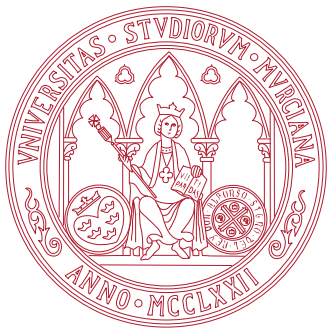

Universidad de Murcia

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 8 – Control de los recursos del sistema

Curso 2021/2022

Índice

1. Introducción

Hoy en día, para gestionar qué software queremos instalar en nuestro sistema, la mayoría de los distribuidores de Linux utilizan paquetes de software. Por esta razón, como administradores del sistema, tenemos que ser capaces de gestionar qué paquetes de software queremos tener instalados en el sistema.

Por otra parte, también debemos controlar el rendimiento del sistema (qué procesos están en ejecución o qué cantidad de memoria tenemos disponible), detectar posibles problemas que aparezcan y, si fuese posible, solucionarlos.

2. Objetivos

Al finalizar este boletín vamos a ser capaces de:

- Administrar los paquetes software del sistema.
- Controlar del uso de la CPU: qué procesos están en ejecución, carga media del sistema, porcentaje de uso de la CPU, etc.
- Priorizar la ejecución de los procesos, asignando para ellos diferentes valores de prioridades.
- Programar tareas para que se ejecuten en un momento concreto, o programar tareas para que se ejecuten con una cierta periodicidad.
- Controlar el uso de la memoria: cantidad de memoria RAM ocupada o libre, cantidad de la memoria de intercambio ocupada o libre, cantidad de memoria usada por un proceso, etc.

3. Órdenes utilizadas

Las órdenes que vamos a usar son:

- `rpm`
- `dnf`
- `pstree`
- `ps`
- `top`
- `nice` y `renice`
- `kill` y `killall`
- `at`
- `crontab`
- `free`
- `vmstat`
- `mkswap`
- `swapon`

4. Administración de paquetes

Un paquete se encuentra en un fichero comprimido con un formato particular, conteniendo información del producto, ficheros de programa, iconos, documentación y scripts de administración. Las aplicaciones de administración utilizan estos ficheros para ubicar, instalar, actualizar y eliminar software de una forma segura. Por ejemplo, el proceso de instalación de Fedora utiliza los paquetes suministrados con la correspondiente distribución de Fedora para construir o actualizar un sistema según sus requerimientos.

Puede ser compilable, si se basa en código fuente, o instalable, si lo hace en binarios compilados ya para una determinada arquitectura o plataforma.

Hay varios formatos de paquete con sus correspondientes herramientas de gestión, los más conocidos son deb creado por Debian y RPM por Red Hat en cuanto a binarios, y los ebuilds de Gentoo en cuanto a compilables. El sistema de paquetes utilizado por Fedora es RPM, heredado de Red Hat.

Normalmente el nombre de fichero de un paquete tiene este formato: [nombrepaquete-versión-revisión.[*arquitectura* | src | noarch].rpm].

La arquitectura de hardware señalada por este nombre de fichero es el mínimo tipo de máquina requerida para utilizar el paquete. De esta manera, teniendo en cuenta las posibles arquitecturas, tenemos:

- `noarch.rpm`: Apropiado para cualquier tipo de arquitectura.
- `i386.rpm`: Apropiado para cualquier procesador compatible con Intel.
- `i586.rpm`: Apropiado para cualquier procesador compatible con Intel Pentium y superior.
- `i686.rpm`: Apropiado para cualquier procesador compatible con Intel Pentium Pro y superior.
- `x86_64.rpm`: Apropiado para cualquier procesador compatible con Intel de 64-bit.

Un fichero de paquete `src.rpm` contiene código fuente, por lo que tendrá que ser compilado, usando la orden `rpmbuild`, para poder hacer uso del mismo.

El sistema gestiona una base de datos de los paquetes instalados que se puede usar para obtener información de su estado y sus ficheros.

Una vez que un paquete está instalado, podremos referenciarlo usando simplemente su nombre de paquete, ya no será necesario indicar ni siquiera la versión. Por ejemplo, cuando hayamos instalado el paquete contenido en el fichero `coreutils-8.10-2.fc15.x86_64.rpm`, podremos referenciarlo simplemente por `coreutils`.

Pueden existir dependencias entre paquetes, de forma que para instalar un paquete sea necesario instalar antes otros paquetes, o que al querer desinstalar un paquete no se pueda porque haya dependencias con otros que hay instalados (al eliminar el paquete los otros no funcionarían correctamente). Estas dependencias son controladas por el sistema RPM.

4.1. La herramienta `rpm`

La herramienta `rpm` (*RPM Package Manager*) proporciona la funcionalidad básica para la instalación, consulta y eliminación de paquetes.

A continuación se presentan algunos modos básicos de uso:

- Instalación: `rpm -i <ficheropaquete.rpm>`. Si se intenta instalar un paquete que necesita que algún otro paquete esté instalado previamente, la operación no se realizará, mostrándose un mensaje de error que indique dicha dependencia.
- Actualización: `rpm -U <ficheropaquete.rpm>`. En caso de que exista un fichero de configuración modificado de la versión anterior lo guarda como `<ficheropaqueteanterior.rpmsave>`. En ocasiones, si un fichero de configuración fue modificado, y el nuevo fichero de configuración ha cambiado de formato y no puede ser adaptado, se deja el fichero antiguo y se crea uno nuevo que se llamará `<ficheropaquete.rpmnew>`. El administrador es el responsable de adaptar el fichero antiguo de configuración al nuevo formato.

- Eliminación: `rpm -e <paquete>`. Si se intenta borrar un paquete que es necesario para algún otro paquete, la operación no se realizará y se recibirá un mensaje de error indicando dichas dependencias.
- Consulta: `rpm -q [query-options]`. A esta opción se le adjunta la subopción correspondiente a lo que queremos consultar (`query-options:-i` para información general de un paquete, `-l` para ver el listado de ficheros que lo componen, `-f` para saber a qué paquete pertenece un fichero,...) de esta manera:

  ```
  rpm -q[li] <paquete>
  rpm -q[f] <fichero>
  ```

- Verificación del estado de un paquete: `rpm -V <paquete>`.

4.2. La herramienta `dnf`

`dnf` es una herramienta que permite actualizar de forma automática o interactiva paquetes. Puede ser usada para mantener actualizados los sistemas que usen paquetes RPM de una manera cómoda y sencilla, pues, a diferencia de la herramienta `rpm`, cuenta con resolución automática de dependencias, de forma que durante la instalación de un paquete detecta las dependencias que le atañen, descargando e instalando los paquetes extras que sean necesarios. De igual forma, a la hora de desinstalar un paquete con `dnf`, de manera automática se desinstalan también todos los paquetes que dependan del paquete que se ha ordenado desinstalar, de cara a no dejarlos inconsistentes.

Esta herramienta permite varias formas de uso para instalar o actualizar paquetes, borrarlos, etc. A continuación se lista un breve resumen de las mismas:

- `dnf check-update`: avisa de los paquetes con actualizaciones pendientes.
- `dnf update paquete`: para actualizar un paquete (si existe actualización).
- `dnf update`: para actualizar todos los paquetes con versiones más recientes.
- `dnf install paquete1` [`paquete2 ...`]: descarga e instala la última versión del paquete o paquetes indicados.
- `dnf download paquete1` [`paquete2 ...`]: descarga la última versión del paquete o paquetes indicados, pero sin llegar a instalarlos.
- `dnf remove | erase paquete1` [`paquete2 ...`]: elimina el paquete que se le indica.
- `dnf clean` [`packages | headers | all`]: permite limpiar los ficheros que se acumulan en el directorio caché de `dnf`.
- `dnf -h`: para obtener más información.

