

PRÁCTICA 1: GESTIÓN DE PROCESOS EN LINUX

- 1. Introducción
- 2. Lanzar un programa desde la línea de comando
- 3. Poner un programa en 2º Plano
- 4. Listado de Procesos
- 5. Matando Procesos
 - 5.1. Utilizando `jobs`
 - 5.2. Un poc más acerca de `kill`
- 6. Otros Comandos útiles
 - 6.1. `top`
 - 6.2. `nice`
- 7. Administrando procesos utilizando la interfaz gráfica
- 8. Tarea:

Objetivos:

- Examine la naturaleza multitarea de Linux
- Administre procesos con la línea de comando y con la interfaz gráfica.

Realizar las capturas de pantalla de cada comando / aplicación que se ejecute y realiza un **documento pdf** con todas las indicaciones necesarias.

1. INTRODUCCIÓN

Como cualquier sistema operativo multitarea, Linux ejecuta múltiples procesos simultáneos. Bueno, parecen simultáneos, de todos modos. En realidad, una computadora con un solo procesador solo puede ejecutar un proceso a la vez, pero el kernel de Linux se las arregla para darle a cada proceso su turno en el procesador y cada uno parece estar ejecutándose al mismo tiempo.

Hay varios comandos que se pueden utilizar para controlar procesos. Son:

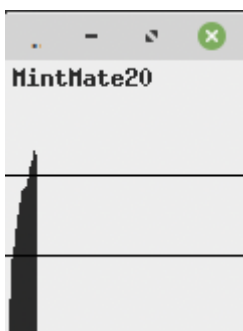
- `ps` lista los procesos que se están ejecutando actualmente en el sistema.
- `kill` enviar una señal a uno o más procesos (generalmente para "matar" un proceso)
- `jobs` una forma alternativa de enumerar tus procesos
- `bg` poner un proceso en segundo plano
- `fg` poner un proceso en primer plano

2. LANZAR UN PROGRAMA DESDE LA LÍNEA DE COMANDO

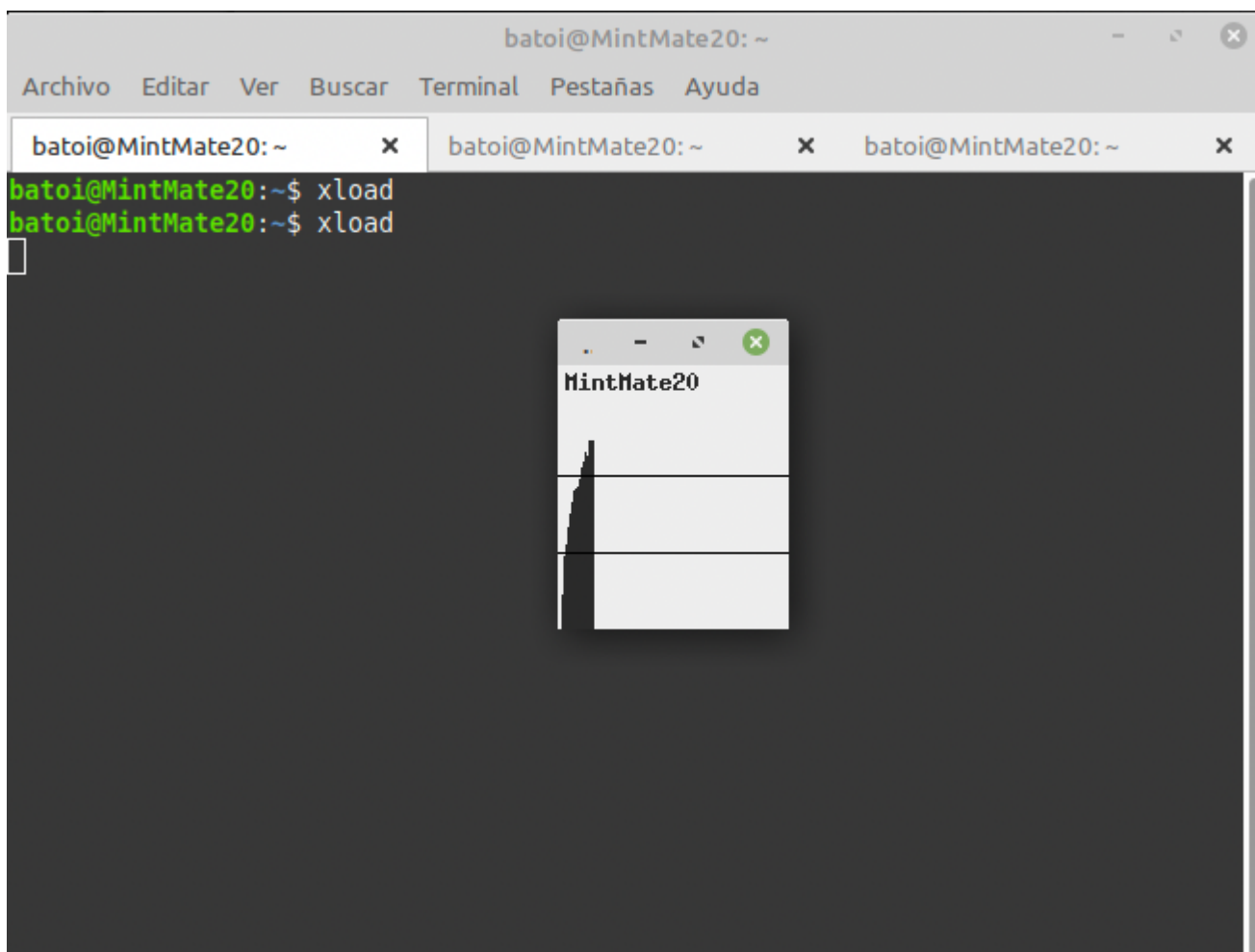
La mayoría de los programas gráficos se pueden iniciar desde la línea de comandos. Aquí hay un ejemplo: hay un pequeño programa suministrado con el sistema *X Windows* llamado **xload**, que muestra un gráfico que representa la carga del sistema.

