Université des Sciences et de la Technologie
Houari Boumediene
USTHB

Système d'exploitation des ordinateurs

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

Septembre 2014

Présentation du module

- Intitulé du module:
  - Systèmes d'exploitation

- Objectifs du module:
  - Approfondir les différents concepts acquis en licence pour la manipulation des processus et la gestion des fichiers;
  - La programmation système.

Contenu du cours (1/4)

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 (2/4)

Chapitre 1: Notion de parallélisme et synchronisation des processus

- Processus séquentiels et concurrents
- Problème de l'exclusion mutuelle
- Synchronisation de processus (mécanismes d'EM)
  - Solutions logicielles;
  - Solution matérielle;
  - Sémaphores;
  - Exemples classiques de synchronisation de processus;
  - Aperçu sur les événements mémorisés.

Chapitre 2: Outils de synchronisation évolués

- Les moniteurs;
- La variante de *Kessel;*
- Les moniteurs avec priorité.

Contenu du cours (3/4)

Chapitre 3: Système de gestion de fichier (SGF)

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

Chapitre 4: Communication inter-processus

- Partage de variables (modèle de producteur/consommateur et lecteurs/rédacteurs);
- Boite 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 (4/4)

Chapitre 5: L'interblocage

- Modèles et représentation
- Prévention
- Détection/ Guérison

Supports de cours

- Les cours sont téléchargeables sur mon site:

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

- Sur facebook:
  - Groupe Wiki-Sys

Les SE: fonctionnalités et architectures

Présenté par Dr BOUYAKOUB
bouyakoub.f.m@gmail.com

Définition d'un système d'exploitation

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

Les fonctions d'un SE

1. Gestion du processeur
2. Gestion de la mémoire vive
3. Gestion des entrées/sorties
4. Gestion de l'exécution des applications
5. Gestion des droits
6. Gestion des fichiers
7. Gestion des informations

Architecture d'un système

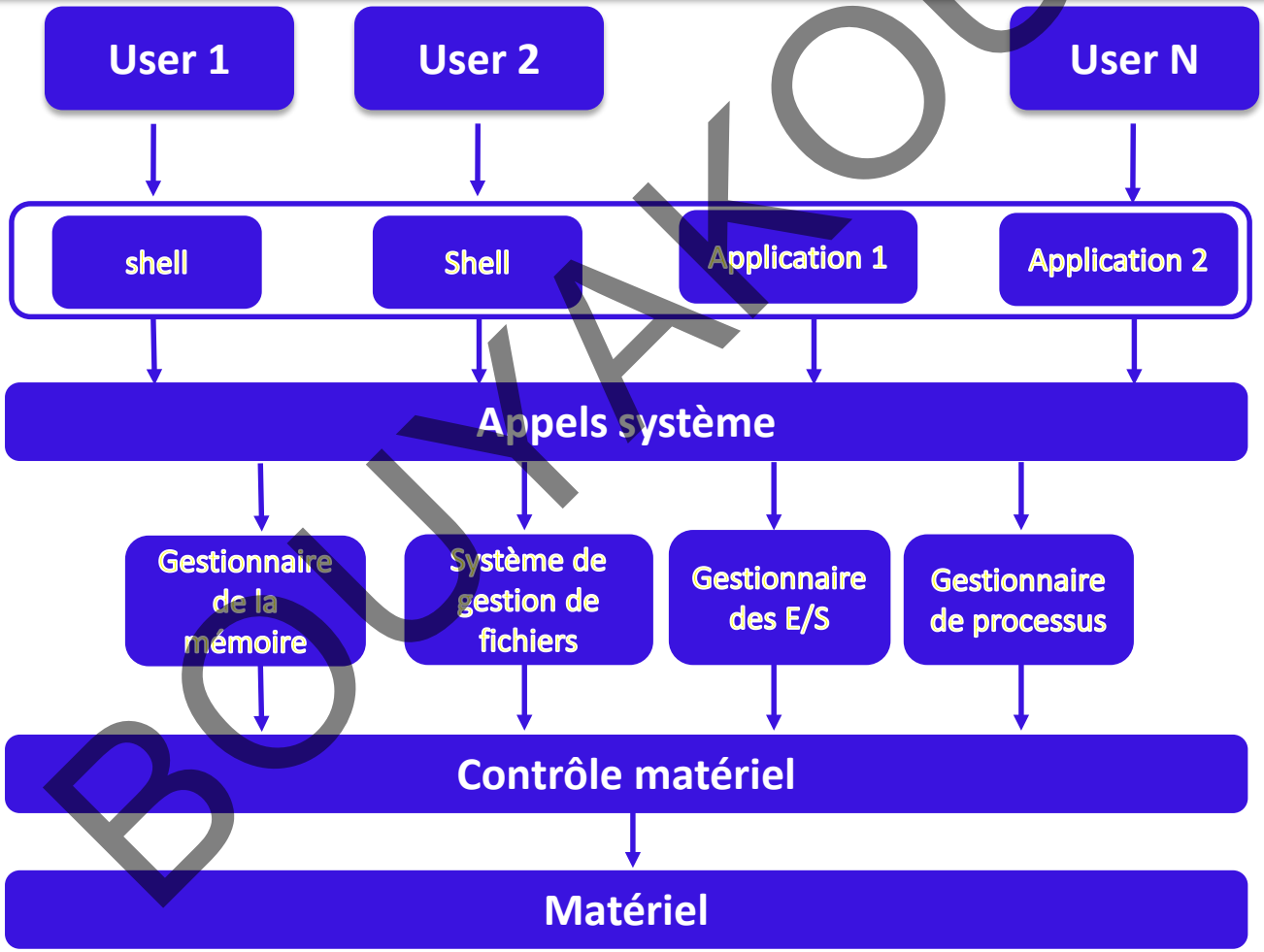
Divisée en 3 couches distinctes

- 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



Processus et multiprogrammation

Présenté par Dr BOUYAKOUB
bouyakoub.f.m@gmail.com

Notion de processus

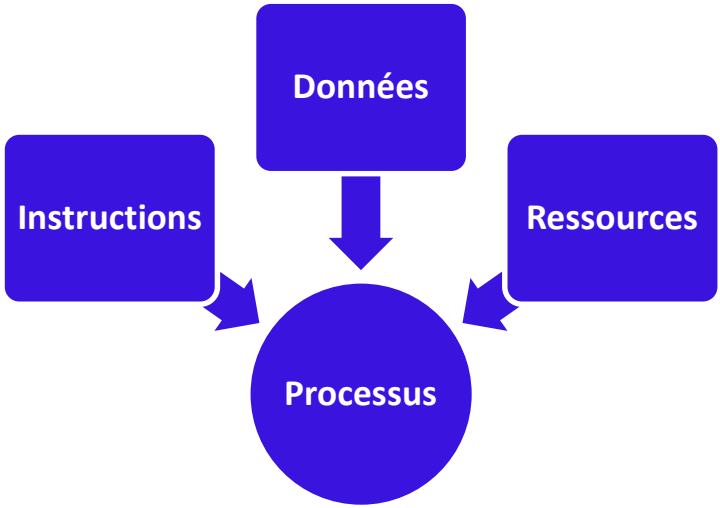
Deux aspects sont liés à la notion de processus:

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.

