Chapitre 0:
Rappels sur les processus et les threads

Cours 0

Plan du cours:

- Qu'est ce qu'un processus ?
- Les états d'un processus
- Les processus sous Linux
- Création d'un processus Linux (fork())
- Terminaison d'un processus Linux (exit())
- Attente de la fin d'un processus fils (wait())
- Remplacement d'espace d'adressage (exec())
- Qu'est ce qu'un thread ?
- Gestion des threads sous Linux

Plan du cours:

- Qu'est ce qu'un processus ?
- Les états d'un processus
- Les processus sous Linux
- Création d'un processus Linux (fork())
- Terminaison d'un processus Linux (exit())
- Attente de la fin d'un processus fils (wait())
- Remplacement d'espace d'adressage (exec())
- Qu'est ce qu'un thread ?
- Gestion des threads sous Linux

Qu’est ce qu’un processus ?

- Un processus est un programme en cours d’exécution, auquel est associé un:

  1) **Environnement mémoire**: zone de code, de données, et de pile.

  2) **Environnement processeur**: CO, PSW, registres généraux, etc.

- Ces deux environnement sont appelés: **Contexte du processus.**

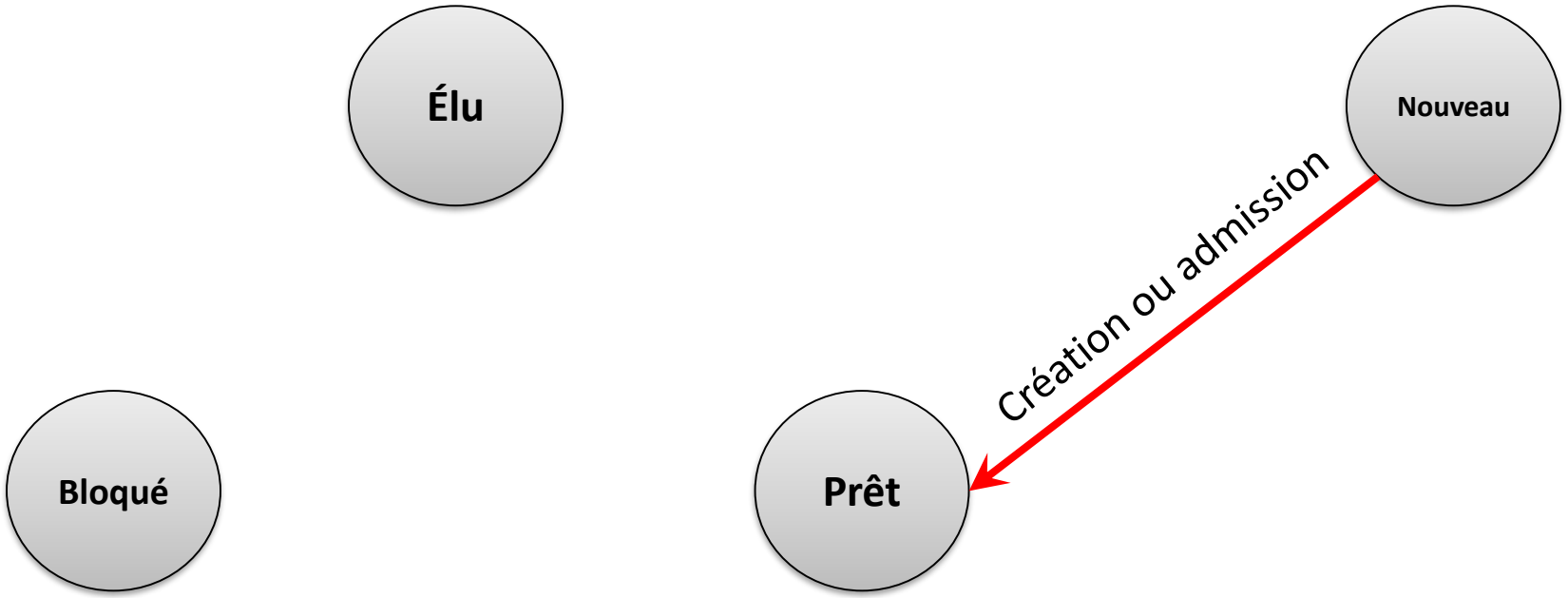
Plan du cours:

- Qu'est ce qu'un processus ?
- Les états d'un processus
- Les processus sous Linux
- Création d'un processus Linux (fork())
- Terminaison d'un processus Linux (exit())
- Attente de la fin d'un processus fils (wait())
- Remplacement d'espace d'adressage (exec())
- Qu'est ce qu'un thread ?
- Gestion des threads sous Linux

Les états d'un processus

Lors de son exécution, un processus passe par les états suivants:

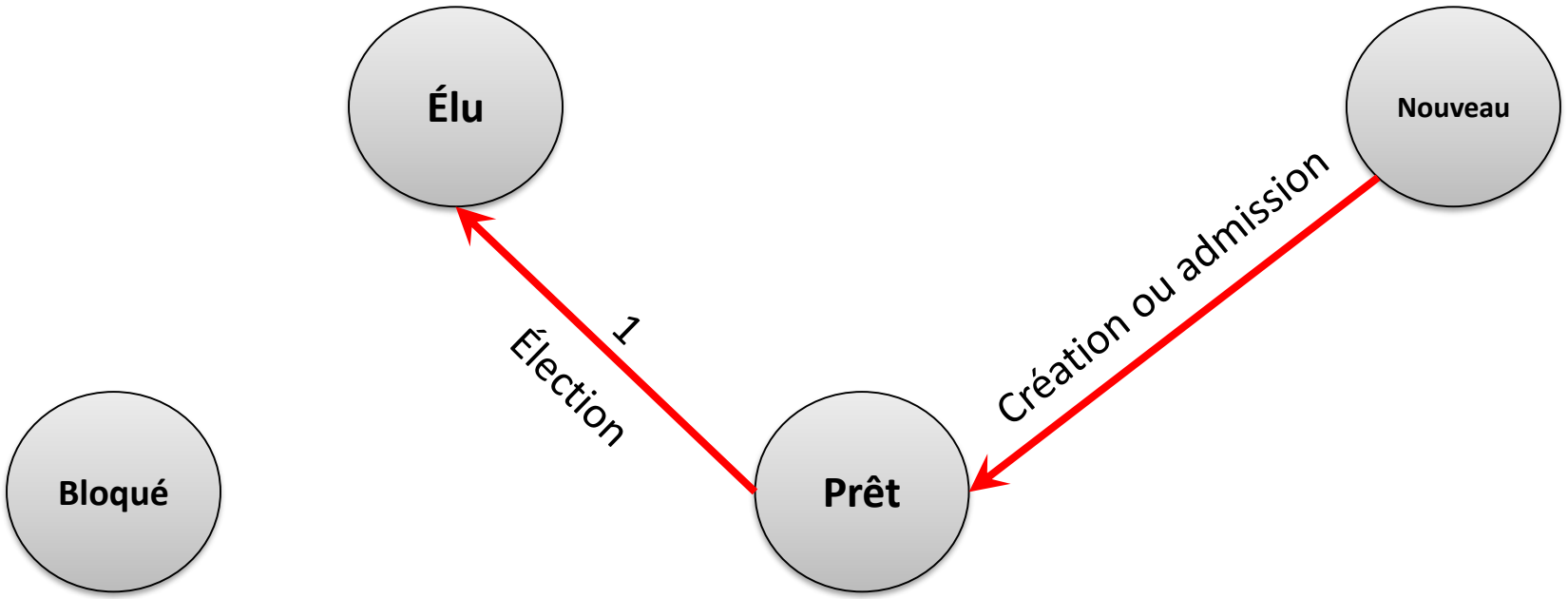
- **Prêt** (Ready)**:** c'est l'état d'**attente** du **processeur**.
- **Élu** (Running): en **cours d'exécution** par le processeur.
- **Bloqué** (Stopped)**:** c'est l'état d'**attente** d'une ressource **autre que** le processeur.