Puede ejecutar este programa escribiendo el siguiente comando:

```
xload
```



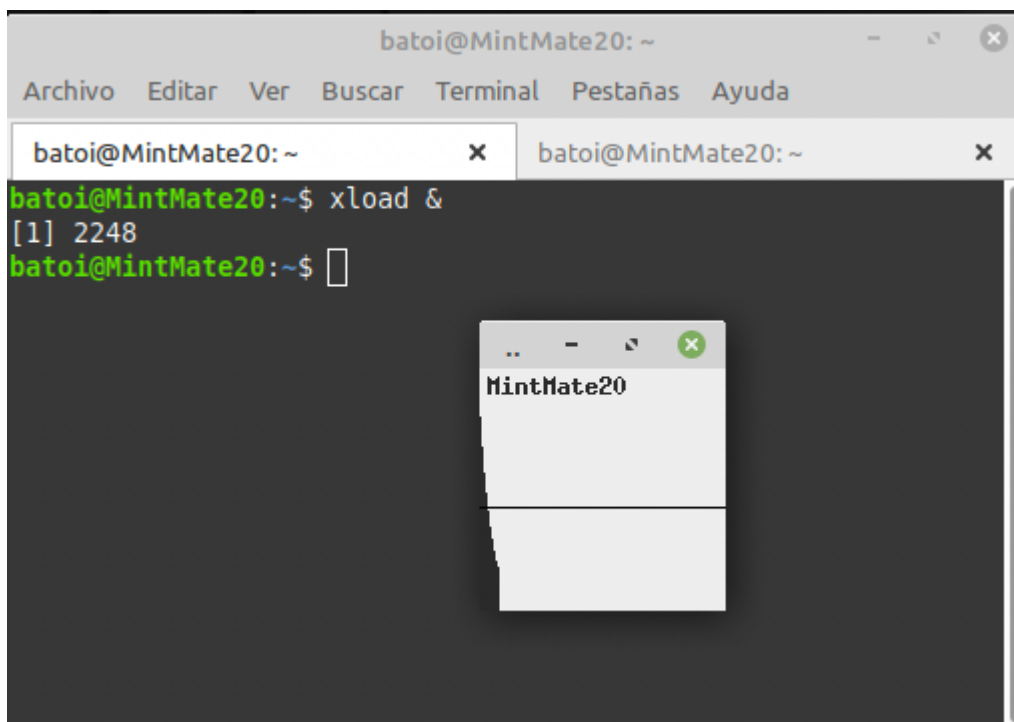
- Observa que aparece una ventana y comienza a mostrar el gráfico de carga del sistema. Tenga en cuenta también que el mensaje de prompt no vuelve a aparecer después de que el programa se inició (está bloqueado).
- El shell está esperando a que finalice el programa antes de que el control vuelva a nosotros. Si se cierra la ventana de **xload**, el programa **xload** termina y el prompt regresa.



