

Fonaments dels Sistemes Operatius

Departament de Informàtica de Sistemes i Computadores (DISCA) *Universitat Politècnica de València*



Pràctica 3 Monitorització de processos en LINUX

Contingut

1. Objectius.....	2
2. Utilitats i ordres del shell per a processos	2
2.1. Ordre ps	2
2.2. Ordre top	3
2.3. Ordre kill	3
3. El directori /proc	4
4. Ferramentes per a monitorització de processos	6
4.1. Ordre grep	6
4.2. Shell script.....	7
4.3. Variables	8
4.4. Bucles for	8
4.5. Sentència if	9
4.6. Ordre test	9
4.7. Ordre awk	11
5. ANEXO. Descripció d'ordres i filtres.....	14
5.1. Variable PATH	14
5.2. Operacions aritmètiques	15
5.3. Bucles while	15
5.4. Awk, els patrons especials BEGIN i END.....	16
5.5. Awk, la sentència IF	16
5.6. Ordre TR	17
6. ANNEX. Consulta de informació de sistema en Mac OS X.....	18

1. Objectius

El objectiu de aquesta pràctica es capacitar a l'alumne per a obtenir informació al voltant de l'execució dels processos que hi ha actius en cada moment en el sistema (p.ej. PID, consum de CPU, ocupació de memòria, etc). Per a això es treballarà amb el sistema de arxius, ja que LINUX emmagatzema en el directori /proc tota la informació de monitorització de processos i, a partir de aquesta informació, se realitzarà un estudio pràctic de les ferramentes:

- ps, top, grep, awk i kill.

2. Utilitats i ordres del shell per a processos

Des del seu directori **home**, amb l'ordre **mkdir**, crea un directori denominat **fso_pract3** per a realitzar aquesta pràctica:

```
$ mkdir fso_pract3
$ cd fso_pract3
```

2.1. Ordre ps

L'ordre **ps** permet llistar informació sobre la execució dels processos en Linux.

```
$ ps u
```

USER	PID	%CPU	%MEM	VSZ	RSS	TT	STAT	STARTED	TIME	COMMAND
gandreu	251	0,0	0,0	2435492	1112	s000	S	4:20PM	0:00.01	-bash

El comando **ps** mostra la llista de processos del sistema basant-se per a això, en la informació del sistema de arxius **/proc**. Dependent dels arguments utilitzats amb **ps** es mostra més o menys informació, distribuïda en un format de columnes. A continuació, es mostra una llista de les columnes més habituals:

PID	Nombre d'identificador del procés.
PPID	Nombre d'identificador del pare del procés
UID	Usuari propietari del procés
TTY	Terminal associada al procés. Si no hi ha apareix un ?
TIME	Temps de uso de CPU acumulat pel procés
CMD	El nombre del programa o comando que va iniciar el procés
RSS	Resident Size, grandària de la part resident en memòria en kilobytes
SIZE	Grandària virtual de la imatge del procés
NI	Valor Nice (prioritat) del procés. Quant menys valor, més prioritat.
PCPU	Percentatge de CPU utilitzat pel procés
STIME	Starting time. Hora d'inici del procés
STAT	Estat del procés.

En la columna STAT es mostra el estat del procés, que pot ser:

- R (Runnable): En execució.
- S (Sleeping): Procés en espera d'algun esdeveniment per a poder continuar.
- T (sTopped): Procés detingut totalment. Pot ser reiniciat posteriorment.
- Z (Zombie): Procés zombie.
- D (uninterruptible sleep): Procés en espera ininterrumpible, associats a accions d'entrada/sortida.

A la seua vegada, junt amb la lletra que indica l'estat del procés, poden aparèixer caràcters addicionals que donen més informació sobre l'estat d'un procés. Els més habituals són:

- +: El procés està executant-se en primer pla.
- X : Procés que està sent depurat.

Per defecte, el comando **ps** mostra la informació referent a PID, TTY, TIME i CMD. Executant-lo amb diferents opcions podem controlar quanta informació es mostrarà.

1.Exercici: Executa l'ordre **ps** amb els següents arguments i observa les diferències en la sortida que generen.

```
$ps
$ps u
$ps -la
$ps f
$ps aux
```

Habitualment **ps** s'utilitza conjuntament amb l'ordre **grep** que se explica més avant.

