

# Système d'exploitation des ordinateurs

Dr BOUYAKOUB F. M  
bouyakoub.f.m@gmail.com

# Présentation du module

- Intitulé du module:
  - Systèmes d'exploitation (SE)
- Objectifs du module:
  - Approfondir les différents concepts acquis pour la **conception** d'un système d'exploitation;
  - La programmation système.

## Contenu du module (1/3)

### Chapitre 0: Rappel sur les notions de base des SE et de la gestion de processus

- Les SE: Fonctionnalités et architectures;
- Processus et multiprogrammation;
- Cas des processus UNIX;
- Manipulation de processus et langage C;
- Processus et *scheduling*;

# Contenu du cours (2/3)

## Chapitre 1: Notion de parallélisme et synchronisation des processus

- Processus séquentiels et concurrents
- Problème de l'exclusion mutuelle
- Synchronisation
  - Sémaphores;
  - Événements;
  - Moniteurs;
  - Régions critiques.

## Chapitre 2: Communication inter-processus

- Partage de variables (modèle de producteur/consommateur et lecteurs/rédacteurs);
- Boîte aux lettres;
- Echange de messages (modèle du client/serveur);
- Etude de cas: Communication sous Unix (partage de segments, tubes, files de messages, sockets).

# Contenu du cours (3/3)

## Chapitre 3: L'interblocage

- Modèles, représentation
- Traitement
- Prévention,
- Evitement,
- Détection/ Guérison

## Chapitre 4: Système de gestion de fichier

- Rappels sur l'interface des systèmes de fichiers
- Structure d'un système de fichiers (organisation, montage)
- Organisation physiques des fichiers (allocation contiguë, chaînée, indexée) et gestion de l'espace libre (vecteur binaire, liste chaînée, groupement)
- Implémentation des répertoires (linéaire, table de hachage)
- Gestion des fichiers actifs: partages de fichiers

## Supports de cours

- Les cours sont téléchargeable sur mon site:

<https://sites.google.com/site/bouyakoubfaycal/e-textbooks>

- Sur facebook:
  - Groupe Wiki-Sys

## Chapitre 0: Rappel sur les notions de base des SE

Présentation du module  
Les SE: Fonctionnalités et architectures  
Processus et multiprogrammation  
Cas des processus UNIX  
Manipulation de processus en langage C  
Processus et *scheduling*

# Les SE: fonctionnalités et architectures

## Définition d'un système d'exploitation (1/2)

- Le système d'exploitation (*SE*, en anglais *Operating System* ou *OS*) est un ensemble de ***programmes*** responsables de la liaison entre les **ressources matérielles** d'un ordinateur et les **applications informatiques** de l'utilisateur
- Le SE gère aussi l'ensemble des pilotes (drivers) permettant aux applications d'accéder à des périphériques évolués (modem, carte réseau, carte vidéo...)
- Il assure aussi l'enchaînement des travaux, l'affectation des ressources aux programmes, la gestion des entrées-sorties et des fichiers, la communication homme-machine via une interface conviviale, etc.



## Définition d'un système d'exploitation (2/2)

Nous pouvons considérer un SE comme:

- Un distributeur de ressources
  - Dans un environnement de plusieurs utilisateurs exécutant des programmes, leurs requêtes doivent être satisfaites de façon équitable et efficace: temps processeur, espace mémoire, espace de stockage des fichiers...
- Un programme de contrôle, qui contrôle l'exécution des programmes utilisateurs pour éviter l'utilisation incorrecte de l'ordinateur, et contrôle les périphériques d'E/S et leur exploitation.

## Les fonctions d'un SE (1/3)

### 1. Gestion du processeur:

Le SE est chargé de gérer l'allocation du processeur entre les différents programmes grâce à un *algorithme d'ordonnancement*.

### 2. Gestion de la mémoire vive:

Le SE est chargé de gérer l'espace mémoire alloué à chaque application et à chaque utilisateur.

En cas d'insuffisance de mémoire physique, le SE peut créer une zone mémoire sur le disque dur, appelée ***mémoire virtuelle***.

## Les fonctions d'un SE (2/3)

### 3. Gestion des entrées/sorties:

Le SE permet d'unifier et de contrôler l'accès des programmes aux ressources matérielles par l'intermédiaire des pilotes (appelés également gestionnaires de périphériques ou gestionnaires d'entrée/sortie).

### 4. Gestion de l'exécution des applications:

Le SE est chargé de la bonne exécution des applications en leur affectant les ressources nécessaires à leur bon fonctionnement.

# Les fonctions d'un SE (3/3)

## 5. Gestion des droits:

Le SE est chargé de la sécurité liée à l'exécution des programmes en garantissant l'utilisation des ressources que par les programmes et utilisateurs possédant les droits adéquats.

## 6. Gestion des fichiers:

Le SE gère la lecture et l'écriture dans le système de fichiers et les droits d'accès aux fichiers par les utilisateurs et les applications.

## 7. Gestion des informations:

Le SE fournit un certain nombre d'indicateurs permettant de diagnostiquer le bon fonctionnement de la machine.

# Architecture d'un système

Divisée en 3 couches distinctes

- La couche physique (Périphériques et BIOS)
- La couche système
- La couche interface

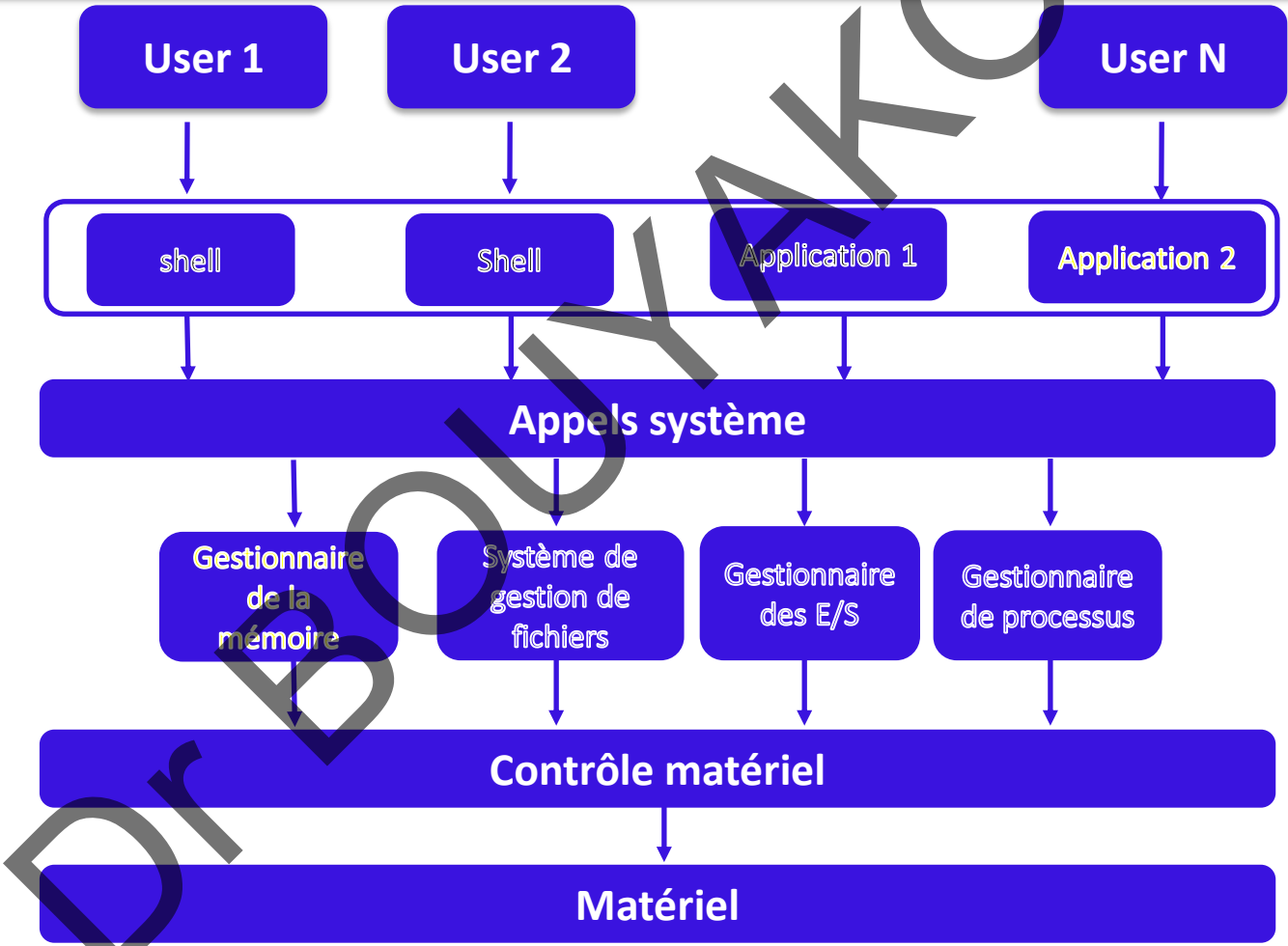
## La couche système

La couche système ou le **noyau** (en anglais **kernel**) intègre les fonctions fondamentales du SE telles que la gestion de la mémoire, des processus, des fichiers, des E/S principales, et des fonctionnalités de communication.

