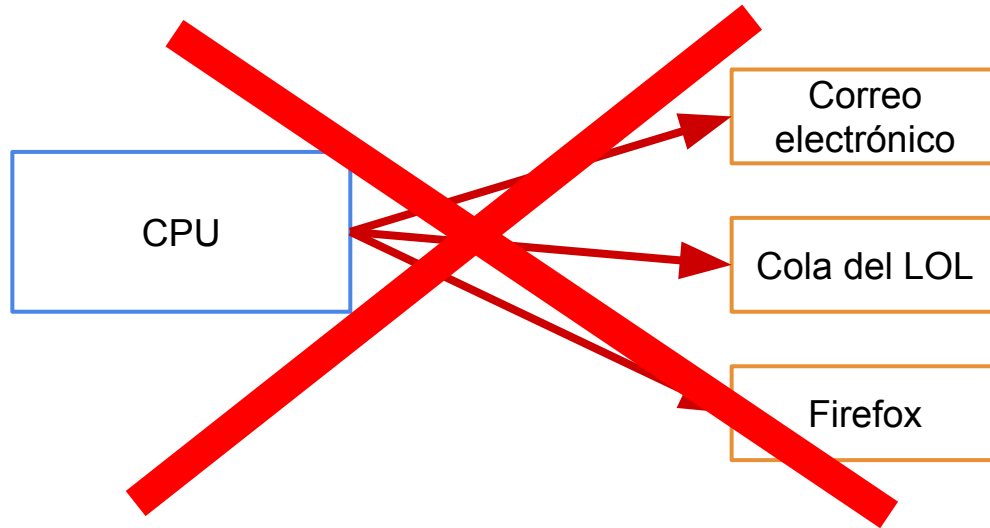


# UT1. Programación de Procesos

PSP - DAM  
Francisco Gallego Perona

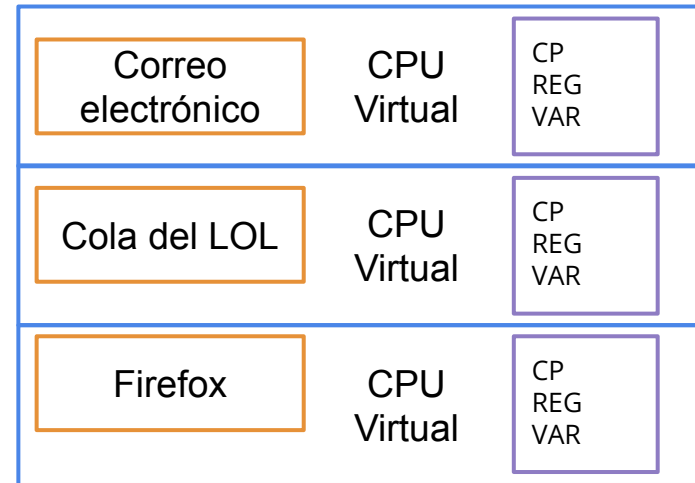
# ¿Qué es un proceso?

- Un proceso es un **programa en ejecución**.
- Proporcionan la capacidad de operar (pseudo) **concurrentemente**.
- 1 CPU física → Varias CPU virtuales



# ¿Qué es un proceso?

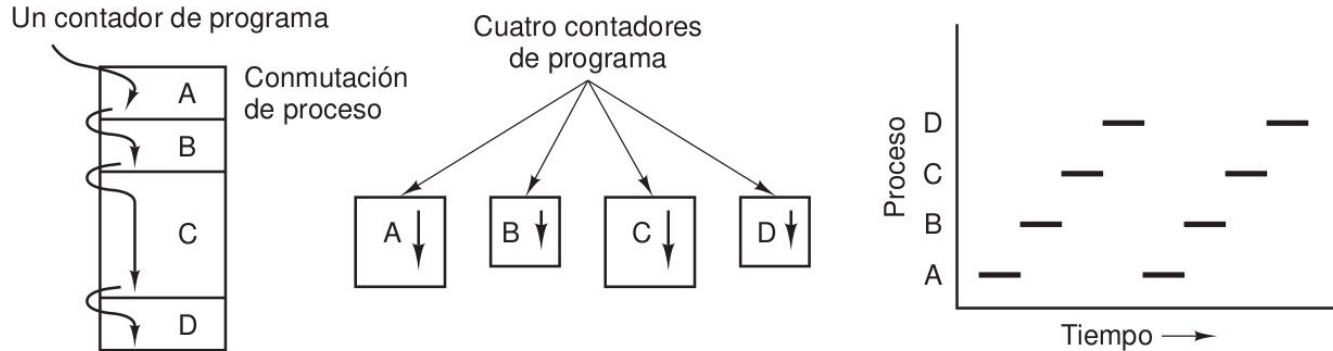
- Un proceso es un **programa en ejecución**.
- Proporcionan la capacidad de operar (pseudo) **concurrentemente**.
- 1 CPU física → Varias CPU virtuales



# ¿Qué es un proceso?

- El software ejecutable de un PC se organiza en **procesos secuenciales**.

Ejemplo de 4 procesos ejecutándose concurrentemente:



# ¿Qué es un proceso?

Los Sistemas operativos multiproceso o multitarea → Pueden ejecutar procesos simultáneamente:

- **Real:** Usando varias CPUs
- **Simulada:** En Sistemas operativos con una sola CPU se alterna la ejecución de los procesos. Esta operación se realiza tan rápido que parece que cada proceso tiene dedicación exclusiva.

La programación multiproceso tiene en cuenta la posibilidad de que múltiples procesos puedan estar ejecutándose simultáneamente sobre el mismo código de programa.

# BCP (Bloque de control del proceso):

El SO (Sistema Operativo) controla la ejecución de los procesos. Decide qué proceso para y qué proceso continúa. Cuando se suspende temporalmente la ejecución de un proceso **debe reiniciarse posteriormente en el mismo estado en el que estaba.**

BCP:

- PID. Identificador del proceso
- Estado del proceso
- Contador de programa
- Registros de CPU
- Otra información: prioridad del proceso, gestión de memoria, cantidad de tiempo de CPU, estado de E/S como la lista de dispositivos asignados, archivos abiertos, etc.

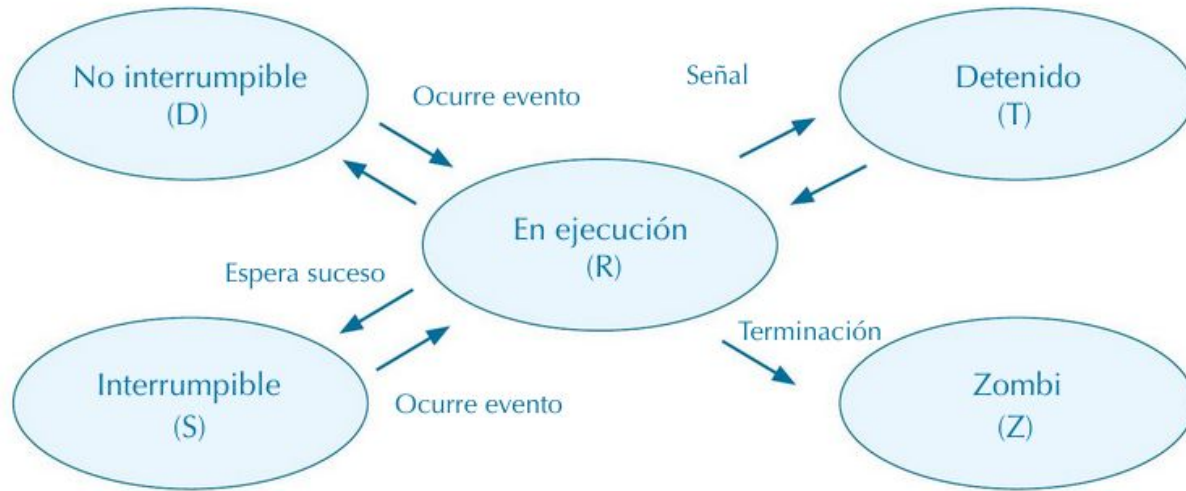
# Procesos en Linux

ps -AF:

- **PID:** identificador del proceso:
- **TTY:** terminal asociado del que lee y al que escribe. Si no hay aparece interrogación.
- **TIME:** tiempo de ejecución asociado, es la cantidad total de tiempo de CPU que el proceso ha utilizado.
- **CMD:** nombre del proceso.
- **UID:** nombre de usuario
- **PPID:** PID del padre de cada proceso.
- **C:** porcentaje de recursos de CPU utilizado por el proceso.
- **STIME:** hora de inicio del proceso.
- **SZ:** tamaño virtual de la imagen del proceso.
- **RSS:** tamaño de la parte residente en memoria en kilobytes.
- **PSR:** procesador que el proceso tiene actualmente asignado.