3. PONER UN PROGRAMA EN 2º PLANO

Ahora, para hacer la vida un poco más fácil, vamos a lanzar de nuevo el programa xload, pero esta vez lo pondremos en segundo plano para que vuelva el prompt. Para hacer esto, ejecutamos xload así:

```
xload &
```




A terminal window titled 'batoi@MintMate20: ~' with a menu bar (Archivo, Editar, Ver, Buscar, Terminal, Pestañas, Ayuda). The terminal shows the command 'xload &' being executed, followed by the output '[1] 2248'. The prompt 'batoi@MintMate20:~\$' is visible. A small window titled 'MintMate20' is also visible in the background.

En este caso, el indicador regresó porque el proceso **xload** se puso en segundo plano.

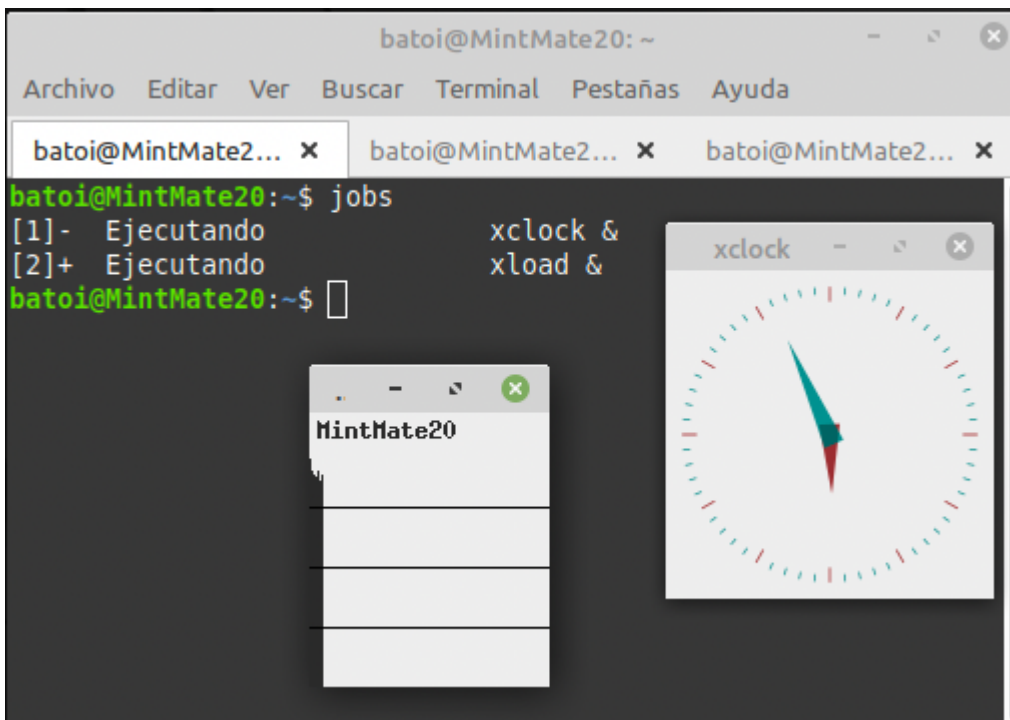
- Pero, imagina que olvidaste usar el símbolo **&** para poner el programa en segundo plano. Todavía se podría arreglar. Puedes escribir **control + x** y el proceso se suspenderá.
- El proceso todavía existe, pero está inactivo. (**control + c** finaliza el programa) Para reanudar el proceso en segundo plano, escriba el comando **bg** (abreviatura de background).

