


# Système d'Exploitation (SE) Mémoire et Processus

Dr. Mohammed BOUGAA  
mbougaa@itescia.fr

ITESCIA\_2018/2019: Système d'Exploitation –Linux-

1



## Plan du cours

- ❖ SE et Gestion des Processus
- ❖ SE et Gestion de la Mémoire

2



## Partie 1: Gestion des Processus

3



### Processus

Un processus est un programme **en cours d'exécution**

- Programme (= instructions machine) : entité statique
- Processus (= réalisation d'actions) : entité dynamique

A un **processus** sont associées plusieurs éléments :

- son code
- ses données
- les ressources qui lui sont attribuées
- un ou plusieurs threads d'exécution

4

**Thread**

Un **thread** est un flot d'exécution dans le code du processus doté :

- d'un compteur programme (suivi des instructions à exécuter)
- de registres systèmes (variables de travail en cours)
- d'une pile (historique de l'exécution)

➤ Plusieurs processus permettent a un ordinateur d'effectuer plusieurs taches a la fois. Ils se partagent les **ressources physiques**.

➤ Plusieurs threads permettent a un processus de décomposer le travail a exécuter en parallèle. Ils se partagent les **ressources physiques et virtuelles**.

5

**Utilisation des thread**

Amélioration de l'interactivité d'un programme (ex : traitement de texte)

➤ Alors qu'un **processus** sert à lancer deux fois le même éditeur de texte par exemple.

6

**Exemple d'arborescence de processus :**

En **Bleu** ce sont des **Processus utilisateur** et en **Vert** ce sont des **Processus système** (démons)

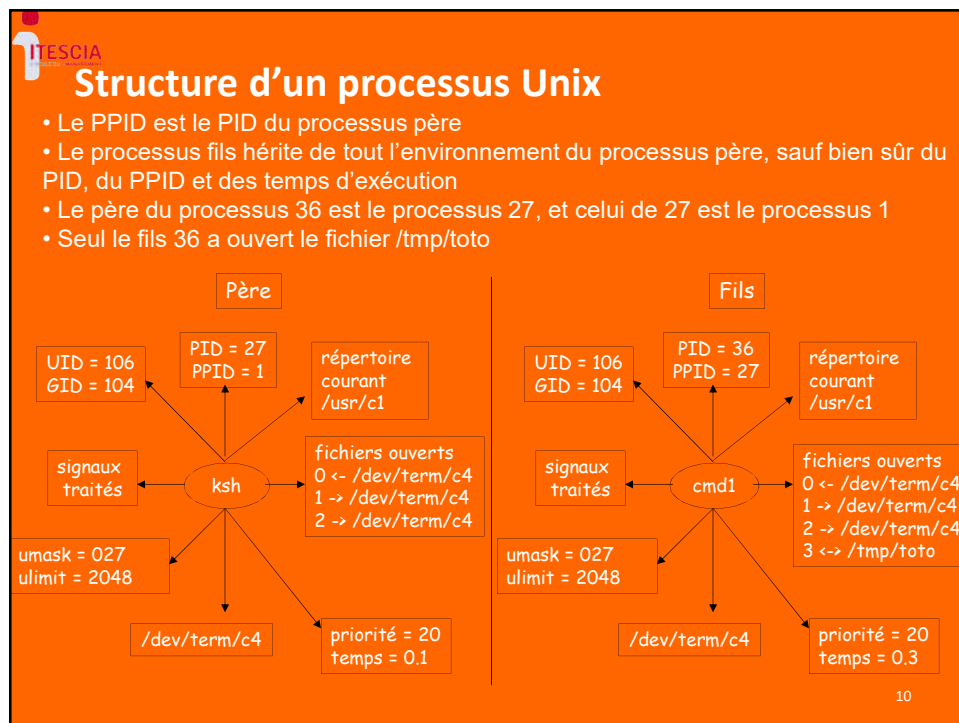
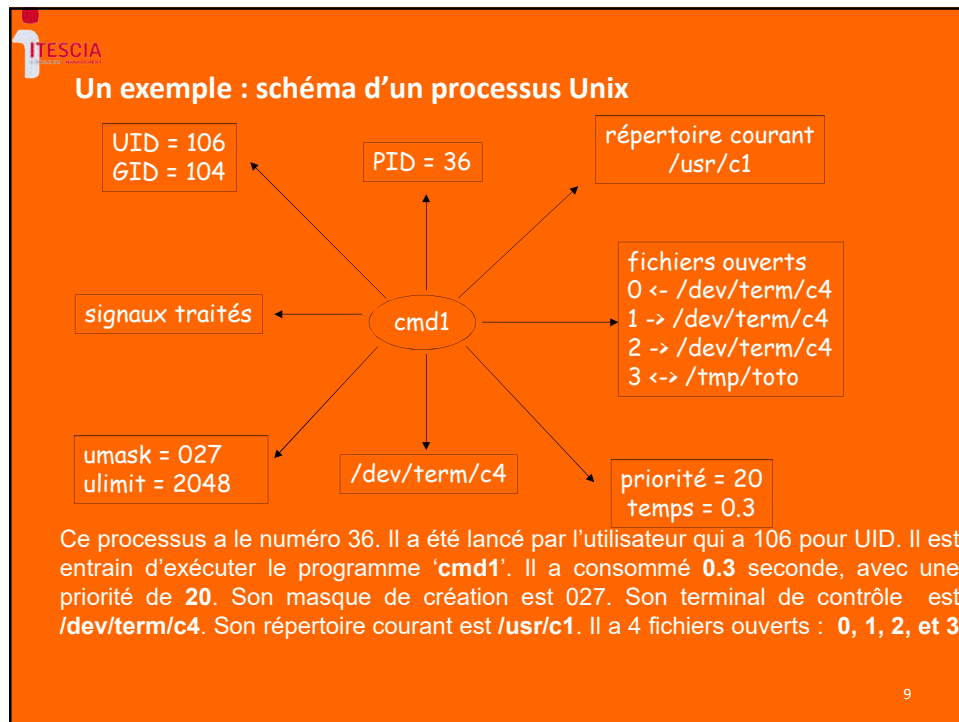
Un processus est créé par un autre processus (**init** est le père de tout les processus)