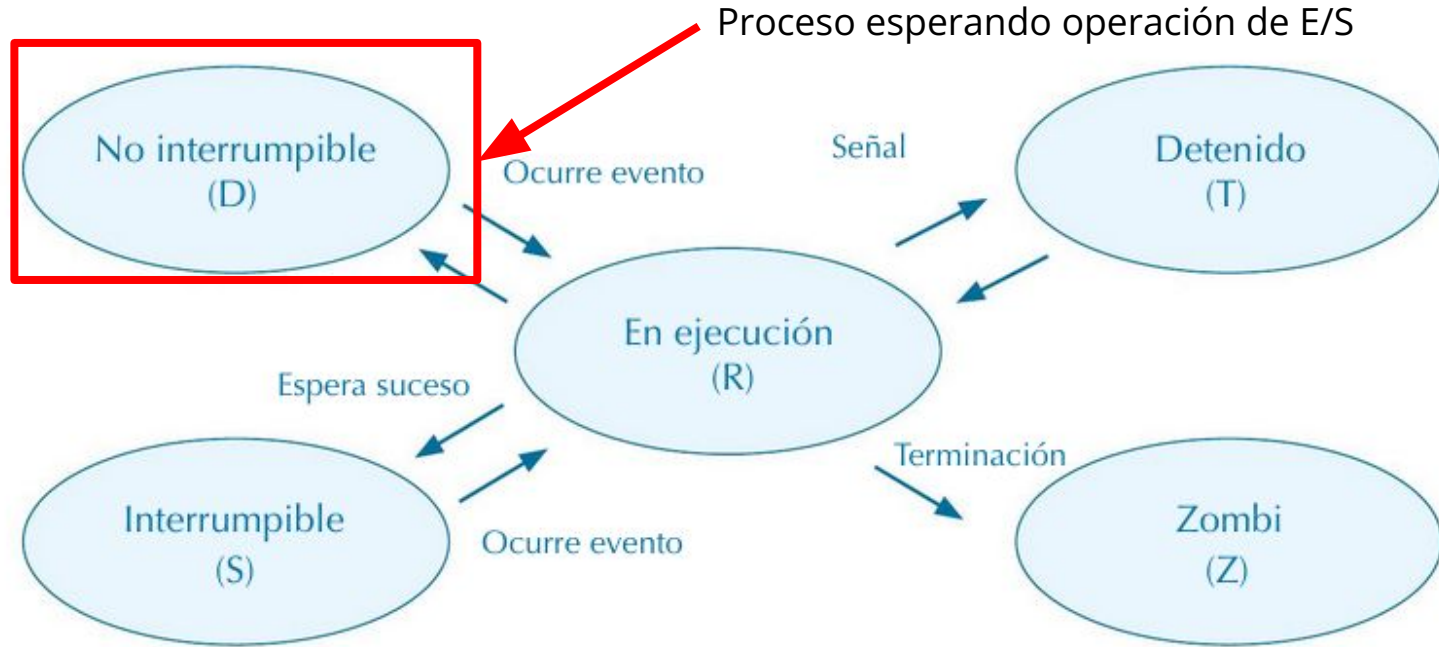
# Estados de un proceso: top

En Linux los procesos pueden estar en diferentes estados, que se pueden ver en el siguiente diagrama (según el comando top):