# Architecture du noyau (1/2)

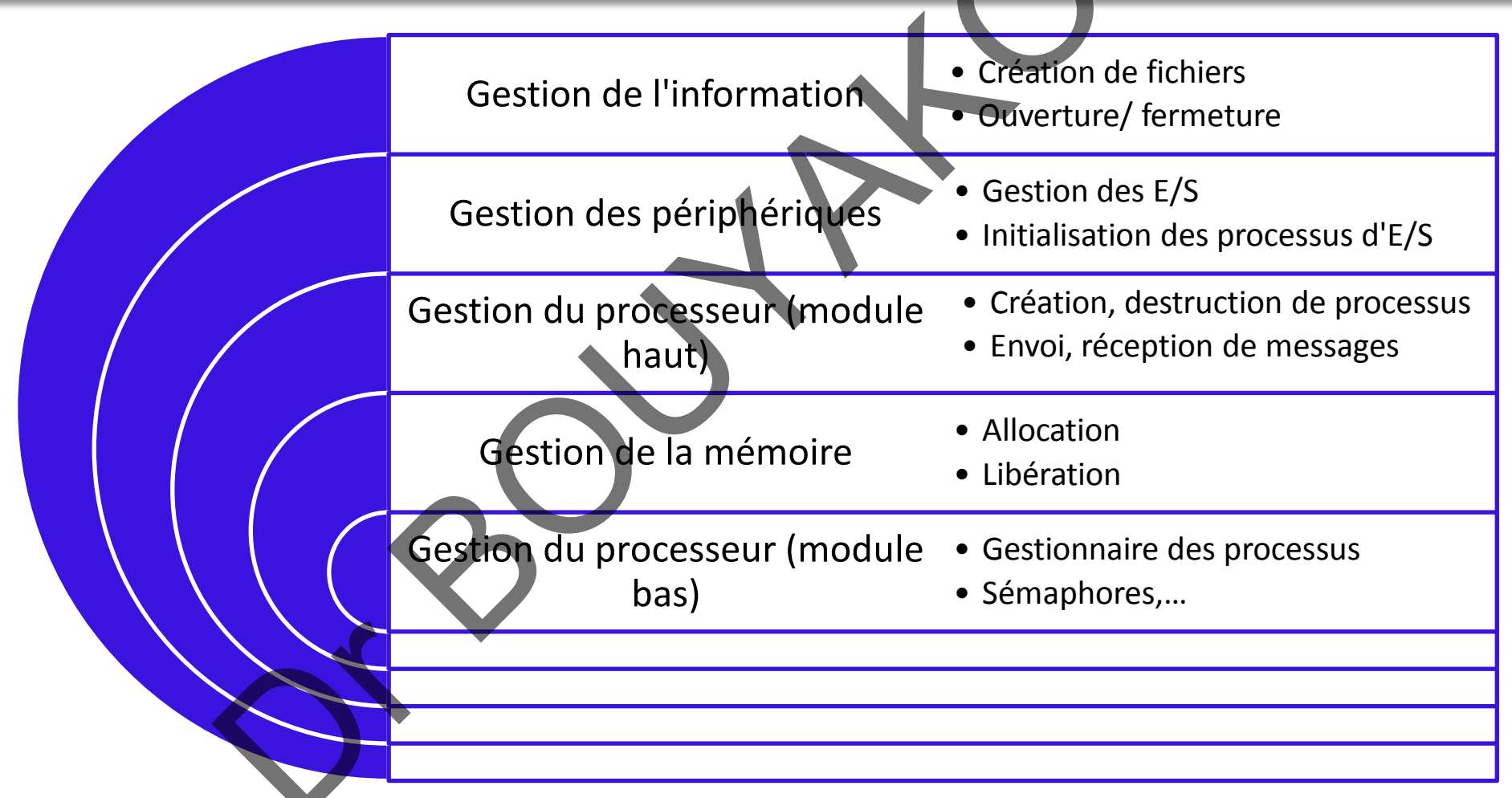
- C'est une architecture logicielle qui permet aux développeurs de structurer le travail de développement.
- Le but est de simplifier la compréhension et le développement du système. Nous cherchons donc ici à décomposer le noyau en parties disjointes (qui sont concevables et programmables de façons disjointes).

# Architecture du noyau (2/2)





# Structure hiérarchique du système



## Chapitre 0: Rappel sur les notions de base des SE

Présentation du module  
Les SE: Fonctionnalités et architectures  
Processus et multiprogrammation  
Cas des processus UNIX  
Manipulation de processus en langage C  
Processus et *scheduling*

# Processus et multiprogrammation

# Notion de processus

Deux aspects sont liés à la notion de processus:

## 1. Gestion des processus

- Création, manipulation, synchronisation...

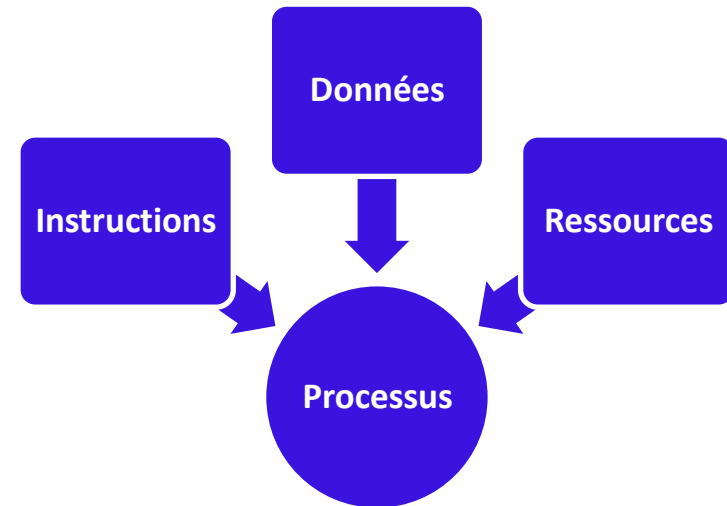
## 2. Ordonnancement des processus (*scheduling*)

- Un SE doit gérer plusieurs processus en même temps
  - 1 processeur et N processus → pseudo-parallélisme
  - S'il y a plusieurs processeurs, l'exécution des processus est distribuée de façon équitable sur ces processeurs.

## Chapitre 0: Rappel sur les notions de base des SE

# Définition d'un processus

- Un processus est l'instance d'un programme en cours d'exécution à un instant  $T$  et son environnement d'exécution.
- Un processus est défini par:
  - Un environnement processeur
  - Un environnement mémoire



# Caractéristiques des processus

## Caractéristiques statiques

- PID: *Process IDentifier*
- PPID: *Parent PID*
- UID: *User ID*
- ...

## Caractéristiques dynamiques

- Quantité de ressources consommée (cpu, mémoire...)
- Etat
- ...

# Type de processus

## Type de processus

Processus système

Processus utilisateur

Processus en arrière-plan  
appelés *Daemon* sous Unix  
et *services* sous Windows

Processus interactif

Correspond à l'exécution  
d'une commande ou d'un  
programme

# Composition d'un processus (1/2)

## 1. L'espace d'adressage

- **Zone programme ou code:** correspond aux instructions en langage machine, du programme à exécuter.
- **Zone données (variables) ou DATA:** contient les variables globales ou statiques du programme.
- **Zone pile utilisateur:** sert pour les appels de fonctions avec leurs paramètres et leurs variables locales.

## Chapitre 0: Rappel sur les notions de base

## Composition d'

### 2. Le bloc d

- Le SE reg processus  
**de Contrôle**  
**Block PCB**
- A la créa correspon
- Ce **PCB** l'ic système.

Identificateur processus
État du processus
Compteur ordinal contexte pour reprise (registres et pointeurs, piles...)
Chaînage selon les files de l'ordonnanceur priorité (ordonnancement)
Informations mémoire (limites et tables pages / segments)
Informations sur les ressources utilisées fichiers ouverts, outils de synchronisation, entrées-sorties
Informations de comptabilisation

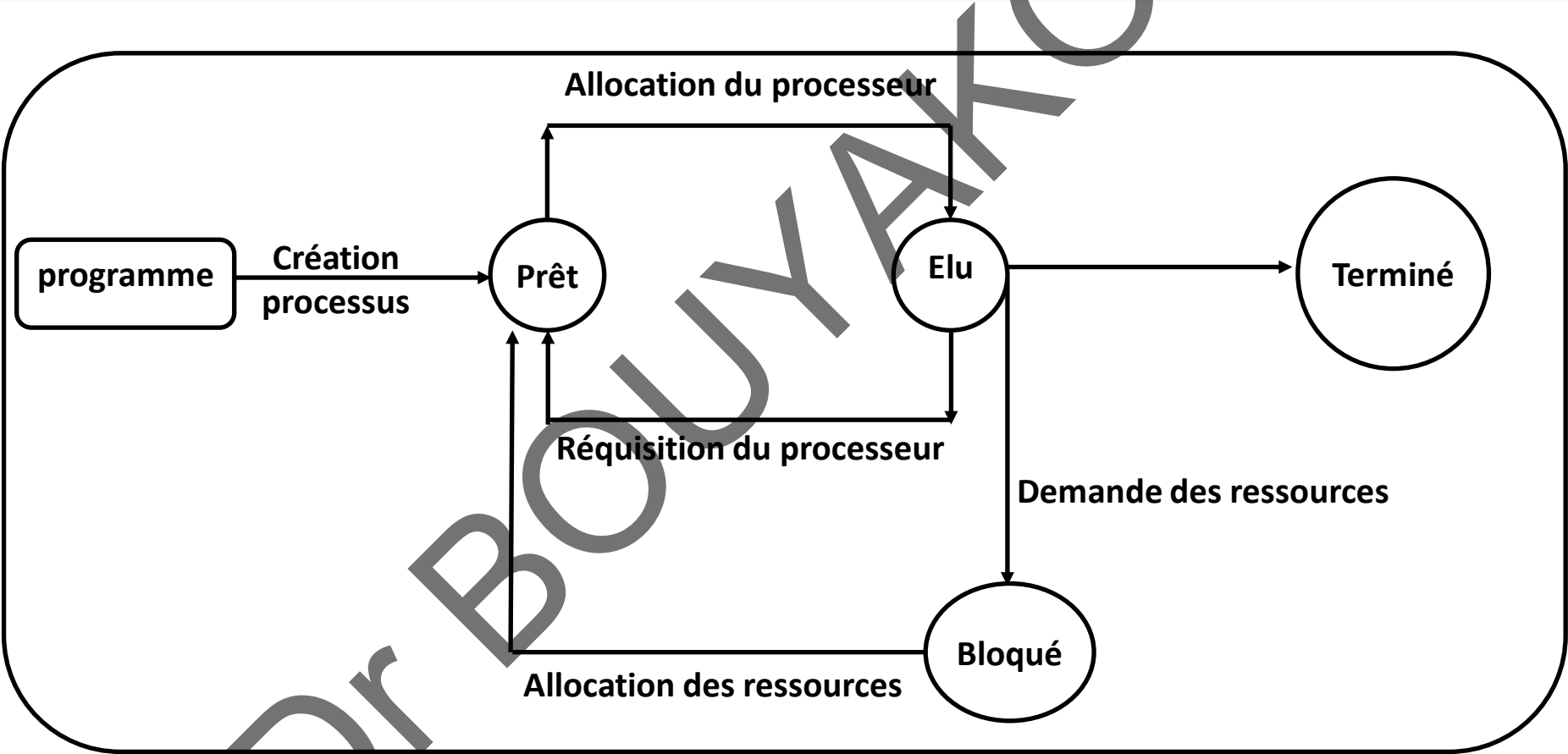
; relatives à un  
es appelée **Bloc**  
**Process Control**