7

**L'environnement d'un processus UNIX**

- le programme qu'il exécute (**CMD**)
- un numéro d'identification que lui affecte le système (**PID**)
- un créateur (**PPID**)
- l'utilisateur pour lequel il fonctionne (**UID**)
- le terminal ou la fenêtre du processus (**TTY**)
- une consommation CPU (**CPU**)
- une consommation mémoire (**MEM**)
- une durée de traitement (**TIME**)
- une heure de lancement (**STIME**)
- un facteur de priorité (**C**)...
- les fichiers ouverts par ce processus
- ...

8






## Commandes UNIX de gestion des processus

### La commande ps

***ps [-options]***

- **-a** : affiche des renseignements sur tous les processus attachés à un terminal
- **-l** : donne, pour chaque processus, le nom de l'utilisateur (user), le pourcentage de cpu (%cpu), la taille totale du processus dans la mémoire (size), la mémoire réservée (rss) en Ko ...
- **-x** : affiche également des informations sur les processus non liés au terminal
- **-t** : ...

11



## Commandes UNIX de gestion des processus

### Commande ps –Exemple:-

% ps				
PID	TTY	STAT	TIME	CMD
746	pts/3	S	00:00:00	-bash
749	pts/3	S	00:00:02	gs
848	pts/3	S	00:03:28	mozilla-bin
965	pts/3	S	00:00:00	ps

