1 Administración de procesos en Linux

1.1 Sumario

- 1 Introdución aos procesos en Linux
  - 1.1 PID e PPID
  - 1.2 O proceso de arranque
- 2 **Upstart** A distribución Ubuntu utiliza *Upstart*, que xestiona as prioridades, os eventos e as dependencias entre os servizos, pero respecta o mesmo principio que *init*.
  - 2.1 Os modos de execución e o proceso de inicialización
- 3 **Importante:** Nunca se deben empregar os modos 0 e 6 como modos de execución por defecto. Se o facemos, o sistema se apagará de inmediato ou se reiniciará unha vez que se termina de por en funcionamento. Poderíase empregar o modo de execución 1 como opción por defecto, pero o máis normal é que se empregue 2, 3 ou 5, dependendo da distribución e o uso que se lle vaia a dar ao sistema.
- 4 **Ubuntu:** Por defecto, esta distribución non ten un ficheiro */etc/inittab*. Se se desexa cambiar o seu modo de execución pódese crear un ficheiro */etc/inittab* que só conteña a liña *initdefault*.
  - 4.1 systemd
- 5 Identificación dos procesos en Linux
  - 5.1 ps
  - 5.2 pstree
  - 5.3 jobs
  - 5.4 top
- 6 Matar un proceso
  - 6.1 kill
  - 6.2 pkill
  - 6.3 killall
  - 6.4 xkill
- 7 Pausar un proceso
- 8 Procesos en primeiro e segundo plano: *fg* e *bg*
  - 8.1 fg
  - 8.2 bg
- 9 Utilidades de manexo de servizos ao inicio
  - 9.1 chkconfig
  - 9.2 update-rc.d
- 10 Outros comandos
  - 10.1 netstat e lsof
  - 10.2 Comando nohup

1.2 Introdución aos procesos en Linux

A definición mais simple dun proceso podería ser que é unha instancia dun programa en execución.
Aos procesos frecuentemente se lles chama "tarefas".
Así, ao **contexto** dun programa que está en execución é o que se chama proceso. Este **contexto** pode ter: máis **procesos fillos** xerados do principal (**proceso pai**), os recursos do sistema que estea consumindo, os seus atributos de seguridade (tales como o seu propietario e permisos de arquivos), etc.

Linux, como se sabe, é un sistema operativo multitarefa e multiusuario. Isto quere dicir que múltiples procesos poden operar simultaneamente sen interferirse uns cós outros. Cada proceso ten a "impresión" de que é o único proceso e que ten acceso exclusivo a todos os servizos do sistema operativo.

Programas e procesos son entidades distintas. Nun sistema operativo multitarefa, múltiples instancias dun programa poden executarse simultaneamente. Cada instancia é un proceso separado. Por exemplo, se cinco usuarios dende equipos diferentes, executan o mesmo programa ao mesmo tempo, habería cinco instancias do mesmo programa, é dicir, cinco procesos distintos.

1.2.1 PID e PPID

Cada proceso que se inicia é referenciado cun número de identificación único coñecido como ***Process* ID PID**, que é sempre un enteiro positivo. Practicamente todo o que se está executando no sistema en calquera momento é un proceso, incluíndo a *shell*. A excepción do visto ata o de agora, temos o **kernel** en si, que é un conxunto de rutinas que residen en memoria e ás que os procesos poden ter acceso a través de chamadas ao sistema.

O mesmo que todos os procesos teñen un atributo **PID**, que é o número de proceso que o identifica no sistema, tamén existe un atributo chamado **PPID**. Este número correspóndese có número **PID** do proceso pai.
Todos os procesos deben de ter un proceso que figure como pai pero, entón, ¿que ocorre se un pai morre antes que algún dos seus fillos? Nese caso o **proceso init** adoptará a estes procesos para que non queden orfos.

1.2.2 O proceso de arranque

Cando arranca o sistema prodúcense unha secuencia de procesos que son os seguintes:

1.- Encéndese o sistema e un circuito especial fai que a **CPU** execute o código almacenado na **BIOS**.
2.- O código da **BIOS** realiza algunhas tarefas, entre as que se inclúen a comprobación do *hardware*, a súa configuración e buscar un sector de arranque. Este sector contén o cargador de arranque (**LILO**, **GRUB** ou **GRUB2**).
3.- O obxectivo final do cargador de arranque é atopar un ***kernel*** (Linux ou de calquera outro sistema operativo), cargalo na memoria e executalo.
4.- Cando o ***kernel*** de Linux toma o control, realiza tarefas como inicializar dispositivos, montar a partición raíz e, por último, cargar e executar o programa inicial do seu sistema, que por defecto é */sbin/init*.
5.- O programa inicial (***init***) recibe a **ID de proceso (PID) 1**, posto que é o primeiro programa a executar no sistema. Supoñendo que */sbin/init* sexa o programa inicial, éste leerá un ficheiro chamado */etc/inittab* para determinar que outros programas executar, entre os que, normalmente, están programas *getty* para accesos dende terminal, *scripts* de inicialización do sistema para montar máis particións ou iniciar servizos do sistema, etc.

1.3  Upstart

A distribución Ubuntu utiliza *Upstart*, que xestiona as prioridades, os eventos e as dependencias entre os servizos, pero respecta o mesmo principio que *init*.

