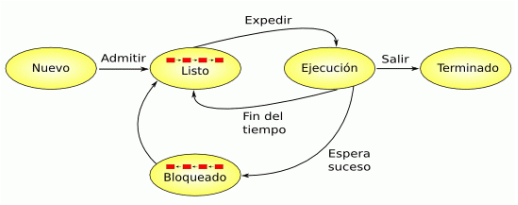
7.Gestión de procesos y servicios



7.1.- Procesos

Un proceso es una instancia en ejecución de un programa. Un programa consiste simplemente en un archivo ejecutable que contiene unas líneas de código; es algo estático. Un proceso es algo dinámico, el código en ejecución. Pueden existir varios procesos que ejecuten el mismo programa.

Un proceso consiste en:

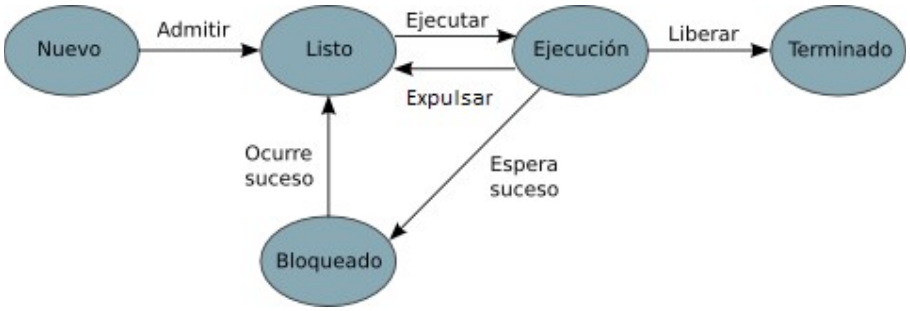
- un espacio de direcciones en memoria y propiedades de seguridad.
- uno o más hilos de ejecución de código de programa.
- estado en que encuentra el proceso: ejecutando, esperando por un recurso, parado, zombie, ...
- entorno: variables globales y locales, descriptores de archivo y puerto, …
- PID (**P**rocess **Id**entifier): número de proceso que se asigna al proceso cuando se inicia y PPID: número de proceso del proceso que lo inició.

Un proceso padre existente en el sistema, duplica su propio espacio de direcciones para crear una nueva estructura de proceso. El primer proceso que se inicia en un sistema CentOS7 es el proceso *systemd*, con PID 1. Se encarga de iniciar el resto de procesos, que serán procesos hijos de *systemd*. En versiones anteriores, p.e., CentOS6, el primer proceso del sistema era el proceso *init.*

Los sistemas Linux son multitarea y mutiusuario, pueden ejecutar de forma simultánea varias tareas de distintos usuarios. Mantienen en memoria dichas tareas y, mientras a uno de ellos se le asigna tiempo de CPU, el resto esperan por algún evento, siendo programada su entrada en la CPU por el kernel, que es el que decide qué tarea es la que tiene que ejecutarse en un momento dado, dependiendo de su prioridad y uso de recursos.

7.1.1.- Estados de un proceso

En un sistema multitarea como es Linux, cada CPU o cada core de la CPU en un momento dado, puede tener ejecutándose un proceso. Un proceso pasará por varios estados dependiendo de si está ejecutando código en la CPU, si está esperando para ejecutar, esperando por operaciones entrada/salida, etc.



Estados de un proceso

Los estados posibles de un proceso, con su flag y su nombre entre paréntesis, son:

- *R (running):* ejecutándose en la CPU o esperando a ser ejecutado.
- *S (sleeping):* espera interrumpible, ejecutando pero sin actividad, esperando a que suceda un evento, una vez que el evento sucede, vuelve a el estado de *running*.
- *D (sleeping):* espera no interrumpible, esperando por algún proceso de entrada/salida.
- *Z (zombie):* el proceso ha terminado pero el proceso padre no está esperando por su finalización, el kernel sigue manteniendo información del proceso aunque ya ha terminado. No es recomendable tener procesos zombis de larga duración en el sistema, indica un problema.
- *T (stopped)*: proceso detenido que puede ser reiniciado.

