

# **TD1**

# **GESTION DES PROCESSUS**

# Exercise 1

2

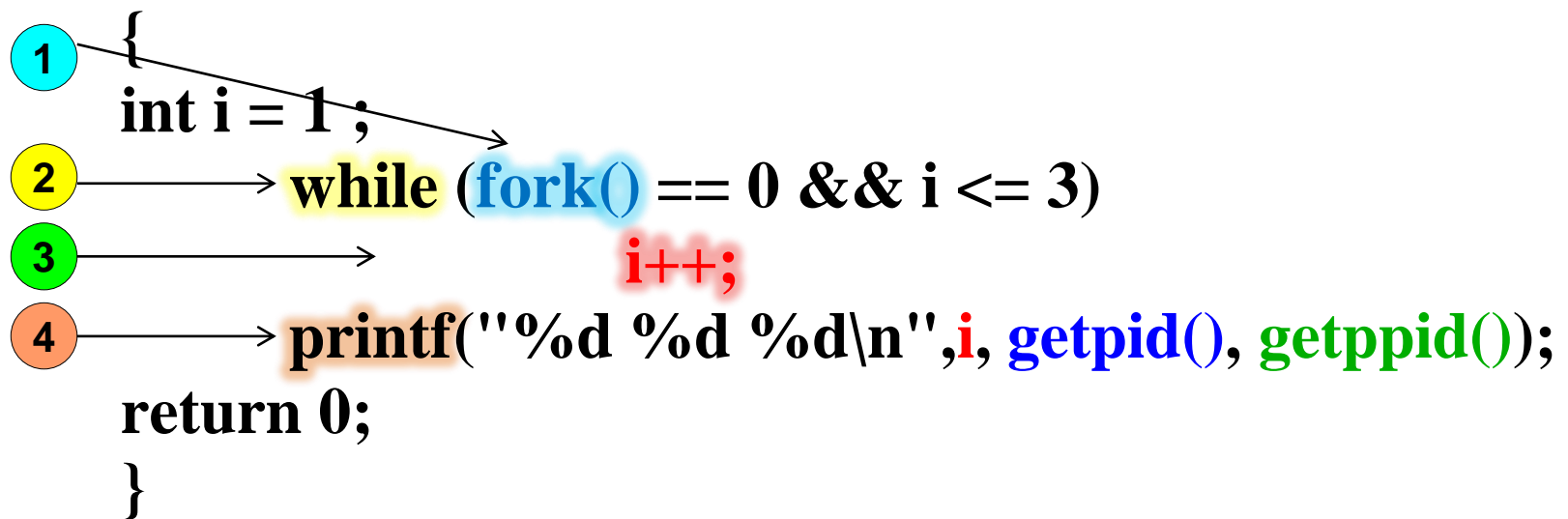
```
int main()
{
    int i = 1 ;
    while (fork() == 0 && i <= 3)
        i++;
    printf("%d %d %d\n", i, getpid(), getppid());
    return 0;
}
```

**1- a) Donner l'arborescence des processus**

# Exercise 1

3

```
int main()
{
1  int i = 1;
2  while (fork() == 0 && i <= 3)
3      i++;
4  printf("%d %d %d\n", i, getpid(), getppid());
  return 0;
}
```



# Exercice 1

Père

4

1 Création → Fils1

2 Faux

3 Sauté

4 Affichage 1

```
int main()
{
    int i = 1 ;

    while (fork() == 0 && i <= 3)
        i++;
    printf("%d %d %d\n", i, getpid(), getppid());

    return 0;
}
```

# Exercice 1

Père

5

1 Création → Fils1

2 Faux

2 Vrai

3 Sauté

3 i=2

4 Affichage 1

Retour

1 Création → Fils11

2 Faux

3 Sauté

4 Affichage 2

```
int main()
{
    int i = 1 ;

    while (fork() == 0 && i <= 3)
    {
        i++;
        printf("%d %d %d\n", i, getpid(), getppid());
    }

    return 0;
}
```

while (fork() == 0 && i <= 3)

i++;

printf("%d %d %d\n", i, getpid(), getppid());

return 0;