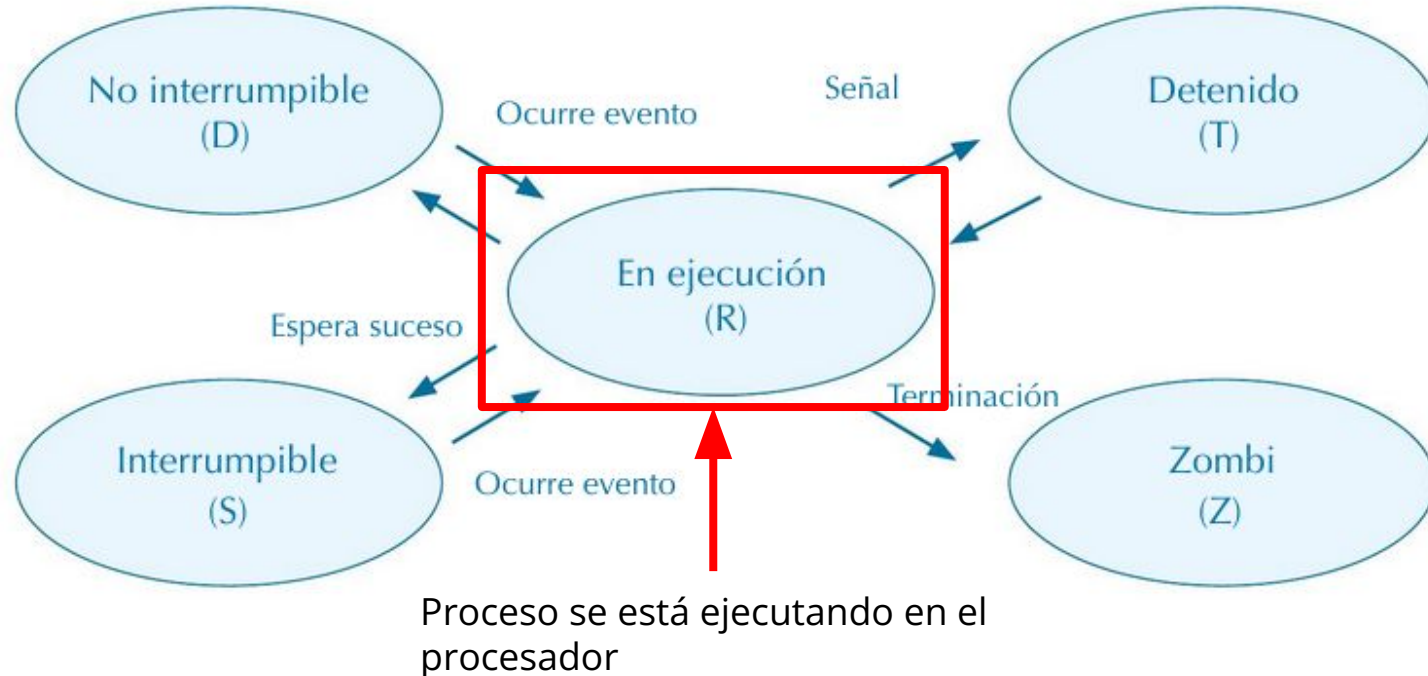




# Estados de un proceso: top



# Estados de un proceso: top



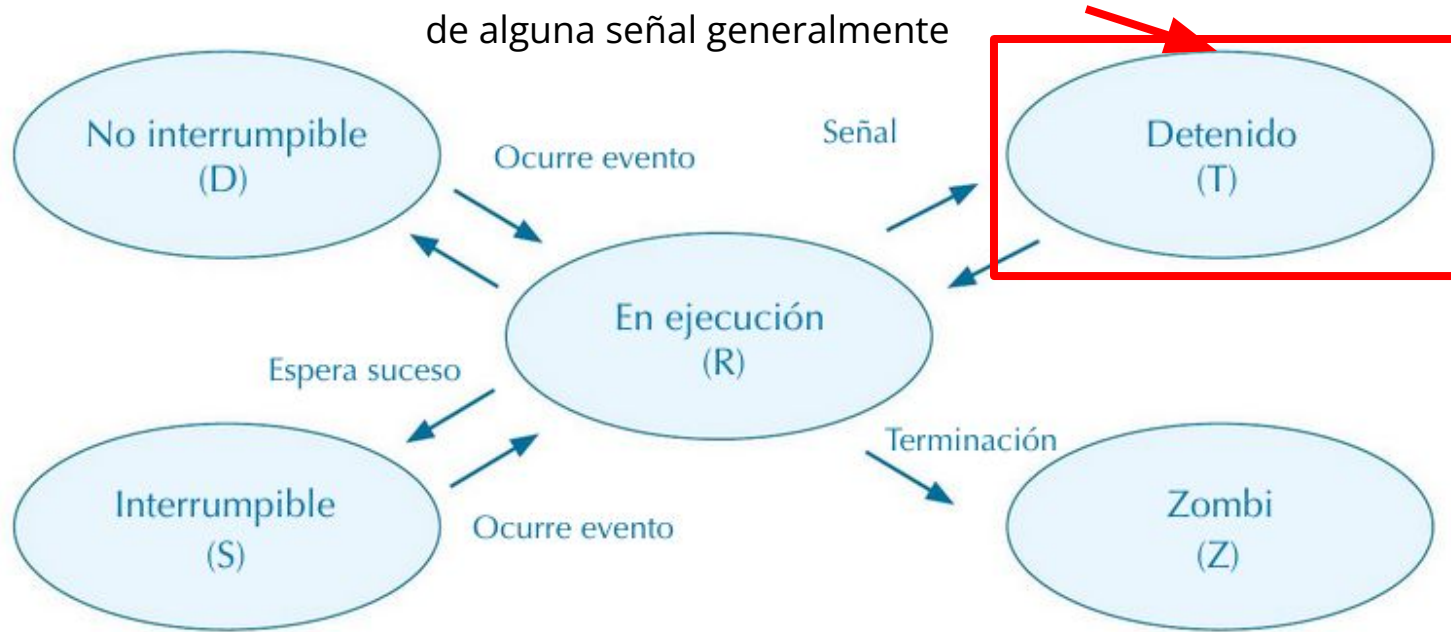
# Estados de un proceso: top



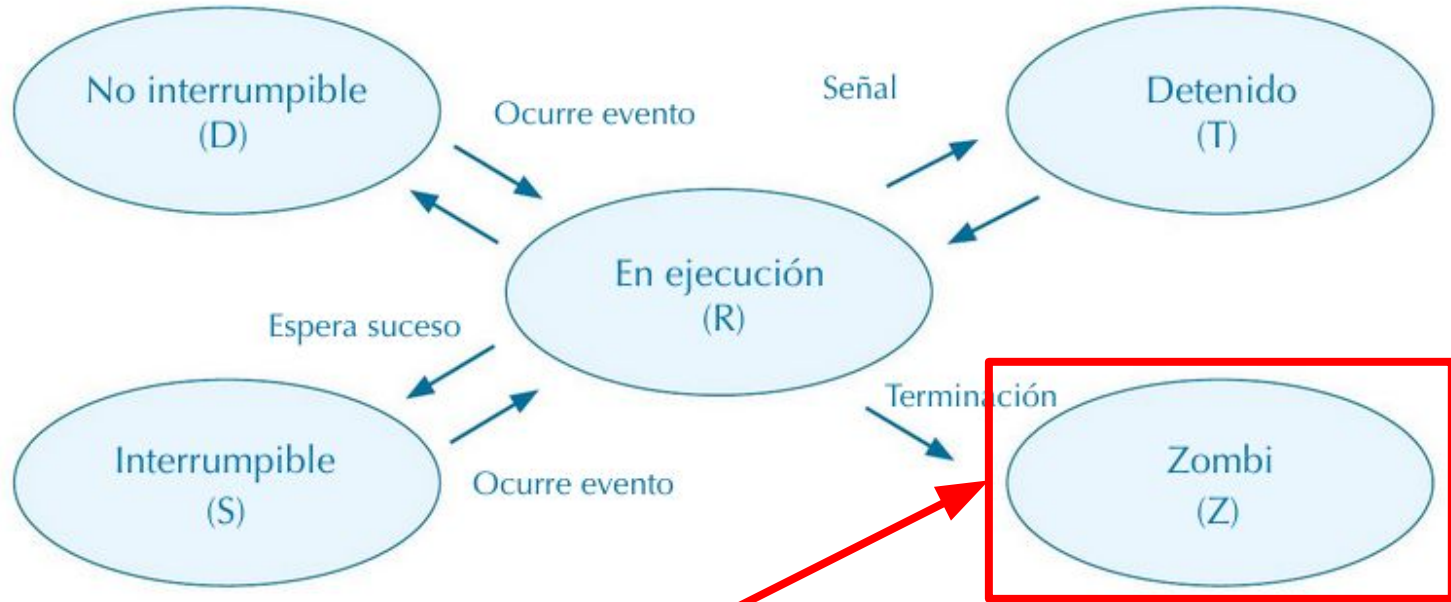
Proceso se encuentra esperando a que se cumpla algún evento para su ejecución

# Estados de un proceso: top

Proceso detenido mediante el envío de alguna señal generalmente



# Estados de un proceso: top



Proceso terminado cuyo padre sigue "vivo".

# Código de estado de procesos

- D   uninterruptible sleep (usually IO)
- I   Idle kernel thread
- R   running or runnable (on run queue)
- S   interruptible sleep (waiting for an event to complete)
- T   stopped by job control signal
- t   stopped by debugger during the tracing
- X   dead (should never be seen)
- Z   defunct ("zombie") process, terminated but not reaped by its parent

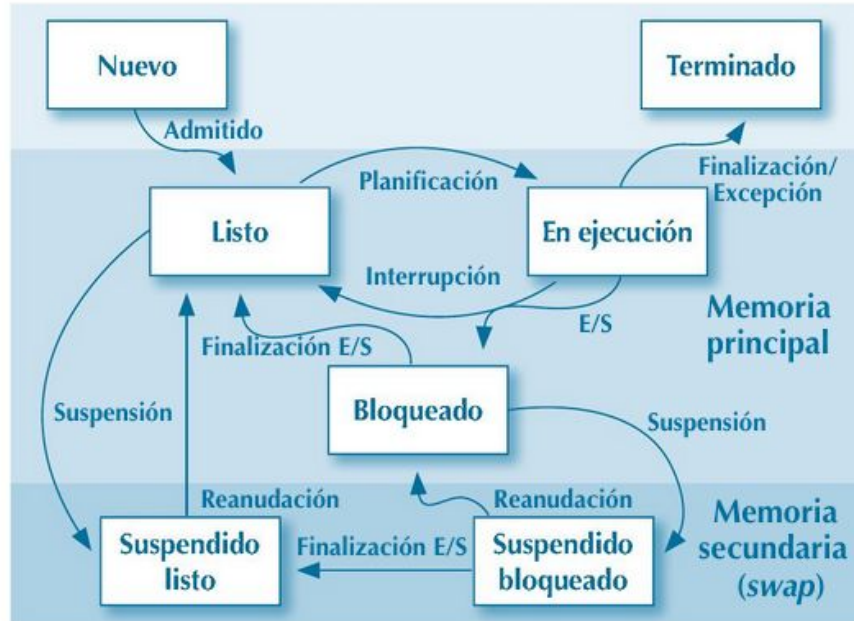
# Planificación de procesos

Desde que se crea un proceso, puede pasar por diversos estados. **El sistema operativo** se encarga de **gestionar los procesos** y realizar dichos **cambios de estado**, teniendo en cuenta eventos que suceden durante el **ciclo de vida de un proceso**.

## Objetivos:

1. **Máximo aprovechamiento de los recursos** del sistema (en particular del procesador o procesadores).
2. Ejecución **lo más eficiente** posible de los procesos.

# Planificación de procesos



Planificación  
a largo plazo

Planificación  
a corto plazo

Planificación  
a medio plazo



# Creación de un proceso

Hay 4 eventos principales que provocan la creación de procesos:

1. El **arranque** del sistema
2. La ejecución, desde un proceso, de una **llamada del sistema** para crear un proceso
3. **Petición de usuario** para crear un proceso
4. Inicio de **trabajo por lotes (batch job)**

# Creación de un proceso

Cuando se arranca el sistema operativo se crean varios procesos:

- En **primer plano**: interactúan con el **usuario**
- En **segundo plano**: asociados a **funciones específicas** (no interactúan con el usuario)

Los procesos en segundo plano para manejar actividades como: correo electrónico, impresión, páginas web, etc → **demonios** (daemons)

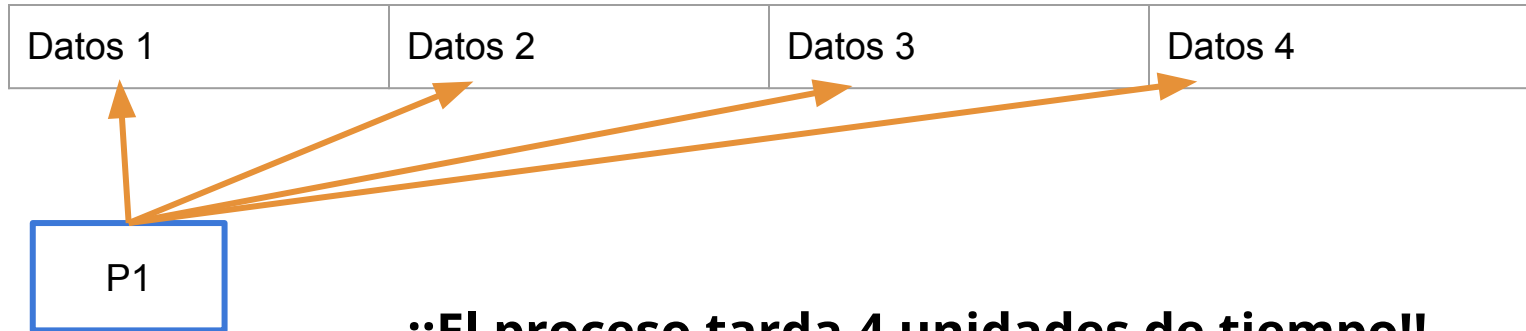
## ¿Cómo ver los procesos en el sistema?

En Linux → Comando ps (comando completo → ps aux)

En windows → Administrador de tareas

# Creación de un proceso

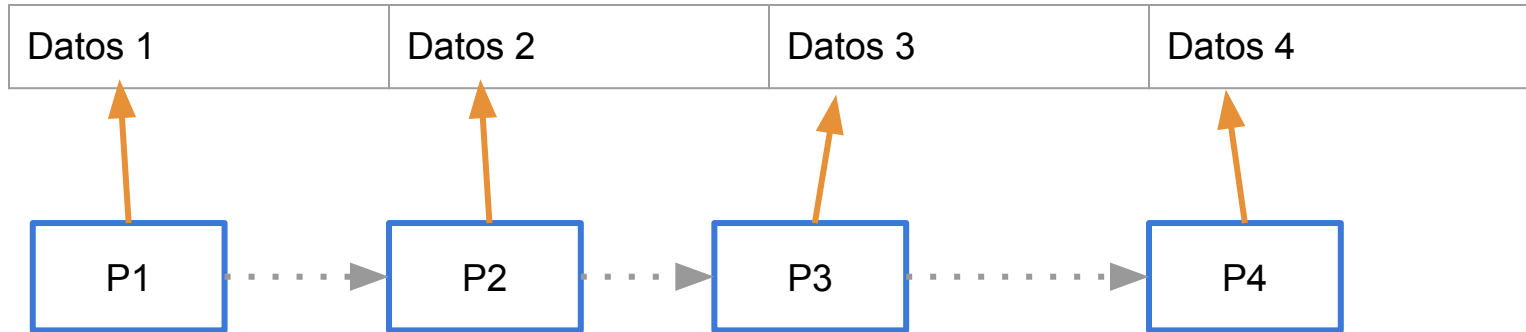
A menudo, un proceso en ejecución puede realizar llamadas al sistema para crear nuevos procesos (estos nuevos procesos le ayudarán a realizar su trabajo).