7.1.2.- Señales para los procesos

A los procesos se les manda señales para modificar su comportamiento a través del kernel. Esto se realiza mediante el comando *kill* o *pkill* con sus correspondientes señales e indicando el número de PID. Algunas de las señales que se pueden mandar a los procesos:

- **SIGHUP, 1:** para y reinicia un proceso. Si la entrada estándar es un terminal, se redirige a */dev/null* y si la salida estándar es también un terminal, se redirige a el archivo *nohup.out* del directorio actual o, si el usuario no tiene permisos para escribir en él, de su *home*.
- **SIGINT, -2**: interrumpe un proceso, equivale a pulsar *CTRL+C*.
- **SIGKILL, -9**: mata un proceso.
- **SIGTERM, -15**: señal por defecto, como *SIGINT* pero de una forma más ordenada.
- **SIGCONT, -18**: reanuda un proceso que se ha parado, p.e. con *SIGSTOP*.
- **SIGSTOP, -19**: parar un proceso, es igual que pulsar **CNTRL+Z**.

Hay algunos casos que los procesos tienen inhibidas ciertas señales, si es así, no responderán al envío de esas señales con *kill* o *pkill*. En ese caso, la única señal no inhibida es *SIGKILL*.

La lista completa de señales se puede ver con *kill -l*.

7.2.- Trabajos

Un trabajo o *job* está asociado con cada pipe o programa que se ejecute desde la shell, el número de *job* es distinto al número de proceso.

Utilizaremos el comando *jobs* para ver las tareas asociadas al terminal que estemos usando. Muestra la lista de tareas en segundo plano o suspendidas con *CTRL+Z*. Y con los comandos *fg* y *bg* podremos enviar tareas a primer o segundo plano respectivamente.

Para enviar un proceso a segundo plano, lo lanzaremos poniendo al final del comando con el que lo lancemos el carácter &. La shell mostrará el número de job entre corchetes y a continuación el número de PID.

Si lanzamos varios comandos con un pipe, cada uno tendrá un PID pero todos tendrán el mismo número de job. El PID que se muestra entonces al añadir & es el del último comando de la línea.

Un proceso en segundo plano no puede leer del terminal ni recibir nada por el teclado, pero si puede mostrar salida en el terminal.

Cuando cerramos un terminal, la Shell nos avisa si tenemos tareas suspendidas o ejecutandose, si aun así cerramos el terminal, todos las tareas se finalizan.

7.3.- Comandos para gestionar procesos y trabajos

→ Comando ps

Es el comando principal para ver la información y estado de los procesos. Se basa en el sistema de archivos */proc;* lee directamente la información de ese directorio.

Soporta 3 tipos de opciones: UNIX-POSIX, precedidas por guión, BSD, sin guión y GNU, con doble guión y nombre de opción largo. Las columnas que suelen mostrar las opciones, sean del tipo que sean, son:

- *PID*: número de identificador del proceso.
- *PPID*: número de identificador del proceso padre que lo lanzó.
- *TTY*: terminal desde donde se lanzó el proceso. Si aparece ?, no tiene terminal asociado.
- *TIME*: tiempo de CPU utilizado por el proceso.
- *CMD*: comando con el que se lanzó el proceso.
- *NI*: valor de *nice*, de prioridad del proceso.
- *PCPU*: porcentaje de utilización de CPU por parte del proceso.
- *STAT*: estado del proceso.

Si no añadimos ningún parámetro, *ps* mostrará los procesos del usuario con el que hemos hecho *login*; si lo ejecutamos como usuario *root*, se puede añadir la opción *-u <usuario>* para ver los procesos de ese usuario en concreto.

Los parámetros más comunes son:

-a: lista los procesos de todos los usuarios.

