launches

- **Process Affinity**: Pin a task to a CPU permanently through taskset command.
  - Syntax: taskset --cpu-list 0-3 [command]
    - can also operate on an existing PID

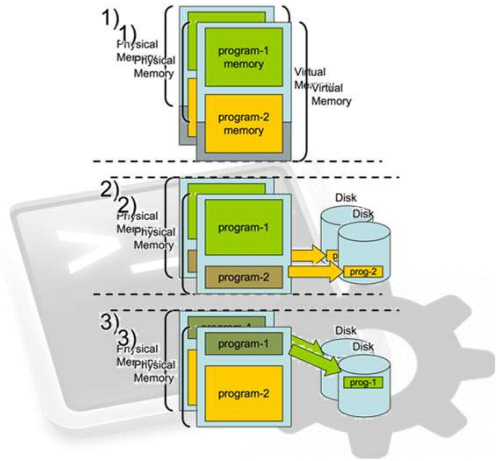
Controlling Shell Process

- Useful utility to control users: **limit shell resources**
- Command **ulimit**: limits the utilization of system resources by the shell.
  - Syntax: #ulimit -<option> [limit]
    - Option –a: informs about the current limits.
    - Option –f: max number of files created by the shell
    - Option –m: max available memory.
      - Option –t: max amount of CPU time (seconds)
      - …
  - The **/etc/security/limits.conf** file:
    - The pam_limits.so module applies ulimit limits.
    - File Syntax: <domain> <type> <item> <value>
      - Domain: username or groupname (* for default entry).
      - Type: hard vs. soft
      - Item: core, data, fsize, memlock, nofile, nproc, maxlogins, nice, …

```
calderon:~> ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
file size               (blocks, -f) unlimited
max locked memory       (kbytes, -l) unlimited
max memory size         (kbytes, -m) unlimited
open files                      (-n) 256
pipe size            (512 bytes, -p) 1
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) 709
virtual memory          (kbytes, -v) unlimited
```

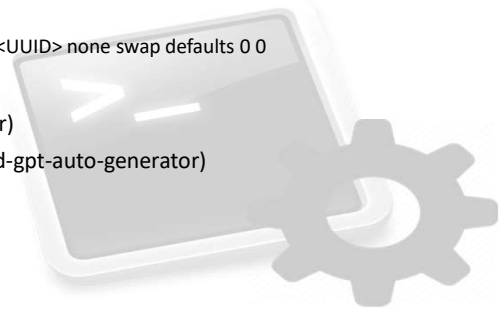
Memory Management: Swap

- Swapping space: Disk space reserved for page exchange between memory/disk.
  - Virtual Memory > Physical Memory (O.C., 2º Course).
- At least one partition required (during installation)
  - It appears in /etc/fstab
- Virtual Memory size is critical.
  - If a process exceeds this value it never starts
  - if it is exceeded dynamically execution is aborted).
- Alternatives to disk swap: in-Memory page compression
  - Trade CPU cycles for reduced disk swapping (I/O).
  - zswap: lightweight compressed cache for swap pages.
    - Previous step before disk swapping, LRU replacement scheme.
  - zram: Systems with no disk swap partition.



Modification of Swap Size

- Adding a new **swap partition**
  - Can be created with most GNU/Linux tools (gdisk/parted).
  - Typically designated as type 82.
  - Process:
    - # mkswap /dev/sdxy (set up the partition as linux swap area)
    - # swapon /dev/sdxy (enable the device for paging)
    - To enable swap partition on boot, add entry to fstab: UUID=<UUID> none swap defaults 0 0
- **Activation** of swap partition with **systemd**
  - Reads the fstab to generate units (systemd-fstab-generator)
  - Inspect the root disk (GPT only) to generate units (systemd-gpt-auto-generator)
    - Adding the swap partition to fstab is not necessary.
  - Units created in /run/systemd/generator
  - systemctl --type swap to see active swap units.



