# LPIC2 2021 Sesión 3 (2021-02-09) - procesos, kernel, systemd/

## Documentación relacionada 200, 201, 202

- Presentaciones/2020/201/200-Capacity Planning.pdf
- Material Practicas LPIC-2/LPIC-201/1-Capacity Planning/Gestion de Procesos/2-Administracion Basica de Procesos.pdf
- Material Practicas LPIC-2/LPIC-201/1-Capacity Planning/Gestion de Procesos/Troubleshooting Active Processes.pdf
- Material Practicas LPIC-2/LPIC-201/1-Capacity Planning/Gestion de Procesos/3-Ejercicio Gestión de procesos y trabajos
- Material Practicas LPIC-2/LPIC-201/1-Capacity Planning/Gestion de Procesos/Limit CPU Usage with CPULimit.pdf (no hecho)
- Presentaciones/2020/201/201 - Linux Kernel.pdf
- Material Practicas LPIC-2/LPIC-201/2-Linux Kernel/1-Introduccion al kernel de Linux.pdf
- Material Practicas LPIC-2/LPIC-201/2-Linux Kernel/2-Kernel en Linux.pdf
- Material Practicas LPIC-2/LPIC-201/2-Linux Kernel/Laboratorion Compilar Kernel Linux.pdf
- Material Practicas LPIC-2/LPIC-201/3-System Startup/
- Presentaciones/2020/201/202 - Arranque del Sistema.pdf

### otros documentos

- Material Practicas LPIC-2/LPIC-201/2-Linux Kernel/Compresión y descompresión de archivos.pdf → completo_incremental
- Material Practicas LPIC-2/LPIC-201/2-Linux Kernel/Gestion de Librerias Compartidas.pdf → Linux dynamic libraries, librerias compartidas
- Material Practicas LPIC-2/LPIC-201/2-Linux Kernel/Udev para la detección y gestión de dispositivos en Linux.pdf

## Clase

### ps

El comando ps es el que permite informar sobre el estado de los procesos. ps esta basado en el sistema de archivos /proc, es decir, lee directamente la información de los archivos que se encuentran en este directorio.

- `ps -efl | more`
  - `-e`: todos los procesos
  - `-f`: opciones completas
  - `-F`: opciones extras completas
  - `-a`: muestra todos
  - `-x`: sin mostrar tty
  - `-u`: muestra usuarios
  - `-eo <campo>,<campo>,<campo>`: o = output personalizado
- `ps aux | more`
- `ps -u <user>`: pocesos de <user>
- `ps -eH`: muestra arbol de procesos

## pstree

- `pstree -AGu` → deprecado, usar → `systemd-cgls`
- aunque pstree está en la certificación

## kill

El comando kill, que literalmente quiere decir matar, sirve no solo para matar o terminar procesos sino principalmente para enviar señales (signals) a los procesos

- `kill l`
- `kill -9 pid` ≡ `kill -SIGKILL pid`

## killall

El comando killall, que funciona de manera similar a kill, pero con la diferencia de en vez de indicar un PID se indica el nombre del programa, lo que afectará a todos los procesos que tengan ese nombre.

- `killall -HUP httpd`:manda una señal de «colgar», detenerse releer sus archivos de configuración y reiniciar
- `killall -KILL -i squid`: manda señal de matar a todos los procesos squid pero pide confirmación en cada uno

## nice/renice

Permite cambiar la prioridad de un proceso. Por defecto, todos los procesos tienen una prioridad igual ante el CPU que es de 0.

- `nice -n <prioridad> <comando>`
- Asi como nice establece la prioridad de un proceso cuando se incia su ejecución, renice permite alterarla en tiempo real, sin necesidad de detener el proceso
- `renice <prioridad> <pid>`: cambia el nice de un proceso en ejecución

## nohup y &

independiza el proceso lanzado desde la consola de la propia consola, le permite ser independiente

- `nohup <comando>`
- `$$`: PID shell actual
- `$!`: PID último proceso ejecutado
- `$?`: código de salida del comando anterior ejecutado. 0 es todo correcto

## jobs, fg, bg

trabajos en background

- **fg** foreground
- **bg** background
- <ejecutar> → ^Z → stopped
- `jobs`: vemos trabajos en background

- `fg <trabajo>`: reactiva el trabajo en background → ZSH `fg %<trabajo>`
- `bg <trabajo>`: reactiva el trabajo en el foreground → ZSH `bg %<trabajo>`
- `kill %<trabajo>`: mata ese trabajo
- `killall <nombre>`:
- `ps -j`: procesos consola actual

### time

medir tiempo de ejecución de un comando/script

- `time <comando>`

### pidof

Este comando muestra el o los PID del programa invocado.

- `pidof <programa>`

### pgrep

El comando pgrep muestra los identificadores de los procesos.