Les états d'un processus

Transition 1 : Élection

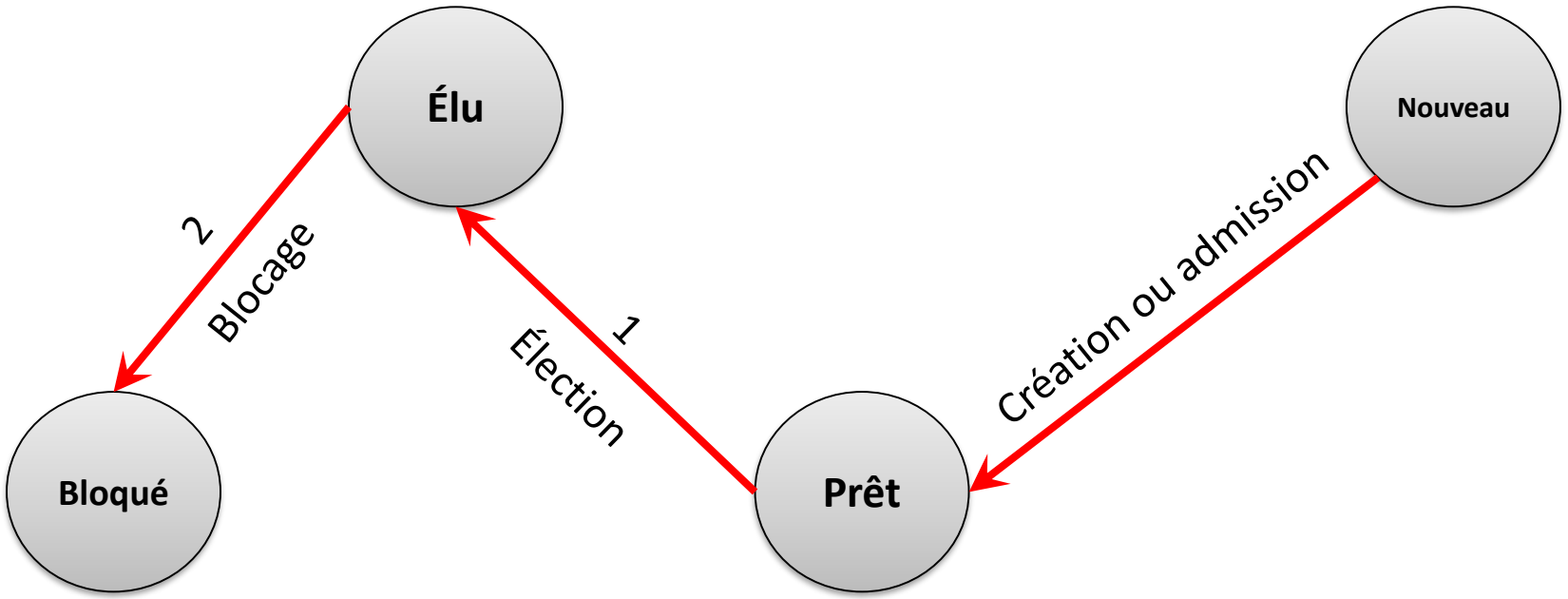
Celle-ci se produit lorsque le processeur est libre, alors le premier processus prêt peut l'utiliser (allouer).



Les états d'un processus

Transition 2 : Blocage

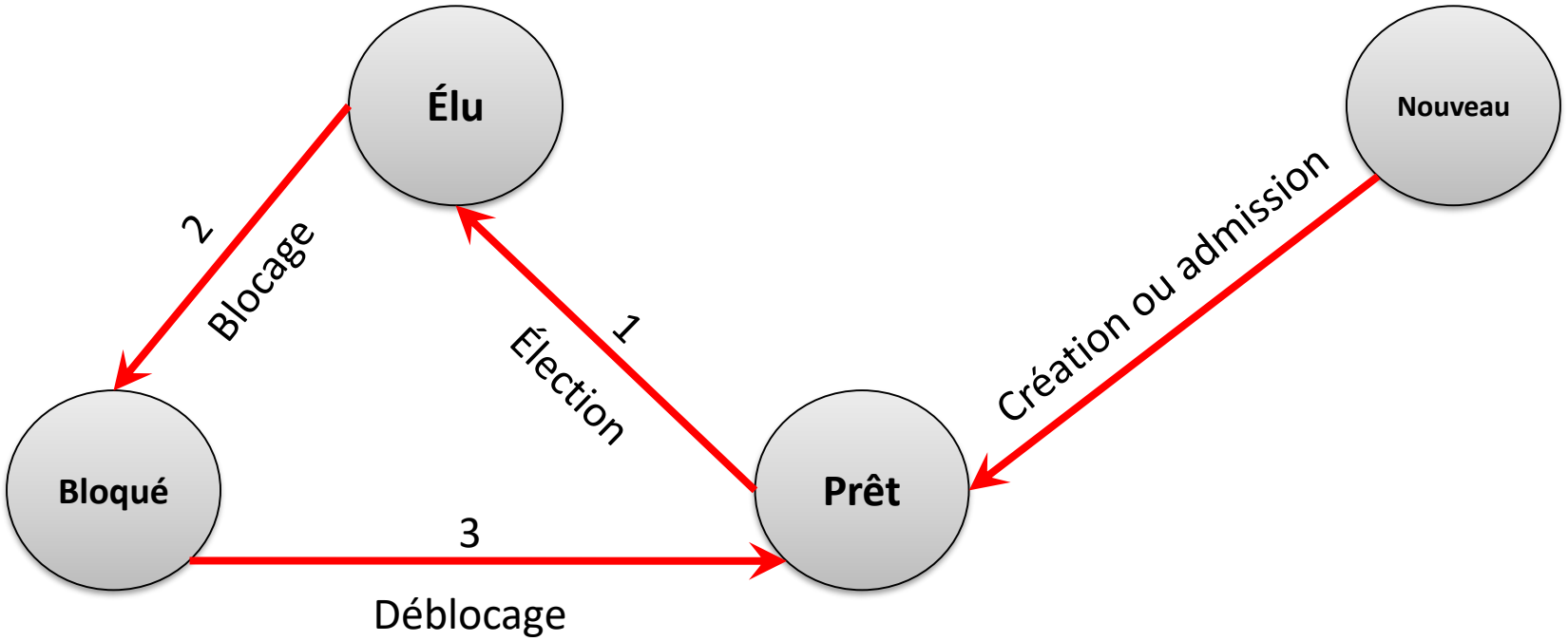
Quand un processus est élu. Il peut demander à accéder à une ressource (ex. E/S). Le processus quitte alors le processeur et passe à l'état bloqué.



