

# Processus

Thomas Lavergne  
lavergne@lisn.upsaclay.fr

## Programme

Un **programme** est une suite d'instructions

## Programme

Un **programme** est une suite d'instructions

## Processeur

Un **processeur** est un automate de traitement

- il peut exécuter un programme
- il modifie son état en fonction des instructions

## Programme

Un **programme** est une suite d'instructions

## Processeur

Un **processeur** est un automate de traitement

- il peut exécuter un programme
- il modifie son état en fonction des instructions

## Processus

Un **processus** est un **programme** exécuté par un **processeur**



## Instructions

Copie du code en mémoire (partie **exécutable** du programme)

## Instructions

Copie du code en mémoire (partie **exécutable** du programme)

## Environnement

- Variables d'environnement héritées : PATH...
- Descripteurs de fichiers (cf. cours 8)
- ...

## Instructions

Copie du code en mémoire (partie **exécutable** du programme)

## Environnement

- Variables d'environnement héritées : PATH...
- Descripteurs de fichiers (cf. cours 8)
- ...

## Mémoire

- Pile : variable locales, contexte...
- Tas : allocation dynamique...



## Création et suppression de processus

Programme → Processus (Munir le programme des **informations** nécessaires pour son **exécution**)

## Création et suppression de processus

Programme → Processus (Munir le programme des **informations** nécessaires pour son **exécution**)

## Suspension et reprise

Multiprogrammation et temps partagé

→ Interrompre et reprendre les processus

## Création et suppression de processus

Programme → Processus (Munir le programme des **informations** nécessaires pour son **exécution**)

## Suspension et reprise

Multiprogrammation et temps partagé

→ Interrompre et reprendre les processus

## Communication et synchronisation (cf. cours 4)

- Partage de données entre plusieurs processus
- **Consistence** de la mémoire

### PCB Process Control Block

**Structure de données** contenant les informations relative à un **processus** utilisée par l'OS pour le gérer

## PCB Process Control Block

**Structure de données** contenant les informations relative à un **processus** utilisée par l'OS pour le gérer

Contenu :

## PCB Process Control Block

**Structure de données** contenant les informations relative à un **processus** utilisée par l'OS pour le gérer

### Contenu :

- Identification (pid et ppid)
- Statut et privilèges

## PCB Process Control Block

**Structure de données** contenant les informations relative à un **processus** utilisée par l'OS pour le gérer

### Contenu :

- Identification (pid et ppid)
- Statut et privilèges
- Informations sur la mémoire (cf. cours 5 à 7)
- Informations d'ordonnancement (cf. cours 3)
- Réservations de périphériques

## PCB Process Control Block

**Structure de données** contenant les informations relative à un **processus** utilisée par l'OS pour le gérer

### Contenu :

- Identification (pid et ppid)
- Statut et privilèges
- Informations sur la mémoire (cf. cours 5 à 7)
- Informations d'ordonnancement (cf. cours 3)
- Réservations de périphériques

