Capítulo 3

Xestión de procesos

3.1. Conceptos básicos

A definición mais habitual de proceso é "programa en execución". Un programa é un "executable" gardado no disco duro do ordenador susceptible de ser cargado en memoria para que se execute. Cando este programa se carga en memoria asignaselle un código único denominado *process ID* (PID), ao cal se lle asigna unha estructura de datos, denominada *task struct* [1] que contén información como o ID do proceso "pai" (PPID), recursos asociados ao proceso, UID do propietario, atributos de seguridade, etc.

Cando arranca o ordenador iniciase un proceso denominado *init* con PID 1, que é o único do sistema que non ten "pai" [2]. Este á súa vez iniciará outros procesos (fillos), e estes outros e asi sucesivamente ata construir a "árbores" de procesos.

3.2. Estados dun proceso

Dende que se lanza un proceso ata que sae da lista de procesos, este pode atravesar distintos estados, tal e como se representa na figura 3.1

- Listo: cando o proceso se carga na memoria entra neste estado, significa que está preparado para entrar en execución en canto lle "toque".

[1] https://www.tldp.org/LDP/tlk/kernel/processes.html
[2] https://www.tecmint.com/linux-process-management/

Figura 3.1: Estados dun proceso (sacado de [3])

- Execución: en ordenadores monoprocesador so un proceso pode ocupar a CPU á vez, polo cal o sistema operativo terá un "controlador de procesos" que mediante un determinado algoritmo vai repartindo o tempo de CPU entre eles. Cando a un proceso lle toca o turno de ocupar a CPU pasará ao estado de "execución", e cando lle toque o turno a outro este volverá ao estado "listo".

- Espera: un proceso en execución, ademais de voltar ao estado "listo" se remata o seu tempo de CPU, pode pasar ao estado de "espera" cando ten que realizar outras operacións que requiran bastante tempo, e non precisen da CPU. O caso mais típico son as operacións de entrada/saída: lectura de datos dun arquivo, introdución de valores por teclado, etc. Cando esa operación remate pasará ao estado de "listo" para volver a ocupar a CPU cando lle toque.

- Parado: un proceso en execución pode ser parado se recibe un "sinal" determinado. Mais adiante falaremos de como o usuario pode parar ou reanudar un proceso. Cando un proceso parado se reanuda pasa ao estado "listo".

- Zombie: como apuntamos antes todo proceso, excepto o *init*, ten un pai, que é quen o "lanza". Por exemplo, cando nunha consola tecleamos un comando, o pai é o propio *shell*. Cando un proceso remata, enviase un sinal ao proceso

[3] https://www.monografias.com/trabajos26/estados-proceso-hilos/estados-proceso-hilos.shtml

pai, que é quen pode sacalo da "lista de procesos en execución". En certos casos é posible que o proceso pai xa non exista, co cal o proceso remata pero quédase nun estado de "espera polo pai", que é ao que denominamos estado "zombie".

3.3. Control de procesos

Unha vez explicados os conceptos básicos imos abordar a parte mais práctica, tratando dos distintos comandos relacionado coa xestión do procesos, tanto para obter información dos procesos en execución, como para "controlalos" ou para modificar as propiedades coas que os lanzamos.

3.3.1. Obter información dun proceso

O comando estándar e mais empregado para obter información dos procesos do sistema é o comando *ps*. Sin opcións simplemente amosa os procesos nosos asociados ao terminal dende o que se invoca o comando. A saída mais frecuente é esta:

```
PID    TTY      TIME        CMD
6555   pts/0    00:00:00    bash
20046  pts/0    00:00:00    ps
```

O primeiro campo é o PID do proceso, o segundo o terminal asociado ao mesmo, o terceiro o tempo de CPU consumido e o cuarto o nome do comando. Como vemos aparece o propio comando *ps* e o *shell* que está activo no terminal.

