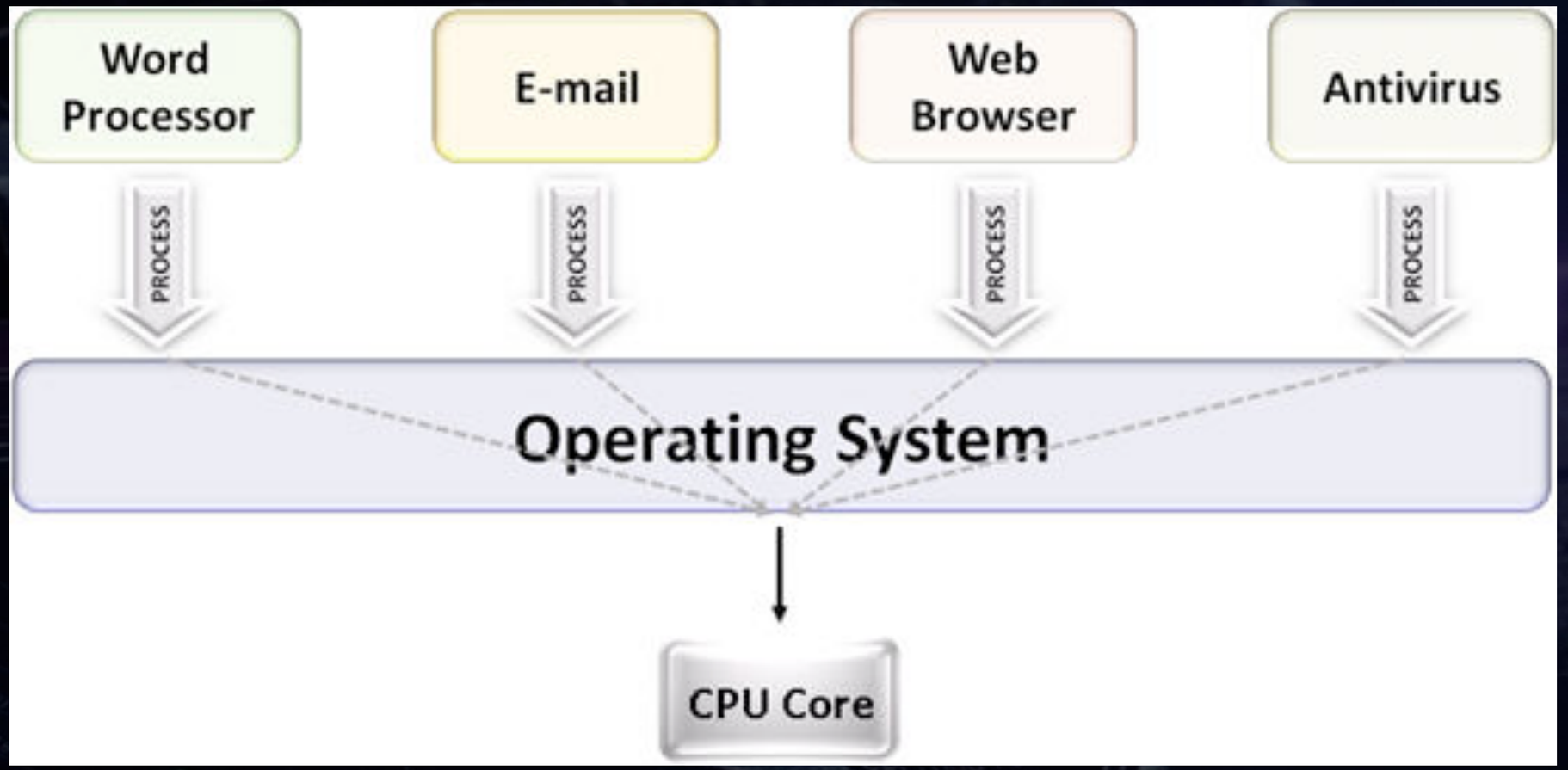


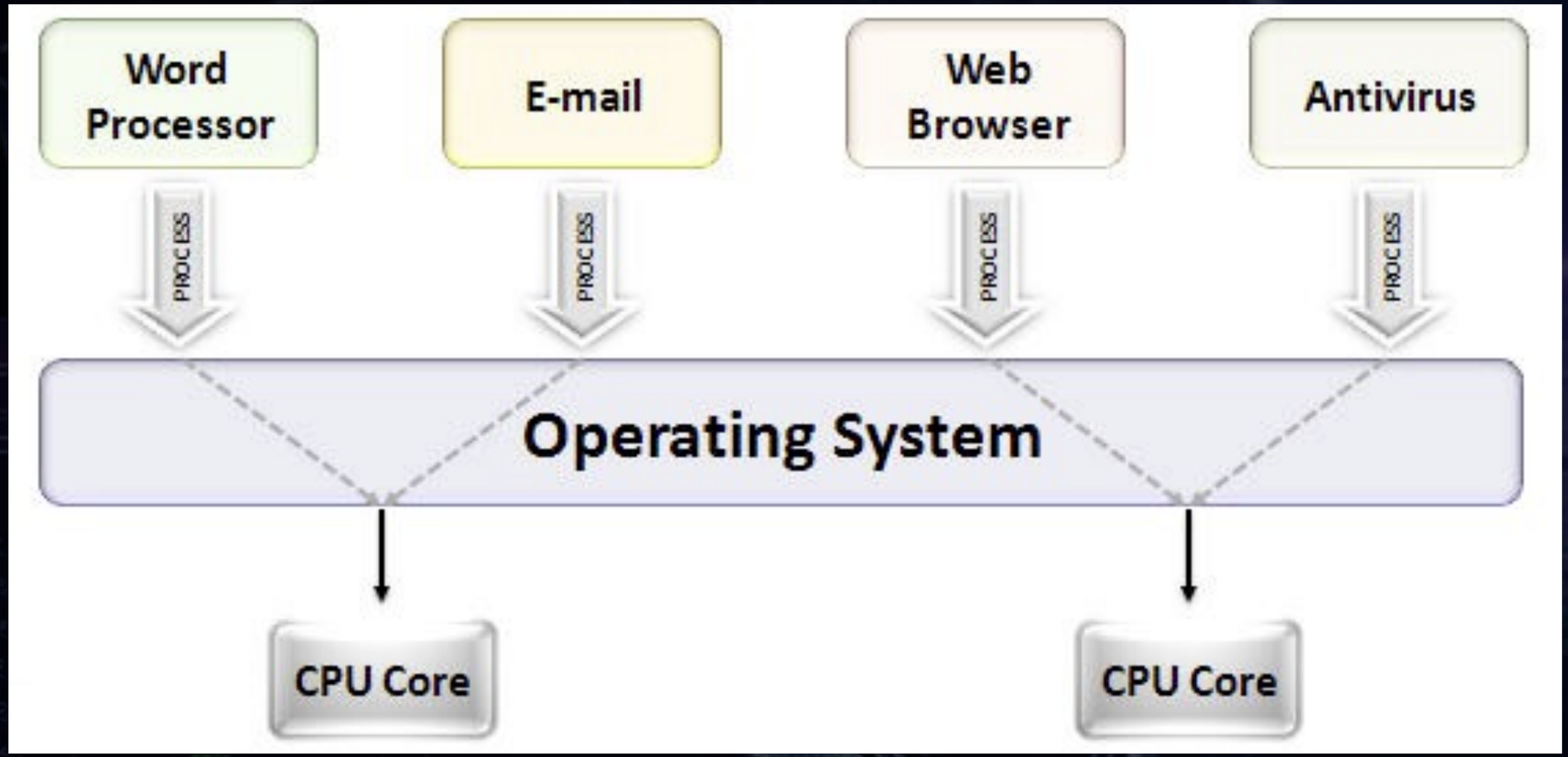
Gestion des processus sous UNIX

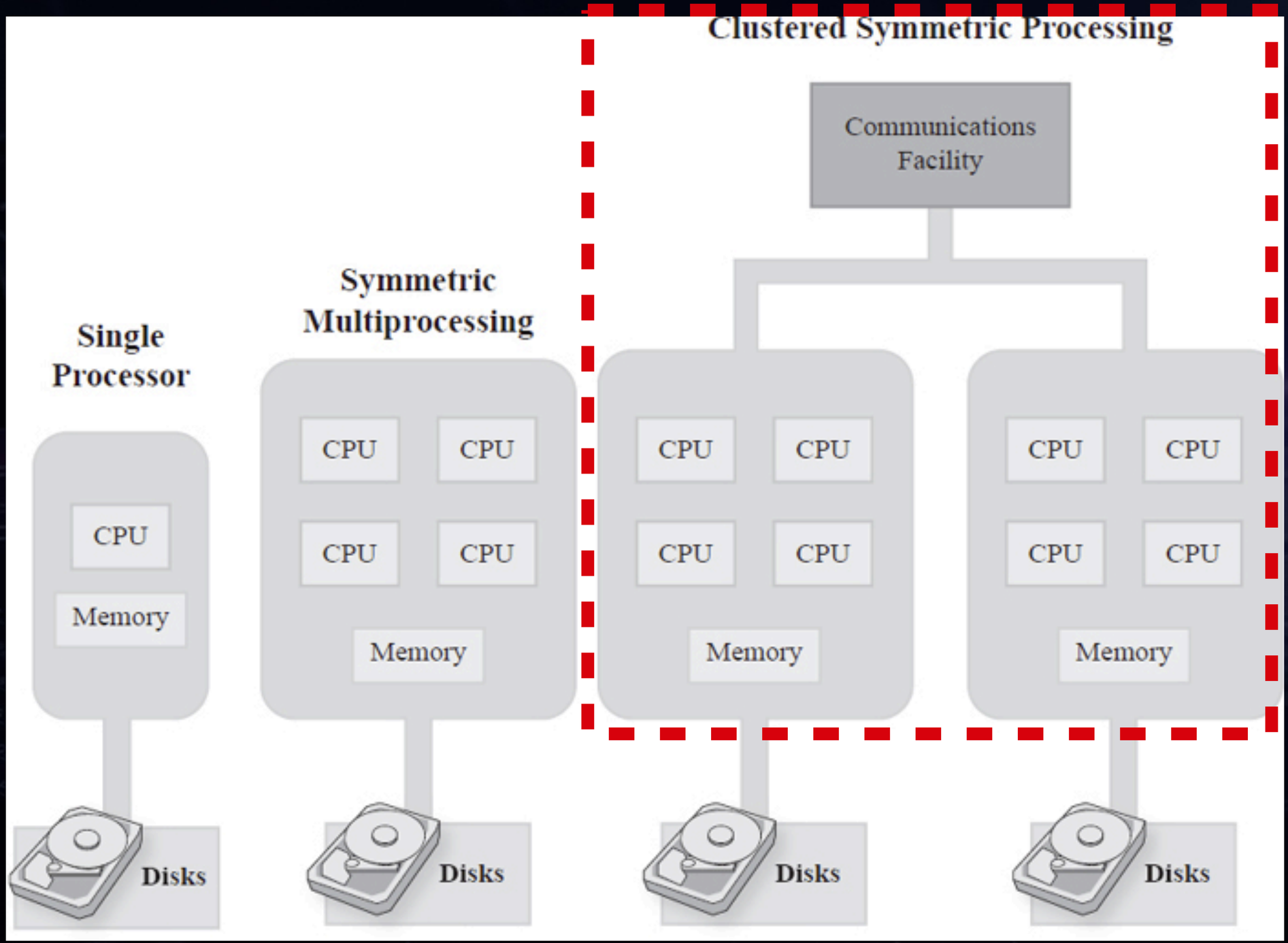
Programmation multitâches

Sous Windows




Word Processor
E-mail
Web Browser
Antivirus
PROCESS
PROCESS
PROCESS
PROCESS
Operating System
CPU Core


Word Processor
E-mail
Web Browser
Antivirus
PROCESS
PROCESS
PROCESS
PROCESS
Operating System
CPU Core
CPU Core

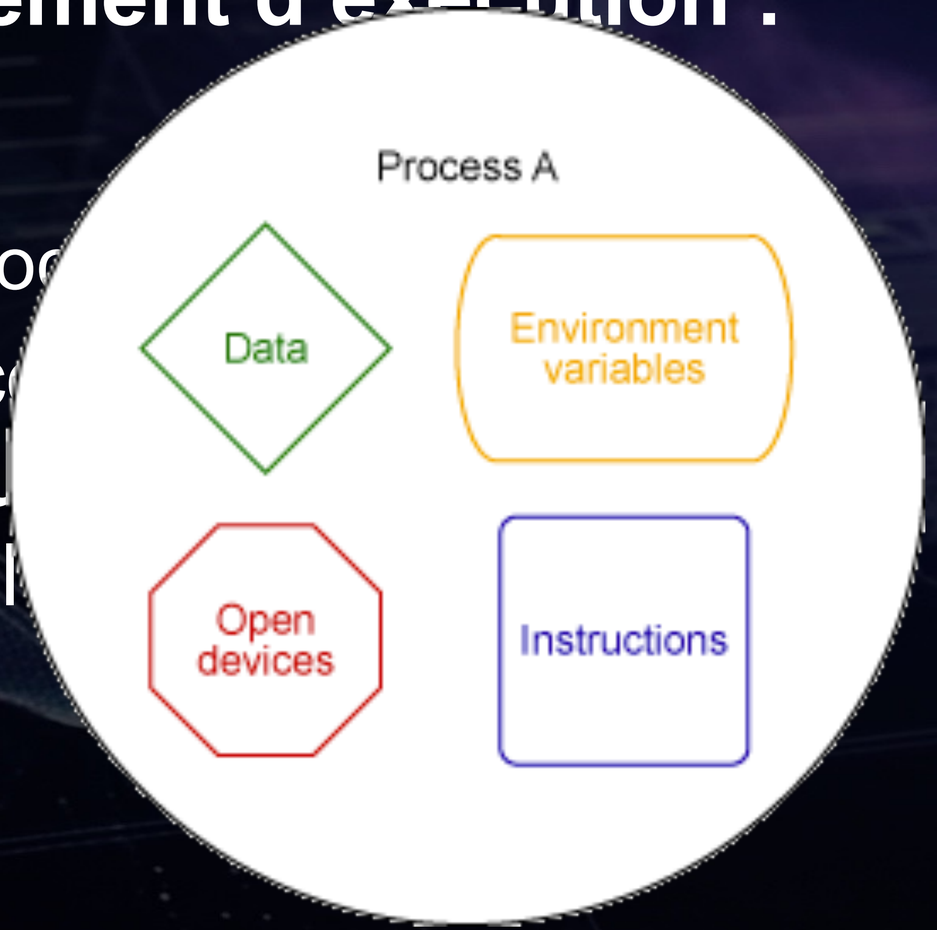

Clustered Symmetric Processing
Communications Facility
Single Processor
Symmetric Multiprocessing
CPU
Memory
Disks

Qu'est-ce qu'un processus ?

- **Un programme en cours d'exécution, avec son environnement d'exécution :**
  - Instructions
  - Variables
  - Caractéristiques du processus
- Chaque processus contient, entre autres, des **informations sur le processus qui est à son origine** (appelé processus « père »).

Qu'est-ce qu'un processus ?