crée son **PCB**

e de vie dans le



# Etats de base d'un processus



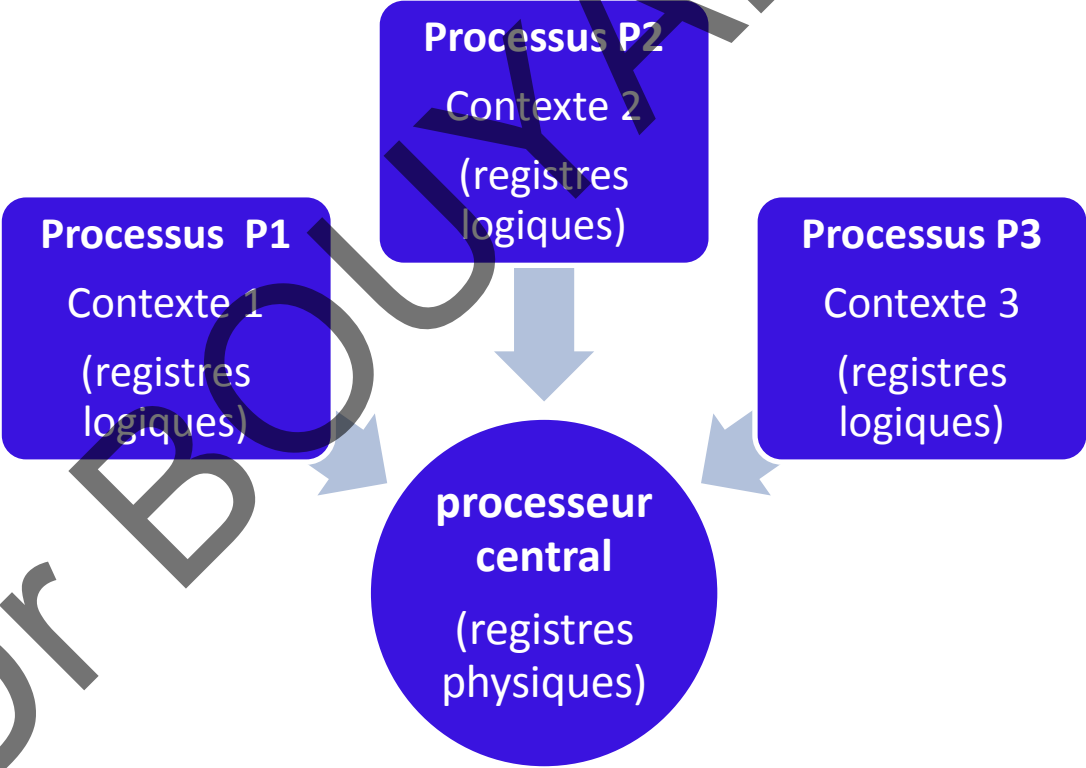
## Chapitre 0: Rappel sur les notions de base des SE

# Contexte d'un processus

- Le contexte d'un processus représente toutes les informations concernant ce processus durant son exécution
- Le contexte d'un processus est composé du contenu des éléments suivants: CO, PSW, RG, RP.
- L'exécution d'un processus nécessite le chargement de son contexte dans le processeur  
→ registres logiques et physiques

# Contexte d'un processus et multiprogrammation

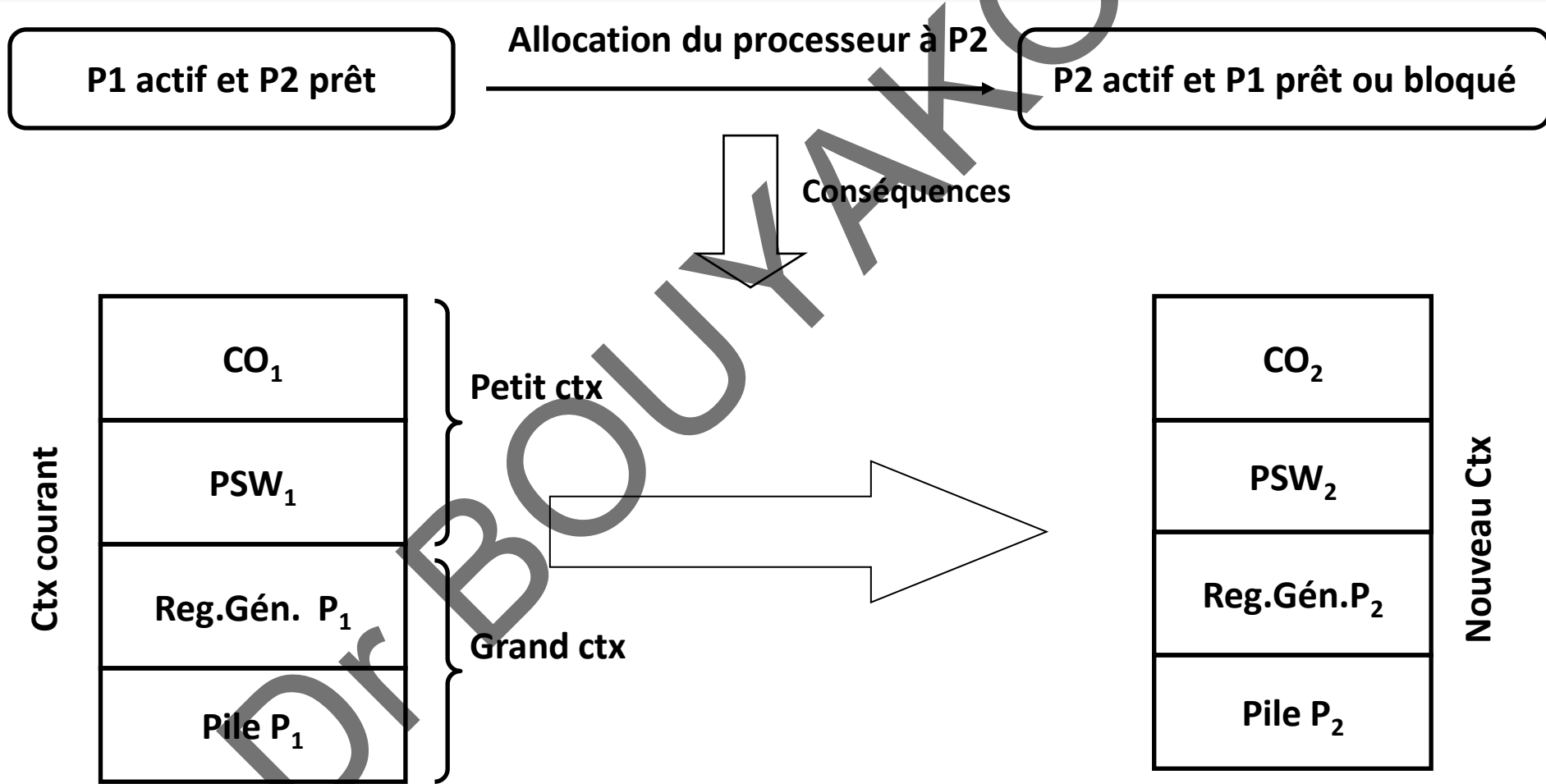
$N \text{ processus} + 1 \text{ processeur} = \text{commutation de contexte}$



## Mécanismes de commutation de contexte (1/2)

- Le passage dans l'exécution d'un processus à un autre nécessite une opération de sauvegarde du contexte du processus interrompu (pour reprendre son exécution) et de chargement de celui du nouveau processus.  
→ **Commutation du contexte.**
- La commutation de contexte consiste à changer les contenus des registres du processeur central par les informations de contexte du nouveau processus à exécuter.