Ejemplo:

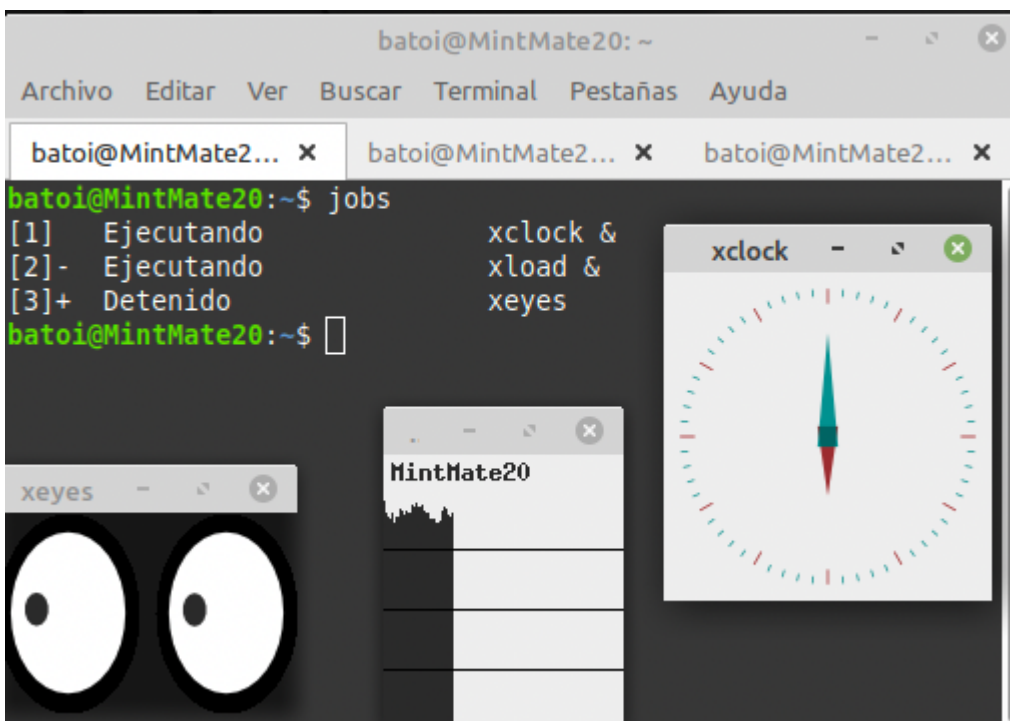


A terminal window titled 'batoi@MintMate20: ~' with a menu bar (Archivo, Editar, Ver, Buscar, Terminal, Pestañas, Ayuda). The terminal shows the command 'xload' being executed, followed by '^Z' (Ctrl-Z) and the output '[1]+ Detenido xload'. The prompt 'batoi@MintMate20:~\$' is visible. The command 'bg' is then entered, followed by the output '[1]+ xload &'. The prompt 'batoi@MintMate20:~\$' is visible. The command 'fg 1' is then entered, followed by the output 'xload'. The prompt 'batoi@MintMate20:~\$' is visible. A small window titled 'MintMate20' is also visible in the background.

- Como puedes ver arriba, el comando `fg` vuelve a poner el proceso más reciente en primer plano. También podemos especificar el número del proceso.
- El comando `jobs` enumera los procesos actuales:



Ahora inicia dos programas nuevos: `xclock` y `xeyes`. Suspended y reanude y practique los comandos `bf`, `fg` y `jobs`.



4. LISTADO DE PROCESOS

Ahora que tenemos un proceso en segundo plano, sería útil mostrar una lista de los procesos que hemos iniciado. Para hacer esto, podemos usar el comando `jobs` o el comando `ps` (estado del proceso) más potente:

```
batoi@MintMate20:~$ ps
  PID TTY          TIME CMD
 2360 pts/0    00:00:00 bash
 2523 pts/0    00:00:00 xclock
 2524 pts/0    00:00:00 xload
 2542 pts/0    00:00:00 xeyes
 2548 pts/0    00:00:00 ps
```

