M1 – Implantació de Sistemes Operatius

Unitat Didàctica 3

Administració de Software Base

Raül Sala / José Luis Antúnez – 2017/2018

Activitat 7

Programes i processos

Descripció de processos i els estats d'aquest, així com les transicions entre els diversos estats.

Programes i processos

- **Programa**: seqüència d'instruccions, definides a priori, que poden ser executades per un processador.

- **Procés**: programa (o programes) en execució.

- Els ordinador realitzen diverses tasques alhora:
  - en un cert instant, cada CPU només pot executar una instrucció, així que fa canvis ràpids, imperceptibles, entre les diverses tasques que executa → **multiprogramació**.
  - això crea una falsa il·lusió de paral·lelisme: el **pseudoparal·lelisme**.

Requisits del sistema operatiu

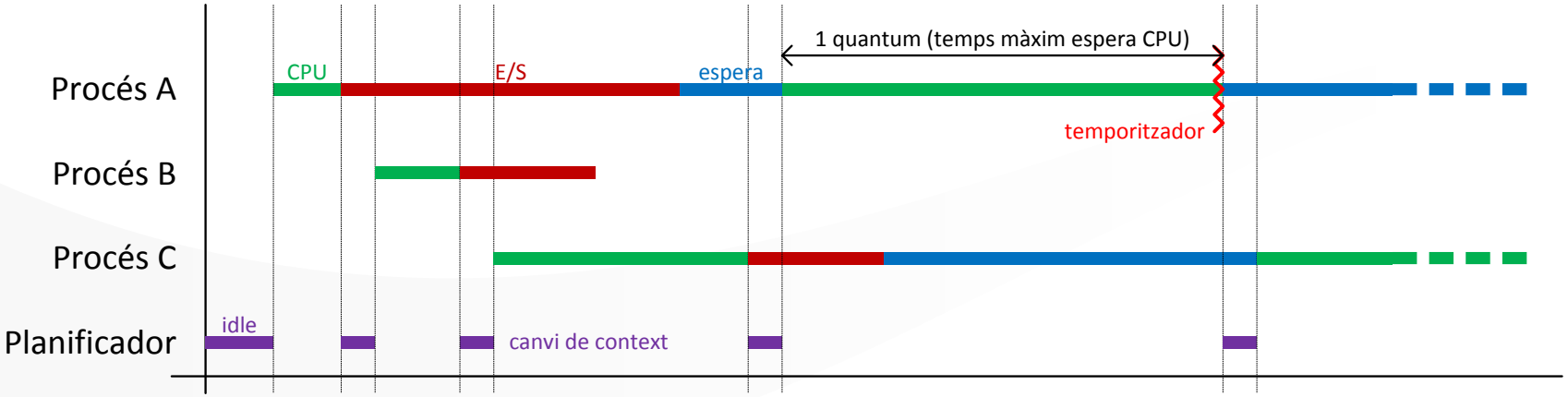
- Per a gestionar correctament els processos, el sistema operatiu:
  - Ha de maximitzar la utilització del processador: màxim de processos, mínim temps.
  - Ha de tenir una política d'assignació de recursos als processos.
  - Ha de permetre la comunicació de processos.

Pare de família

- Un pare de família està cuinant un pastís, del qual en té la recepta.
  - La recepta és el **programa**.
  - El pare és el processador.
  - Els ingredients són les dades d'entrada.
  - El **procés** és l'activitat de llegir la recepta, buscar els ingredients i cuinar el pastís.
- Imaginem que al seu fill li ha picat una abella i entra corrents i plorant a casa.
  - El pare deixa la recepta en el punt on s'ha quedat.
  - Busca un llibre de primers auxilis i comença a **seguir les instruccions** d'aquest.
  - El processador ha **alternat** d'un procés a un altre de **més prioritat**, cadascun amb un programa totalment diferent.