# Mécanismes de commutation de contexte (2/2)

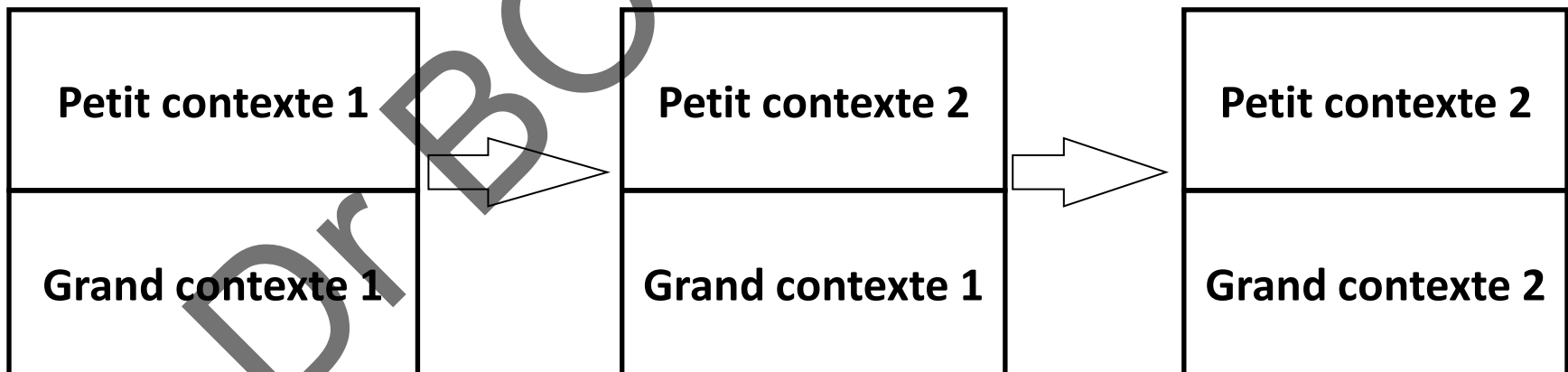


## Chapitre 0: Rappel sur les notions de base des SE

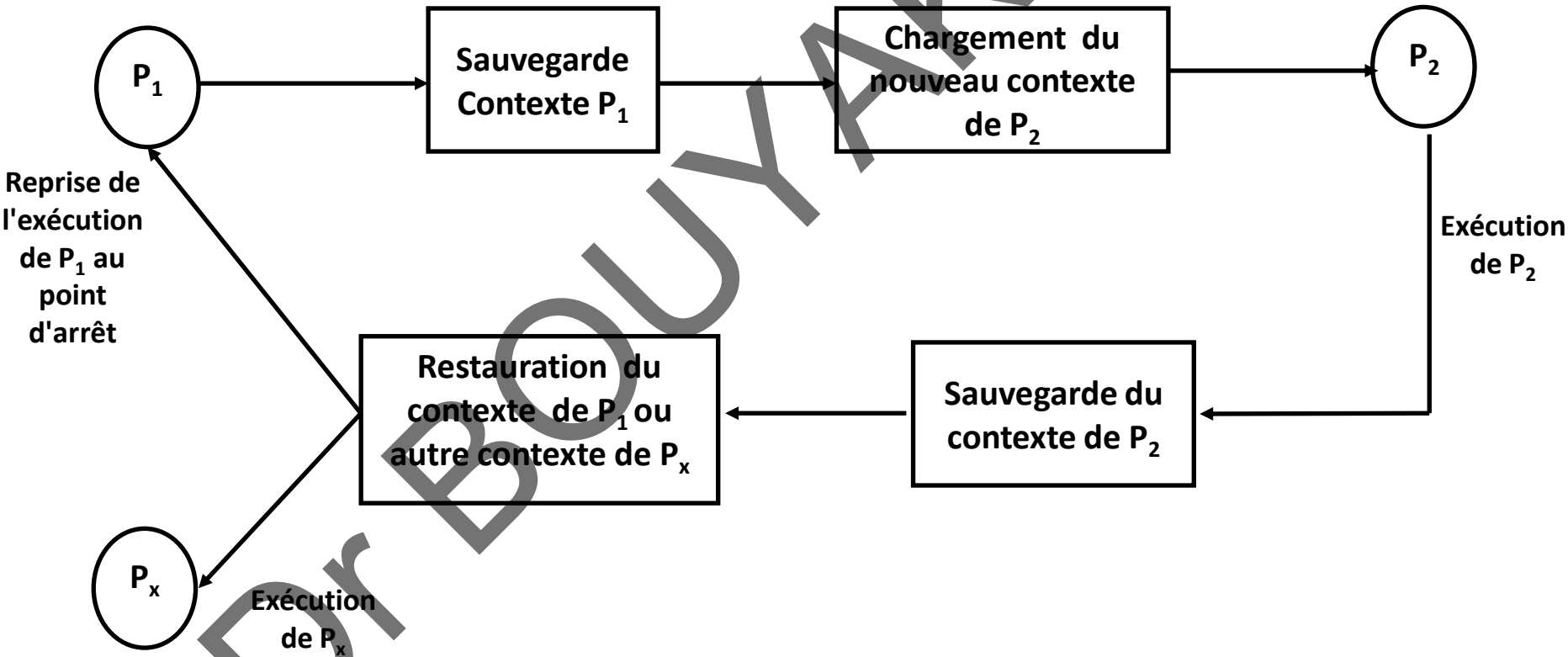
# Les étapes de la commutation du contexte

La commutation du contexte se fait en deux phases:

1. C'est le petit contexte (co, psw) qui est commuté par une instruction ***indivisible***.
2. Commutation du grand contexte par le nouveau processus.



# La restauration du contexte



## Chapitre 0: Rappel sur les notions de base des SE

# Opérations sur les processus: la création

- Un processus peut créer un ou plusieurs processus → utilisation d'un appel système de création.
- Le processus créateur est appelé le processus père (représenté par son **PPID** ou **Parent PID**) tandis que le ou les processus créés constituent les processus fils (représentés par leur **PID**).
- Ces processus fils peuvent à leur tour créer des processus, devenant ainsi également des processus père.  
→ arborescence de processus.



# Opérations sur les processus: la destruction

Lors de la destruction d'un processus les ressources qui lui sont allouées sont libérées et son PCB est détruit.

La destruction d'un processus intervient:

1. Lorsque le processus termine son exécution
  - ➔ Dans ce cas, le processus s'autodétruit en appelant une routine système de fin d'exécution;
2. Lorsque le processus commet une erreur irrécouvrable
  - ➔ Dans ce cas, un déroutement est déclenché et le processus est terminé par le système;
3. Lorsqu'un autre processus demande la destruction du processus, par le biais d'un appel à une routine système.

## Opérations sur les processus: la suspension

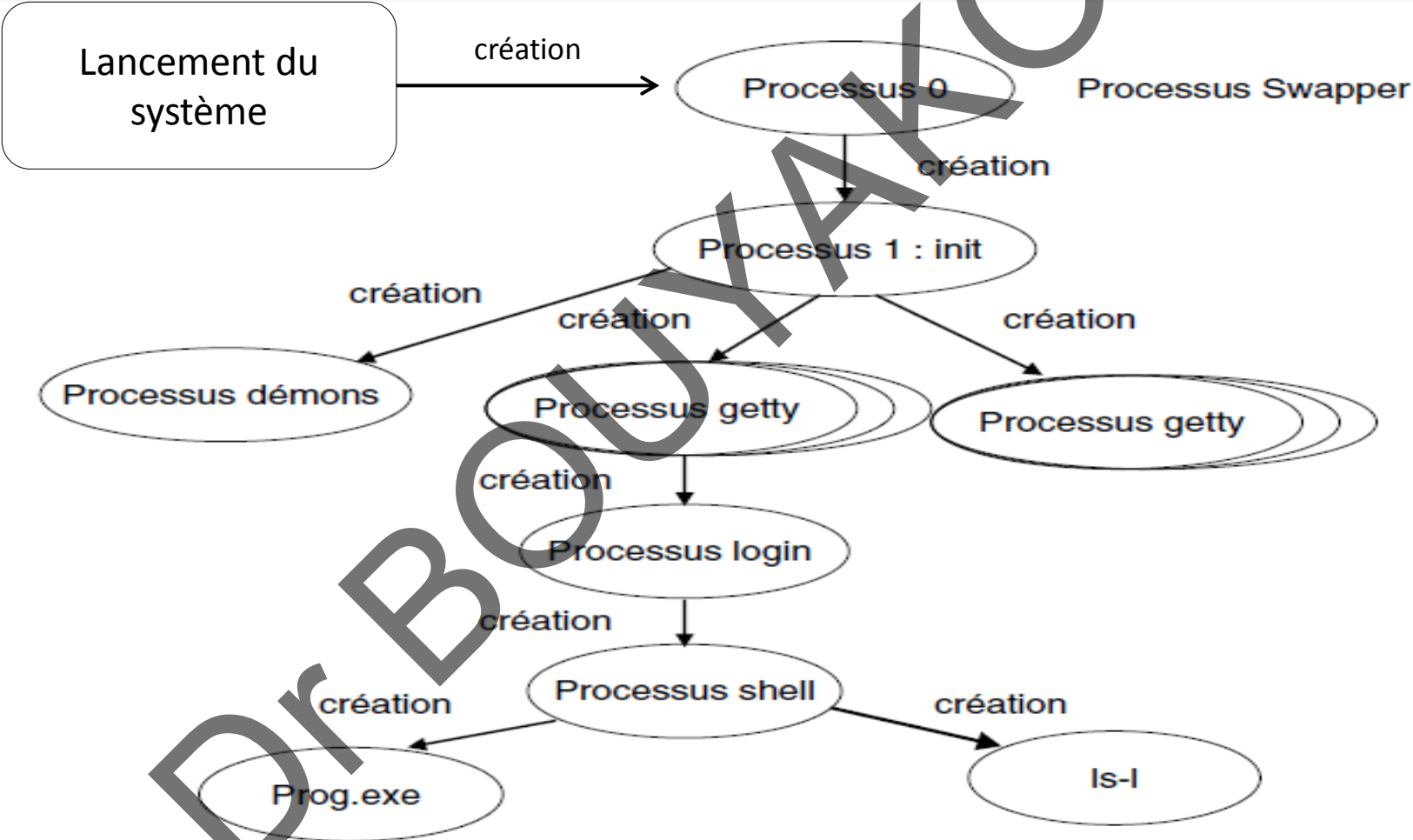
- La suspension d'exécution est une opération qui consiste à arrêter, momentanément, l'exécution d'un processus pour la reprendre ultérieurement. Lors de l'opération de suspension, le contexte du processus est sauvegardé dans son PCB afin de pouvoir reprendre l'exécution, là où elle a été suspendue.