Les états d'un processus

Transition 3 : Déblocage

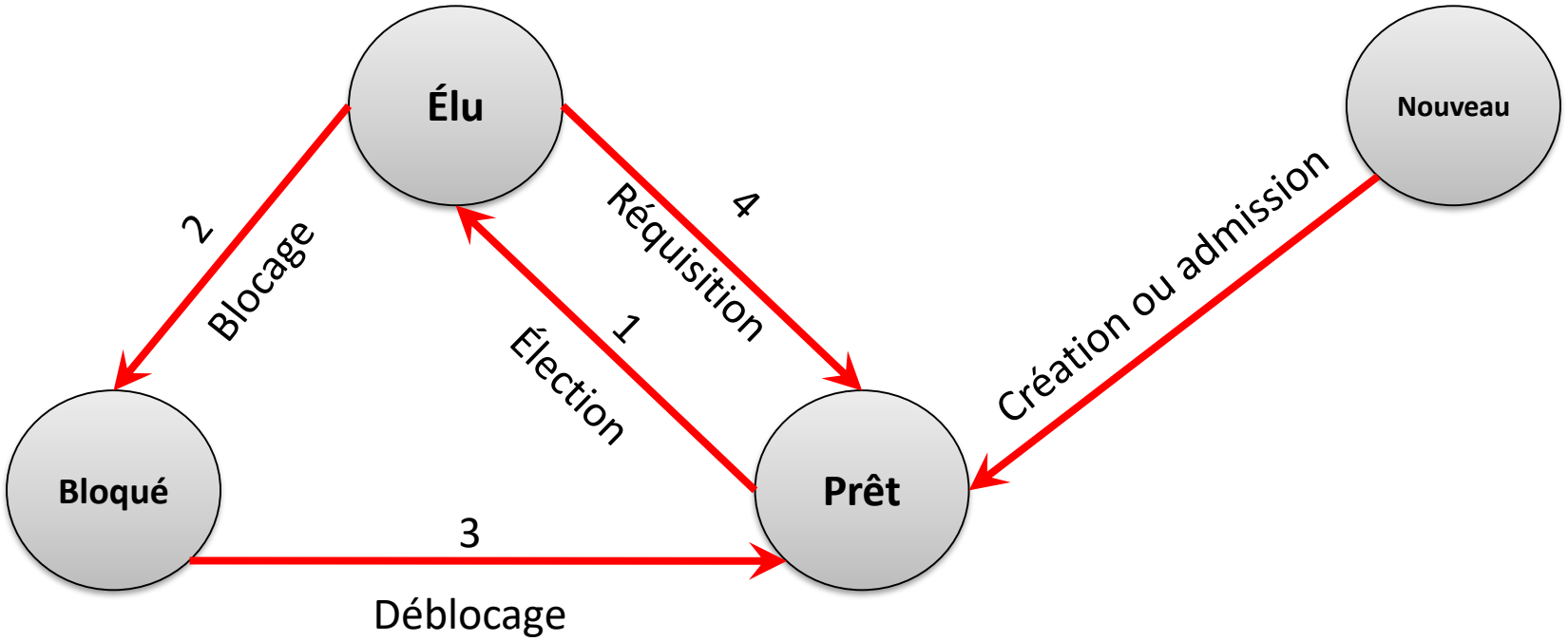
Elle est provoquée si la ressource attendue par le processus bloqué s'est produite.



Les états d'un processus

Transition 4 : Réquisition

Elle se produit lorsque l'**ordonnanceur** décide de retirer le processeur à un processus et l'allouer à un autre processus prêt (ex. fin de quatum).