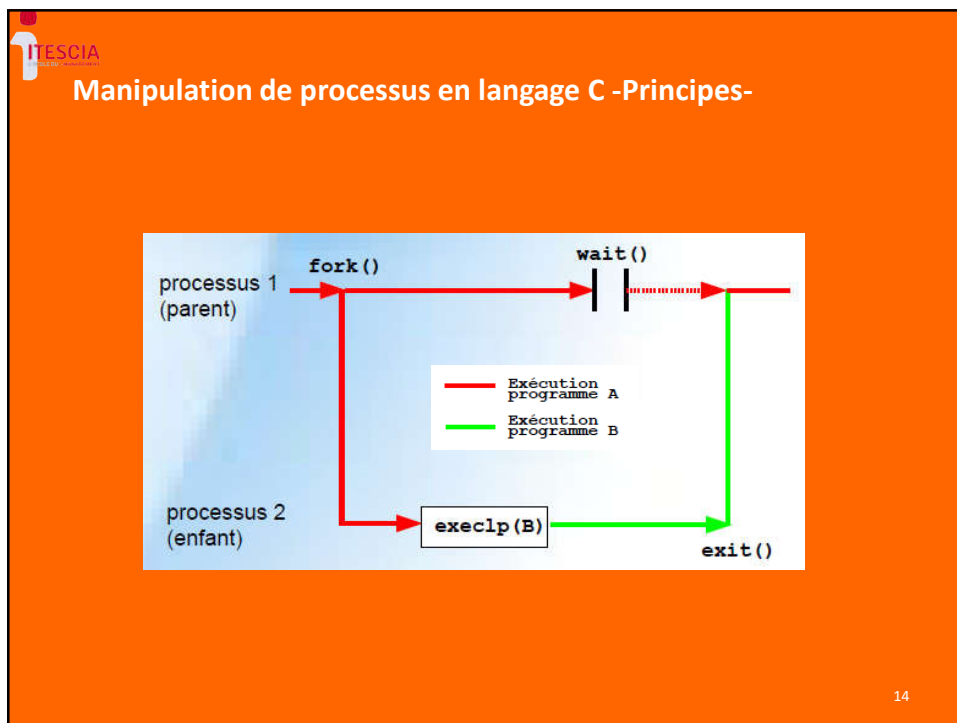
- PID : le numéro d'identification du processus
- TTY : le terminal depuis lequel le processus a été lancé
- STAT : l'état du processus au moment du lancement de la commande
  - R : le processus est en cours d'exécution
  - T : le processus est stoppé
  - S : le processus dort depuis moins de 20 secondes
  - I : le processus dort depuis plus de 20 secondes
  - Z : le processus en attente d'un message du noyau
- TIME : le temps d'exécution de la commande
- CMD : le libellé de la commande lancée

12

**Commandes UNIX de gestion des processus**

<b>La commande nice</b> Changement de la priorité d'un processus	<i>nice -valeur commande</i> <i>nice -5 gcc programme.c]</i>
<b>La commande kill</b> Destruction d'un processus	<i>kill -9 no_processus</i> <i>kill -9 521</i>
<b>La commande &amp;</b> Lancement en arrière-plan d'un processus	<i>nom_processus&amp;</i> <i>prog1&amp;</i>
<b>La commande jobs</b> Affichage des processus en background	<i>jobs</i>

13



**Manipulation de processus en langage C -Principes-**

Fonctions primitives de gestion de processus en C

- `int fork ();`
- `int execlp (char *comm, char *arg0, ..., NULL);`
- `void exit (int status);`
- `int wait (int *ptr_status);`
- `pid_t waitpid(pid_t pid, int *status, int opts)`
- `int sleep (int seconds);`
- `int getpid ();`
- `int getppid ();`

15

**Différents états d'un processus**

Les processus peuvent être dans plusieurs états, les plus courants étant :

```
graph TD; actif((actif)) -- "blocage" --> bloqué((bloqué)); bloqué -- "réveil" --> prêt((prêt)); prêt -- "élection" --> actif; actif -- "préemption" --> prêt;
```


**actif** : le processus s'exécute sur le processeur  
**prêt** : le processus est prêt à s'exécuter mais n'a pas le processeur  
**bloqué** : il manque au processus une ressource (en plus du processeur) pour qu'il puisse s'exécuter

16



**Allocation du processeur aux processus**

Le noyau gère l'ordonnancement de tous les processus du système  
Les processus peuvent être dans plusieurs états, les plus courants étant :



L'ordonnanceur est un composant (procédure) du noyau du système d'exploitation

L'ordonnancement consiste à **choisir** le processus à exécuter à un instant  $t$  et à **déterminer** le temps durant lequel le processeur lui sera alloué et d'optimiser certains aspects des performances du système