1.3.1 Os modos de execución e o proceso de inicialización

Linux baséase en "modos de execución" para determinar que funcionalidades hai dispoñibles. Os modos de execución están numerados do 0 ao 6 e cada un ten asignado un conxunto de servizos que deberían estar activos. Só os modos de execución 0, 1 e 6 son estándar, o resto poden variar nas distintas distribucións de linux.
Ao iniciarse, Linux pasa a un modo de execución determinado, que se pode configurar. É importante saber cales son estas funcións e como xestionar os modos de execución para poder controlar o proceso de arranque de Linux e as operacións en curso.

- **Funcións dos modos de execución**

  ◊ **Modo de execución 0:** É un modo de execución transitorio, o que significa que se emprega para que o sistema pase dun estado a outro. En concreto, cerra o sistema. En equipos con *hardware* moderno, o sistema apagaríase por completo. En caso contrario, espera que o usuario reinicie o sistema manualmente ou que o apague de todo.
  ◊ **Modo de execución 1, s ou S:** Modo monousuario. Os servizos que se inician neste modo de execución, de existir, varían en cada distribución. Emprégase, normalmente, para realizar un mantemento a baixo nivel do sistema que podería verse afectado polo seu funcionamento normal, como o redimensionado de particións.
  ◊ **Modo de execución 2:** En Debian e os seus derivados (como Ubuntu), é un modo multiusuario completo con **X** en execución e un acceso gráfico. A maioría do resto de distribucións deixan este modo de execución sen definir. Dicir que é o modo no que arranca **Ubuntu Server**.
  ◊ **Modo de execución 3:** En Fedora, Mandriva, Red Hat e a mairía das demais distribucións, trátase dun modo multiusuario cunha pnatalla de acceso por terminal (non gráfico).
  ◊ **Modo de execución 4:** Normalmente está sen definir por defecto, polo que está dispoñible para configuracións personalizadas.
  ◊ **Modo de execución 5:** En Fedora, Mandriva, Red Hat e a maioría das demais distribucións, ten o mesmo comportamento que o modo de execución 3, có engadido de que **X** se executa cun acceso gráfico.
  ◊ **Modo de execución 6:** Emprégase para reiniciar o sistema. Este modo de execución é tamén transitorio. O sistema apágase por completo e o ordenador reiníciase automáticamente.

1.4  Importante:

Nunca se deben empregar os modos 0 e 6 como modos de execución por defecto. Se o facemos, o sistema se apagará de inmediato ou se reiniciará unha vez que se termina de por en funcionamento. Poderíase empregar o modo de execución 1 como opción por defecto, pero o máis normal é que se

empregue 2, 3 ou 5, dependendo da distribución e o uso que se lle vaia a dar ao sistema.

A distribución Debian (e as que derivan dela) considera tamén os niveis 2 a 5 como multiusuario, pero non establece diferencias entre estes niveis. Por defecto, como vimos, arráncase no nivel 2, onde se inicia todo, incluso, se é o caso, a interfaz gráfica.

- **/etc/inittab**

Neste arquivo defínese o comportamento do proceso *init* e dos *runlevels*. A sintaxe dunha liña é a seguinte:
Id:[niveis]:acción:comando
- **Id** : Identificador de liña sobre catro caracteres.
- **Niveis** : Indica se se debe ter en conta o comando para o nivel requirido.
- **Acción** : Tipo de acción a efectuar segundo as circunstancias para esta liña.
- **Comando** : O comando a executar cós seus parámetros e as redireccións.

A acción é moi importante, xa que define as actividades de *init* durante o arranque e cambio de nivel (manpage del inittab).

1.5  Ubuntu:

Por defecto, esta distribución non ten un ficheiro */etc/inittab*. Se se desexa cambiar o seu modo de execución pódese crear un ficheiro */etc/inittab* que só conteña a liña *initdefault*.

As principais accións son as seguintes:

· **initdefault** - Define o nivel por defecto durante o *boot* e o inicio de *init*.
· **sysinit** - Este proceso se executará durante o arranque do sistema, antes que calquera entrada *boot* ou *bootwait*. O campo **Niveis** ignórase.
· *boot*, *bootwait*, *off*, *once*, *wait*,...

- **Cambio de nivel**

Pódese cambiar de nivel despois de arrancar a máquina có comando /**sbin**/**init** ou /**sbin**/**telinit**, sendo este un simple vínculo simbólico a *init*. O seguinte comando pasa o sistema ao nivel 5:

```
$ sudo telinit 5
```

O nivel de execución pode verse có comando /**sbin**/**runlevel**. O primeiro valor devolto corresponde ao nivel que precede o nivel actual. Unha **N** significa que non hai un nivel precedente. O segundo valor é o nivel actual:

```
$ runlevel
N 2
```

- **Configuración do sistema básico.**

Sexa o que sexa o nivel de execución especificado por defecto, *init* inicia sempre o comando asociado ás accións *sysinit*, *bootwait* ou *boot* no momento de arracar o sistema.
A acción *sysinit* é a primeira, sendo en **Debian**:
*si::sysinit:/**etc/init.d/rcS***
Logo, máis ou menos igual en todas as distribucións, se executan as seguintes tarefas:
- Configuración dos parámetros do núcleo presentes en /**etc**/**sysctl.conf**.
- Instalación dos ficheiros periféricos (/dev...).
- Configuración do reloxio do sistema.
- Carga das táboas de caracteres do teclado.
- Activación das particións de intercambio SWAP.
- Definición do nome de anfitrión.
- Control e montaxe do sistemas de ficheiros raíz.
- Engadir os periféricos RAID, LVM ou ambos.
- Activación das cuotas de disco.
- Control e montaxe dos outros sitemas de ficheiros.