`dnf` usa como fichero de configuración `/etc/dnf/dnf.conf`. Pero, ¿cómo sabe `dnf` de dónde descargar los paquetes? El punto de inicio es el directorio `/etc/yum.repos.d/`, el cual contiene, por lo general, varios ficheros `repo`. Un fichero `repo` común se divide en tres secciones: una para los paquetes normales, otra para los paquetes de depuración, y una tercera para los paquetes fuentes. Por lo general, habrá varias copias de un paquete de distribución disponible en varias ubicaciones, o espejos, denominados repositorios. Por lo tanto, el fichero `repo` le informa a `dnf` sobre dónde puede encontrar la última lista de espejos para cada sección. Por ejemplo:

- El fichero `fedora.repo` contiene la información para localizar la instalación base.
- Por otro lado, en el fichero `fedora-updates.repo` se indica la información para localizar actualizaciones de paquetes.

Podemos añadir nuevos ficheros `repo` de cara a poder instalar algún paquete que no está recogido en ninguno de los repositorios que tenemos en un momento dado.

Por ejemplo, si partimos de este conjunto de ficheros `repo`:

```
# ls -l /etc/yum.repos.d/
total 28
-rw-r--r--. 1 root root  728 ago 25 22:31 fedora-cisco-openh264.repo
-rw-r--r--. 1 root root 1303 ago 25 22:31 fedora-modular.repo
-rw-r--r--. 1 root root 1239 ago 25 22:31 fedora.repo
-rw-r--r--. 1 root root 1349 ago 25 22:31 fedora-updates-modular.repo
-rw-r--r--. 1 root root 1286 ago 25 22:31 fedora-updates.repo
-rw-r--r--. 1 root root 1391 ago 25 22:31 fedora-updates-testing-modular.repo
-rw-r--r--. 1 root root 1344 ago 25 22:31 fedora-updates-testing.repo
```

y probamos a instalar el paquete `xmltv`, vemos que no lo tenemos referenciado en ninguno de esos repositorios:

```
# dnf install xmltv
Última comprobación de caducidad de metadatos hecha hace 1:12:21,
el Wed Feb 28 10:56:20 2018.
Error: No se encontraron coincidencias.
```

Este paquete forma parte del repositorio `rpmfusion`, por lo que pasamos a instalar este nuevo repositorio para la versión de Fedora que tengamos (podemos consultar qué versión de Fedora tenemos instalada ejecutando `rpm -E %fedora`):

```
# dnf install http://download1.rpmfusion.org/free/fedora/
rpmfusion-free-release-$(rpm -E %fedora).noarch.rpm
```

Podemos ver que hemos aumentado el conjunto de ficheros `repo`:

```
# ls -l /etc/yum.repos.d/total 32

total 40
-rw-r--r--. 1 root root  728 ago 25 22:31 fedora-cisco-openh264.repo
-rw-r--r--. 1 root root 1303 ago 25 22:31 fedora-modular.repo
-rw-r--r--. 1 root root 1239 ago 25 22:31 fedora.repo
-rw-r--r--. 1 root root 1349 ago 25 22:31 fedora-updates-modular.repo
-rw-r--r--. 1 root root 1286 ago 25 22:31 fedora-updates.repo
-rw-r--r--. 1 root root 1391 ago 25 22:31 fedora-updates-testing-modular.repo
-rw-r--r--. 1 root root 1344 ago 25 22:31 fedora-updates-testing.repo
-rw-r--r--. 1 root root 1248 abr 24  2020 rpmfusion-free.repo
-rw-r--r--. 1 root root 1264 abr 24  2020 rpmfusion-free-updates.repo
-rw-r--r--. 1 root root 1324 abr 24  2020 rpmfusion-free-updates-testing.repo
```

Ahora ya podríamos instalar el paquete:

```
# dnf install xmltv
```

Además de descargar+instalar paquetes desde los repositorios, `dnf` puede simplemente instalar paquetes previamente descargados, como hace la herramienta `rpm`, pero con la ventaja añadida de la resolución automática de dependencias anteriormente explicada. En este caso, a `dnf` hay que pasarle como argumento una ruta hasta ese fichero de paquete que queramos instalar.

5. Control y gestión de la CPU

En esta sección nos vamos a centrar en el control de la CPU como principal recurso del sistema.

5.1. La orden ps

La orden `ps` muestra información de la actividad de los procesos en ejecución. Si no añadimos ninguna opción ni parámetro, `ps` mostrará únicamente los procesos iniciados por el usuario desde la consola actual. Esta orden admite tres juegos de opciones diferentes:

- Opciones de BSD: Estas opciones de un único carácter se pueden agrupar y no van precedidas por ningún guión.
- Opciones de Unix98: Estas opciones de un único carácter se pueden agrupar y van precedidas de un único guión.
- Opciones GNU largas: Estas son opciones multi-carácter y nunca van juntas. Están precedidas por dos guiones.

En principio, se pueden usar mezcladas opciones de estos diferentes juegos, pero no es aconsejable porque pueden aparecer conflictos entre ellas. Si nos centramos en las opciones de BSD (sin guión), una de las combinaciones de opciones que más útil nos va resultar va ser `axu`, siendo:

- `a`: Para listar los procesos de todos los usuarios. Si no está acompañada de otras opciones, se mostrarán únicamente los procesos con terminal asociada.
- `x`: Para listar todos los procesos, tengan o no una terminal asociada. Si no está acompañada de otras opciones, se mostrarán únicamente los procesos del usuario que ejecuta la orden.
- `u`: Para mostrar la información más importante de cada proceso: el usuario que lo está ejecutando, la utilización de CPU y memoria, etc.

Así, por ejemplo:

```
# ps axu

USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          2  0.0  0.0      0     0 ?        S    nov13   0:00 [kthreadd]
root          3  0.0  0.0      0     0 ?        I<   nov13   0:00 [rcu_gp]
root          5  0.0  0.0      0     0 ?        I<   nov13   0:00 [kworker/0:0H]
root          7  0.0  0.0      0     0 ?        I<   nov13   0:00 [mm_percpu_wq]
alumno     7687  0.0  0.2 225820  5152 pts/0    Ss   12:22   0:00 bash
alumno     7899  0.0  0.1 258788  3944 pts/0    R+   12:44   0:00 ps axu
...
```

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

- `USER`: el usuario efectivo del proceso.
- `PID`: identificador del proceso.
- `%CPU`: porcentaje de utilización de la CPU por parte del proceso. Se calcula con el tiempo de CPU transcurrido desde que se inicio el proceso dividido por el tiempo en el que el proceso ha estado ejecutándose, expresado en porcentaje.
- `%MEM`: fracción de memoria RAM consumida (es una estimación).
- `VSZ`: tamaño de la memoria virtual usada por el proceso (suma de los tamaños de las regiones del mapa de memoria del proceso), en KiB.