- **Un programme en cours d'exécution, avec son environnement d'exécution :**
  - Instructions
  - Variables
  - Caractéristiques du proc
- Chaque processus c
  des **informations su**
  à son origine (appel



Caractéristiques d'un processus

- Chaque processus travaille dans une **zone d'adressage virtuel séparée**, pour éviter qu'il soit gêné, volontairement ou non, par un autre processus.

- L'accès à la CPU se fait de manière séquentielle (sauf sur les machines multicoeur, mais on verra cela plus tard). L'ordre de ces accès et leur durée sont définis par ***l'ordonnanceur*** (ou ***scheduler***).

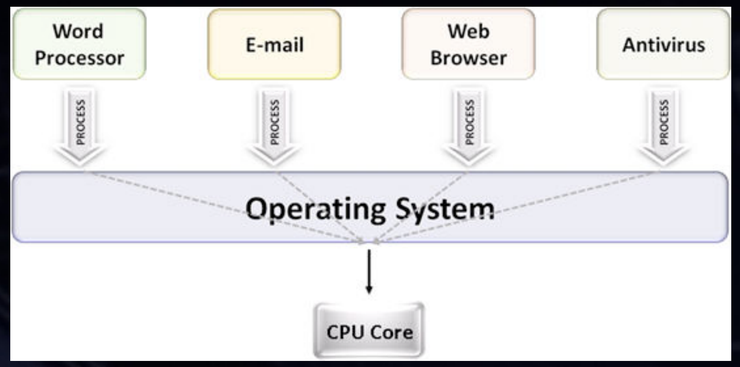
- Les processus peuvent échanger des données par divers types de ***communication inter-processus***.

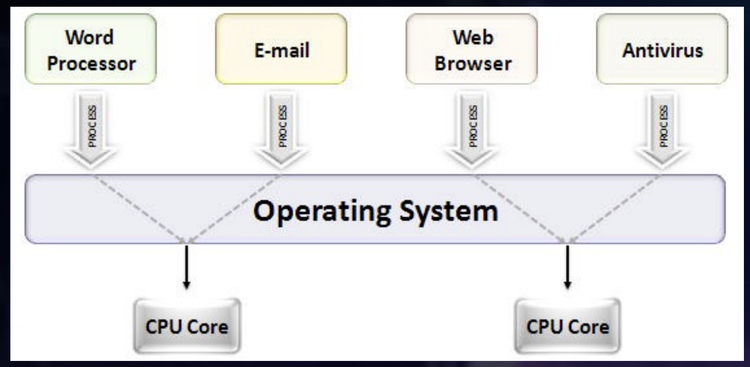
Identification d’un processus

- Les processus sont repérés dans le système par un numéro : le **PID** (Process IDentification).



