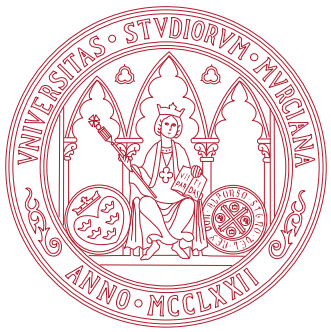

Universidad de Murcia

Facultad de Informática

Departamento de Ingeniería y Tecnología de Computadores

Área de Arquitectura y Tecnología de Computadores

Prácticas de

Introducción a los Sistemas Operativos

2º de Grado en Ingeniería Informática

Boletín de prácticas 9 – Gestión de la E/S

Curso 2021/2022

Índice

1. Introducción

El diseño e implementación del subsistema de E/S es otro aspecto importante de un sistema operativo, ya que capacita a éste para manejar una gran variedad de dispositivos de E/S, tanto presentes como futuros. También determina los procesos necesarios para instalar, configurar y usar un dispositivo, lo que a su vez condiciona el grado de satisfacción de los usuarios durante su trabajo con el sistema.

La administración de los dispositivos de E/S es, por su naturaleza, una tarea compleja. Comprenderla totalmente requeriría una gran cantidad de tiempo. Por eso, en este boletín nos vamos a centrar sólo en algunos de los aspectos básicos de la administración de la E/S en Linux, pero que tienen también un impacto más directo sobre el usuario final.

2. Objetivos

Al terminar este boletín, el alumno debe ser capaz de:

- Describir qué son los ficheros especiales de dispositivo, distinguir entre ficheros especiales de bloques y de caracteres, y explicar el papel que juegan los números mayor y menor de dispositivo.
- Definir qué es un módulo del núcleo de Linux y cuándo surgen dependencias entre ellos.
- Listar, cargar y descargar módulos.
- Explicar qué son los sistemas de ficheros *proc* y *sysfs* y para qué se utilizan.
- Describir el papel de `udev` en la gestión dinámica de los dispositivos de E/S.
- Enumerar los planificadores del núcleo de Linux, y explicar su funcionamiento básico y situaciones en las que se suele usar cada uno.
- Cambiar de planificador de disco.
- Ejecutar procesos con una determinada prioridad de E/S y cambiar dicha prioridad para los procesos ya en ejecución.
- Identificar los procesos en ejecución limitados por la E/S.

3. Ficheros y órdenes utilizados

En este boletín, nos centraremos en los siguientes ficheros y directorios:

- `/dev`
- `/etc/modprobe.d`
- `/etc/udev/rules.d/`
- `/lib/modules/<versión_kernel>`
- `/lib/modules/<versión_kernel>/modules.dep`
- `/lib/udev/rules.d/`
- `/proc`
- `/proc/interrupts`
- `/proc/iomem`
- `/proc/ioports`
- `/proc/modules`
- `/sys`

También veremos, entre otras, las siguientes órdenes:

- `mknod`
- `lsmod`, `modinfo`, `insmod`, `rmmod`, `modprobe`, `depmod`
- `lspci`, `lsusb`, `lshw`
- `ionice`
- `udevd`
- `iotop`

4. Ficheros especiales de dispositivo

Además de los ficheros regulares, existen otros *ficheros especiales* que se utilizan para referenciar a diferentes elementos del sistema. Entre estos ficheros encontramos los *ficheros especiales de dispositivo* que, como su nombre indica, representan a dispositivos de E/S. Dentro de estos ficheros tenemos los ficheros especiales de bloques y los ficheros especiales de caracteres. Todos estos ficheros se encuentran en el directorio `/dev` o en alguno de sus subdirectorios.

Los *ficheros especiales de bloques* sirven para dar nombre a los dispositivos de bloques, como los discos duros, las unidades de CD/DVD, las memorias flash, etc. Estos ficheros se distinguen del resto porque sus permisos comienzan por la letra «b». A continuación, mostramos algunos ficheros de este tipo que podemos encontrar habitualmente en `/dev`:

```
brw-rw----. 1 root disk        7,   0 abr 19 19:44 loop0
brw-rw----. 1 root disk        7,   1 abr 19 19:44 loop1
brw-rw----. 1 root disk        7,   2 abr 19 19:44 loop2
brw-rw----. 1 root disk        7,   3 abr 19 19:44 loop3
brw-rw----. 1 root disk        7,   4 abr 19 19:44 loop4
brw-rw----. 1 root disk        7,   5 abr 19 19:44 loop5
brw-rw----. 1 root disk        7,   6 abr 19 19:44 loop6
brw-rw----. 1 root disk        7,   7 abr 19 19:44 loop7
brw-rw----. 1 root disk        8,   0 abr 19 19:44 sda
brw-rw----. 1 root disk        8,   1 abr 19 19:44 sda1
brw-rw----. 1 root disk        8,   2 abr 19 19:44 sda2
brw-rw----. 1 root disk        8,   3 abr 19 19:44 sda3
brw-rw----. 1 root disk        8,   4 abr 19 19:44 sda4
brw-rw----+ 1 root cdrom      11,   0 abr 19 19:44 sr0
```

En este ejemplo podemos encontrar ficheros especiales de bloques para los dispositivos *loop*, un disco duro (`sda`) y sus particiones (`sda1`, `sda2`, `sda3` y `sda4`), y una unidad de DVD (`sr0`).

Por otro lado, los *ficheros especiales de caracteres* representan a dispositivos de caracteres (puertos serie y paralelo, terminales de texto, el *frame buffer* de la tarjeta de vídeo, ficheros de control de ciertos dispositivos, etc.). Los permisos de estos ficheros comienzan por la letra «c». A continuación, mostramos algunos ficheros de este tipo:

```
crw-rw----. 1 root video      29,   0 abr 19 19:44 fb0
crw-rw----. 1 root lp          6,   0 abr 19 19:44 lp0
crw-rw-rw-. 1 root root        1,   3 abr 19 19:44 null
crw-rw-rw-. 1 root root        1,   8 abr 19 19:44 random
crw-rw-rw-. 1 root tty         5,   0 may  1 19:20 tty
crw--w----. 1 root tty         4,   0 abr 19 19:44 tty0
crw--w----. 1 root tty         4,   1 abr 19 19:44 tty1
crw-rw----. 1 root tty         4,   2 abr 19 19:44 tty2
crw-rw----. 1 root tty         4,   3 abr 19 19:44 tty3
crw-rw-rw-. 1 root root        1,   5 abr 19 19:44 zero
```

Entre los ficheros mostrados, encontramos el del *frame buffer* (`fb0`), un puerto paralelo (`lp0`), terminales (ficheros `tty*`), el «sumidero» de capacidad infinita (`null`), la «fuente» inagotable de ceros (`zero`) y un fichero para generar bytes aleatorios (`random`).

