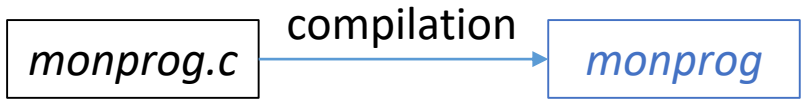
Séance 4
Processus

Différence entre programme et processus

Un programme : du code exécutable



un fichier exécutable (un programme)

Un processus : une session d'exécution d'un programme

Sur un terminal :
$ *./monprog* ==> création d'un processus qui exécute monprog

$ *fin exécution* ==> fin du processus

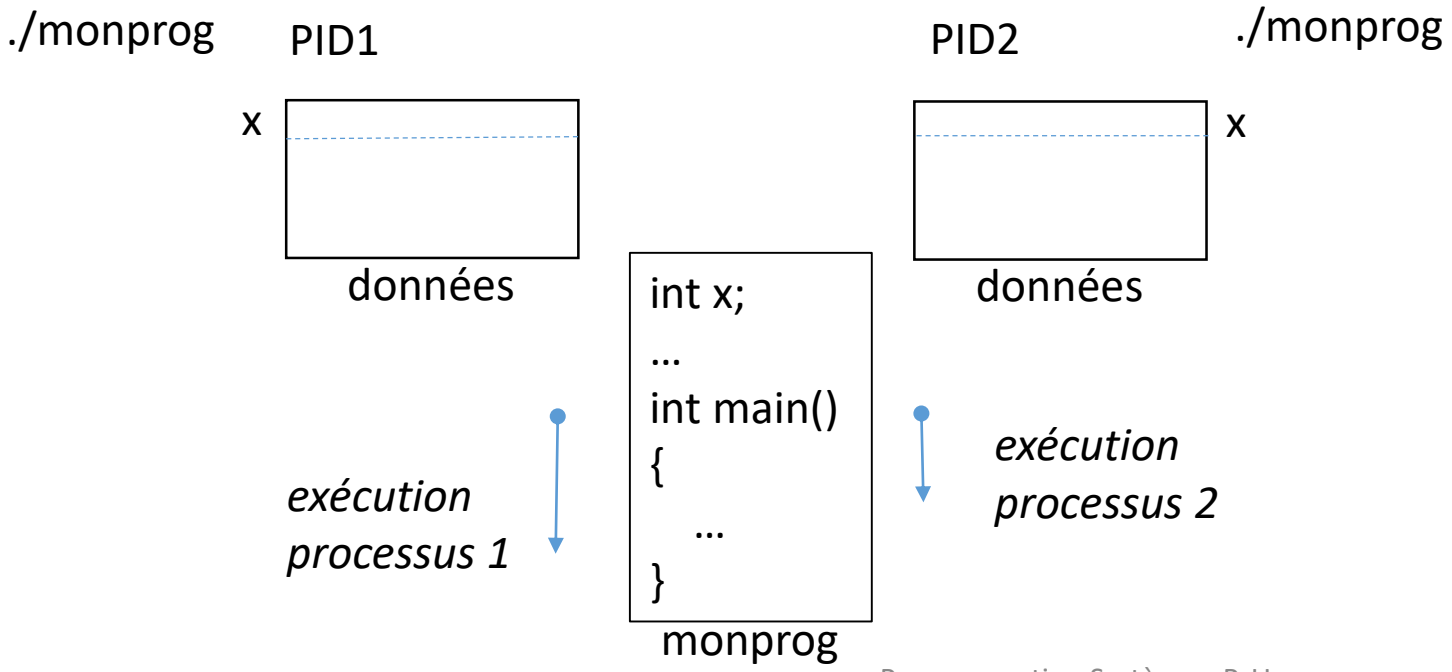
Sur un autre terminal :
$ ./monprog ==> création d'un autre processus qui exécute monprog

…

Caractéristiques d'un processus

Un processus est caractérisé par :