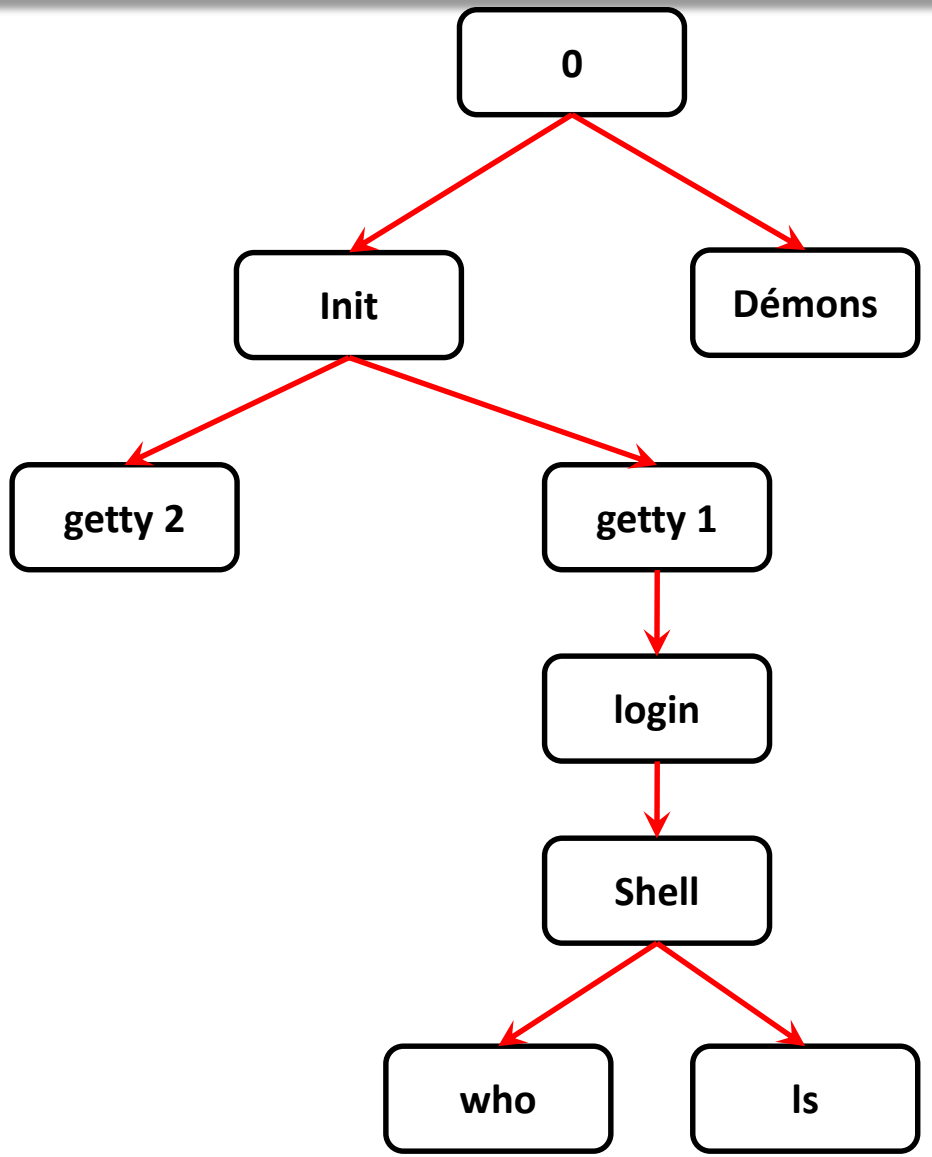


Plan du cours:

- Qu'est ce qu'un processus ?
- Les états d'un processus
- Les processus sous Linux
- Création d'un processus Linux (fork())
- Terminaison d'un processus Linux (exit())
- Attente de la fin d'un processus fils (wait())
- Remplacement d'espace d'adressage (exec())
- Qu'est ce qu'un thread ?
- Gestion des threads sous Linux

Les processus sous linux

- Lors du démarrage du PC, le **Bootstrap** (un programme de la ROM) charge une partie du SE dans la mémoire (RAM) pour créer le processus **0.**
- Ce processus **0** va créer deux processus: **Init** de pid **1** et **Démons** de pid **2**.
- Ensuite, d'autres processus sont crées à partir du processus **Init**.
- Chaque processus a un père. S'il perd son père, il est adopté par le processus **Init.**