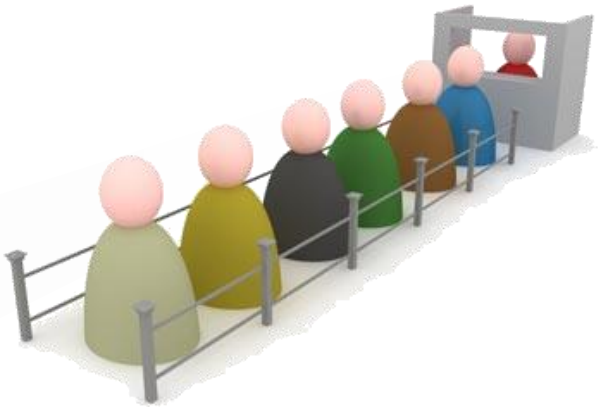
Planificador de processos

- La missió del processador és **executar les instruccions** de la màquina que resideixen a la memòria principal.
  - El processador intercala l'execució de processos per raons d'eficiència. El **planificador de processos** ho gestiona:
    - Aprofita que els temps d'E/S són molt més alts que els d'execució de la CPU.
    - Si un procés porta massa temps executant-se (1 quantum) salta el temporitzador per a que no monopolitzi el processador.
    - Les accions necessàries per a passar d'un procés a un altre s'anomenen ***canvi de context***.
    - Si en un moment no hi ha res executant-se, s'executa el ***procés inactiu (idle)***.



Prioritats de processos

- No totes les tasques tenen les mateixes exigències pel que fa al temps i disponibilitat de recursos: aquesta gestió es fa mitjançant les **prioritats dels processos**.
- Les prioritats poden ser assignades pel propi **sistema operatiu** o pel **propietari** de la tasca.
- La prioritat d'un procés **no es fixa** al llarg del temps, i una de les operacions més comunes del sistema operatiu és el canvi de prioritat d'un procés.



Processos a Unix → El Process identifier (PID)

- A Unix tot procés s'identifica amb un número únic, el PID.
- L'encarregat de gestionar els processos és el **kernel** i disposa d'una taula de processos on cada procés es identificat pel seu PID.
  - **Una aplicació no pot crear un procés directament!.** Es necessari demanar al sistema operatiu que crei el procés i el gestioni. Es disposa de dos crides de sistema:
  - **fork:** permet que un procés crei un altre procés. Aquest nou procés es anomenat procés fill
  - **exec:** permet executar una aplicació externa. Substitueix el procés en execució per un altre procés.

Tipus de processos

- **Procés fill:** Jerarquia de processos. Creats per fork.
- **Procés orfe:** el pare ha finalitzat la seva execució abans que el fill.
- **Procés zombie:** Procés que ha acabat però que resta a l'espera d'una resposta del pare.
- **Procés parat:** es poden parar processos, pausa (Ctrl+z).
- **Procés dimoni (servei):** s'executa permanentment i en segon terme (background). No té interfícies d'usuari associades.

Jerarquia de processos Unix

- **Unix/Linux**
  - Els processos només es poden executar a partir de processos ja existents
  - Els processos acabin tenint una jerarquia (relacions pare/fill).
  - La crida de sistema que permet en Unix crear un nou procés és fork (forquilla). Fork crea un procés fill que és una còpia casi exacta del procés pare i que s'executa en paral·lel al procés pare
- **Dos conceptes:**
  - Herència: El procés hereta totes les propietats del pare (de fet és una còpia casi exacta del seu pare).
  - Sincronització entre processos pare i fill: Mecanismes IPC com senyals.

Procés init

- **El primer procés en executar-se és el nucli (kernel)**
  - És executat pel gestor d'arrancada
  - El procés del nucli que gestiona tota la resta de processos s'anomena **scheduler (planificador de processos). No el podeu visualitzar (té el PID 0)**

- **Procés init**
  - És el procés pare de tota la resta de processos. PID 1
  - Si un procés queda sense pare (orfe) l'init l'adoptarà.