Podemos ver que estos ficheros no tienen un tamaño asociado, sino un par de números que son utilizados internamente por el núcleo de Linux para referirse a los dispositivos que representan. El primero de estos números es el *número mayor* (*major number*) que identifica con qué *driver* está asociado el fichero (es decir, qué *driver* maneja el dispositivo). El segundo número es el *número menor* (*minor number*) que identifica al dispositivo concreto asociado al fichero (por eso a este número también se le llama *número de unidad* o *unit number*). En general, dispositivos iguales o similares están representados por ficheros que tienen el mismo número mayor (ya que el mismo *driver* suele gestionar todos esos dispositivos), pero un número menor distinto para cada uno (ya que cada fichero corresponde a un dispositivo diferente).

A la hora de referirse a un dispositivo, lo importante son su número mayor y su número menor. El fichero especial de dispositivo es sólo una forma sencilla y cómoda de referirse a un dispositivo, sin tener que recordar sus números mayor y menor. Así, si ejecutamos `mknod primerdiscoduro b 8 0`, se creará un fichero especial de dispositivo a través del cual podremos hacer referencia al primer disco duro (al igual que con `/dev/sda`). Si además ajustamos los permisos de acuerdo a los de `/dev/sda`, podremos usar de forma indistinta uno u otro fichero:

```
brw-rw----. 1 root disk        8,   0 abr 19 19:44 sda
brw-rw----. 1 root disk        8,   0 abr 19 19:44 primerdiscoduro
```

Algunos de estos ficheros son creados a mano con la herramienta `mknod` durante el arranque del ordenador (bien cuando se ejecuta el script *init* del disco RAM del sistema operativo, bien en alguno de los pasos posteriores). Otros ficheros, en cambio, los crea automáticamente la herramienta `udev` que, además, crea y borra estos ficheros dinámicamente cuando los dispositivos correspondientes se conectan y desconectan.

Debemos tener en cuenta que un fichero especial de dispositivo representa a un dispositivo de una manera genérica, es decir, sin tener en cuenta ni el fabricante, ni el modelo, ni nada similar. Así, `/dev/sda` representa al primer disco duro del ordenador. Si cambiamos el disco duro por otro distinto, `/dev/sda` representará al nuevo disco duro. Del mismo modo, `/dev/sr0` representa a la primera unidad de CD/DVD, independientemente del modelo concreto y de qué CD o DVD haya insertado (si es que hay alguno).

Como ya hemos visto, estos ficheros especiales no dejan de ser ficheros en el sentido de que podemos abrirlos para leer de y escribir en ellos. La diferencia está en que realmente estamos leyendo o escribiendo a bajo nivel sobre el dispositivo que representan. Al ser ficheros, los permisos, propietarios y grupos asociados a los mismos también son importantes, pues determinan quién puede acceder directamente a los dispositivos y qué operaciones se pueden hacer con ellos. Así, por ejemplo, si el fichero especial de dispositivo `/dev/sda1` pertenece al `root` y al grupo `disk`, con permisos `rw-rw----`, entonces sólo el superusuario, o un usuario perteneciente al grupo `disk`, podrá dar formato al dispositivo, ya que `mkfs` usa este fichero especial de bloques para acceder a él.

5. Módulos del núcleo

El núcleo de Linux sigue una arquitectura monolítica, si bien el diseño es modular. Esto significa que, aunque el núcleo puede ser visto como un único programa en ejecución, hay porciones de código (los *módulos*) que se pueden cargar/descargar en caliente para añadir/eliminar código al/del núcleo de forma dinámica, sin reiniciar el sistema, y, por tanto, para añadir/eliminar soporte para ciertas características.

Cuando un módulo se carga, su código pasa a ser parte del propio núcleo y se ejecuta en el modo núcleo (o supervisor) del procesador, con acceso sin restricciones a todas las funciones del núcleo, a todas las funciones exportadas por módulos previamente insertados y a todo el hardware de la máquina. En este sentido, los módulos son como cualquier otro trozo de código del sistema operativo. Sin embargo, los módulos proporcionan varias ventajas respecto a un núcleo basado en un único

ejecutable: facilitan el diseño y la implementación del sistema operativo, permiten crear núcleos más pequeños y flexibles que se pueden adaptar al hardware subyacente, se pueden descargar cuando ya no son útiles, liberando así recursos, etc.

Entre las características que Linux proporciona mediante módulos está el soporte para distintos sistemas de ficheros (Ext3, Ext4, XFS, ...), periféricos (discos duros, tarjetas de red, tarjetas de sonido, ...) y protocolos de red (IPv4, IPv6, ...). Por lo tanto, podemos considerar a los módulos como los *drivers* para Linux.

Los módulos se almacenan en ficheros objeto con extensión `.ko.xz` bajo el directorio `/lib/modules/<versión_kernel>` (por ejemplo, `/lib/modules/5.8.8-200.fc32.x86_64`), por lo que cada versión del núcleo tiene sus propios módulos[1]. Dentro de este directorio, la mayor parte de los módulos se encuentran en el subdirectorio `kernel` que, a su vez, está organizado en subdirectorios. Algunos de los subdirectorios dentro de `kernel` son:

- `drivers`: contiene módulos para la gestión de los dispositivos hardware.
- `fs`: contiene módulos que proporcionan soporte para distintos sistemas de ficheros.
- `net`: contiene módulos para el soporte de diferentes protocolos de red.
- `sound`: contiene módulos para la gestión de distintas tarjetas de sonido.

Aunque la mayor parte del código de Linux se encuentra en los módulos, hay código que, por su importancia, forma parte del núcleo de Linux de forma permanente, por ejemplo, el código para el acceso al bus PCI. También es posible incluir el código de un módulo de forma permanente en el núcleo de Linux, indicándolo así en la configuración del núcleo y recompilando el mismo.

5.1. Dependencias

Cuando implementamos un programa, usamos funciones (`printf`, `read`, ...) que ya existen y que se encuentran en bibliotecas ya instaladas en el sistema. Con los módulos pasa algo similar: cuando se implementa un módulo, se usan funciones que ya están definidas, en este caso, dentro del núcleo del sistema operativo.