**¡¡El proceso tarda 4 unidades de tiempo!!**

# Creación de un proceso

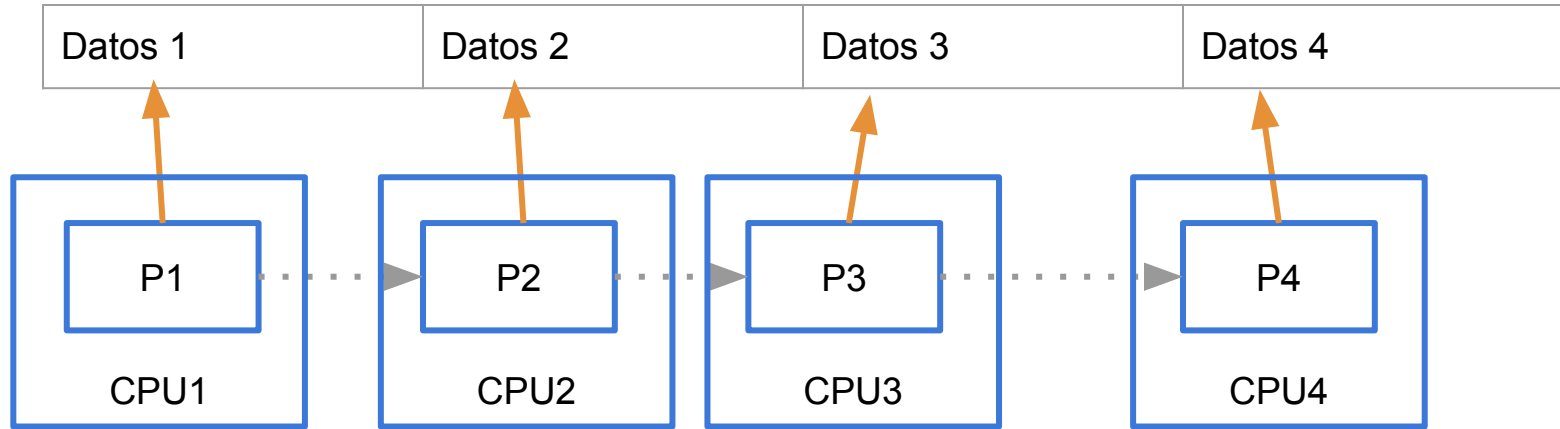
A menudo, un proceso en ejecución puede realizar llamadas al sistema para crear nuevos procesos (estos nuevos procesos le ayudarán a realizar su trabajo).



**¡¡El proceso tarda mucho menos!!**

# Creación de un proceso

A menudo, un proceso en ejecución puede realizar llamadas al sistema para crear nuevos procesos (estos nuevos procesos le ayudarán a realizar su trabajo).



# Listado de procesos: ps

Para ver los procesos utilizamos el comando **ps**. Las opciones del comando más habituales son:

- **ps aux**: muestra todos los procesos del sistema
- **ps axjf**: muestra un árbol jerárquico con la ruta del programa al que pertenece el proceso
- **ps aux | grep [NOMBRE\_PROCESO]**: realiza un filtrado sobre ps para obtener únicamente los procesos pertenecientes a [NOMBRE\_PROCESO]

# Listado de procesos: ps

Las **opciones básicas** del comando ps son:

- **-a**: Lista los procesos de todos los usuarios.
- **-u**: Lista información del proceso como, por ejemplo, el usuario que lo está ejecutando, la utilización de CPU.
- **-x**: Lista los procesos de todas las terminales y usuarios.
- **-l**: Muestra información que incluye el UID y el valor nice.

# Listado de procesos: ps


Algunos de los datos más importantes que nos muestra por cada proceso son:

- **USER:** Usuario que lanzó el proceso.
- **PID:** Identificador del proceso.
- **PPID:** Identificador del proceso padre.
- **%CPU:** porcentaje entre el tiempo usado realmente y el que lleva en ejecución.
- **%MEM:** Fracción de memoria consumida (estimada).
- **VSZ:** Tamaño virtual del proceso (código + datos + pila), en KB.
- **RSS:** Memoria real usada, en KB.
- **TTY:** Terminal asociado con el proceso.



# Listado de procesos: top

Comando **top**:



```
top - 17:26:26 up 2:06, 1 user, load average: 0,48, 0,36, 0,38
Tasks: 395 total, 1 running, 394 sleeping, 0 stopped, 0 zombie
%Cpu(s): 1,5 us, 0,6 sy, 0,0 ni, 97,6 id, 0,1 wa, 0,0 hi, 0,2 si, 0,
MiB Mem : 15397,4 total, 7183,9 free, 4795,0 used, 3418,4 buff/cache
MiB Swap: 2048,0 total, 2048,0 free, 0,0 used. 10038,6 avail Mem
```

Info sistema

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+
5	root	0	-20	0	0	0	I	0,0	0,0	0:00.00
7	root	0	-20	0	0	0	I	0,0	0,0	0:00.00
9	root	0	-20	0	0	0	I	0,0	0,0	0:00.04
10	root	0	-20	0	0	0	I	0,0	0,0	0:00.00
11	root	20	0	0	0	0	S	0,0	0,0	0:00.00
12	root	20	0	0	0	0	S	0,0	0,0	0:00.00
13	root	20	0	0	0	0	S	0,0	0,0	0:00.32
14	root	20	0	0	0	0	I	0,0	0,0	0:04.00
15	root	rt	0	0	0	0	S	0,0	0,0	0:00.01
16	root	-51	0	0	0	0	S	0,0	0,0	0:00.00
18	root	20	0	0	0	0	S	0,0	0,0	0:00.00
19	root	20	0	0	0	0	S	0,0	0,0	0:00.00
20	root	-51	0	0	0	0	S	0,0	0,0	0:00.00
21	root	rt	0	0	0	0	S	0,0	0,0	0:00.23

# Listado de procesos: htop

Comando htop:

```
pacotoh@blade: ~

0[||||| 12.0%] 4[||||| 8.0%] 8[||||| 4.0%] 12[||||| 25.0%]
1[||||| 1.0%] 5[||||| 6.0%] 9[||||| 4.0%] 13[||||| 2.0%]
2[||||| 15.0%] 6[||||| 1.0%] 10[||||| 5.0%] 14[||||| 1.0%]
3[||||| 0.0%] 7[||||| 0.0%] 11[||||| 1.0%] 15[||||| 15.0%]
Mem[||||||||| 5.05G/15.0G] Tasks: 157, 929 thr: 1 running
Swp[||||| 0K/2.0G] Load average: 0.35 0.33 0.36
Uptime: 02:11:15

  PID USER   PRI  NI  VIRT   RES   SHR  S  CPU% MEM%   TIME+  Command
2102 pacotoh   20    0 4572M   508M  130M  S 36.5  3.3  4:39.84 /usr/bin/gnome-shell
12377 pacotoh   20    0 3288M   659M  167M  S 23.9  4.3  5:17.64 /snap/firefox/1810/u
3703 pacotoh   20    0 13.0G 1085M  365M  S 17.2  7.1 17:34.79 /snap/firefox/1810/u
1666 pacotoh   20    0 26.5G 197M   110M  S  6.6  1.3  4:37.97 /usr/lib/xorg/Xorg v
17962 pacotoh   20    0 2897M 43768 32780  S  6.6  0.3  0:00.10 /usr/bin/gjs /usr/sh
6291 pacotoh   20    0 14.8G 385M   88708  S  4.6  2.5  4:25.96 /usr/share/teams/tea
15274 pacotoh   20    0 13.0G 1085M  365M  S  4.6  7.1  0:25.41 /snap/firefox/1810/u
3844 pacotoh   20    0 13.0G 1085M  365M  S  4.0  7.1  3:38.19 /snap/firefox/1810/u
15276 pacotoh   20    0 13.0G 1085M  365M  S  4.0  7.1  0:32.60 /snap/firefox/1810/u
1593 pacotoh   39   19  628M 36488 18920  S  2.7  0.2  0:02.16 /usr/libexec/tracker
6239 pacotoh   20    0 2027M 505M   91652  S  2.0  3.3  2:17.12 /usr/share/teams/tea
17956 pacotoh   20    0 196M   22968 16220  S  2.0  0.1  0:00.03 /usr/libexec/gnome-c
17960 pacotoh   20    0 340M   23348 16612  S  2.0  0.1  0:00.03 /usr/libexec/gnome-c
17990 pacotoh   20    0 2897M 43768 32780  S  2.0  0.3  0:00.03 /usr/bin/gjs /usr/sh
17994 pacotoh   39   19  628M 36488 18920  S  2.0  0.2  0:00.03 /usr/libexec/tracker
 902 root      20    0 11268 5628   5016  S  1.3  0.0  0:20.87 /usr/lib/bluetooth/b
6298 pacotoh   20    0 14.8G 385M   88708  S  1.3  2.5  0:49.19 /usr/share/teams/tea
17816 pacotoh   20    0  635M 59736 41856  S  1.3  0.4  0:00.78 eog /home/pacotoh/Pi
17935 pacotoh   20    0 21104 6312   3696  R  1.3  0.0  0:00.67 htop
1573 pacotoh   20    0  460M 7444   6756  S  0.7  0.0  0:00.12 /usr/libexec/xdg-doc
1734 pacotoh   20    0 26.5G 197M   110M  S  0.7  1.3  0:25.76 /usr/lib/xorg/Xorg v
2234 pacotoh   20    0 4372M 508M   130M  S  0.7  3.3  0:00.09 /usr/bin/gnome-shell

F1Help F2Setup F3Search F4Filter F5Tree F6SortBy F7Nice F8Nice F9Kill F10Quit
```