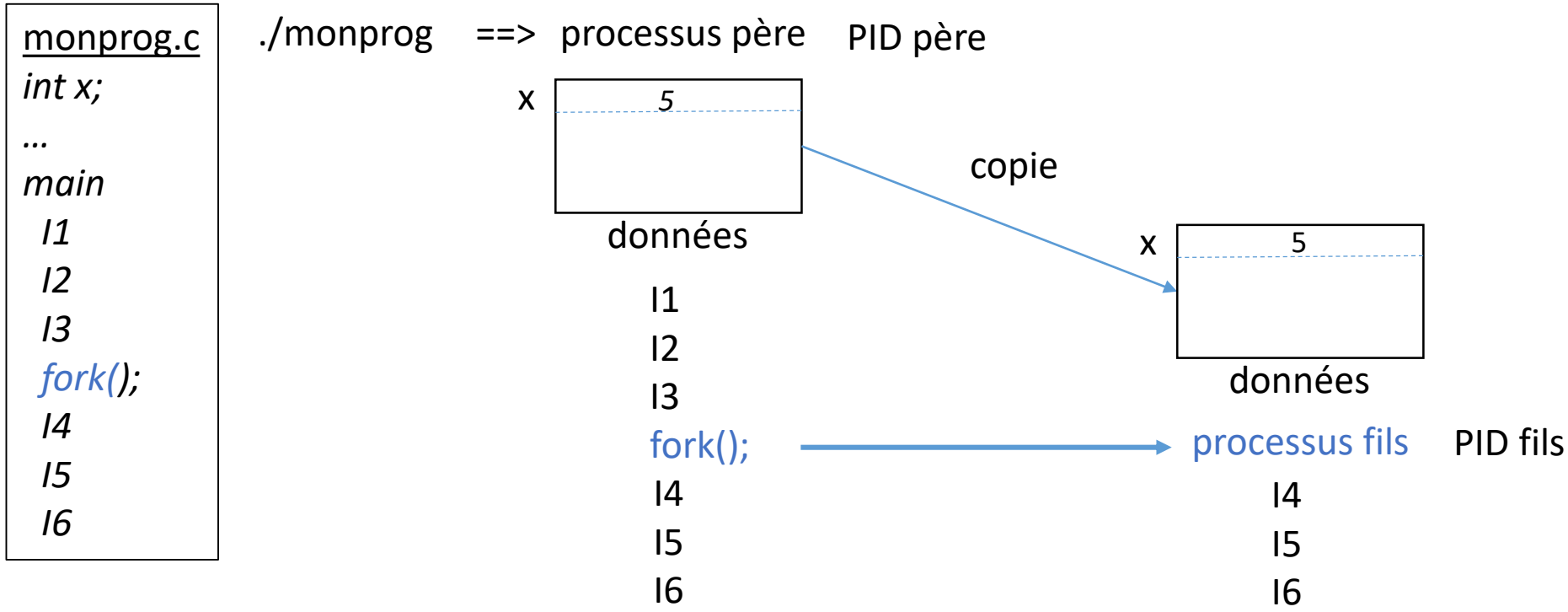
- le programme qu'il exécute (chargé en mémoire)
- son PID (process identification) : un no. > 0 attribué par Linux et qui l'identifie
- sa zone de données allouée en mémoire par Linux : variable globales + pile (variables locales) + tas (malloc)
- etc



Création d'un processus : fork

fork()

- création d'un nouveau processus dit processus fils
  - programme exécuté : le même que celui du père, l'exécution débute par l'instruction qui suit fork
  - zone de données : copie de celle du père au moment de fork



Création d'un processus : retour de fork

int fork() retourne **0** chez le **fils**

> 0 chez le **père** (égal au PID du fils) -1 si échec

monprog.c

```
...
int p;
...
p = fork();
if (p == 0)
   J1
   J2
   J3
else
   I4
   I5
   I6
```