- RSS: memoria RAM usada, en KiB.
- TTY: terminal asociado con el proceso.
- STAT: estado del proceso:
  - R: en ejecución o listo para ejecutarse.
  - N: prioridad baja (valor mayor que 0).
  - <: prioridad alta (valor menor que 0).
  - T: parado por una señal de stop.
  - t: parado por el depurador.
  - Z: proceso zombie (proceso que ha completado su ejecución, pero aún tiene una entrada en la tabla de procesos, estando a la espera de que el proceso padre ejecute wait).
  - S: durmiendo (bloqueado).
  - D: durmiendo (bloqueado) ininterrumpiblemente (normalmente por E/S de corta duración, como el disco). El proceso está realizando una llamada al sistema, no pudiendo ser interrumpido (o eliminado) hasta que esta llamada al sistema se complete.
  - l: el proceso tiene varios hilos.
  - +: proceso ejecutándose en primer plano.
  - s: proceso líder de sesión[1].
  - I: hilo del núcleo desocupado.
- START: momento en qué se inicio el proceso. Si fue hace menos de 24 horas, el formato en que se muestre será HH:MM (hora y minuto), en otro caso, será mmmDD (iniciales del mes y el día).
- TIME: tiempo acumulado de uso de CPU.
- COMMAND: nombre del proceso.

Si se usa la opción l en lugar de la u, podremos ver otros campos de interés como el identificador del proceso padre (PPID), el UID y los valores de los campos de prioridad (PRI y NI)(ver sección 5.4). También aparece el campo de los flags del proceso, F (la suma de: 1, si el proceso ha nacido de un fork pero no ha realizado un exec, y de 4, si es un proceso con privilegios de root) y el campo WCHAN, que muestra para aquellos procesos que están bloqueados en alguna función del núcleo la dirección de dicha función. Por ejemplo:

```
# ps axl

F   UID   PID  PPID PRI  NI    VSZ   RSS WCHAN  STAT TTY TIME COMMAND
1     0     2     0  20   0      0     0 -      S    ?   0:00 [kthreadd]
1     0     6     2   0 -20      0     0 -      I<   ?   0:00 [mm_percpu_wq]
1     0     7     2  20   0      0     0 -      S    ?   0:13 [ksoftirqd/0]
1     0     8     2  20   0      0     0 -      I    ?   0:57 [rcu_sched]
1     0     9     2  20   0      0     0 -      I    ?   0:00 [rcu_bh]
1     0    10     2 -100  -      0     0 -      S    ?   0:00 [migration/0]
5     0    11     2 -100  -      0     0 -      S    ?   0:00 [watchdog/0]
1     0    12     2  20   0      0     0 -      S    ?   0:00 [cpuhp/0]
...
```

[1]Normalmente, una sesión está asociada con un grupo de procesos. Por ejemplo, en el caso del grupo de procesos que se van creando desde una terminal, el líder de la sesión sería el primer proceso de la terminal, es decir, el correspondiente a la terminal en sí misma

Otra combinación de opciones interesante sería utilizando la opción `o` en lugar de la `u`, seguida de la lista concreta de campos que queremos ver por cada proceso. Por ejemplo:

```
# ps axo uid,user,pid

 UID USER       PID
    0 root     12985
    0 root     13000
 1000 javiercm 13037
 1000 javiercm 13038
    4 lp       21448
 1000 javiercm 21757
...
```

5.2. La orden **pstree**

La orden `pstree` muestra los procesos del sistema en una estructura de árbol que es muy útil para mostrar las relaciones padre-hijo entre ellos, comenzando por el ancestro de todos, el proceso `systemd`. En el formato en que se muestra esta información, cuando existen varios procesos idénticos, hijos del mismo padre, en lugar de repetir la rama de árbol, se simplifica la notación, colocándose una sola rama con el nombre del proceso entre corchetes junto al contador de repeticiones. Por otro lado, los hilos que se crean desde un proceso se muestran como una rama a partir de la del proceso creador, con el nombre del proceso entre llaves junto al contador de repeticiones.

En el siguiente ejemplo, vemos que el proceso `systemd` tiene, entre otros, 3 procesos hijos idénticos, de nombre `abrt-dump-journ`. Por otro lado, se observa que el proceso `accounts-daemon` genera durante su ejecución 2 hilos.

```
# pstree

systemd─┬─NetworkManager─┬─dhclient
        │                └─2*[{NetworkManager}]
        ├─abrt-dbus───2*[{abrt-dbus}]
        ├─3*[abrt-dump-journ]
        ├─abrtd───2*[{abrtd}]
        ├─accounts-daemon───2*[{accounts-daemon}]
        ├─alsactl
        ├─atd
        .
        .
        .
```

Por último, indicar que podemos usar la opción `-p` para que nos muestre el PID de cada proceso listado en esta estructura de árbol.

5.3. La orden **top**

La orden `top` proporciona una visión dinámica de la actividad del sistema en tiempo real, mostrando las tareas que más uso hacen de la CPU y con una interfaz interactiva para manipular procesos. Cada cierto tiempo preestablecido esta información se actualiza, reflejando, en gran medida, lo ocurrido en el intervalo transcurrido desde la última vez que se refrescó. Por defecto, el listado de los procesos se hace por orden decreciente de uso de la CPU, actualizándose la lista normalmente cada 3 segundos. Para salir de esta orden se debe pulsa la letra `q`.

Esta orden nos puede ser muy útil para saber, por ejemplo, si hay algún proceso que puede estar afectando al rendimiento del sistema, haciendo que éste vaya anormalmente lento. Veamos con más detalle la información que nos muestra siguiendo un ejemplo:

```
# top
top - 12:25:42 up 15 days, 2:27, 3 users, load average: 0,08, 0,06, 0,01
Tasks: 200 total, 1 running, 199 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0,2 us, 0,2 sy, 0,0 ni, 99,3 id, 0,0 wa, 0,2 hi, 0,2 si, 0,0 st
MiB Mem : 973,4 total, 60,9 free, 562,1 used, 350,4 buff/cache
MiB Swap: 1536,0 total, 1405,6 free, 130,4 used. 249,1 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND

  677 root      20   0   30672   3560   3268 S   0,3  0,2   0:34.72 qemu-ga
 8340 root      20   0  521316  50696  34628 S   0,3  2,5   0:02.14 Xorg
 8436 alumno    20   0 3628660 251284  98768 S   0,3 12,3   0:06.48 gnome-shell
 8875 alumno    20   0  830600  38324  28376 S   0,3  1,9   0:00.44 gnome-term+
 8929 alumno    20   0  263460   5016   4140 R   0,3  0,2   0:00.10 top
    1 root      20   0  247384  12204   7580 S   0,0  0,6   0:09.57 systemd
    2 root      20   0       0      0      0 S   0,0  0,0   0:00.04 kthreadd
    3 root       0 -20       0      0      0 I   0,0  0,0   0:00.00 rcu_gp
    5 root       0 -20       0      0      0 I   0,0  0,0   0:00.00 kworker/0:+
    7 root       0 -20       0      0      0 I   0,0  0,0   0:00.00 mm_percpu_+
    8 root      20   0       0      0      0 S   0,0  0,0   0:00.56 ksoftirqd/0
    9 root      20   0       0      0      0 I   0,0  0,0   0:03.08 rcu_sched
   10 root      20   0       0      0      0 I   0,0  0,0   0:00.00 rcu_bh
   11 root      rt   0       0      0      0 S   0,0  0,0   0:00.01 migration/0
   12 root      rt   0       0      0      0 S   0,0  0,0   0:00.38 watchdog/0
   13 root      20   0       0      0      0 S   0,0  0,0   0:00.00 cpuhp/0
```

Las cinco primeras líneas muestran información general del sistema:

1. La hora actual, cuánto tiempo lleva en marcha el sistema, el número de terminales de usuario abiertas, y la carga media del sistema en forma de número promedio de procesos en los últimos 1, 5 y 15 minutos (esta carga se expresa en tanto por uno, sin estar normalizado al número de cores de la CPU, por lo que en CPU multicores es habitual valores mayores que 1). Los procesos que se tienen en cuenta son tanto los que están en estado ejecutable (estado `R`), es decir, en ejecución o listo, como los que están en estado de bloqueado no interrumpible a la espera de una operación de E/S de corta duración, normalmente de disco (estado `D`).