- Limpeza dos bloqueos (*stale locks*) e dos ficheiros PID no caso de parada brusca.

- **Script /etc/init.d/rc**

Este *script* colle como parámetro o nivel de execución por defecto, ou o parámetro especificado durante a chamada manual dos comandos **init** ou **telinit** e o inicializa, sendo responsable do inicio e da parada dos servizos asociados cando o nivel de execución cambia.

- **Xestión dos niveis e dos servizos:**

O nivel de execución define os servizos que se deben iniciar para este nivel. Correspóndelle ao script **rc** cargar os servizos. Contrólanse os servizos (inicio, parada, reinicio, estatus, etc.) mediante *scripts* presentes no directorio **/etc/init.d**.

```
$ cd /etc/init.d
$ ls -l
total 140
lrwxrwxrwx 1 root root   21 dic  8  2011 acpid -> /lib/init/upstart-job
-rwxr-xr-x 1 root root 4596 abr 12 13:17 apparmor
lrwxrwxrwx 1 root root   21 sep  4 19:52 apport -> /lib/init/upstart-job
lrwxrwxrwx 1 root root   21 oct 25  2011 atd -> /lib/init/upstart-job
-rwxr-xr-x 1 root root 2444 abr 14 11:26 bootlogd
lrwxrwxrwx 1 root root   21 abr 19 18:18 console-setup -> /lib/init/upstart-job
lrwxrwxrwx 1 root root   21 jun 19 22:26 cron -> /lib/init/upstart-job
lrwxrwxrwx 1 root root   21 sep 14 17:47 dbus -> /lib/init/upstart-job
lrwxrwxrwx 1 root root   21 mar 30  2012 dmesg -> /lib/init/upstart-job
-rwxr-xr-x 1 root root 1242 dic 13  2011 dns-clean
lrwxrwxrwx 1 root root   21 mar 14  2012 friendly-recovery -> /lib/init/upstart-job
-rwxr-xr-x 1 root root 1105 may 17 09:07 grub-common
-rwxr-xr-x 1 root root 1329 abr 14 11:26 halt
...
```

Para cada nivel de execución **n**, existe un directorio **rcn.d** que contén vínculos simbólicos cara os servizos presentes en **/etc/init.d** que se queren iniciar ou parar. Este directorio pódese atopar en diferentes lugares segundo a distribución, sendo en Debian e Ubuntu **/etc/rcn.d**.

```
$ cd /etc/rc2.d
$ ls -l
total 4
-rw-r--r-- 1 root root 677 jul 26 20:23 README
lrwxrwxrwx 1 root root  18 sep 25 16:24 S21quotarpc -> ../init.d/quotarpc
lrwxrwxrwx 1 root root  15 sep 13 10:27 S50rsync -> ../init.d/rsync
lrwxrwxrwx 1 root root  19 sep 13 10:27 S70dns-clean -> ../init.d/dns-clean
lrwxrwxrwx 1 root root  18 sep 13 10:27 S70pppd-dns -> ../init.d/pppd-dns
lrwxrwxrwx 1 root root  14 sep 13 10:35 S75sudo -> ../init.d/sudo
lrwxrwxrwx 1 root root  21 sep 13 10:28 S99grub-common -> ../init.d/grub-common
lrwxrwxrwx 1 root root  18 sep 13 10:21 S99ondemand -> ../init.d/ondemand
lrwxrwxrwx 1 root root  18 sep 13 10:21 S99rc.local -> ../init.d/rc.local
```

O prefixo do nome de cada un dos vínculos define a súa orden de execución ou parada. Os seus nomes teñen a seguinte forma:

[SK]nnservizo

- **S** : Start
- **K** : Kill (Stop)
- **nn** : Orden numérico de execución ou parada (00=primeiro, 99=último).
- **servizo** : Nome do servizo.

- **Control manual dos servizos:**

- Mediante *scripts*.

Este método é único por defecto en Debian.
Cada servizo presente en **/etc/init.d** acepta, polo menos, dous parámetros:
- **start** : o servizo se inicia.
- **stop** : o servizo se para.
Vexamos un exemplo có servidor SSH:

```
$ sudo /etc/init.d/ssh start
```

```
ssh start/running, process 1924
$ sudo /etc/init.d/ssh stop
ssh stop/waiting
```

Algúns servizos poden aceptar outros parámetros como:
- **status** : Facilita o estado do servizo.
- **probe** : Indica se é necesario cargar a configuración (se, por exemplo, se modificaron os ficheiros de configuración).
- **reload** / **forcereload** : Indica ao servizo que volva a ler a súa configuración.
- **restart** : Para e volve a iniciar o servizo, sexa cal sexa o final da parada.
- **try-restart** : Para e volve a iniciar o servizo só en caso de parada con éxito.

- Mediante *scripts*.