Para obter información sobre outros procesos ou obter mais campos de información hai que empregar opcións. Este comando é un pouco extrano porque permite indicar opcións seguindo tres estilos diferentes. O primeiro é o estilo "UNIX" no cal as opcións van precedidas dun guión, o segundo é o estilo "BSD", sin guión, e o terceiro é o estilo "GNU" con nomes largo precedidos de dobre guión.

O mais práctico será ver exemplos de uso habituais. Para ver todos os procesos e información completa deles podemos empregar a sintaxe *ps -ef* ou *ps aux*. A saída sería algo como isto:

```
USER     PID %CPU %MEM    VSZ   RSS TTY  STAT START TIME COMMAND
root       1  0.0  0.1 169456 10420 ?    Ss   dic05  1:32 /sbin/init splash
root       2  0.0  0.0      0     0 ?    S    dic05  0:00 [kthreadd]
...
anton  22956  0.0  0.3 199396 28980 ?    Sl   dic16  0:00 /usr/bin/python3 ...
anton  23362  0.0  1.5 2275744 123348 ?  Sl   dic16  0:02 /usr/bin/gnome-calendar ...
anton  23785  0.0  0.3 237460 26512 ?    Sl   dic16  0:03 /usr/lib/firefox/firefox ...
...
```

O primeiro campo de información agora é USER, indicando o nome do usuario que lanzou o proceso. Outros campos son %CPU e %MEM, que indican o uso da CPU e a memoria. O campo STAT indica o estado do proceso (S sería *sleep*).

Con esta sintaxe obteremos un longo listado, posiblemente de centos de procesos. Se estamos interesados en algún en concreto sempre podemos filtrar a saída cun *grep*.

Para unha saída personalizada podemos empregar a opción "-o lista_de_campos". Por exemplo *ps -u anton -o pid,stat,cmd* amosaría so os procesos do usuario "anton" e indicando so os campos PID, STAT e CMD. Tamén é interesante a opción "-H" que permite apreciar mellor a árbores de procesos. Vexámolo cun exemplo, tecleando a orde *ps -u anton -H -o pid,ppid,stat,cmd*.

```
PID  PPID STAT  CMD
...
1969  1921 Ssl+ /usr/lib/gdm3/gdm-x-session ...
1973  1969 Sl+      /usr/lib/xorg/Xorg vt2 ...
1983  1969 Sl+      /usr/lib/gnome-session/gnome-session-binary ...
2055  1983 Ss           /usr/bin/ssh-agent ...
...
```

Como vemos o proceso 1969 ten dous fillos, o 1973 e 1983 (fixarse no campo PPID), e eso indícase tabulando os seus nomes. A súa vez o proceso 1983 ten un fillo, o 2055, que de novo se indica mediante un tabulador adicional.

Hai moitas páxinas [4] con exemplos interesantes para obter información dun proceso cun PID concreto, dunha terminal concreta, baixar ao nivel de *threads*, ordenados por diferentes criterios, etc.

Para ver a xerarquía de procesos existe un comando específico, *pstree* [5]. Sin opcións incluiría todos os procesos do sistema pero podese indicar un UID, para

[4] https://www.tecmint.com/ps-command-examples-for-linux-process-monitoring/
[5] https://www.howtoforge.com/linux-pstree-command/

ver os procesos dun usuario ou u PID para ver a aŕbore dun proceso específico. Unha opción interesante é a "-p" para que amose o PID dos procesos. Outra opción útil é "-T" que oculta os *threads*, aspecto no que nos imos entrar. Vexamos un exemplo: *pstree -T -p anton*

```
systemd(1939)-+-(sd-pam)(1940)
              |-at-spi-bus-laun(2075)---dbus-daemon(2083)
              |-at-spi2-registr(2118)
              |-dbus-daemon(1971)
              |-dconf-service(2171)
              |-dropbox(1408)
              |-evolution-addre(2286)
              |-evolution-calen(2252)
              |-evolution-sourc(2226)
              |-firefox(22603)-+-RDD Process(23785)
              |                |-Web Content(15652)
              |                |-Web Content(16874)

              ...
```

Amosamos so unha parte da saída, correspondente aos fillos do proceso 1939, o proceso *systemd*, do que falaremos en capítulos posteriores. Vemos que un dos seus fillos é o proceso 22603, que a súa vez ten varios procesos dependentes del.