2.2. Ordre top

L'ordre **top** mostra en temps real un llistat dels processos que se estan executant en el sistema, especificant el % de CPU i memòria utilitzat, els seus PID's, usuaris, etc. Executa l'ordre **top** i en pantalla t'apareixerà la informació dividida en dues parts:

```
Processes: 72 total, 2 running, 1 stuck, 69 sleeping, 322 threads
Load Avg: 0.15, 0.17, 0.16 CPU usage: 2.87% user, 3.34% sys, 93.77% idle SharedLibs: 16M resident, 11M data, 0B linkedit.
MemRegions: 11390 total, 385M resident, 39M private, 431M shared.
PhysMem: 627M wired, 936M active, 193M inactive, 1756M used, 2340M free.
VM: 164G vsize, 1123M framework vsize, 84839(0) pageins, 0(0) pageouts. Networks: packets: 7535/6626K in, 5609/1252K out.
Disks: 22675/1057M read, 12667/355M written.
16:58:05
```

PID	COMMAND	%CPU	TIME	#TH	#WQ	#POR	#MREG	RPRVT	RSHRD	RSIZE	VPRVT	VSIZE	PGRP	PPID	STATE	UID	FAULTS	COW
288	Grab	3.8	00:01.72	5	4	108-	211+	4088K+	40M-	15M+	29M+	2483M+	288	143	sleeping	504	13620+	366
286	quicklookd	0.0	00:00.05	4	1	72	73	2468K	7384K	6016K	534M	2920M	286	143	sleeping	504	1921	228
281-	CVMCompiler	0.0	00:00.03	1	0	27	38	904K	460K	1892K	18M	591M	281	143	sleeping	504	622	81
280	xpchelper	0.0	00:00.02	2	2	39	41	972K	220K	4468K	30M	2390M	280	1	sleeping	504	1401	168
278	mdworker	0.0	00:00.12	3	1	55	63	1392K	9564K	5584K	23M	2411M	278	142	sleeping	89	2424	202

En la part superior del terminal se visualitza la informació global del sistema, com el nombre de processos i tipus; la càrrega i utilització de la CPU, utilització de la memòria i discos. A continuació apareix un llistat de processos, que poden ser ordenats per ús de CPU o memòria, la qual cosa és una excel·lent ajuda per a detectar processos que consumeixen excessius recursos en el servidor. Aquest llistat, mostra atributs dels processos, com poden ser el PID de procés, usuari que ho executa, percentatge de CPU i memòria que consumeix, comando que està executant o temps d'execució del procés entre altres.

¡Atenció! Per a eixir de l'ordre **top**, cal teclejar q (quit).

2.3. Ordre kill

El comando **kill** (matar) ens permet enviar senyals (signals) als processos. L'acció per defecte que ocasiona el enviament d'un senyal a un procés, és acabar-lo o matar el procés; la sintaxi d'aquesta ordre és:

```
$ kill SIGNAL PID
```

On SIGNAL representa el senyal a enviar i PID el nombre de ID del procés al qual se li entrega el senyal. Obtinga el llistat de tots els senyals de la seua màquina, per a fer-ho execute:

```
$ kill -l
```

```
ivanovic:~ gandreu$ kill -l
1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGILL
5) SIGTRAP     6) SIGABRT     7) SIGEMT      8) SIGFPE
9) SIGKILL     10) SIGBUS     11) SIGSEGV    12) SIGSYS
13) SIGPIPE    14) SIGALRM    15) SIGTERM    16) SIGURG
17) SIGSTOP    18) SIGTSTP    19) SIGCONT    20) SIGCHLD
21) SIGTTIN    22) SIGTTOU    23) SIGIO      24) SIGXCPU
25) SIGXFSZ    26) SIGVTALRM  27) SIGPROF    28) SIGWINCH
29) SIGINFO    30) SIGUSR1    31) SIGUSR2
```

El senyal SIGKILL (9) provoca l'acabament del procés, aquest senyal no es pot emmascarar, manejar o ignorar.

2.Exercici: Llança l'ordre **yes** i finalitza el procés enviant el senyal SIGINT teclejant la combinació **ctrl-c**

```
$ yes
i
y
ctrl-c
```

Llança l'ordre **yes** en *background*, i esbrina el PID del procés, a continuació finalitza el procés i **yes** utilitzant l'ordre **kill**, després comprova amb **ps** que ja no existeixi el procés.

```
$ yes >/dev/null &
$ ps -la
$ kill -SIGKILL PID
```

En l'ordre **kill** permet referenciar els senyals pel seu nombre o nom, per exemple:

```
#> kill -9 11428      (termina el procés amb PID 11428)
#> kill -SIGKILL 11428 (El mateix que l'anterior)
```

El comando **killall** és similar a **kill SIGKILL** però indicant el nom del programa. **Killall** finalitza tots els processos amb el nom especificat.

3.Exercici: Executa la següent seqüència i comprova el funcionament de **killall**

```
$ ies >/dev/null &
$ ies >/dev/null &
$ ies >/dev/null &
$ ps -la
...
$ killall yes
$ ps -la
```

4. Exercici: En un terminal executa l'editor Kate en *background*. En el mateix terminal, executa l'ordre **ps** per a trobar el PID del procés que has iniciat. Utilitza l'ordre **kill** per a terminar el procés iniciat.