**Inaccessible au processus !**

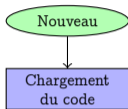


# Cycle de vie

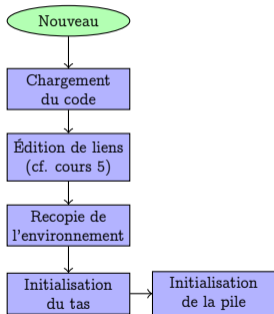


Nouveau

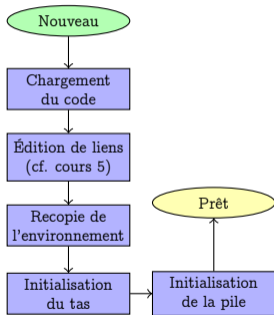
# Cycle de vie du processus



# Cycle de vie du processus

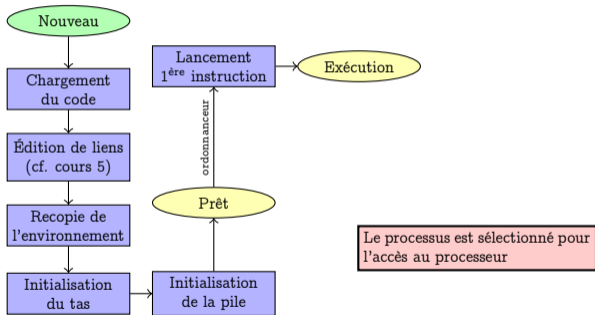


# Cycle de vie du processus

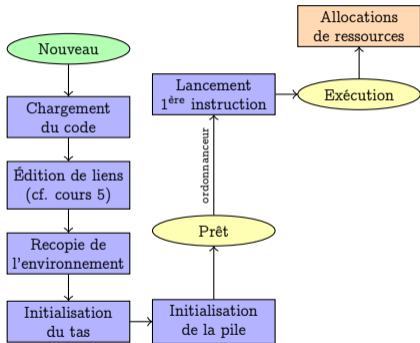


Le processus prêt est mis dans la file d'attente

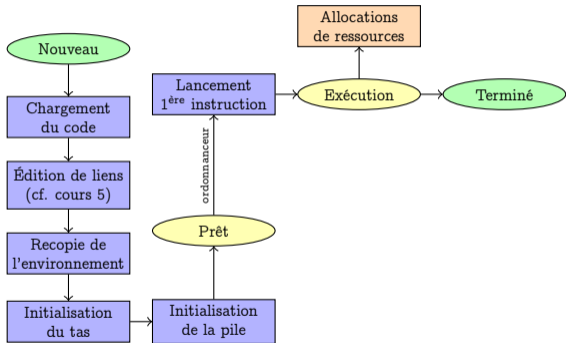
# Cycle de vie du processus



# Cycle de vie du processus

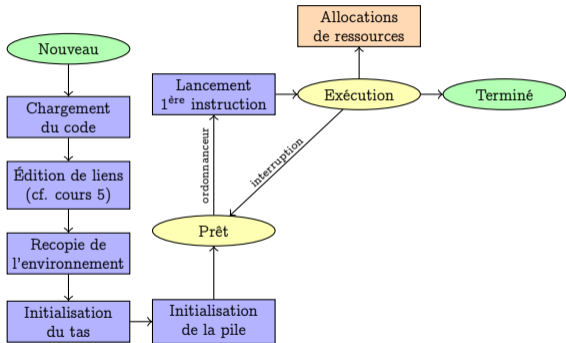


# Cycle de vie du processus

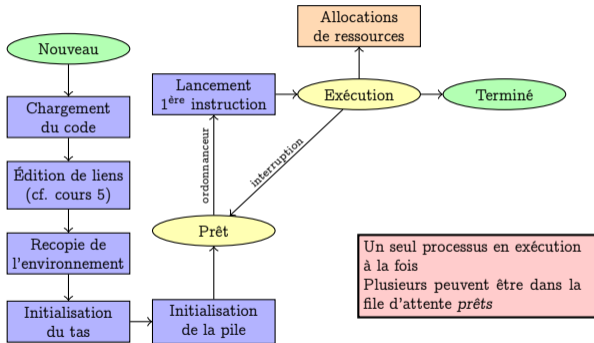




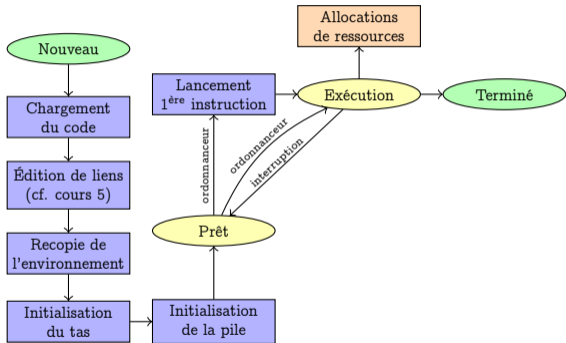
# Cycle de vie du processus



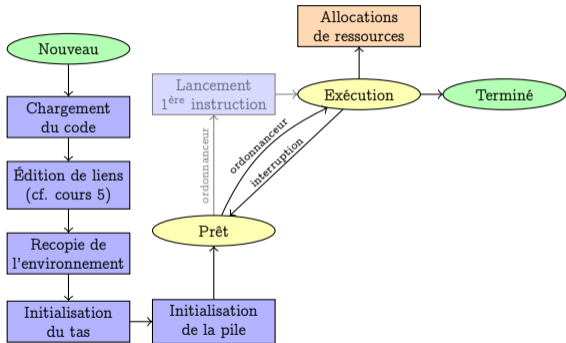
# Cycle de vie du processus



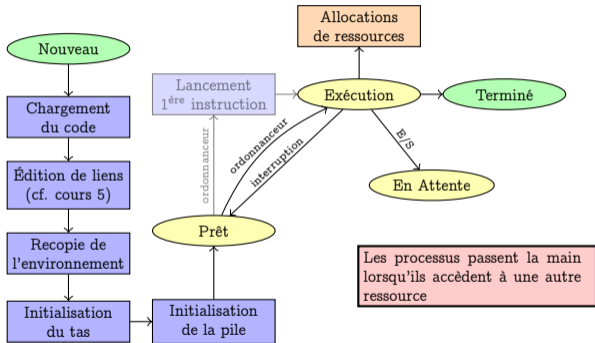
# Cycle de vie du processus



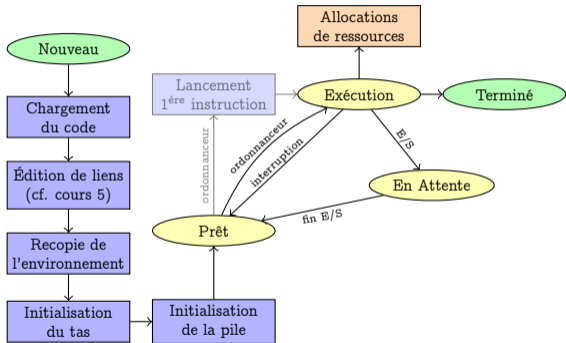
# Cycle de vie du processus



# Cycle de vie du processus

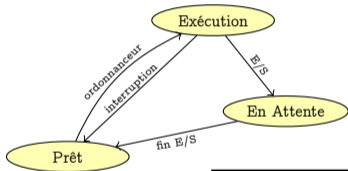


# Cycle de vie du processus





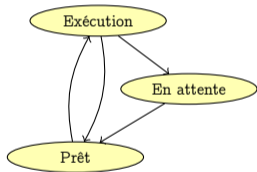