Comanda `pstree`

- **Executeu:** `pstree –p | head`
  - També hi ha l'opció: `$ ps --forest`

- **Permet visualitzar la jerarquia de processos:**
  - Localitzeu l'interpret d'ordres on heu executat pstree
  - Executeu: `$ sleep 10&`
  - Torneu a executar pstree
  - Vegeu com el procés sleep és fill de bash i germà de pstree

Creació de processos

- **Primer pla (foreground – fg):** és el mode per defecte amb el qual executem ordres a l'interpret d'ordres. Les ordres bloquejen l'execució de l'interpret
  - `$ sleep 10`

- **Segon pla (background – bg): es pot fer amb el símbol & (o l'ordre bg)**
  - `$ sleep 10 &`
  - Ens proporciona el PID
  - Es per poder gestionar el procés
    - p. ex. Matar el procés amb kill

Control i monitorització de tasques

- **`ps`**: mostra la taula de processos del terminal i usuari actual
- **`jobs`**: mostra els processos que s'estan executant a l'interpret de comandes (dóna un número de tasca a cadascuna)
- **`top`**: mostra els processos ordenats per percentatge d'ús de CPU (de més a menys) i de forma continuada.
- **`fg`**: passa un procés de segon terme a primer terme (amb un número de tasca/job)
- **`bg`**: passa un procés de primer terme a segon terme (amb un número de tasca/job)

Matar processos

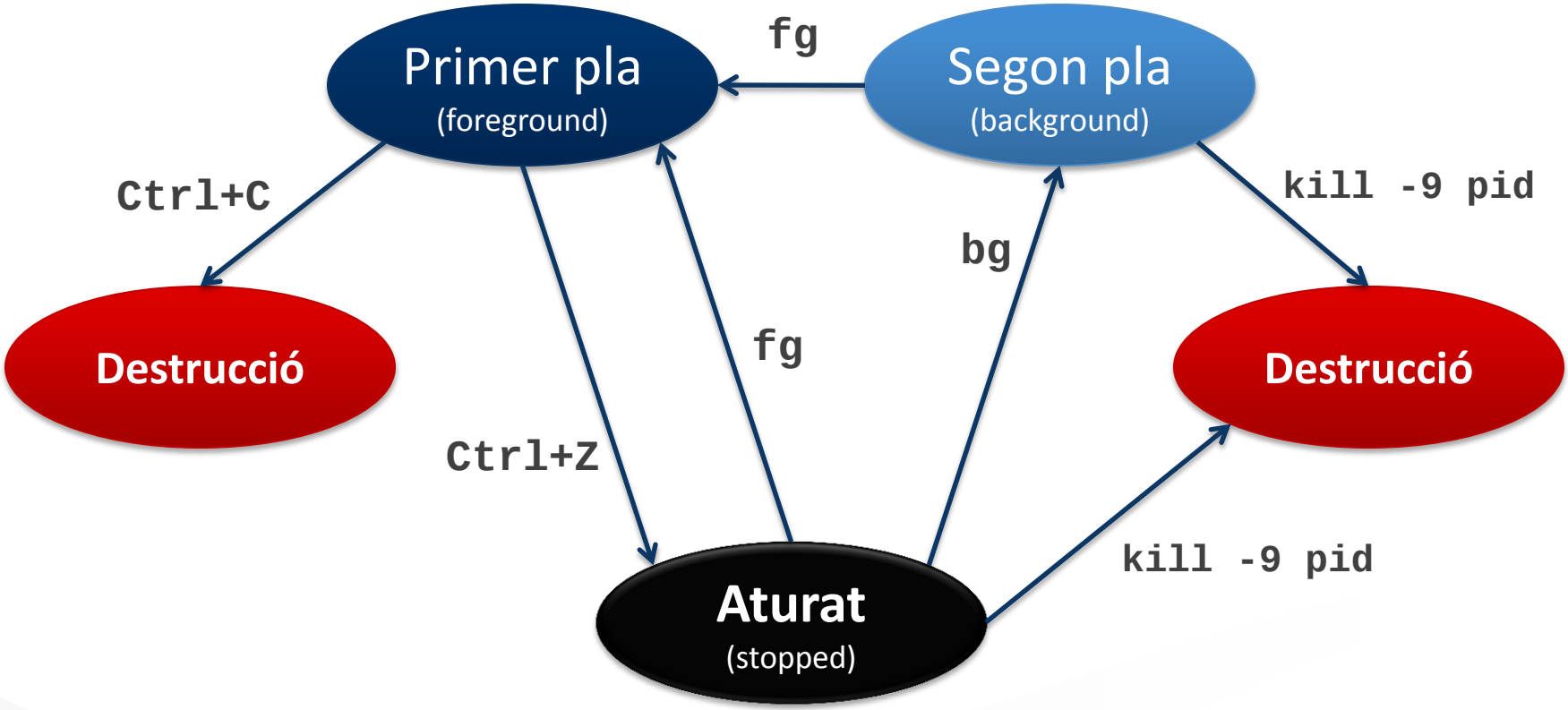
- 3 formes d'acabar prematurament un procés que s'està executant en **primer pla:**
  - **`Ctrl-c`:** Envia el senyal 2 (SIGINT). La majoria d'aplicacions estan programades per finalitzar l'execució quan reben aquesta senyal. Algunes comandes no ho fan així ho requereixen d'una doble confirmació abans de finalitzar el procés)
  - **`Ctrl+\`:** S'envia el senyal 3 (SIGQUIT). També depèn de com estigui programada l'aplicació.
  - Utilitzar **`kill`**
- L'única forma de finalitzar un procés que s'executa en segon pla és utilitzar la comanda **`kill`**.