O comando **service** permite prescindir da ruta cara o *script* de inicio do servizo e utilizar simplemente o seu nome:

```
$ sudo service ssh stop
...
$ sudo service ssh start
...
```

- **Parada do sistema**

Varios métodos permiten parar correctamente unha máquina en Linux.

```
# Para apagar o ordenador:
$ sudo init 0
# Para reiniciar o ordenador:
$ sudo init 6
```

Por outro lado, o comando máis correcto, máis propio e máis seguro para parar o sistema é **shutdown**.

```
# Reinicio para dentro de 10 minutos cunha mensaxe de aviso:
$ sudo shutdown -r +10 "Reinicio para mantemento en 10 minutos"
# Para cancelar ese reinicio:
$ sudo shutdown -c "Mantemento cancelado"
# Apagar o equipo agora mesmo:
$ sudo shutdown -h now
# Reiniciar o equipo agora mesmo:
$ sudo shutdown -r now
```

Outros comandos:
- **halt** ou **poweroff** - Para apagar o equipo. Tamén se pode facer có comando: **shutdown -h**
- **reboot** - Para reiniciar o equipo. Como vimos, tamén se pode facer có comando: **shutdown -r**

1.5.1 systemd

systemd é un conxunto de *daemons* de administración de sistema, bibliotecas e ferramentas deseñados como unha plataforma de administración e configuración central para interactuar có núcleo do Sistema operativo GNU/Linux.
Creouse systemd para remprazar o sistema de inicio (init) herdado dos sistemas operativos estilo UNIX System V e Berkeley Software Distribution (BSD).
No proceso de arranque en Linux, é o primeiro proceso que se executa no espacio de usuario, polo tanto, tamén é proceso pai de todos os procesos fillos no espazo de usuario.
systemd deseñouse para o núcleo de Linux e programado exclusivamente para a API de Linux. Escrito por Lennart Poettering e publicado como Software libre e de código aberto baixo os termos da GNU General Public License (LGPL) versión 2.1 ou posterior.
Dende o ano 2015 prácticamente todas as distribucións Linux optaron por empregar **systemd**.

En systemd falamos de "unidades", que poden ser: *services* (.service), *mount points* (.mount), *devices* (.device) ou *sockets* (.socket).
- Ver o estado dunha unidade:

```
$ systemctl status "unit"
```

- Iniciar unha unidade:

```
$ systemctl start "unit"
```

- Parar unha unidade:

```
$ systemctl stop "unit"
```

- Ver si una unidad está realmente activada o no:

```
$ systemctl is-enabled "unit"
```

- Activar que una unidad se inicie en el *boot*:

```
# systemctl enable "unit"
```

- Desactivar una unidad para que NO se inicie en el *boot*:

```
# systemctl disable "unit"
```

Os *unit files* que nos permiten configurar os procesos a arrancar gárdanse nos directorios:

/usr/lib/systemd/system/: units provided by installed packages
/etc/systemd/system/: units installed by the system administrator

Enlaces interesantes:

- tecmint.com
- archlinux.org
- Freedesktop

1.6 Identificación dos procesos en Linux

1.6.1 ps

O comando ps é o que permite informar sobre o estado dos procesos.
ps esta baseado no sistema de arquivos /**proc**, é dicir, lé directamente a información dos arquivos que se encontran neste directorio. Ten unha grande cantidade de opcións.
Este comando pódenos devolver moita información sobre todos os programas que corren no sistema.

```
$  ps
PID   TTY          TIME      CMD
3081  tty1        00:00:00  bash
3209  tty1        00:00:00  ps
$
```

Por defecto o comando *ps* mostra só os procesos executados polo usuario que o chama e no propio terminal.
No exemplo que se ve só se están executando o propio terminal *bash* e, como non, o comando *ps*.
A saída básica devolve o ID do proceso (**PID**), o terminal dende foi executado (**TTY**) e o tempo de CPU que empregou dito proceso.

A utilidade *ps* ten moitos parámetros, vexamos algúns exemplos interesantes:

- Pódense empregar os parámetros **-a**, **-u** e **-x** para ver todos os procesos que se executan nun momento dado no equipo, incluíndo os de outros usuarios. Estes parámetros pódense empregar con e sin o guión diante:

```
$ ps aux
```

Para ver información dun proceso en particular, por exemplo **sshd**:

```
$ ps aux | grep sshd
root       742  0.0  0.2  49948  2816 ?        Ss   16:57   0:00 /usr/sbin/sshd -D
root      1173  0.0  0.3  73352  3488 ?        Ss   17:45   0:00 sshd: usuario [priv]
usuario   1313  0.0  0.1  73352  1756 ?        S    17:45   0:00 sshd: usuario@pts/0
usuario   1415  0.0  0.0  11908   888 pts/0    S+   17:45   0:00 grep --color=auto sshd
```

- Outro xeito de ver todos os procesos que corren no sistema nun momento dado:

```
usuario@usuario-pc:~$ ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
```

```
root         1     0  0 21:11 ?        00:00:02 /sbin/init
root         2     0  0 21:11 ?        00:00:00 [kthreadd]
root         3     2  0 21:11 ?        00:00:00 [migration/0]
...
```

Este exemplo emprega dous parámetros, o parámetro **-e**, que mostra todos os procesos que corren no sistema, e o parámetro **-f**, que expande a saída para mostrar as columnas mais usuais.

