

S.I.

Unidad 11

Administración de GNU/Linux

Gestión de procesos



Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

Gestión de procesos



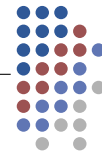
• Índice

- Introducción
- Monitorización de procesos
- Señales a procesos
- Prioridades de procesos
- Planificación de procesos
- Trabajos
- Directorio /proc

Ivens Huertas

2

Gestión de procesos



• Índice

- Introducción
- Monitorización de procesos
- Señales a procesos
- Prioridades de procesos
- Planificación de procesos
- Trabajos
- Directorio /proc

Ivens Huertas

3

Introducción



- Un usuario puede ejecutar **varios procesos a la vez**
 - *Recuerda: GNU/Linux = multitarea*
- En cada terminal, **sólo uno** de los procesos en ejecución puede **interactuar directamente** con el usuario
 - Se trata del proceso en **primer plano**
 - **Foreground**
 - No se puede hacer otra tarea hasta que el proceso termine
 - *Ejemplo: al lanzar el siguiente comando, va a tardar un buen rato, por lo que no podremos hacer otra cosa que esperar a que termine*

\$!s -R /

Ivens Huertas

4

Introducción



- El resto de procesos se ejecutan en **segundo plano**
 - **Background**
 - La salida de un proceso en segundo plano **sí** que se muestra por pantalla **si no se redirige**
- ¿Cómo se puede ejecutar un proceso en segundo plano?
 - Añadimos al final de la instrucción un **&**

Introducción



Ejemplo:

- Queremos ejecutar desde terminal el programa gráfico **Firefox**

\$firefox

- Vemos que se inicia, pero no podemos utilizar el terminal hasta que finalice la ejecución de *firefox*

Introducción



Ejemplo:

- Para evitar todo esto, debemos ejecutar:

\$firefox &

- Al escribir esto, vemos que GNU/Linux devuelve el control a la consola al instante, mostrando una línea similar a esta:

[1] 17860

- Esto significa que el proceso *firefox* se ha ejecutado con el **PID 17860**
 - El **PID** es el identificador del proceso (**Process ID**)
 - Cada proceso lanzado en el sistema tiene un número identificador

Introducción



• Estados de un proceso

- **Ejecutándose (*running*)**
 - El proceso se está ejecutando, usando CPU
 - Forma parte de la cola de ejecución del kernel

• **En espera (*sleeping*)**

- El proceso se está ejecutando, pero está esperando a que se produzca un evento

Ejemplo:

\$comando1 ; comando2

- **comando1** estará en ejecución
- **comando2** estará en espera hasta que no finalice **comando1**

Introducción



- **Estados de un proceso**

- **Parado (Stopped)**
 - Cuando paramos un proceso, este se queda en este estado esperando a ser “reactivado”
 - Es como si estuviera “en pausa”
- **Zombie**
 - El proceso ha finalizado irregularmente y está en memoria
 - El superusuario debería comprobar que no quedan procesos de este tipo en memoria, no es deseable

Gestión de procesos



- **Índice**

- Introducción
- Monitorización de procesos
- Señales a procesos
- Prioridades de procesos
- Planificación de procesos
- Trabajos
- Directorio /proc

Monitorización de procesos



- **ps**

- Información de los procesos en ejecución **del usuario** que ejecuta el comando
- Sólo aparecen los comandos del terminal actual

- **ps u**

- Lo mismo, pero con más información

R = Ejecutándose
S = En espera
T = Parado
Z = Zombie

- **ps aux**

- Información de todos los procesos **de todos los usuarios** del sistema

¿Cómo filtrar datos?

Monitorización de procesos



- **pgrep**

- Muestra el **PID** del proceso cuyo nombre es pasado por parámetro

\$pgrep firefox

- Otra opción para poder **filtrar** un proceso de todo el listado es utilizar el comando **ps** anidado con **grep**

- *Ejemplo: queremos filtrar el proceso firefox de todo el listado de procesos*

\$ps aux | grep firefox

Monitorización de procesos



- **ps tree**

- Muestra un árbol de procesos
- Nos permite ver quién es el proceso padre de cada proceso