Modification of Swap Size

- Alternative to swap partitions: creating a **swap file**:
  - More flexible: vary file size on the fly and more easy removing.
  - Step 1: Create the swap file:
    - # dd if=/dev/zero of=/swapfile bs=1M count=512 (create a 512Mb swap file)
    - chmod 600 /swapfile
    - mkswap /swapfile
  - Step 2: Create the systemd unit file
    - Do it in /etc/systemd/system

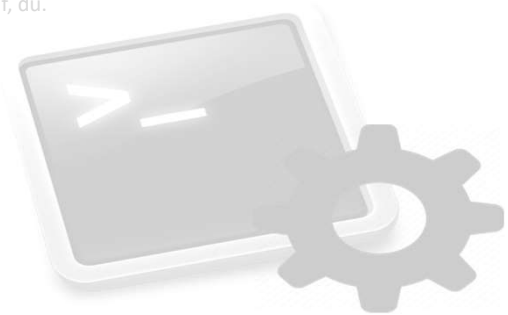
```
[Unit]
Description= Turn on swap

[Swap]
What=/swapfile

[Install]
WantedBy=multi-user.target
```

  - Step 3: Enable the unit and start using swap
    - systemctl enable –now file.swap
    - swapon to check if it is available.

Disk Management: Quotas

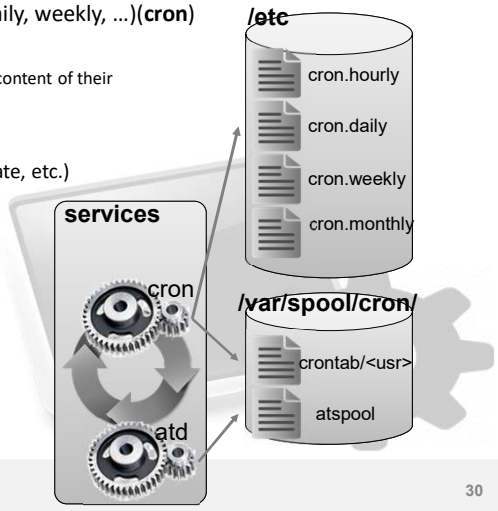
- Tool to limit the capacity for each user/group (delegated disk management).
- Two different limits: Soft vs Hard
  - Soft limit can be exceeded only during a established period. Hard limit is never exceeded.
- Requires quota support in the kernel (usually set up by default)
- Requires setting up in the File System
  - Modify /etc/fstab with usrquota and grpquota (restart after this, mount -a).
- Command **edquota**: modify the limits of a user/group.
  - Syntax: edquota -<options> [user] [-g group] (text editor)
- Command **quotaon/quotaoff**: power on/off quotas system.
- Command **repquota**: reports the content of quota system database
- Command **quota –v user**: see quota and status of a user.

Index



Periodic Processes

- **Scheduling** Services
  - Programmed execution (in the future) of periodic tasks (daily, weekly, …)(**cron**) or one-time (**atd**).
    - cron and atd services read periodically (every minute by default) the content of their configuration files to check if any command must be executed.
  - Some examples of periodic tasks:
    - Deleting /tmp directory, Backups, Database update(man, locate, etc.)



cron/at

- File **crontab**: one line per programmed task.
  - crontab –l: list all the tasks programmed
  - crontab –r: delete programmed tasks
  - crontab –e: edit file crontab.
  - Examples:
    - * * * * * * <command>
    - 9 0 1-7,9-16 * * 1-5 <command>
- Command **at**: controls atd daemon.
  - Sending a task: at TIME (it opens its own shell, where commands are specified)
    - #> at 13:00
    - at> ls –R /
  - Standard output via mail (can be redirected to a file)
  - See pending tasks: at –l
  - Remove tasks: at –d <job>

- **Security:**
  - /etc/cron.allow /etc/cron.deny
  - /etc/at.allow /etc/at.deny

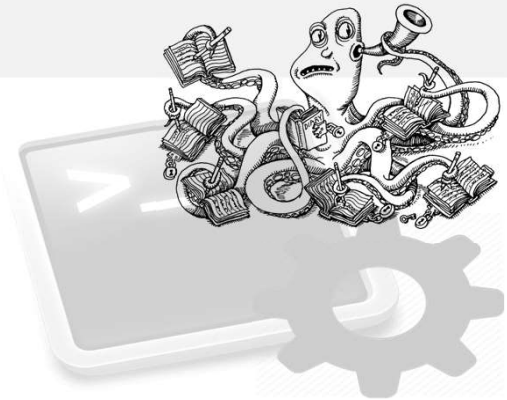
systemd timers

- systemd timers can be an alternative to cron
- Timer unit files control service unit files
  - For each .timer file a matching –service file exists
- Two different types of timer units:
  - **Realtime** timers: activate on a calendar event (same as cronjobs)
  - **Monotonic** timers: activate after a time span relative to a varying starting point.

```
[Unit]
Description= A timer that runs the service the
first four days of each month at 12:00 PM,
but only if that day is a Monday or a Tuesday

[Timer]
OnCalendar= Mon,Tue *-*-01..04 12:00:00
Persistent=true

[Install]
WantedBy=timers.target
```