}

# Exercice 1

Père

6

1 Création → Fils1

2 Faux

2 Vrai

3 Sauté

3 i=2

4 Affichage 1

Retour

1 Création → Fils11

2 Faux

2 Vrai

3 Sauté

3 i=3

4 Affichage 2

retour

1 création → Fils111

2 Faux

3 Sauté

4 Affichage 3

```
int main()
{
    int i = 1 ;

    while (fork() == 0 && i <= 3)
        i++;
    printf("%d %d %d\n", i, getpid(), getppid());

    return 0;
}
```

while (fork() == 0 && i <= 3)

i++;

printf("%d %d %d\n", i, getpid(), getppid());

return 0;

}

# Exercice 1

Père

```
int main()
{
    int i = 1 ;

    while (fork() == 0 && i <= 3)
        i++;
    printf("%d %d %d\n", i, getpid(), getppid());

    return 0;
}
```

7

1 Création → Fils1

2 Faux

2 Vrai

3 Sauté

3 i=2

4 Affichage 1

Retour

1 Création → Fils11

2 Faux

2 Vrai

3 Sauté

3 i=3

4 Affichage 2

retour

1 création → Fils111

2 Faux

2 Vrai

3 Sauté

3 i=4

4 Affichage 3

retour

1 création → Fils1111

2 Faux

3 Sauté

4 Affichage 4

# Exercice 1

Père

```
int main()
{
    int i = 1 ;

    while (fork() == 0 && i <= 3)
        i++;
    printf("%d %d %d\n", i, getpid(), getppid());

    return 0;
}
```

8

1 Création → Fils1

2 Faux

2 Vrai

3 Sauté

3 i=2

4 Affichage 1

Retour

1 Création → Fils11

2 Faux

2 Vrai

3 Sauté

3 i=3

4 Affichage 2

retour

1 création → Fils111

2 Faux

2 Vrai

3 Sauté

3 i=4

4 Affichage 3

retour

1 création → Fils1111

2 Faux

2 Faux

3 Sauté

3 Sauté

4 Affichage 4

4 Affichage 4



# Exercice 1

Père

```
int main()
{
    int i = 1 ;

    while (fork() == 0 && i <= 3)
        i++;
    printf("%d %d %d\n", i, getpid(), getppid());

    return 0;
}
```

while (fork() == 0 && i <= 3)

i++;

printf("%d %d %d\n", i, getpid(), getppid());