3. El directori /proc

En /proc el sistema manté tota la informació dels processos que es troben actualment en execució. Processant la informació del directori /proc podem extraure informació idèntica a la mostrada per **ps** o **top**.

NOTA: El directori /proc es propi d'alguns sistemes Unix, com Linux. Altres SO basats en Unix, com

Mac OS X, no generen la informació del sistema en aquest directori. Veure apèndix per a més informació.

El directori `/proc` conté un subdirector `/proc/$pid` per cada procés en execució i un conjunt d'arxius amb informació del sistema:

- **cpuinfo**: Conté informació al voltant del processador.
- **stat**: Conté informació del rendiment del processador: inclou informació al voltant del temps que duu el processador encès, el nombre de processos en execució, etc.
- **version**: Mostra informació al voltant de la versió de kernel, compilador gcc i debian... del nostre ordinador.

5. Exercici: Executa la següent seqüència per a comprovar el contingut de `/proc`. Verifica també que existeix un directori el nom del qual és el pid del teu Shell. Per a conèixer aquest pid execute l'ordre `ps` (substitueix `$pid_bash` pel PID del teu shell):

```
$ps
.....
$ls /proc
...
$ls /proc/$pid_bash
$more /proc/cpuinfo
```

En cada directori d'un procés (`/proc/$pid`) apareixen fitxers amb informació d'eixe procés. Alguns d'ells són:

- **/proc/\$pid/cmdline**: Conté nom del procés, incloent la ruta completa del mateix i els arguments.
- **/proc/\$pid/stat**: Es tracta d'un fitxer de text compostat de diversos camps, els quals es troben separats pel caràcter espai. Entre els distints camps tenim el Pid, el nom del procés (només el nom del procés, la ruta completa no), etc.
- **/proc/\$pid/maps**: Conté informació del mapa de memòria del procés.
- **/proc/\$pid/status**: Este fitxer conté informació relativa al procés com el Pid, PPid, estat del procés (línia State) i atributs relacionats amb el consum de memòria.

En la següent figura es mostra un exemple del contingut d'un fitxer status:

- **VmSize**: Indica el grandària total de l'espai virtual que ha sol·licitat el procés.
- **VmLck**: Indica l'espai ocupat per aquelles pàgines que han sigut bloquejades pel procés en el seu espai virtual. El bloqueig evita que aquestes pàgines puguin ser seleccionades com víctimes per l'algorisme de reemplaçament.
- **VmRSS**: Proporciona el "resident set size", és a dir, la grandària del conjunt de pàgines actualment carregat en memòria. En altres paraules, ens indica quanta memòria física està consumint actualment el procés.
- **VmData**: Indica la grandària virtual assignada a les dades del procés.
- **VmStk**: Indica la grandària virtual assignada a la pila.
- **VmExe**: Indica la grandària virtual assignada al codi del programa que executa el procés.
- **VmLib**: Indica la grandària virtual assignada a les biblioteques d'enllaç dinàmic que està utilitzant el procés.

```
gandreu@shell-sisop:~$ more /proc/$$/status
Name:   bash
State:  S (sleeping)
Tgid:   10956
Pid:    10956
PPid:   10955
TracerPid: 0
Uid:    1692409199      1692409199      1692409199      1692409199
Gid:    1692452651      1692452651      1692452651      1692452651
FDSize: 256
Groups: 1692441063 1692452651 1692453521 1692453528 1692459925 1692459927 169247
8937 1692515549
VmPeak: 21168 kB
VmSize: 21168 kB
VmLck:  0 kB
VmHWM:  4108 kB
VmRSS:  4108 kB
VmData: 2672 kB
VmStk:   88 kB
VmExe:   760 kB
VmLib:  1920 kB
VmPTE:   56 kB
Threads: 1
SigQ:   0/8191
SigPnd: 0000000000000000
ShdPnd: 0000000000000000
SigBlk: 0000000000010000
SigIgn: 0000000000384004
SigCgt: 000000004b813efb
CapInh: 0000000000000000
CapPrm: 0000000000000000
CapEff: 0000000000000000
Cpus_allowed: 03
Mems_allowed: 00000000,00000001
voluntary_ctxt_switches: 88
nonvoluntary_ctxt_switches: 59
gandreu@shell-sisop:~$
```

Informació exhaustiva d'aquests fitxers i altres, pot consultar-se usant el comando:

```
$ man proc
```

4. Ferramentes per a monitorització de processos

4.1. Ordre grep

L'ordre grep ens permet seleccionar informació d'un fitxer. Actua com un filtre que busca línies que continguin una cadena. Per exemple:

```
$grep hola fitxer //mostra les línies que contenen la cadena "hola" en el fitxer "fitxer"
$grep -i Hola fitxer //mostra les línies amb "hola" del "fitxer" ignorant majúscules/minúscules
$ls -l | grep pract3 //mostra les línies amb pract3 en la seqüència resultant d'executar ls -l.
```