Ordonnancement



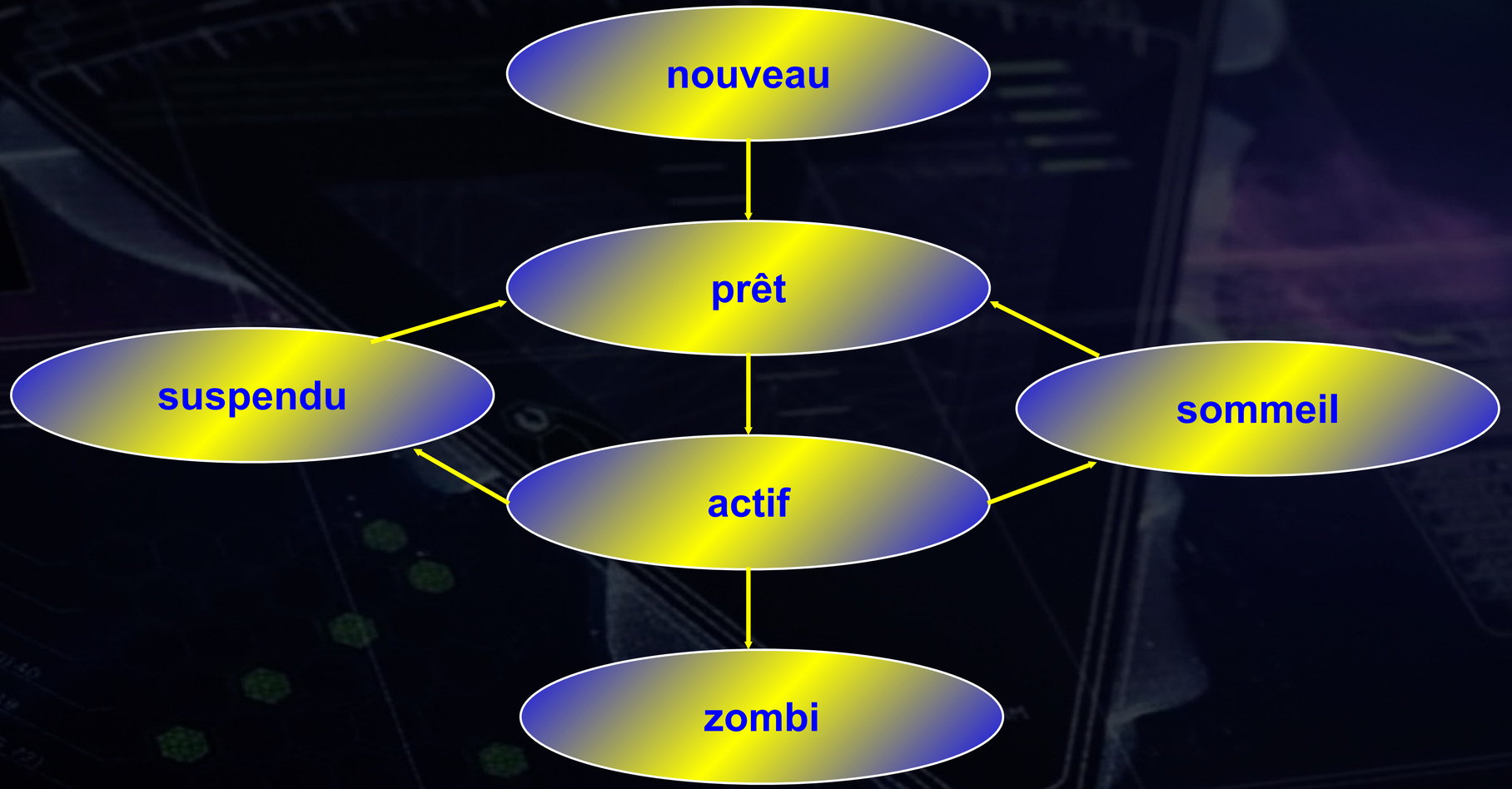


- C’est le noyau du système qui effectue la gestion des différents processus à un instant donné au travers de ***l’ordonnancement*** (***scheduling***).
- Le noyau choisit à des instants particuliers parmi les différents processus en attente du (ou d’un) processeur un (ou plusieurs) processus en vue de leur activation.

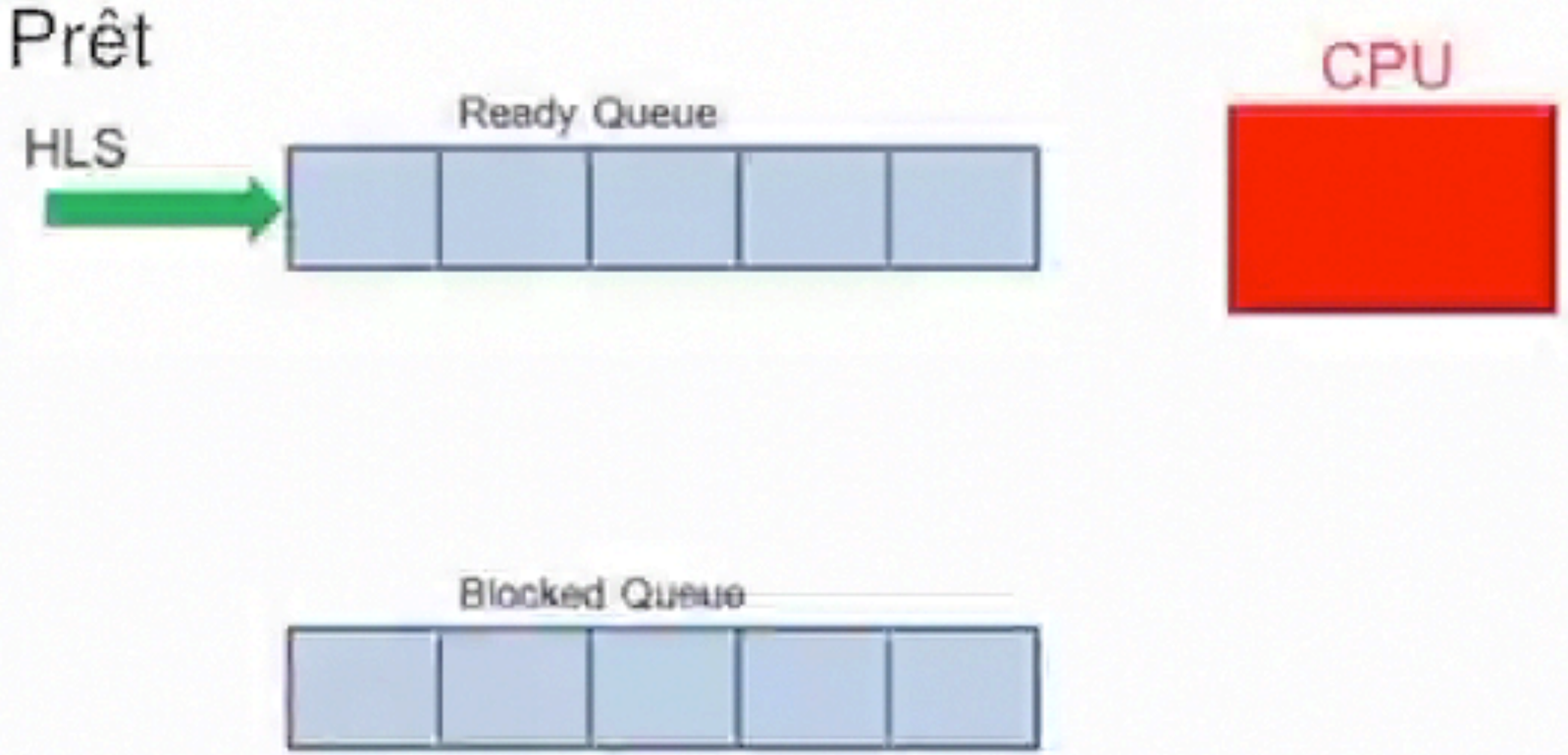
Déterminisme

- Stratégie d'ordonnancement d'Unix : recalcul périodique de la priorité des processus en fonction de différents critères.
- Il ne peut être fait aucune hypothèse sur les résultats de l'ordonnancement des processus impliqués :
  - Le comportement des applications est non-déterministe.
  - Si l'on veut un certain déterminisme, il faut avoir recours à des mécanismes de synchronisation : **blocage temporaire** de certains processus afin qu'ils ne soient pas pris en compte par le module d'ordonnancement.