- Només podem matar processos nostres o qualsevol procés si som root.
  - **`kill -9 250`** (mata un procés pel seu PID)
  - **`kill -9 -1`** (mata tots els processos)
  - **`killall mozilla`** (mata un procés a partir del seu nom)

La comanda `kill`

- Quan executem la comanda kill, realment no matem un procés, si no que li enviem un senyal determinat.
  - El sistema disposa de 64 senyals que s'identifiquen per un número o un nom simbòlic (SIG+Nom_del_senyal).
  - Cada procés pot adaptar tots els senyals excepte 2:
    - **`SIGKILL (9)`:** permet finalitzar un procés de forma abrupta.
    - **`SIGSTOP (19)`:** permet aturar l'execució d'un procés.
  - A més, hi ha altres senyals habitualment utilitzats:
    - **`INT (2)`:** Interrupt. És enviada quan fem CTRL+c. Els programes poden decidir què fan en rebre aquest senyal.
    - **`TERM (15)`:** Terminate. Finalitza el procés de forma controlada (per exemple, demanaria guardar els canvis). És el que executa kill per defecte i el que s'envia als programes quan finalitzem el sistema.
    - … **`kill -l`** ens mostra la taula completa de signals amb els seus noms.

Estats d'un procés Unix



Prioritats dels processos – valors `nice`

- Indiquen la prioritat d'un procés. **Rang: -20 a 19**
  - Valor més alt de prioritat és -20 (màxima prioritat)
  - Valor més baix de prioritat és 19(mínima prioritat).
  - La prioritat per defecte és 0
- Els processos amb més prioritats utilitzen més recursos. Els processos amb prioritat mínima (+19), només s'executaran quan el sistema no executa cap altra tasca.
- Els usuaris, només poden modificar els processos dels quals són propietaris en un interval de **0 a 19.**
- L'usuari root, pot canviar la prioritat de qualsevol procés a qualsevol valor.