-u: lista información del proceso como por ejemplo el usuario que lo está ejecutando, la utilización de CPU y memoria, etc.

-x: lista procesos de todas las terminales y usuarios.

-l: muestra información que incluye el UID y el valor *nice*.

--forest: muestra el listado procesos con un formato árbol que permite ver como los procesos están relacionados entre sí; similar al comando *pstree*.

-aux: lista los procesos de todos los usuarios con información añadida.

→ Comando pstree

Muestra los procesos en forma de árbol. Con las opciones:

- -A y -G*:* muestra el árbol con con líneas estilo ASCII y de terminal VT100 respectivamente.
- *-u:* muestra entre paréntesis al usuario propietario del proceso.
- *-p:* muestra los PID de los procesos.

Si pasamos como argumento un usuario o un PID, muestra los procesos de ese usuario o de ese PID.

→ Comando kill

Envía una señal a un proceso o a varios. La señal por defecto que se envía, si no se indica otra, es la de terminar (15, *SIGTERM*). La sintaxis del comando es:

```
kill -señal <pid_proceso>
```

se pueden indicar varios PID separados por espacios. La señal se puede indicar con su código o su número.

Se puede utilizar la señal *SIGKILL* (-9) para forzar la terminación del proceso pero hay que tener cuidado ya que finaliza el proceso sin terminar las peticiones pendientes, cerrar los descriptores de archivo en uso, etc.

En lugar del PID del proceso, se le puede pasar a *kill* como argumento el número de job, utilizando el carácter % delante de dicho número.

→ Comando killall

Funciona de forma similar a *kill*, pero en lugar de indicar el PID del proceso, se indica el nombre del comando con el que se lanzó el/los proceso/s, con lo que afectará a cualquier proceso lanzado con dicho comando. Su sintaxix es:

killall -<señal> <comando>

al igual que con *kill*, si no se indica la señal a enviar, se envía la señal *SIGTERM* y las señales se pueden indicar con su número o su nombre.

Si se ha realizado una instalación básica de CentOS, puede no venir instalado, entonces instalaremos el paquete *psmisc*.

→ Comando pgrep

Muestra los procesos que encajan con un patrón dado. Podría definirse como una simbiosis entre los comandos **ps** y **grep**. Sintaxis:

pgrep <patrón>

Las opciones más comunes:

-*U <usuario>*, -*u <usuario_efectivo>:* muestra los procesos cuyo usuario propietario sea el uid o el uid efectivo dado y encaje con el patrón de comando.

-*G <grupo>*, -*g <grupo_efectivo>:* muestra los procesos cuyo grupo propietario sea el *gid* o *gid* efectivo dado y encaje con el patrón de comando.

-*l:* muestra la lista de los procesos con nombre y PID.

-*t <terminal>:* muestra los procesos de ese terminal.

→ Comando pkill

Similar a *pgrep* pero en lugar de mostrar los procesos que encajan con el patŕon de comando dado, les envía una señal dada con -*<señal>* o si no se indica, la señal *SIGTERM*.

→ Comando jobs

Muestra las tareas actuales en ejecución del terminal que estamos usando. Aparece una línea por tarea con el número de *job* entre corchetes a continuación el estado de la tarea y por último el comando usado para lanzar dicha tarea. Si aparece el carácter + después del número de *job*, indica que es la última tarea enviada a segundo plano. Opciones:

-*l*: lista además los PID de los procesos.

-*n*: sólo muestra las tareas que se han detenido o cerrado desde la última notificación.

-*r*: muestra las tareas con estado *running*.

-*s*: muestra las tareas con estado *stopped*.

→ Comando fg

Trae a primer plano, a ejecutarse de nuevo en el terminal actual, una tarea que está en segundo plano. Hay que indicar el número de tarea anteponiendo el carácter %. Para enviar un proceso en primer plano a segundo plano y pararlo, se utiliza CTRL+Z.