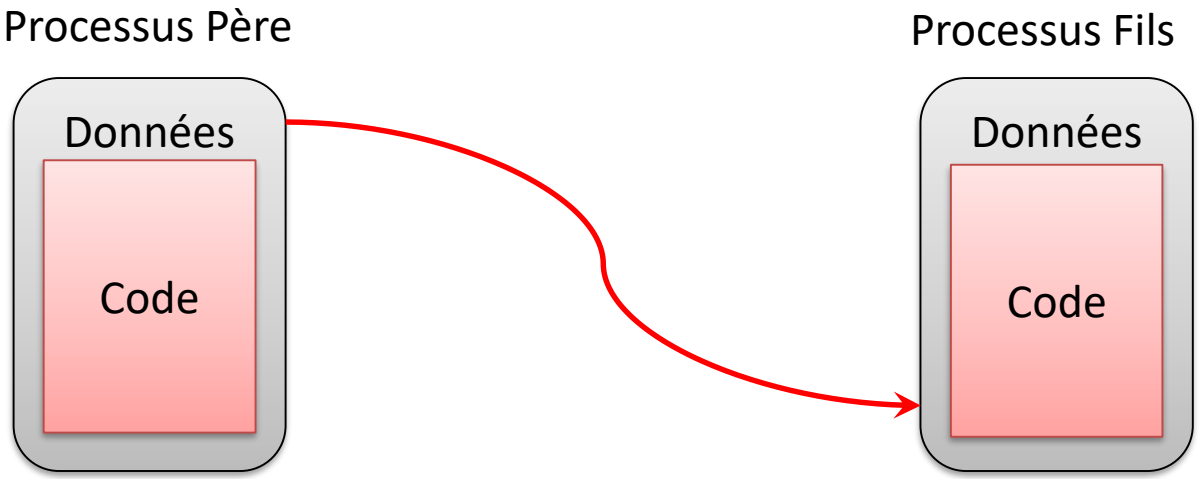


Plan du cours:

- Qu'est ce qu'un processus ?
- Les états d'un processus
- Les processus sous Linux
- Création d'un processus Linux (fork())
- Terminaison d'un processus Linux (exit())
- Attente de la fin d'un processus fils (wait())
- Remplacement d'espace d'adressage (exec())
- Qu'est ce qu'un thread ?
- Gestion des threads sous Linux

Création d'un processus Linux

- La création d'un nouveau processus est réalisée grâce à l'appel système **fork()** appartenant à la bibliothèque unistd.h
- Cette création se fait par duplication du code et des données du processus père,



- fork() est une fonction exécutée par le processus père, et qui retourne **-1** en cas d'echec de création, Sinon, elle retourne une valeur **nulle** au processus fils et une valeur **positif** (le PID du fils) au processus père.

Création d'un processus Linux

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
main ( )
{
int p;
p = fork();
if (p == 0)
    {                          / Code du fils /
    printf (" je suis le fils; mon pid est %d\n",getpid());
    // getpid() permet de récupérer le pid du processus.
    printf («  pid de mon père, %d\n ", getppid());
    // getppid() permet de récupérer le pid du processus.
    }
else
    {                          / Code du père /
    printf (" je suis le père ; mon pid est %d\n ", getpid());
    printf (" pid de mon fils ,%d\n" , p);
    }
}
```

Plan du cours:

- Qu'est ce qu'un processus ?
- Les états d'un processus
- Les processus sous Linux
- Création d'un processus Linux (fork())
- Terminaison d'un processus Linux (exit())
- Attente de la fin d'un processus fils (wait())
- Remplacement d'espace d'adressage (exec())
- Qu'est ce qu'un thread ?
- Gestion des threads sous Linux

Terminaison d'un processus

- Un processus se termine par une demande d'arrêt **volontaire** via :
  - ✓ l'appel système **void exit (int vstatus),**
  - ✓ ou par un arrêt forcé provoqué par un autre processus (le signal **kill**),
  - ✓ ou une erreur.
- Lorsqu'un processus fils se termine:
  - ✓ Son état de terminaison est enregistré dans son PCB.
  - ✓ La plupart des autres ressources allouées au processus sont libérées (ex. les fichiers ouverts).
  - ✓ Le processus passe à l'état **Zombie**.
- Si un processus est tué par un signal kill, ce signal est aussi envoyé à tous ses processus fils.

Plan du cours:

- Qu'est ce qu'un processus ?
- Les états d'un processus
- Les processus sous Linux
- Création d'un processus Linux (fork())
- Terminaison d'un processus Linux (exit())
- Attente de la fin d'un processus fils (wait())
- Remplacement d'espace d'adressage (exec())
- Qu'est ce qu'un thread ?
- Gestion des threads sous Linux

Attente de la fin d'un processus fils

L'appel système **int wait (int *state)** de la bibliothèque **<sys/wait.h>** permet à un processus père, soit:

- d'attendre la terminaison d'un de ses fils, soit
- de vérifier la terminaison d'un de ses fils.

Remarques:

- **wait (state)** bloque le processus appelant jusqu'à ce que quelconque de ses fils se termine.
- Dès que le père prends connaissance de la mort de son fils, il détruira son PCB.
- Le code retour de wait est Le PID du fils qui vient de se terminer.
- **waitpid (p, state, 0)** avec p>0, bloque le processus appelant jusqu'à ce que son fil de pid **p** soit terminé.

Attente de la fin d'un processus fils