- Se queremos que se mostre a saída en formato longo empregarase o parámetro **-l**:

```
usuario@usuario-pc:~$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000  5065  5062  0  80   0 -  1427 wait   pts/0    00:00:00 bash
4 S  1000  5167  5088  0  80   0 -  1016 wait   pts/0    00:00:00 su
0 S  1000  5173  5167  0  80   0 -  1439 wait   pts/0    00:00:00 bash
0 R  1000  5233  5173  0  80   0 -   629 -      pts/0    00:00:00 ps
```

- Outro parámetro interesante é o **-H**, este parámetro organiza os procesos en formato xerárquico, mostrando a orden na que os procesos se iniciaron.

```
usuario@usuario-pc:~$ ps -efH
UID        PID  PPID  C STIME TTY          TIME CMD
root         2     0  0 16:37 ?        00:00:00 [kthreadd]
root         3     2  0 16:37 ?        00:00:00   [migration/0]
root         4     2  0 16:37 ?        00:00:01   [ksoftirqd/0]
root         5     2  0 16:37 ?        00:00:00   [watchdog/0]
root         6     2  0 16:37 ?        00:00:00   [events/0]
root         7     2  0 16:37 ?        00:00:00   [khelper]
...
```

A columna **CMD** é interesante pois nos mostra o anidamento dos procesos.

- pgrep

Trátase dunha utilidade de liña de comandos derivada dos comandos **ps** e **grep**.
pgrep toma unha expresión regular da liña de comandos e devolve o ID dos procesos con nome compatible con esa expresión regular.

1.6.2 pstree

O comando pstree mostra unha vista en forma de árbore (de forma xerárquica) dos procesos en execución.

Para ver os PID dos procesos:

```
# Para ver os PID dos procesos:
$ pstree -p
init(1)---acpi(889)
        |-atd(897)
        |-cron(896)
        |
        ...
        |-login(938)--bash(1075)--pstree(1426)
        |...
# Para velos por orden:
$ pstree -nps
...
```

1.6.3 jobs

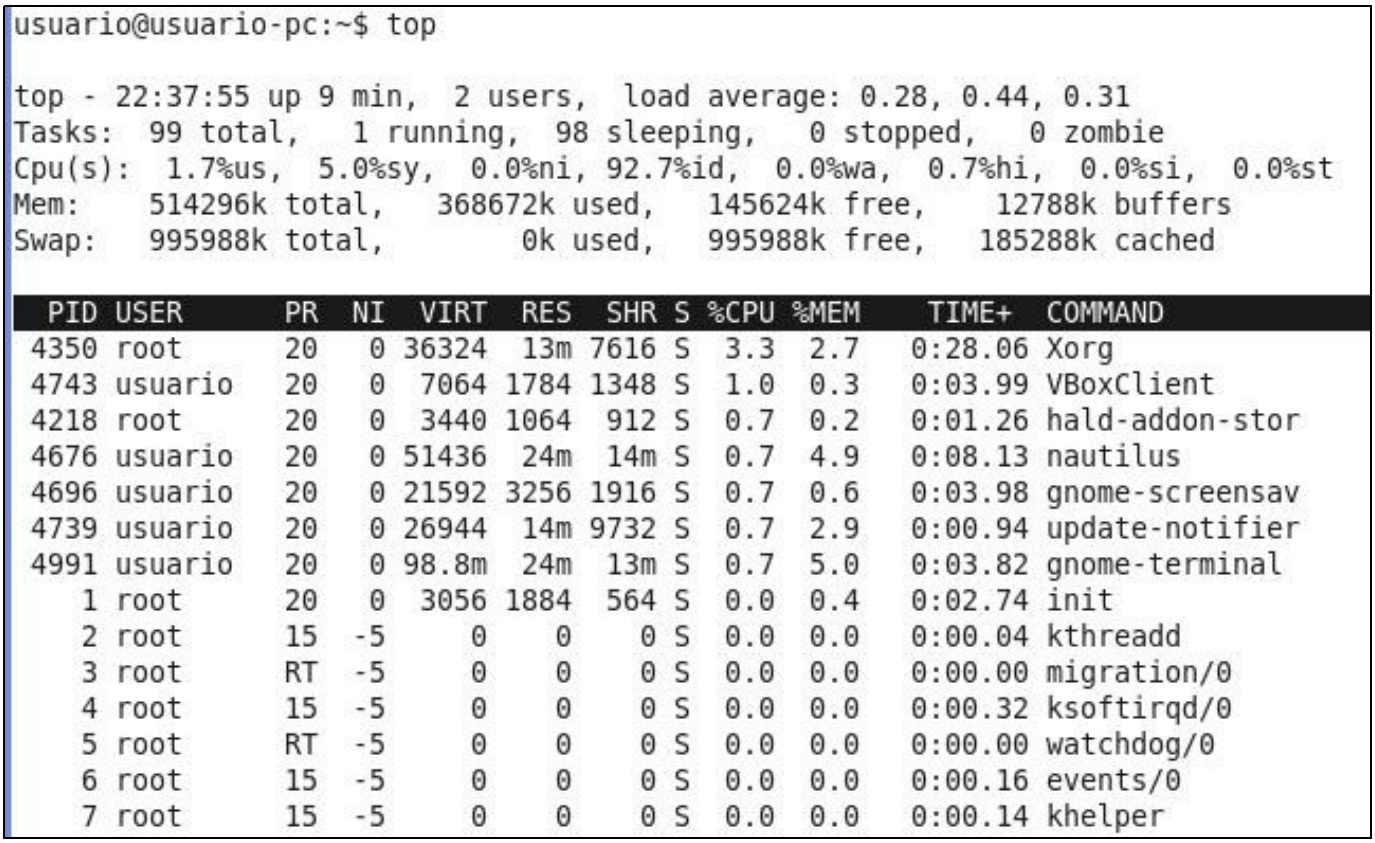
O comando jobs utilízase para listar procesos que se estén executando en segundo plano ou en primeiro plano. Se non devolve ningún tipo de resposta é que non hai procesos presentes.
Vexamos uns exemplos:

```
# Mostra os traballos que se están executando en primeiro plano ou en segundo plano.
$ jobs -l
# Mostra só o identificador de proceso para os traballos en execución.
$ jobs -p
```