Ivens Huertas

13

Monitorización de procesos



- **top**

- Nos permite ver:
 - Información del sistema (memoria libre, carga del sistema, uptime,...)
 - La lista de procesos y sus estados
- Ordenación
 - Por ocupación de **CPU** → Mayúsculas + P
 - Por ocupación de **memoria** → Mayúsculas + M
- Salir
 - q

```
top - 14:01:50 up 22 min, 1 user, load average: 0.54, 0.18, 0.08
Tareas: 165 total, 1 ejecutar, 130 hibernar, 0 detener, 1 zombie
Mem: 0.0 usuario, 1.0 sist, 0.0 adecuado, 99.0 inact, 0.0 en espera, 0.0 hardw int, 0.0 soft
Kilobytes Mem: 2041304 total, 1029920 libre, 391652 usado, 619724 buffer/cache
Kilobytes Intercambio: 969960 total, 969960 libre, 0 usado, 1495976 dispon Mem
```

PID	USUARIO	PR	NI	VIRT	RES	SHR	%CPU	%MEM	TIME+	ORIGEN
1840	root	20	0	326452	54800	31284	5	2.3	2.7	0:01.90 Xorg
2311	ivens	20	0	483508	34300	28380	5	1.3	1.7	0:00.45 xfce4-terminal
2343	ivens	20	0	49484	3756	3152	R	1.0	0.2	0:00.43 top
1834	root	20	0	0	0	0	1	0.3	0.0	0:00.02 kworker/0:2-3
2877	ivens	20	0	126260	2208	1096	5	0.3	0.1	0:00.05 VBoxClient
2117	ivens	20	0	193764	21916	17884	5	0.3	1.1	0:00.35 xfwm4
1	root	20	0	195568	8820	6616	5	0.0	0.4	0:01.54 systemd
2	root	20	0	0	0	0	5	0.0	0.0	0:00.00 kthreadd
4	root	0	-20	0	0	0	1	0.0	0.0	0:00.00 kworker/0:0H
5	root	20	0	0	0	0	1	0.0	0.0	0:00.14 kworker/u2:0
6	root	0	-20	0	0	0	1	0.0	0.0	0:00.00 mm_percpu_wq
7	root	20	0	0	0	0	5	0.0	0.0	0:00.12 ksoftirqd/0
8	root	20	0	0	0	0	1	0.0	0.0	0:00.20 rcu_sched
9	root	20	0	0	0	0	1	0.0	0.0	0:00.00 rcu_bh
10	root	rt	0	0	0	0	5	0.0	0.0	0:00.00 migration/0
11	root	rt	0	0	0	0	5	0.0	0.0	0:00.00 watchdog/0
12	root	20	0	0	0	0	5	0.0	0.0	0:00.00 cpupd/0
13	root	20	0	0	0	0	5	0.0	0.0	0:00.00 kdevtmpfs
14	root	0	-20	0	0	0	1	0.0	0.0	0:00.00 netns
15	root	20	0	0	0	0	5	0.0	0.0	0:00.00 rcu_tasks_kthre
16	root	20	0	0	0	0	5	0.0	0.0	0:00.00 kauditd
17	root	20	0	0	0	0	5	0.0	0.0	0:00.00 khungtaskd
18	root	20	0	0	0	0	5	0.0	0.0	0:00.00 oom_reaper
19	root	0	-20	0	0	0	1	0.0	0.0	0:00.00 writeback

Ivens Huertas

Monitorización de procesos



- **htop**

- Se trata de una **versión modernizada de top**, aunque no suele venir instalada por defecto en las distribuciones

#apt install htop

Nos va a facilitar la gestión de procesos que veremos más adelante en este bloque, tales como mandar señales a procesos (tecla F9)

```
CPU: 0.7% Tasks: 95, 129 thr: 1 running
Mem: 250M/1.95G Load average: 0.11 0.33 0.26
Swap: 375M/947M Uptime: 00:13:37
```