   ```
   top - 12:25:42 up 15 days,  2:27,  3 users,  load average: 0,08, 0,06, 0,01
   ```

2. Estadísticas sobre los procesos del sistema (número de procesos, procesos en ejecución, durmiendo, parados y zombies).

   ```
   Tasks: 200 total,   1 running, 199 sleeping,   0 stopped,   0 zombie
   ```

3. El estado de la CPU:

   - `us`: porcentaje de uso de la CPU para tareas a nivel de usuario.
   - `sy`: porcentaje de uso de la CPU para tareas del sistema (no incluye el tratamiento de interrupciones) (modo núcleo).
   - `ni`: porcentaje de uso de la CPU para tareas a nivel de usuario con prioridad base (número `nice`) mayor que 0 (ver sección 5.4).
   - `id`: porcentaje en que la CPU está ociosa (*idle*).
   - `wa`: porcentaje de tiempo que la CPU ha estado esperando, desocupada, a que el sistema finalice operaciones de E/S (normalmente de disco).

- `hi`: porcentaje de tiempo que la CPU ha estado tratando interrupciones hardware.
- `si`: porcentaje de tiempo que la CPU ha estado tratando interrupciones software.
- `st`: porcentaje de tiempo de una máquina virtual que ha sido robado por el hypervisor de dicha maquina.

```
%Cpu(s): 0,2 us, 0,2 sy, 0,0 ni, 99,3 id, 0,0 wa, 0,2 hi, 0,2 si, 0,0 st
```

4. La memoria: la memoria total disponible (`total`), la que se encuentra libre completamente (`free`), la usada (`used`) y, finalmente, la utilizada para caché de disco y memoria caché de páginas (`buff/cache`). El último valor mostrado en línea siguiente (`avail Mem`) realmente está relacionado con la información de la línea actual, pues es una estimación de la cantidad de memoria disponible que se podría usar, tanto por los procesos existentes como para crear nuevos procesos, por lo que incluye la memoria que no está siendo usada en absoluto (la mostrada en la columna `free`) y otras zonas de memoria que también se pueden utilizar dinámicamente cuando sea oportuno (como parte de la mostrada en la columna `buf/cache`).

```
MiB Mem :    973,4 total,     60,9 free,    562,1 used,    350,4 buff/cache
                                                           249,1 avail Mem
```

5. El espacio de swap: swap total disponible (`total`), libre (`free`) y usada (`used`).

```
MiB Swap:   1536,0 total,   1405,6 free,    130,4 used.
```

Los datos de las dos últimas líneas (4 y 5) se corresponden con los que muestra la orden `free`, como veremos más adelante.

A partir de aquí, los datos de la parte inferior son en su mayoría similares a los del `ps`, indicando para cada proceso:

- Columna `PR`: El valor de la prioridad en tiempo real (ver sección 5.4).
- Columna `NI`: El valor `nice`, que indica la prioridad base (ver sección 5.4).
- Columna `VIRT`: La cantidad de memoria virtual que usa. Se corresponde con la columna `VSZ` de `ps`.
- Columna `RES`: La cantidad de memoria física, no intercambiada a disco, que usa. Se corresponde con la columna `RSS` de `ps`.
- Columna `SHR`: La cantidad de memoria que podría ser compartida con otros procesos. Por ejemplo, si el proceso crea un proceso hijo, corresponderá con el tamaño de la zona de memoria (datos y código) que compartirán padre e hijo, con copia en escritura.
- Columna `S`: El estado del proceso. Se corresponde con la columna `STAT` de `ps`.
- Columna `%CPU`: El porcentaje de CPU usado durante el intervalo de actualización del `top`. Es importante resaltar que, como ya vimos anteriormente, en la orden `ps` la columna con este nombre muestra información diferente (porcentaje de tiempo que ha estado en la CPU desde que se creó el proceso).
- Columna `%MEM`: El porcentaje de memoria RAM ocupada. Se corresponde con la columna `%MEM` de `ps`.
- Columna `TIME+`: El total de tiempo de CPU utilizado desde que se creó. Se corresponde con la columna `TIME` de `ps`.

Una ventaja de `top` respecto a `ps` es que con `top` podemos tener información actualizada en una pantalla en todo momento. Otra ventaja es que permite ordenar los procesos según un determinado criterio. Por ejemplo, si pulsamos `M`, `top` ordenará los procesos según la columna `%MEM`, es decir, según el porcentaje de memoria RAM ocupada, mientras que si pulsamos `P` los ordenará por consumo de CPU (columna `%CPU`)(comportamiento por defecto). Una desventaja de `top`, sin embargo, es que no muestra información para todos los procesos en ejecución en el sistema (ya que, por lo general, hay más procesos que líneas se pueden ver en pantalla).

La orden `top` permite realizar una serie de tareas sobre los procesos y sobre la información mostrada de ellos, simplemente pulsando alguna tecla concreta mientras se está ejecutando, como por ejemplo:

- Con `r` se puede cambiar la prioridad de un proceso (ver sección 5.4).
- Con `k` se puede enviar una señal a un proceso (ver sección 5.5).
- Se puede ordenar la lista de procesos mostrados según diferentes criterios, tal como se ha indicado anteriormente (por uso de memoria, pulsando `M`, por uso de CPU, pulsando `P`, por PID, pulsando `N`, por tiempo, pulsando `T`, etc.).
- Con `n` se puede cambiar el número de procesos que se muestran.
- Con `L` se puede buscar un texto entre todo lo que se muestra en pantalla.
- Con `H` se muestra la información por hilos en lugar de por procesos y viceversa.
- Con `1` se muestra la información de la CPU (tercera línea del encabezado) por cada núcleo en lugar de por la CPU en su conjunto y vicerversa.
- Para salir se utiliza la letra `q`, tal como se dijo anteriormente.

5.4. Número `nice` y prioridad de procesos

Inicialmente, al lanzar un proceso de usuario se le asigna un valor de prioridad base (el número `nice`, que podemos ver en la columna `NI` al ejecutar `ps axl` y al ejecutar `top`). El rango de esta prioridad base va desde -20 (máxima prioridad) a 19 (mínima prioridad). Por defecto, cada proceso hereda la prioridad base de su proceso padre. Asignar un valor que mejore la prioridad base del proceso sólo puede hacerlo el `root`.

