
RA2. Administració de processos del sistema

Índex

Introducció	1
1. Concepte de procés. Tipus, estats i cicle de vida	1
1.1. Què és un procés	1
1.2. Tipus de processos	1
1.3. Estats i cicle de vida	2
2. Interrupcions i excepcions	3
2.1. Interrupcions (asíncrones)	3
2.2. Excepcions (síncrones)	3
3. Procés, fil i treball	4
3.1. Veure fils d'un procés a Linux	4
3.2. Gestió de treballs (<i>jobs</i>) al shell	4
4. Creació, manipulació i acabament de processos	5
4.1. Creació a Linux: <code>fork()</code> + <code>exec()</code>	5
4.2. Creació i gestió des de la línia d'ordres	5
4.3. A Windows	6
5. El sistema de fitxers com a registre dels processos	6
6. Eines gràfiques i ordres de control i seguiment	7
6.1. Línia d'ordres (Linux)	7
6.2. Entorn gràfic	7
6.3. Gestió de serveis/dimonis amb <code>systemd</code>	7
7. Seqüència d'arrencada del sistema i dimonis	8
7.1. Fases de l'arrencada (Linux amb <code>systemd</code>)	8
7.2. Dimonis (<i>daemons</i>)	8
7.3. Windows: seqüència equivalent	8
8. Mesures de seguretat davant processos no identificats	9
9. Documentació dels processos habituals del sistema	10
9.1. Què ha de recollir la documentació	10
9.2. Exemple de fitxa de procés	10
10. Resum d'ordres	11
Índex de figures	12

Cicle: Administració de sistemes informàtics en xarxa (ASIX)

Mòdul: 0374. Administració de sistemes operatius

Introducció

Un sistema operatiu no és res més que un gestor de recursos: CPU, memòria, disc i xarxa. L'element central que representa "algú que fa servir aquests recursos" és el **procés**. Entendre com neix, viu i mor un procés ---i com el nucli decideix a qui li toca CPU en cada instant--- és la base per administrar, depurar i protegir qualsevol màquina, sigui Linux o Windows.

1. Concepte de procés. Tipus, estats i cicle de vida

1.1. Què és un procés

Un **procés** és una instància d'un programa en execució: codi carregat en memòria, un espai d'adreces propi, un conjunt de registres de CPU, fitxers oberts i un estat intern gestionat pel nucli. Dos usuaris poden executar el mateix binari (per exemple `firefox`) i el sistema operatiu els tractarà com a dos processos totalment independents, cadascun amb el seu propi **PID** (*Process ID*).

El nucli manté aquesta informació en una estructura de control anomenada **PCB** (*Process Control Block*), que inclou com a mínim:

- PID i PID del pare (**PPID**)
- Estat actual del procés
- Comptador de programa (adreça de la següent instrucció)
- Registres de CPU (es desen quan el procés es treu de la CPU)
- Taula de memòria (espai d'adreces)
- Taula de fitxers oberts (*file descriptors*)
- Prioritat i informació de planificació
- Usuari propietari (UID/GID a Linux, *token* de seguretat a Windows)

1.2. Tipus de processos

Criteri	Tipus	Exemple
Interacció	Interactiu (primer pla, espera E/S d'usuari)	un editor de text
	Per lots (<i>batch</i>)	un cron job de backup
Origen	Dimoni (<i>daemon</i> /servei, sense terminal)	sshd, nginx
	Pare i fill	bash crea ls
	Zombi	fill acabat, el pare no ha llegit el codi de sortida
	Orfe	pare mort abans que el fill; l'hereta <code>init/systemd</code> (PID 1)
Propietari	De sistema (arrenca amb el SO, sovint com a <code>root/SYSTEM</code>)	<code>systemd</code> , <code>svchost.exe</code>
	D'usuari	processos llançats des d'una sessió

1.3. Estats i cicle de vida

Un procés no està sempre “corrent”. El planificador (*scheduler*) del nucli el va movent entre estats segons si té CPU disponible i si està esperant algun recurs:

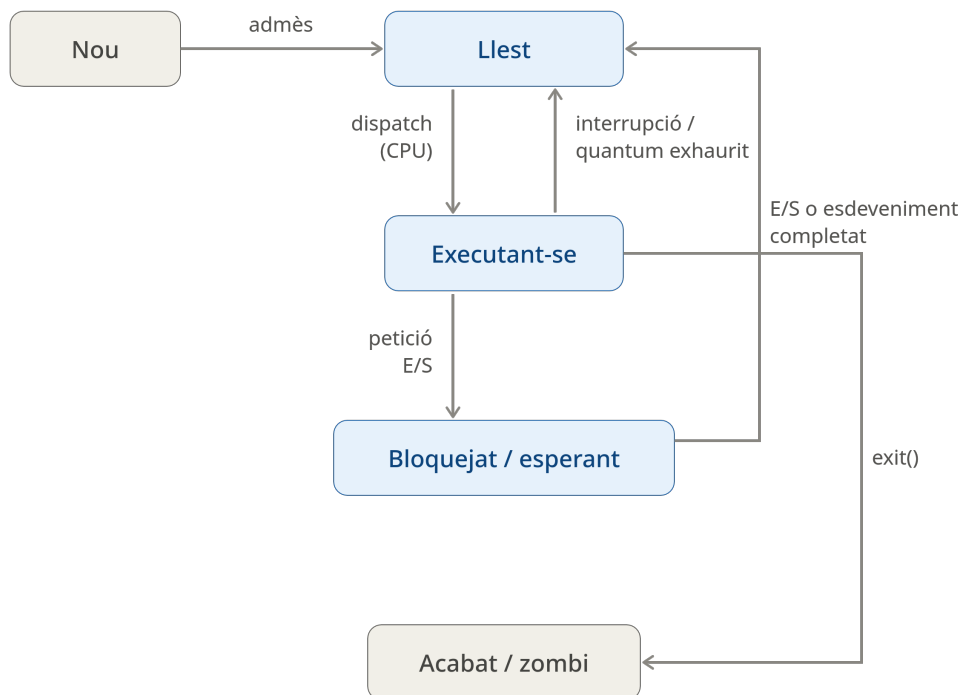


Figura 1: Estats i cicle de vida

- **Nou (New):** el procés s'està creant; encara no té recursos assignats.
- **Llest (Ready):** té tot el que necessita per executar-se, només li falta CPU.
- **Executant-se (Running):** té la CPU assignada en aquest instant.
- **Bloquejat/Esperant (Waiting/Blocked):** espera un esdeveniment extern (E/S de disc, xarxa, un senyal, entrada de teclat...).
- **Acabat (Terminated/Zombie):** ha finalitzat l'execució, però el seu codi de sortida encara no ha estat recollit pel pare (`wait()`).

NOTA

Un procés zombi **no consumeix CPU ni memòria útil**, només ocupa una entrada a la taula de processos (el seu PCB (*Process Control Block*)). Si en teniu molts, el problema real és el **pare**, que no crida `wait()`.

2. Interrupcions i excepcions

El processador necessita un mecanisme per reaccionar a esdeveniments que no formen part del flux seqüencial d'instruccions. Aquí és on entren interrupcions i excepcions.

2.1. Interrupcions (asíncrones)

Provocades per un dispositiu extern (targeta de xarxa, disc, teclat, temporitzador), **independentment** de la instrucció que s'estigui executant.

Flux típic:

1. El dispositiu activa una línia d'interruptió (IRQ).
2. La CPU acaba la instrucció actual i consulta la **IDT** (*Interrupt Descriptor Table*, x86) per trobar la rutina de gestió (ISR, *Interrupt Service Routine*).
3. Es desa el context del procés interromput (registres, comptador de programa).
4. S'executa l'ISR corresponent (per exemple, llegir dades de la targeta de xarxa cap al buffer del nucli).
5. Es restaura el context i el procés continua com si res.

El **rellotge del sistema** (*timer interrupt*) és la interrupció més important per a la multitasca: és la que permet al planificador "arrabassar" la CPU a un procés encara que no hagi acabat (*preemptió*).