6.Exercici: *Escriu una línia d'ordres que mostre per pantalla la freqüència dels processadors del seu sistema. Es tracta d'accedir al fitxer /proc/cpuinfo i utilitzar el grep per a seleccionar la línia adequada.*

7.Exercici: *Escriba una línia d'ordres que mostre per pantalla la grandària de memòria cache dels processadors del seu sistema. Es tracta d'accedir al fitxer /proc/cpuinfo i utilitzar el grep per a seleccionar la línia adequada.*

4.2. Shell script

El shell o intèrpret d'ordres de LINUX (p.e., bash) és un programa que s'executa a nivell d'usuari, que està disponible en qualsevol entorn LINUX, l'objectiu del qual és llegir línies d'ordres, analitzar-les i executar-les realitzant les crides al sistema necessàries.

Aquest intèrpret defineix, al igual que ocorre amb qualsevol altre intèrpret o traductor, un llenguatge de programació propi que posseeix característiques com:

- Procediments, més coneguts com a Shell scripts
- Paraules i caràcters reservats (també coneguts com a metacaràcters)
- Variables
- Estructures per al control del flux tipus `if`, `while`, etc.
- Maneig d'interrupcions

Realment, en un sistema LINUX existeixen diversos intèrprets d'ordres, podent triar cada usuari el que prefereixi. L'intèrpret d'ordres Bourne shell o `sh` és el bàsic, que està disponible en qualsevol sistema. Existeixen altres shells, com el Korn shell o `ksh`, el shell amb sintaxi del tipus de llenguatge C o `csh` i el Bourne-Again SHell o `bash`, que és una versió estesa del `sh`. Aquesta pràctica introdueix les característiques i la sintaxis del `sh`, les quals són aplicables al `ksh` i al `bash`.

Les ordres a executar pel shell poden ser proporcionades des de teclat com un programa contingut en un fitxer. Es denomina **shell script** a un fitxer de text que conté ordres per a ser executades pel shell. Per exemple, es pot crear un fitxer que continga:

```
#!/bin/bash
# content
Num_process=$(ls -d /proc/[1-9]*|wc -l)
echo Number of System process is: $Num_process
```

Les línies que comencen pel caràcter `#` són comentaris. La resta són sentències que executarà el shell. En particular, l'ordre `echo` mostra allò que se li indica per pantalla. L'ordre `ls` serveix per a obtenir un llistat dels fitxers directoris, mentre que "`wc -l`" compta el nombre de línies.

8. Exercici: Crea el fitxer anterior utilitzant un editor de text i anomena-ho "content". A continuació executa-lo donant-lo els permís adequats:

```
$ chmod +x content
$ ./content
```

Observa que amb `./content` se li està indicat al shell que el fitxer a executar es troba en el directori actual de treball (directori "."). Per a no haver d'indicar cada vegada on es troba el fitxer a executar (indicant la ruta completa), consulta l'annexe1 d'aquesta pràctica.

Un shell script pot invocar-se amb arguments, els quals poden ser referenciats com `$0`, `$1`, `$2`, `$3`, etc... L'argument `$0` referència el propi nom del programa, en tant que `$@` referència tots els arguments i `$#` indica el nombre d'arguments.

9.Exercici: Crea un arxiu denominat arguments amb el següent contingut

```
#!/bin/bash
```

```
# arguments
echo El nombre d'arguments es: $#
echo L'ordre teclejada es: $0
echo El primer argument: $1
echo El segon argument: $2
echo El tercer argument: $3
```

A continuació execute arguments amb diferents paràmetres

```
$/arguments un dos tres
```

```
$ ./arguments FSO TCO ESO
```

4.3. Variables

Una variable es crea utilitzant el signe d'igualtat "=" per a assignar l'identificador al contingut, és a dir, identificador=contingut. Per a accedir al contingut d'una variable utilitzem \$ davant del seu identificador. Exemples:

```
$ name=Alberto
$ subject=FSO
$ msg="Hello World"
$
$ echo $name
Alberto
$ echo $subject
FSO
$ echo $msg
Hello World
```

¡Compte! si deixem espais entre el = i l'identificador o el valor, el shell creurà que són comandos a executar i no l'assignació d'una variable. Per a accedir al contingut d'una variable utilitzem \$ davant del seu identificador:

4.4. Bucles for

La sentència *for* itera sobre una llista de valors i assigna en cada iteració un valor a la variable associada. La seua sintaxis és la següent:

```
for variable in llista de valors
do
sentències
done
```

Podem utilitzar el bucle *for* per a llistar el nombre d'arguments de l'exemple anterior:

```
#!/bin/bash
echo El nombre d'arguments es: $#
echo L'ordre teclejada es: $0 $@
echo Llista de arguments:
for i in $@
do
echo $i
done
```