→ Comando bg

Manda a segundo plano una tarea que se está ejecutando en primer plano o continúa un proceso en segundo plano que estuviese parado. Hay que indicar el número de tarea anteponiendo el carácter %.

→ Comando nice

Permite iniciar un proceso con una prioridad determinada. Por defecto todos los procesos tienen igual prioridad para acceder a la CPU, con valor de 0. El kernel de Linux se encarga de gestionar el acceso a la CPU para los procesos. El valor de la prioridad o valor nice, va de -20, la más alta, a 19, la más baja. La sintaxis es:

```
nice -n <prioridad> <comando>
```

Sólo como usuario root podremos asignar prioridades negativas a un proceso, lo que significa que con un usuario normal, sólo podremos lanzar procesos con el comando *nice* con menor prioridad de la que se establecería por defecto.

→ Comando renice

Permite cambiar la prioridad establecida en un proceso cuando se lanzó sin necesidad de detenerlo. La sintaxis es:

```
renice <prioridad> <comando>
```

Al igual que con *nice*, sólo *root* podrá hacer más prioritario un proceso, los usuarios regulares sólo podrán hacer sus procesos menos prioritarios.

→ Comando top

El comando TOP muestra a tiempo real un listado de los procesos que se están ejecutando en el sistema, especificando el porcentaje de CPU y Memoria que están utilizando, sus IDs, usuarios que lo están ejecutando, etc.

La salida por pantalla puede dividirse en dos partes, la cabecera que muestra entre otras cosas, el *uptime* del servidor, número de usuarios conectados y la media de carga del sistema. En la siguiente línea podemos ver el número de procesos en ejecución, así como el uso de disco, memoria y CPUs.

A continuación aparece la lista de procesos que se pueden ordenar por uso de CPU, memoria, etc.

Para cada uno de ellos, muestra el PID, usuario que lo ejecuta, porcentaje de CPU y memoria que consume, comando que ejecuta o tiempo de ejecución del proceso entre otros.

Opciones comunes:

- *-c*: visualiza la línea de comandos completa de cada proceso, activándolo mostrará las rutas completas, mientras que desactivándolo solo muestra el nombre del programa.
- *-d <segundos>*: intervalo de actualización y refresco en segundos que determina cada cuanto se actualiza la información.
- *-U <uid>*: monitorizará solamente los procesos de un determinado UID.
- *-p <pid>*: monitorizará solamente los ID de procesos especificados.
- *-n <número>*: se especifica el nº de veces que actualizará hasta que finalice la ejecución de top.

7.4.- Caso práctico

Vamos a lanzar en **server1** varias veces unos comandos que escribirán un texto en el mismo archivo:

(while true;do echo -n "<texto>" >> ~/jobs.txt;sleep1;done).

Lo lanzaremos dos veces utilizando los textos de : “hola ” y “adiós ” dejando ambos en segundo plano en ejecución y luego, tras llevarlos a primer plano, los cancelamos.

Se lanzará también en **server1**, en segundo plano el comando *md5sum* del archivo */dev/zero,* se cancelará, se volverá a lanzar con un valor de *nice* de 5 y se modificará el valor de *nice* a 10.