Uno de los indicadores más utilizados para `ps` es la opción `-f` (f significa completo), que proporciona más información, como se muestra en el siguiente ejemplo:

```
UID          PID    PPID  C STIME TTY          TIME CMD
batoi      2360    1924  0 17:47 pts/0    00:00:00 bash
batoi      2523    2360  0 17:54 pts/0    00:00:00 xclock
batoi      2524    2360  0 17:54 pts/0    00:00:00 xload
batoi      2542    2360  0 17:59 pts/0    00:00:00 xeyes
batoi      2571    2360  0 18:05 pts/0    00:00:00 ps -f
```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
batoi	2360	1924	0	17:47	pts/0	00:00:00	bash
batoi	2523	2360	0	17:54	pts/0	00:00:00	xclock
batoi	2524	2360	0	17:54	pts/0	00:00:00	xload
batoi	2542	2360	0	17:59	pts/0	00:00:00	xeyes
batoi	2571	2360	0	18:05	pts/0	00:00:00	ps -f

Columna	Descripción
UID	ID de usuario al que pertenece este proceso (la persona que lo ejecuta).
PID	Identificación de proceso.
PPID	ID del proceso padre (el ID del proceso que lo inició).
C	Utilización de CPU del proceso.
STIME	Hora de inicio del proceso.
TTY	Tipo de terminal asociado al proceso
TIME	Tiempo de CPU que toma el proceso.
CMD	El comando que inició este proceso.

Podemos listar todos los procesos si ejecutamos el comando `ps` con los parámetros `aux`:

```

batoi@MintMate20:~$ ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.5 102056 11400 ?        Ss   17:04   0:00 /sbin/init splash
root         2  0.0  0.0      0     0 ?        S    17:04   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        I<   17:04   0:00 [rcu_gp]
root         4  0.0  0.0      0     0 ?        I<   17:04   0:00 [rcu_par_gp]
root         6  0.0  0.0      0     0 ?        I<   17:04   0:00 [kworker/0:0H-kblockd]
root         7  0.0  0.0      0     0 ?        I    17:04   0:00 [kworker/0:1-events]
root         9  0.0  0.0      0     0 ?        I<   17:04   0:00 [mm_percpu_wq]
root        10  0.0  0.0      0     0 ?        S    17:04   0:00 [ksoftirqd/0]
root        11  0.0  0.0      0     0 ?        I    17:04   0:00 [rcu_sched]
root        12  0.0  0.0      0     0 ?        S    17:04   0:00 [migration/0]
root        13  0.0  0.0      0     0 ?        S    17:04   0:00 [idle_inject/0]
root        14  0.0  0.0      0     0 ?        S    17:04   0:00 [cpuhp/0]
root        15  0.0  0.0      0     0 ?        S    17:04   0:00 [kdevtmpfs]

```

El significado de los comandos es el siguiente:

- **a** muestra todos los procesos **tty** conocidas
- **x** muestra los **tty** desconocidos
- **u** muestra de qué usuario es cada proceso

Para saber más: ¿Qué es un TTY?

Columna	Descripción
USER	ID de usuario al que pertenece este proceso (la persona que lo ejecuta).
PID	Identificación de proceso.
%UPC	% De CPU consumido por el proceso.
%MEM	% MEM tomado por el proceso.
VSZ	Uso de memoria virtual
RSS	Memoria tomada en Kbytes
TTY	Tipo de terminal asociado al proceso
STAT	Estado del proceso: S leeping, R unning, T erminated, Z ombie.
START	Hora de inicio del proceso.
TIME	Tiempo de CPU que toma el proceso.
COMMAND	El comando que inició este proceso.

5. MATANDO PROCESOS

Imagina un proceso que deja de responder, ¿cómo podemos desahacernos de él? Utilizando el comando **kill**, por supuesto.

Primero, debes identificar el proceso que desea eliminar. Puede utilizar **jobs** o **ps**. Si utilizas **jobs**, obtendrás un número de proceso. Con **ps**, se le da un ID de proceso (PID). También podemos usar **killall** para matar el proceso usando su nombre.