La prioridad final del proceso de usuario se calcula, a partir de la prioridad base, teniendo en cuenta factores como el consumo de CPU realizado, ejecución de código dentro del núcleo, etc. Normalmente, esta prioridad final suele ser igual al valor `nice` + 20, quedando, por tanto, en el rango de 0 a 39. Esta prioridad final de cada proceso la podemos visualizar en la columna `PRI` al ejecutar `ps axl` y en la columna `PR` al ejecutar `top`.

Órdenes relacionadas:

- `nice -incremento orden_a_ejecutar`
  - Ejemplo: Ejecuta el proceso `firefox`, aumentando en 5 el valor por defecto del número `nice` para empeorar la prioridad de este proceso:

    ```
    nice -5 firefox
    ```

  - Ejemplo: Ejecuta el proceso `firefox`, disminuyendo en 10 el valor por defecto del número `nice` para mejorar la prioridad de este proceso (sólo puede hacerlo el root):

    ```
    nice --10 firefox
    ```

- `renice nueva_prioridad pid`

- Ejemplo: Fija el valor del número `nice` en 14 para el proceso cuyo PID es 890:

```
renice 14 890
```

Por último, indicar que, de cara a completar todo el sistema de prioridades de Linux, este rango de 0 a 39 para procesos de usuario queda mapeado en el rango absoluto de 100 a 139, mientras que los procesos a nivel de núcleo conforman el rango de prioridades de tiempo real, `rt`, desde 0 a 99.

5.5. Envío de señales a procesos

Una señal es un aviso que puede enviar un proceso a otro proceso. Cuando un proceso recibe una señal, se pone inmediatamente a ejecutar la función de tratamiento de esta señal. Cuando acaba el tratamiento, el proceso continúa con su ejecución normal (si la señal no es de terminación). Todas las señales tienen una función de tratamiento por defecto. Sin embargo, un proceso puede decidir capturar una señal, cambiando esta función de tratamiento por otro código que él disponga.

Se pueden enviar señales a los procesos para pararlos, con la señal SIGSTOP (19), para eliminarlos, con la señal SIGKILL (9), para hacer que continúen, con la señal SIGCONT (18), etc. Este envío se realiza mediante la orden `kill`, con la sintaxis:

```
kill [-señal] pid...
```

La señal que se envía por defecto es SIGTERM (15), que ordena al proceso receptor que termine su labor, de forma correcta y controlada. Esta señal puede ser capturada. En cambio, si la señal que se envía es SIGKILL (9) tendremos la seguridad de que el proceso receptor finalizará. Por ejemplo:

```
$ gedit &
[1] 26651

$ ps
PID TTY          TIME CMD
24930 pts/3    00:00:00 bash
26651 pts/3    00:00:00 gedit
26654 pts/3    00:00:00 ps

$ kill -9 26651
[1]+  Killed                 gedit
```

Se puede hacer un envío más genérico usando el nombre del proceso, en lugar de su PID, con la orden `killall`, siguiendo la sintaxis:

```
killall [-señal] orden
```

Por ejemplo:

```
$ gedit&
[1] 26666

$ gedit&
[2] 26667

$ ps
  PID TTY          TIME CMD
24930 pts/3    00:00:00 bash
26666 pts/3    00:00:00 gedit
```

```
26667 pts/3    00:00:00 gedit
26668 pts/3    00:00:00 ps


$ killall -9 gedit
[1]-  Killed                  gedit
[2]+  Killed                  gedit

$ ps
  PID TTY          TIME CMD
24930 pts/3    00:00:00 bash
26670 pts/3    00:00:00 ps
```

Finalmente, habría que indicar que hay procesos que no mueren a pesar de recibir la señal SIGKILL: los procesos zombies (estado `Z`), porque ya están muertos, y los procesos que esperan una petición de E/S hecha a un dispositivo de disco (estado `D`), porque están realizando una llamada al sistema y no pueden ser interrumpidos (o eliminados) hasta que esta llamada se complete, tal como se dijo anteriormente.

5.6. Procesos por lotes

En este apartado se describirá como se puede programar la ejecución de tareas para que se lleve a cabo en un momento concreto o bien con cierta periodicidad.

5.6.1. Programación puntual de tareas

La orden `at` se utiliza para ejecutar tareas a una determinada hora y/o día, pudiendo recibir como parámetro un fichero de texto con las órdenes a ejecutar (mediante la opción `-f`), o bien recibir estas órdenes a ejecutar desde teclado, con un prompt especial (`at>`), hasta finalizar con `Ctrl+D`.

Así, por ejemplo, si queremos que a las 11 de esta noche se guarde en el fichero `procesosnoche.txt` del directorio `/root` el listado de los procesos que se están ejectutando y en el fichero `ficherosnoche.txt` del mismo directorio el listado de los ficheros del directorio `/tmp`, podríamos indicarle línea a línea las tareas a realizar de esta manera:

```
# at 11pm
at> /usr/bin/ps axu > /root/procesosnoche.txt
at> /usr/bin/ls -l /tmp > /root/ficherosnoche.txt
at> <EOT> # Finalizar con Ctrl+D
job 9 at Sun Dec  6 18:40:00 2020
```

O bien, indicarle el fichero a ejecutar, usando la opción `-f`:

```
# cat > /root/tarea.sh
/usr/bin/ps axu > /root/procesosnoche.txt
/usr/bin/ls -l /tmp > /root/ficherosnoche.txt
<EOT>

# at 11pm -f /root/tarea.sh
job 9 at Sun Dec  6 18:45:00 2020
```

Para establecer el momento de ejecución podemos indicar una hora absoluta (como en el ejemplo anterior), o bien una hora relativa, a partir del momento actual. Igualmente, podemos establecer una fecha absoluta o relativa. Veamos algunos ejemplos:

- Para que se ejecute el próximo domingo a las 4 de la tarde:

```
# at 4pm sunday
```

- Para que se ejecute dentro de 3 días, a las 11 de la mañana:

```
# at 11am +3days
```

- Para que se ejecute a las 3 de la madrugada de la nochebuena de 2025:

```
# at 3am 12/25/2025
```

- Para que se ejecute el próximo viernes a la hora actual:

```
# at friday
```

- Para que se ejecute dentro de 5 horas:

```
# at now +5hours
```

El servicio encargado de ejecutar las órdenes es `atd.service`. Es importante tener en cuenta que si este servicio no está activo las tareas no se lanzarán. Para consultar su estado debemos ejecutar `systemctl status atd.service` y para activarlo `systemctl start atd.service`.

La orden `atq` sirve para consultar la lista de órdenes pendientes, mientras que con `atrm` podremos eliminar algunas de estas órdenes.

5.6.2. Programación de tareas periódicas

Para ejecutar tareas periódicamente podemos hacer uso del servicio `crond.service` mediante la orden `crontab` (como ocurría con el servicio `atd.service` visto para la orden anterior, si este servicio no está activo las tareas no se lanzarán. Para consultar su estado debemos ejecutar `systemctl status crond.service` y para activarlo `systemctl start crond.service`). Opciones de la orden `crontab`:

- `-e`: para añadir/modificar tareas. Al ejecutarlo, se abre un editor (normalmente el editor por defecto es el `vim`[2]), donde en cada línea podemos indicar la periodicidad de una tarea siguiendo este formato:

  ```
  minutos  hora  día_mes  mes  día_semana   tarea
  ```

  Por ejemplo:

  ```
  # crontab -e
  05 09      *    * *       tareadiaria     #cada día a las 9:05
  15 14      1    * *       tareames        #primer día del mes, a las 14:15
  00 22      *    * 1-5     tareasemanal    #de lunes a viernes, a las 22:00
  23 0-23/2  *    * *       tareacada2horas #a las 2, 4,..., y 23 minutos
  05 04      *    * sun     tareadomingos   #domingo  a las 4:05
  ```

- `-l`: para listar todas las tareas programadas.