```
main()
{
   int ret;
   ret = fork() ;
   if (ret == 0)
     {
     printf("je suis le fils; mon pid est %d\n ",getpid());
     printf("pid de mon père,%d\n ",getppid());
     exit(0);
     }
   else
     {
    printf("je suis le père; mon pid est %d\n ",getpid());
    printf("pid de mon fils,%d\n ",ret) ;
    wait(NULL);// son équivalent waitpid(-1, state, 0)
    exit(0);
    }
}
```

Plan du cours:

- Qu'est ce qu'un processus ?
- Les états d'un processus
- Les processus sous Linux
- Création d'un processus Linux (fork())
- Terminaison d'un processus Linux (exit())
- Attente de la fin d'un processus fils (wait())
- Remplacement d'espace d'adressage (exec())
- Qu'est ce qu'un thread ?
- Gestion des threads sous Linux

Remplacement d'espace d'adressage

- Linux offre une famille d'appels système **exec** de la librairie **<sys/types.h>** qui permettent à un processus de remplacer son code exécutable par un autre programme sous forme de **Path** ou **File**, spécifié comme paramètre de la fonction **exec.**
- La zone mémoire concernant le code et les données sera remplacé par celui du nouveau programme.
- On dit qu'il y a **recouvrement** en mémoire, mais pas de création d'un nouveau processus.

Remplacement d'espace d'adressage

- En cas de succès de l'appel système **exec**, l'ancien code est remplacée par le nouveau (test.c).

```
int main(){
pri  ("Hello 1"
execlp(  .c",…);
printf(  ");
ret  0;
}
```

Succès

test.c

```
int main(){
printf("Hello 3");
printf("Hello 4");
return 0;
}
```

Affichage à l'écran:

Hello 1

Hello 3

Hello 4

Remplacement d'espace d'adressage

- En cas d'échec, le processus poursuit l'exécution de son code à partir de l'instruction qui suit l'appel (donc il n'y a pas de remplacement de code).

```
int main(){

printf("Hello 1");

execlp(…,"test.c",…);

printf("Hello 2");

return 0;

}
```

Échec

```
int main(){

printf("Hello 1");

execlp(…,"test.c",…);

printf("Hello 2");

return 0;

}
```

Affichage à l'écran:

Hello 1

Hello 2

Remplacement d'espace d'adressage

Exemple: Écrasement du code hérité pour exécuter la commande ls -l