Diagramme de changement de contexte



Ordonnancement - Scheduling



État des processus

- L'ordonnanceur gère l'aspect multitâche du système. Pour cela, il fait passer les processus par différents états.
  - État prêt : le processus est éligible par l'ordonnanceur
  - État actif : le processus est exécuté par le microprocesseur
  - État sommeil : le processus est en attente d'événement (libération d'une ressource par ex.)
  - État suspendu : arrêt momentané du processus (mode trace de debug par ex.)
  - État zombi : le processus est terminé, mais le processus père n'a pas pris connaissance de sa disparition

Commandes shell associées

- **ps** : liste des processus créés par l'utilisateur dans la session en cours
- **ps –l** : idem précédent + informations supplémentaires
- **ps –e** : liste de tous les processus du système
  - **ps –el** : combinaison des options précédentes
  - **ps –el|more** : affichage page par page

Commande ps -el



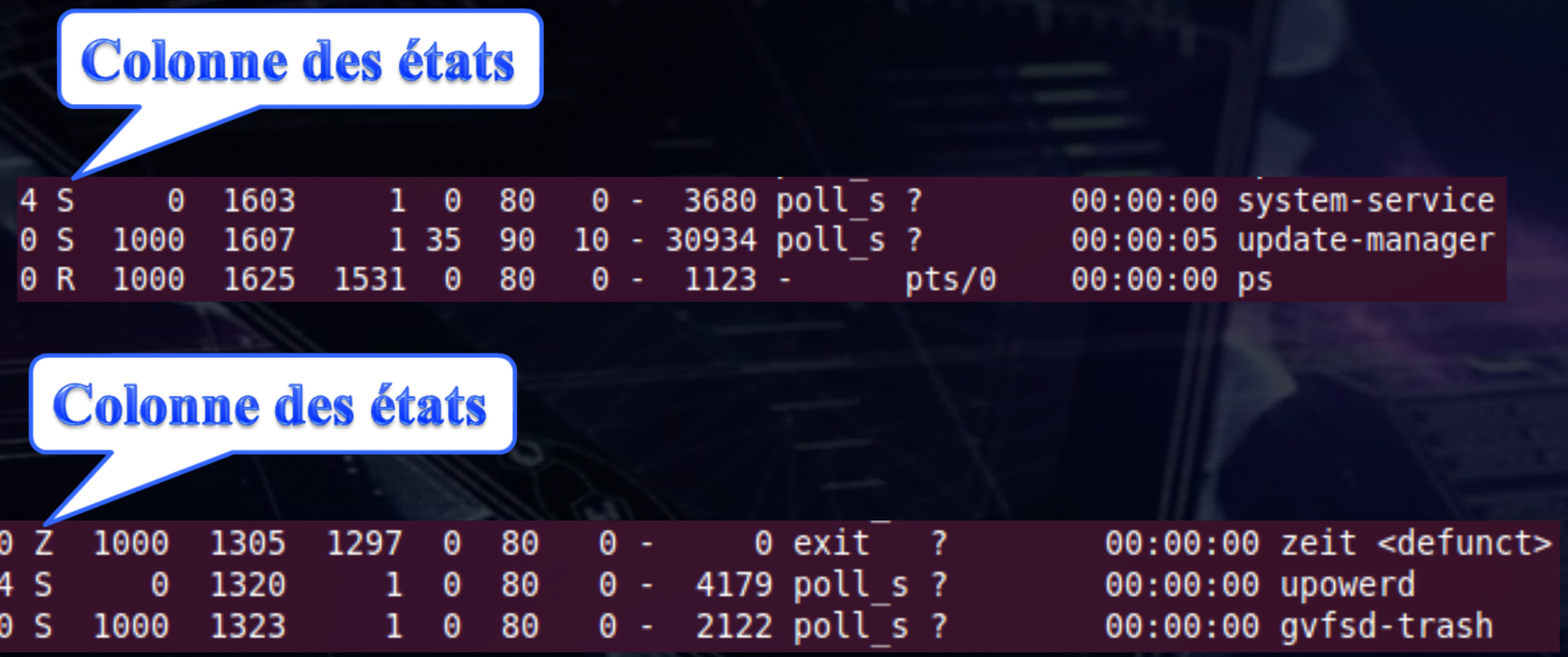
PID

PPID

File Actions Edit View Help

```
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0       1       0  0  80   0 -  5768 -      ?        00:00:00 systemd
1 S     0       2       0  0  80   0 -     0 -      ?        00:00:00 kthreadd
1 S     0       3       2  0  80   0 -     0 -      ?        00:00:00 pool_workqueue_release
1 I     0       4       2  0  60 -20 -     0 -      ?        00:00:00 kworker/R-rcu_g
1 I     0       5       2  0  60 -20 -     0 -      ?        00:00:00 kworker/R-rcu_p
1 I     0       6       2  0  60 -20 -     0 -      ?        00:00:00 kworker/R-slub_
1 I     0       7       2  0  60 -20 -     0 -      ?        00:00:00 kworker/R-netns
1 I     0       8       2  0  80   0 -     0 -      ?        00:00:00 kworker/0:0-events
1 I     0       9       2  0  60 -20 -     0 -      ?        00:00:00 kworker/0:0H-events_highpri
1 I     0      10       2  0  80   0 -     0 -      ?        00:00:00 kworker/0:1-events
5 I     0      11       2  0  80   0 -     0 -      ?        00:00:00 kworker/u4:0-events_unbound
1 I     0      12       2  0  60 -20 -     0 -      ?        00:00:00 kworker/R-mm_pe
1 I     0      13       2  0  80   0 -     0 -      ?        00:00:00 rcu_tasks_rude_kthread
1 I     0      14       2  0  80   0 -     0 -      ?        00:00:00 rcu_tasks_trace_kthread
1 S     0      15       2  0  80   0 -     0 -      ?        00:00:00 ksoftirqd/0
1 I     0      16       2  0  80   0 -     0 -      ?        00:00:00 rcu_sched
1 S     0      17       2  0 -40   - -     0 -      ?        00:00:00 migration/0
1 S     0      18       2  0  80   0 -     0 -      ?        00:00:00 cpuhp/0
1 S     0      19       2  0  80   0 -     0 -      ?        00:00:00 cpuhp/1
1 S     0      20       2  0 -40   - -     0 -      ?        00:00:00 migration/1
1 S     0      21       2  0  80   0 -     0 -      ?        00:00:00 ksoftirqd/1
1 I     0      22       2  0  80   0 -     0 -      ?        00:00:00 kworker/1:0-events
```