2.2. Excepcions (síncrones)

Provocades per la **mateixa execució** d'una instrucció: divisió per zero, accés a memòria no vàlida (*segmentation fault*), instrucció il·legal, o una crida al sistema (*system call*, la forma controlada que té un procés d'usuari de demanar un servei al nucli, com obrir un fitxer).

	Interrupció	Excepció
Origen	Extern (maquinari)	Intern (execució d'instrucció)
Sincronia	Asíncrona	Síncrona
Exemple	Arribada d'un paquet de xarxa	divide by zero, page fault, syscall

A Linux podeu veure el recompte d'interrupcions ateses per cada CPU amb:

```
cat /proc/interrupts
```

I les excepcions greus (com un `segfault`) queden registrades al `journal`:

```
journalctl -k | grep -i segfault  
dmesg | grep -i "segfault\|general protection"
```

3. Procés, fil i treball

Sovint es fan servir com a sinònims, però són conceptes diferents:

- **Procés (process):** unitat d'execució amb el seu **propri espai de memòria** aïllat. Si un procés es penja, no afecta directament un altre.
- **Fil d'execució (thread):** unitat d'execució **dins** d'un procés. Tots els fils d'un mateix procés comparteixen l'espai de memòria, els fitxers oberts i altres recursos, però cadascun té la seva pròpia pila (*stack*) i registres. Són més "lleugers" de crear i canviar de context que un procés sencer.
- **Treball (job):** concepte de la línia d'ordres del *shell*, no del nucli. Un *job* és un procés (o *pipeline* de processos) llançat des d'una mateixa línia d'ordres, que el *shell* pot gestionar en primer o segon pla.

Procés A (espai de memòria propi)

```
├─ Fil 1 ┤
├─ Fil 2 ┤ } - comparteixen memòria, fitxers oberts, UID...
└─ Fil 3 ┘
```

Procés B (espai de memòria propi, aïllat d'A)

```
└─ Fil 1
```

3.1. Veure fils d'un procés a Linux

```
ps -eLf | grep firefox      # -L mostra els LWP (lightweight
↪ processes = fils)
ls /proc/<PID>/task/         # cada subdirectori és un fil
top -H -p <PID>             # mode fil-a-fil
```

3.2. Gestió de treballs (*jobs*) al shell

```
sleep 300 &                 # llança en segon pla, mostra [1] <PID>
jobs                        # llista els treballs del shell actual
fg %1                       # el porta a primer pla
Ctrl+Z                      # suspèn el treball en primer pla (estat Stopped/T)
bg %1                       # el reprèn en segon pla
disown %1                   # el desvincula del shell (sobreviu si tanqueu la
↪ terminal)
```

4. Creació, manipulació i acabament de processos

4.1. Creació a Linux: `fork()` + `exec()`

A Linux, un procés nou sempre neix a partir d'un altre (excepte `init`/PID 1):

1. **`fork()`** duplica el procés actual: crea un fill gairebé idèntic al pare (mateix codi, mateixa memòria en mode *copy-on-write*), amb un PID nou.
2. **`exec()`** (o alguna de les seves variants, `execve`, `execvp`...) substitueix el codi del procés fill pel d'un programa nou, sense crear cap procés addicional.

Aquest és el mecanisme que fa servir el shell cada cop que executeu una ordre: es fa `fork()` de `bash` i el fill fa `exec()` de, per exemple, `ls`.

```
strace -f -e trace=fork,execve,clone,exit ls    # veure fork/exec en
↪ directe
```

4.2. Creació i gestió des de la línia d'ordres

```
ordre &                # crea en segon pla
nohup ordre &          # sobreviu al tancament del terminal
nice -n 10 ordre       # crea amb prioritat baixa (nice més alt =
↪ menys prioritat)
renice -n 5 -p <PID>    # canvia la prioritat d'un procés ja creat
kill -l                # llista tots els senyals disponibles
kill -TERM <PID>        # petició "educada" d'acabament (per defecte)
kill -KILL <PID>        # (senyal 9) acabament immediat i
↪ incondicional pel nucli
kill -HUP <PID>         # molts dimonis l'interpreten com "recarrega
↪ la configuració"
pkill -f "patró_nom"    # mata per nom/patró en lloc de PID
killall nom_procés      # mata tots els processos amb aquell nom exacte
```