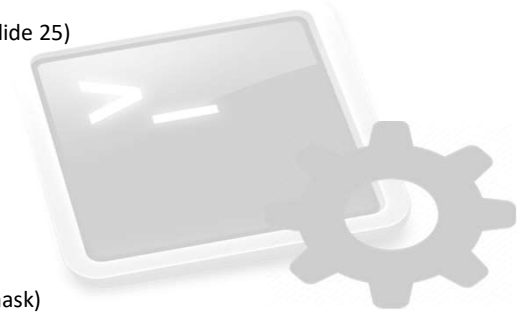
```
[Unit]
Description= A timer that starts 15 minutes after
boot and again every week while the system is
running

[Timer]
OnBootSec=15min
OnUnitActiveSec=1w

[Install]
WantedBy=timers.target
```

Appendix: rsyslog



Rsyslog configuration

- Default configuration file is **/etc/rsyslogd.conf**
  - Additional files can be included in /etc/rsyslogd.conf.d/
  - Three different parts: Modules, Directives and Rules
- **Modules** extend the capabilities of the core processing engine
  - Syntax: module(load="module-name")
  - imjournal: integrates rsyslog with the systemd journal (slide 25)
  - imuxsock: default when systemd is not present
  - imklog: understand kernel messages
  - omfile: writes messages to a file
  - omfwd: forward messages to a remote syslog server
- **Directives**: rsyslogd daemon configuration
  - Syntax: $Directive value
  - Permissions for all log files (FileOwner, FileGroup, …, Umask)



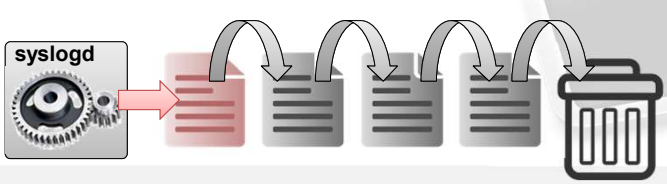
Rsyslog configuration

- Default configuration file is **/etc/rsyslogd.conf**
  - Additional files can be included in /etc/rsyslogd.conf.d/
  - Three different parts: Modules, Directives and Rules
- **Rules** construct simple filters for a specific input
  - Syntax: facility.severity action
  - **facility** program sending the log message
  - **severity** defines the priority level
    - emerg/alert/crit/err/warning/notice/info/debug
  - **action**: what to do with the received message
    - filename to append the message to a file
    - @hostname to fwd the message to hostname
    - user1,user2,… to write the message to the screen of users.

Facility	Programs that use it
*	All facilities
auth	security and authorization commands
cron	cron daemon
daemon	system daemons
kern	the kernel
local0-7	local message
lpr	printer spooling system
mail	mail-related software
syslog	syslogd internal messages
user	user processes (default)

Log Rotation

- Maintaining Log information (**log files**) is essential for control and repair
- But larger logged information-> More disk consumed
  - Can exhaust disk quota.
  - Hard to find information in a file with millions of lines.
- **Log rotation:** mechanism consisting on periodically writing to a new log file, creating a new empty one and deleting the oldest ones
  - Manual Rotation: Example script performing it.




```
#!/bin/sh
cd /var/log/
mv messages.2 messages.3
mv messages.1 messages.2
mv messages messages.1
cat /dev/null > messages
chmod 600 messages
#Reiniciar syslog
service restart rsyslog
```

Log Rotation

- Automatic Rotation: **logrotate**
  - unsupervised organization of log rotation. Avoids disk overflow and keeps access to a historic of the system.
  - Configuration through the file /etc/logrotate.conf
    - Applied by default to every service.
  - Particularization for a service: /etc/logrotate.d/
    - Overwrites the options in logrotate.conf

```
# rotate log files weekly, monthly
weekly
# keep 4 weeks worth of backlogs
rotate 4
# send errors to root
errors root
# create new(empty)log files after rotating old ones
create
# compressed log files
compress
# DEB packages drop log rotation info into this dir
include /etc/logrotate.d
#no packages own lastlog or wtmp, rotate them here
/var/log/wtmp cd /var/log/{
            monthly
            create 0664 root utmp
            rotate 1
}
```

```
/var/log/dpkg.log {
    monthly
    rotate 12
    compress
    notifempty
    create 0664 root adm
}
```