- `-r`: para eliminar todas las tareas programadas.

[2]Para poder cambiar el editor por defecto se puede establecer el valor de las variables de entorno `EDITOR` o `VISUAL` con la ruta al editor deseado. Por ejemplo, para establecer `nano` como nuestro editor por defecto, podemos poner `export EDITOR=/usr/bin/nano` en nuestro fichero `.bash_profile`, tal como vimos en el boletín 5.

Normalmente poner un valor en lugar de un asterisco introduce una restricción. Tomemos para ilustrarlo la segunda línea del ejemplo anterior. Si ponemos 15 minutos, 14 horas y 1 como día del mes, estamos limitando la ejecución al primer día de cada mes a las 14:15. Sin embargo esto no sucede con el día del mes y el día de la semana. En este caso se toma una restricción o la otra, pero no las dos. Por ejemplo, en el caso anterior, si pusiéramos `15 14 1 * sun`, no estaríamos haciendo que se ejecutara los días 1 de cada mes que caigan en domingo, sino que se ejecutaría los días 1 de cada mes, y además los domingos. Si queremos poner más restricciones, tendremos que incluirlas en la orden que se ejecute. Por ejemplo, podemos hacer que se ejecute cada primero de mes, y en la orden comprobar con `date` si es domingo o no.

Otra opción para indicar la periodicidad de ejecución una tarea consiste en sustituir los 5 primeros campos de la línea correspondiente por uno de estos *nicknames*:

- `@reboot` : Se ejecuta justo después de rearrancar.
- `@yearly` : Se ejecuta una vez al año, es decir, sería equivalente a: `0 0 1 1 *`.
- `@monthly` : Se ejecuta una vez al mes, es decir, sería equivalente a: `0 0 1 * *`.
- `@weekly` : Se ejecuta una vez a la semana, es decir, sería equivalente a: `0 0 * * 0`.
- `@daily` : Se ejecuta una vez al día, es decir, sería equivalente a: `0 0 * * *`.
- `@hourly` : Se ejecuta una vez cada hora, es decir, sería equivalente a: `0 * * * *`.

El fichero `/etc/cron.d/0hourly` es el fichero para el servicio `cron` del sistema. Está preparado para ejecutar una serie de tareas cada hora. Para ello, usa el script `run-parts`, que se encarga de ejecutar por orden alfabético todos los ficheros con permiso de ejecución que encuentra en el directorio que se le pasa como parámetro. Las tareas a ejecutar se copian como ficheros ejecutables en el directorio `/etc/cron.hourly/`. De esta manera, el fichero `/etc/cron.d/0hourly` podría tener un formato como este:

```
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/
# run-parts
01 * * * * root run-parts /etc/cron.hourly
```

Si observamos en el directorio `/etc/cron.hourly` las tareas planificadas para ejecutarse cada hora encontramos al ejecutable `anacron`. Se trata de un planificador de tareas periódicas (inicialmente configurado para diarias, semanales o mensuales) que no asume que la máquina tenga que estar encendida continuamente. La configuración de `anacron` está establecida en el fichero `/etc/anacrontab`, por ejemplo, con este contenido:

```
SHELL=/bin/sh
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO:root

#the maximal random delay added to the base delay of the jobs
RANDOM_DELAY=45
#
# the jobs will be started during the following hours only
START_HOURS_RANGE=3-22
#
#period   delay
#in days  in minutes  job-identifier command
#-------  ---------- -------------- --------------------------------
```

```
1         5          cron.daily     nice run-parts /etc/cron.daily
7         25         cron.weekly    nice run-parts /etc/cron.weekly
@monthly 45          cron.monthly   nice run-parts /etc/cron.monthly
```

De esta forma, las tareas que queramos planificar para ejecutarse cada día, semana y mes se deben copiar como ficheros ejecutables en los directorios `/etc/cron.daily/`, `/etc/cron.weekly/` y `/etc/cron.monthly/`, respectivamente. Cada hora se pondrá a funcionar `anacron`, que leerá la lista de trabajos especificados, comprobará si se han ejecutado en el último periodo establecido y en caso negativo los pondrá a funcionar (tras esperar unos minutos de retardo más una variación aleatoria) utilizando el script `run-parts`.

6. Control y gestión de la memoria

La memoria tiene un gran efecto sobre el rendimiento global del sistema, por tanto es necesario que el sistema disponga de una cantidad de memoria adecuada.

La paginación y el intercambio son las técnicas que Linux usa para distribuir la memoria disponible entre los procesos. El intercambio consiste en llevar un proceso a disco para liberar la memoria que está utilizando, posteriormente el proceso volverá a memoria para continuar su ejecución. La paginación implica llevar parte de la memoria de los procesos, en unidades llamadas páginas, a disco para liberar memoria. Estas páginas se traerán de vuelta a memoria cuando se vuelvan a necesitar. Es responsabilidad del administrador la gestión de RAM y de la zona o espacio de intercambio o swap.

6.1. La orden `free`

La orden `free` presenta información sobre la ocupación de la memoria principal y del espacio de swap del que se disponga (Por defecto, toda la información se muestra en KiBs). La información mostrada por esta orden es similar a la que aparece en las últimas 2 líneas de la información mostrada por la orden `top` en la cabecera.

Por ejemplo:

```
# free
              total        used        free      shared  buff/cache   available
Mem:        3715932     2794660      195448      130664      725824      521028
Swap:       3817468      278272     3539196
```

- La memoria: memoria total disponible (`total`) , usada (`used`), libre completamente (`free`), cantidad usada en buffers y en memoria caché de páginas (`buff/cache`). El último valor de esta línea (`available`) es una estimación de la cantidad de memoria disponible que se podría usar, tanto por los procesos existentes como para crear nuevos procesos, por lo que incluye la memoria que no está siendo usada en absoluto (`free`) y otras zonas de memoria que también se pueden utilizar dinámicamente cuando sea oportuno. El campo `shared` puede ser ignorado pues presenta un valor obsoleto que ya no se utiliza.
- El espacio de swap: swap total disponible (`total`), usada (`used`) y libre (`free`).

6.2. La orden `vmstat`

La orden `vmstat` presenta información sobre el uso de la memoria principal, las lecturas/escrituras de/en el espacio de swap, actividad del sistema operativo en cuanto a interrupciones y cambios de proceso, así como uso de la CPU.

La sintaxis de uso de esta orden es:

```
vmstat [intervalo número]
```

siendo:

- `intervalo`: Cada cuántos segundos debe mostrar los datos.
- `número`: Cuántos muestreos se solicitan.