1.6.4 top

O comando top nos permite monitorizar o sistema en tempo real, execútase dende a liña de comandos, é interactivo e, por defecto, actualízase cada 3 segundos.

Vexamos que devolve se o executamos:



Estando dentro da apliación, premendo a letra **h** mostra unha axuda dos posibles comandos que permiten configurar top, por exemplo, ao presionar **s** pregunta polo tempo en segundos de actualización, etc.

A versión mellorada de *top* é htop:

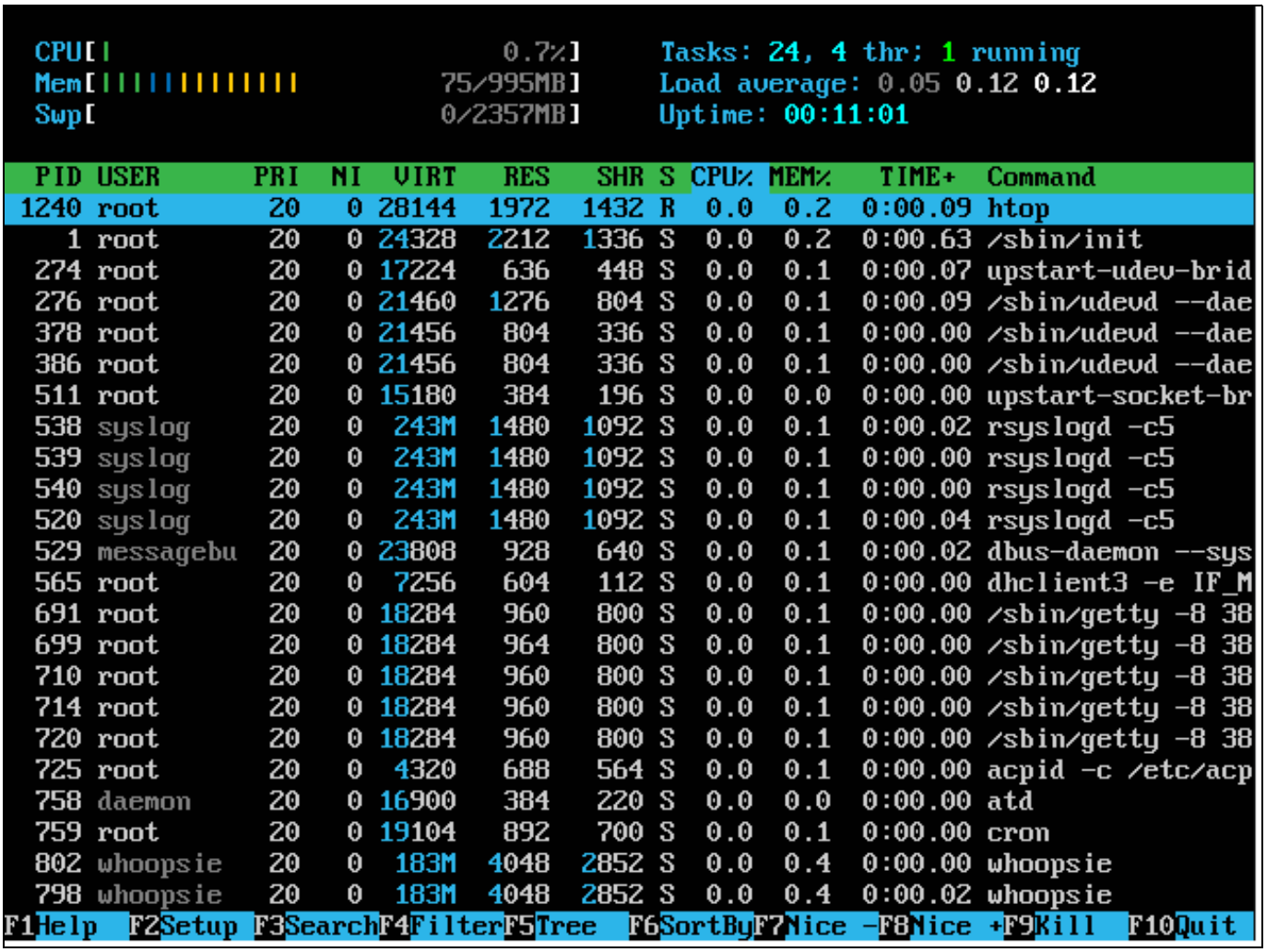
- Este comando pode que non se atope instalado no noso equipo, polo que será preciso instalalo.

```
$ sudo apt-get install htop
```

- Para executalo:

```
$ sudo htop
```

E nos atoparemos unha saída como esta:



· Unha das características é que nos podemos mover polas tarefas.
· Podes modificar o xeito de mostrar os procesos en pantalla presionando "F2" e logo escollendo as opcións axeitadas.
· Pódense ordenar as tarefas por CPU, Memoria, Tempo, PID,... o xeito de facelo é presionando "F6" e logo escollendo a opción que máis nos interese.
· Tamén podes facer un filtro polo nome do proceso pulsando "F4".
· Tamén podes matar un proceso directamente poniéndote sobre el y pulsando "F9".