# Listado de procesos: pstree

```
pacotoh@blade:~$ pstree
systemd├─ModemManager─2*[{ModemManager}]
      ├─NetworkManager─2*[{NetworkManager}]
      ├─accounts-daemon─2*[{accounts-daemon}]
      ├─acpid
      ├─avahi-daemon─avahi-daemon
      ├─bluetoothd
      ├─colord─2*[{colord}]
      ├─cron
      ├─cups-browsed─2*[{cups-browsed}]
      ├─cupsd
      ├─dbus-daemon
      ├─fwupd─4*[{fwupd}]
      └─gdm3├─gdm-session-wor├─gdm-x-session├─Xorg─21*[{Xorg}]
          │               │               │   └─gnome-session-b─2*[{gnome-sessi+
          │               │               │       └─2*[{gdm-x-session}]
          │               └─2*[{gdm-session-wor}]
          └─2*[{gdm3}]
      └─gnome-keyring-d─3*[{gnome-keyring-d}]
      └─irqbalance─{irqbalance}
      └─2*[{kerneloops}]
      └─mount.ntfs
      └─networkd-dispat
      └─nvidia-persiste
      └─packagekitd─2*[{packagekitd}]
      └─polkitd─2*[{polkitd}]
      └─power-profiles-─2*[{power-profiles-}]
      └─rsyslogd─3*[{rsyslogd}]
      └─rtkit-daemon─2*[{rtkit-daemon}]
      └─snapd─27*[{snapd}]
      └─switcheroo-cont─2*[{switcheroo-cont}]
      └─systemd└─(sd-pam)
```

# Listado de procesos: System Monitor

**System Monitor** es la herramienta gráfica de Ubuntu para listar y realizar operaciones sobre procesos (similar al administrador de tareas de Windows).

Processes										
Process Name	User	% CPU	ID	Memory	Disk read total	Disk write total	Disk read	Disk write	Priority	
at-spi2-registr...	pacotoh	0,00	2370	802,8 kB	81,9 kB	N/A	N/A	N/A	Normal	
at-spi-bus-lau...	pacotoh	0,00	1994	737,3 kB	32,8 kB	N/A	N/A	N/A	Normal	
bash	pacotoh	0,00	16100	2,3 MB	11,5 MB	192,5 kB	N/A	N/A	Normal	
dbus-daemon	pacotoh	0,00	1529	1,9 MB	782,3 kB	N/A	N/A	N/A	Normal	
dbus-daemon	pacotoh	0,00	2000	720,9 kB	20,5 kB	N/A	N/A	N/A	Normal	
dconf-service	pacotoh	0,00	2341	856,1 kB	77,8 kB	553,0 kB	N/A	N/A	Normal	
evolution-addr...	pacotoh	0,00	2347	3,4 MB	2,1 MB	36,9 kB	N/A	N/A	Normal	
evolution-alarm-notify	pacotoh	0,00	2442	16,1 MB	2,7 MB	N/A	N/A	N/A	Normal	
evolution-calendar-factory	pacotoh	0,00	2331	4,0 MB	5,1 MB	N/A	N/A	N/A	Normal	
evolution-source-registry	pacotoh	0,00	2308	4,0 MB	3,3 MB	N/A	N/A	N/A	Normal	
firefox	pacotoh	0,00	3703	599,8 MB	531,9 MB	1,5 GB	N/A	N/A	Normal	
gdm-x-session	pacotoh	0,00	1664	536,6 kB	884,7 kB	N/A	N/A	N/A	Normal	
gjs	pacotoh	0,00	2367	5,0 MB	45,1 kB	N/A	N/A	N/A	Normal	
gjs	pacotoh	0,00	2519	5,0 MB	20,5 kB	N/A	N/A	N/A	Normal	
gnome-boxes-search-provider	pacotoh	0,00	19922	852,0 kB	N/A	N/A	N/A	N/A	Normal	
gnome-calculator-search-provider	pacotoh	0,00	19924	6,9 MB	N/A	N/A	N/A	N/A	Normal	
gnome-calendar	pacotoh	0,00	6088	14,2 MB	1,5 MB	N/A	N/A	N/A	Normal	

# Procesos en Primer y Segundo plano

Cuando se inicia un proceso existen dos maneras de ejecutarlo:

- En primer plano (foreground)
- En segundo plano (background)

Para ejecutar un proceso en segundo plano en UNIX se usa el símbolo & al final de la ejecución del comando:

Número de  
procesos en  
segundo plano

PID

```
pacotoh@blade:~$ firefox &  
[2] 16445  
pacotoh@blade:~$ █
```

# Procesos en Primer y Segundo plano

Comandos útiles para trabajar con procesos de primer y segundo plano:

- **jobs**: sirve para obtener el listado de procesos en segundo plano.
- **fg**: usando **fg % [PID]** pasamos un proceso de segundo a primer plano.
- **bg**: usando **bg [ID]** pasamos un proceso de primer a segundo plano (importante! → El proceso debe estar detenido antes).

# Ejercicio procesos Primer y Segundo plano

1. Crea un proceso en primer plano usando la terminal (puedes usar la calculadora de gnome, gedit, o cualquier otra aplicación de escritorio).

¿Qué ocurre en la terminal utilizada para crear el proceso? ¿Por qué?

2. Mata el proceso utilizando una interrupción de teclado dentro de la terminal (Ctrl + C). Para crear un proceso en segundo plano podemos utilizar el símbolo & después del comando. ¿Qué ocurre en la terminal después de usarlo? ¿Por qué?

# Redirección de entrada y salida estándares y de error

Tipos de redirección:

## **De salida estándar hacia fichero:**

proceso > fichero → Machaca el fichero con la salida del proceso

proceso >> fichero → Añade la salida del proceso al final del fichero

Ejemplos:

*cat /etc/hosts > fichero\_hosts*

*echo ultima\_linea >> fichero\_hosts*



# Redirección de entrada y salida estándares y de error

## **De salida de error hacia fichero:**

proceso 2> fichero → Machaca el fichero con la salida de error del proceso

proceso 2>> fichero → Añade la salida del proceso al final del fichero

Ejemplos:

ls /no\_existe 2> error.log

ls /tampoco\_existe 2>> error.log

# Redirección de entrada y salida estándares y de error

**De salida estándar de proceso a entrada estándar de otro:**

proceso1 | proceso2 → El proceso2 es alimentado por la salida del proceso1

Ejemplo:

```
ls -l | grep root
```

# Creación de un proceso: fork y CreateProcess

- En UNIX → la llamada para crear un nuevo proceso es **fork**

Después del fork, los procesos padre e hijo tienen la misma imagen de memoria, las mismas cadenas de entorno y los mismos archivos abiertos.

El proceso hijo ejecuta a **execve** (o llamada al sistema similar) después del **fork**, para cambiar su imagen de memoria y ejecutar un nuevo programa.

# Creación de un proceso: fork y CreateProcess

- En Windows → la llamada para crear un nuevo proceso es una función de Win32 (CreateProcess)

La llamada tiene 10 parámetros:

- Programa a ejecutar
- Parámetros de línea de comandos
- Atributos de seguridad
- ...

Además de CreateProcess, Win32 tiene cerca de 100 funciones más para administrar y sincronizar procesos y temas relacionados.

# Ejemplo de fork en C

En esta actividad vamos a compilar y ejecutar un pequeño programa en C que hace uso de la función fork.

Enlace: <https://drive.google.com/file/d/1KY2O1a78DLlgeo6SP37lKwrlr0bj15sr/view?usp=sharing>

Para compilar el programa podemos utilizar:

**gcc fork.c**

Esto nos creará un ejecutable con el nombre por defecto **a.out**.

Para ejecutar este a.out debemos usar el comando ./a.out

Nos mostrará por pantalla la salida del programa.

Introduce una llamada fork() debajo de la ya existente. ¿Qué se imprime? ¿Por qué?

# Concurrencia en Java

La JVM y la biblioteca de clases estándar de Java se han diseñado para soportar programación concurrente.

Clase Process → Funcionalidad básica para procesos

Clase Thread → Funcionalidad básica para hilos. Esta clase implementa la interfaz Runnable.

# Process y ProcessBuilder

Process es una clase abstracta. Se pueden obtener instancias de subclases tuyas que proporcionan implementaciones para una plataforma en particular, con métodos de la clase ProcessBuilder.

Con ProcessBuilder se puede configurar previamente el entorno de ejecución de los procesos que crea y, en particular, redirigir su entrada y salida.