Outro comando moi útil é o comando *kill* [6]. Aínda que o seu nome parace indicar que sirve para "matar" procesos en realidade sirve para enviar sinais a un proceso, sexa para matalo ou para outro fin. A sintaxe xeral é a seguinte:

```
kill [opcións] pid
```

Mediante as opcións permítese enviar diferente sinais ao proceso identificado polo seu PID. Para obter un listado de sinais posible podemos facer *kill -l*. O sinal por defecto é SIGTERM ou 15. Cando un proceso recibe este sinal trata de finalizar, realizando antes as tarefas que teña programadas ou pendentes. Podemos decir que é un final "ordeado".

As veces un proceso non responde a este sinal, porque está bloqueado por algunha causa, ou porque non foi ben programado. Neses casos pódeselle enviar

[6] https://www.linuxtotal.com.mx/index.php?cont=info_admon_012

o sinal SIGKILL ou 9 (*kill -9 PID*). Este sinal é unha orde directa ao *kernel* para que finalice inmediatamente o proceso, sin permitirlles realizar mais tarefas.

Unha variante deste comando é *killall* que permite enviar sinais a un proceso por nome, en vez de por PID. Pero ademais ten unha opción interesante *-u user* que permite enviar o sinal á vez a todos os procesos do usuario indicado. Tamén pode ser útil a opción *-i* que solicita confirmación antes de eliminar cada proceso.

Se queremos facer unha monitorización contínua dos procesos en execución o mellor é empregar o comando *top*. Sen argumentos amosa unha pantalla con información detallada dos procesos, que se vai actualizando continuamente. Algo como o indicado a continuación.

```
top - 12:18:16 up 13 days,  4:04,  1 user,  load average: 0,32, 0,14, 0,10
Tasks: 264 total,   1 running, 263 sleeping,   0 stopped,   0 zombie
%Cpu(s):  1,0 us,  0,4 sy,  0,0 ni, 97,9 id,  0,7 wa,  0,0 hi,  0,0 si,  0,0 st
MiB Mem :   7855,3 total,    404,8 free,   3295,3 used,   4155,2 buff/cache
MiB Swap:   2048,0 total,   1889,0 free,    159,0 used.   3841,4 avail Mem

PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
24110 anton   20   0 1838284 170748  95828 S   1,1   2,1  23:46.04 franz
25037 anton   20   0 2697424 264432 129440 S   1,1   3,3   4:59.22 Web Content
24401 anton   20   0 4106776 468240 176672 S   0,4   5,8  57:21.25 firefox
...
```

As filas da parte superior amosas información xeral do sistema, como o uso da memoria. As filas inferiores conteñen información sobre os procesos en execución. Das columnas PR e NI falaremos mais adiante. Destacar o porcentaxe de uso da CPU (%CPU) e da memoria (%MEM) asi como o tempo total consumido de CPU dende o inicio do proceso (TIME+).

Por defecto os datos actualizanse cada 5 segundo, para empregar outro periodo podemos facer *top -d n*, sendo o "n" o número de segundos de retardo.

Existe unha versión mellorada deste comando denominada *htop* [7] que emprega colores e melloras en canto a control da información a visualizar.