Para que un módulo se pueda cargar, es necesario que el núcleo tenga exportadas todas las funciones externas que usa el módulo. Algunas de estas funciones pueden estar implementadas en otros módulos por lo que, para que sea posible la carga de nuestro módulo, es necesario que otros módulos se hayan cargado ya previamente. Esto hace que surjan *dependencias* entre los módulos.

Las dependencias entre los módulos de un núcleo determinado se encuentran en el fichero `/lib/modules/<versión_kernel>/modules.dep`. Básicamente, para cada módulo con dependencias, este fichero indica qué otros módulos tienen que estar cargados previamente para que el módulo también se pueda cargar.

5.2. Carga de los módulos

Los módulos se pueden cargar en distintos momentos:

- Al ejecutar el script `init` del disco RAM. El contenido de este disco RAM, usado durante el arranque del sistema de operativo, se obtiene de un fichero que, entre otras cosas, almacena algunos módulos que son necesarios para que el sistema operativo pueda acceder, al menos, al sistema de ficheros raíz. Entre estos módulos están los que dan soporte a los discos duros y a los sistemas de ficheros (Ext3, Ext4, etc.).
- Durante la ejecución de los distintos scripts y programas por parte del proceso `systemd`.
- Al conectar un nuevo dispositivo. En este caso, `udev` detectará el dispositivo y cargará el módulo correspondiente.
- A mano, en cualquier momento, por parte del administrador.

[1] Podemos ejecutar la orden `uname -r` para obtener la versión del núcleo de Linux en ejecución en un instante determinado.

5.3. Herramientas para la gestión de módulos

La utilidad que nos permite ver los módulos cargados es `lsmod`. Para cada módulo, esta herramienta muestra su nombre (columna `Module`), su tamaño (columna `Size`), un contador de usos y los módulos que lo usan (columna `Used by`[2]). `lsmod` obtiene toda esta información del fichero `/proc/modules`. A continuación, aparece una posible salida de esta orden:

```
# lsmod
Module                  Size  Used by
9p                     65536  1
fscache               389120  1 9p
fuse                  118784  2
uinput                 20480  1
snd_hda_codec_generic    86016  1
snd_hda_intel          45056  10
snd_hda_codec         151552  2 snd_hda_codec_generic,snd_hda_intel
snd_hda_core           94208  3 snd_hda_codec_generic,snd_hda_intel,snd_hda_codec
crct10dif_pclmul       16384  0
snd_hwdep              16384  1 snd_hda_codec
crc32_pclmul           16384  0
...                   .....  .. ...
```

Podemos ver, por ejemplo, que el módulo `fat` es usado por `vfat` y que éste último no está en uso en este momento.

La orden `modinfo` muestra información sobre un módulo del núcleo en concreto: fichero, autor, descripción, licencia, dependencias, parámetros (si los hay), etc. Ejemplo:

```
# modinfo vfat
filename:       /lib/modules/4.17.18-200.fc28.x86_64/kernel/fs/fat/vfat.ko.xz
author:         Gordon Chaffee
description:    VFAT filesystem support
license:        GPL
alias:          fs-vfat
depends:        fat
retpoline:      Y
intree:         Y
name:           vfat
vermagic:       4.17.18-200.fc28.x86_64 SMP mod_unload
sig_id:         PKCS#7
signer:
sig_key:
sig_hashalgo:   md4
signature:      30:82:02:CF:06:09:2A:86:48:86:F7:0D:01:07:02:A0:82:02:C0:30:
...
```

Para cargar módulos podemos usar `insmod`. Esta herramienta no resuelve dependencias, por lo que un módulo sólo se podrá cargar si ya están cargados los módulos de los que depende. Como parámetro, `insmod` recibe la ruta del fichero .ko.xz del módulo.

Un módulo se puede descargar con `rmmod`. Esta orden recibe como parámetro el nombre del módulo, tal cual nos lo muestra `lsmod`. Debemos tener en cuenta que un módulo no se puede descargar si está siendo usado. Al igual que `insmod`, `rmmod` no tiene en cuenta las dependencias entre módulos en el sentido de que, si un módulo se descarga, y otros de los que depende ya no son necesarios, `rmmod` no descargará esos otros módulos, que habrá que descargar uno a uno.

Si deseamos que se tengan en cuenta las dependencias, debemos usar `modprobe`. Esta orden carga un módulo verificando sus dependencias, cargando previamente, en caso necesario, los módulos de los

[2]El número mostrado en la columna `Used by` se refiere al número de veces que cualquier código del núcleo (módulo o no) ha tomado una referencia al módulo en cuestión. Para drivers y módulos de sistemas de ficheros esto ocurre cuando un fichero/dispositivo está abierto.

que depende. A diferencia de `insmod`, sólo es necesario indicar el nombre del módulo a cargar y no la ruta de su fichero .ko.

También podemos usar `modprobe` para descargar un módulo, indicando el módulo a descargar junto con la opción `-r`. Ya que `modprobe` tiene en cuenta las dependencias, si el módulo descargado dependía de otros que ya no se necesitan, también se descargarán.

Otra opción útil de `modprobe` es `--show-depends` que, dado un módulo, muestra todos aquellos módulos de los que depende.

Podemos cambiar el comportamiento de `modprobe` creando un fichero de configuración en `/etc/modprobe.d`. El fichero puede tener cualquier nombre, pero su extensión debe ser `.conf`. Entre las líneas que podemos encontrar en un fichero de este tipo tenemos:

- `alias <nombre_alias><nombre_módulo>`: permite asignar un alias (es decir, un nombre alternativo) a un módulo.
- `options <nombre_módulo><opciones ...>`: permite configurar las opciones de un módulo (si las tiene). Estas opciones se usan cuando se carga el módulo.
- `install <nombre_módulo><orden_a_ejecutar>`: ejecuta la orden indicada en lugar de insertar el módulo, como se haría normalmente.
- `remove <nombre_módulo><orden_a_ejecutar>`: es similar a la línea anterior, pero cuando se elimina un módulo.
- `blacklist <nombre_módulo>`: impide que se ponga en marcha un módulo de forma automática. El módulo podría ser cargado sin embargo si algún otro que depende de él y que no aparece en `blacklist` se carga. También podría cargarse manualmente. Para forzar que un módulo no pueda ser cargado de ningún modo habría que hacer uso de `install`, indicando como orden a ejecutar `/bin/false`. Observa que en este caso se estaría impidiendo la carga del módulo en cuestión y de cualquier otro que dependiese de él.