RESOLUCIÓN

- Lanzamos la escritura de “hola ”, lo suspendemos con CTRL+Z y lanzamos la escritura de “adiós”:

```
[admin@server1 ~]$ (while true;do echo -n "hola " >> ~/jobs.txt;sleep 1;done)
^Z
[1]+  Stopped                 ( while true; do echo -n "hola " >> ~/jobs.txt; sleep 1; done)
[admin@server1 ~]$ (while true;do echo -n "adios " >> ~/jobs.txt;sleep 1;done) &
[2] 1777
[admin@server1 ~]$ jobs
[1]+  Stopped                 ( while true; do echo -n "hola " >> ~/jobs.txt; sleep 1; done )
[2]-  Running                 ( while true; do  echo -n "adios " >> ~/jobs.txt; sleep 1; done ) &
[admin@server1 ~]$ bg %1
[1]+ ( while true; do echo -n "hola " >> ~/jobs.txt; sleep 1; done ) &
[admin@server1 ~]$ jobs
```

```
[1]-  Running                 ( while true; do    echo -n "hola " >> ~/jobs.txt; sleep 1; done ) &
[2]+  Running                 ( while true; do     echo -n "adios " >> ~/jobs.txt; sleep 1;done ) &
[admin@server1 ~]$ fg %1
( while true; do  echo -n "hola " >> ~/jobs.txt; sleep 1; done )
^C
[admin@server1 ~]$ fg %2
( while true; do  echo -n "adios " >> ~/jobs.txt; sleep 1; done )
^C
```

- Lanzamos el comando *md5sum /dev/zero* en segundo plano y lo cancelo:

```
[admin@server1 ~]$ md5sum /dev/zero &
[1] 2429
[admin@server1 ~]$ jobs
[1]+  Running                 md5sum /dev/zero &
[admin@server1 ~]$ pgrep -l md5
2429 md5sum
[admin@server1 ~]$ pkill md5
[admin@server1 ~]$ pgrep -l md5
[1]+  Terminated              md5sum /dev/zero
```

- Volvemos a lanzar el comando *md5sum /dev/zero* con un valor de *nice* de 5 y lo modificamos a 10:

```
[admin@server1 ~]$ nice -5 md5sum /dev/zero&
[1] 2488
[admin@server1 ~]$  ps -opid,pcpu,nice,comm
  PID %CPU  NI COMMAND
 1747  0.0   0 bash
 2488 99.7   5 md5sum
 2495  0.0   0 ps
[admin@server1 ~]$ renice -n 10 2488
2488 (process ID) old priority 5, new priority 10
[admin@server1 ~]$ ps -opid,pcpu,nice,comm
  PID %CPU  NI COMMAND
 1747  0.0   0 bash
 2488 99.8  10 md5sum
 2498  0.0   0 ps
[admin@server1 ~]$ top
top - 13:49:38 up  2:41,  2 users,  load average: 0.57, 0.82, 0.64
```

```
Tasks: 113 total,   2 running, 111 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  6.0 sy, 94.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  1016536 total,   710632 free,   134160 used,   171744 buff/cache
KiB Swap:  1048572 total,  1048572 free,        0 used.   714432 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
 2488 admin     30  10  107912    604    516 R 99.9  0.1   0:19.58 md5sum
 2507 admin     20   0  157680   2092   1452 R  0.3  0.2   0:00.01 top
    1 root      20   0  128096   6712   3960 S  0.0  0.7   0:01.53 systemd
    2 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kthreadd
    3 root      20   0       0      0      0 S  0.0  0.0   0:00.05 ksoftirqd/0
    6 root      20   0       0      0      0 S  0.0  0.0   0:00.03 kworker/u2:0
    7 root      rt   0       0      0      0 S  0.0  0.0   0:00.00 migration/0
    8 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcu_bh
….
[admin@server1 ~]$ kill 2488
[admin@server1 ~]$ ps -opid,pcpu,nice,comm
  PID %CPU  NI COMMAND
 1747  0.0   0 bash
 2501  0.0   0 ps
          [1]+  Terminated              nice -5 md5sum /dev/zero
```