Treballar amb la prioritat dels processos

- **`nice`**: permet executar un procés amb una prioritat concreta
  - `nice –n 15 gedit`

- **`renice`**: permet modificar la prioritat d'un procés en execució
  - `renice +15 785` (modifica la prioritat del procés amb PID=785)
  - `renice +19 –u estudiant` (com a superusuari, modifica la prioritat dels processos d'un usuari determinat)

“A Unix, tot són fitxers”

- El sistema de fitxers muntat a /proc mostra tota la informació respecte als processos que hi ha en execució.
  - Si us situeu al directori i feu ls veureu un seguit de fitxers i directoris.
  - Els fitxers ofereixen informació general sobre la màquina
    - `cpuinfo`
    - `meminfo`
    - `version,...`
  - Cada procés està representat per un directori (amb el PID del procés)
  - Dins d’aquests directoris hi ha informació general sobre el procés:
    - La línia de comandes que l’ha creat: `cmdline`
    - Les variables d’entorn: `environ`
    - L’estat: `status`
- Si executeu `man proc` podeu veure informació sobre aquest sistema de fitxers.

Comunicació entre processos

- Els diversos processos que s'executen a la màquina es poden comunicar, els uns amb els altres, mitjançant diversos mecanismes.

- En aquesta segona part de la presentació veurem com funcionen aquests mecanismes de comunicació i algunes de les eines que més habitualment s'utilitzen en aquest àmbit.

Concatenació d'ordres

- **Tenim diferents operadors per executar múltiples ordres en una mateixa línia:**
  - **Operador AND:** només s'executa la comanda 2 si la primera comanda s'ha executat amb èxit.

```
$ mkdir directori && cd directori
```

  - **Operador OR:** només s'executa la comanda 2 si la primera comanda no s'ha executat amb èxit.

```
$ mkdir directori1 || mkdir directori2
```

  - **Llistat d'ordres:** per a executar diverses ordres de forma consecutiva, en una mateixa línia, les separem amb punts i comes.

```
$ mkdir directori3; mkdir directori4
```

Flux de dades

- **Flux de dades (stream)**
  - És un conjunt d'informació que es desplaça d'un origen a una destinació
  - Els flux de dades són bàsics per tal de realitzar tasques complexes combinant programes senzills.
  - Internament, el sistema operatiu els tracta com si fossin fitxers.

- **3 fluxos de dades estàndard a Linux** (n'hi poden haver més)
  - **Entrada estàndard (stdin):** És el flux d'entrada de les ordres i aplicacions. **Habitualment és el teclat.**
  - **Sortida estàndard (stdout):** És el flux de sortida de les ordres i aplicacions. **Normalment és la terminal.**
  - **Error Estàndard (stderr):** Linux proporciona un segon tipus de flux de dades de sortida. La idea és tenir una sortida per aquelles dades amb alta prioritat, com per exemple els missatges d'error. **Per defecte és la terminal.**

La filosofía de Linux

> Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.
>
> Doug McIlroy, creador del conductes Unix

- Programes petits amb una sola utilitat.
- Eficiència des programes.
- Les operacions complexes es realitzen combinant programes.
- Es treballa amb fluxos de dades de text.

Redireccions

- **Permeten modificar els flux de dades estàndard:**
  - Podem modificar l'entrada o la sortida estàndard que utilitza una ordre o aplicació.
  - Redireccionar la sortida d'una ordre cap a un fitxer:

```
$ echo Hola! > hola.txt
$ cat hola.txt
Hola!
```

- **2 tipus principals:**
  - Redirecció de la sortida: **>**
  - Redirecció de l'entrada: **<**

Operadors de redirecció del **bash**

Operador	Descripció
>	Crea un nou fitxer contenint la sortida estàndard (no la d'error!) de la comanda executada. Atenció, si el fitxer existeix es sobreescrit (i es perden les dades existents)
>>	**Afegeix** (*append*) un fitxer la sortida estàndard (no la d'error!). Si el fitxer no existeix es crea. Les noves dades s'afegeixen al final del fitxer i no s'eliminen les dades existents.
2>	Crea un nou fitxer contenint la sortida d'error estàndard de la comanda executada. Atenció, si el fitxer existeix es sobreescrit (i es perden les dades existents)
2>>	**Afegeix** (*append*) a un fitxer la sortida d'error estàndard. Si el fitxer no existeix es crea. Les noves dades s'afegeixen al final del fitxer i no s'eliminen les dades existents.
&>	Crea un nou fitxer contenint la sortida estàndard i la sortida d'error estàndard de la comanda executada. Atenció, si el fitxer existeix es sobreescrit (i es perden les dades existents)
<	Envia les dades del fitxer especificat a la dreta com a entrada estàndard de la comanda de l'esquerre.
<<	Accepta un text de diverses línies com a entrada estàndard. Aquest filtre es coneix també com a heredoc (aquí document).
<>	El fitxer especificat a la dreta es tant la entrada estàndard com la sortida estàndard de la comanda de l'esquerre.