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

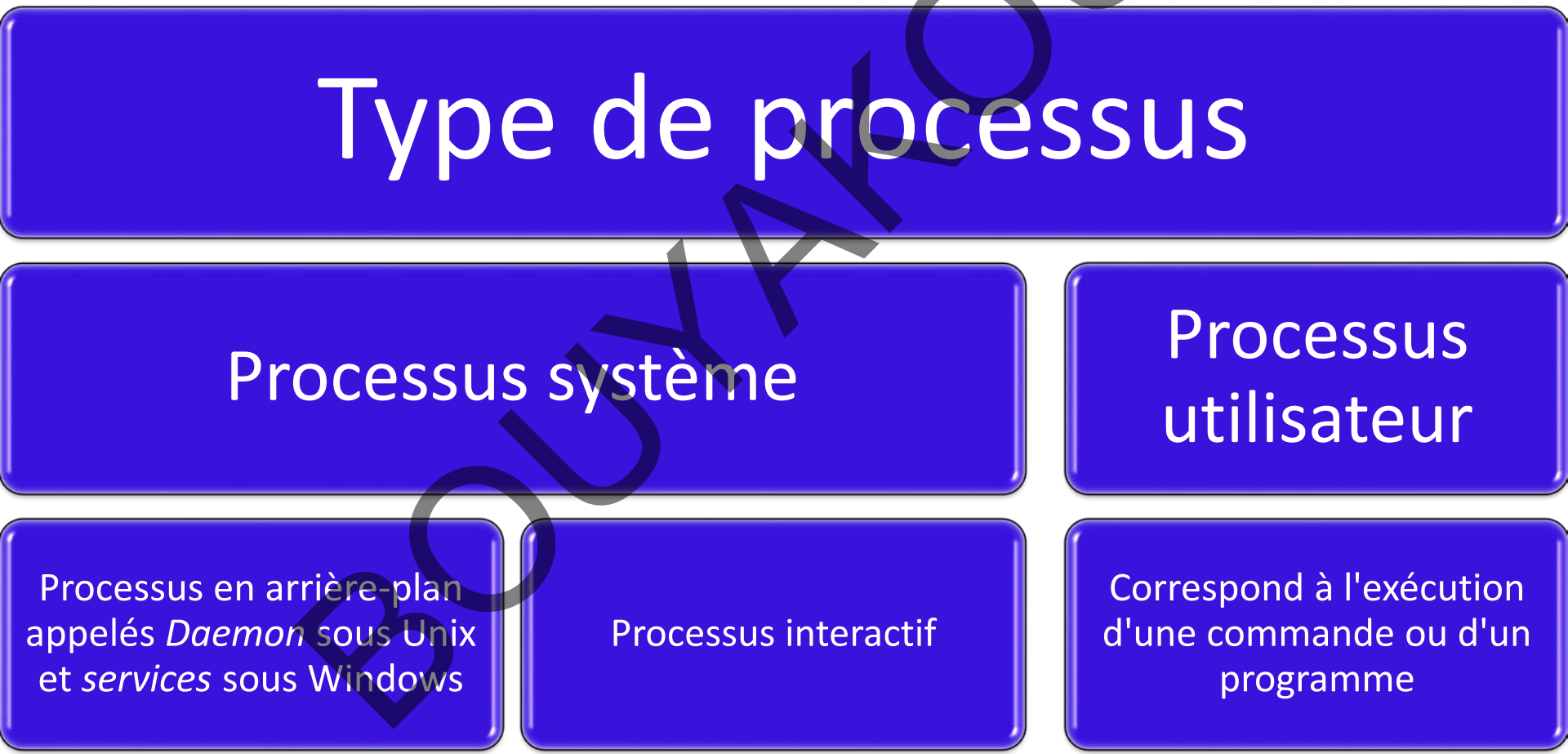
Caractéristiques statiques

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

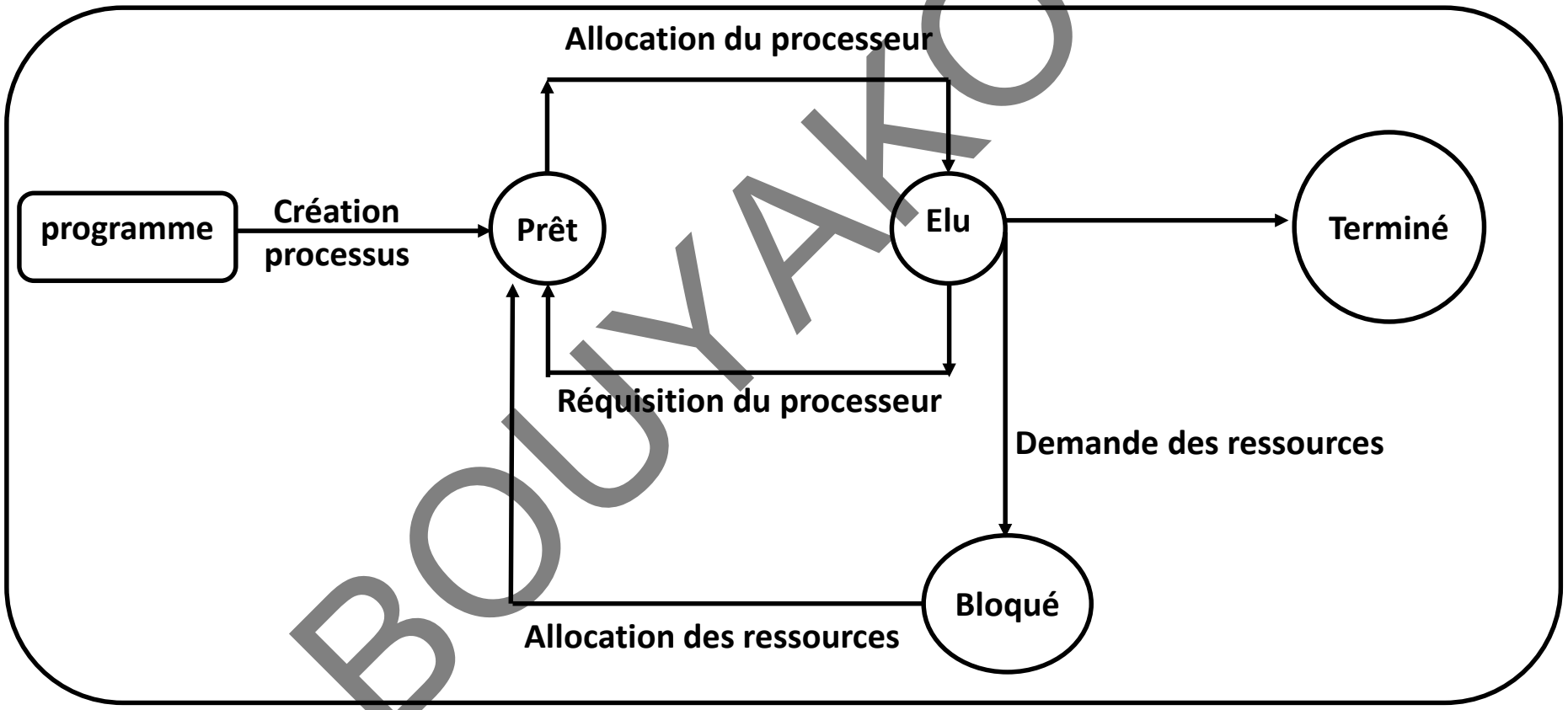
Caractéristiques dynamiques

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