# Process

## Métodos de la clase Process:

Método	Funcionalidad
<code>void destroy()</code> <code>public Process</code> <code>destroyForcibly()</code>	Termina el proceso. El primer método permite una terminación limpia y ordenada del proceso. El segundo lo termina inmediatamente.
<code>int exitValue()</code>	Devuelve el valor de salida, o código de retorno, del proceso. Por convención, un valor 0 indica terminación normal, y otro valor se interpretará como un código de error. Se puede terminar un programa en Java con un código de retorno distinto de 0 con <code>System.exit(código)</code> .
<code>ProcessHandle.Info info()</code>	Devuelve la información actual del proceso.
<code>boolean isAlive()</code>	Comprueba si el proceso está vivo.
<code>long pid()</code>	Devuelve el PID o identificador de proceso.
<code>int waitFor()</code>	Hace que el hilo en ejecución espere hasta que el proceso haya terminado. Devuelve el valor de salida del proceso. Un valor cero se entiende que corresponde a una ejecución sin errores, mientras que un valor distinto de cero corresponde a un código de error. Si el proceso es de un programa en Java, es el valor devuelto por <code>System.exit()</code> , o cero si no se terminó la ejecución con <code>System.exit()</code> .
<code>boolean waitFor(long</code> <code>timeout,</code> <code>TimeUnit unit)</code>	Hace que el hilo en ejecución espere hasta que el proceso haya terminado, durante un tiempo máximo indicado por <code>timeout</code> . Devuelve <code>true</code> si el proceso ha terminado por sí mismo antes del tiempo máximo indicado, y <code>false</code> en caso contrario.



# Ejercicio 1 de ProcessBuilder

Crea un programa que lance un proceso y utilice el método `isAlive()` para comprobar si se sigue ejecutando. El programa debe comprobar cada 3 segundos si el proceso está en ejecución, hasta que ya no esté, y entonces debe terminar. Tras cada comprobación del estado de ejecución, debe mostrar un mensaje indicando ese estado.

Para hacer una pausa con una duración determinada se puede utilizar `Thread.sleep(int tiempo_ms)`.

# Tipos de terminación de procesos

Un proceso no dura para siempre. Estas son las condiciones de terminación de un proceso:

1. Salida normal (voluntaria)
2. Salida por error (voluntaria)
3. Error fatal (involuntaria)
4. Eliminado por otro proceso (involuntaria)

# Terminación de procesos

El proceso hijo realizará su **ejecución completa**, terminando y liberando sus recursos al finalizar.

Esto se produce cuando el hijo realiza la operación **exit** para finalizar su ejecución.

Un proceso padre puede además terminar de forma abrupta un proceso hijo que creó, mediante la operación destroy.

- Esta operación **elimina el proceso hijo** indicado liberando sus recursos en el sistema operativo subyacente.
- En java, los recursos correspondientes los eliminará el **garbage collector** cuando considere.

# Tipos de terminación de procesos

## 1. Salida normal (voluntaria)

La mayoría de procesos terminan cuando ha acabado su trabajo.

Cuando un programa termina realiza una llamada al sistema para indicar al sistema operativo que ha terminado.

En UNIX la llamada es **exit**, mientras que en Windows tenemos la llamada `ExitProcess`.

La terminación voluntaria puede ocurrir también en los programas con interfaz gráfica, haciendo clic en la salida del programa (x en la mayoría de interfaces).

# Tipos de terminación de procesos

## 2. Salida por error (voluntaria)

Esta terminación ocurre si **el proceso descubre un error**.

Un ejemplo es el de intentar compilar un programa que no existe:

```
→ Downloads gcc no_existe.c  
cc1: fatal error: no_existe.c: No such file or directory  
compilation terminated.  
→ Downloads █
```

# Tipos de terminación de procesos

## **3. Error fatal (involuntaria)**

Esta terminación viene dada por un error fatal producido por un proceso, a menudo debido a un error en el programa.

Ejemplos:

- Ejecutar una instrucción ilegal
- Referencias a memoria no existentes
- División por 0

# Tipos de terminación de procesos

## 4. Eliminado por otro proceso (involuntaria)

Ejecución de una llamada al sistema que indica al SO que elimine otros procesos.

En UNIX tenemos la llamada **kill**.

En Win32 la llamada es **TerminateProcess**.

El proceso que elimina debe tener la **autorización necesaria** para eliminar el otro proceso.

# Señales de interrupción: Ctrl + C y Ctrl + Z

En la terminal de Linux podemos enviar señales para interrumpir procesos mediante las combinaciones:

- **Control + C**: Envía una señal **SIGINT**, que interrumpe el proceso.
- **Control + Z**: Envía una señal **SIGTSTP** a un proceso en primer plano, haciendo que pase a segundo plano en estado suspendido.

SIGINT y SIGTSTP son señales del sistema operativo. Un buen artículo que explica estas señales y otras es:

<https://www.howtogeek.com/devops/linux-signals-hacks-definition-and-more/>



# Terminar procesos

Existen 3 comandos cuyo objetivo es terminar con un proceso o aplicación:

- **Kill:** Si se quiere matar un proceso creado por el usuario se puede usar el comando kill de la siguiente forma:

*kill -9 [PID]*

El -9 indica que queremos matar el proceso.

PID es el identificador del proceso que queremos matar.

# Terminar procesos

- **pkill**: Si, en lugar de utilizar el PID del programa, queremos usar el nombre específico, usaremos pkill:

*pkill firefox*

- **killall**: Si se quiere matar el proceso y todos los procesos que dependan de dicho proceso:

*killall firefox*

# Ejercicio de procesos

Entra en la máquina virtual y haz un listado de los procesos ejecutados en el sistema.

1. Utiliza el comando **grep** para filtrar por aquellos procesos cuyo usuario sea root.
2. Ahora utiliza la opción para **filtrar por usuario** para mostrar solo aquellos procesos cuyo usuario sea el tuyo.
3. Abre **gedit** y busca el **PID** del proceso para matarlo, utiliza el comando **kill -9 [PID]**. Busca de nuevo el proceso. Muestra el ps con el PID del proceso antes de matarlo y muestra el después.
4. Busca la alternativa para matar el proceso por nombre, no por PID. Abre de nuevo gedit y mata el proceso por su nombre usando el comando. Muestra el comando utilizado.

# Ejercicio de terminación de procesos con kill

1. Investiga la opción de kill que te da una lista de señales de sistema disponibles. Utiliza esta opción e indica cuál de las señales es

2. Mata un proceso con killall.

3. Mata un proceso con kill -9.

4. ¿Qué diferencia existe entre el comando killall y el comando kill?

Más información acerca de las señales en UNIX se puede encontrar en:

[https://linux.die.net/Bash-Beginners-Guide/sect\\_12\\_01.html](https://linux.die.net/Bash-Beginners-Guide/sect_12_01.html)

# Prioridades en procesos

Cuando el núcleo (kernel) del sistema operativo retira un proceso debe decidir **cuál será el siguiente en entrar en ejecución**. Esta decisión se realiza mediante un cálculo de prioridades.

Todos los procesos en Linux se ejecutan con una prioridad → Un NÚMERO ENTERO.

Este valor de prioridad varía **GENERALMENTE** entre **-20 y 19**, siendo:

**-20** → Prioridad más favorable o más alta

**19** → Prioridad más baja o menos favorable

# Prioridades en procesos

En los comandos `ps`, `top` y `htop` podemos ver la prioridad en la columna `NI`. Este nombre de columna viene dado por el comando **`nice`**, que sirve para dar un valor de prioridad específico a un proceso concreto.

Un ejemplo de este comando `nice` es:

```
nice -n 10 gedit &
```

De esta forma se ejecutará el comando `gedit` en segundo plano con una prioridad de 10, por lo que tendrá menos prioridad que la mayoría de procesos ejecutados en el sistema.

**El valor de `nice` por defecto de los procesos iniciados por un usuario regular es 0.**

# Prioridades en procesos

Si tenemos un proceso ya existente y queremos cambiar su prioridad podemos utilizar el comando renice.

```
pacotoh@blade:~$ renice -n 15 21723
21723 (process ID) old priority 10, new priority 15
pacotoh@blade:~$ renice -n 3 21723
renice: failed to set priority for 21723 (process ID): Permission denied
pacotoh@blade:~$ sudo renice -n 3 21723
[sudo] password for pacotoh:
21723 (process ID) old priority 15, new priority 3
```

# Demonios

Un demonio es un script, un proceso normalmente **cargado en memoria** esperando una señal para ser ejecutado.

- Aunque estén cargados en memoria no significa que ocupen CPU
- Se ejecutan en segundo plano (background)
- Normalmente, cada demonio tiene asociado un script (/etc/init.d/)

¿Qué podemos hacer con un demonio?