1.7 Matar un proceso

1.7.1 kill

O comando kill emprégase para finalizar procesos en linux.
Para finalizar un proceso empregando o comando ***kill*** é preciso coñecer o número de identificación do proceso (**PID**) ou o número da tarefa.
Para coñecer estes números basta con listar os procesos empregando o comando **ps**.
Unha vez coñecido o PID basta con executar nun terminal:

```
$ kill [PID]
```

Tamén podemos matar a tarefa:

```
$ kill %[Numero tarefa]
```

É conveniente comprobar que o proceso finalizou e xa non se atopa en execución.
Pode suceder en ocasións que algún proceso non finalice logo de executar **kill**. Nestes casos pódese intentar finalizalos có comando **kill** e un sinal mais forte que forza os procesos a rematar mais aló da súa vontade, como por exemplo:

```
$ sudo kill -9 [PID]
```

Matar un proceso e todos os seus procesos fillos:

```
$ sudo kill -TERM [PPID]
# Utilizando pgrep para matar apache2 e todos os seus procesos fillos:
$ sudo kill -TERM -$(pgrep -f apache2)
# Con pgrep -f buscamos polo nome do proceso
```

Exemplo:

Para saber cal é o PID dun proceso podemos executar un *ps aux* previo para coñecer o PID a partires do nome do comando:

```
ps aux | grep bash
```

Mostraría algo así:

```
USER   PID

1000   3114   ...etc
```

O interesante é a parte correspondente á segunda columna (PID), entón escribiriamos:

```
kill SIGKILL 3114
```

(SIGKILL é o mesmo que poñer -9, facer un kill -l para comprobar que o sinal 9 corresponde a SIGKILL)

Co comando *pidof* obteremos o, ou os PID, asociados a un determinado proceso a partir de nome do mesmo

```
echo $(pidof soffice)
```

1.7.2 pkill

Este comando é moi útil, pois permite matar un proceso utilizando o seu nombre. Deste xeito o seu funcionamento é similar a kill acompañado de pidof

```
sudo pkill soffice
```

Mata o proceso soffice (asociado a libreoffice)

1.7.3 killall

As veces ocorre que, ao listar os procesos activos, o que se desexa finalizar ten distintas instancias abertas, é dicir, dispón de varias PID e para finalizalo precísase utilizar varias veces o comando **kill**. O comando killall utilízase para finalizar todos os procesos que abre un comando.
Como se pode supoñer, ao dispor de distintas PID non é este dato o que se debe indicar ao comando **killall** senón o nome do proceso. Este nome ven dado no listado de procesos como **CMD**; entón:

```
$ sudo killall [CMD]
```

1.7.4 xkill

O comando xkill é unha utilidade para forzar o Servidor X a cerrar conexións. Este comando pódese utilizar como o comando **kill**:

```
$ sudo xkill -id  [PID]
```

Este comando pódese empregar sen necesidade de indicar o PID, xa que se se introduce nun terminal:

```
$ xkill
```

O cursor cambiarase a unha caveira e a aplicación matará o proceso que controle a fiestra onde se faga *click* coa caveira.

1.8 Pausar un proceso

Para pausar un proceso, como para matalo, é preciso coñecer o número de identificación do proceso (PID).

Unha vez coñecido o PID basta con executar no terminal:

```
$ sudo kill -STOP [PID]
```

Para reanudalo basta con teclear:

```
$ sudo kill -CONT [PID]
```

1.9 Procesos en primeiro e segundo plano: *fg* e *bg*

Para traballar con procesos en primeiro e segundo plano Linux ten as ferramentas fg e bg.

Pódese lanzar un proceso de xeito normal (en primeiro plano), paralo e despois relanzalo en segundo plano.

1.9.1 fg

Vexamos un exemplo no que se empregará a utilidade yes e fg.

```
# Lanzamos a utilidade "yes"
$ yes
# Pausamos a execución da utilidade
# premendo Control + Z
# Comprobamos que "yes" está detido:
$ jobs
[1]+   Detenido    yes
# Reanudamos a execución da tarefa
# empregamos o comando "fg":
$ fg %1
# Paramos completamente a execución da utilidade
# premendo Control + C
```

1.9.2 bg

Para executar un comando directamente en segundo plano podemos facelo introducindo o carácter "&" ao final:

```
# Lanzamos "yes" en segundo plano
# enviando a súa saida a "null"
$ yes > /dev/null &
[1] 746
# Vemos que nos devolve o número de tarefa
# e, ademais, o PID
$ jobs
[1]+ Ejecuntando    yes > /dev/null &
$ ps
746 tty1    00:10:10 yes
# Matamos a tarefa
$ kill %1
$ jobs
[1]+   Terminado    yes > /dev/null
```

Vexamos un exemplo no que se empregará a utilidade yes e bg.

```
# Lanzamos a utilidade "yes"
# a súa saída a redireccionamos a "null"
$ yes > /dev/null
# Pausamos a execución da utilidade
# premendo Control + Z
# Comprobamos que "yes" está detido:
$ jobs
[1]+   Detenido    yes > /dev/null
# Reanudamos a execución da tarefa
# empregamos o comando "bg":
$ bg %1
[1]+ yes > /dev/null &
# Comprobamos con "jobs" que o comando segue executándose
$ jobs
[1]+ Ejecutando    yes > /dev/null &
# Paramos completamente a execución da utilidade:
```

```
$ kill %1
# E miramos se segue en execución:
$ jobs
[1]+ Terminado    yes > /dev/null
```

Vemos que utilizando **&** ao final executamos o comando e liberamos o terminal.