Hagámoslo de ambas formas:

5.1. Utilizando `jobs`

```
[me@linuxbox me]$ xload

[1] 1292

[me@linuxbox me]$ jobs

[1]+  Running xload &

[me@linuxbox me]$ kill %1

[1] Terminated xload
```

Utilizando `PID`:

```
[me@linuxbox me]$ ps

PID TTY TIME CMD
1280 pts/5 00:00:00 bash
1293 pts/5 00:00:00 xload
1294 pts/5 00:00:00 ps

[me@linuxbox me]$ kill 1293
[2]+  Terminated xload
```

5.2. Un poc más acerca de `kill`

Si bien el comando `kill` se usa para "matar" procesos, su propósito real es enviar señales a los procesos. La mayoría de las veces, la señal está destinada a indicarle al proceso que se vaya, pero hay más que eso. Los programas (si están escritos correctamente) escuchan las señales del sistema operativo y responden a ellas, la mayoría de las veces para permitir algún método elegante de terminación. Por ejemplo, un editor de texto puede escuchar cualquier señal que indique que el usuario está cerrando la sesión o que la computadora se está apagando. Cuando recibe esta señal, guarda el trabajo en curso antes de salir. El comando `kill` puede enviar una variedad de señales a los procesos. Si escribe `kill -l` se obtendrá una lista de las señales que admite el comando.

La mayoría son bastante oscuras (llevan desde los inicios de UNIX), pero es útil conocer varias:

Señal	Nombre	Descripción
1	SIGHUP	Colgar. Los programas pueden escuchar esta señal y actuar (o no actuar) sobre ella.

Señal	Nombre	Descripción
2	SIGINT	Interrumpir. Esta señal se da a los procesos para interrumpirlos. Los programas pueden procesar esta señal y actuar en consecuencia. También puedes emitir esta señal directamente escribiendo <code>control+c</code> en la ventana de terminal donde el programa se está ejecutando.
15	SIGTERM	Terminación. Esta es la señal predeterminada enviada por el comando <code>kill</code> si no se especifica ninguna señal. Esta señal se da a los procesos para terminarlos. Nuevamente, los programas pueden procesar esta señal y actuar sobre ella. También puede emitir esta señal directamente escribiendo <code>control+c</code> en la ventana de terminal donde se encuentra el programa corriendo.
9	SIGKILL	Matar. Esta señal provoca la terminación inmediata del proceso por parte del kernel de Linux. Los programas no pueden escuchar esta señal.

Ahora supongamos que tiene un programa que está bloqueado irremediablemente y ya no responde y quiere deshacerse de él. Esto es lo que haces:

1. Utilizar el comando `ps` para obtener el ID de proceso (PID) del proceso que desea finalizar.
2. Emitir un comando de interrupción para ese PID.
3. Si el proceso se niega a terminar (es decir, ignora la señal), envíe señales cada vez más duras hasta que termine.

```
[me@linuxbox me]$ ps x
PID TTY STAT TIME COMMAND
2931 pts/5 S  0:00 xclock

[me@linuxbox me]$ kill -SIGTERM 2931
```

En el ejemplo anterior se ha utilizado el comando `kill` de forma formal. En la práctica, es más común hacerlo de la siguiente manera, ya que la señal predeterminada enviada por `kill` es `SIGTERM` y `kill` también puede usar el número de señal en lugar del nombre de la señal:

```
[ me @ linuxbox me] $ kill 2931
```

Luego, si el proceso no termina, forzarlo con la señal `SIGKILL`:

```
[ me @ linuxbox me] $ kill -9 2931
```