- Iniciallo con el comando **start**
- Pararlo con el comando **stop**
- Reiniciarlo con el comando **restart** (esta opción es interesante cuando cambiamos archivos de configuración, porque vuelve a leer dichos archivos cambiados)



# Demonios

Son útiles para ejecutar:

- Programas independientes de una sesión de usuario
- Procesos que deben iniciar de manera automática al arrancar el sistema
- Servicios que permanecen a la espera de ejecutar una tarea específica

# Demonios

Demonios listados en /etc/init.d/ → Son shell scripts, se pueden revisar y hasta modificar (no hacer, por ahora).

```
pacotoh@blade:~$ ls /etc/init.d/
acpid          grub-common    rsync
alsa-utils     hwclock.sh    saned
anacron        irqbalance    speech-dispatcher
apparmor       kerneloops    spice-vdagent
appport        keyboard-setup.sh sysstat
avahi-daemon   kmod          udev
bluetooth      lvm2          ufw
console-setup.sh lvm2-lvmpolld unattended-upgrades
cron           openvpn       uuidd
cups           plymouth      virtualbox
cups-browsed   plymouth-log  whoopsie
dbus           procps        x11-common
gdm3           pulseaudio-enable-autospawn
pacotoh@blade:~$
```

# Servicios en Linux

Antiguamente  
por el nuevo

Para gestión

Comando para

**systemd**

UNIT FILE	STATE	VENDOR PRESET
proc-sys-fs-binfmt_misc.automount	static	-
-.mount	generated	-
boot-efi.mount	generated	-
dev-hugepages.mount	static	-
dev-mqueue.mount	static	-
proc-sys-fs-binfmt_misc.mount	disabled	disabled
run-qemu.mount	enabled	enabled
snap-bare-5.mount	enabled	enabled
snap-core18-2560.mount	enabled	enabled
snap-core18-2566.mount	enabled	enabled
snap-core20-1611.mount	enabled	enabled
snap-core20-1623.mount	enabled	enabled
snap-eclipse-48.mount	enabled	enabled
snap-eclipse-61.mount	enabled	enabled
snap-firefox-1794.mount	enabled	enabled
snap-firefox-1810.mount	enabled	enabled
snap-gnome\x2d3\x2d28\x2d1804-161.mount	enabled	enabled
snap-gnome\x2d3\x2d38\x2d2004-112.mount	enabled	enabled
snap-gnome\x2d3\x2d38\x2d2004-115.mount	enabled	enabled
snap-gtk\x2dcommon\x2dthemes-1535.mount	enabled	enabled
snap-intellij\x2ddidea\x2dcommunity-384.mount	enabled	enabled
snap-intellij\x2ddidea\x2dcommunity-387.mount	enabled	enabled
snap-kde\x2dframeworks\x2d5\x2d96\x2dqt\x2d5\x2d15\x2d5\x2dcore20-3.mount	enabled	enabled
snap-kde\x2dframeworks\x2d5\x2d96\x2dqt\x2d5\x2d15\x2d5\x2dcore20-7.mount	enabled	enabled
snap-obs\x2dstudio-1284.mount	enabled	enabled
snap-snap\x2dstore-582.mount	enabled	enabled
snap-snap\x2dstore-592.mount	enabled	enabled
snap-snapd-16292.mount	enabled	enabled
snap-snapd-16778.mount	enabled	enabled
snap-snapd\x2ddesktop\x2dintegration-14.mount	enabled	enabled
sys-fs-fuse-connections.mount	static	-
sys-kernel-config.mount	static	-
sys-kernel-debug.mount	static	-
sys-kernel-tracing.mount	static	-
acpid.path	enabled	enabled
apport-autoreport.path	enabled	enabled
cups.path	enabled	enabled

lines 1-38

ha sido sustituido

o **systemctl**.

istema:

Serv

Con el  
sistem

system

OJO: Q

UNIT	LOAD	ACTIVE	SUB	DESCRIPTION
accounts-daemon.service	loaded	active	running	Accounts Service
acpid.service	loaded	active	running	ACPI event daemon
alsa-restore.service	loaded	active	exited	Save/Restore Sound Card State
apparmor.service	loaded	active	exited	Load AppArmor profiles
apport.service	loaded	active	exited	LSB: automatic crash report generation
avahi-daemon.service	loaded	active	running	Avahi mDNS/DNS-SD Stack
blk-availability.service	loaded	active	exited	Availability of block devices
bluetooth.service	loaded	active	running	Bluetooth service
colord.service	loaded	active	running	Manage, Install and Generate Color Profiles
console-setup.service	loaded	active	exited	Set console font and keymap
cron.service	loaded	active	running	Regular background program processing daemon
cups-browsed.service	loaded	active	running	Make remote CUPS printers available locally
cups.service	loaded	active	running	CUPS Scheduler
dbus.service	loaded	active	running	D-Bus System Message Bus
gdm.service	loaded	active	running	GNOME Display Manager
irqbalance.service	loaded	active	running	irqbalance daemon
kerneloops.service	loaded	active	running	Tool to automatically collect and submit kernel o
keyboard-setup.service	loaded	active	exited	Set the console keyboard layout
kmmod-static-nodes.service	loaded	active	exited	Create List of Static Device Nodes
lvm2-monitor.service	loaded	active	exited	Monitoring of LVM2 mirrors, snapshots etc. using
ModemManager.service	loaded	active	running	Modem Manager
● modprobe@efi_pstore.service	loaded	failed	failed	Load Kernel Module efi_pstore
● modprobe@pstore_blk.service	loaded	failed	failed	Load Kernel Module pstore_blk
● modprobe@pstore_zone.service	loaded	failed	failed	Load Kernel Module pstore_zone
● modprobe@ramoops.service	loaded	failed	failed	Load Kernel Module ramoops
networkd-dispatcher.service	loaded	active	running	Dispatcher daemon for systemd-networkd
NetworkManager-wait-online.service	loaded	active	exited	Network Manager Wait Online
NetworkManager.service	loaded	active	running	Network Manager
nvidia-persistenced.service	loaded	active	running	NVIDIA Persistence Daemon
openvpn.service	loaded	active	exited	OpenVPN service
packagekit.service	loaded	active	running	PackageKit Daemon
plymouth-quit-wait.service	loaded	active	exited	Hold until boot process finishes up
plymouth-read-write.service	loaded	active	exited	Tell Plymouth To Write Out Runtime Data
plymouth-start.service	loaded	active	exited	Show Plymouth Boot Screen
polkit.service	loaded	active	running	Authorization Manager
power-profiles-daemon.service	loaded	active	running	Power Profiles daemon
qemu-kvm.service	loaded	active	exited	QEMU KVM preparation - module, ksm, hugepages

lines 1-38

# Servicios en Linux

Para cada servicio se puede ejecutar comandos del tipo:

**systemctl [ACCIÓN] [SERVICIO]**

Acción	Función
status	Muestra el estado de ejecución del servicio.
start	Arranca el servicio si no está en ejecución.
stop	Para el servicio si está en ejecución.
restart	Reinicia el servicio. Es decir, lo para si está en ejecución, y luego lo arranca.
reload	Alternativa a <b>restart</b> . Se utiliza para que el servicio tome en consideración los cambios realizados en su configuración, pero sin que se reinicie el servicio.
enable	Establece el arranque automático del servicio cuando arranca el sistema.
disable	Establece que el servicio no arranque automáticamente cuando arranca el sistema.



# Servicios en Linux

```
pacotoh@blade:~$ systemctl status cron.service
```

```
● cron.service - Regular background program processing daemon
   Loaded: loaded (/lib/systemd/system/cron.service; enabled; vendor preset: enabled)
   Active: active (running) since Sat 2022-09-17 09:23:06 CEST; 3h 49min ago
     Docs: man:cron(8)
  Main PID: 1094 (cron)
    Tasks: 1 (limit: 18388)
   Memory: 952.0K
      CPU: 140ms
   CGroup: /system.slice/cron.service
           └─1094 /usr/sbin/cron -f -P
```

```
sep 17 12:30:01 blade CRON[27072]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
sep 17 12:30:01 blade CRON[27072]: pam_unix(cron:session): session closed for user root
sep 17 12:35:01 blade CRON[27180]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
sep 17 12:35:01 blade CRON[27180]: pam_unix(cron:session): session closed for user root
sep 17 12:45:01 blade CRON[27328]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
sep 17 12:45:01 blade CRON[27328]: pam_unix(cron:session): session closed for user root
sep 17 12:55:01 blade CRON[27689]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
sep 17 12:55:01 blade CRON[27689]: pam_unix(cron:session): session closed for user root
sep 17 13:05:01 blade CRON[28967]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
sep 17 13:05:01 blade CRON[28967]: pam_unix(cron:session): session closed for user root
```

# Automatización de tareas

## Ventajas de la automatización

1. Reducción de costos
2. Aumento de la productividad
3. Mayor confiabilidad: menor necesidad de control y resolución de problemas

# Automatización de tareas

Hay muchas formas de automatizar procesos, en esta unidad vamos a ver:

- **Cron**
- **At**
- **Timers**

Gráficas:

- Zeit: <https://github.com/loimu/zeit>
- Gnome-schedule: <https://sourceforge.net/projects/gnome-schedule/>



# CRON

Es un demonio que se ejecuta en el mismo instante en el que arranca el sistema, comprobando si existe alguna tarea para ser ejecutada de acuerdo a la hora configurada en el sistema. Nos permite programar tareas para su ejecución en un determinado momento o de forma periódica.

Formado por:

- **crond**: Es el demonio que se encarga de ejecutar las tareas
- Fichero **/etc/crontab**: En este fichero se guardan las tareas programadas
- Comando **crontab**: Utilizado para administrar las tareas

<https://docs.oracle.com/cd/E19253-01/817-0403/6mg741bt4/index.html>

# CRON

Para comprobar el estado de crond tenemos varias opciones:

- **/etc/rc.d/init.d/crond status**
- **/etc/init.d/crond status**

Otra opción, si se tiene instalado el comando service:

- **service crond status**

Si no se tiene instalado cron en el sistema se puede instalar con el comando:

- **apt-get install cron**

# CRON

Los ficheros más importantes para el funcionamiento de cron son:

- **/etc/crontab** (fichero de configuración)
- **/etc/init.d/cron** (fichero de inicio y parada del demonio)
- **/var/log/cron** (sistema de informes/log)

# Crontab

Si editamos el fichero `/etc/crontab` podremos ver algo similar a lo siguiente:

```
sudo nano /etc/crontab
GNU nano 6.2 /etc/crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
# You can also override PATH, but by default, newer versions inherit it from the
#PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# Example of job definition:
# .----- minute (0 - 59)
# | .----- hour (0 - 23)
# | | .----- day of month (1 - 31)
# | | | .----- month (1 - 12) OR jan,feb,mar,apr...
# | | | | .----- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fr
# | | | | |
# * * * * * user-name command to be executed
17 * * * * root cd / && run-parts --report /etc/cron.hourly
25 6 * * * root test -x /usr/sbin/anacron || ( cd / && run-parts --repo
47 6 * * 7 root test -x /usr/sbin/anacron || ( cd / && run-parts --repo

^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute   ^C Location
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify   ^_ Go To Line
```

# Crontab

- Existen **directorios** en Linux que nos facilitan la ejecución de tareas repetitivas (de esta forma simplificamos el fichero crontab).