Différents états des processus

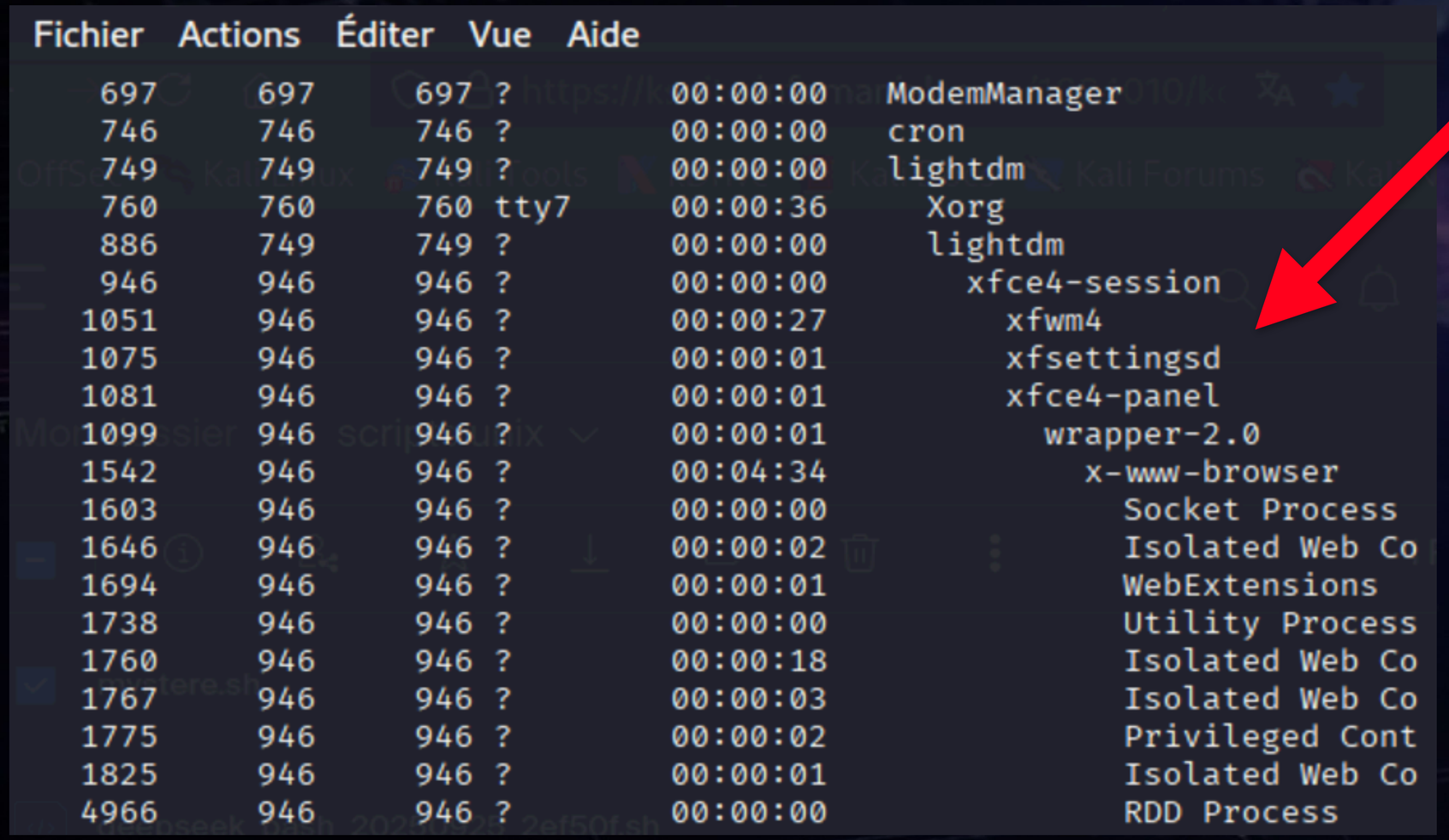


Arrêter un processus

- sudo kill PID_du_processus
  ou encore plus efficace :
- sudo kill -9 PID_du_processus

Commande ps -ejH

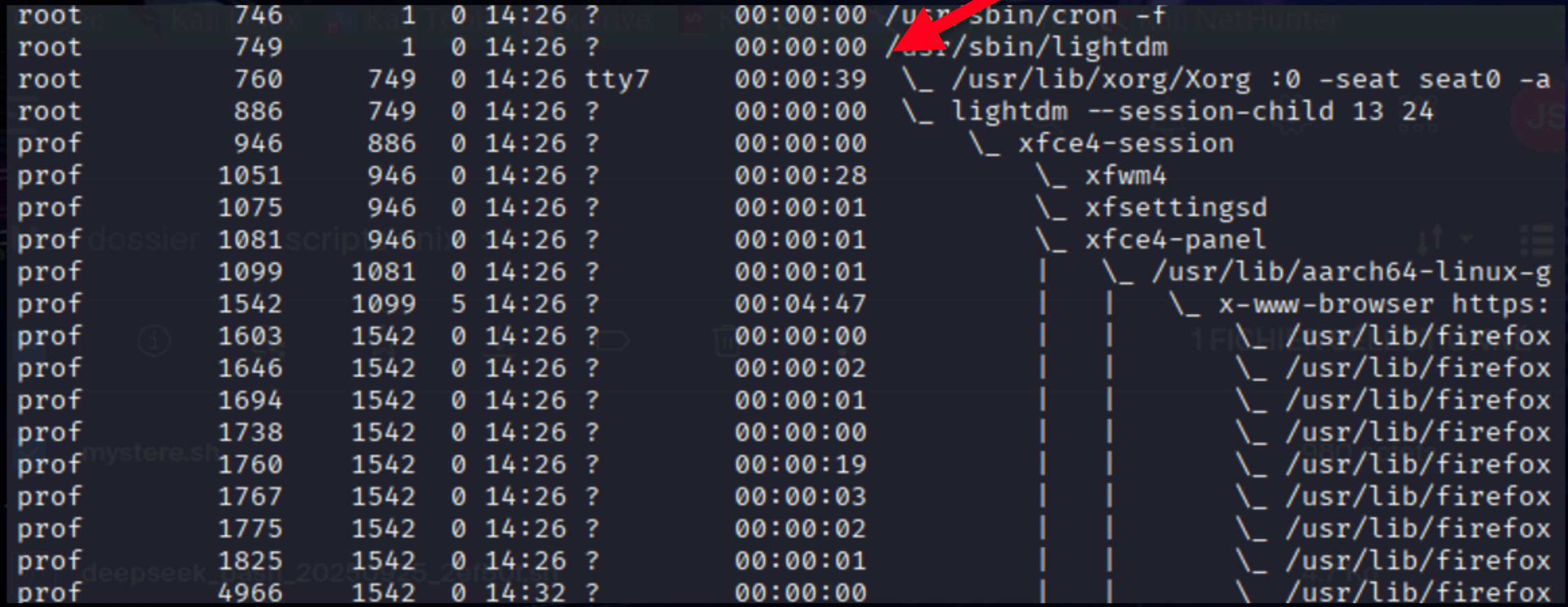
Arborescence des processus



```
Fichier  Actions  Éditer  Vue  Aide
    697      697     697 ?        00:00:00 ModemManager
    746      746     746 ?        00:00:00 cron
    749      749     749 ?        00:00:00 lightdm
    760      760     760 tty7     00:00:36   Xorg
    886      749     749 ?        00:00:00   lightdm
    946      946     946 ?        00:00:00     xfce4-session
   1051      946     946 ?        00:00:27       xfwm4
   1075      946     946 ?        00:00:01       xfsettingsd
   1081      946     946 ?        00:00:01       xfce4-panel
   1099      946     946 ?        00:00:01         wrapper-2.0
   1542      946     946 ?        00:04:34           x-www-browser
   1603      946     946 ?        00:00:00             Socket Process
   1646      946     946 ?        00:00:02             Isolated Web Co
   1694      946     946 ?        00:00:01             WebExtensions
   1738      946     946 ?        00:00:00             Utility Process
   1760      946     946 ?        00:00:18             Isolated Web Co
   1767      946     946 ?        00:00:03             Isolated Web Co
   1775      946     946 ?        00:00:02             Privileged Cont
   1825      946     946 ?        00:00:01             Isolated Web Co
   4966      946     946 ?        00:00:00             RDD Process
```

Commande ps -ef --forest

Arborescence des processus



```
root         746       1  0 14:26 ?        00:00:00 /usr/sbin/cron -f
root         749       1  0 14:26 ?        00:00:00 /usr/sbin/lightdm
root         760     749  0 14:26 tty7     00:00:39  \_ /usr/lib/xorg/Xorg :0 -seat seat0 -a
root         886     749  0 14:26 ?        00:00:00  \_ lightdm --session-child 13 24
prof         946     886  0 14:26 ?        00:00:00      \_ xfce4-session
prof        1051     946  0 14:26 ?        00:00:28          \_ xfwm4
prof        1075     946  0 14:26 ?        00:00:01          \_ xfsettingsd
prof        1081     946  0 14:26 ?        00:00:01          \_ xfce4-panel
prof        1099    1081  0 14:26 ?        00:00:01          |   \_ /usr/lib/aarch64-linux-g
prof        1542    1099  5 14:26 ?        00:04:47          |   |   \_ x-www-browser https:
prof        1603    1542  0 14:26 ?        00:00:00          |   |       \_ /usr/lib/firefox
prof        1646    1542  0 14:26 ?        00:00:02          |   |           \_ /usr/lib/firefox
prof        1694    1542  0 14:26 ?        00:00:01          |   |           \_ /usr/lib/firefox
prof        1738    1542  0 14:26 ?        00:00:00          |   |           \_ /usr/lib/firefox
prof        1760    1542  0 14:26 ?        00:00:19          |   |           \_ /usr/lib/firefox
prof        1767    1542  0 14:26 ?        00:00:03          |   |           \_ /usr/lib/firefox
prof        1775    1542  0 14:26 ?        00:00:02          |   |           \_ /usr/lib/firefox
prof        1825    1542  0 14:26 ?        00:00:01          |   |           \_ /usr/lib/firefox
prof        4966    1542  0 14:32 ?        00:00:00          |   |           \_ /usr/lib/firefox
```