Type de processus



Etats de base d'un processus

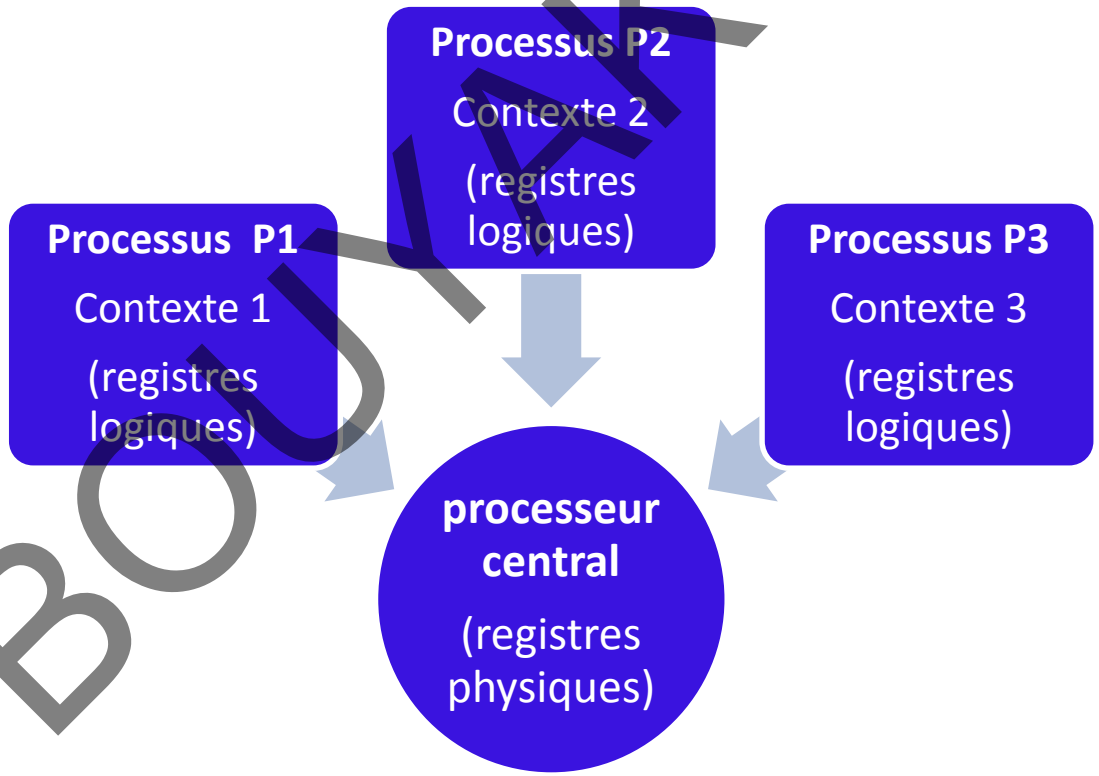


Contexte d'un processus

- 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 processus + 1 processeur = commutation de contexte



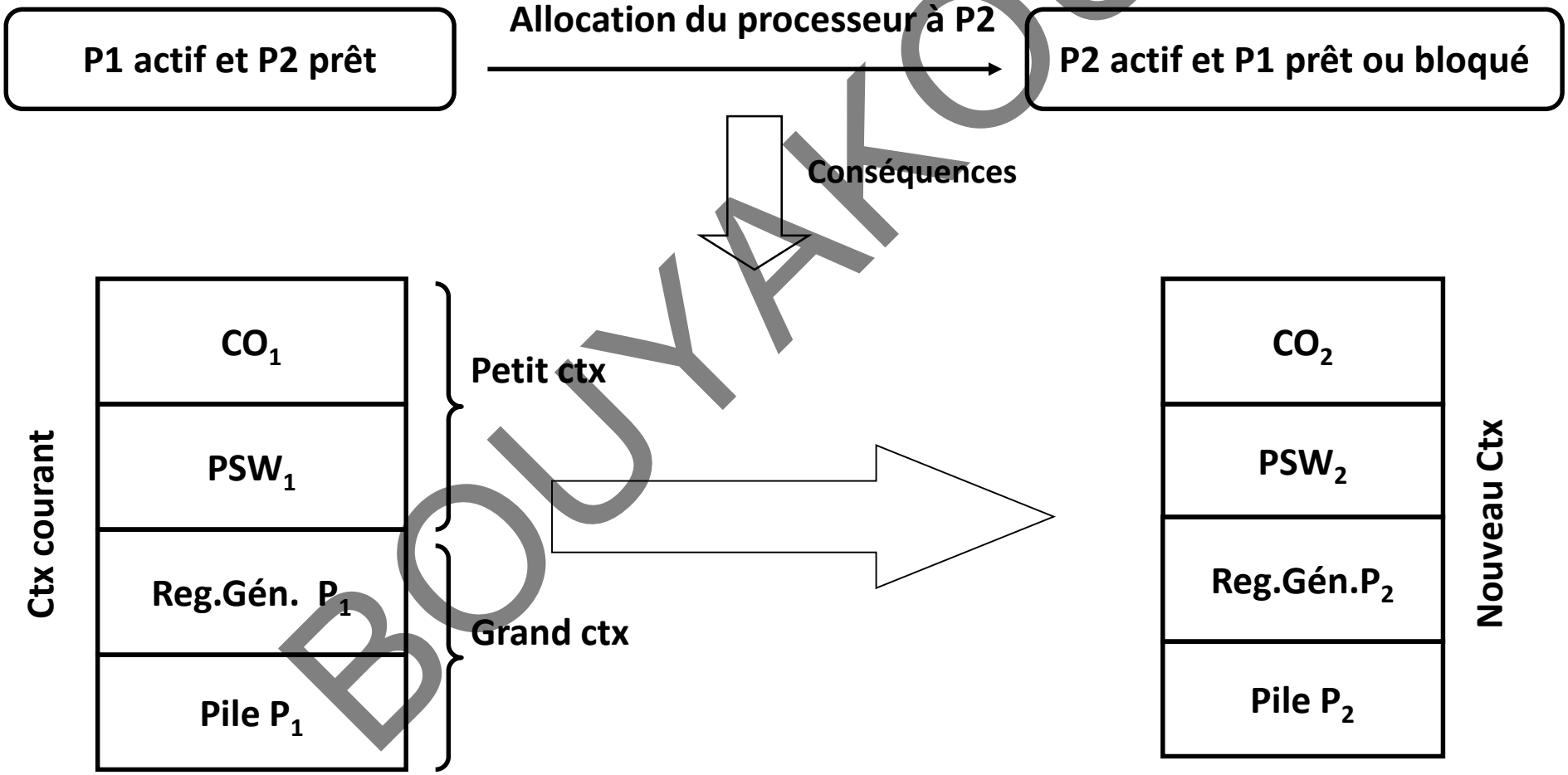
Mécanismes de commutation de contexte (1/2)