PID	USER	PRI	NI	VIRT	RES	SHR	%CPU	%MEM	TIME+	Command
3326	ivens	20	0	34664	4216	3252	R	0.7	0.2	0:00.48 htop
747	root	20	0	334M	32064	4884	S	0.0	1.6	0:41.07 /usr/lib/xorg/Xorg -core :0 -seat seat0 -auth
768	root	20	0	334M	32064	4884	S	0.0	1.6	0:41.07 /usr/lib/xorg/Xorg -core :0 -seat seat0 -auth
1049	ivens	20	0	724M	44696	13280	S	0.0	2.2	0:11:02 xfdesktop
2621	ivens	20	0	453M	34516	25744	S	0.0	1.7	0:00.67 /usr/bin/xfce4-terminal
1041	ivens	20	0	198M	12752	10004	S	0.0	0.6	0:03.04 xfwm4 --replace
1072	ivens	20	0	215M	484	344	S	0.0	0.0	0:00.15 /usr/lib/at-spi2-core/at-spi2-registrd --use-
1004	ivens	20	0	123M	0	0	S	0.0	0.0	0:00.84 /usr/bin/VBoxClient --draganddrop
1188	ivens	20	0	396M	5384	3392	S	0.0	0.3	0:00.24 /usr/lib/x86_64-linux-gnu/xfce4/panel/wrapper-
1189	ivens	20	0	681M	6860	3932	S	0.0	0.3	0:00.65 /usr/lib/x86_64-linux-gnu/xfce4/panel/wrapper-
739	root	20	0	249M	28	0	S	0.0	0.0	0:00.10 /usr/sbin/VBoxService --pidfile /var/run/vboxa-
436	root	20	0	477M	232	12	S	0.0	0.0	0:00.01 /usr/sbin/NetworkManager --no-daemon
1002	ivens	20	0	123M	0	0	S	0.0	0.0	0:00.84 /usr/bin/VBoxClient --draganddrop
1234	ivens	20	0	681M	6860	3932	S	0.0	0.3	0:00.18 /usr/lib/x86_64-linux-gnu/xfce4/panel/wrapper-
1	root	20	0	155M	2720	2512	S	0.0	0.1	0:01.49 /sbin/init splash
233	root	19	1	106M	6704	6444	S	0.0	0.3	0:00.25 /lib/systemd/systemd-journald
246	root	20	0	46600	256	4	S	0.0	0.0	0:00.20 /lib/systemd/systemd-udev
287	systemd-r	20	0	70748	2248	2080	S	0.0	0.1	0:00.05 /lib/systemd/systemd-resolved
338	root	20	0	38892	4	4	S	0.0	0.0	0:00.00 /usr/sbin/cron -f
345	root	20	0	491M	3484	3112	S	0.0	0.2	0:00.00 /usr/lib/udisks2/udisksd
402	root	20	0	491M	3484	3112	S	0.0	0.2	0:00.00 /usr/lib/udisks2/udisksd
435	root	20	0	491M	3484	3112	S	0.0	0.2	0:00.00 /usr/lib/udisks2/udisksd
470	root	20	0	491M	3484	3112	S	0.0	0.2	0:00.00 /usr/lib/udisks2/udisksd
340	root	20	0	491M	3484	3112	S	0.0	0.2	0:00.08 /usr/lib/udisks2/udisksd
349	avahi	20	0	47256	584	460	S	0.0	0.0	0:00.03 avahi-daemon: running [vbox.local]
365	root	20	0	4552	168	136	S	0.0	0.0	0:00.01 /usr/sbin/acpid
379	root	20	0	288M	64	0	S	0.0	0.0	0:00.02 /usr/lib/accounts-service/accounts-daemon

Ivens Huertas

Gestión de procesos



- **Índice**

- Introducción
- Monitorización de procesos
- Señales a procesos
- Prioridades de procesos
- Planificación de procesos
- Trabajos
- Directorio /proc

Ivens Huertas

16

Señales a procesos



- Una señal es una manera que tiene un proceso de comunicarse con otro
- Las más importantes son:
 - **SIGSTOP** (nº19)
 - Detiene un proceso
 - **SIGCONT** (nº18)
 - Continúa un proceso previamente parado
 - **SIGTERM** (nº15)
 - Termina un proceso de forma ordenada
 - **SIGKILL** (nº9)
 - Mata un proceso

Señales a procesos



- Sólo podremos enviar señales a **nuestros procesos**
 - **root** podrá enviar señales a cualquier proceso
- Para enviar una de estas señales a un proceso, utilizaremos la orden **kill**
- Podemos utilizarla de dos formas:

\$kill nombre_señal proceso

- Ejemplo:

\$kill SIGTERM 1560

\$kill -número_señal proceso

- Ejemplo:

\$kill -15 1560

Señales a procesos



Ejemplo:

- Ejecutamos el navegador **firefox** en segundo plano:

\$firefox &

- Comprobamos cuál es su PID

\$pgrep firefox

- Enviamos una señal de parada a su proceso

\$kill SIGTERM 16507

\$kill -15 16507

A elegir

...y vemos que el proceso *firefox* tiene, por ejemplo, el PID = 16507

Repite el ejemplo, pero ahora enviando una señal para matar el proceso

Gestión de procesos



• Índice

- Introducción
- Monitorización de procesos
- Señales a procesos
- Prioridades de procesos
- Planificación de procesos
- Trabajos
- Directorio /proc

Prioridades de procesos



- Los valores de prioridad que GNU/Linux asigna a un proceso van:
 - Desde el **-20** (máxima prioridad)...
 - ...Hasta el **19** (mínima prioridad)



Prioridades de procesos



- Mediante el comando **nice** podremos establecer la prioridad con la que se ejecuta un proceso

```
$nice [-n valor_nice] [comando]
```

```
$nice -n 7 find / -name "*.mp3"
```

```
$nice -n -15 tar -cvzf backup.tgz ~/datos/
```

nice ≈ "gentil"

"Un proceso muy gentil deja paso al resto de procesos para que utilicen la CPU"

Prioridades de procesos



- Si hemos ejecutado un comando y queremos cambiar su prioridad, podemos hacerlo usando el comando **renice**

```
$renice [nuevo_valor_nice] [PID]
```

```
$renice 19 6535
```

```
$renice -20 1407
```

- La aplicación **htop** también permite modificar los valores nice de forma "gráfica" utilizando las teclas F7 y F8

Gestión de procesos

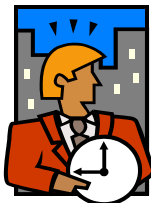


- Índice
 - Introducción
 - Monitorización de procesos
 - Señales a procesos
 - Prioridades de procesos
 - Planificación de procesos
 - Trabajos
 - Directorio /proc

Planificación de procesos

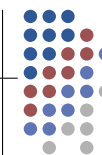


- Existe un proceso que se lanza automáticamente al iniciar el sistema llamado **cron**
- El usuario puede programar la ejecución de tareas y **cron** se encargará de lanzarlas cuando corresponda



- Las tareas se programan en el fichero **/etc/crontab**

Planificación de procesos



- Contenido del fichero **/etc/crontab**
 - Una **tarea** programada por línea
 - **Campos separados** por espacios / tabulaciones
 - Minuto [0 - 59]
 - Hora [0 - 23]
 - Día del mes [1 - 31]
 - Mes [1 - 12]
 - Día de la semana [0 - 6] (Domingo = 0)
 - Usuario que lo lanza
 - Comando

Un asterisco en un campo indicaría cualquier valor

Planificación de procesos



- *Ejemplos:*

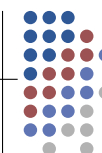
#min	hor	ddm	mes	dds	user	cmd
15	8	*	*	*	paco	/usr/bin/backup
45	19	15	*	*	juan	/home/juan/uno
25	21	7	3	*	root	/usr/bin/otro.sh

El usuario paco lanzará todos los días, a las 8:15h, el programa backup que está en /usr/bin

El usuario juan lanzará todos los días 15 del mes, a las 19:45h el comando uno, localizado en su directorio personal

root lanzará todos los 7 de marzo, a las 21:25h, el comando otro.sh, localizado en /usr/bin

Planificación de procesos



- *Ejemplos:*

#min	hor	ddm	mes	dds	user	cmd
00	11	*	*	6	paco	/home/paco/mas
15	8,9	*	*	*	rosa	/home/rosa/bck
45	19	1-10	*	*	pepe	/usr/bin/otro.sh

El usuario paco lanzará todos los sábados (día 6), a las 11:00h, el programa mas que está en su directorio personal

El usuario rosa lanzará todos los días, a las 8:45h y 9:45h, el comando bck, localizado en su directorio personal

El usuario pepe lanzará los primeros 10 días de cada mes, a las 19:45h, el comando otro.sh, localizado en /usr/bin

Planificación de procesos



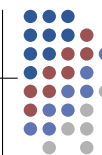
- Ejemplos:

#min	hor	ddm	mes	dds	user	cmd
25	21	7	3,6,9	*	root	/usr/bin/otro2.sh
00	11	*	*	1-5	luis	/usr/bin/listar

root lanzará todos los 7 de marzo, 7 de junio y 7 de septiembre, a las 21:25h, el programa otro2.sh que está en /usr/bin

El usuario luis lanzará de lunes a viernes, a las 11:00h, el comando listar, localizado en /usr/bin