6. OTROS COMANDOS ÚTILES

6.1. top

Este comando proporciona una vista continua de la actividad del procesador en tiempo real. Muestra una lista de las tareas más intensivas de la CPU en el sistema, la memoria y el uso de la CPU, etc.

```
top - 17:07:00 up 2:38, 1 user, load average: 0.45, 0.70, 0.62
Tasks: 104 total, 1 running, 102 sleeping, 0 stopped, 1 zombie
Cpu(s): 23.8% us, 3.3% sy, 0.0% ni, 72.8% id, 0.0% wa, 0.0% hi, 0.0% si
Mem: 768600k total, 691768k used, 76832k free, 37960k buffers
Swap: 979956k total, 0k used, 979956k free, 368184k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
4421	tv	15	0	148m	81m	18m	S	9.9	10.8	13:56.36	firefox-bin
4090	root	6	-10	173m	41m	3036	S	6.9	5.5	7:32.76	XFree86
6109	tv	15	0	29604	14m	12m	S	4.3	2.0	0:01.32	ksnapshot
4339	tv	15	0	31440	14m	11m	S	2.3	1.9	0:20.42	kicker
4335	tv	15	0	27736	12m	9.8m	S	2.0	1.6	0:13.47	kwin
4424	tv	15	0	31380	14m	11m	S	0.7	2.0	0:17.15	konsole
6090	root	16	0	2272	1152	872	R	0.3	0.1	0:00.25	top
1	root	16	0	1940	664	568	S	0.0	0.1	0:00.25	init
2	root	34	19	0	0	0	S	0.0	0.0	0:00.00	ksoftirqd/0
3	root	10	-5	0	0	0	S	0.0	0.0	0:00.35	events/0
4	root	11	-5	0	0	0	S	0.0	0.0	0:00.01	khelper

Una buena alternativa a `top` es `htop`, una evolución con características realmente sorprendentes.

The screenshot shows the `htop` interface in a terminal window. At the top, system statistics are displayed: CPU usage at 0.7%, memory usage at 541M/1.94G, swap usage at 1.51M/1.83G, 92 tasks (183 threads, 1 running), load average of 0.02, and uptime of 02:12:54. Below this is a table of running processes. The table has columns for PID, USER, PRI, NI, VIRT, RES, SHR, S, CPU%, MEM%, TIME+, and Command. Processes include `/usr/libexec/xdg-permiss`, `/usr/libexec/xdg-document-p`, `/usr/libexec/xdg-document`, `/usr/libexec/xdg-desktop-pd`, `sh -c /usr/lib/x86_64-li`, `/usr/lib/x86_64-linux`, `/usr/lib/x86_64-li`, `/usr/lib/x86_64-li`, `/usr/libexec/xdg-desktop`, `mintmenu`, and `mintmenu`. At the bottom, there is a status bar with keyboard shortcuts: F1 Help, F2 Setup, F3 Search, F4 Filter, F5 Sorted, F6 Collap, F7 Nice, F8 Nice +, F9 Kill, and F10 Quit.

Si el comando `htop` no se encuentra instalado puedes hacerlos mediante el comando `apt install htop` en LinuxMint o con `brew install htop` en macOS con el gestor de paquetes `brew` instalado.

6.2. nice

Si deseas (o necesitas) ejecutar un comando que normalmente utiliza una gran cantidad de CPU (por ejemplo, ejecutar `tar` en un archivo grande), probablemente no quieras ralentizar todo el sistema con él.

Los sistemas Linux proporcionan el comando `nice` para controlar la prioridad con la que un proceso se ejecutará, o `renice` para cambiar la prioridad de un proceso que ya se está ejecutando.

El comando es muy fácil de usar:

```
nice -n nivel_de_prioridad comando_a_ejecutar
```

El nivel de prioridad va de **-20** (máxima prioridad) a **20** (menor). Por ejemplo, para ejecutar `tar` y `gzip` en el nivel de prioridad más bajo:

```
nice -n 20 tar -czvf file.tar.gz bigfiletocompress
```

De manera similar, si tiene un proceso en ejecución, use `ps` para encontrar el ID del proceso y luego use `renice` para cambiar su nivel de prioridad:

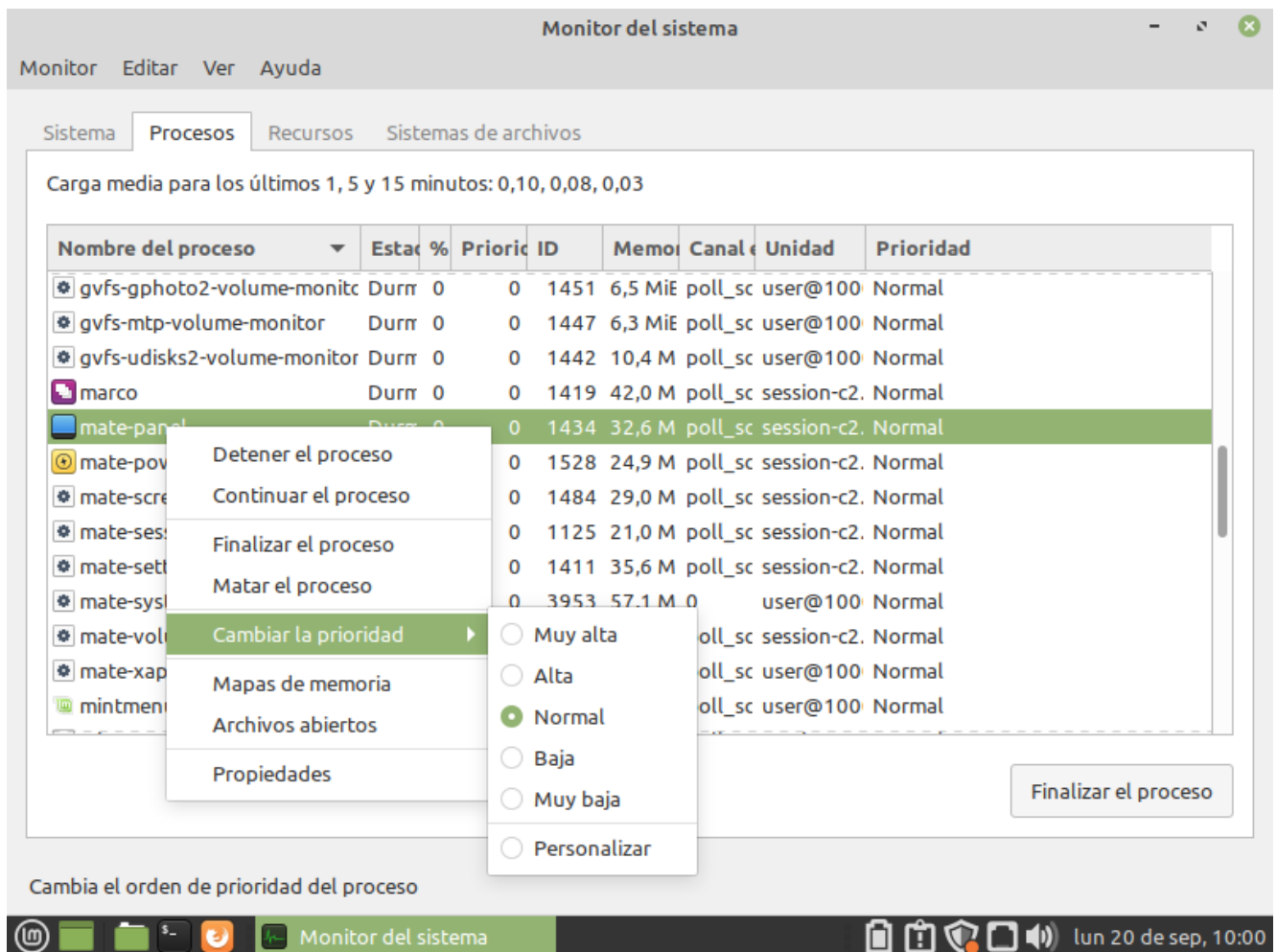
```
$ renice -n 19 -p 987 32
```

Esto cambiará la prioridad de los procesos 987 y 32 a 19.

7. ADMINISTRANDO PROCESOS UTILIZANDO LA INTERFAZ GRÁFICA

La aplicación **Gnome System Monitor** permite a los usuarios mostrar información básica del sistema y monitorizar los procesos del sistema, el uso de los recursos y el sistema de archivos.

La aplicación muestra los procesos activos y cómo se relacionan entre sí. También proporciona información detallada sobre procesos individuales.



8. TAREA:

- ☐ Practica todos los comandos explicados
- ☐ Presta atención a los cambios de estado, prioridades, etc.
- ☐ Investiga que hace el comando `xkill`. Pruébalo y proporciona ejemplos de uso.
- ☐ ¿Qué hace el comando `killall`?
- ☐ Busca qué utilidades (de **línea de comandos** y de **interfaz gráfica**) están disponibles en Windows/Mac para la administración de procesos (list,kill,etc.) Prueba las opciones más importantes disponibles.