Un dos problemas cos que nos atopamos a veces é a ralentización do ordenador por un acceso continuo ao disco duro. Un comando que nos permite obter información acerca dos procesos que están facendo accesos ao disco é *iotop* [8], que require ser executado co comando *sudo*. Un exemplo de uso sería o seguinte: *sudo iotop -o -d 5* que indica que so queremos ver os procesos que están accedendo

[7] https://www.atareao.es/software/utilidades/monitorizar-nuestro-sistema-con-htop/
[8] https://likegeeks.com/es/gestion-de-procesos-de-linux/#comandonbspps

ao disco (-o) e que se actualice a información cada 5 segundos. A saída sería algo como o seguinte:

```
Total DISK READ:         0.00 B/s | Total DISK WRITE:        95.38 K/s
Current DISK READ:       0.00 B/s | Current DISK WRITE:      88.23 K/s
TID   PRIO  USER    DISK READ  DISK WRITE  SWAPIN     IO>    COMMAND
  922 be/3 root     0.00 B/s    11.92 K/s  0.00 %  0.35 %   [jbd2/sda7-8]
 3892 be/4 root     0.00 B/s     0.00 B/s  0.00 %  0.11 %   [kworker/u8:0-i915]
24363 be/4 anton    0.00 B/s  1627.79 B/s  0.00 %  0.00 %   java ...
24541 be/4 anton    0.00 B/s    23.84 K/s  0.00 %  0.00 %   firefox ...
```

De novo hai unhas liñas de cabeceira con información global e logo detalles de todos os procesos que están accedendo a disco, indicando o ancho de banda consumido para lectura (DISK READ) e escritura (DISK WRITE).

3.3.2. Procesos en primeiro e segundo plano

Cando lanzamos un proceso dende unha terminal este bloqueao ata que remate, non sendo posible lanzar outro proceso mentres tanto. Este modo por defecto de lanzar os procesos denomíname "en primeiro plano" ou *foreground* [9].

Se o noso proceso non require interacción co usuario (por medio de teclado e pantalla) existe un método alternativo de lanzalo para que non manteña o terminal bloqueado. Coñecese como "segundo plano" ou *background*. Para iso o único que temos que facer é engadir o símbolo "&" ao final do comando. Vexamos un exemplo:

```
find / -name "*.c" -print 2>/dev/null |wc -l >saida.txt &
```

Con este comando pretendemos buscar todos os arquivos con extensión ".c" e contalos. Se so executamos *find / -name "*.c" -print | wc -l* o proceso bloquearía o terminal ata rematar, e sacaría por pantalla o número total de arquivos. Esto pode tardar varios minutos. Ademais disto, como existen carpetas ás que un usuario normal non pode acceder estarían saíndo por pantalla mensaxes de erro do tipo "find: ‘./.cache/dconf’: Permission denied". Coa solución proposta o que facemos é lanzar o proceso en segundo plano e ademais redireccionamos a saída cara o arquivo "saida.txt" para non molestar no propio terminal. Tamén redireccionamos a saída de error cada o dispositivo */dev/null*, que equivale a descartalos.

Cando lanzamos o proceso deste xeito vemos que en pantalla sae unha mensaxe como esta:

[9] https://www.geeksforgeeks.org/processes-in-linuxunix/

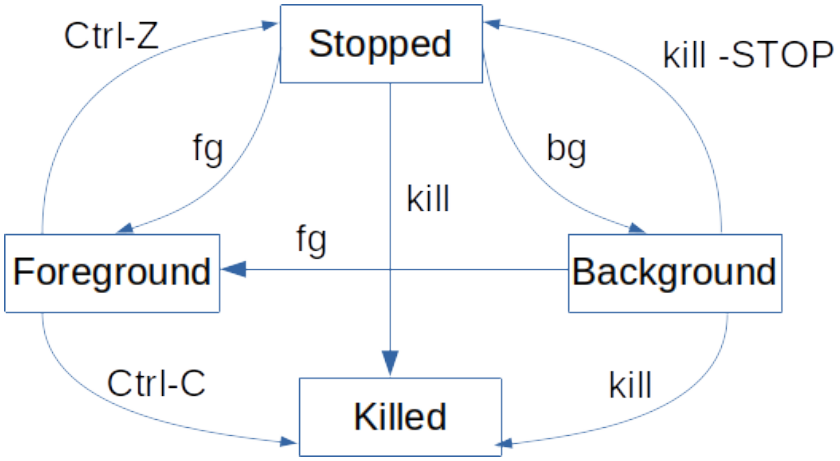

Figura 3.2: Comandos para o control dos traballos

```
[1] 5078
```

e a continuación devolvenos o *prompt* do sistema para que podamos seguir traballando. O segundo número é o PID do proceso, por se queremos interaccionar con el. O primeiro número, entre "[ ]" é o número de traballo (*job*).