17

**Inter-Process Communication (IPC) : méthodes permettant à plusieurs processus de communiquer entre eux**

**3 catégories de mécanismes :**

Outils permettant aux processus de s'échanger des données

- les fichiers
- la mémoire partagée


Outils permettant de synchroniser des processus

- les sémaphores
- les signaux

Outils permettant d'échanger des données et de synchroniser des processus

- les tubes
- les files d'attente de messages

18




## Exemple de création de processus

### Exemple

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

int main() {
    pid_t pid;
    pid = fork();
    switch(pid) {
        case 0 :
            printf("Je suis le fils : mon PID est %d
                et mon PPID est %d\n", getpid(), getppid());
            break;
        case -1 :
            perror("Erreur de creation de processus avec fork");
            break;
        default :
            printf("Je suis le pere : mon PID est %d
                et mon PPID est %d\n", getpid(), getppid());
            break;
    }
    return 0;
}
```

19




## Exemple de synchronisation entre processus

### Synchronisation

- `exit(i)`
  - ▶ Termine un processus, `i` est un octet (donc valeurs possibles : 0 à 255) renvoyé dans une variable du type `int` au processus père.
- `pid_t wait(int *status)`
  - ▶ Suspend le processus jusqu'à la terminaison de l'un de ses fils
  - ▶ Retourne le pid du fils, ou -1 dans le cas où il n'y a pas (ou plus) de fils
  - ▶ Achèvement du père : fils pris en charge par `init`
  - ▶ `status` : valeur de `exit` ou autre (dans le cas d'un signal)
  - ▶ `waitpid` pour attendre un processus particulier dont on connaît son pid.

20




### Exemple de synchronisation entre processus

#### Exemple

```
#include <unistd.h> /* Symbolic Constants */
#include <sys/types.h> /* Primitive System Data Types */
#include <stdio.h> /* Input/Output */
#include <sys/wait.h> /* Wait for Process Termination */
#include <stdlib.h> /* General Utilities */