- Existeix un fitxer molt especial que serveix per a silenciar la sortida d'una comanda: `/dev/null` **(forat negre)**

Conductes (pipes)

- **Els conductes són un mecanisme que permet enllaçar la sortida estàndard d'una ordre amb l'entrada estàndard d'una altra ordre.**
- Permeten concatenar l'execució de diverses comandes:

```
$ comanda1 | comanda2 | comanda3 | comanda4
```

- La sortida de **comanda1** serà l'entrada de **comanda2**.
- La sortida de **comanda2** serà l'entrada de **comanda3**.
- La sortida de **comanda3** serà l'entrada de **comanda4**.
- La sortida de **comanda4** es mostrarà al terminal.

Comanda tee (T)

- **Dividir la sortida estàndard:**
  - Per tal que sigui mostrada per pantalla i guardada en tants fitxers com s'indiquin.

Entrada estàndard

Sortida estàndard

Fitxer(s) de text

```
$ ls –la | tee sortidaLs.txt
```

- Mostrarà el resultat de `ls -la` i el guardarà també al fitxer `sortidaLS.txt`.
- Per defecte `tee` copia la seva entrada estàndard sobre la seva sortida estàndard , i a més, sobreescriu els fitxers que indiquem com a paràmetres(elimina els continguts previs si el fitxer ja existia).

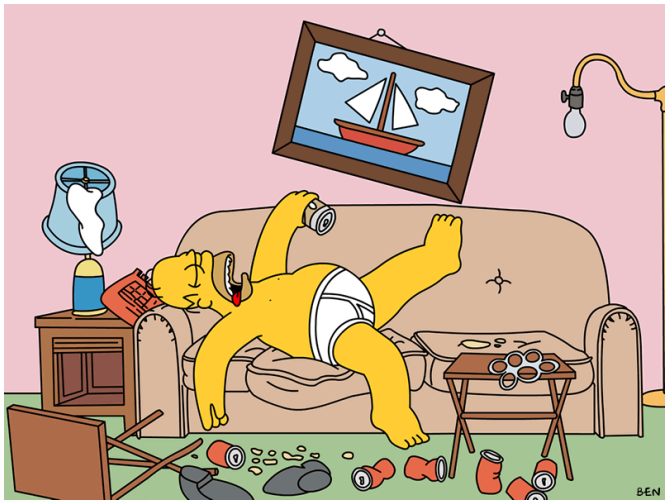
Comandes **echo** i **sleep**

- En aquest bloc farem servir la comanda **echo** en diverses ocasions, la qual permet escriure per la sortida estàndard.

```
$ echo “Sistemes Operatius”
```

- Una altra comanda que trobarem freqüentment serà **sleep**, que permet adormir o aturar l’execució durant el nombre de segons que es passin com a paràmetre.

```
$ sleep 10
```



Comandes `cat`, `more`, `less`, `pg` i `tac`

- És l'abreviatura de **concatenate**: concatena fitxers.

```
$ cat colors numeros
vermell
blau
verd
1
2
3
```

- S'acostuma a combinar amb paginadors:
  - `more`: `cat /etc/passwd /etc/shadow | more`
  - `less`: `cat /etc/passwd /etc/shadow | less`
  - `pg`: `cat /etc/passwd /etc/shadow | pg`

- La comanda `tac` permet fer el mateix, ordenant inversament.