- `pgrep <opciones> <programa>`
  - `-l`: muestra el nombre proceso
  - `-u <user>`: procesos de <user>

## kernel

`/boot/vmlinuz*`: nuestro kernel compilado

### compresión kernel (siempre gzip)

- zImage: tamaño máximo de 520KB
- bzImage: sin límite de tamaño, para kernels grandes (big zImage)
- `uname -r`: versión kernel
- `uname -a`
- `/usr/src/linux` o `/usr/src/linux`
- `/usr/src/linux/Documentation`

### versiones (https://www.kernel.org/)

- mainline
- stable
- prepatch
- longterm

### compilar

- descargar el kernel

- fichero configuración kernel con opciones actuales: **/boot/config...**
- copiar en directorio kernel para compilar como **.config** para que recoja los valores de hard de la máquina del kernel ya existente
- `make config`: preguntará paso a paso
- `make menuconfig`: modo menú
  - leer fichero configuración alternativo → **.config**
  - modificar opciones
  - guardar en el **.config**
- `make xconfig`
  - has de tener librerias gráficas para que funcione
  - o usando emulador (en Windows) de unas X-Windows:
    - `export DISPLAY=<ip_remoto>:0.0`
    - servidores X:
      - **xming X server**: https://sourceforge.net/projects/xming/
        - **xlaunch**
      - https://hummingbird-exceed.software.informer.com/11.0/
- `make all && make modules install && make install`
  - **make install** añade una nueva opción al grub → **/boot/grub/menu.lst**
- `make clean && make mrproper`: limpian el fichero .config con las opciones que se habían seleccionado
  - clean: Remove most generated files but keep the config and enough build support to build external modules
  - mrproper: Remove all generated files + config + various backup files

## módulos

- ***.ko**
- **/lib/modules/kernel_version**
- `lsmod` lista módulos cargados
- `modinfo <modulo>`
  - ubicación y dependencias
- `modprobe`: carga el módulo (y las dependencias)
  - `-r`: descarga el módulo (y las dependencias)
- `insmod`: carga un módulo (pero no las dependencias)
- `rmmod` descarga un módulo (pero no las dependencias)
- **/etc/modules**
- **/etc/modprobe.d/**

## parches de kernel

- descargar de la página el .bz o -gz
- `bzip2 -dc patch…. | patch -p1 –dry-run`
- `unpatch`

## mkinitrd && mkinitramfs

Un disco inicial de RAM es una colección de modulos críticos del kernel y de utilidades de sistema que el bootloader. Lee del disco y pasa al kernel en el momento del arranque.

Si el kernel ya incluye todos los drivers que se necesitan para arrancar dentro de su fichero inicial, este paso sería innecesario. Cada distribución utiliza una utilidad diferente para la generación del disco inicial de RAM, siendo los más comunes **mkinitrd** y **mkinitramfs**.

- **/boot/initrd...img**

### kernel panic

- Hard: **Aieee!**
- Soft: **Ooops!**
  - modificar **/etc/fstab** con algún error sintáctico

### modificación parámetros kernel

- `sysctl`
  - `sysctl <clave>`: lee el valor de la clave (separado por puntos)
    - `echo <clave>`: idem anterior, pero en lugar de puntos, es un FS
  - `sysctl -w <clave>=<valor>`: escribe el valor en la clave (separadoa por puntos)
    - `echo «1» > /proc/sys/net/ipv4/icmp_echo_ignore_all`
  - `sysctl -p[=myfile.conf]`: carga **/etc/sysctl.conf** (por defecto) o el fichero especificado.
- **/etc/sysctl.conf**
- **/etc/sysctl.d/**

## systemV VS systemD

### systemV

- es el más antiguo de los dos
- **init** es el primer proceso (PID=1)
  - debian10 muestra **init** pero es **systemd** → `cat /proc/1/stat`, `systemctl --version`
- `cat /etc/inittab` ← en qur runlevel arraca la máquina
- cambio de runlevel en caliente: **init <runlevel>** o **telinit <runlevel>**
- **/etc/init.d/<servicio> <accion>** o **service <servicio> <accion>**
- runlevels (0-6)
  - servicios: se gestionan a través de **/etc/rc.d/rcX.d/** (X representa el runlevel)
  - enlaces simbólicos a los servicios a arrancar o parar (normalmente a **/etc/rc.d/init.d/**)
  - tienen un formato [S|K]DDNombre
  - si precede una **S**, el servicio se arranca
  - si precede una **K**, el sercicio se para
  - **DD** establece el orden de arranque
  - para crear los servicios para que arranquen:
    - crear los enlaces simbólicos correspondientes
    - usar **chkconfig**
      - si usamos **chkconfig** lee una línea del script chkconfig para obtener la información
        - formato: chkconfig <runlevels> <orden arranque> <orden parada>
      - `chkconfig --add <servicio>`
      - `chkconfig --del <servicio>`
      - `chkconfig --list` o `service –status-all`
      - `chkconfig --level <levels> servicio [on|off]`
- **/etc/rc.local**
  - script independiente del nivel de ejecución
  - se ejecuta posterior a los scripts del runlevel

# TODO

1. ~~mirar documento **Recopilación de información de hardware**, sección /sys~~ → sysfs

> 2. Material Practicas LPIC-2/LPIC-201/2-Linux Kernel/Laboratorio Copia de seguridad y restauración GRUB.pdf

    1. GRUB
    2. MGR
    3. comando dd

# tips & tricks

- cgroups
- https://www.raulprietofernandez.net/blog/hacking/como-crear-una-bomba-fork-para-gnu-linux-y-como-protegernos
- `ulimit`: muestra límites establecidos para el usuario (CPU, RAM, Procesos, ficheros...)
    - https://linuxhint.com/linux_ulimit_command/
    - **/etc/security/limits.conf**
- Emulador (en Windows) de unas X-Windows para Linux sin «gráficos»:
    - `export DISPLAY=<ip_remoto>:0.0`
    - servidores X:
        - **xming X server**: https://sourceforge.net/projects/xming/
            - **xlaunch**
        - https://hummingbird-exceed.software.informer.com/11.0/