- 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.

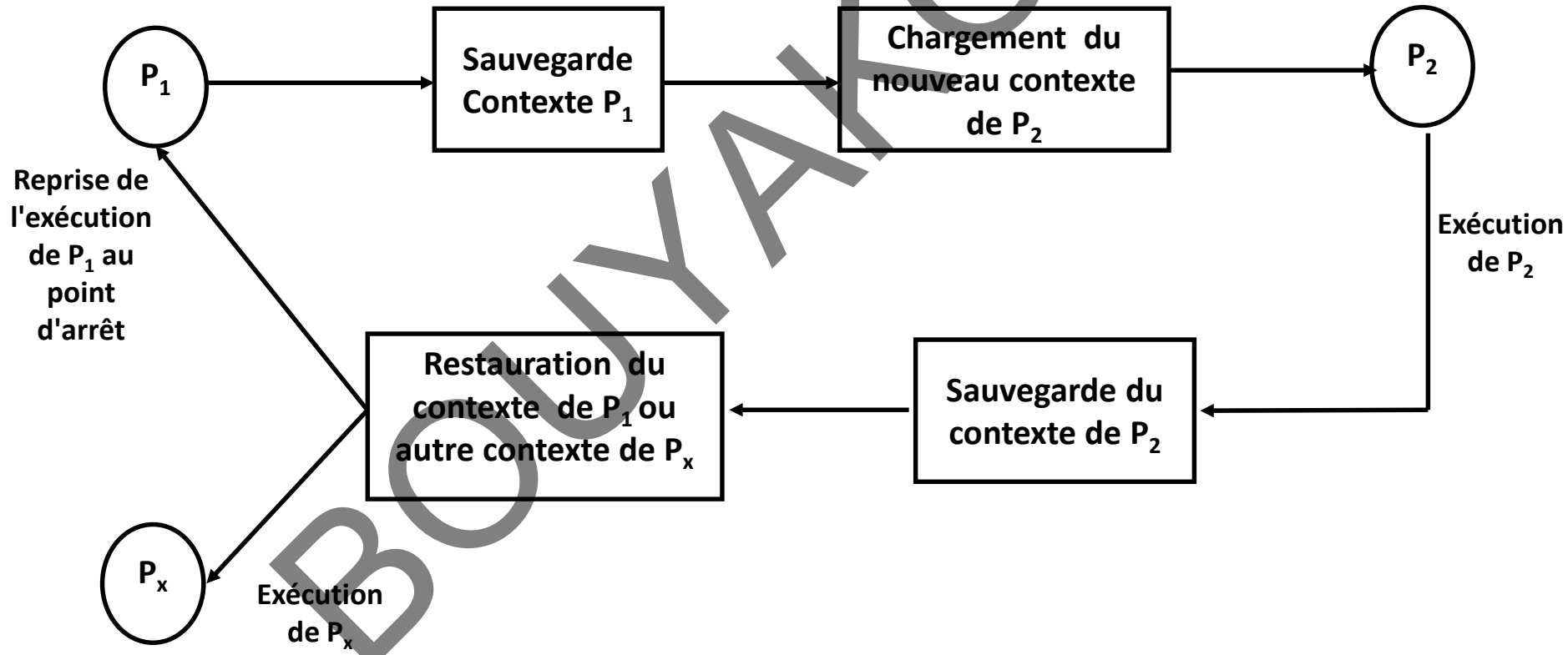
  ➔ **Commutation du contexte**.

- 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)