Autre commande : pstree



```
┌──(prof㉿kali)-[~]
└─$ pstree
systemd─┬─ModemManager───3*[{ModemManager}]
        ├─NetworkManager───3*[{NetworkManager}]
        ├─accounts-daemon───3*[{accounts-daemon}]
        ├─agetty
        ├─colord───3*[{colord}]
        ├─cron
        ├─dbus-daemon
        ├─haveged
        ├─lightdm─┬─Xorg───3*[{Xorg}]
        │         └─lightdm─┬─xfce4-session─┬─Thunar───4*[{Thunar}]
        │                   │               ├─agent───3*[{agent}]
        │                   │               ├─applet.py
        │                   │               ├─blueman-applet───4*[{blueman-applet}]
        │                   │               ├─firefox-esr─┬─2*[Isolated Web Co───15*[{Isolated Web Co}]]
        │                   │               │             ├─Isolated Web Co───14*[{Isolated Web Co}]
        │                   │               │             ├─Privileged Cont───15*[{Privileged Cont}]
        │                   │               │             ├─RDD Process───3*[{RDD Process}]
        │                   │               │             ├─Socket Process───5*[{Socket Process}]
        │                   │               │             ├─Utility Process───3*[{Utility Process}]
        │                   │               │             ├─3*[Web Content───13*[{Web Content}]]
        │                   │               │             ├─WebExtensions───14*[{WebExtensions}]
        │                   │               │             └─71*[{firefox-esr}]
        │                   │               ├─light-locker───4*[{light-locker}]
        │                   │               ├─nm-applet───5*[{nm-applet}]
        │                   │               ├─polkit-mate-aut───3*[{polkit-mate-aut}]
        │                   │               ├─qterminal─┬─zsh───pstree
        │                   │               │           └─3*[{qterminal}]
        │                   │               ├─qterminal─┬─zsh
        │                   │               │           └─3*[{qterminal}]
        │                   │               ├─ssh-agent
        │                   │               ├─xfce4-panel─┬─panel-1-whisker───4*[{panel-1-whisker}]
```

Autre commande : top

prof@kali: ~

File Actions Edit View Help

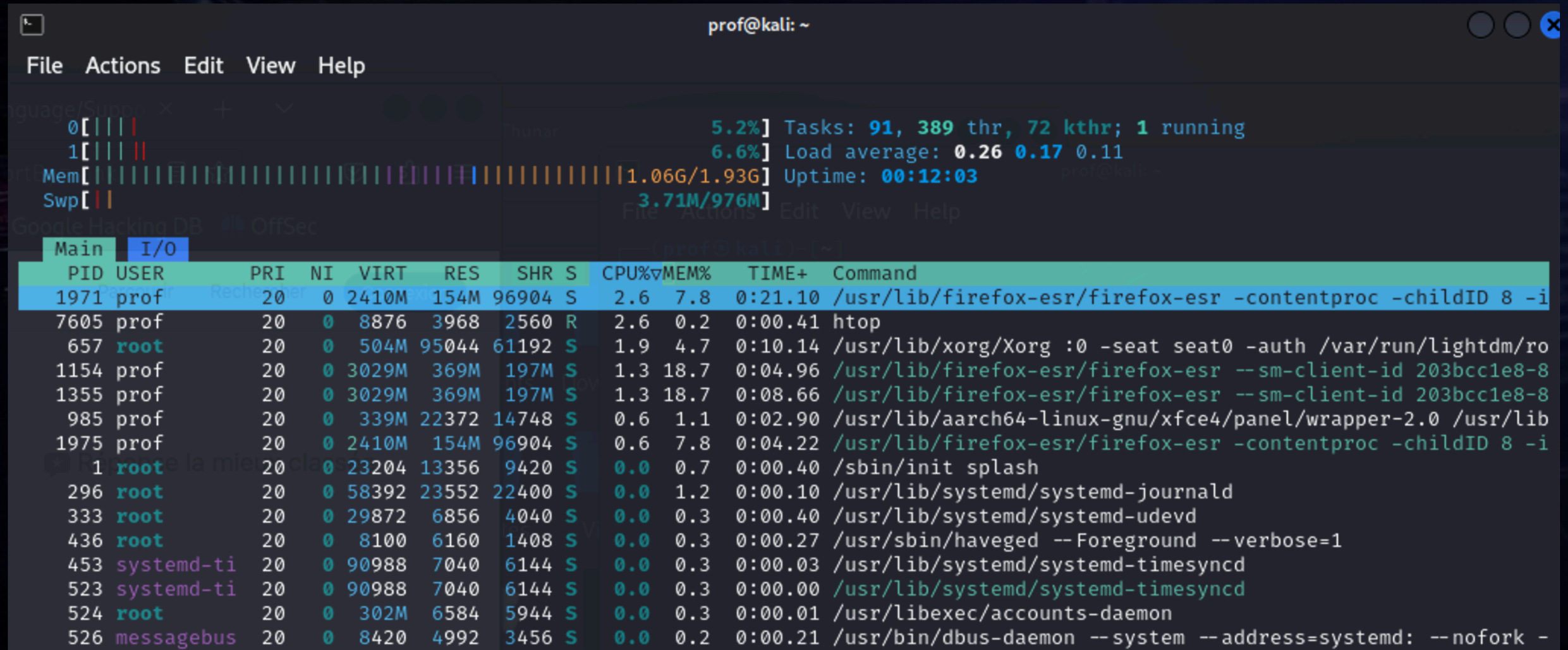
```
top - 08:46:19 up 10 min,  2 users,  load average: 0.01, 0.13, 0.09
Tasks: 163 total,   1 running, 162 sleeping,   0 stopped,   0 zombie
%Cpu(s):  6.3 us,  2.7 sy,  0.0 ni, 89.9 id,  0.8 wa,  0.0 hi,  0.3 si,  0.0 st
MiB Mem :   1972.5 total,    49.3 free,   1154.7 used,   1079.3 buff/cache
MiB Swap:    976.0 total,    972.0 free,      4.0 used.    817.8 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1971	prof	20	0	2466060	155064	96904	S	3.7	7.7	0:23.95	Isolated Web Co
1068	prof	20	0	3102232	375464	202392	S	2.3	18.6	0:26.85	firefox-esr
1	root	20	0	23204	13356	9420	S	0.7	0.7	0:00.39	systemd
333	root	20	0	29872	6856	4040	S	0.7	0.3	0:00.37	systemd-udevd
32	root	20	0	0	0	0	S	0.3	0.0	0:00.83	kcompactd0
43	root	20	0	0	0	0	S	0.3	0.0	0:00.12	kswapd0
171	root	20	0	0	0	0	I	0.3	0.0	0:00.05	kworker/u4:4-events_unbound
530	root	20	0	18152	8348	7168	S	0.3	0.4	0:00.05	systemd-logind
657	root	20	0	516908	95044	61192	S	0.3	4.7	0:09.46	Xorg
1059	prof	20	0	530524	93980	77032	S	0.3	4.7	0:01.01	qterminal
1064	root	20	0	317864	8704	6784	S	0.3	0.4	0:00.08	upowerd
6828	**prof**	**20**	**0**	**9660**	**4352**	**2432**	**R**	**0.3**	**0.2**	**0:00.08**	**top**
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	0:00.00	pool_workqueue_release
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/R-rcu_g
5	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/R-rcu_p
6	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/R-slub_
7	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/R-netns
8	root	20	0	0	0	0	I	0.0	0.0	0:00.36	kworker/0:0-inet_frag_wq
12	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/R-mm_pe
13	root	20	0	0	0	0	I	0.0	0.0	0:00.00	rcu_tasks_rude_kthread

- Quitter avec 'q'

Autres outils de visualisation

- htop : à installer : sudo apt install htop
  quitter par q



Nombre de processeurs : nproc

- Commande Linux issue du paquet coreutils
- Affiche le nombre de processeurs disponibles pour le processus courant
- Tient compte de :
  - Du nombre de cœurs physiques
  - Du Hyper-Threading (cœurs logiques)
  - De l'affinité CPU (cgroup, taskset, conteneurs Docker)
- Retourne un entier sur la sortie standard

nproc vs autres commandes

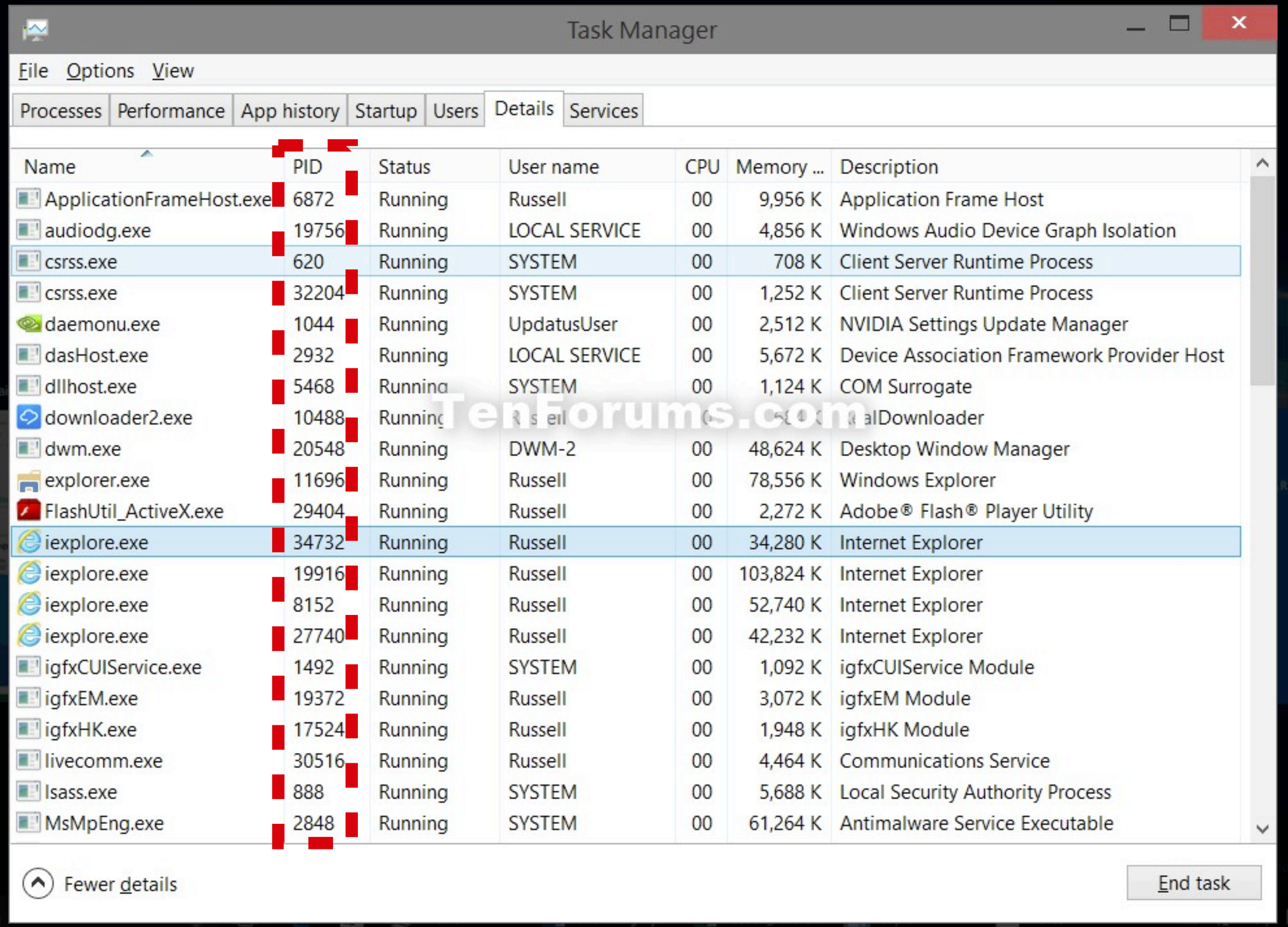
Commande	Ce qu'elle mesure
`nproc`	CPU **disponibles** pour le process
`nproc --all`	CPU **physiques/logiques totaux**
`lscpu`	Détails complets (archi, threads, sockets)
`grep -c ^processor /proc/cpuinfo`	Nombre de CPU vus par le noyau
`getconf _NPROCESSORS_ONLN`	CPU en ligne

👉 `nproc` est **le plus fiable dans un conteneur** (respecte les limites cgroup).


BEE TABLE

Windows

- Il existe les mêmes notions sur Windows
- Pour afficher les processus sur Windows :
  - Gestionnaire de tâches
  - Processus
  - Onglet « Détails »
  - Affichage : ajouter colonnes : PID, priorités, etc



TenForums.com

Exercice

- Retrouvez les différents états des processus de votre système. Vous les verrez dans la première colonne ‘S’ de la commande « ps –el ». Testez également : top

  – Notez les lettres employées par votre système pour les identifier (en anglais sur l’écran)
    - Sommeil :
    - Actif :
    - Suspendu :
    - Zombi :

Identification d'un processus

- Les processus sont repérés dans le système par un numéro : le **PID** (Process IDentification).
- Dans la norme **POSIX** d'Unix, ce PID est d'un type prédéfini dans **<unistd.h>** : **pid_t**
- En langage C, pour connaître l'identité du processus courant, il existe la fonction ***getpid()***, Syntaxe :