père

```
...
...
...
p = fork();
if (p == 0)
I4
I5
I6
```

p non nul

fils

```
if (p == 0)
J1
J2
J3
```

p nul

codage de fork

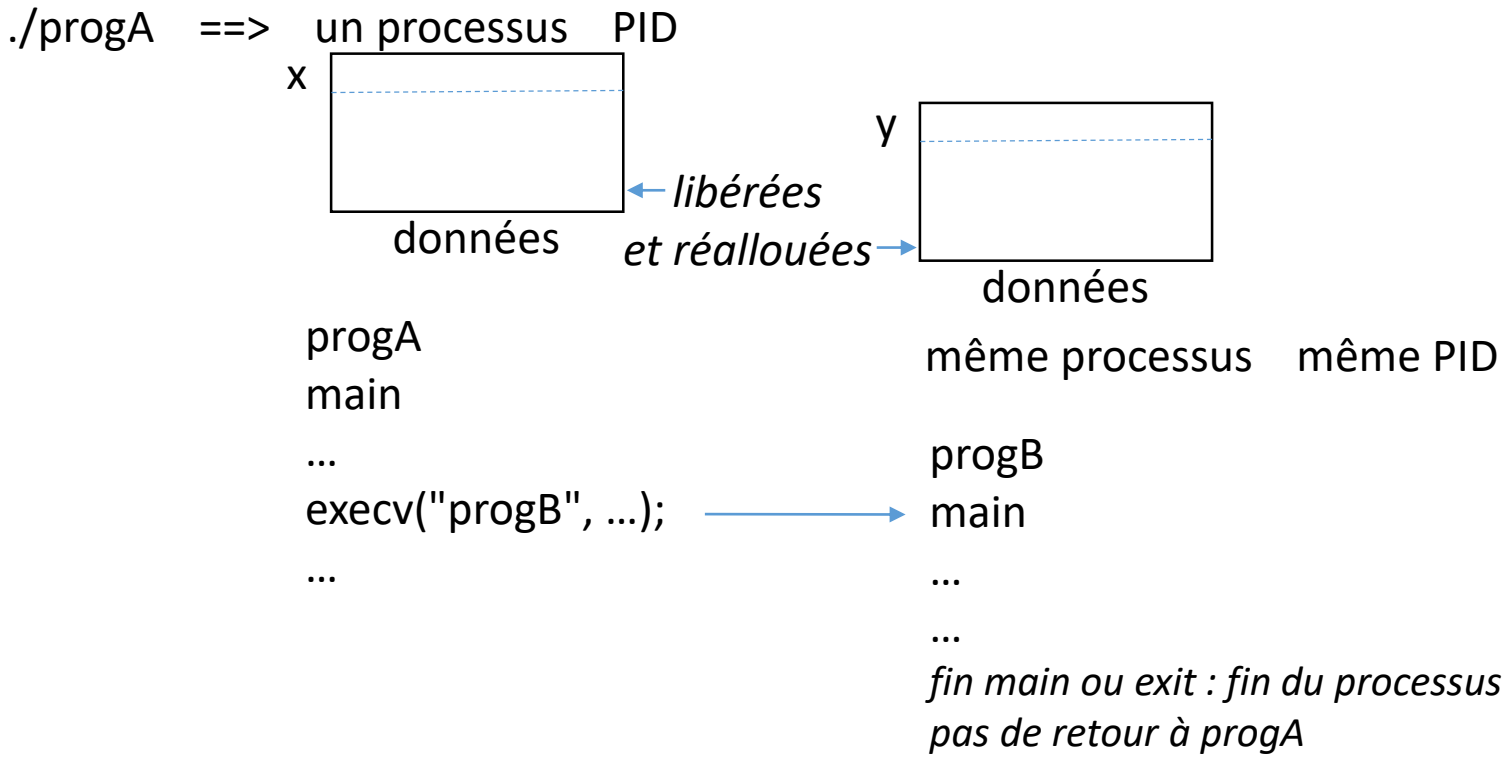
```
...
int p;
...
p = fork();
if (p == 0) {
   instructions du fils
}
else {
   instructions du père
}
si code ici : exécuté par père et fils
sauf si sortie avant (exit, …)
```

Changement du programme exécuté : execv

exec : famille de fonctions (execv, …) qui changent le programme exécuté par un processus

```
progA
int x;
…
main
…
execv("progB", …);
…
```

```
progB
float y;
…
main
…
```



Paramètres de execv

execv (*chemin du fichier exécutable*, *tableau des arguments à passer au programme*)

```
char* args[4];

args[0] = "progB";    // nom de la cde
args[1] = "42"        // 1er arg. de progB
args[2] = "hello"     // 2ème arg. de progB
args[3] = NULL;

execv("progB", args);
```

```
progB
…
int main(int argc, char* argv[])
….
```

Si pas d'argument à passer :

```
char* args[2];
args[0] = "progB";
args[1] = NULL;
execv("progB", args);
```

fork + exec

fork + exec pour faire exécuter au fils un programme différent de celui du père

```
...
int p;
...
p = fork();
if (p == 0) {
   // fils
   execv("progB", ...);
}
else {
   // père
}
```

Terminaison d'un processus

Terminaison d'un processus : **exit** ou return dans main.

exit (code);

code : valeur entière (8 bits) décidée par le programmeur
par convention exit(0) si sortie sans échec

Attente de la fin du fils : wait

Père et fils :

- exécutions indépendantes
- fin de l'un n'entraine pas fin de l'autre

wait : fonction à disposition du père s'il veut attendre la fin du fils

- bloque en attente de la fin du fils
- retourne le PID du fils qui s'est terminé (ou -1 si échec)

Si père pas intéressé par le code d'exit du fils : wait (NULL);

Si père veut récupérer le code d'exit du fils :
int status;
wait (&status); récupère dans status le code d'exit du fils (8 bits à l'intérieur de l'entier status)
+ appel à WEXITSTATUS (status) pour extraire de status le code d'exit

Fonctions getpid, getppid

getpid() : retourne le PID du processus appelant

getppid() : retourne le PID du père du processus appelant