La restauration du contexte



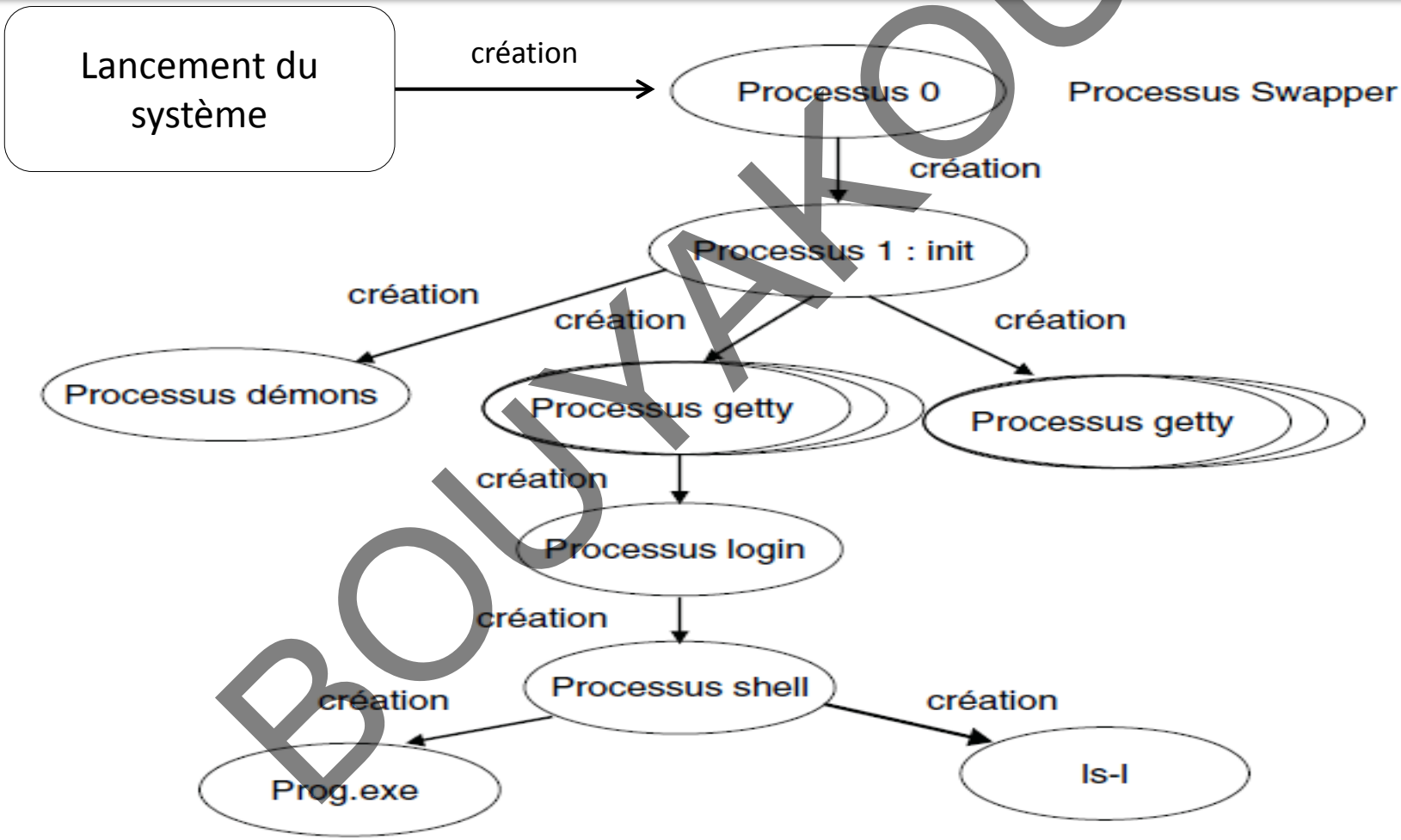
Opérations sur les processus

- La création;
- La suspension;
- La destruction.

Etude de cas: les processus sous Unix

Présenté par Dr BOUYAKOUB
bouyakoub.f.m@gmail.com

Arborescence de processus sous Unix



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…

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 limité. Par exemple, un processus dans cet état peut aller lire dans la mémoire d'un autre.

Manipulation de processus en langage C

Présenté par Dr BOUYAKOUB
bouyakoub.f.m@gmail.com

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*.

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().

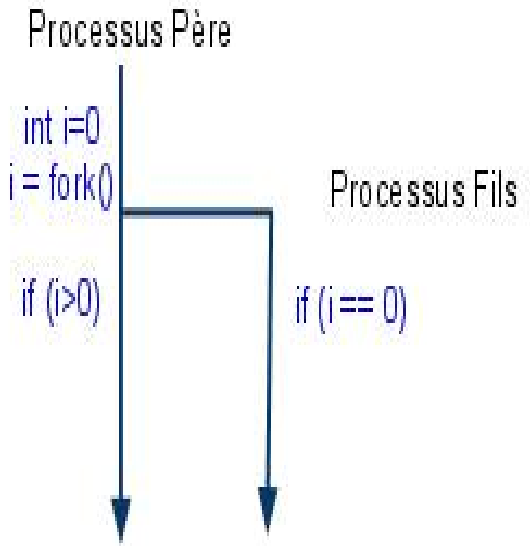
➔ *Fork()* crée un processus identique à l'appelant (père):
  - Même code;
  - Même pile;
  - Même zone de données;
  - Même zone U;
  - Seule différence: PID et PPID.

- Les primitives utilisées avec fork():
  - *getpid()* retourne le PID du processus.
  - *getppid()* retourne le PID du processus père.

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

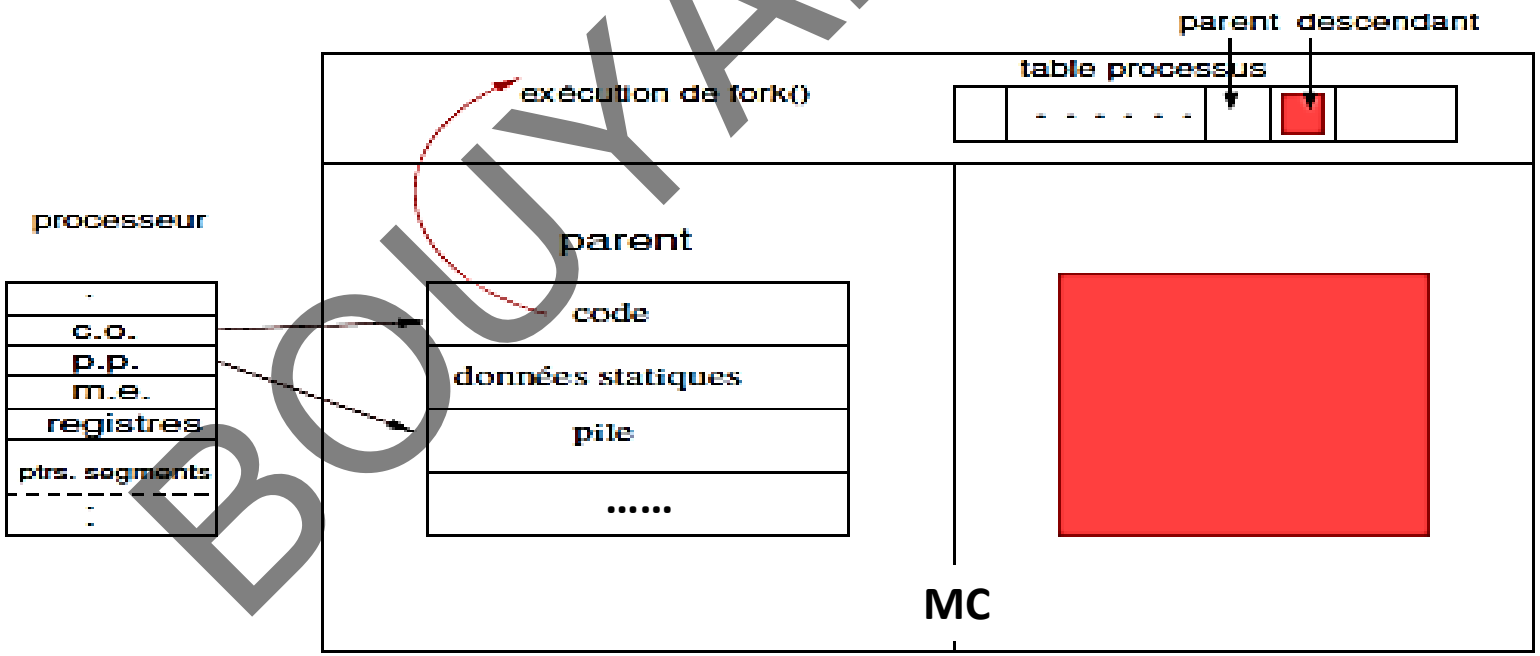
```c
#include <unistd.h>
int main(void) {
    int i=0;
    i= fork();
    if(i==-1) {
        //erreur durant la création du processus fils
    }
    else if(i>0) {
    //poursuite du processus Père
        //le pid de mon fils est i
    }
    else {
    // poursuite du processus Fils (i=0)
    //le pid de mon père est: getpppid()
    }

    return 1;
}
```