return 0;  
}

9

1 Création → Fils1

2 Faux

2 Vrai

3 Sauté

3 i=2

4 Affichage 1

Retour

1 Création → Fils11

2 Faux

2 Vrai

3 Sauté

3 i=3

4 Affichage 2

retour

1 création → Fils111

2 Faux

2 Vrai

3 Sauté

3 i=4

4 Affichage 3

retour

1 création → Fils1111

2 Faux

2 Faux

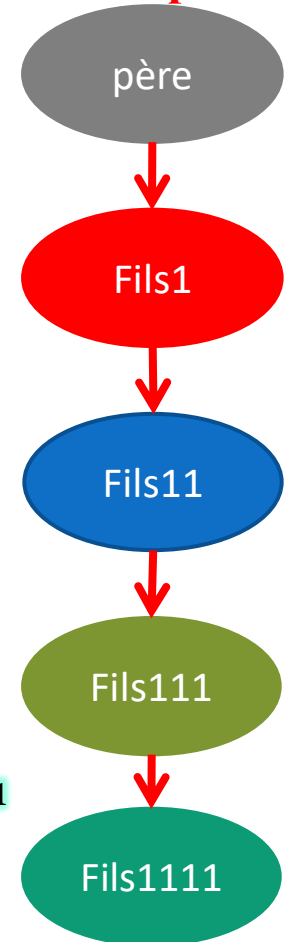
3 Sauté

3 Sauté

4 Affichage 4

4 Affichage 4

## L'arborescence des processus



# Exercise 1

10

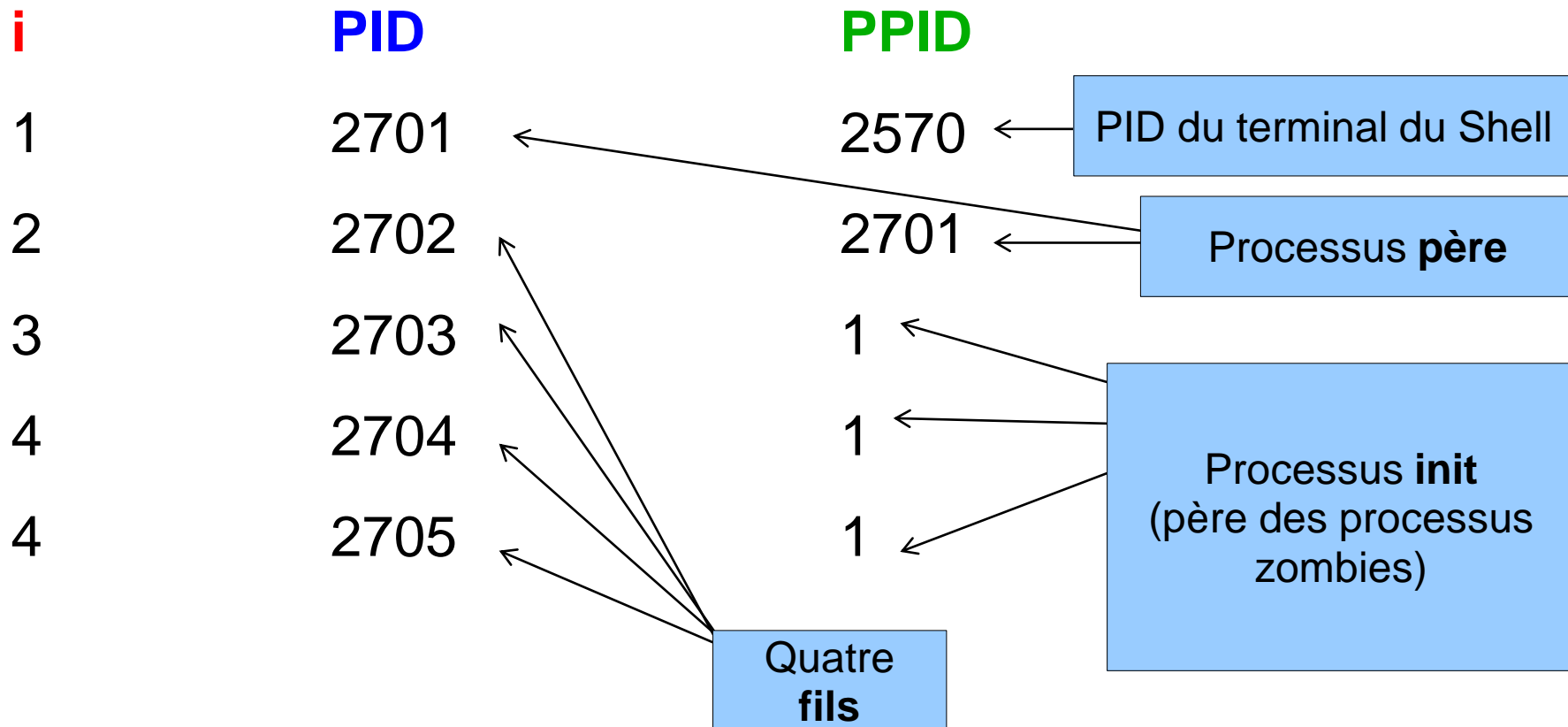
```
int main()
{
    int i = 1 ;
    while (fork() == 0 && i <= 3)
        i++;
    printf("%d %d %d\n", i, getpid(), getppid());
    return 0;
}
```

1- b) Donner l'output de ce programme

# Exercice 1

11

Output :



# Exercice 2

12

```
int main()
{
    fork() ; printf(" 1 \n") ;
    fork() ; printf(" 2 \n") ;
    return(0) ;
}
```