La llista de valors d'un bucle *for* també poden ser els fitxers del directori actual. Per exemple, el següent programa crea una còpia de seguretat de cadascun dels fitxers del directori actual:

```
for k in *
do
cp $k $k.bak
echo Creada copia de $k
done
```

Finalment la llista de valors d'un bucle *for* també pot provenir de l'execució d'una ordre mitjançant l'ús de `$()`. Per exemple:

```
for i in $(ls)
do
echo $i
done
```

4.5. Sentència if

La sentència *if* permet l'execució condicional d'ordres. La seua sintaxi és:

```
if ordre s'executa amb èxit
then
sentencies
else
sentencies alternatives
fi
```

Observa que la condició de la sentència *if* no és una expressió sinó una ordre de LINUX. La condició és certa si l'ordre "acaba correctament", (en aquest cas s'executen les sentències que segueixen el *then*) i falsa en cas contrari (en aquest cas s'executen les sentències que segueixen el *else*). La clàusula *else* és opcional.

4.6. Ordre test

L'ordre **test** permet avaluar condicions i, per tant, resulta de gran utilitat per a utilitzar-la conjuntament amb la sentència *if*. Els tipus d'expressions que pot avaluar l'ordre *test* són les següents:

Expressions numèriques. La forma general de les expressions és:

$N <\text{primitiva}> M$

on N i M són interpretats com valors numèrics. Les primitives que se poden utilitzar són:

-eq	N i M són iguals.
-ne	N i M són distints.
-gt	N es major que M.
-lt	N es menor que M.
-ge	N es major o igual que M.
-le	N es menor o igual que M.

10.Exercici. Edita un fitxer anomenat "*my_process*" amb els següents comandos i executa-lo:

```
#!/bin/bash
# my_process
```

```
process=$(ps u | grep $USER | wc -l)
if test $process -gt 2
then
echo "More than 2 user process"
else
echo "Few process"
fi
```

Expressions alfanumèriques. Sean S i R cadenes alfanumèriques. Podem tenir dos tipus d'expressions:

<primitiva> S
S <primitiva> R

Les primitives que es poden utilitzar són:

S=R	les cadenes S i R són iguals.
S != R	les cadenes S i R són distintes.
-z S	comprova si la cadena S té longitud zero.
-n S	comprova si la cadena S te una longitud distinta de zero

Tipus de fitxers. La forma general de les expressions és:

<primitiva> fitxer

Les primitives que se poden utilitzar són:

-s	comprova que el fitxer existeix i no està buit.
-f	comprova que el fitxer existeix i es regular (no directori).
-d	comprova si el fitxer es un directori.
-r	comprova si el fitxer té permís de lectura.
-w	comprova si el fitxer té permís d'escriptura.
-x	comprova si el fitxer té permís d'execució.

11.Exercici: Realitza un script *esborrar_fitxer*, al qual se li passa un argument amb el nom d'un fitxer. Aquest script ha de comprovar que el fitxer existeix i que no és un directori. En el cas de que no existeixi es traurà un missatge de text per pantalla, en el cas de que siga un fitxer regular, s'esborrarà aquest fitxer i en el cas de que siga un directori es traurà un missatge indicat que és un directori i que no es pot esborrar.

4.7. Ordre awk

L'ordre *awk* és una ferramenta del sistema UNIX útil per a modificar arxius, buscar i transformar conjunts de dades, generar informes simples, etc. Donat que és un llenguatge de programació, *awk* té una gran varietat de possibilitats i en aquesta secció només s'il·lustra el seu ús més bàsic.

La funció bàsica de *awk* és buscar línies en fitxers u altres unitats de text que contenen certs patrons. Quan en una línia es troba un patró, *awk* realitza les accions especificades per a aquest patró sobre la línia. *Awk* segueix realitzant el processament de les línies d'entrada d'aquesta forma fins que arriba al final del fitxer.

4.7.1. Sintaxi

Awk se li ha d'indicar quins patrons cerquem, quines accions ha de realitzar sobre les línies que continguem els patrons i com adquireix les línies d'entrada. Els patrons i accions se li indiquen mitjançant un programa.

```
$ awk 'programa' fitxer de entrada 1 fitxer de entrada 2 ...
```

Aquest format indica al shell que execute *awk* i use programa per a processar les línies dels fitxers d'entrada fitxer_de_entrada_1 fitxer_de_entrada_2 ...

Quan el programa és llarg és millor ficar-ho en un fitxer i executar-ho de la següent manera:

```
$ awk -f prog-file fitxer_de_entrada_1 fitxer_de_entrada_2 ...
```

en aquest cas prog-file és un fitxer que conté el programa.