```
main()
{
int pid;
pid = fork();
if(pid == 0)
        {
        execlp("ls ", "ls ", "-l ", "/ ", NULL);
        printf("Bonjour tous le monde");
        }
else
        {
        wait();
        }
}
```

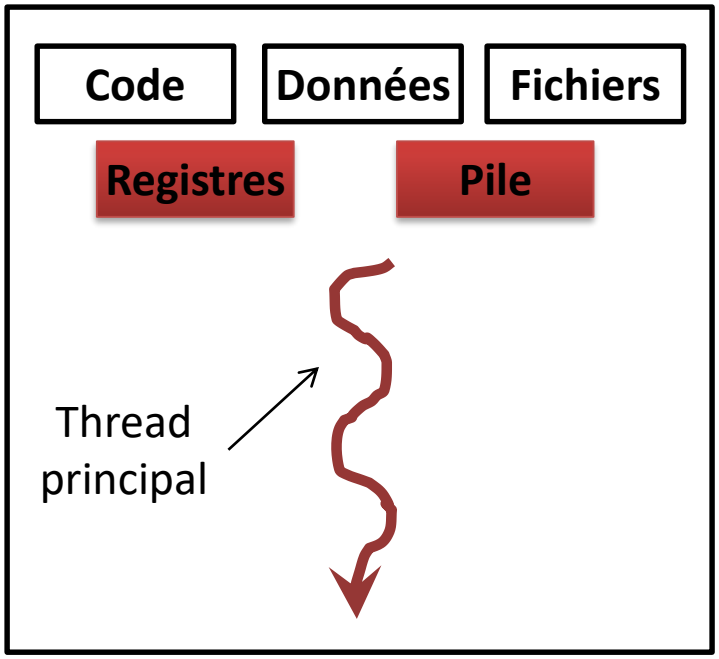
Remarque: Le processus conserve son PID, son espace mémoire alloué, sa table de descripteurs de fichier, et ses liens parentaux (processus fils et père).

Plan du cours:

- Qu'est ce qu'un processus ?
- Les états d'un processus
- Les processus sous Linux
- Création d'un processus Linux (fork())
- Terminaison d'un processus Linux (exit())
- Attente de la fin d'un processus fils (wait())
- Remplacement d'espace d'adressage (exec())
- Qu'est ce qu'un thread ?
- Gestion des threads sous Linux

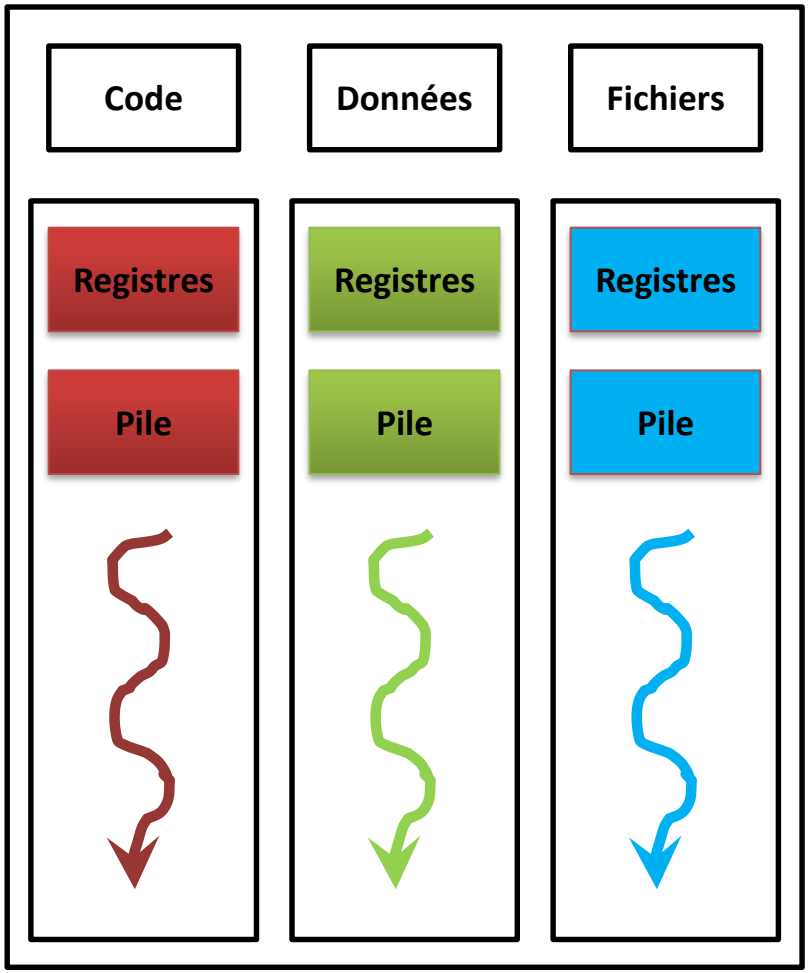
Qu’est ce qu’un thread ?

- En C, chaque processus commence par exécuter le programme principal **main**, qu’on appelle aussi le **thread principal**.
- Ce thread constitue l’unité d’exécution du processus appelé brièvement le **fil d’exécution**.



Qu’est ce qu’un thread ?

- D’une manière générale, un **thread** est une unité rattachée à un processus, chargée d’exécuter une partie du processus (une fonction).
- Les threads s’exécutent en concurrence avec le processus lourd.



Processus multithread

Plan du cours:

- Qu'est ce qu'un processus ?
- Les états d'un processus
- Les processus sous Linux
- Création d'un processus Linux (fork())
- Terminaison d'un processus Linux (exit())
- Attente de la fin d'un processus fils (wait())
- Remplacement d'espace d'adressage (exec())
- Qu'est ce qu'un thread ?
- Gestion des threads sous Linux

Gestion des threads sous Linux

Création de thread :

```
#include <pthread.h>

int pthread_create (pthread_t  *tid,
                    pthread_attr_t *attr,
                    void *nomFonction,
                    void *arg);
```

- Le service **pthread_create()** crée un processus léger qui exécute la fonction **nomFonction** avec l'argument **arg** et les attributs **attr**.
- La valeur **NULL** donne les valeurs par défauts aux attributs.

Gestion des threads sous Linux

Suspension de thread :

```
#include <pthread.h>

void pthread_join (pthread_t tid,

                   void *status);
```

- Attend la fin d'un thread. L'équivalent de **waitpid** des processus sauf qu'on doit spécifier le **tid** du thread à attendre.

Gestion des threads sous Linux

Terminaison de thread :

```
#include <pthread.h>

void pthread_exit( void *valeur_de_retour);
```

- **pthread_exit()** permet à un processus léger de terminer son exécution, en retournant l'état de terminaison.

Gestion des threads sous Linux

```
void* presentation()
    {
    printf("Je suis un thread.\n");
    pthread_exit(NULL);
    }

int main()
    {
    printf("Je suis le thread principal.\n");
    pthread_t toto;
    pthread_create(&toto, NULL, presentation, NULL);
    pthread_join(toto, NULL));
    printf("Je suis le thread principal.\n");
    return 0;
    }
```

Plan de la matière SEA

1) Les processus et les threads
2) Les interruptions
3) Les signaux
4) Système de gestion des Entrées / Sorties
5) Systèmes de gestion de fichiers
6) Gestion de la mémoire en C
7) Compilation et édition des liens