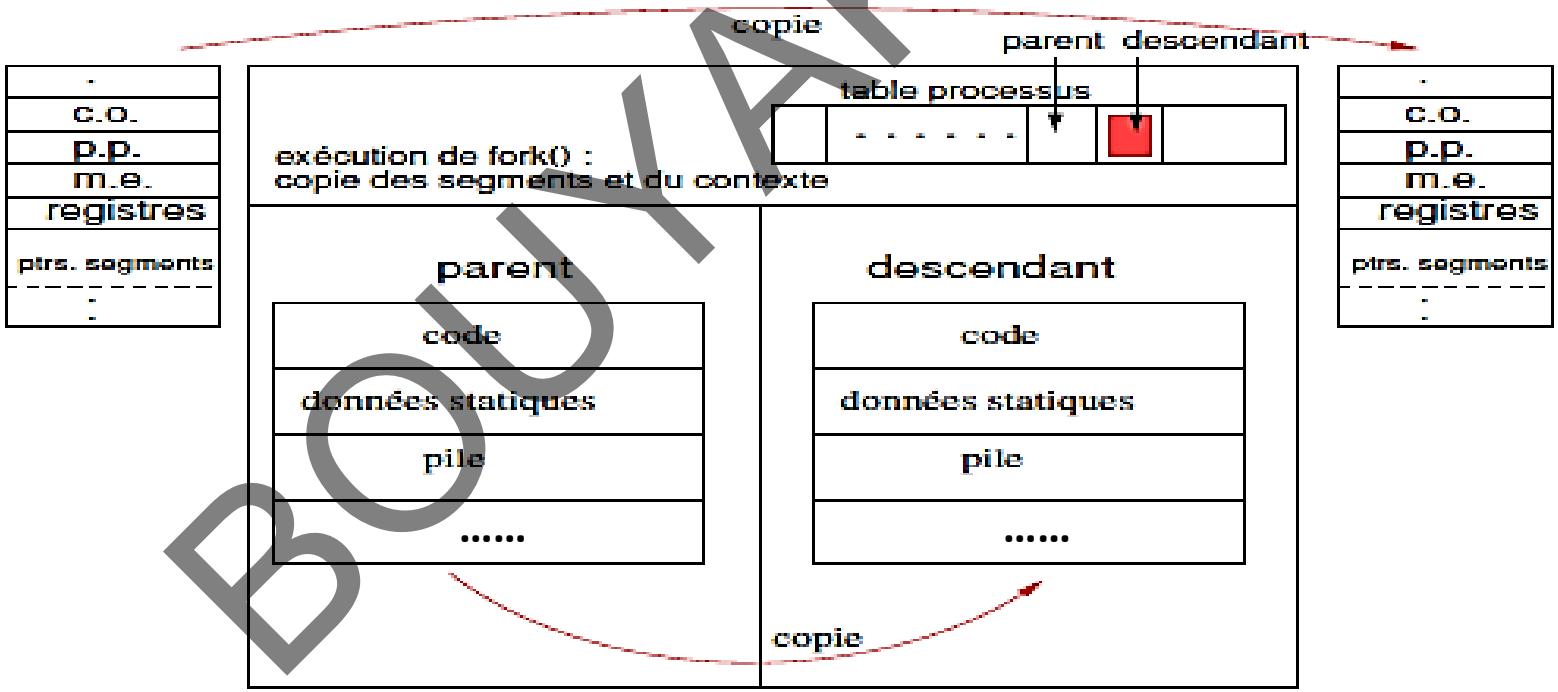
Effet du *fork()* (2/3)