int main() {
    int retval; int status;
    pid_t p = fork();
    if (p == 0) {
        sleep(1); /* sleep for 1 second */
        printf("CHILD: Enter an exit value (0 to 255): ");
        scanf("%d", &retval);
        exit(retval);
    } else if (p > 0) {
        printf("PARENT: I will wait for my child to exit.\n");
        wait(&status);
        printf("PARENT: Child exit code is: %d\n",
            WEXITSTATUS(status));
        exit(0);
    } else { exit(-1); }
}
```


21



## Partie 2:

## La mémoire

22




## C'est quoi une Mémoire ?

On appelle « **Mémoire** » tout composant électronique capable de stocker temporairement des données

Caractéristiques principales d'une mémoire

- **La capacité**
- **Le temps d'accès**
- **Le temps de cycle**
- **Le débit**
- **La non volatilité**

23



## Différents types de Mémoire ?

### 3 types de memoires :

**Mémoire morte** (appelée également *mémoire non volatile*)

- Mémoire ROM (read-only memory)
- Mémoire ne s'effaçant pas en absence de courant électrique
- Mémoire conservant les données nécessaires au démarrage de l'ordinateur
- Temps d'accès de l'ordre de 150ns

**Mémoire vive** (appelée également *mémoire volatile*)

- Mémoire RAM (random access module)
- Données ne perdurant pas en l'absence de courant électrique
- 2 types de memoire RAM : DRAM et SRAM
  - Temps d'accès pour la DRAM de l'ordre de 50ns
  - Temps d'accès pour la SRAM de l'ordre de 10ns

**Memoire flash**

- Compromis entre la mémoire RAM et la mémoire ROM
- Non volatilité de la mémoire morte
- Accès en lecture/écriture de la mémoire vive

24


**Différents niveaux de gestion de la Mémoire ?**

**Niveau matériel**

- les registres du processeur
- la mémoire cache

**Niveau système d'exploitation**

- la mémoire principale (appelée également *mémoire centrale* ou *interne*)
- la mémoire secondaire (appelée également *mémoire de masse* ou *physique*)



25

**Mécanismes de gestion de la mémoire par le SE**

**Dans l'objectif de:**


- Partager la mémoire (système multi-tache)
- Allouer des blocs de mémoire aux différents processus
- Protéger les espaces mémoire utilisés

**Afin d'optimiser la quantité de mémoire disponible**

Le gestionnaire de la mémoire du SE, a recourt à deux familles de mécanismes:

- Mécanisme de mémoire **virtuelle**
- Mécanismes de **découpage** de la mémoire

26



## Mécanismes de gestion de la mémoire par le SE

**Mécanisme de mémoire virtuelle:** permettant d'exécuter des programmes dont la taille excède la taille de la mémoire réelle

**Mécanismes de découpage de la mémoire:**

- **Par pagination:** consiste à diviser la mémoire en blocs, et les programmes en pages de longueur fixe.
- **Par segmentation:** les programmes sont découpés en parcelles ayant des longueurs variables appelées segments.
- **Par segmentation paginée:** certaines parties de la mémoire sont segmentées, d'autres paginées

27



## La pagination:


La mémoire virtuelle et la mémoire physique sont structurées en unités d'allocation

- L'espace d'adressage virtuel est découpé en **pages**
- L'espace d'adressage physique est découpé en **cadres**

**Sachant que:**

- Taille d'une page = taille d'un cadre
- La taille d'une page est **fixe** (de 2Ko à 16Ko)
- Toutes les pages sont de la même taille
- Il peut y avoir plus de pages que de cadres (c'est là tout l'intérêt)

28




## La segmentation

Un **segment mémoire** est un espace d'adressage indépendant défini par 2 valeurs :

- Une **adresse** ou il commence (aussi appelée *base* ou *adresse de base*)
- Une **taille** ou un décalage (aussi appelé *limite* ou *offset*)

Cette adresse virtuelle est traduite en adresse physique par le biais d'une **table des segments**

29



## Pagination vs. Segmentation ?

**Pagination :**  
Sert à obtenir un grand espace d'adressage linéaire sans avoir à acheter de la mémoire physique

**Segmentation :**

- Permet la séparation des programmes et des données dans des espaces d'adressage logiquement indépendants
- Facilite le partage et la protection

30

**La segmentation paginée**

Chaque segment est composé d'un ensemble de pages

Une adresse virtuelle est définie par :

- un numéro de segment
- un numéro de page
- un déplacement dans la page

La traduction des adresses virtuelles en adresses physiques est réalisée grâce à une table des segments et une table des pages

31

**La mémoire sous Linux**

Le format des fichiers exécutables est divisé en régions (ou zones) :


- Type du fichier (les 2 premiers octets)
- Zone de code (**.text**)
- Zone des données (**.data**)
- Zone des données non initialisées (**.bss**)
- Zone de la pile

Chaque région est constituée d'un ensemble de pages de 4Ko

Le gestionnaire de mémoire est un **système de segmentation paginée**

32





## La mémoire sous Linux- Quelques fichiers et appels système

- Cartographie mémoire d'un processus → affichage du fichier **maps**
- Informations sur l'état de la mémoire utilisée par le processus → affichage du fichier **statm**
- Etat actuel d'un processus → affichage du fichier **status**
- Statistiques sur l'usage global de la mémoire virtuelle du système → appel système **vmstat**
- Modifie la taille d'un segment de données → appel système **brk(adr)**

33



## Bibliographie:

- [1] Cours de **Vincent Granet** (Polytech'Nice-Sophia)
- [2] Introduction à Unix et GNU / Linux, par Michael Opdenacker (Free Electrons), Traduction française par Julien Boibessot. Mise à jour Fabien Deleu (Département GTR de l'IUT de Béthune)
- Introduction à LINUX, de M. Abdallah ELKHYARI, Univ. Jean Monet St Etienne
- <https://moodle.polymtl.ca/mod/url/view.php?id=47398>
- Cours "Systèmes d'exploitation", Audrey Queudet, Univ Nantes, 2010

### Pour aller plus loin:

#### Livres

- *Linux pour les nuls*, Dee-Ann Leblanc, First Interactive, 2006.
- *Linux en pratique*, Arnold Robbins, Campus Press, 2007.
- *Linux programmation système et réseau , cours exemples et exercices corrigés en C-C++*, Joëlle Delacroix, Dunod, 2007

#### Sites web

- <http://www.linux.org/>
- <http://www.linux-france.org/>

34