Comandes **head** i **tail**

- Sovint només volem consultar les primeres o últimes línies d'un fitxer:
  - **`head`**: mostra les primeres 10 línies d'un fitxer. Amb el paràmetre **`-n`** podem indicar quantes línies volem mostrar.
    - **`head /etc/passwd`** és equivalent a **`cat /etc/passwd | head`**
    - **`ls | head -n 20`**
  - **`tail`**: mostra les últimes 10 línies d'un fitxer. Amb el paràmetre **`-n`** podem indicar quantes línies volem mostrar.
    - **`ls | tail -n 5`**

Comanda `sort`

- La comanda **`sort`** disposa de moltes opcions que permeten ordenar les línies d'un fitxer d'entrada, segons el criteri que indiquem.
  - Podem ordenar alfabèticament o numèricament, ascendentment o descendentment, definir la part de la línia per ordenar (camp), determinar quin delimitador utilitzem per a separar els camps (per defecte, l'espai)...

```
$ sort /etc/passwd
$ sort –t: -k3 –n /etc/passwd
$ ls –l /etc | sort –k5 –n –r | more
```

Comandes `wc`, `nl` i `pr`

- **`wc`**: per defecte, retorna el nombre de línies, paraules i bytes d'un fitxer. Disposem de paràmetres per a comptar només els caràcters, bytes, línies o paraules:

```
$ wc /etc/passwd
$ wc –c /etc/passwd
$ ls –l / | wc -l
```

- **`nl`**: numera les línies d'un fitxer, amb diverses opcions pel que fa a aquesta numeració.

```
$ cat quijote.txt | nl
$ nl quijote.txt
```

- **`pr`**: prepara un fitxer per a la seva impressió: afegeix pàgines, capçaleres i peus.

```
$ cat quijote.txt | pr
```

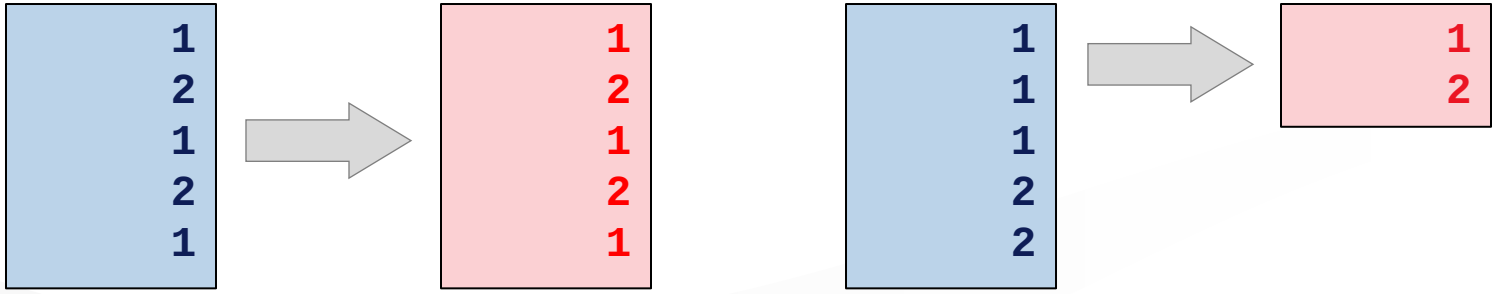
Comandes `fmt` i `uniq`

- `fmt`: formata línies llargues, segons la mida (en caràcters) que se li indiqui. Per defecte limita a 75 caràcters per línia, però es pot modificar. Té en compte els espais.

```
$ fmt quijote.txt
$ ls –l | fmt –w 10
```

- `uniq`: compacta en una única línia totes les línies **repetides i consecutives**. Disposa d'altres opcions de compactació.

```
$ uniq numeros.txt
```



- Se sol combinar amb `sort` per tal d'aconseguir el resultat anterior.