## Chapitre 0: Rappel sur les notions de base des SE

Présentation du module  
Les SE: Fonctionnalités et architectures  
Processus et multiprogrammation  
Cas des processus UNIX  
Manipulation de processus en langage C  
Processus et *scheduling*

# Etude de cas: les processus sous Unix

# Arborescence de processus sous Unix

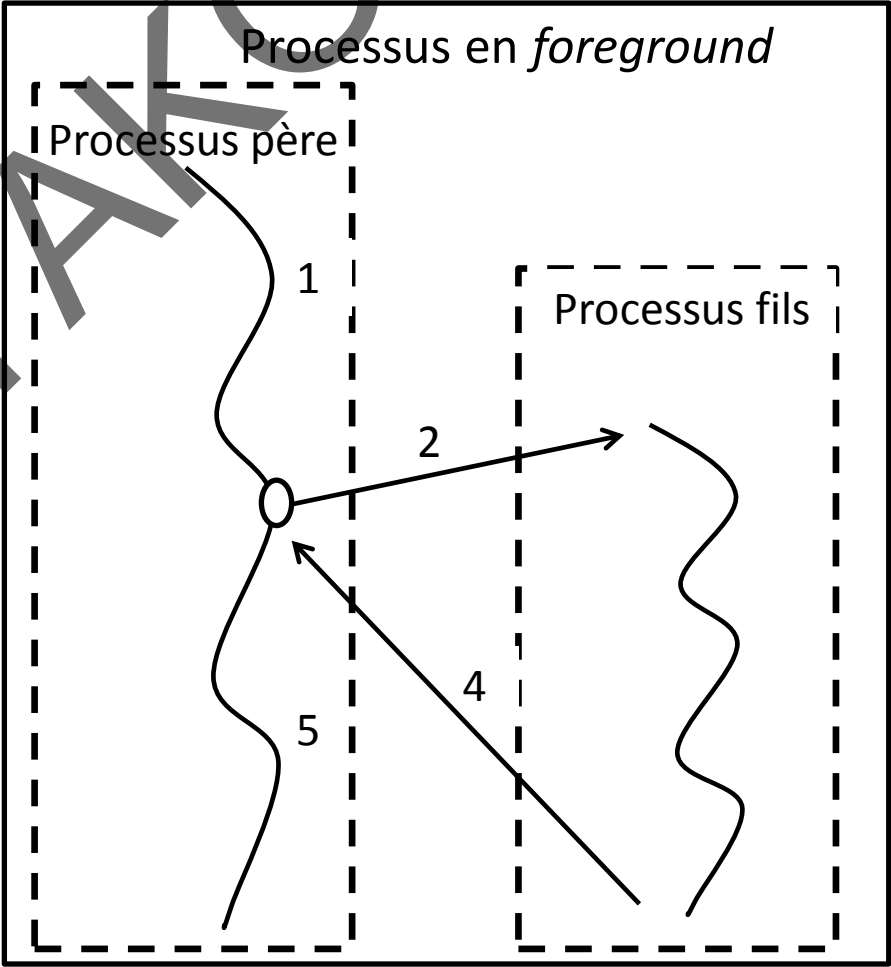
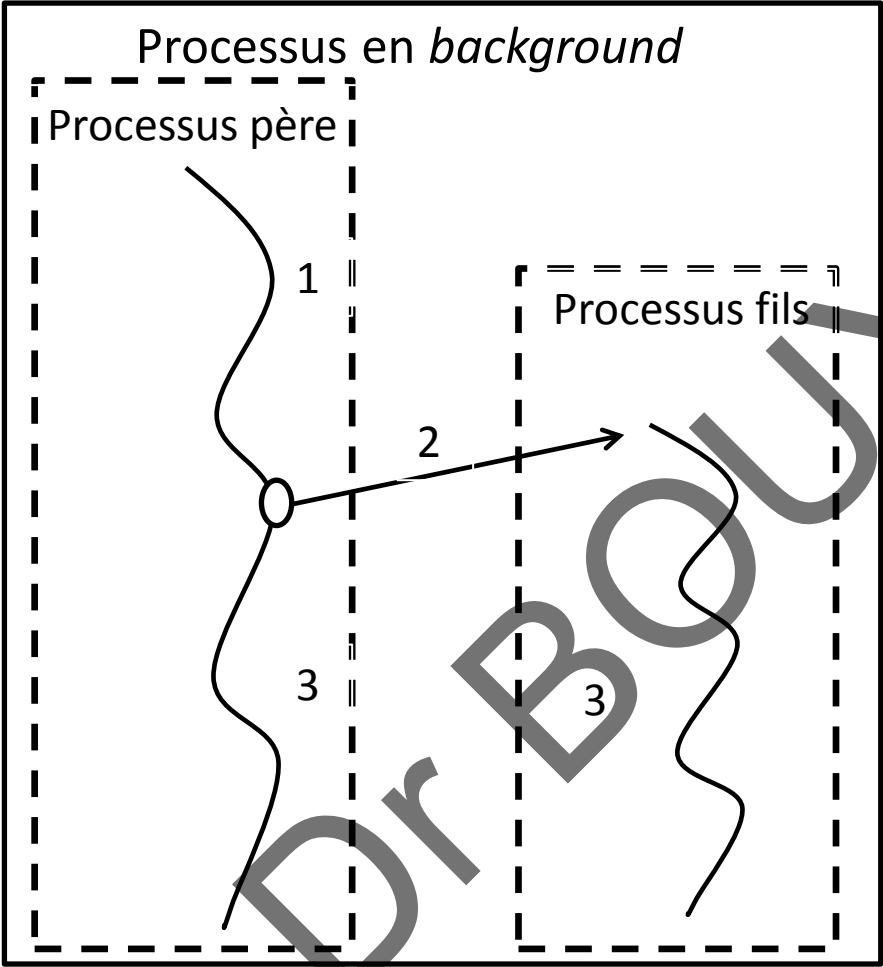


## Chapitre 0: Rappel sur les notions de base des SE

# Création de processus

- A bas niveau, il n'y a qu'une seule façon de créer un nouveau processus, la "**duplication**" via la primitive **fork()**.
- Le système crée une copie complète du processus père (même code) avec un **PID** différent.
- Après duplication, le fils va **changer de programme** en utilisant la primitive **exec()** qui conserve l'identité du processus mais remplace son code exécutable et ses données par celui d'une nouvelle commande.
- Le processus fils hérite du processus père des informations comme: la priorité, le propriétaire...

# Processus en *background* et *foreground*

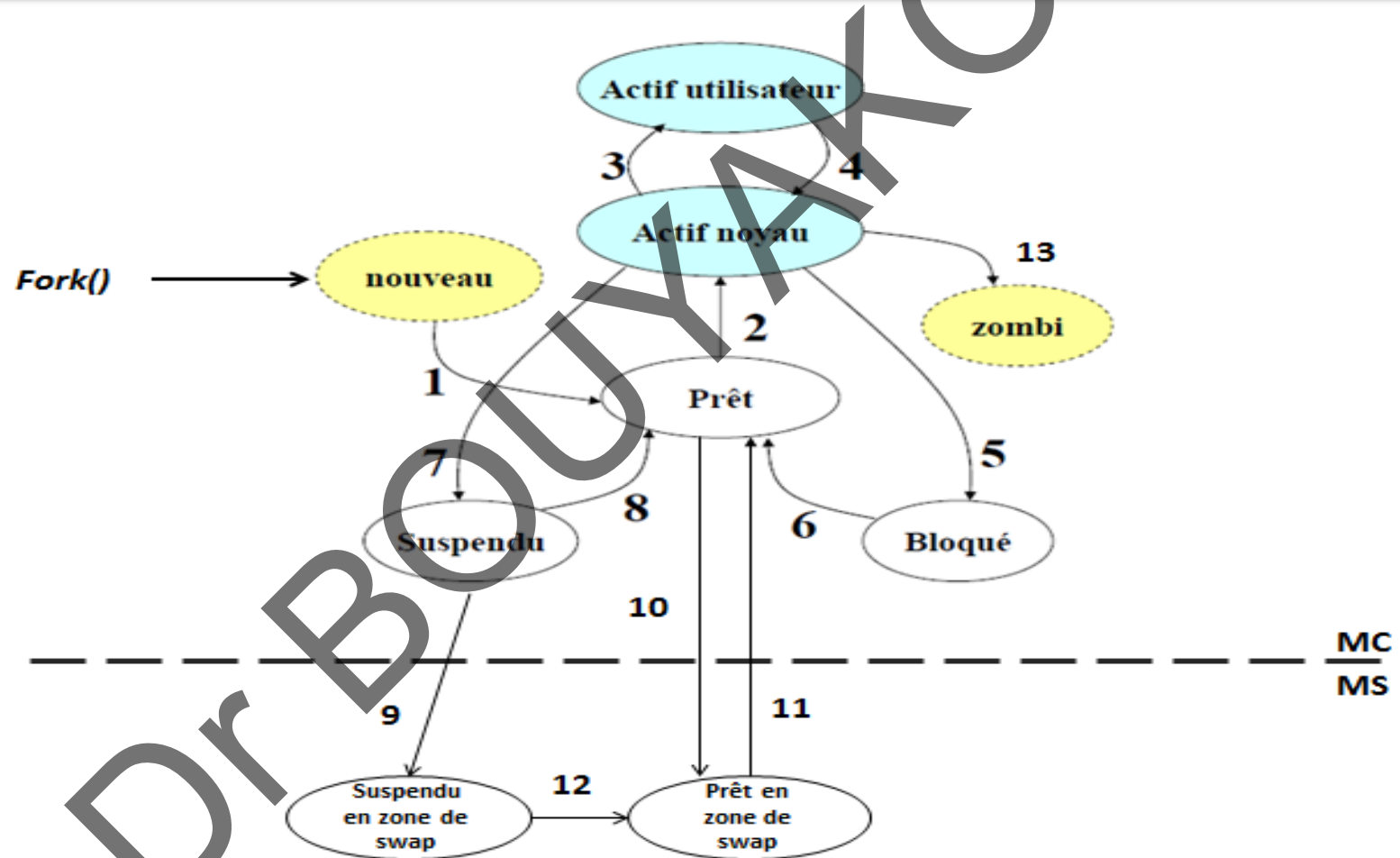


# Les états d'un processus sous Unix

- **Les états standards:**
  - *Initialisation* (en anglais, *created* ou *new*);
  - *Prêt* ou *En attente* (en anglais, *ready* ou *runnable*);
  - *Élu* ou *Exécution* (en anglais, *running* );
  - *Endormi* ou *Bloqué* (en anglais, *blocked* ou *waiting*);
  - *Terminé* (en anglais, *terminated*);
- **Les états particuliers:**
  - *Zombi*: Si un processus terminé ne peut pas être déchargé de la mémoire, par exemple, si un de ses fils n'est pas terminé, il passe dans un état appelé *zombi*.
  - *Swappé* : Lorsque qu'un processus est transféré de la mémoire centrale vers la mémoire virtuelle, il est dit « swappé ».
  - *Exécution en mode utilisateur* : L'exécution a lieu dans un espace limité, seul certaines instructions sont disponibles.
  - *Exécution en mode noyau*: L'exécution du processus n'est pas limitée. Par exemple, un processus dans cet état peut aller lire dans la mémoire d'un autre.