A continuación, mostramos algunas de las líneas que pueden aparecer en los ficheros de configuración de `modprobe`:

```
install snd-pcm /sbin/modprobe --ignore-install snd-pcm && /sbin/modprobe snd-seq
blacklist radeonfb
blacklist nvidiafb
blacklist snd-pcsp
alias sound-service-*-0 snd-mixer-oss
alias sound-service-*-1 snd-seq-oss
alias sound-service-*-3 snd-pcm-oss
alias sound-service-*-8 snd-seq-oss
alias sound-service-*-12 snd-pcm-oss
alias block-major-8-* sd_mod
alias block-major-9-* md
```

Como hemos dicho, la dependencias entre módulos se guardan en el fichero `/lib/modules/<versión_kernel>/modules.dep`. Este fichero se puede generar o actualizar (por ejemplo, si se instalan nuevos módulos) ejecutando `depmod -a`.

6. Los sistemas de ficheros *proc* y *sysfs*

Los directorios `/proc` y `/sys` son dos puntos de montaje en los que solemos encontrar montados, respectivamente, los sistemas de ficheros virtuales *proc* y *sysfs*. Son virtuales en el sentido de que no se almacenan en ningún disco ni dispositivo similar. Su información sólo se almacena en RAM y existen mientras el núcleo de Linux esté en ejecución.

Estos dos sistemas de ficheros proporcionan una interfaz sencilla y conocida (la de los directorios y ficheros) para leer y modificar información almacenada dentro del núcleo de Linux. En lo que

a la E/S se refiere, *proc* y *sysfs* nos permiten acceder a la información que el núcleo tiene sobre los dispositivos hardware disponibles, buses, módulos, etc. También son utilizados por ciertas herramientas para la gestión del hardware. En concreto, `udev` usa `sysfs` para obtener información sobre los dispositivos que se conectan, y así poder cargar los módulos y crear los ficheros especiales de dispositivo correspondientes. En la sección 8, también veremos que podemos cambiar el planificador de un disco a través de este sistema de ficheros.

En el caso de `/proc`, podemos encontrar muchos ficheros y directorios con información interesante sobre el sistema: hardware, procesos en ejecución, configuración del núcleo (memoria virtual, protocolos de red, ...), etc. Entre los ficheros relacionados con la E/S, tenemos:

- `devices`: dispositivos de bloques y de caracteres actualmente configurados (sólo incluye los dispositivos cuyos módulos están cargados).
- `interrupts`: interrupciones tratadas y dispositivos asociados.
- `iomem`: rangos de memoria usados por los dispositivos para la E/S mapeada a memoria.
- `ioports`: puertos registrados para el acceso a los dispositivos mediante instrucciones específicas de E/S (es decir, para E/S no mapeada a memoria).
- `modules`: listado de todos los módulos cargados actualmente.

Debemos tener en cuenta que, aunque `/proc` contiene ciertos datos sobre el hardware, la información más completa y detallada se encuentra en `/sys`.

Podemos recorrer los árboles de directorios en `/proc` y `/sys`, y leer el contenido de los ficheros, con las herramientas habituales para ficheros (`cd`, `ls`, `cat`, ...). No obstante, hay algunas órdenes que nos facilitan la tarea y nos muestran información sobre el hardware de una manera sencilla.

La orden más importante para realizar esto es `lshw`, que nos da información detallada sobre la configuración hardware de la máquina. Más concretamente a través de la ejecución de esta orden podemos obtener, entre otra información, el modelo de CPU que hay instalado junto con sus características, la configuración de las caches, la configuración exacta de memoria, buses PCI y USB...

A través de otras órdenes, sin embargo, podemos obtener información acerca de aspectos concretos del hardware. Entre estas órdenes tenemos `lsusb`, que nos muestra información sobre todos los buses USB del sistema y todos los dispositivos conectados a ellos. Ejemplo:

```
# lsusb
Bus 002 Device 002: ID 046d:c016 Logitech, Inc.
Bus 001 Device 001: ID 0000:0000
...
```

Esta orden admite las opciones `-v`, para imprimir información detallada de los dispositivos, y `-d <vendor>:<product>`, para obtener información sólo sobre el dispositivo indicado, donde `<vendor>:<product>` son los valores hexadecimales mostrados tras ID al ejecutar la orden sin opciones (por ejemplo, `046d:c016`).

Otra orden es `lspci`, que muestra información sobre todos los buses PCI del sistema y todos los dispositivos conectados a ellos. También admite la opción `-v` para imprimir información detallada de los dispositivos. A continuación se muestra un ejemplo de salida obtenida con esta orden:

```
# lspci
00:02.0 VGA compatible controller: Intel Corp. 82852/855GM
          Integrated Graphics Device (rev 02)
...
```

7. udev

Gran parte de la gestión de periféricos en Linux se hace en espacio de usuario mediante *udev*, que es un *userspace device manager* o gestor de dispositivos en espacio de usuario. Así, cuando un dispositivo se conecta o se desconecta, el núcleo simplemente genera un evento del tipo «se ha conectado un nuevo dispositivo» o «se ha desconectado un dispositivo». *udev* recibe este evento y, mediante la información accesible a través de `/sys`, determina qué módulos hay que cargar o descargar, qué fichero (o ficheros) especial de dispositivo hay que crear o borrar, qué programas hay que ejecutar para terminar de configurar un dispositivo conectado, etc.

El objetivo principal de *udev* es que, al conectar un dispositivo, el usuario pueda utilizarlo directamente, sin intervenir en el proceso de preparación del dispositivo. Aunque esto es lógico y natural hoy en día, no ha sido así durante muchos años, puesto que la preparación de los dispositivos para su uso se hacía a mano. Otro objetivo de *udev* es proporcionar un directorio de dispositivos `/dev` dinámico, que contenga sólo los ficheros de los dispositivos conectados en un momento dado.

El proceso que se encarga de realizar las tareas que hemos comentado cuando un dispositivo se conecta/desconecta es el demonio `systemd-udevd`. Este demonio utiliza información de `/sys` y de los *ficheros de reglas* (*rules*), almacenados en `/lib/udev/rules.d/` y `/etc/udev/rules.d/`, para saber exactamente qué hacer: qué nombre asignar al fichero especial de dispositivo, qué permisos y propietario darle, qué programa ejecutar (si es necesario), qué módulos se han de cargar, etc.