Comanda `split`

- Permet dividir un fitxer en altres fitxers. Podem triar si volem fraccionar el fitxer:
  - En funció d'una mida determinada de bytes
  - En funció d'un nombre de caràcters
  - En funció d'un nombre de línies

```
$ ls –l > prova.txt
$ split –l 1 prova.txt linia
$ cat linia*
```

- La comanda `split` no accepta que li passem el contingut a dividir per l'entrada estàndard:

```
$ ls –l | split –l 1 linia
```

- Per això utilitzem el guió (-), que permet utilitzar l'entrada estàndard com si fos un fitxer:

```
$ ls –l | split –l 1 – linia
```

Comandes `paste` i `join`

- **`paste`**: permet fusionar les línies de diversos fitxers, en un de sol, línia a línia. Utilitza el tabulador com a separador.

```
$ paste lletres.txt numeros.txt
```

```
A       1
B       2
C       3
D
```

- **`join`**: permet fusionar les línies de **dos** fitxers, respecte un camp:

```
$ join usuaris.txt contrasenyes.txt
```

```
1 Camilo Nuevo Memola
2 Ana Conda
3 Jesús Tomas Dao
```

```
1 contrasenya1
2 contrasenya2
3 contrasenya3
```

```
1 Camilo Nuevo Memola contrasenya1
2 Ana Conda contrasenya2
3 Jesús Tomas Dao contrasenya3
```

Comanda **cut**

- Selecciona una porció de totes les línies d'un fitxer d'entrada.
  - Podem especificar la porció de la línia com a rang de caràcters (no recomanable) o com a rang de camps (coma per a camps específics, guió per a rangs).

```
$ date | cut –c11-18
$ cut –d: –f3,5 --output-delimiter=“ “ /etc/passwd
```

  - Es poden indicar rangs oberts (fins al final / des del principi).
  - Podem indicar altres paràmetres com el delimitador de sortida.
  - També se li pot indicar que faci el complement del que es retalla (tot menys el que es demana a la comanda).

Comanda `tr`

- Fa una traducció caràcter a caràcters de l'entrada estàndard.

```
$ tr aeiou AEIOU < /etc/passwd
$ cat /etc/passwd | tr aeiou AEIOU
```

- Disposa de moltes opcions per a la traducció de caràcters:
  - Permet eliminar caràcters (en comptes de substituir).
  - **Permet comprimir caràcters repetits (MOLT UTILITZAT, COMBINAT AMB CUT).**
  - Permet fer conversions directes de conjunts coneguts: majúscules, minúscules, números, rangs,...
  - Si s'utilitzen caràcters especials (tabuladors, cometes,...) cal escapar-los (amb la contrabarra: ****).

Comanda `grep`

- Mostra les línies que continguin una o diverses cadenes de caràcters a unes posicions determinades.
- La comanda grep disposa d'un seguit de metacaràcters per a especificar les cadenes de caràcters (que són diferents dels del shell).
- **Exemples:**

```
$ grep ro /etc/passwd contenen
$ ls -l /etc | grep ^d comencen
$ ls -l /etc | grep -c ^d compta
$ ls -l /etc | grep ^-......r-x . és qualsevol caràcter
$ grep bash$ /etc/passwd acaben
$ grep -v /bin/bash /etc/passwd no contenen
$ grep ^n[aeiou] /etc/passwd na, ne, ni, no nu
```

Bibliografia i recursos utilitzats

- Estruch, J. Esteve; Carpintero, M. Àngel (2008). *Sistemes Operatius*. Institut Obert de Catalunya.
- Raya, Laura; Martín, Alejandro; Rodrigo, Víctor (2003). Sistemas Informáticos Monousuario y Multiusuario. RA-MA.
- Tamembaum, Andrew S (1992). *Sistemas Operativos Modernos*. Pretence-Hall Hispanoamericana.
- Stallings, William (2001). *Sistemas Operativos*. Pearson Educación.
- Morancho, Enric (2006). *Unix – Crides al sistema i comandes*. Edicions UPC.
- Smith, Roderich W. (2006). *LPIC 1: Linux Professional Institute Certification*. Sybex.
- LPIC-1. Examen 101. Objectiu 103.5 – Sergi Tur Badenas http://acacha.org/mediawiki/index.php/LPI_103.5