```
pid_t getpid() ; // Ne pas écrire cette ligne dans votre programme !
```

Exemple d'appel :

```
#include <sys/types.h>
#include <unistd.h>
pid_t pid;
pid = getpid();
```

Affichage de l'identification d’un processus

Exemple d’appel :

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

int main(void){
        pid_t pid;
        pid = getpid();
        printf( "PID %d \n ", (int)pid) ;
}
```

Vérification du PID

Exemple d’appel :

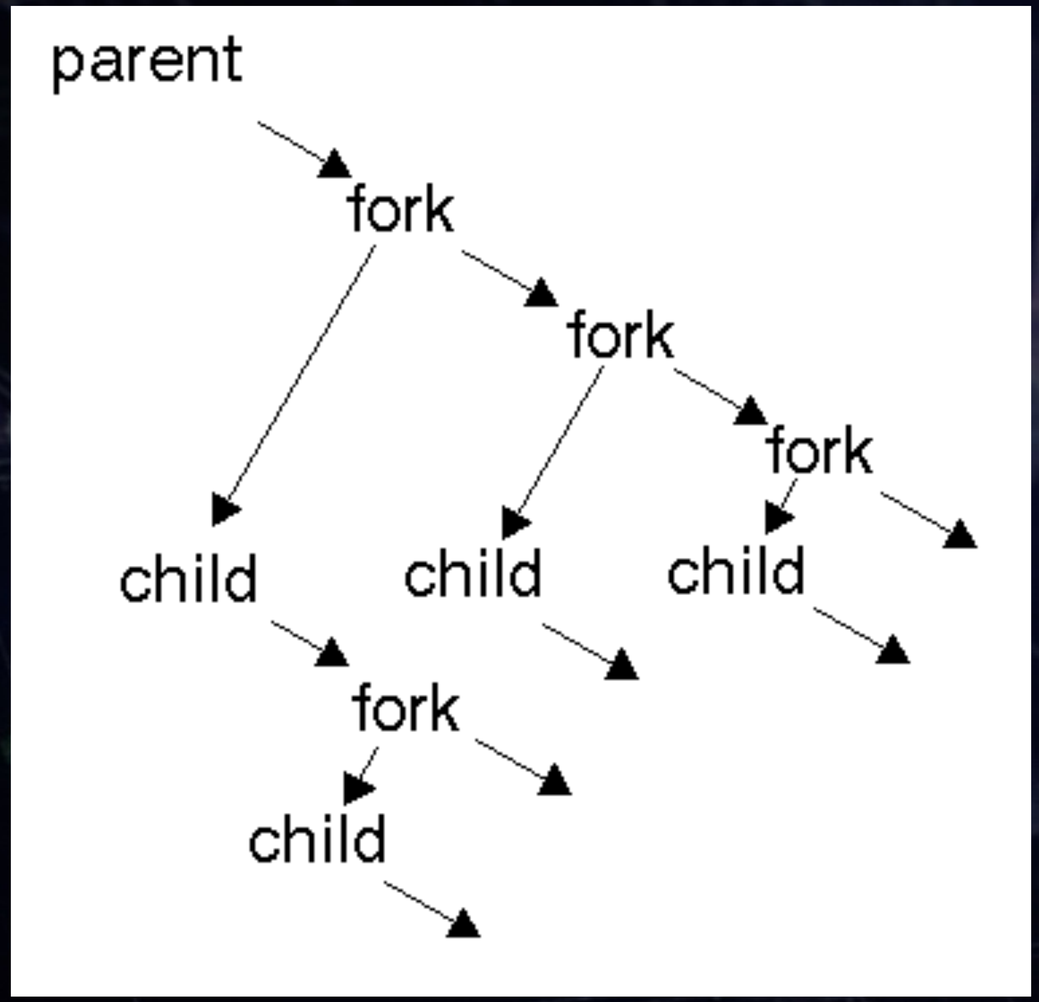
```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
int main(void){
        pid_t pid;
        pid = getpid();
        system("ps -l");
       printf( "PID %d \n ", (int)pid) ;


}
```

La création de processus

- Est réalisée par la primitive **fork()**
- Elle permet la création dynamique d'un nouveau processus qui s'exécute de façon concurrente avec le processus qui l'a créé.
- Tout processus Unix, excepté le processus originel, ***systemd*** (de PID 1), est créé par un appel à cette primitive. Visible avec `ps -p 1`
- Lors de la création d'un processus par un autre, le processus créateur s'appelle ***le père*** et le processus créé ***le fils***. La plupart des attributs du père (instructions, valeur des variables, …) sont hérités par le fils.