Ejemplo:

```
$ vmstat 5 3
procs ---------memory---------- -swap-- ---io---- --system-- -----cpu------
 r  b swpd   free   buff  cache  si so  bi    bo   in   cs us sy id wa st
 0  0   92 187340 290868 2366352  0  0  48    79    0    0  1  0  99  0  0
 0  0   92 186952 290872 2366352  0  0   0    31 2229 2063  0  0 100  0  0
 0  0   92 186952 290888 2366344  0  0   0    12 2163 2038  0  0 100  0  0
```

Concretamente estos son los campos de los que muestra información relacionada con la gestión de la memoria del sistema:

- Procesos:
  - `r`: Número de procesos ejecutables (en ejecución o listos para ejecutarse) (Estado `R` visto en las órdenes `ps` y `top`).
  - `b`: Número de procesos bloqueados (Estado `D` visto en las órdenes `ps` y `top`).
- Memoria (`-memory-`) (Información semejante a la mostrada en la cabecera de la orden `top` y en la orden `free`)
  - `swpd`: Cantidad empleada del espacio de intercambio (en KiBs).
  - `free`: Cantidad de memoria RAM sin usar (en KiBs).
  - `buff`: Cantidad de memoria RAM empleada como buffers para E/S (en KiBs).
  - `cache`: Cantidad de memoria RAM empleada como caché de páginas (en KiBs).
- Espacio de intercambio (`-swap-`)
  - `si`: Cantidad de memoria traída del espacio de intercambio a memoria (en KiB/s).
  - `so`: Cantidad de memoria intercambiada al disco (en KiB/s).
- Entrada/Salida (`-io-`)
  - `bi`: Cantidad de información recibida desde un dispositivo de bloques (en KiB/s).
  - `bo`: Cantidad de información enviada a un dispositivo de bloques (en KiB/s).

Esta orden mostrará en cada línea valores en el instante actual para la información relativa a la memoria (`-memory-`). Para la información relativa al espacio de intercambio (`-swap-`) y la información relativa a la Entrada/Salida (`-io-`) esta orden muestra la información correspondiente a cada periodo de tiempo establecido, siendo la información que muestra en la primera línea la que corresponde al intervalo de tiempo desde que se arrancó el sistema hasta el momento actual. En caso de que no se le pasen argumentos, se muestra únicamente esta primera línea.

6.3. Espacio para paginación

El tamaño adecuado para la partición de intercambio depende de varios aspectos, como la memoria requerida por los trabajos, el número de trabajos que se están ejecutando simultáneamente, el uso de hibernación en ordenadores portátiles, etc.

Se puede utilizar una partición de intercambio o bien un fichero de paginación. El rendimiento usando un fichero de paginación es menor que con la partición debido a la sobrecarga de la gestión

que se precisa por parte del sistema de ficheros. Además, a la partición no le afectan los problemas de fragmentación que puede tener cualquier fichero individual, incluido un fichero de intercambio.

Se puede asignar valores de prioridad de uso mediante la opción `pri=valor` (números altos significan mayor prioridad). En tiempo de arranque se activan todas las zonas de intercambio o paginación indicadas en el fichero `/etc/fstab`. Así, por ejemplo, podemos activar en el arranque una partición de intercambio y un fichero de intercambio, indicando que se use la partición prioritariamente, si tenemos en el `/etc/fstab`:

```
/dev/mapper/fedora-swap  none    swap    defaults,pri=5    0 0
/.fichero_swap           swap    swap    defaults,pri=3    0 0
```

Con la orden `swapon` se activa una partición de intercambio, mientras que con `swapoff` se desactiva. Una vez desactivada una partición, ésta se podrá usar para otras cosas pues tendremos la garantía de que toda la información de la memoria virtual que maneja cada proceso que está en funcionamiento se encuentra ya definitivamente almacenada en la memoria secundaria asignada a dicho proceso.

La creación de un fichero de intercambio se realiza en varios pasos, como podemos ver en este ejemplo:

```
# dd if=/dev/zero of=/.fichero_swap bs=1048576 count=1024
# mkswap /.fichero_swap
# sync
# swapon /.fichero_swap
```

donde se crea el fichero `.fichero_swap` de 1GiB en el directorio raiz y, a continuación, se configura este fichero como nueva zona de intercambio. Tras ello, se sincroniza la caché de páginas y, finalmente, se activa este fichero de intercambio.

7. Ejercicios

7.1. Administración de paquetes

1. ¿Cuántos paquetes hay instalados en el sistema? (Usa `man` para saber qué opción de búsqueda hay que pasarle a la opción `-q` de la orden `rpm`).

2. Utiliza las opciones apropiadas de la orden `rpm` para:

   a) Averiguar a qué paquete pertenece el fichero de configuración `/etc/passwd`.

   b) Consultar las características de ese paquete y averiguar cuándo fue instalado.

   c) Averiguar qué ficheros pertenecen a ese paquete.

3. Utilizando la orden `dnf` con la opción apropiada, descarga (SIN LLEGAR A INSTALAR) el paquete `fortune-mod`. Comprueba que has descargado dicho paquete en tu directorio `$HOME`.

4. Utilizando la orden `rpm` con la opción apropiada, prueba a instalar el paquete descargado en el apartado anterior. No olvides poner el nombre completo del paquete, incluyendo su ruta absoluta o relativa. ¿Qué ocurre? ¿Por qué?

5. Utilizando la orden `dnf` con la opción apropiada, prueba a instalar (SIN VOLVER A DESCARGARLO) el paquete descargado. No olvides poner el nombre completo del paquete, incluyendo su ruta absoluta o relativa. ¿Qué ocurre? ¿Además del mencionado paquete, se descarga e instala alguno más? ¿Cuál es el tamaño total de los paquetes a instalar? ¿Cuál es el tamaño total de la descarga?¿Coinciden ambos tamaños? ¿Por qué?

6. Comprueba que se ha realizado la instalación del paquete de forma correcta ejecutando la orden `fortune`.

7. Utilizando la orden `dnf` con la opción apropiada, elimina el paquete `recode`. ¿Se elimina algún otro paquete? ¿Por qué?

8. Prueba a ejecutar la orden `fortune` de nuevo. ¿Funciona? ¿Por qué?

7.2. Gestión de recursos

Antes de empezar a resolver estos ejercicios es importante que tengas en cuenta que no debes limitarte a ejecutar las órdenes de manera automática. Lo interesante es analizar los números que muestran, y ver cómo éstos varían cuando producimos cambios en el contexto de ejecución. Es importante, por tanto, que prestes atención a las órdenes que ejecutas, y sobre todo a los datos que se obtienen.

1. Utilizando la orden `ps` con las opciones apropiadas, apoyada con alguna-s orden-es de filtrado mediante tuberías, lista:

   1.1 Los 10 procesos que más porcentaje de CPU están utilizando, ordenados de mayor a menor uso de CPU. Para cada uno de estos procesos, deberá mostrarse su porcentaje de uso de CPU, su nombre, PID, y usuario propietario.

   1.2 Todos los procesos que estén durmiendo (bloqueados). Para cada uno de estos procesos, se debe mostrar su estado, nombre, PID, así como el PID de su proceso padre.

   1.3 Los PIDs de todos los procesos que tengan algún proceso hijo que esté durmiendo (bloqueado).

2. Jerarquía de procesos:

   2.1 Utilizando la orden `pstree`, visualiza y describe la jerarquía familiar que forman los procesos: `gdm`, `gdm-session-worker`, `gdm-wayland-session`.

   2.2 Comprueba ahora con la orden `ps`, con las opciones apropiadas, que los PID y los PPIDs de estos procesos son coherentes con la jerarquía familiar indicada por `pstree` en el apartado anterior.