## Chapitre 0: Rappel sur les notions de base des SE

# L'automate d'états d'un processus





## Chapitre 0: Rappel sur les notions de base des SE

Présentation du module  
Les SE: Fonctionnalités et architectures  
Processus et multiprogrammation  
Cas des processus UNIX  
Manipulation de processus en langage C  
Processus et *scheduling*

# Manipulation de processus en langage C

## Création de processus: la primitive *fork()* (1/2)

- Sous LINUX la création de processus est réalisée, en langage C, par l'appel système:

```
pid_t fork();
```

- Valeur retournée par *fork()*:
  - Dans le processus père, retourne le PID du fils.
  - Dans le processus fils, retourne 0.
  - Si le *fork()* échoue, renvoie -1:
    1. Le nombre maximum de processus en exécution par l'utilisateur est atteint (variable suivant les systèmes);
    2. Il ne reste pas suffisamment de mémoire système disponible pour dupliquer le processus;
    3. Il n'y a pas assez d'espace *swap*.

## Chapitre 0: Rappel sur les notions de base des SE

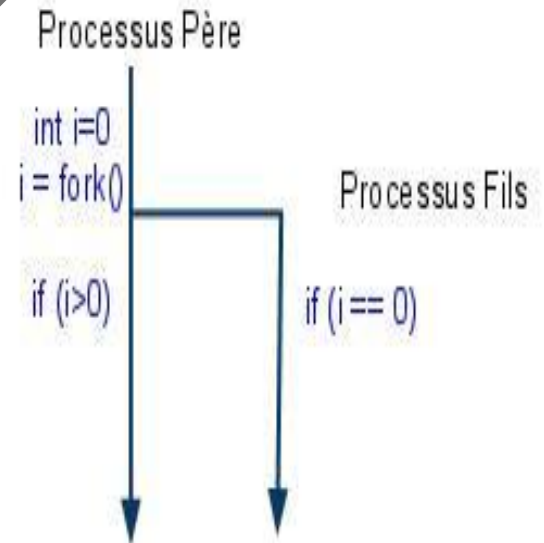
# Création de processus: la primitive *fork()* (2/2)

- L'appel de *fork()* duplique le processus.
  - L'exécution continue dans les deux processus.
  - Le père et le fils continuent leurs exécutions à l'instruction suivant l'appel de *fork()*.
- ➔ *Fork()* crée un processus identique à l'appelant (père):
  - Même code;
  - Même pile;
  - Même zone de données;
  - Même zone U;
  - Seule différence: PID et PPID.
- Les primitives utilisées avec *fork()*:
  - *getpid()* retourne le PID du processus.
  - *getppid()* retourne le PID du processus père.

## Chapitre 0: Rappel sur les notions de base des SE

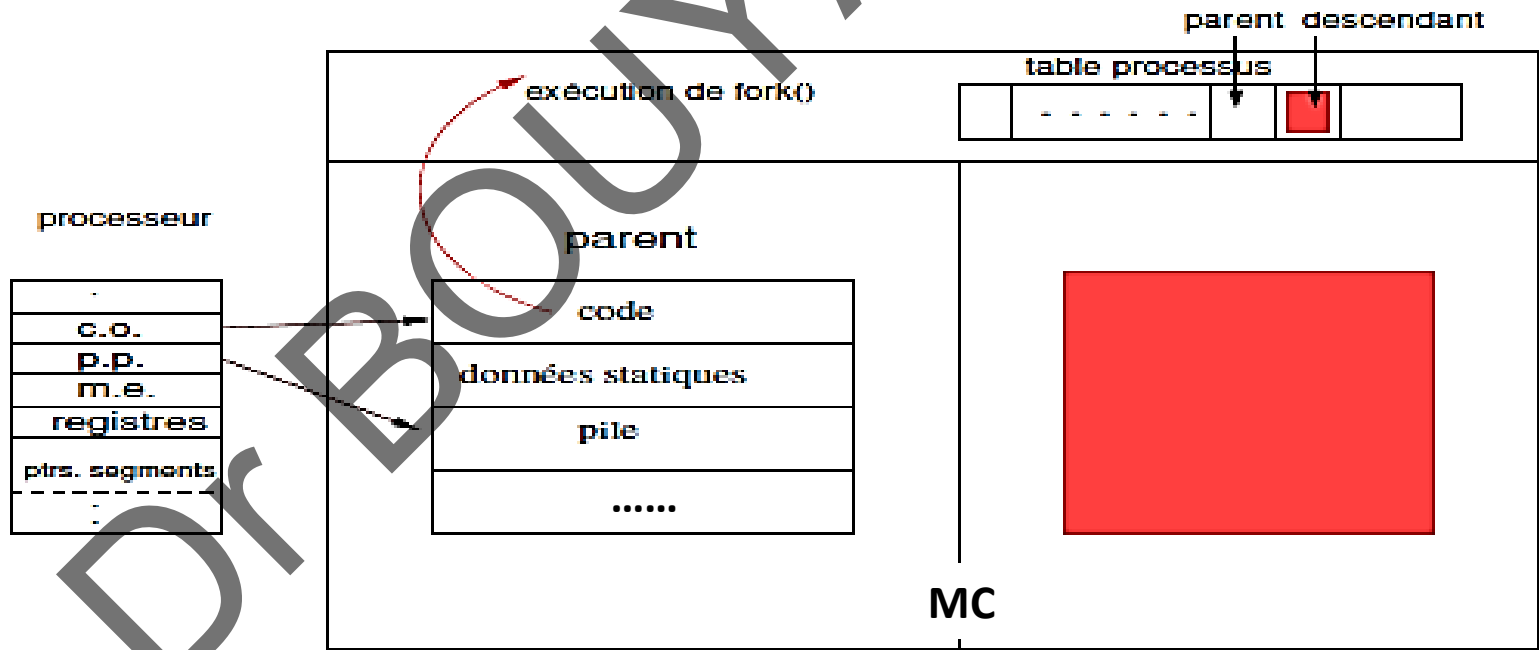
### Effet du *fork()* (1/3)

```
1  #include <unistd.h>
2  int main(void) {
3      int i=0;
4      i= fork();
5      if(i==-1) {
6          //erreur durant la création du processus fils
7      }
8      else if(i>0) {
9          //poursuite du processus Père
10         //le pid de mon fils est i
11     }
12     else {
13         // poursuite du processus Fils (i=0)
14         //le pid de mon père est: getppid()
15     }
16
17     return 1;
18 }
```



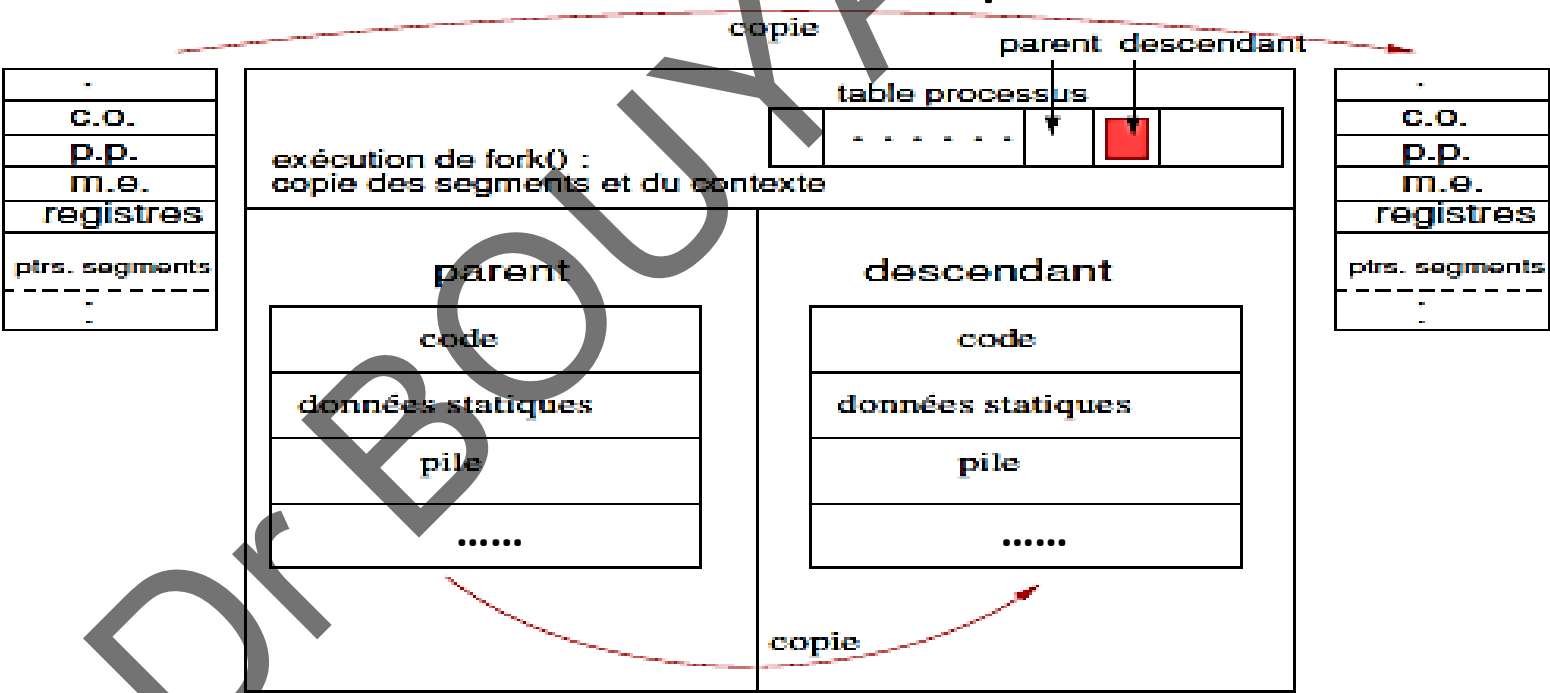
# Effet du *fork()* (2/3)

- Vérification de la disponibilité des ressources: dans la table des processus, dans l'espace mémoire...  
➔ réservation de l'espace nécessaire



# Effet du *fork()* (3/3)

- Création d'une copie du processus père avec comme différence l'identité du processus