7.5.- Servicios

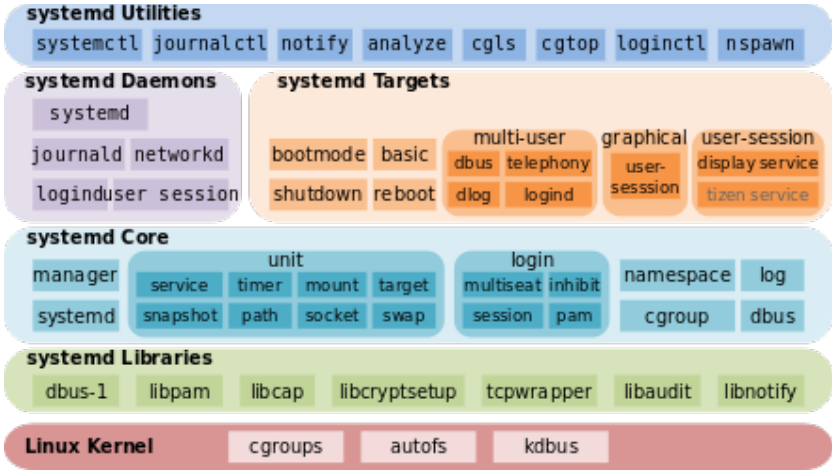
7.5.1.- Introducción a SystemD

SystemD es una nueva forma de inicio del kernel de Linux que proporciona la forma de administrar recursos, demonios de servicios y otros procesos tanto en el arranque del sistema como durante su ejecución.

Además del demonio de *systemd*, incluye a los demonios de *journald*, *logind* y *networkd* junto con otros componentes de bajo nivel.

Creado por Lennart Poethering y Red Hat a partir de 2015 ha sido el sustituto más moderno de init y LSB, reemplazado en la mayoría de las distribuciones Linux a *sysvinit*, *upstart* y *udev* siendo compatible con ellos.

Se ha comenzado a usar en las distribuciones de Linux a partir de estas versiones: CentOS 7, RHEL 7, Fedora 15, ...



Componentes de Systemd

Tiene las mejoras frente a otros inicios del kernel de:

- Gestionar de forma automática la dependencia entre servicios, p.e., no arrancará un servicio que necesite de la red hasta que esta no esté disponible.
- Crea sockets y puntos de montaje necesarios para los servicios.
- Utiliza *cgroups* para realizar el seguimiento de los procesos del servicio en lugar de PIDs impidiendo así que los servicios puedan evadir la administración de *systemd* y optimizando el uso de recursos.
- Permite lanzar en paralelo servicios incrementando la velocidad de arranque de un sistema y arrancar servicios bajo demanda.

7.5.2.- Unidades de SystemD

SystemD trabaja con unidades. Una unidad es un archivo que contiene informacion acerca de un servicio, un *socket*, un punto de montaje/automontaje, o un punto de inicio de una aplicación lanzada por *systemd*.

Podemos ver los tipos de las unidades disponibles en el sistema con *systemctl -t help* y las unidades en ejecución en el sistema y su estado con *systemctl list-units*.

Los tipos de unidades más habituales son:

- *Servicios*: con extensión .service, representan a los servicios del sistema.
- *Sockets*: con extensión .socket, para comunicar procesos. El control del *socket* se pasa al demonio o servicio cuando el cliente realiza la conexión.
- *Paths*: con extensión .path, se usan para retrasar la activación de un servicio hasta que un cambio en un sistema de archivos ocurra.

7.5.3.- Estado de un servicio

El estado de un servicio se puede ver con el comando *systemctl status <nombre>.service*, podemos omitir el *.service* ya que es el valor por defecto si no se proporciona. Si añadimos la opción *-l*, veremos más detalle.

Para ver el estado de todos los servicios del sistema, utilizaremos *systemctl --type-service*.
Los posibles estados que nos podemos encontrar son:

- *loaded*: el archivo de configuración de la unidad ha sido procesado.
- *active (running)*: en ejecución con uno o más procesos continuados.
- *active (exited)*: configuración bien completada.
- *active (waiting)*: en ejecución pero esperando por un evento.
- *inactive*: no ejecutándose.
- *enabled*: Se iniciará en el arranque del sistema.
- *disabled*: no se iniciará en el arranque del sistema.
- *static*: no se puede activar pero se arrancará automáticamente por una unidad activa.