2 Utilidades de manexo de servizos ao inicio

---- Depreciadas, a día de hoxe a maioría das distribucións empregan systemctl xa explicada antes ----

As utilidades chkconfig, sysv-rc-conf e update-rc.d poden usarse para activar ou desactivar servizos.

2.1 chkconfig

O comando **chkconfig** emprégase para cambiar, actualizar e consultar información de *runlevel* para os servizos do sistema.

Opcións:

--add servicio : Crea un inicio en cada *runlevel* especificado para o servizo especificado de acordo ao comportamento por defecto especificado no script de inicialización do servicio.
--list : Mostra se o servizo especificado está activo ou non en cada nivel.
--level números : Especifica mediante números un ou varios niveis de execución a cambiar.
--del servicio : Elimina entradas para o servizo especificado en todos os niveis de execución.

Exemplos:

- Ver unha lista dos servizos do sistema e se estos están arrancados (*on*) ou detidos (*off*) nos niveis de execución 0-6.

```
$ chkconfig --list
```

- Agregar o servicio web(***httpd***) nos modos de execución por defecto:

```
$ chkconfig --add httpd
```

- Agregar o servicio web (***httpd***) e facer que arranque nos niveis 2, 3, 4 e 5:

```
$ chkconfig --level 2345 httpd on
```

- Eliminar o servicio web(***httpd***) do listado de servicios:

```
$ chkconfig --del httpd
```

- Deshabilitar o servicio web(***httpd***) no nivel de execución 5:

```
$ chkconfig --level 5 httpd off
```

2.2 update-rc.d

O comando **update-rc.d** permítenos automatizar o proceso de creación e borrado de enlaces aos *scripts* de inicio, có fin de iniciar/parar servizos.

Exemplos con update-rc.d

- Se queremos eliminar servizos do proceso de arranque, non temos máis que executar o seguinte comando: **update-rc.d -f nome_proceso remove**

```
# Eliminar o proceso squid do arranque:
$ update-rc.d -f squid remove
```

- Se queremos crear enlaces usando parámetros por defecto, executaremos: **update-rc.d nome_proceso defaults**

```
# Crear enlace a squid empregando os parámetros por defecto:
```

```
$ update-rc.d squid defaults
```

Con esta opción crearanse enlaces para arrancar o servizo nos niveis de execución 2345 e parar o servizos nos niveis de execución 016. Por defecto, todos os enlaces terán o código de secuencia 20.

- Se preferimos especificalo todo:

```
# Crear enlaces para arrancar o servizo ssh nos niveis 2345
# e paralo nos niveis 016. O código de secuencia sería 20.
$ update-rc.d ssh start 20 2 3 4 5 . stop 20 0 1 6 .
```

3 Outros comandos

3.1 netstat e lsof

A veces resulta útil coñecer a qué proceso está vinculado un servicio TCP/IP, é dicir un servicio que esté funcionando en rede e que admita peticións de clientes

Podemos visualizar esta información utilizando

netstat -anp | grep numero_porto

```
netstat -anp | grep 80
```

ou mediante lsof -i:numero_porto

```
lsof -i:80
```

Os exemplos anteriores mostrarían a información do servizo vinculado ó porto 80 TCP. Si quixéramos ver con netstat a información relativa a portos UDP usariamos a opción -u

3.2 Comando nohup

Cando se crea un proceso, por exemplo mediante a invocación dun comando, éste crearase como fillo do contexto no que se crea. Si estamos traballando nunha sesión de terminal e escribimos:

```
find / -user `whoami` > `whoami`.proc
```

Buscaría en todo o sistema aqueles arquivos e directorios que son propiedade do usuario que realiza a invocación e garda o resultado da búsqueda nun arquivo **<nombre_usuario>.proc.** Notade que `whoami` devolvería o nome do usuario que o invoca. Esta búsqueda pode levar o seu tempo e, si durante ese proceso necesitamos ausentarnos e pechar a terminal, o proceso deixaría de executarse, pois foi creado como fillo da shell da sesión da terminal actual que, de deterse, causaría que se eliminaran todos os seus procesos fillos.

Para evitar este comportamento podemos usar o comando **nohup**. Este comando, seguido do comando que se vai a executar a continuación, causa que éste non se execute como fillo da shell dende a que se invoca, senón como un proceso nun nivel superior que, no caso de cerrarse a shell, non se eliminará ó non tratarse dun proceso fillo da mesma.

```
nohup find / -user `whoami` > `whoami`.proc
```

Ahora sí podería pecharse a shell e o proceso de búsqueda continuaría ata a súa finalización

Volver