Cuando se conecta o desconecta un dispositivo, `systemd-udevd` procesa todos los ficheros de reglas en orden alfabético. Los nombres de estos ficheros suelen ser de la forma `<dos_dígitos>-<nombre_descriptivo>.rules`, por ejemplo, `50-udev-default.rules` o `60-cdrom_id.rules`. Si hay una regla con el mismo nombre en `/etc/udev/rules.d/` y `/lib/udev/rules.d/`, la primera tiene preferencia.

De forma resumida, el funcionamiento de `systemd-udevd`, cuando se conecta un dispositivo, es el siguiente:

1. Recibe una notificación del núcleo, que le indica que un nuevo dispositivo ha sido conectado.
2. Mira en `/sys` si el *driver* correspondiente proporciona un fichero `dev` que contenga el número mayor y menor para el fichero especial de dispositivo a crear.
3. Usa las reglas para preparar el dispositivo. En concreto:
   - Crea el fichero especial de dispositivo y los enlaces simbólicos al mismo con los nombres alternativos propuestos (si los hay).
   - Establece los permisos y los propietarios del fichero especial de dispositivo creado
   - Si es necesario, carga los módulos y el *firmware* para poder trabajar con el dispositivo.
   - Si alguna regla lo indica, ejecuta los programas especificados para inicializar el dispositivo, etc.

Cuando un dispositivo se desconecta, el funcionamiento de `systemd-udevd` es similar, si bien las reglas determinan qué hacer para liberar los recursos usados por el dispositivo.

8. Planificadores de disco

Como veremos en la parte de teoría, existen diversos planificadores de E/S que determinan el orden en el que se atienden las peticiones de lectura y escritura de disco. En particular, el objetivo de estos planificadores es aumentar la productividad (peticiones de disco completadas por unidad de tiempo) reordenando las peticiones en función de las direcciones lógicas de los bloques asociados a las peticiones e intentando agruparlas. Aunque esto puede aumentar la productividad general, puede conllevar también que algunas peticiones esperen demasiado tiempo, causando problemas de latencia. De esta forma, los planificadores de E/S intentan equilibrar la necesidad de alta productividad con una atención lo más equitativa posible de las peticiones de disco de los distintos procesos.

En el caso de Linux, y más concretamente en versiones del núcleo a partir de la 5.0, se dispone de diversos planificadores multicola (*multiqueue I/O schedulers*), cada uno potenciando unas metas (en general, no hay un planificador óptimo para los diferentes tipos de demandas de E/S que podemos encontrar en un sistema). Estos planificadores distribuyen las peticiones en varias colas, las cuales son gestionadas por hilos del kernel que idealmente se ejecutan en diferentes núcleos del procesador. En concreto, los planificadores de los que disponemos son los siguientes:

bfq (*Budget Fair Queuing*): es un planificador diseñado para proporcionar una buena respuesta interactiva, especialmente para discos lentos. En concreto, este planificador trata de repartir el tiempo de uso del disco entre los distintos procesos de forma equitativa. A cada proceso que realiza peticiones de lectura y escritura sobre un dispositivo de almacenamiento, se le asocia un peso y una cola. El planificador otorga acceso exclusivo al dispositivo a una de las colas cada vez (a un proceso), definiendo para cada cola, en función de su peso, una cantidad máxima de accesos (*budget*), medida en número de sectores. Se trata de un planificador complejo, que introduce bastante sobrecarga por cada petición de disco, por lo que no es recomendable su uso con CPUs lentas o discos de alto rendimiento (SSD modernos y NVMe).

kyber: este planificador, más simple, y por lo tanto, más rápido, está inspirado en las técnicas de gestión de colas activas empleadas en encaminamiento en redes. En concreto, las peticiones de disco se distribuyen en dos colas: una para las peticiones síncronas (ej. lecturas bloqueantes) y otra para las peticiones asíncronas (ej. escrituras). Además, se establecen límites estrictos en el número de peticiones enviadas a cada cola (a través de un mecanismo basado en *tokens*), lo que en teoría limita el tiempo de espera de las peticiones para ser atendidas, y por lo tanto debería proporcionar un tiempo de finalización corto para las peticiones de alta prioridad.

mq-deadline: este planificador asigna plazos de tiempo a las peticiones de las distintas colas e intenta evitar que dichos plazos expiren. Cuando todas las peticiones están dentro de sus plazos, estas se atienden según un algoritmo SCAN. Si hay peticiones con plazos cumplidos, entonces son las primeras en ser atendidas según un algoritmo FCFS.

none: funciona como un planificador FCFS (no se reordenan las peticiones de disco).

El que se use uno u otro planificador depende del tipo de dispositivo de almacenamiento y de la carga de trabajo que reciba. Así, para dispositivos SSD o NVMe, se observa poca diferencia en productividad entre los diferentes planificadores, con lo que lo mejor es utilizar *none*, dado que es el que menos tiempo de CPU requiere. El resto de planificadores introducen una mayor sobrecarga que lo único que hace es que las peticiones tarden más de forma innecesaria, lo que acaba perjudicando el rendimiento.

Por el contrario, para discos duros tradicionales (con platos magnéticos giratorios) es importante ordenar las peticiones teniendo en cuenta las direcciones de disco de las mismas. Esto, como comentamos en el Tema 6 de teoría, ayuda a reducir el tiempo de búsqueda, y por ende, a minimizar la latencia media de las peticiones de disco, mejorando, por lo tanto, el número de peticiones completadas por unidad de tiempo (productividad). Para servidores con gran carga de peticiones de disco, se ha comprobado que *mq-deadline* suele ser la mejor opción. Por el contrario, para equipos de sobremesa se ha visto que *bfq* es capaz de cargar algunas aplicaciones de manera más rápida, por lo que debiera constituir la opción a usar.

8.1. Cambio de planificador

Cada dispositivo de almacenamiento tiene su propio planificador, es decir, no hay un único planificador de disco global para todos ellos. Además, el planificador se puede cambiar en caliente, sin reiniciar el sistema. Podemos consultar y modificar el planificador usado por un disco a través de `/sys`. Así, si queremos conocer el planificador utilizado para el primer disco duro (es decir, `/dev/sda`), basta con ejecutar la siguiente orden:

```
# cat /sys/block/sda/queue/scheduler
mq-deadline kyber [bfq] none
```

que, como vemos, muestra el planificador actual entre corchetes (*bfq*). Cambiar de planificador es tan sencillo como ejecutar lo siguiente:

```
# echo mq-deadline > /sys/block/sda/queue/scheduler
# cat /sys/block/sda/queue/scheduler
[mq-deadline] kyber bfq none
```

Hay que tener en cuenta que, muchas veces, se usa *bfq* para un disco SSD o una memoria flash por ser el planificador por defecto. En estos casos, podemos ejecutar las órdenes anteriores para cambiar de planificador. También podemos cambiar el planificador por defecto pasándole al núcleo de Linux el parámetro `elevator=<planificador>` (siendo el planificador `mq-deadline`, `kyber`, `bfq` o `none`) a través de GRUB (en el punto 9.4 de los ejercicios que aparecen al final de boletín se explica cómo pasar parámetros al kernel de Linux).