Un programa en *awk* consisteix en una sèrie de regles. Cada regla especifica un patró a cercar, i una acció a realitzar quan es trobe el patró en el registre d'entrada. L'acció es tanca entre claus per a separar-la dels patrons.

```
patró { acció } patró  
{ acció }  
...  
patró { acció }
```

4.7.2. Patrones

En els patrons se poden utilitzar expressions regulars tancades entre barres diagonals /. L'expressió regular més simple és una seqüència de lletres, nombres o ambdós. Per tant, el patró /root/ es correspon amb qualsevol registre que continga la cadena root.

4.7.3. Acciones

L'acció dona lloc a que alguna cosa passe quan un patró concorda. Si no s'especifica una acció, *awk* suposa {print}. Aquesta acció còpia el registre (sol ser una línia) del fitxer d'entrada en la sortida estàndard. La instrucció print pot anar seguida d'arguments fent que *awk* imprimeixi només els arguments. A continuació es mostraran exemples de l'ús de *awk*.

Si es genera el fitxer prova.txt del següent mode:

```
echo -e "Col1\tCol2\tCol3\tCol4\n" > exemple.txt
```

Y es visualitza amb l'ordre cat:

```
cat exemple.txt
```

Retornarà el següent contingut:

```
Col1    Col2    Col3    Col4
```

Si s'utilitza *awk* per a que només mostre la columna 1 i la columna 3 del següent mode:

```
awk '{ print $1, $3}' exemple.txt
```

La sortida retornarà el següent:

```
Col1 Col2
```

Si s'afegeixen dades al fitxer *exemple.txt* del següent mode:

```
echo -e "Dato1\tDato2\tDato3\tDato4\n" >> exemple.txt  
echo -e "Dato5\tDato6\tDato7\tDato8\n" >> exemple.txt
```

```
echo -e "Dato9\tDato10\tDato11\tDato4\n" >> exemple.txt
```

Y es visualitza amb l'ordre cat:

```
cat exemple.txt
```

Retornarà el següent contingut:

```
Col1    Col2    Col3    Col4
Dato1    Dato2    Dato3    Dato4
Dato5    Dato6    Dato7    Dato8
Dato9    Dato10   Dato11   Dato4
```

Si s'utilitza novament *awk* per a que només mostre la columna 3 i la columna 1(en eixe ordre) del següent mode:

```
awk '{ print $3, $1}' exemple.txt
```

La sortida retornarà el següent:

```
Col3 Col1
Dato3 Dato1
Dato7 Dato5
Dato11 Dato9
```

Si s'utilitza l'ordre *awk* de la següent manera per a que mostre només la línia la columna de la qual continga l'expressió regular Dato5:

```
awk '/Dato5/ { print }' exemple.txt
```

La sortida retornarà lo següent:

```
Dato5    Dato6    Dato7    Dato8
```

Si s'utilitza l'ordre *awk* de la següent manera per a que mostre només la línia la columna de la qual continga l'expressió regular Dato5, i a més només les columnes 1 i 4:

```
awk '/Dato5/ { print $1, $4}' exemple.txt
```

La sortida retornarà lo següent:

```
Dato5 Dato8
```

12. Exercici: Cerca dins del fitxer */etc/passwd* la cadena de caràcters *root*, i imprimeix per pantalla la línia on s'ha trobat la cadena.

13. Exercici: Realitza un Shell script denominat *inf_process* que prengui com a argument el pid d'un procés e imprimeixi per pantalla en format de columnes el PID, PPID, ESTAT i COMANDO que executa el procés. Recorde que aquesta informació la pots trobar en els fitxers */proc/\$pid/status* i */proc/\$pid/cmdline*. La sortida ha de ser del tipus:

```
PID    PPID    ESTAT    COMANDO
8900   8880    S        /bin/bash
```

14. Exercici: Realitza un Shell script denominat *system_process* que proporcione al scripts *inf_process* tots els PID dels processos del sistema, per a que imprimeixi la seua informació.

4.7.4. Dividint l'entrada en registres i camps

L'entrada en el programa *awk* típic es llig en unitats anomenades registres, i aquests són processats per les regles un a un. Per defecte, cada registre és una línia del fitxer d'entrada.

El llenguatge *awk* divideix els seus registres de entrada en camps. Per defecte assumeix que els camps estan separats per espais en blanc. El mode en que *awk* divideix registres d'entrada en camps es controlada pel separador de camps, el qual és un caràcter simple o una expressió regular. *Awk* recorre el registre d'entrada en cercant les coincidències del separador de camps; els camps són el text que es troba entre els separadors de camps trobats. Per exemple, si el separador de camp és ':', aleshores la següent línia:

```
patricia:x:1000:1000:Patricia Balbastre Betoret:/home/patricia:/bin/bash
```

serà partionada en 7 camps:

```
'patricia', 'x', '1000', '1000', 'Patricia Balbastre Betoret', '/home/patricia' i '/bin/bash'.
```

El separador de camps pot ser fixat en la línia d'ordres utilitzant l'argument '-F'. Per exemple:

```
$ awk -F: 'programa' fitxers_de_entrada
```

Fixa com separador de camps el caràcter ':'.
.