Gestión de procesos



- Índice

- Introducción
- Monitorización de procesos
- Señales a procesos
- Prioridades de procesos
- Planificación de procesos
- Trabajos
- Directorio /proc

Trabajos



- En GNU/Linux, cuando arranca un proceso, se asigna (además del PID) un **número de trabajo**

- Recordemos cuando hemos ejecutado un comando en segundo plano:

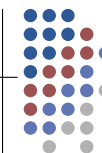
\$firefox &

- Vimos que el sistema mostraba una línea similar a esta:

[1] 17860

- Significa que GNU/Linux ha creado el **trabajo 1** para el proceso 17860

Trabajos



- Un trabajo puede contener varios procesos
 - Si no fuera así, no tendría sentido que existieran los números de trabajo (con el PID valdría)

- ¿Cómo crear un trabajo de varios procesos?

\$(comando1; comando2; comando3) &
[1] 1517

- Se ejecutará en 2º plano el proceso **comando1**
- Cuando éste termine, se ejecutará en 2º plano **comando2**
- Cuando éste termine, se ejecutará en 2º plano **comando3**
- Cuando el último termine, el trabajo habrá concluido

Se ha creado el trabajo **nº1**
El PID del **último** proceso es el 1517

Trabajos



- Al **terminar** un trabajo, el número de trabajo **queda libre** para ser utilizado
 - Esto no ocurre con los PID: el contador sigue incrementándose
- Para ver los trabajos activos en el sistema, utilizaremos el comando **jobs**

\$jobs

Trabajos



Ejemplo:

\$jobs

```
[1] Ejecutando comando1; comando2 &
[2]- Hecho      comando3; comando4
[3]+ Ejecutando comando5; comando6 &
```

- El signo "+" → **Último** trabajo que se ha ejecutado
- El signo "-" → El **anterior** trabajo
- Sin signo → **Otros** trabajos
- En este ejemplo, el trabajo 2 ya ha terminado
 - Se muestra para informar al usuario
 - La próxima vez que se ejecute **jobs** ya no aparecerá en la lista

Trabajos



- En Bash, los trabajos se especifican con el carácter **%**

Ejemplo:

- Matar el trabajo número 3

\$kill -9 %3

Trabajos



- Podemos pasar a **primer plano** un trabajo o proceso que estuviera en **segundo plano**
 - Comando **fg** (foreground)

\$fg %2

*Esto pasaría a un primer plano el **trabajo** número 2*

\$fg 1563

*Esto pasaría a un primer plano el **proceso** con PID = 1563*

Trabajos



- Podemos pasar a **segundo plano** un proceso o trabajo que estuviera en **primer plano**
 - Comando **bg** (background)

\$bg %2

*Esto pasaría a un segundo plano el **trabajo** número 2*

\$bg 1765

*Esto pasaría a un segundo plano el **proceso** con PID = 1765*

Trabajos



☞ Pero para esto último... ¿cómo es posible si, mientras se está ejecutando en primer plano, el usuario no puede continuar trabajando (no puede teclear nada)?

☞ Solución:

- Disponemos de **varias consolas**
- Podemos parar el proceso previamente
 - **Ctrl+Z**
 - **\$kill SIGSTOP proceso/trabajo**

Gestión de procesos



- Índice
 - Introducción
 - Monitorización de procesos
 - Señales a procesos
 - Prioridades de procesos
 - Planificación de procesos
 - Trabajos
 - Directorio /proc

Directorio /proc



- GNU/Linux contiene un pseudo-sistema de ficheros montado en el directorio /proc
- Es un **directorio virtual**
 - Los ficheros que contiene **no están almacenados** en ningún disco duro
 - Son una forma curiosa que tiene GNU/Linux de **obtener información del kernel** en tiempo real
 - ps, top,... cogen la información de este directorio

Directorio /proc



- Para leer la información de estos ficheros virtuales, usaremos un simple **cat**
- Algunos de ellos son:
 - **cpuinfo** Información de la/s CPU del sistema
 - **meminfo** Información de la memoria del sistema
 - **partitions** Particiones montadas en el sistema
 - **swaps** Particiones destinadas a swapping
 - **version** Información del kernel
 - ...

\$cat /proc/cpuinfo

Directorio /proc



- Para cada proceso que está en ejecución, encontraremos un subdirectorio dentro de */proc* con la siguiente apariencia:

/proc/numero_proceso

- Dentro de él encontraremos información relativa a dicho proceso