8.2. Prioridad de la E/S de disco

Por defecto, las peticiones de disco de los procesos tienen la misma prioridad: todas se envían a disco y se tratan de la misma manera, ordenadas para ser servidas según el planificador usado. Sin embargo, cuando utilizamos el planificador *bfq*, esta prioridad se puede cambiar de dos formas.

La primera forma, y también la más adecuada, es utilizar la orden `ionice` a la hora de ejecutar un proceso. Esta orden define las siguientes tres clases o categorías de planificación, donde cada proceso solo puede pertenecer a una ellas:

Desocupado (*idle*): las peticiones de E/S de un proceso perteneciente a esta clase solo se atenderán cuando ningún otro proceso haya enviado peticiones de E/S durante un cierto tiempo. El impacto de un proceso de esta clase sobre la actividad normal de un sistema debería ser cero.

Mejor esfuerzo (*best-effort*): esta clase define 8 niveles de prioridad, desde 0 hasta 7, donde un número menor significa mayor prioridad. Los procesos que pertenecen al mismo nivel de prioridad, dentro de esta clase, se atienden siguiendo un algoritmo *round-robin*.

Tiempo real (*realtime*): las peticiones de los procesos de esta clase son las primeras que se envían a disco, sin importar que haya otros procesos accediendo a disco. Por lo tanto, esta clase debe usarse con precaución, ya que un proceso puede impedir que otros usen el disco. Al igual que en la clase anterior, se definen 8 niveles de prioridad, que indican no qué peticiones se atienden primero sino cómo de grande es la porción de tiempo de disco que recibirá cada proceso, en función del nivel en el que se encuentre.

Según las clases descritas, podemos ver que la planificación de las peticiones de disco es similar a una planificación de procesos multinivel sin realimentación (ya que la peticiones no pueden cambiar de cola). Es decir, primero se atenderán las peticiones de disco de los procesos pertenecientes a la clase *tiempo real*, donde cada proceso recibirá más o menos tiempo de disco en función de su nivel de prioridad. Solo cuando no haya peticiones en esta clase se atenderán las peticiones de los procesos de la clase *mejor esfuerzo*. Dentro de esta clase, primero recibirán tiempo de disco los procesos pertenecientes al nivel de mayor prioridad. Finalmente, los procesos de la clase *desocupado* solo recibirán el disco cuando no existan peticiones pendientes del resto de procesos.

Por motivos obvios, solo el superusuario puede ejecutar procesos dentro de la clase de tiempo real. El resto de usuarios tiene que usar las otras dos clases.

Al ejecutar un programa con `ionice`, si no se indica ninguna clase, el proceso que se cree pertenecerá al nivel de prioridad 4 de la clase *mejor esfuerzo*. Una vez en ejecución, se puede cambiar la clase (y el nivel de prioridad, si es el caso) a la que pertenece un proceso con la misma orden `ionice`, utilizando la opción `-p <pid>`. A diferencia de `renice`, donde un usuario normal solo puede disminuir la prioridad de un proceso, con `ionice` es posible tanto aumentar como disminuir la prioridad

de la E/S de disco en cualquier momento (exceptuando el uso de la clase *tiempo real* que, como ya hemos dicho, solo puede ser usada por el superusuario).

Si un programa se ejecuta sin usar `ionice`, entonces pertenecerá a la clase *ninguna* (o *none*). Esta clase es, en realidad, la clase *mejor esfuerzo*, donde el nivel o prioridad al que pertenece un proceso depende de su valor *nice* de CPU, siguiendo la fórmula:

$$prioridad\_E\_S = \frac{valor\_nice + 20}{5}$$

Por lo tanto, la segunda forma (de las dos que hemos dicho que existen) de cambiar la prioridad de la E/S de un proceso es cambiando su valor *nice*, siempre que el programa correspondiente no se haya ejecutado usando `ionice`. Ya que el valor *nice* de un proceso de usuario normal varía entre 0 y 19 (siendo 0 el valor por defecto), la fórmula anterior nos dice que la prioridad de E/S de uno de estos procesos puede estar entre 4 y 7. Solo el superusuario puede ejecutar procesos con un valor *nice* negativo y, por tanto, solo él podrá ejecutar procesos pertenecientes a los niveles 0, 1, 2 ó 3 dentro de la clase *mejor esfuerzo*.

`ionice` tiene tres opciones principales:

- `-c <clase>` o `--class <clase>`, para indicar la clase a la que debe pertenecer el proceso. `<clase>` puede tomar los valores 0 para *ninguna*, 1 para *tiempo real*, 2 para *mejor esfuerzo* y 3 para *desocupado*.
- `-n <nivel>` o `--classdata <nivel>`, que permite indicar el nivel, dentro de una clase con niveles, al que debe pertenecer el proceso.
- `-p <PID>` o `--pid <PID>`, para especificar el PID del proceso en ejecución al que establecer la prioridad o del que obtener su prioridad de E/S.

Por ejemplo, la orden:

```
# ionice -c 3 -p 89
```

pasa al proceso con PID 89 a la clase *desocupado*, mientras que:

```
# ionice -c 2 -n 0 bash
```

ejecuta el programa `bash` con la prioridad más alta dentro de la clase *mejor esfuerzo*. En cambio, la orden:

```
# ionice -p 5225
```

muestra la prioridad de E/S actual del proceso con PID 5225.

8.3. La orden `iotop`

Imaginemos ahora que nuestro sistema empieza a registrar una gran actividad de disco, lo que provoca una caída en su rendimiento y cierta lentitud en las respuestas. Muy probablemente, esta actividad esté generada por uno o varios procesos que están lanzando muchas peticiones de E/S[3]. Podemos mejorar el rendimiento de nuestro sistema si identificamos a los procesos «culpables» y bajamos su prioridad de E/S. Para ello, podemos usar la orden `iotop`.