Per a referir-se a un camp en un programa *awk*, s'utilitza el símbol \$ seguit pel nombre del camp. Per tant \$1 es refereix al primer camp, \$2 al segon i així successivament. \$0 és un cas especial, representa el registre de entrada complet.

Altre exemple, si escriguerem:

```
$ awk -F: '/model name/ {print $2}' /proc/cpuinfo
```

S'imprimiria el segon camp separat per ':' de les línies que contingueren `model name` del fitxer `/proc/cpuinfo`. Cal destacar que l'argument F: funciona indistintament amb i sense cometes.

Exercici Optatiu: Llança 5 processos *kate*, alguns en primer pla i altres en segon pla. Realitza un script que, cercant la informació en el directori `/proc` mostre per pantalla el nombre de processos *kate* en execució a més del seu PID, el seu estat i la memòria que està utilitzant el procés. Intenta generalitzar aquest script de recerca per a que pugui introduir-se un paràmetre amb el nom del procés a cercar.

Per a això hauràs d'utilitzar els coneixements apresos en aquesta pràctica del Shell script i *awk*.

5. ANEXO. Descripció d'ordres i filtres

5.1. Variable PATH

Per a no haver d'indicar cada vegada on es troba el fitxer a executar (indicant la ruta completa), el shell utilitza la variable d'entorn denominada PATH. Aquesta variable conté la llista de noms de directoris (separats pel caràcter ":") on el shell buscarà els fitxers executables.

Consulta el valor de la variable PATH executant l'ordre echo:

```
$echo $PATH
```

Comprova si el PATH que està utilitzant el shell conté el directori actual de treball ("."). En cas afirmatiu execute el fitxer contingut com:

```
$contingut
```

En cas negatiu, ha d'incloure-lo executant l'ordre:

```
$PATH=$PATH: .
```

Aquest canvi només durarà fins que finalitzi l'execució del shell (o siga, es tanque la terminal). Si desitja que el canvi siga permanent edita el fitxer de configuració \$HOME/.bashrc i afegeixi al final l'ordre anterior.

Exercici Annex1: Edita el fitxer de configuració \$HOME/.bashrc i afegeixi al final l'ordre "PATH=\$PATH: ."
Una vegada afegit, obri un nou terminal i comprova que el canvi s'ha aplicat.

```
$xemacs $HOME/.bashrc
```

```
$ echo $PATH
```

5.2. Operacions aritmètiques

Per a que el Shell avalue una operació aritmètica:

```
$((expressió))  
avalua l'expressió aritmètica i reemplaça el bloc pel resultat
```

Per exemple:

```
$ echo 1+1
```

```
1+1
```

```
$ echo $((1+1))
```

```
2
```

Alguns operadors aritmètics suportats:

```
+ suma  
* multiplicació  
- resta  
/ divisió entera  
% reste de la divisió sencera  
( ) agrupar operacions
```

5.3. Bucles while

Es tracta d'una altra estructura de bucle que permet executar un bloc de comandos mentre s'avalua una condició d'encert:

```
while CONDICIO; do  
    bloc de comandos
```

```
done
```

Cada iteració s'avalua la condició i en el moment que no siga certa, el bucle termina. Exemples de bucles:

```
# equivalent a seq 1 5
i=1
while [ $i -lt 6 ]; do
    echo $i
    i=$((i+1))
done

# llig de stdin fins que s'introdueix 'quit'
read linia

while [ "$linia" != "quit" ]; do
    read linia
done
```

5.4. Awk, els patrons especials BEGIN i END

BEGIN i END són patrons especials. S'utilitzen per a subministrar al awk què fer abans de començar a processar i després d'haver processat els registres d'entrada. Una regla BEGIN s'executa una vegada, abans de llegir el primer registre d'entrada. I la regla END una vegada després de que se hagen llegit tots els registres d'entrada. Per exemple, el següent mandat especifica que a l'inici s'imprimeixi en la sortida la frase "Hola mon" i acabar el processament.

```
awk 'BEGIN { print "Hola mon"; exit }'
```

L'ordre anterior haurà de tornar una sortida com la següent:

```
Hola mon
```

Altre exemple:

```
$ awk 'BEGIN {print "Quantes vegades apareix Dato4" ; dato=0 ; }
>/Dato4/ { ++dato }
>END {print "Dato4 apareix " dato " vegades"}' exemple.txt
```

Aquest programa esbrina quantes vegades apareix la cadena "Dato4" en el fitxer exemple.txt generat en exemples anteriors. La regla BEGIN imprimeix un títol per al informe i inicialitza el comptador dato a 0, tot i que no haurà necessitat ja que awk ho fa per nosaltres automàticament. La segona regla incrementa el valor de la variable dato cada vegada que es llig de la entrada un registre que conté el patró D a t o 4 '. La regla END imprimeix el valor de la variable al final de

5.5. Awk, la sentència IF

La sentència if-else es una sentència per a la presa de decisions de awk. Presenta la següent forma:

if (condició) cos-then [else cos-else]

on condició és una expressió que controla què es allò que realitzarà la resta de la sentència. Si la condició es veritat, aleshores s'executa el cos-then; sinó s'executa cos-else (assumint que estiga present la clàusula else). La part else de la sentència és opcional. La condició es considera falsa si el seu valor és 0 o una cadena nul·la, sinó es considerarà veritable.