La création de processus



Relations père-fils

Type de la valeur renvoyée

- Syntaxe de fork():

```
pid_t fork() ; // Ne pas écrire cette ligne dans votre programme !
```

Exemple d'appel :

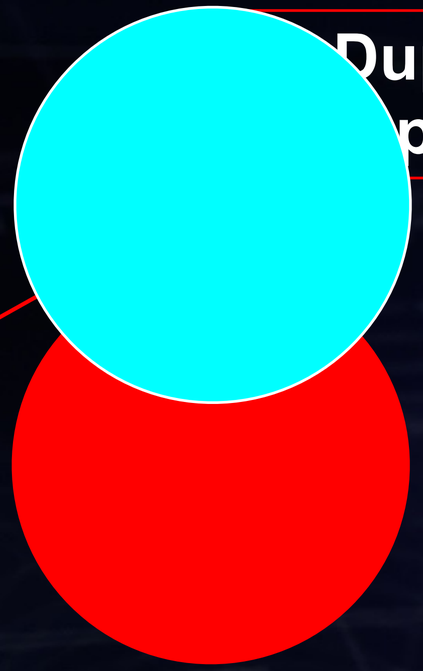
```
#include <sys/types.h>
#include <unistd.h>
pid_t retour;
retour = fork();
```

- Après l'appel à fork(), un processus fils est créé par le père. Ce processus exécutera le même programme que son père sur une copie des données de celui-ci.
- Si le fils modifie des données, le père ne le verra pas, car les données sont différentes, même si elles ont les mêmes valeurs au départ.
- Pour connaître le PID du père : `pid_t getppid();`

1 programme s'exécute

fork()

2 programmes s'exécutent en parallèle



Duplication du processus

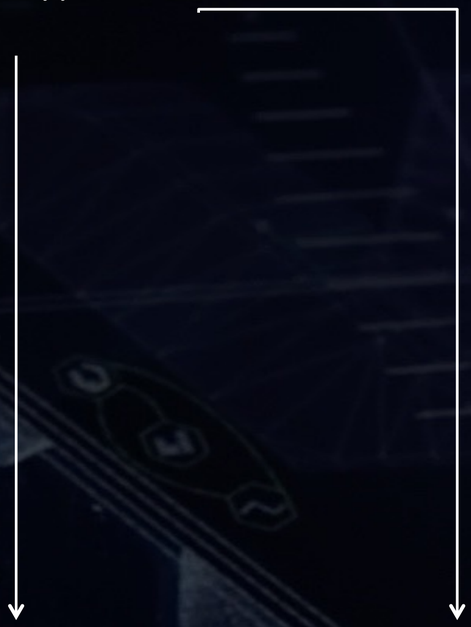
- fork() entraîne la création d’un processus fils qui est en tout point (ou presque) identique au père : même code, mêmes instructions.
- Le processus fils une fois lancé va donc également comporter un fork()
- Chacun des deux processus reprendra son exécution au même point et effectuera donc un retour d’appel de la fonction.
- Le programme étant identique, il est nécessaire, pour que le comportement des deux processus ne soit pas identique, de pouvoir distinguer le retour dans le processus père de celui dans le processus fils.

- La valeur de retour de la fonction a une valeur différente dans le processus père et dans le processus fils :
- 0 : dans le processus fils
- >0 : dans le processus père. La valeur renvoyée est le PID du processus fils.
- -1 : erreur lors de la création. Le fils n’a pas pu être créé.
- L’ordre dans lequel les différents instructions seront exécutées par les deux processus dépend de l’ordonnanceur du système.

Parallélisme

Un seul processus ici

```
pid_t retour;
retour = fork();
```



retour > 0
On est dans le processus père

retour = 0
On est dans le processus fils

Les deux processus s’exécutent en parallèle

Combien de lignes s'affichent ?

```
for (int i = 0; i < 3; i++) fork();

printf("coucou\n");
```

TRAVAUX PRATIQUES



Exercice PID

- Écrire un programme permettant d'afficher les PID :
  - Le processus doit afficher son PID, et le PID du processus père sur une même ligne
  - Comparer les PID affichés avec ceux donnés par le shell
  - Tester ce programme et faire valider le fonctionnement par le professeur
- Note : pour exécuter une commande système dans un programme : **system("ps –l"); Ajouter #include <stdlib.h> pour cette fonction**

Exercice faisable en C ou C++

- **Exemples en C** : compilation avec gcc
  - system("clear"); // Efface l'écran
  - printf("PERE :-PID : %d\n", getpid());
- **En C++** : compilation avec g++
  - *Ajouter :*
  - #include <iostream>
  - *en début de source et mettre :*
  - using namespace std ;

Exercice FORK

- Écrire un programme permettant de créer un processus fils :
  - Chaque processus doit afficher son PID , la valeur de retour de fork(), et le PID du processus père sur une même ligne
  - Commencer la ligne d'affichage par « PERE » dans le père et « FILS » dans le fils
  - Comparer les PID affichés avec ceux donnés par le shell
  - Tester ce programme et faire valider le fonctionnement par le professeur
- Note : pour exécuter une commande système dans un programme : **system("ps –l"); Ajouter #include <stdlib.h> pour cette fonction**

Résultat attendu

- Affichage à l'écran :

  PERE : PID : XXX PPID : YYY fork : AAA
  FILS : PID : ZZZ PPID : WWW fork : BBB
  Puis résultat de : system("ps -l")

Compte-rendu

- Réaliser un compte-rendu des exercices précédents en faisant une copie d'écran du résultat obtenu et en commentant et en entourant de couleurs les zones intéressantes. Montrez en particulier le parallèle entre les résultats affichés par votre programme et la commande "ps –l"
- Vous joindrez également le source à ce compte-rendu
- Quel est le processus qui a créé le processus « père » ?

Exemple de compte-rendu

Résultat :

```
ben@ben-VirtualBox:~$ proces
PERE : mon pid est 6711, le pid de mon père est 6387
                processus pére: fork a retourné au pére la valeur 6712
FILS : mon pid est 6712, le pid de mon père est 6711
                fork a retourné au pére la valeur 0
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000    6387    6345  0  80   0 -  4812 do_wai pts/0    00:00:00 bash
0 S  1000    6711    6387  0  80   0 -   622 do_wai pts/0    00:00:00 proces
1 S  1000    6712    6711  0  80   0 -   622 do_wai pts/0    00:00:00 proces
0 S  1000    6713    6711  0  80   0 -   652 do_wai pts/0    00:00:00 sh
0 R  1000    6714    6713  0  80   0 -  5007 -      pts/0    00:00:00 ps
0 S  1000    6715    6712  0  80   0 -   652 do_wai pts/0    00:00:00 sh
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
ben@ben-VirtualBox:~$ 0 S  1000    6387    6345  0  80   0 -  4812 do_wai pts/0    00:00:00 bash
1 S  1000    6712    1520  0  80   0 -   622 do_wai pts/0    00:00:00 proces
0 S  1000    6715    6712  0  80   0 -   652 do_wai pts/0    00:00:00 sh
0 R  1000    6716    6715  0  80   0 -  5007 -      pts/0    00:00:00 ps
```

: processus père

: processus fils

: fork

Exercice multitâches 1

- Écrire un programme C qui crée un fils, le père affiche les entiers de 1 à 5, et le fils affiche les entiers de 6 à 10.
- Fixer un intervalle d’une seconde entre l’affichage de chaque entier.
- On peut faire une pause par sleep(1) par exemple

Exercice multitâches 2

- Écrire un programme qui va créer un deuxième processus.
- Le père et le fils comptent de 0 à 100000 et l'affiche à l'écran.
- Le père place un P devant son comptage et le fils un F.
- Analysez le travail de l'ordonnanceur.

Exercice ZOMBI ORPHELIN

- Créer un programme afin de mettre en évidence l'état « zombi » : le fils se termine avant le père (on prolonge l'exécution du père par sleep(1) par exemple).

- Que se passe-t-il si le processus père se termine avant le processus fils ?

- Qui devient le père de ce processus fils ?

Compte-rendu

- Réaliser un compte-rendu de l'exercice précédent en faisant une copie d'écran du résultat obtenu et en commentant et en entourant de couleurs les zones intéressantes. Montrez en particulier le processus ZOMBI et le processus ORPHELIN
- Vous joindrez également le source à ce compte-rendu

Éviter les processus zombies

wait(NULL) est utilisé pour plusieurs raisons importantes :

1. Éviter les processus zombies

   Quand un processus fils se termine, il ne disparaît pas complètement du système immédiatement. Il devient un processus zombie - il a fini son exécution mais son entrée reste dans la table des processus. Le père doit appeler wait() pour "récolter" le statut de sortie du fils et libérer complètement ses ressources.

2. Synchronisation père-fils

   Sans wait(), le père pourrait se terminer avant le fils. Cela pourrait causer :

   Le fils devient orphelin (adopté par le processus init)

   La sortie du fils peut être tronquée si le terminal se ferme

   Le shell peut reprendre la main avant que le fils ait fini d'afficher

Éviter les processus zombies

3. Récupérer le statut de sortie

wait() retourne le statut de sortie du fils. On peut utiliser :

```
int status;
wait(&status);  // Récupère le statut
```