`iotop` proporciona una funcionalidad similar a la de la orden `top`, pero aplicada a la E/S: muestra varias columnas, con información sobre la actividad de E/S de cada proceso, y permite ordenar los procesos en base a esas columnas. A continuación mostramos una posible salida de la orden `iotop`:

[3]La actividad de disco también podría estar provocada por un gran número de fallos de página. Este extremo podría ser confirmado por la orden `vmstat`, entre otras.

```
Total DISK READ:      37.45 M/s | Total DISK WRITE:       0.00 B/s
  TID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN     IO>    COMMAND
 4092 be/4 alumno     37.45 M/s    0.00 B/s  0.00 % 60.38 % dd if=disco.img of=/dev/null
    2 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [kthreadd]
    3 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [ksoftirqd/0]
    4 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [kworker/0:0]
    6 rt/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [migration/0]
    7 rt/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [watchdog/0]
   13 be/0 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [cpuset]
   14 be/0 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [khelper]
   15 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [kdevtmpfs]
   16 be/0 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [netns]
   17 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [sync_supers]
   18 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [bdi-default]
   19 be/0 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [kintegrityd]
   20 be/0 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [kblockd]
   21 be/0 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [ata_sff]
   22 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [khubd]
   23 be/0 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [md]
   26 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [kswapd0]
   27 be/5 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [ksmd]
   28 be/7 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [khugepaged]
   29 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [fsnotify_mark]
   30 be/0 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [crypto]
   36 be/0 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [kthrotld]
   44 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [scsi_eh_0]
   45 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [scsi_eh_1]
   46 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [scsi_eh_2]
   47 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [scsi_eh_3]
   48 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [scsi_eh_4]
```

Como podemos ver, `iotop` muestra una línea por hilo en ejecución (mediante la opción `-P` se le puede decir que muestre procesos y no hilos) y, para cada hilo, las columnas que aparecen proporcionan la siguiente información: identificador del hilo (`TID`), su clase y el nivel de prioridad de E/S, si es aplicable a la clase (`PRIO`), propietario (`USER`), tasa de transferencia en lecturas (`DISK READ`) y en escrituras (`DISK WRITE`), porcentaje de tiempo que el hilo pasa esperando a que se lean las páginas para las que ha producido fallos (`SWAPIN`), porcentaje de tiempo que el hilo espera a que se atiendan sus peticiones de E/S de disco (`IO`) y línea de órdenes correspondiente (`COMMAND`).

En el ejemplo dado, la información está ordenada por la columna `IO`, de mayor a menor. Esto se indica con el símbolo «>» a la derecha de `IO`. Si pulsamos la tecla «r» se cambiará el «>» por un «<» y la información aparecerá ordenada en orden ascendente. Si volvemos a pulsar «r» se volverá al orden descendente. Podemos cambiar la columna por la que se ordena usando los cursores izquierdo y derecho.

Una vez identificado el proceso que produce una mayor actividad de disco, podemos bajar su prioridad de E/S usando `ionice`. Observa que, aunque podemos hacer esto aumentando el valor *nice* del proceso, nadie nos garantiza que eso sea suficiente. La mejor opción es, sin duda, usar `ionice` y mover el proceso a la clase *desocupado*.

Una pregunta que cabe hacerse cuando se usa `iotop` es por qué no se muestra información sobre las páginas que se expulsan o por qué un proceso que escribe mucho en disco apenas espera en E/S. La respuesta es que, debido a las cachés que usa el sistema operativo, las modificaciones de páginas y bloques de disco producidas por los procesos quedan en RAM hasta que los demonios responsables del sistema operativo (implementados como hilos del núcleo) deciden guardarlas en disco, por lo que `iotop` muestra la actividad de dichas escrituras en los hilos del núcleo y no en los procesos que originalmente produjeron las modificaciones.

9. Ejercicios

Realiza los siguientes ejercicios sobre la máquina virtual:

9.1. Ficheros especiales de dispositivo

1. Ya que los ordenadores actuales no suelen disponer de disquetera, muchas distribuciones no crean el fichero especial de dispositivo correspondiente (que es `/dev/fd0`) cuando ésta sí existe.

   a) Comprueba si existe el fichero `/dev/fd0`.

   b) Crea de nuevo el fichero ejecutando la siguiente la orden:

   ```
   # mknod /dev/fd0 b 2 0
   ```

   Nota que hemos creado un fichero especial de dispositivo de bloques («b») con número mayor 2 y número menor 0.

2. ¿Qué número mayor corresponde al primer disco duro del ordenador? ¿Qué número menor corresponde a cada una de sus particiones?

3. Los ficheros especiales de dispositivo permiten acceder a los dispositivos como si fueran ficheros. Vamos a aprovechar esto para hacer una copia de seguridad de la tabla de particiones del primer disco duro. Para ello, ejecuta la siguiente orden:

   ```
   # dd if=/dev/sda of=mbr.img count=1
   ```

4. Aprovecha la idea del ejercicio anterior para hacer una copia, sector a sector, de la primera partición del primer disco duro (`sda1`) a un fichero. ¿Puedes montar este fichero como dispositivo `loop`?. En caso afirmativo, hazlo y comprueba qué hay en el fichero. Por último, no olvides borrar el fichero cuando termines este ejercicio.

9.2. Gestión de módulos

5. Lista los módulos que hay cargados en el sistema en este momento. Comprueba que el listado dado contiene los mismos módulos que los indicados en el fichero `/proc/modules`.

6. Si el módulo `vfat` está cargado, descárgalo haciendo uso de la orden `modprobe -r`. Si no puedes descargarlo es porque está siendo usado. Mira si hay montada alguna partición de Windows y, en ese caso, desmóntala y después prueba a descargarlo de nuevo.

7. Consulta la información de los módulos `fat` y `vfat`. ¿Qué dependencias presenta cada uno?

8. Haciendo uso exclusivo de `insmod` (no uses `modprobe`), ejecuta los pasos necesarios para cargar el módulo `vfat`. Recuerda que, para saber qué versión del núcleo estás usando, puedes ejecutar la orden `uname -r`. Comprueba si realmente se ha cargado el módulo.

9. Comprueba, con la orden `modprobe` (no uses `modinfo`), las dependencias de los módulos `raid1` y `raid456`.

10. Con `modprobe`, carga el módulo `raid456` y comprueba si la carga se ha realizado de forma correcta, es decir, si se han cargado ese módulo y sus dependencias.