Podemos ejecutar *systemctl is-active <servicio>* y *systemctl is-enabled <servicio>* para ver si el servicio está en ejecución y activado en el arranque del sistema respectivamente.
Y con s*ystemctl list-unit-files --type=service* nos muestra las configuraciones de activo o desactivado para todas las unidades.

7.5.4.- Gestionar servicios

Se utiliza el comando *systemctl <accion> <servicio>* donde *<acción>*:

- *status:* ver el estado del servicio.
- *start*: para arrancar el servicio.
- *stop*: para parar el servicio.
- *enable*: activar el arranque del servicio cuando el sistema se inicie.
- *disable*: desactivar el arranque del servicio cuando el sistema se inicie.
- *restart*: reiniciar el servicio.
- *reload*: se leen los archivos de configuración del servicio y se aplica la configuración de dichos archivos al servicio.
- *mask*: evita que el servicio se pueda arrancar de forma manual o automática en el inicio del sistema.

- *umask*: hace que un servicio enmascarado *con* mask vuelva a estar disponible.
- *list-dependencies*: lista las unidades requeridas por el servicio.

También se puede ver los servicios que están fallado con *systemctl --failed*.

7.6.- Gestionando el arranque del sistema

7.6.1.- Parar y arrancar el sistema

Se usa el comando *systemctl* para gestionar el arranque y parada del sistema con la sintaxis:

```
systemctl <acción>
```

donde <*acción*> en este caso será:

- *poweroff* y *halt*: para todos los servicios en ejecución de forma ordenada, desmonta los sistemas de archivos y apaga el sistema.
- *reboot*: igual que *poweroff* pero a continuación arranca el sistema de cero.
- *suspend*: hiberna el sistema.

Los comandos *poweroff, halt, reboot* y *suspend* ahora son enlaces simbólicos al comando *systemctl* con la acción correspondiente.

7.6.2.- Seleccionar un target de SystemD

Un *target* es un conjunto de unidades de SystemD que se deben de arrancar para alcanzar un estado del sistema. Se pueden ver todos los *targets* disponibles del sistema con:

```
systemctl list-units --type=target  --all.
```

Los *targets* relevantes, los que se pueden seleccionar, son:

- *graphical.target*: sistema multiusuario con interfaz gráfico y *logins* basados en texto.
- *multi-user.target*: sistema multiusuario con *logins* basados en texto.
- *rescue.target*: la inicialización básica del sistema se ha completado y aparece un *prompt sulogin*. Para tareas de mantenimiento.
- *emergency.target*: se completó parte de la inicialización básica del sistema, el / está montado en /sysroot en sólo lectura. Cuando hay un problema en el arranque del sistema, es el target con el que arranca para que solucionemos dichos problemas.

Unos *targets* pueden formar parte de otros, por ejemplo, el g*raphical.target* incluye *multi-user.target* que a su vez depende de *basic.target*. Las dependencias entre los diferentes *targets* se puede ver con:

```
systemctl list-dependencies <target> | grep target
```

Para seleccionar un *target* en *runtime*:

systemctl isolate <target>

No todos los *targets* son seleccionables de esta forma, sólo los que tienen el parámetro *AllowIsolate=yes* en sus archivos de unit.

Para seleccionar un *target* para el arranque del sistema:

systemctl set-default <target>

y para ver el que hay ahora por defecto seleccionado:

systemctl get-defaut

Existe un enlace simbólico en */etc/systemd/system* llamado *default.target* que apunta al *target* seleccionado por defecto.

Cuando un sistema se inicia, se puede elegir el *target* con el que se quiere arrancar interrumpiendo el arranque y añadiendo en los parámetros de inicio, en la línea del kernel: *systemd.unit=<target>* y pulsar *Ctrl+X* para que continue con el proceso.