## Chapitre 0: Rappel sur les notions de base des SE

### Exemple 1

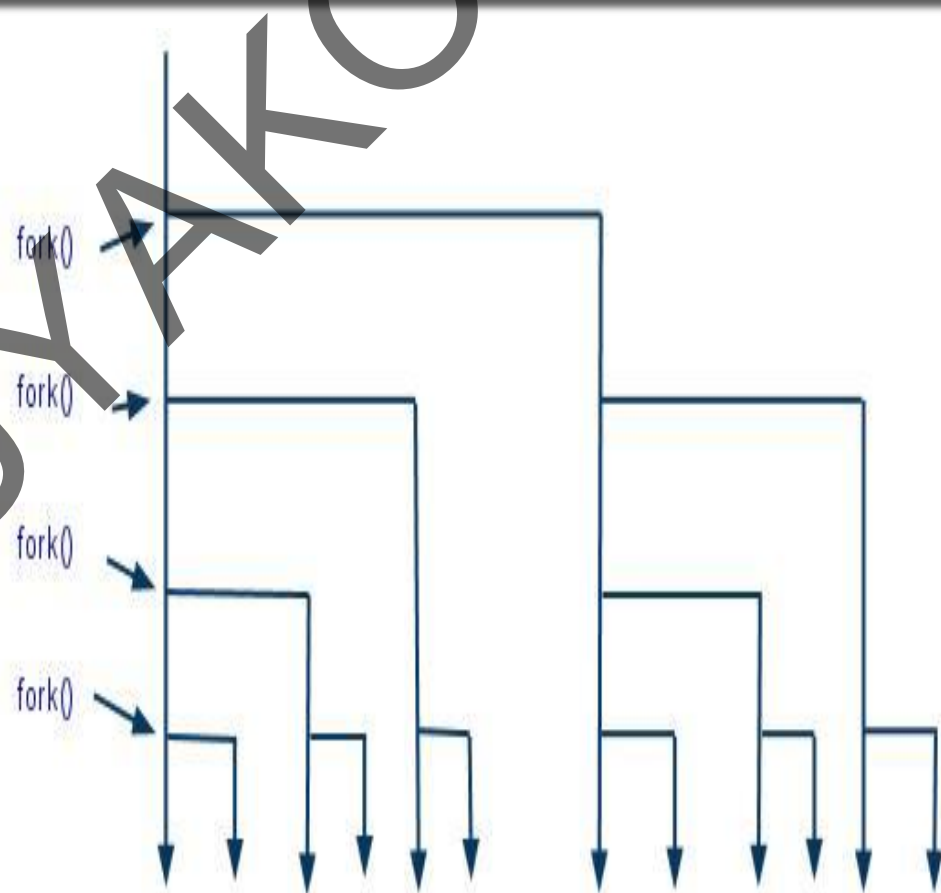
```
1 int main () {  
2     fork();  
3     fork();  
4     fork();  
5     return 1;  
6 }
```

fork()



# Exemple 2

```
1  int main() {  
2      fork();  
3      if( fork() > 0 )  
4          fork();  
5      fork();  
6      return 1;  
7  }
```





## Chapitre 0: Rappel sur les notions de base des SE

# Notion de recouvrement et primitive *exec()*

Le recouvrement consiste à charger un programme dans un processus existant. Dès que l'appel système *fork()* crée un nouveau processus, un autre programme peut démarrer à l'intérieur de ce processus.

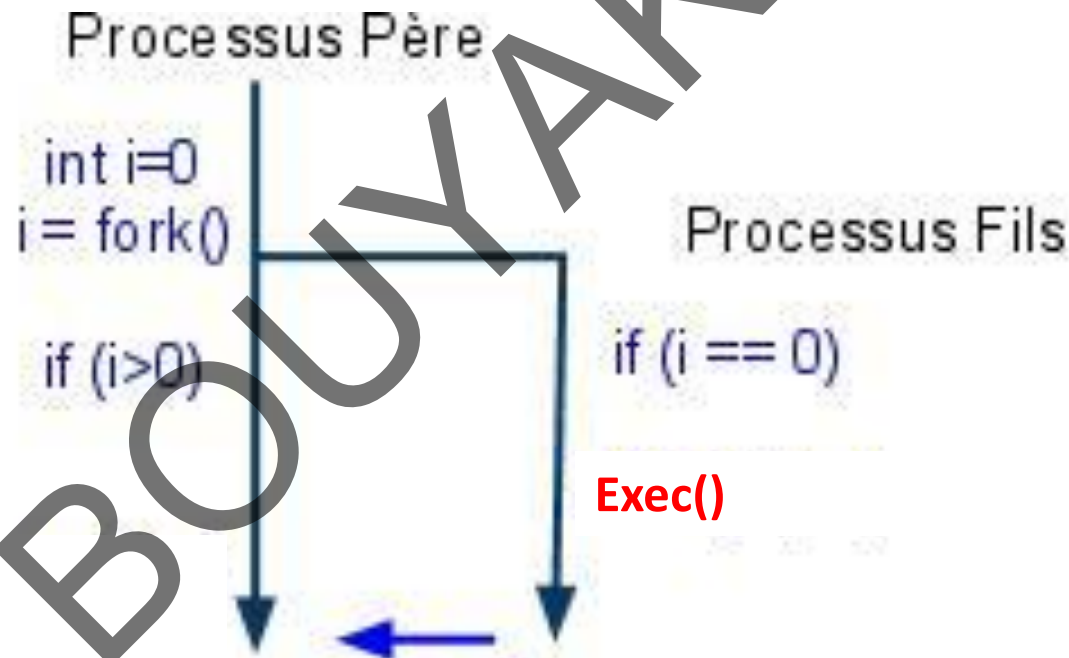
Le recouvrement modifie le segment de code et le segment de données du nouveau processus, sans modifier toutefois l'environnement du processus.

### Le recouvrement en pas à pas:

- Vérifier l'existence et l'accessibilité (droits) du fichier exécutable;
- Ecraser son propre segment de code par le nouvel exécutable;
- Passer éventuellement à ce code des paramètres d'exécution;
- Générer un nouveau segment de données;
- Vider la pile;
- Modifications du PCB dans la table des processus (espace mémoire alloué, compteur ordinal, etc).

## Chapitre 0: Rappel sur les notions de base des SE

# Héritage et recouvrement lors de la création



## Chapitre 0: Rappel sur les notions de base des SE

# Variantes de la primitive *exec()*

Le nom générique de la primitive *exec()* est suivi d'une ou deux lettres indiquant le type des arguments transmis à la fonction.

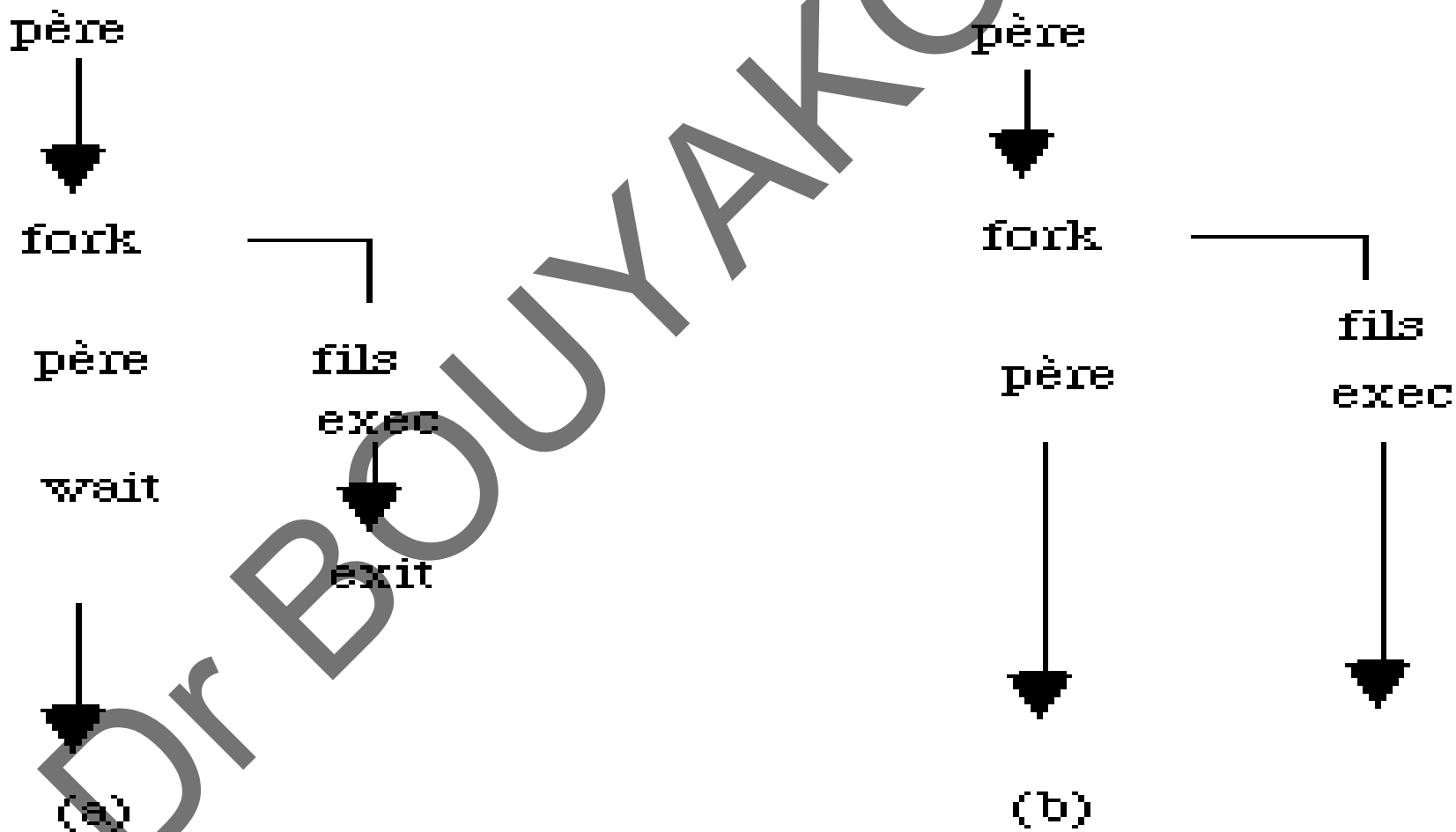
La première lettre, " l " ou " v " désigne la façon dont la commande et ses arguments sont transmis à la fonction.