11. En el directorio `/etc/modprobe.d/`, crea un fichero llamado `misalias.conf` con el siguiente contenido:

    ```
    alias sfwindows vfat
    alias moduloraid1 raid1
    ```

    Realiza ahora los siguientes ejercicios:

    a) Carga los módulos `raid1` y `vfat`, usando para ello los alias asignados. Verifica que la carga se ha realizado correctamente.

b) Con `rmmod`, intenta descargar el módulo `fat`. ¿Qué sucede? ¿Por qué?

c) Utilizando los nombres de los alias y la orden `rmmod`, descarga los módulos cargados anteriormente. ¿Qué sucede? ¿Por qué? ¿Ocurre lo mismo si usas la orden `modprobe`?

d) Vuelve a cargar el módulo `vfat`. Descárgalo ahora usando `rmmod`. ¿Se ha hecho bien? ¿Se ha descargado el módulo del que dependía? Si no es así, descarga también este segundo módulo usando de nuevo `rmmod`.

9.3. Gestión de dispositivos

Para la realización de estos ejercicios vamos a emplear un *pendrive*, al que podremos acceder desde la máquina virtual una vez lo hayamos insertado (si no aparece automáticamente en la máquina virtual, habilítalo usando la opción «USB device selection» del menú «Fichero» del visor remoto):

12. Sin insertar el *pendrive*, observa el contenido del directorio `/dev/` y comprueba que determinados ficheros especiales de dispositivo, como, por ejemplo, `/dev/sde*`, no existen.

13. Ejecuta la orden `lsusb` y observa la información que ofrece sobre los dispositivos USB conectados.

14. Conecta el *pendrive* y comprueba que se han creado los dispositivos correspondientes, `/dev/sde` y `/dev/sde<N>`, donde, de los segundos, habrá tantos como particiones haya en el *pendrive*, siendo «N» el número de partición. Lo normal será que tengamos una única partición y que, por tanto, sólo exista `/dev/sde1`.

15. De nuevo, ejecuta la orden `lsusb` y verifica si aparece la información sobre el dispositivo conectado. A continuación, ejecuta otra vez dicha orden, pero con la opción `-v` para obtener más información sobre el dispositivo.

16. Desconecta el *pendrive* y observa que los ficheros creados en `/dev/` desaparecen.

17. Finalmente, usa la orden `lspci` para mostrar los dispositivos conectados al bus PCI (y/o PCIe, según el ordenador). ¿Qué modelo de tarjeta gráfica muestra?

9.4. Planificadores de disco

Para la realización de los siguientes ejercicios debes arrancar la máquina virtual con una configuración de memoria de 1 GiB. Para ello debes editar una entrada en el menú de GRUB[4] y asignarle el valor «1024m» al parámetro del núcleo `mem` (es decir, `mem=1024m`)[5], y seguidamente, arrancar con esta entrada modificada pulsando «Ctrl-x»:

18. Averigua qué planificador está usando el primer disco duro de la máquina virtual. Cámbialo a BFQ si no se está usando este planificador.

19. Vamos a lanzar dos procesos intensivos en E/S, y vamos a medir el tiempo que tardan en ejecutarse. Para ello, ejecuta las siguientes órdenes (lo mejor es escribir las tres órdenes seguidas, en una única línea, lo que nos permitirá ejecutarlas a la vez):

```
# time dd if=/dev/sda of=/dev/null bs=1M count=4096 &
# time dd if=/dev/sda of=/dev/null bs=1M count=4096 skip=2048 &
# wait
```

[4]El menú de GRUB aparece inmediatamente después de iniciar la máquina virtual. Verás que se listan un total de cuatro entradas. Edita, por ejemplo, la primera de ellas pulsando la tecla «e» sobre la misma. Una vez hecho esto, se pasa a una interfaz de edición en la que se puede modificar el contenido de la entrada.

[5]Tendrás que añadir el valor `mem=1024m` después de «quiet», en la línea que comienza con «linux»

Estas tres órdenes hacen lo siguiente. La primera lee los primeros 4096 MiB del primer disco duro del ordenador. La segunda lee también 4096 MiB, pero tras saltar unos 2 GiB en el disco. Finalmente, la última orden espera a que las otras dos terminen (las hemos ejecutado en segundo plano). La orden `time` se utiliza para mostrar cuánto tarda en ejecutarse cada orden `dd`.

Observa que, en total, se leen 8 GiB de datos. Con esto se trata de evitar que la información a leer quepa en RAM, lo que haría que la segunda lectura y sucesivas (que usaremos en los siguientes ejercicios) fueran muy rápidas, ya que la información se leería de RAM y no del disco duro.

Ejecuta en una segunda terminal la orden `iotop` a la misma vez y comprueba que, efectivamente, las dos órdenes `dd` son las que pasan la mayor parte de su tiempo esperando en E/S. Para verlo es conveniente ordenar en `iotop` por la columna `DISK READ`.

20. Podemos ver que en el caso anterior las dos órdenes `dd` tardan más o menos el mismo tiempo en terminar. Vamos a cambiar esto usando `ionice` y haciendo que la primera orden se ejecute en la clase *desocupado*. Para ello, ejecuta las tres órdenes anteriores de la siguiente manera (recuerda que es mejor escribirlo todo en una única línea de órdenes):

```
# time ionice -c 3 dd if=/dev/sda of=/dev/null bs=1M count=4096 &
# time dd if=/dev/sda of=/dev/null bs=1M count=4096 skip=2048 &
# wait
```

Observa ahora que la segunda orden `dd` termina mucho antes que la primera. Si consultamos la salida de `iotop`, veremos también que esta segunda orden es, de las dos, la que pasa un porcentaje menor de tiempo esperando en E/S.

21. Podemos conseguir el mismo resultado aumentando la prioridad de la segunda orden `dd`. Esto lo vamos a hacer ejecutándola en la clase *tiempo real*. Para ello, ejecuta lo siguiente:

```
# time dd if=/dev/sda of=/dev/null bs=1M count=4096 &
# time ionice -c 1 dd if=/dev/sda of=/dev/null bs=1M count=4096 skip=2048 &
# wait
```

Observa que, ahora, el sistema parece estar congelado hasta que termina de ejecutarse la segunda orden `dd`. Esto pone de manifiesto el riesgo que supone usar esta clase, la cual debe emplearse con precaución.

22. Consigue, de nuevo, que la segunda orden `dd` termine antes que la primera, pero haciendo ahora que las dos pertenezcan a la clase *mejor esfuerzo*.

23. Cambia el planificador a «none» y repite el ejercicio 20. ¿Por qué razón no se observan ahora las diferencias en tiempo vistas antes?