- /etc/cron.**hourly**
- /etc/cron.**daily**
- /etc/cron.**weekly**
- /etc/cron.**monthly**

```
→ ~ ls -ld /etc/cron*  
drwxr-xr-x 2 root root 4096 Aug 10 16:42 /etc/cron.d  
drwxr-xr-x 2 root root 4096 Jul 31 20:19 /etc/cron.daily  
drwxr-xr-x 2 root root 4096 Jun 29 22:44 /etc/cron.hourly  
drwxr-xr-x 2 root root 4096 Jun 29 22:44 /etc/cron.monthly  
-rw-r--r-- 1 root root 1136 Aug  6 2021 /etc/crontab  
drwxr-xr-x 2 root root 4096 Jun 29 22:49 /etc/cron.weekly
```

# Crontab

Para controlar quién puede utilizar cron se utilizan los ficheros:

- **/etc/cron.allow**: solo los usuarios presentes en este fichero podrán ejecutar cron.
- **/etc/cron.deny**: los usuarios presentes en este fichero tendrán denegada la ejecución de cron.

```
$ cat /etc/cron.d/cron.deny
daemon
bin
smtp
nuucp
listen
nobody
noaccess
```

# Crontab

¿Cómo editamos las tareas en crontab?

Mediante el comando **crontab -e**

La primera vez que ejecutamos crontab -e se nos pide qué editor queremos usar (para facilitar la edición vamos a utilizar /bin/nano):

```
~ crontab -e
no crontab for pacotoh - using an empty one

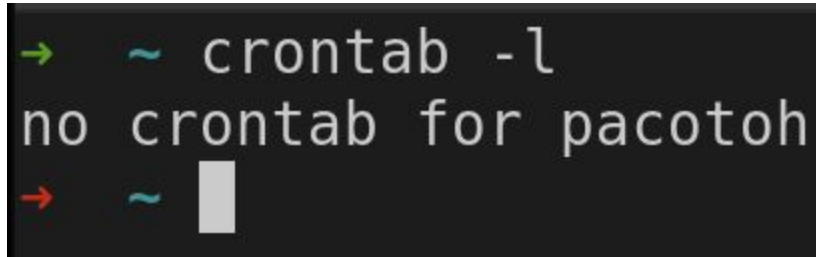
Select an editor. To change later, run 'select-editor'.
 1. /bin/nano          <---- easiest
 2. /usr/bin/vim.tiny
 3. /usr/bin/code
 4. /bin/ed

Choose 1-4 [1]:
```

# Crontab

Para comprobar el listado de tareas planificadas en cron usamos el comando:

**crontab -l**



```
→ ~ crontab -l
no crontab for pacotoh
→ ~
```

En este caso no tenemos planificadas tareas para el usuario pacotoh. En otro caso nos mostraría las tareas listadas en el mismo formato usado en el fichero crontab.



# Crontab

Otros parámetros de crontab son:

- Opción **-r** → Borra el fichero de configuración del usuario
- Opción **-u usuario** → Sirve para especificar el usuario propietario de la tarea

# Crontab - Ejercicio

1. Crea un usuario con tu nombre y contraseña 1234 (usa los comandos **useradd** y **passwd**)
2. Con el usuario root, utiliza crontab para **denegar la planificación a tu usuario**.
3. Intenta ejecutar crontab para añadir nuevas tareas con tu usuario.
4. De nuevo, con el usuario root, utiliza crontab para permitir la planificación a tu usuario. ¿Qué pasa si se deniega y se permite a la vez?

MUESTRA CAPTURAS DE CADA UNO DE LOS PASOS REALIZADOS.

RESPONDE A LA PREGUNTA.

# Anacron

Cron funciona siempre que el sistema esté activo, pero, ¿qué pasa con las tareas programadas que no se llevan a cabo porque están planificadas en un momento en el que el sistema está apagado?

Hay un programador de tareas llamado Anacron que se encarga de revisar las tareas que no se han llevado a cabo y las realiza.

Podemos instalarlo usando:

**apt-get install anacron**

# At

- Para **tareas puntuales** o que solo se ejecutan en un momento concreto.
- El formato puede ser el siguiente:
  - HH:MM con sufijos AM o PM para establecer hora y minuto
  - MMDDYY, MM/DD/YY, DD.MM.YY o YYYY-MM-DD
  - now +tiempo (minutes, hours, days o weeks)
  - Reconoce las palabras today, tomorrow, midnight, noon y teatime

La utilización de at sigue el siguiente esquema:

**HH[:]MM[am | pm] [Mes día] programa\_script**

- **atq**: muestra las tareas
- **at -c N**: muestra el contenido de la tarea N (tarea número N)
- **atrm**: borra una tarea

# At

Ficheros de configuración de at:

- **/etc/at.allow**
- **/etc/at.deny**: lo crea automáticamente con un listado de usuarios denegados, al contrario que cron, que no crea ninguno de los ficheros.

Si no existe ninguno de estos, solo el **root** puede ejecutar at.

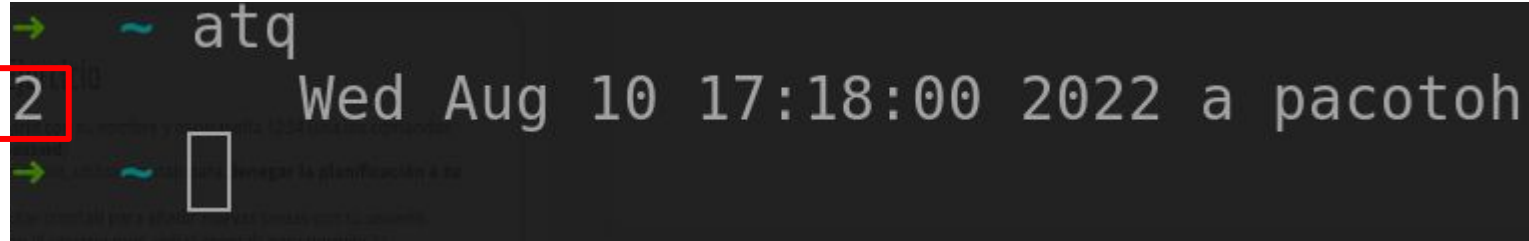
# At

Ejemplo de uso de at:

```
→ ~ at now +2 minutes  
warning: commands will be executed using /bin/sh  
at Wed Aug 10 17:14:00 2022  
at> touch /home/pacotoh/test  
at> <EOT>  
job 1 at Wed Aug 10 17:14:00 2022
```

# At

Para ver la lista de tareas planificadas usamos el comando **atq**:



```
→ ~ atq
2      Wed Aug 10 17:18:00 2022 a pacotoh
→ ~ [ ] [ ]
```

El número **2** de la primera columna es el identificador de la tarea. Podemos usar el comando **at -c 2** para poder el contenido de la tarea programada.

# At

Otro ejemplo del comando at puede ser:

- **at 20:00 tomorrow:** se planifica la tarea para las 20:00 de mañana
- **at 01242023:** se planifica la tarea para el día 24 de enero de 2023 (se establece la hora en la que se ha planificado la tarea)
- **at 05:00 PM:** se planifica la tarea para las 17:00
- **at 17:00:** en este caso no hace falta AM o PM



# Timers con systemd

- Permiten programar tareas que se repiten de forma periódica o de forma puntual.
- Hay dos tipos de timers:
  - **Monolítico:** ejecución de tareas tras el inicio del sistema

**OnBootSec=15min**

**OnUnitActiveSec=1w**

- **Tiempo real:** ejecución de tareas de forma periódica

**OnCalendar=Sat \*-\*-1..7 19:00:00 (DayOfWeek Year-Month-Day Hour:Minute:Second)**

# Timers con systemd

Es necesario crear un fichero .timer y otro .service con el mismo nombre para el servicio que ejecutará el timer.

Ejemplo de Timers en Arch Linux:

[https://wiki.archlinux.org/title/Systemd\\_\(Espa%C3%B1ol\)/Timers\\_\(Espa%C3%B1ol\)](https://wiki.archlinux.org/title/Systemd_(Espa%C3%B1ol)/Timers_(Espa%C3%B1ol))

# Timers con systemd

Con el comando `systemd-run` nos evitamos tener que crear un fichero `.timer` y `.service` asociado a dicho timer.

Ejemplo:

**`systemd-run --on-active="5h" /bin/touch /tmp/foo`**

El comando `touch (/bin/touch)` se ejecutará dentro de 5 horas, creando un fichero dentro del directorio `/tmp/` llamado `foo`.