7.7.- Equivalencia entre run-levels y targets

En SystemV se utilizaban los *run-levels* y ahora en SystemD los *targets*, una posible equivalencia entre ambos sería:

SystemV	SystemD	Uso
0	runlevel0.target	Apaga el sistema
	poweroff.target	
S ó 1	runlevel1.target	Nivel monousuario
	rescue.target	
2,3,4	runlevel2.target, runlevel3.target, runlevel4.target	Nivel multiusuario sin servidor de video
	multiuser.target	
5	runlevel5.target	Nivel multiusuario con servidor de video
	graphical.target	
6	runlevel6.target	Reinicia el sistema
	reboot.target	
emergency	emergency.target	Intérprete de comandos de emergencia

7.8.- Proceso para recuperar la contraseña de root

Sin utilizar un disco de inicio, se puede poner otra contraseña de root siguiendo los pasos:

1. Reiniciar el sistema: *systemctl reboot*.
2. Interrumpir el arranque pulsando cualquier tecla.
3. Editar la línea del kernel (comienza con *linux16*) pulsando *e* y añadir al final *rd.break*.
4. CTRL+X para que continue con el arraque.
5. Montar el sistema de archivos */sysroot* como lectura y escritura: *mount -oremount,rw /sysroot*
6. Hacer: *chroot /sysroot*
7. Poner una nueva contraseña a *root*: *passwd root* o mejor para asergurarnos de cuál estamos escribiendo con: *echo "<password>" | passwd root --stdin*.
8. Crear el archivo *.autorelabel* en el / para que se re-etiqueten todos los archivos con SELinux: *touch /.autorelabel*
9. Teclear exit y pulsar INTRO dos veces. El sistema ya arranca normal con la nueva contraseña de root.

7.9.- Caso práctico

En **server1**, se instalará la utilidad *psacct* que permite monitorizar procesos en CentOS. Se arrancará y activará el servicio *psacct, y* tras pararlo y desactivarlo, hay que enmascararlo.

Ver cuál es el *target* por defecto que tiene el sistema **server1** configurado.

RESOLUCIÓN

- Instalamos, arrancamos, activamos el servicio *psacct*:

[root@server1 ~]# yum install -y psacct

[root@server1 ~]# systemctl status psacct

● psacct.service - Kernel process accounting
Loaded: loaded (/usr/lib/systemd/system/psacct.service; disabled; vendor preset: disabled)
Active: inactive (dead)

[root@server1 ~]# systemctl start psacct

[root@server1 ~]# systemctl enable psacct

Created symlink from /etc/systemd/system/multi-user.target.wants/psacct.service to /usr/lib/systemd/system/psacct.service.

[root@server1 ~]# systemctl status -l psacct

● psacct.service - Kernel process accounting
Loaded: loaded (/usr/lib/systemd/system/psacct.service; enabled; vendor preset:

```
disabled)
   Active: active (exited) since Sun 2017-09-17 14:34:46 CEST; 38s ago
 Main PID: 2770 (code=exited, status=0/SUCCESS)

Sep 17 14:34:45 server1.miempresa.com systemd[1]: Starting Kernel process accounting...
Sep 17 14:34:46 server1.miempresa.com accton[2770]: Turning on process accounting, file
set to '/var/account/pacct'.
Sep 17 14:34:46 server1.miempresa.com systemd[1]: Started Kernel process accounting
```

- Paramos el servicio *psacct*, lo desactivamos y enmascaramos:

```
[root@server1 ~]# systemctl stop  psacct
[root@server1 ~]# systemctl disable psacct
Removed symlink /etc/systemd/system/multi-user.target.wants/psacct.service.
[root@server1 ~]# systemctl mask psacct
Created symlink from /etc/systemd/system/psacct.service to /dev/null.
```

- Verifico que ahora con el servicio enmascarado, no puedo arrancarlo:

```
[root@server1 ~]# systemctl start psacct
Failed to start psacct.service: Unit is masked.
```

- Vemos cuál es el *target* por defecto:

```
[root@server1 ~]# systemctl get-default
multi-user.target
```