- **La lettre " l "**, utilisée lorsque le nombre d'arguments est connu, indique que les arguments sont listés les uns derrière les autres lors de l'appel de la fonction. Le dernier argument doit être "NULL" pour marquer la fin de la liste.
- **La lettre " v "** indique que la primitive reçoit un tableau de pointeurs. Chaque pointeur pointe sur une chaîne de caractères contenant un des arguments de la commande.
  - ➔ Ce tableau doit se terminer par un pointeur "NULL".

# Exemple de la primitive *execl()*

- `int execl(char const *path, char const *arg0, ...);`
  - path: indique le chemin de la commande à exécuter.
  - const: donne le nom de la commande
  - arg0...argN: les arguments de la commande
- Exemple:  
`execl ( "/bin/ls", "ls", "-l", "/usr", NULL ) ➔ ls -l /usr`

# La primitive *wait()*



## La primitive *exit()*

La primitive *exit()* permet de terminer le processus qui l'appelle.

- void *exit*(int status)
- status permet d'indiquer au processus père qu'une erreur s'est produite.
- Si status=0 alors pas d'erreur.

# La primitive *sleep()*

- Le processus qui appelle *sleep()* est bloqué pendant le nombre de secondes spécifié.  

sleep(int secondes)
- Différence avec *wait()*:
  - *wait()* bloque jusqu'à la fin du fils,
  - *sleep()* bloque un temps spécifié.

## Chapitre 0: Rappel sur les notions de base des SE

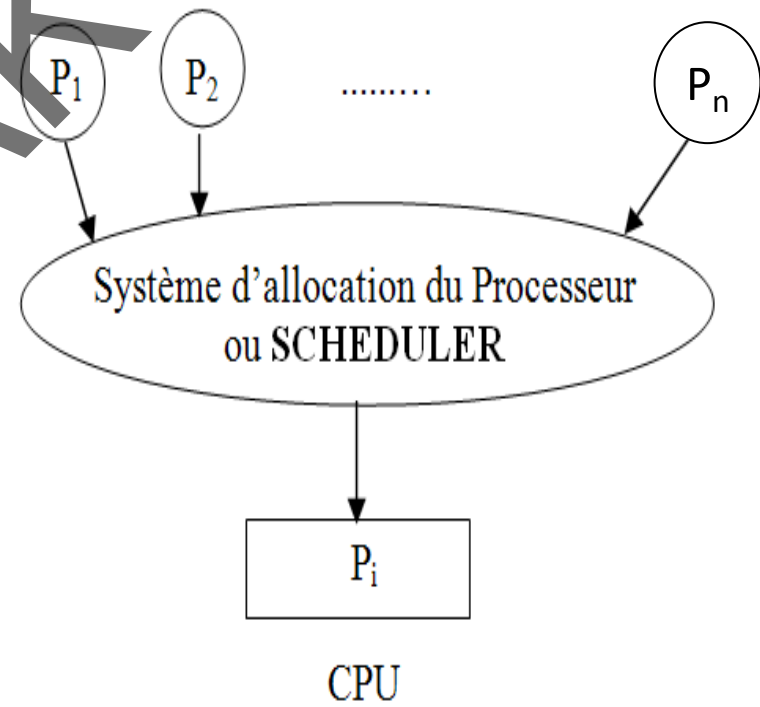
Présentation du module  
Les SE: Fonctionnalités et architectures  
Processus et multiprogrammation  
Cas des processus UNIX  
Manipulation de processus en langage C  
Processus et *scheduling*

# Processus et *scheduling*



# Scheduling de processus

- On appelle ordonnancement ou "**Scheduling**" du processeur l'organisation liée à l'allocation du processeur central aux programmes.
- On appelle ordonnanceur ou "**Scheduler**" la partie du SE qui s'occupe de cette organisation et répartit le temps processeur entre ces programmes.



## *Scheduling* avec et sans préemption

- Une politique de *scheduling* est dite **non préemptive** si, une fois le processeur central est alloué à un programme, celui-ci le gardera jusqu'à la fin de son exécution, ou jusqu'à se bloquer sur une ressource non disponible ou en attente de la satisfaction d'une demande de ressource.

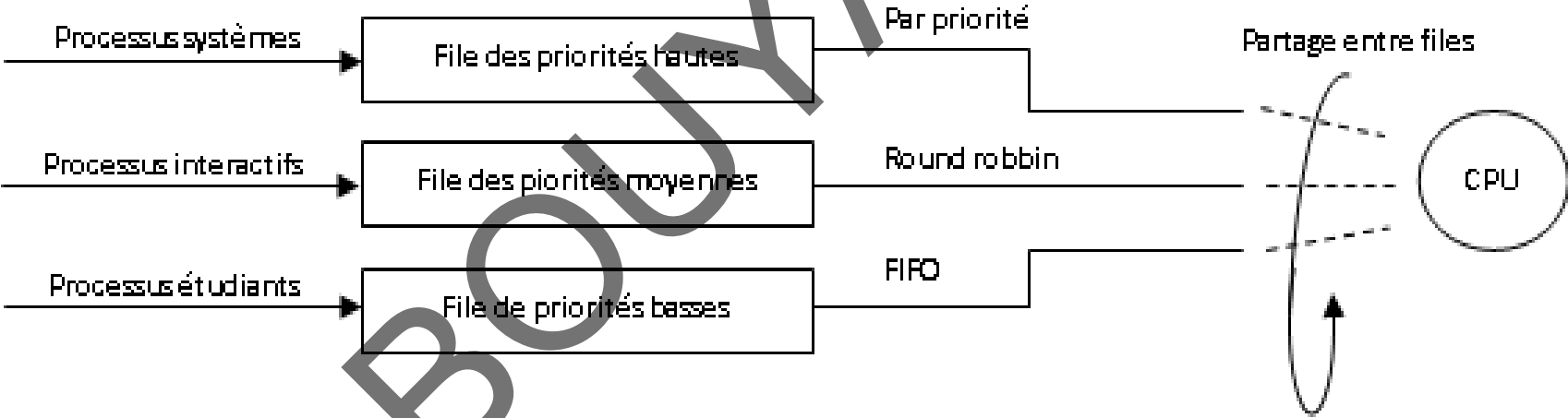
## *Scheduling* et priorité

- **Priorité statique:** Une fois qu'une priorité est allouée à un programme, celle-ci ne changera pas jusqu'à la fin de son exécution.
- **Priorité dynamique:** La priorité affectée à un programme change en fonction de l'environnement d'exécution du programme. Cet environnement est donné en général par la charge du système (nombre de processus en cours d'exécution), l'ancienneté (à chaque fois que le programme prend plus de temps, on diminue sa priorité).

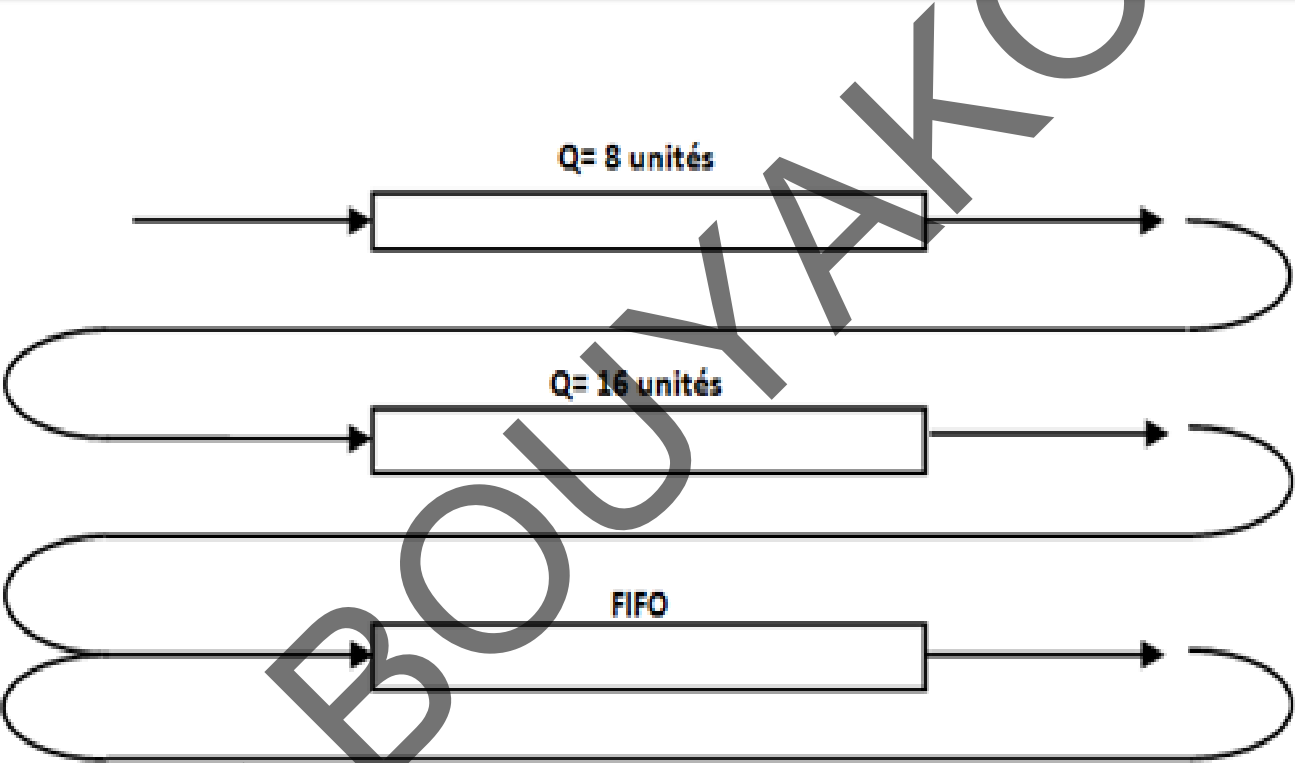
# Les politiques de *scheduling* des processus

- FIFO
- SJF
- RR (tourniquet)
- Politique avec priorité
- Politique à plusieurs niveaux de queue (*Multilevel queue*)
- Politique à plusieurs niveaux de queue dépendants (*Multilevel feed back queue*)

# Politique à plusieurs niveaux de queue



# Politique à plusieurs niveaux de queue dépendants



## Questions