- 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

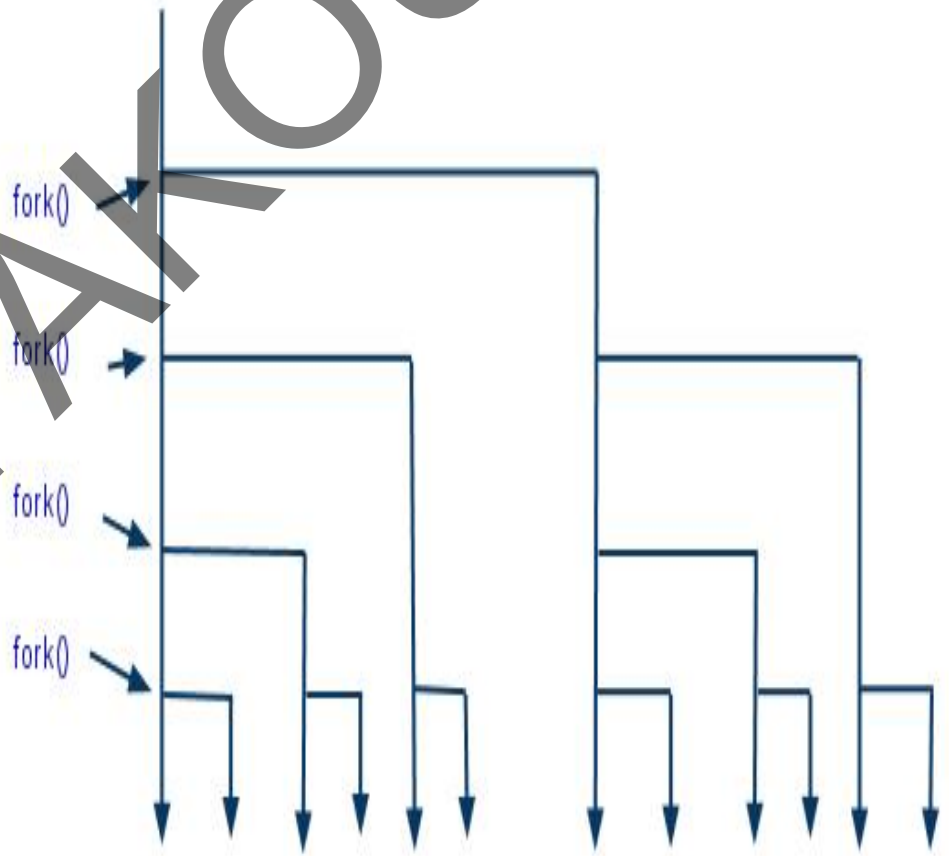


Exemple 1

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

Exemple 2

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



Notion de recouvrement et primitive *exec()*

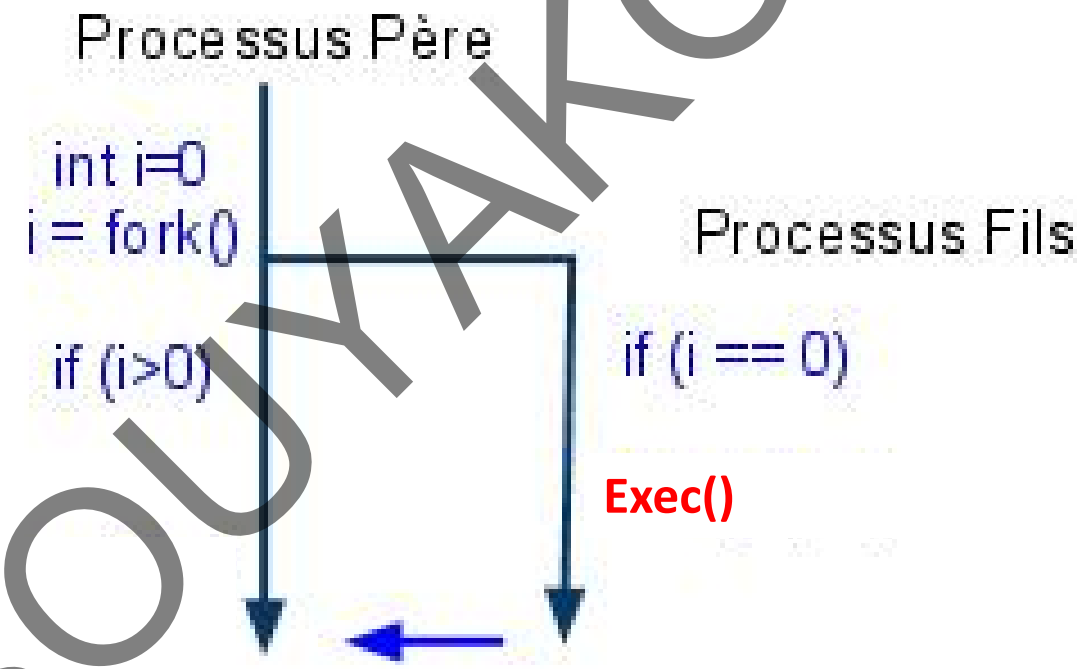
Le recouvrement consiste à charger un programme dans un processus existant. Dès que l'appel système *fork()* crée un nouveau processus, un autre programme peut démarrer à l'intérieur de ce processus.

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).

Héritage et recouvrement lors de la création

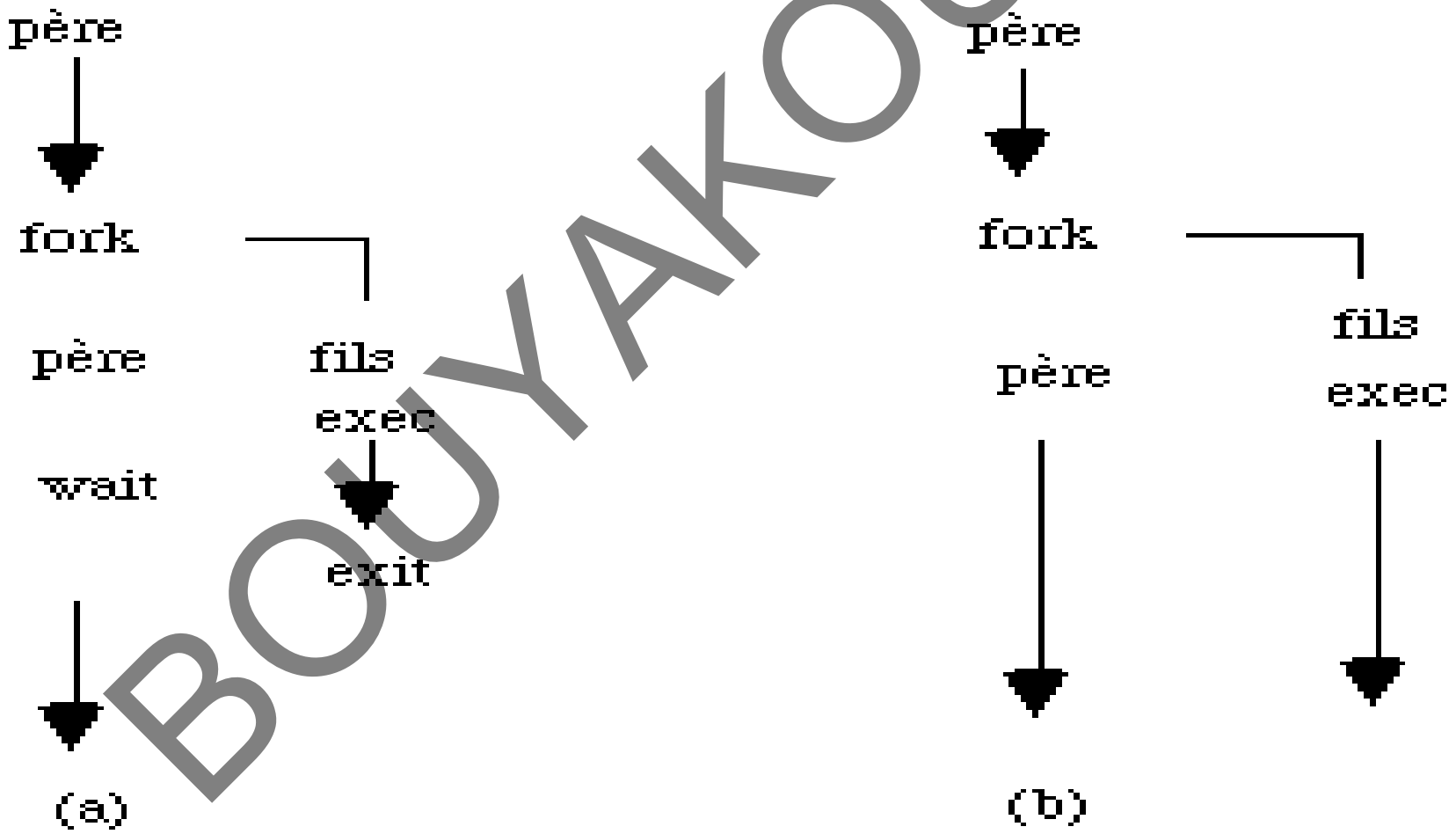


Exemple de la primitive *execl()*

- 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)