NOTA

Diferència clau TERM vs. KILL: `SIGTERM` (15) es lliura al procés perquè pugui alliberar recursos, tancar fitxers i sortir net; el procés pot fins i tot capturar-lo i ignorar-lo. `SIGKILL` (9) el gestiona directament el nucli, el procés **no el pot capturar ni ignorar**: mor tant sí com no, però sense oportunitat de netejar (pot deixar fitxers temporals o transaccions a mitges).

4.3. A Windows

```
Start-Process notepad.exe           # crear procés
Get-Process                        # llistar
Stop-Process -Name notepad -Force  # acabar per nom
Stop-Process -Id 1234              # acabar per PID
Start-Process programa.exe -Priority High # prioritat en la
    ↪ creació
(Get-Process notepad).PriorityClass = "High" # canviar prioritat en
    ↪ calent
```

Equivalent conceptual: Get-Process/Stop-Process fan de ps/kill, però Windows no té un fork() real (fa servir CreateProcess(), que crea el procés nou directament amb el seu propi executable, sense el pas intermedi de duplicar-se).

5. El sistema de fitxers com a registre dels processos

A Linux, el nucli exposa **tot** l'estat dels processos com un sistema de fitxers virtual muntat a /proc (*procfs*). No hi ha “fitxers” reals al disc: el nucli genera el contingut a l'instant en llegir-lo.

```
ls /proc/                # un directori numèric per cada PID viu
cat /proc/<PID>/status    # estat, memòria, UID/GID, fils...
cat /proc/<PID>/cmdline    # ordre exacta amb què es va llançar
ls -l /proc/<PID>/fd/      # fitxers/sockets oberts per aquest procés
cat /proc/<PID>/maps       # mapa de memòria del procés
readlink /proc/<PID>/exe   # binari executat
readlink /proc/<PID>/cwd   # directori de treball actual
cat /proc/loadavg         # càrrega mitjana del sistema
cat /proc/meminfo         # memòria global del sistema
```

Aquesta idea (“tot és un fitxer”) és filosofia central d'Unix i és el que permet a eines com ps o top funcionar sense necessitat de crides especials al nucli: simplement **llegeixen fitxers de text** dins de /proc.

A Windows no existeix un equivalent directe a /proc; l'estat dels processos es consulta via l'**API de Windows** (WMI, *Windows Management Instrumentation*) i s'exposa a eines com el Task Manager, PowerShell (Get-Process, Get-CimInstance Win32_Process) o el Registre en alguns aspectes de configuració de serveis.

6. Eines gràfiques i ordres de control i seguiment

6.1. Línia d'ordres (Linux)

Ordre	Ús
ps aux / ps -ef	fotografia instantània de tots els processos
ps -ef --forest	mostra la jerarquia pare-fill en arbre
top	monitor en temps real (CPU, memòria, per procés)
htop	versió millorada i interactiva de top
pstree	arbre de processos de forma visual
pgrep -a patró	cerca PID per nom/patró
lsof -p <PID>	fitxers oberts per un procés
vmstat 1	estadístiques de memòria/CPU cada segon
iostat	consum d'E/S de disc per procés

```
ps -eo pid,ppid,user,ni,stat,%cpu,%mem,cmd --sort=-%cpu | head
```

Camps útils de ps/top a recordar: STAT (S=dormint, R=corrent, D=E/S no interrompible, Z=zombi, T=aturat), NI (*niceness*, -20 a 19), %CPU/%MEM.

6.2. Entorn gràfic

- **Linux:** GNOME System Monitor, KSysGuard (KDE), o eines web com **Netdata/Cockpit** per a servidors sense entorn gràfic.
- **Windows:** **Task Manager** (pestanya *Details* per PID i prioritat), **Resource Monitor** (resmon), i **Process Explorer** (Sysinternals) per a una vista jeràrquica amb DLL carregades i handlers oberts, molt més potent que el Task Manager per a diagnòstic.