# Cycle de vie du processus

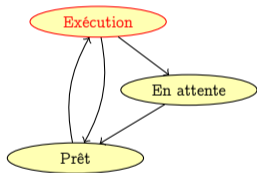


0 – 1 processus en exécution  
0 – N processus prêts  
0 – N processus en attente

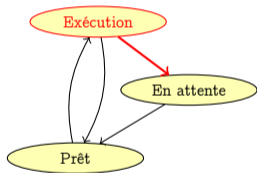


# Cycle de vie du processus



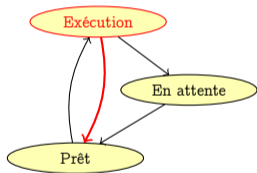


## Suspension de l'exécution



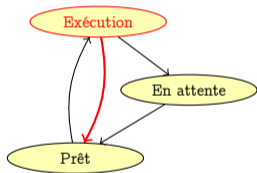
## Suspension de l'exécution

- Demande d'E/S  
→ File en attente



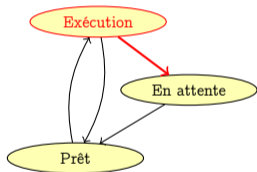
## Suspension de l'exécution

- Demande d'E/S  
→ File en attente
- Temps écoulé  
→ File prêt



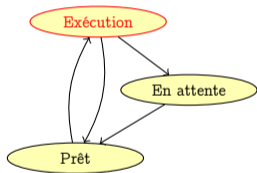
## Suspension de l'exécution

- Demande d'E/S  
→ File en attente
- Temps écoulé  
→ File prêt
- Crée un processus  
→ File prêt



## Suspension de l'exécution

- Demande d'E/S  
→ File en attente
- Temps écoulé  
→ File prêt
- Crée un processus  
→ File prêt
- Appel système *wait*  
→ File en attente



## Suspension de l'exécution

- Demande d'E/S  
→ File en attente
- Temps écoulé  
→ File prêt
- Crée un processus  
→ File prêt
- Appel système *wait*  
→ File en attente