Ací se presenta un exemple:

```
$ awk '{ if ($0 % 2 == 0)
> print $0 "es par"
> else
> print $0 "es impar" }'
```

En este exemple, si l'expressió `$0%2==0` es certa (es a dir, el valor que se li passa és divisible entre 2), aleshores s'executa la primera sentència *print*, sinó s'executa la segona sentència *print*.

Si el *else* apareix en la mateixa línia que el *cos-then*, i el *cos-then* no és una sentència composta (no apareix entre claus) aleshores un punt i coma ha de separar el *cos-then* del *else*. Per a il·lustrar això anem a veure el següent exemple.

```
$ awk '{ if ($0 % 2 == 0) print $0 "es par"; else print $0 "x es impar"}'
```

Si s'oblida el `;` provocarà un error de sintaxis. Es recomana no escriure la sentència *else* d'aquesta forma, ja que pot dur al lector a confusió.

Exercici Annex5: Utilitza el que has après fins per a fer un script que mate el procés Kate (de diversos que hages llançat prèviament) de menor PID.

5.6. Ordre TR

Aquesta ordre permet canviar o traduir els caràcters procedents de l'entrada d'acord a regles que s'especifiquen. El formato general és:

```
$ tr [opcions] cadena_1 cadena_2
```

Exemples d'utilització d'aquesta ordre són:

Per a canviar un caràcter per altre: per exemple, el utilitzat com separador entre camps d'un arxiu (':') amb altre (per exemple, el tabulador):

```
$ tr : '\t' < nom_fich_entrada
```

Per a canviar un conjunt de caràcters: per a ficar en majúscules tots els caràcters que apareixen en un arxiu:

```
$ tr '[a-z]' '[A-Z]' < nom_fich_entrada
```

Substituir els caràcters nuls que conté un fitxer per blancs:

```
$ cat nom_fich_entrada | tr "\000" " " > nom_fich_salida
```

Per exemple:

```
tr 'Dato4' 'Dato5' < exemple.txt
```

Exercici Annex6: Substitueix la cadena *Dato4* per *Dato5* en tot l'arxiu *exemple.txt*.

6. ANNEX. Consulta de informació de sistema en Mac OS X.

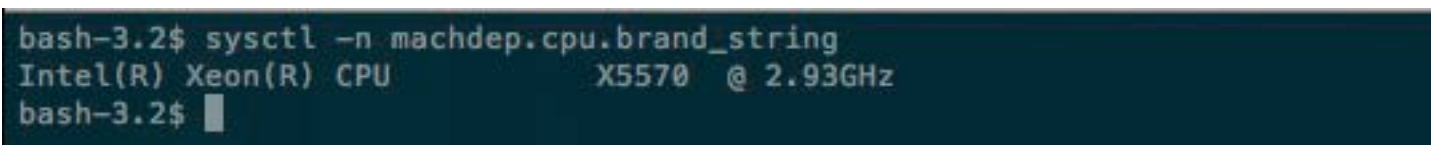
A continuació es descriu breument com es pot consultar la informació del sistema en Mac OS X. A part de les aplicacions Informació del Sistema i Monitor d'Activitat que proporcionen gràficament tota la informació en temps real dels processos i del sistema, també se pot consultar la informació per línia de comandos.

En LINUX la informació de la CPU es troba en l'arxiu `/proc/cpuinfo`. Per exemple per a mostrar el nom de la CPU en Linux es fa:

```
$ cat /proc/cpuinfo | grep "model name"
```

En OSX, s'utilitza el comando **sysctl**, per a consultar determinats aspectes de l'estat del kernel de OSX. Per exemple, per a obtenir el nom en OSX es faria:

```
$ sysctl -n machdep.cpu.brand_string
```



```
bash-3.2$ sysctl -n machdep.cpu.brand_string
Intel(R) Xeon(R) CPU X5570 @ 2.93GHz
bash-3.2$
```

I per a obtenir la informació completa en OSX:

```
$ sysctl -a | grep machdep.cpu
```

Si desitja obtenir el nombre de nuclis i fils, es pot utilitzar les següents ordres:

```
$ sysctl -a | grep machdep.cpu | grep core_count
$ sysctl -a | grep machdep.cpu | grep thread_count
```