6.3. Gestió de serveis/dimonis amb systemd

```
systemctl status sshd          # estat d'un servei
systemctl list-units --type=service --state=running
systemctl restart sshd
journalctl -u sshd -f          # seguiment de logs en directe
```

7. Seqüència d'arrencada del sistema i dimonis

7.1. Fases de l'arrencada (Linux amb systemd)

1. **Firmware (BIOS/UEFI)**: POST (autotest de maquinari) i selecció del dispositiu d'arrencada.
2. **Bootloader (GRUB2)**: carrega el nucli (vmlinuz) i la initramfs (sistema de fitxers temporal amb els mòduls necessaris per muntar l'arrel real, per exemple controladors RAID/LVM).
3. **Nucli (kernel)**: s'inicialitza, munta l'arrel definitiva i llança el **primer procés d'espai d'usuari: PID 1**.
4. **PID 1 (systemd)**: processa les *unitats* (.service, .target, .socket...) i arriba al *target* per defecte (normalment multi-user.target o graphical.target), arrencant en paral·lel tots els serveis/dimonis necessaris respectant les dependències declarades (Requires=, After=).

```
systemd-analyze                # temps total d'arrencada
systemd-analyze blame          # temps per unitat, de més lent a més
    ↳ ràpid
systemd-analyze critical-chain  # cadena crítica de dependències
systemctl get-default          # target per defecte
pstree -p 1                    # arbre complet de processos des de PID 1
```

7.2. Dimonis (daemons)

Un **dimoni** és un procés de sistema que s'executa en segon pla, sense terminal de control, normalment des de l'arrencada, oferint un servei (xarxa, impressió, base de dades...). Convenció clàssica: acaben en d (sshd, crond, nginx... bé, aquest últim no, però httpd sí). Amb systemd, un dimoni sol correspondre a una unitat .service.

7.3. Windows: seqüència equivalent

1. **UEFI/BIOS** → **Boot Manager** (bootmgr)
2. **Windows Boot Loader** (winload.exe) carrega el nucli ntoskrnl.exe i els controladors crítics
3. El nucli inicialitza el **Session Manager** (smss.exe)
4. smss.exe llança wininit.exe, que al seu torn arrenca el **Service Control Manager** (services.exe), responsable d'arrencar tots els **serveis de Windows** (equivalent conceptual als dimonis de Linux) segons el seu tipus d'inici (Automàtic, Manual, Desactivat).

```
Get-Service | Where-Object {$_.Status -eq "Running"}
Get-CimInstance Win32_Service -Filter "StartMode='Auto'"
```

8. Mesures de seguretat davant processos no identificats

Trobar un procés “estrany” (nom desconegut, consum anòmal de CPU/xarxa, ubicat en un directori sospitós) requereix un protocol sistemàtic:

1. **Identificar-lo sense matar-lo primer** (matar-lo abans d'investigar pot destruir evidència útil):

```
ps -eo pid,ppid,user,cmd | grep <PID>
readlink /proc/<PID>/exe          # binari real que s'executa
ls -l /proc/<PID>/exe              # per si el binari ha estat
    ↳ esborrat del disc (indicador d'infecció)
cat /proc/<PID>/status | grep -i state
lsof -p <PID>                     # connexions de xarxa i
    ↳ fitxers oberts
```

2. **Comprovar la seva procedència:** és fill de qui? (`ps -o ppid= -p <PID>`), quin usuari l'ha llançat?, és una ubicació legítima del binari (`/usr/bin`, `/usr/sbin`) o sospitosa (`/tmp`, `/dev/shm`, directoris ocults)?
3. **Verificar la integritat del binari:** comparar amb el paquet original (`dpkg -S /ruta/binari`, `rpm -qf`, o sumes de comprovació contra un dipòsit de confiança).
4. **Comprovar connexions de xarxa actives** associades:

```
ss -tulpn | grep <PID>
netstat -antp | grep <PID>          # sistemes més antics
```