# Interface



### Création par clonage

Un processus est toujours créé par un autre processus

## Création par clonage

Un processus est toujours créé par un autre processus

## Appel système fork

**Le processus est dupliqué**

- Code, mémoire et environnement sont identiques dans le père et le fils
- Seule la valeur de retour, le pid et le ppid diffèrent

## Création par clonage

Un processus est toujours créé par un autre processus

## Appel système fork

**Le processus est dupliqué**

- Code, mémoire et environnement sont identiques dans le père et le fils
- Seule la valeur de retour, le pid et le ppid diffèrent

## (En pratique)

Une technique de copie à l'écriture (COW) est utilisée par soucis d'efficacité.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main() {
    printf("Démarrage...\n");
    return 0;
}
```

```
#include ...  
int main() {  
    printf("Démarrage...\n");  
    return 0;  
}
```

```
#include ...  
int main() {  
    printf("Démarrage...\n");  
    pid_t proc = fork();  
    return 0;  
}
```

```
#include ...  
int main() {  
    printf("Démarrage...\n");  
    pid_t proc = fork();  
    if (proc == -1) {  
        fprintf(stderr, "fork failed\n");  
    }  
    return 0;  
}
```

```
#include ...
int main() {
    printf("Démarrage...\n");
    pid_t proc = fork();
    if (proc == -1) {
        fprintf(stderr, "fork failed\n");
    } else if (proc != 0) {
        /* c'est le père */
    } else {
        /* c'est le fils */
    }
    return 0;
}
```



```
#include ...
int main() {
    printf("Démarrage...\n");
    ...
} else {
    /* c'est le fils */
    printf("%d:fils de %d \n",
           getpid(), getppid());
    sleep(5);
    printf("%d:termine \n",getpid());
    exit(5);
}
...
```

```
#include ...
int main() {
    printf("Démarrage...\n");
    ...
} else if (proc != 0) {
    /* c'est le père */
    printf("%d:père de %d \n",
           getpid(), proc);
    int r;
    waitpid(proc, &r, 0); /* attente */
    printf("%d:fils %d sort (code %d)\n",
           getpid(), proc, r);
} else{
    ...
}
```

À l'exécution

À l'exécution  
Démarrage...

À l'exécution

Démarrage...

29227:père de 29228

29228:fils de 29227

À l'exécution

Démarrage...

29227:père de 29228

29228:fils de 29227

29228:termine

## À l'exécution

Démarrage...

29227:père de 29228

29228:fils de 29227

29228:termine

29227:fils 29228 sort (code 1280)

### À l'exécution

Démarrage...

29227:père de 29228

29228:fils de 29227

29228:termine

29227:fils 29228 sort (code 1280)

1280 != 5

Utilisation des fonctions WIFEXITED, WEXITSTATUS...  
pour décoder la valeur de retour.



## Lancement d'un programme

L'exécution d'un programme se fait par recouvrement d'un processus

## Appel système exec

### Le processus est remplacé

- Le nouveau code est chargé à la place de l'ancien
- La mémoire (pile et tas) est réinitialisée
- L'environnement est conservé

```
#include <stdio.h>
#include <unistd.h>
int main() {
    char *arg[] = {"uname", "-a", NULL};
    execv("/bin/uname", argu);
    printf("Jamais exécuté !\n");
    return 0;
}
```

### Remarques :

- Différentes variantes de exec\* existent
- Le 1<sup>er</sup> argument est le nom du programme

## Problème

Comment lancer un autre programme depuis un autre processus sans disparaître ?

Solution : `fork` puis `exec`

```
pid_t fils = fork();
if (fils == -1)
    perror("fork failed !");
else if (fils == 0)
    execlp("uname", "uname", "-a", NULL);
/* suite du programme père */
```

# Conclusion

## Processus

- Un **processus** est un programme exécuté par un processeur
- Un processus est composé de **code** et de **données** : variables d'environnements, pile et tas
- Cycle de vie: **prêt**, **en exécution** et **en attente**

## Processus

- Un **processus** est un programme exécuté par un processeur
- Un processus est composé de **code** et de **données** : variables d'environnements, pile et tas
- Cycle de vie : **prêt**, **en exécution** et **en attente**

## Rôle de l'OS

- L'OS stocke les informations sur les processus dans des **PCB**
- L'OS assure la **consistance** de la mémoire
- Création de processus via **fork** et **exec**