Na figura 3.2 amósase un equema cos comandos que nos permiten cambiar o estado dun proceso.

Un proceso que esté en primeiro plano pode ser detido pulsando as teclas Ctrl-Z ou eliminado pulsando Ctrl-C. Un proceso que esté en segundo plano pode ser parado enviandolle o sinal STOP ou parando co comando *kill*, tamén pasado a primeiro plano co comando *fg %n*, sendo "n" o número de traballo. Por último un proceso parado pode ser reanudado en primeiro plano co comando *fg %n*, en segundo plano con *bg %n* ou parado con *kill PID*.

3.3.3. Modificar a prioridade dos procesos

Como antes apuntamos, nun sistema multiusuario e multitarea, como é o caso de Linux, unha das tarefas que realiza o sistema operativo é a xestión da CPU para repartir o tempo entre os distintos procesos. Existen múltiples algoritmos para iso que soen empregar *slots* de tempo e os van asignando de xeito cíclico aos proceso que estén en estado "listo". Sobre esta base hai variantes que teñen en conta outros aspectos para alterar este reparto "equitativo".

Un dos mecanismos permitidos por Linux e asignar distintas prioridades aos procesos, de xeito que os de maior prioridade reciban mais *slots* que os de baixa

prioridade. Por defecto os procesos lanzados polos usuarios "normais" reciben todos a mesma prioridade por defecto, pero é posible alterala.

Para iso temos o comando *nice* que ten a seguinte sintaxe:

```
nice -n comando
```

O que facemos é lanzar a execución do comando correspondente incrementando a "amabilidade" (*niceness*) do proceso unha cantidade "n". Para entendelo mellor voltemos ao listado da páxina 33. Nese listado había duas columnas, PR e NI. NI é a abreviatura de *niceness* e como vemos todos os procesos teñen un valor 0. A columna PR é a abreviatura de *priority* é a efectos prácticos non é mais que a suma de 20+niceness.

Se lanzamos agora un comando con nice, `nice -10 sleep 1500 &` por exemplo, e observamos o seu estado con *top -p PID* obteremos o seguinte:

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 8524 anton     30  10   10596    780    716 S   0,0   0,0   0:00.00 sleep
```

Como vemos o parámetro NI agora vale 10 e PR 30. Hai que ter en conta que maior valor da prioridade significa realmente menor prioridado do proceso, ou sexa, que se lle adica menor porcentaxe de uso da CPU.

Os usuarios normais so poden baixar a prioridade por defecto. Para subila hai que actuar como administrador, por exemplo:

```
sudo nice -n -10  sleep 1500 &
```

Tamén se pode variar a prioridade dun proceso en curso co comando *renice*. Por exemplo:

```
renice -n 20  1560
```

sendo 1560 o PID do proceso.

3.3.4. Comando *nohup*

Cando lanzamos un proceso dende unha *shell* este proceso será fillo da propia *shell*. Por iso, cando a pechamos, ben directamente, ben porque saimos da sesión, a *shell* enviará a ese proceso fillo o sinal SIGHUP (relacionado con *hangup*, colgar en inglés) que normalmente fará que ese finalice.

Se queremos lanzar un proceso que continue executándose aínda que pechemos a sesión podemos facelo mediante o comando *nohup*. A sintaxe sería a seguinte:

```
nohup programa &
```

Eso fará que o proceso se lance desvinculado da propia *shell* e enviando a súa salida a un arquivo, por defecto *nohup.txt*. Deste xeito o proceso seguirá executándose ainda que finalice a *shell* ou a propia sesión.