5. **Aïllar abans d'eliminar:** si es confirma sospita real, és preferible aïllar la màquina de la xarxa (o suspendre'n la interfície) abans d'actuar, per evitar propagació i preservar evidència forense.
6. **Actuar:** SIGSTOP (aturar-lo sense matar, per congelar-lo i inspeccionar-lo en calma) abans de decidir entre SIGKILL o conservar-lo per a anàlisi forense.
7. **Auditar la persistència:** revisar `cron`, `systemd timers/services`, `~/.bashrc`, `/etc/rc.local` i claus d'autoarrencada per si el procés té mecanismes per reaparèixer després de reiniciar.
8. **Documentar l'incident** (vegeu 2.9) i, si escau, aplicar les polítiques de resposta a incidents del centre/empresa.

Eines de suport habituals: **AIDE/Tripwire** (integritat de fitxers), **Wazuh/OSSEC** (HIDS), **auditd** (registre d'esdeveniments en l'àmbit del nucli), i a Windows **Sysinternals Process Explorer/Autoruns** per detectar persistència i **Windows Defender/Event Viewer** per als registres de seguretat.

NOTA

A Windows, un indicador equivalent d'alarma és un procés amb el nom “gairebé correcte” d'un procés de sistema legítim (per exemple `svch0st.exe` en lloc de `svchost.exe`) o un `svchost.exe` executant-se des d'una ruta diferent de `C:\Windows\System32`.

9. Documentació dels processos habituals del sistema

Documentar no és opcional en un entorn professional: permet detectar anomalies (comparant contra la “normalitat” esperada) i facilita el manteniment per part d'altres tècnics.

9.1. Què ha de recollir la documentació

Per cada procés/servei rellevant del sistema:

- **Nom del procés/servei** i binari que l'executa
- **Funció:** què fa i per què és necessari
- **Usuari** amb què s'executa (privilegis)
- **Dependències:** de quins altres serveis depèn / quins en depenen
- **Ports i protocols** que utilitza, si escau
- **Fitxers de configuració i de log** associats
- **Comportament normal** (consum típic de CPU/memòria, freqüència d'execució)
- **Procediment d'arrencada/aturada/reinici**

9.2. Exemple de fitxa de procés

Camp	Valor
Nom	sshd
Binari	/usr/sbin/sshd
Funció	Servidor SSH, accés remot xifrat
Usuari	root (procés principal), sshd (subprocessos privilegi mínim)
Dependències	network.target, sshd.socket (opcional, activació sota demanda)
Ports	TCP/22
Configuració	/etc/ssh/sshd_config
Logs	journalctl -u sshd
Comportament normal	Un procés pare + un fill per sessió activa

Aquesta documentació es pot generar de forma semiautomàtica combinant `systemctl list-units`, `ps --forest` i man de cada servei, i mantenir-la actualitzada com a part dels procediments d'explotació del sistema (*runbooks*).

10. Resum d'ordres

Acció	Linux	Windows (PowerShell)
Llistar processos	ps aux, top, htop	Get-Process, Task Manager
Arbre de processos	pstree, ps --forest	Process Explorer
Matar un procés	kill, pkill, killall	Stop-Process
Prioritat	nice, renice	-Priority, .PriorityClass
Fils d'un procés	ps -eLf, top -H	pestanya <i>Threads</i> de Process Explorer
Fitxers/connexions obertes	lsof, ss -tuln	Get-NetTCPConnection, handle.exe
Estat del nucli/registre de processos	/proc/<PID>/*	WMI / Get-CimInstance Win32_Process
Gestió de serveis	systemctl	Get-Service, sc.exe
Anàlisi d'arrencada	systemd-analyze	Event Viewer (log System)

Índex de figures

1	Estats i cicle de vida	2
---	----------------------------------	---

Versions d'aquest document

- [HTML](#)
- [PDF](#)
- [ODT](#)
- [MD](#)

[Domini Públic \(CC0\)](#)