3. Usando la orden `top`, averigua:

   - ¿Cuánto tiempo lleva en marcha el sistema?
   - ¿Cuántas terminales de usuario hay abiertas?
   - ¿Cuál es la carga media del sistema en los últimos 15 minutos?

4. Crea el fichero `/tmp/bucle` con el siguiente contenido y asígnale permisos de ejecución.

   ```
   #!/bin/bash
   echo 'nada' > /dev/null
   exec /tmp/bucle
   ```

   NOTA: la orden `exec` realiza una llamada al sistema `exec()`. Esta llamada, tal como se ha visto en el tema 2 de teoría, produce la sustitución del código y datos del proceso que la invoca, por los de otro programa (pasado como argumento) y que, a continuación, comience la ejecución del nuevo código desde el principio, sin crear ningún nuevo proceso.

   4.1 Ejecuta la orden `top` en una terminal y comprueba el estado del sistema. Mantén ejecutándose esta orden en esta terminal. A continuación lanza `/tmp/bucle` en otra terminal. Observa en la información mostrada por `top` en la primera terminal, cómo cambia el estado del sistema al lanzar el script.

   4.2 Usa la combinación de teclas `Control-Z` para detener el proceso `bucle`. Una vez detenido, la información mostrada por `top` va cambiando, hasta que no muestra más información sobre dicho proceso. Fíjate que ha aumentado el número de procesos parados.

4.3 Reinicia el proceso con la orden `fg` y comprueba que vuelve a aparecer la información sobre el proceso.

4.4 Observa si mientras está en ejecución ese proceso cambia la carga media del sistema.

4.5 ¿Por qué el proceso `bucle` siempre tiene el mismo PID si se lanza a sí mismo una y otra vez durante su ejecución?

4.6 Desde el `top`, cambia la prioridad base (el número `nice`) del proceso (que está actualmente en valor 0), dándole un valor menor, por ejemplo -19.

4.7 Usando la orden `nice` lanza otro proceso `bucle`, en segundo plano, con la prioridad base (valor del número `nice`) de 19. Teniendo en cuenta las prioridades asignadas, se le debería asignar mucho más porcentaje de CPU al primer proceso que al segundo. ¿Ocurre eso? Muestra y razona el resultado.

4.8 Lanza 6 procesos más como en el apartado anterior, con prioridad 19, todos en segundo plano. Observa, en la información de `top`, el comportamiento en cuanto a porcentaje de uso de CPU para los 8 procesos lanzados y razona la situación.

4.9 Haciendo uso de la orden `kill` o de la orden `killall`, elimina los 8 procesos lanzados.

5. Vamos a generar, ejecutar y monitorizar un proceso multihilo:

   - Descarga el programa `generando_3_hilos.c` del Aula Virtual de la asignatura.
   - Observa en su código fuente cómo esta previsto que al ejecutar este programa se genere un proceso con 3 threads que funcionen indefinidamente.
   - En caso de que no lo tengas ya instalado en tu máquina, descarga e instala el paquete que contiene el compilador que vamos a usar:

     ```
     # dnf install gcc
     ```

   - Compila este programa:

     ```
     # gcc -o generando_3_hilos generando_3_hilos.c -lpthread
     ```

   - Pon el programa a funcionar en segundo plano:

     ```
     # generando_3_hilos &
     ```

   5.1 Comprueba con la orden `ps` que el proceso correspondiente al programa `generando_3_hilos` se ha puesto a funcionar y anota su PID.

   5.2 Comprueba, usando la orden `pstree`, los procesos que son los ancestros de este proceso puesto en marcha. Observa igualmente los threads que produce dicho proceso: el thread principal del proceso y los threads hijos, que aparecen a continuación encerrados entre {}.

   5.3 Ejecuta la orden `top`. Para el proceso `generando_3_hilos`, anota los valores de los campos PID, VIRT, RES, %CPU y TIME+. Explica su significado.

   5.4 Observa que, en una máquina con X cores, por cada segundo de tiempo real transcurrido, el valor de TIME+ de este proceso, que cuenta con 3 threads, puede llegar a aumentar aproximadamente:

   - X segundos, si $X \leq 3$.
   - 3 segundos, si $X \geq 3$.

   ¿Por qué crees que ocurre esto?

   5.5 Desde `top` ejecuta la orden `H` para ver al detalle todos los threads de cada proceso en funcionamiento. Para todos los threads generados por el proceso `generando_3_hilos`, anota los valores de los campos PID, VIRT, RES, %CPU y TIME+. Compara estos valores con los obtenidos en el ejercicio anterior para el proceso en su conjunto y razona tu respuesta.

5.6 Siguiendo en `top`, ejecuta ahora la orden `1`. Observa y anota la información que ahora te ofrece esta orden sobre el uso de los diferentes cores de la CPU.

5.7 Ejecuta la orden `ps Ho pid,nlwp,lwp,time,stime,comm`. Anota, para todos los threads que genera el proceso `generando_3_hilos`, los valores de los campos denominados PID, NLWP, LWP, TIME y STIME. Razona apropiadamente los valores obtenidos para cada campo.

5.8 Finaliza el proceso `generando_3_hilos` usando la orden `kill`.

6. Órdenes: `at`, `atq`, `atrm`.

   6.1 Comprueba si el servicio `atd` está en ejecución (`systemctl status atd.service`). Si este servicio no está en ejecución, ponlo a funcionar (`systemctl start atd.service`).

   6.2 Programa con la orden `at`:

   - La ejecución de `ps axo command,pid,user >$HOME/listado_procesos.txt` para dentro de 3 minutos.
   - La ejecución de `ls /tmp >$HOME/listado_tmp.txt` para dentro de 5 minutos.

   Pulsa en cada caso `CTRL+D` para finalizar la redacción de la orden programada para `atd`.

   6.3 Comprueba con la orden `atq` que tareas hay programadas en la cola del servicio `atd`.

   6.4 Comprueba que, pasado el tiempo oportuno, las tareas pendientes van abandonando la cola y se van ejecutando. Comprueba los ficheros donde se vuelcan las salidas de dichas tareas.

7. Comprueba si el servicio de impresión está activo (`systemctl status cups.service`). En caso de que esté activo, páralo (`systemctl stop cups.service`). Si tomamos X como el momento actual, programa la herramienta `crontab` para que este servicio se active (`systemctl start cups.service`) de lunes a viernes a las $X + 5$ minutos (Por ejemplo, si ahora mismo fuesen las 11:35, la hora a fijar sería las 11:40). De igual forma, configura `crontab` para que el servicio de impresión se detenga de lunes a viernes a las $X + 10$ minutos.

8. Comprueba la correcta configuración de la programación de las tareas anteriores: Pasado el tiempo oportuno, comprueba la activación y luego la desactivación del servicio de impresión, siguiendo el horario establecido con `crontab`.

9. Con la orden `free` averigua el estado de la memoria principal y de la zona de intercambio del sistema en este momento. Anota el tamaño total de la zona de intercambio.

10. Ejecuta la orden `vmstat` obteniendo 4 muestras cada 2 segundos. Anota, de la última muestra, la información mostrada sobre la cantidad de memoria de intercambio usada.

11. Crea un fichero de paginación llamado `.swap1` de 1 GiB en el directorio raiz y añádelo a tu sistema.

12. ¿Cambian los valores mostrados por `vmstat`? ¿Y los mostrados por `free`? ¿Por qué?

13. ¿Qué habría que hacer para que el fichero se active en el momento del arranque del sistema?