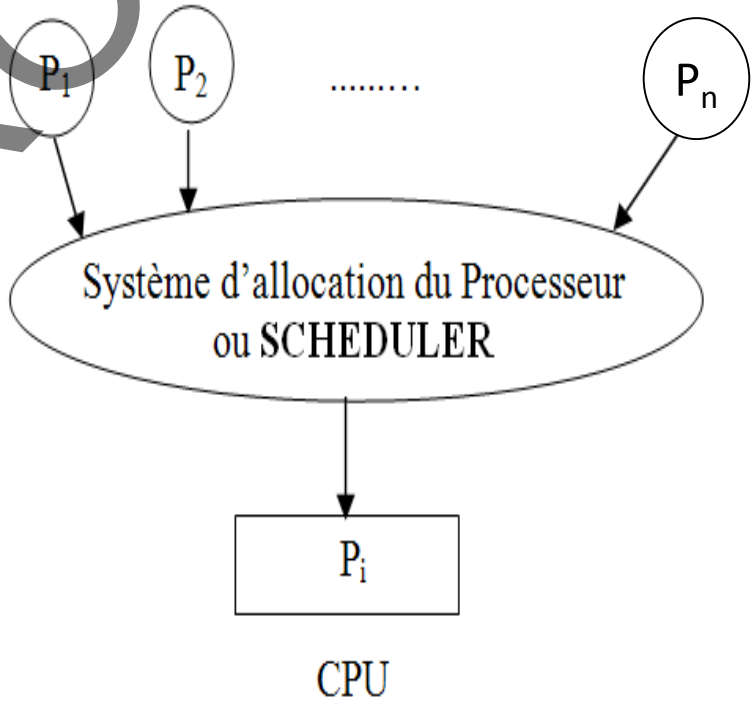
- Différence avec wait():
  - *wait()* bloque jusqu'à la fin du fils,
  - *sleep()* bloque un temps spécifié.

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.

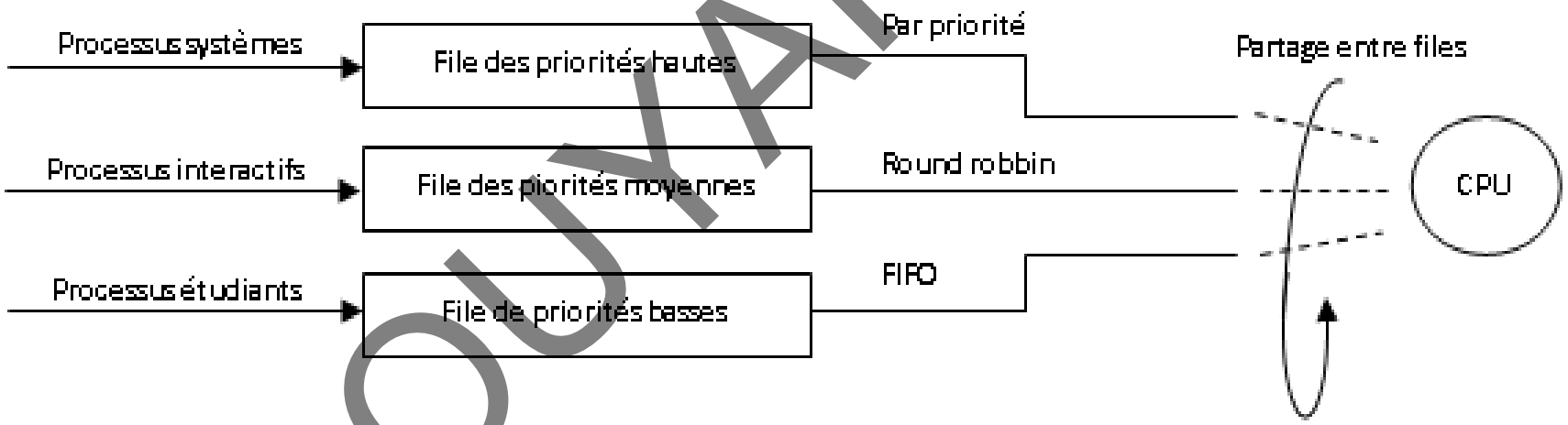
Scheduling et priorité

- **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).

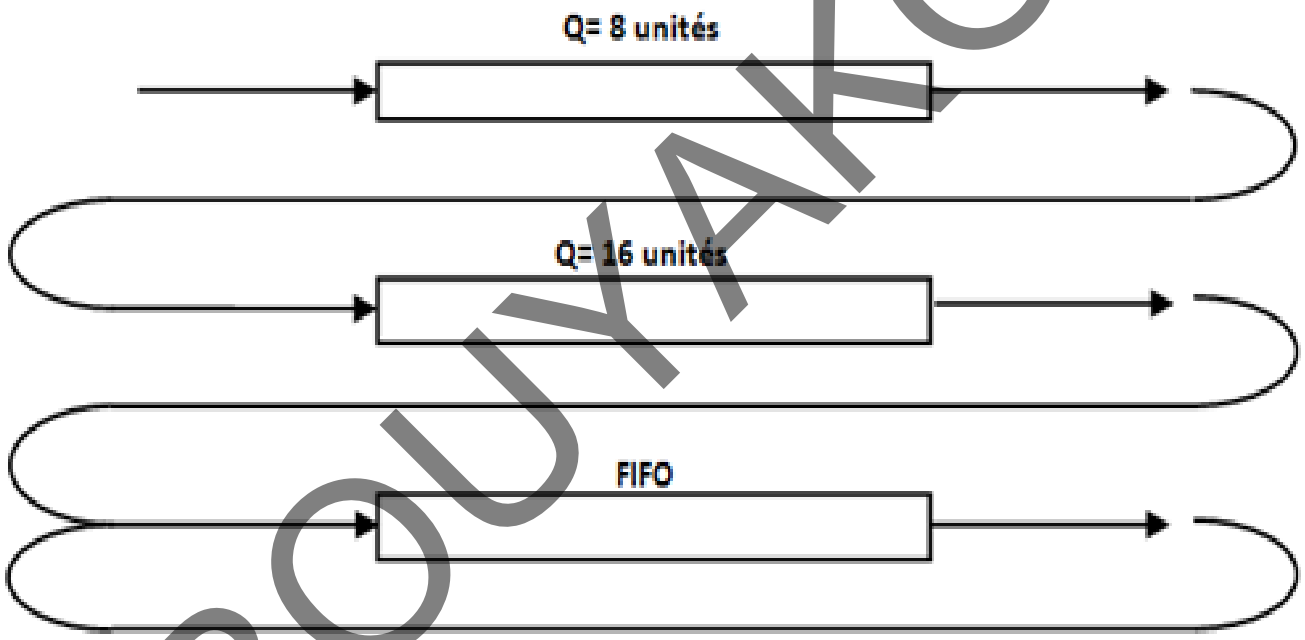
Les politiques de *scheduling* des processus

- 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