Corrigé ED 2

.

Exercice 1

Solution

```
# include <unistd.h>
# include <stdio.h>

int main() {

int pid,i;

for (i=0; i<3;i++) {
   pid = fork();

   if (pid < 0 )
       {
      /* code si échec : printf ("le fork ( ) a échoué \n") */
       }
   else if (pid == 0)
       {
      printf("(i : %d) je suis le processus : %d, mon pere est : %d \n", i,
getpid(),getppid()) ;
        }
   else
       {
      printf("(i : %d) je suis le processus : %d, mon pere est : %d \n", i,
getpid(),getppid()) ;          }
}

return 0 ;
}
```

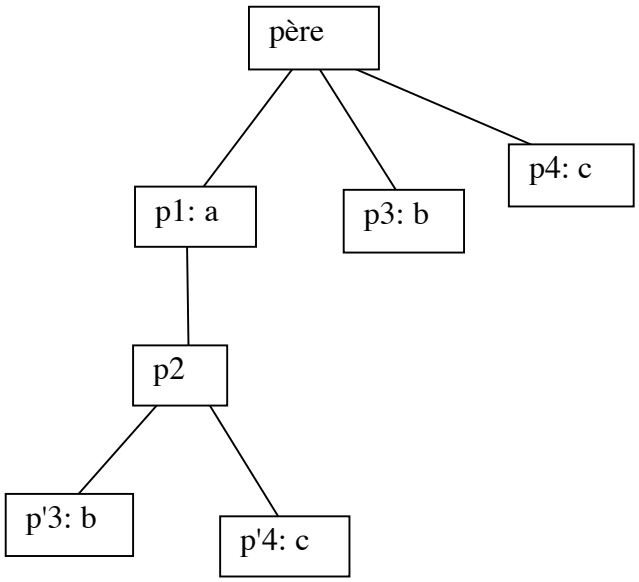
Combien de processus sont créés par le programme ?
Le processus de départ (PID1) va exécuter 3 fork donc créer 3 processus (PID2, PID3, PID4). Chacun de ces 3 fils possède le même segment de code que le père, et va donc s'exécuter là où PID1 en était :

- PID2 va donc exécuter 2 fork (PID5 et PID6)
- PID5 va reprendre là où en était PID2 et exécuter 1 fork (PID7)
- PID6 créé par le 3ème fork ne va rien faire
- PID3 va exécuter 1 fork (PID8)
- PID8 créé par le 3ème fork ne va rien faire
- En tout, il y aura 8 processus en parallèle.

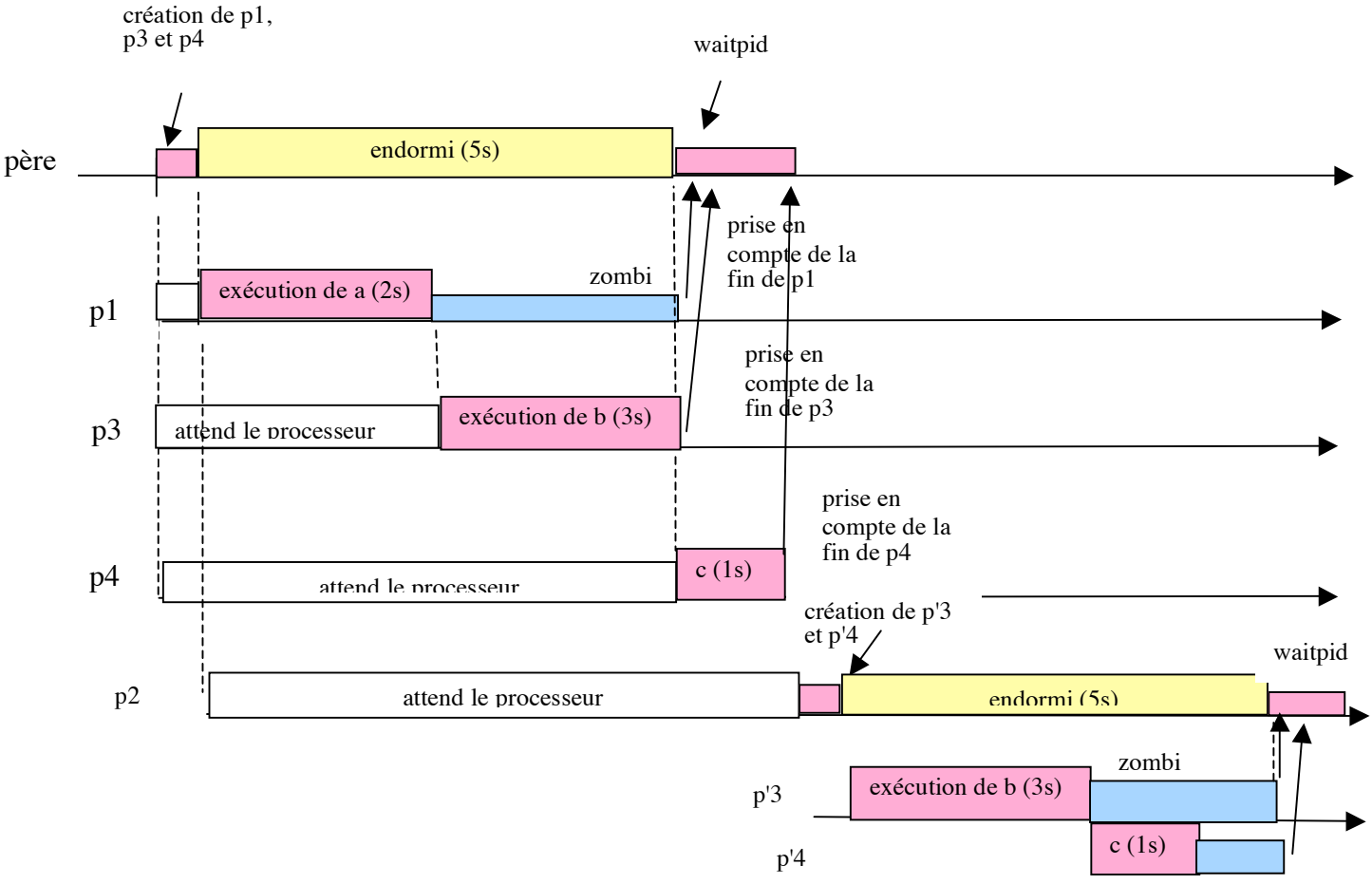
Exercice 2

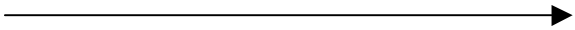
1. Le processus père crée 3 processus fils p1, p3 et p4. Le processus p1 crée un processus fils p2. Contrairement aux autres processus, le code du processus p2 ne se termine pas par une

primitive exit() qui terminerait le processus fils et retournerait un code au processus père p1. Si bien que p2 poursuit son exécution en créant un processus fils p'3 et p'4.
L'arborescence des processus est la suivante :



2. Dans cette solution, on suppose que les processus sont exécutés dans l'ordre de leur création. Dans la pratique, le dernier processus créé peut obtenir le processeur avant les processus créés avant lui.





L’objectif de la question 2 n’est pas de montrer comment on ordonnance les processus mais de mettre en évidence les différents états des processus