**1- a) Donner l'arborescence des processus crée**

# Exercice 2

13

```
int main()
```

```
{
```

```
1 → fork() ; printf(" 1 \n") ; ← 2
```

```
3 → fork() ; printf(" 2 \n") ; ← 4
```

```
return(0) ;
```

```
}
```

**1- a) Donner l'arborescence des processus crée**

# Exercice 2

14

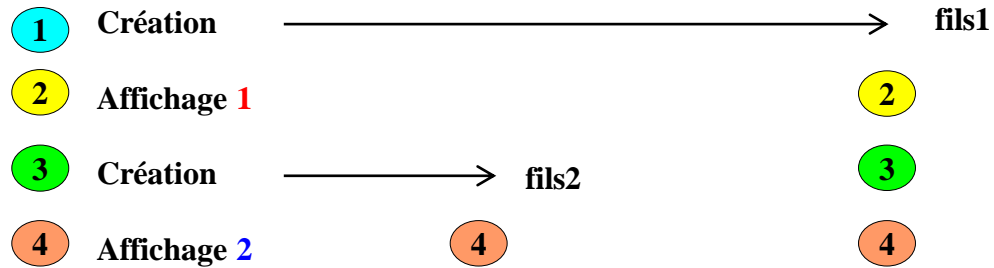
Père



# Exercice 2

15

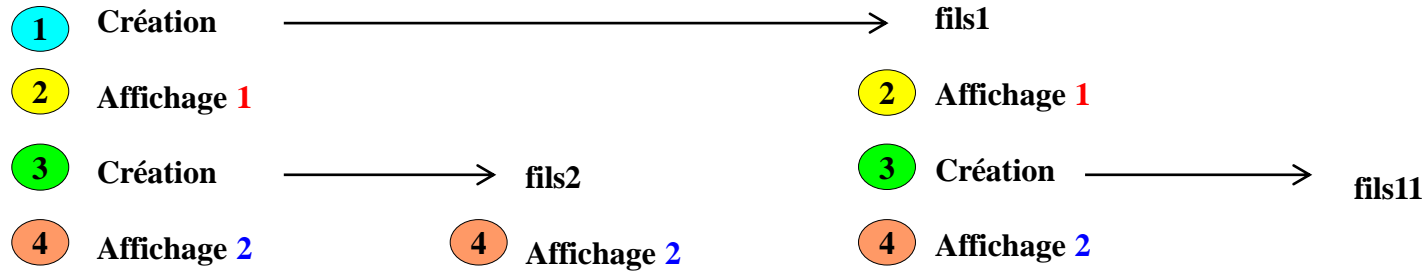
Père



# Exercice 2

16

Père

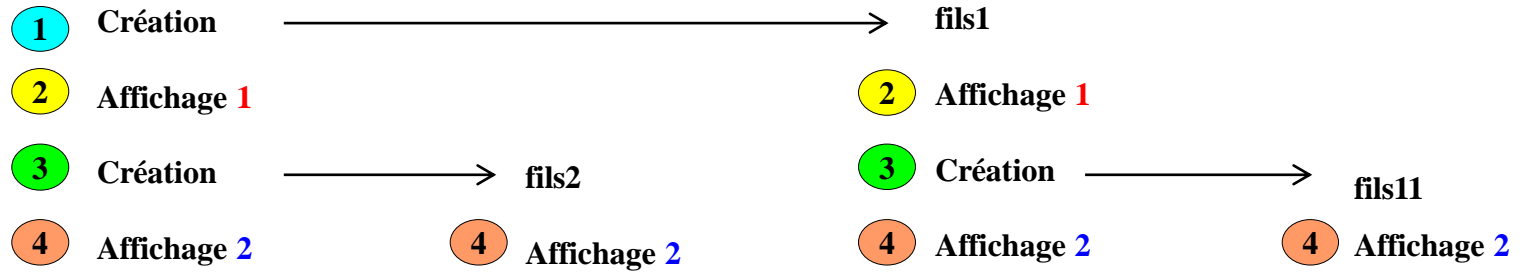




# Exercice 2

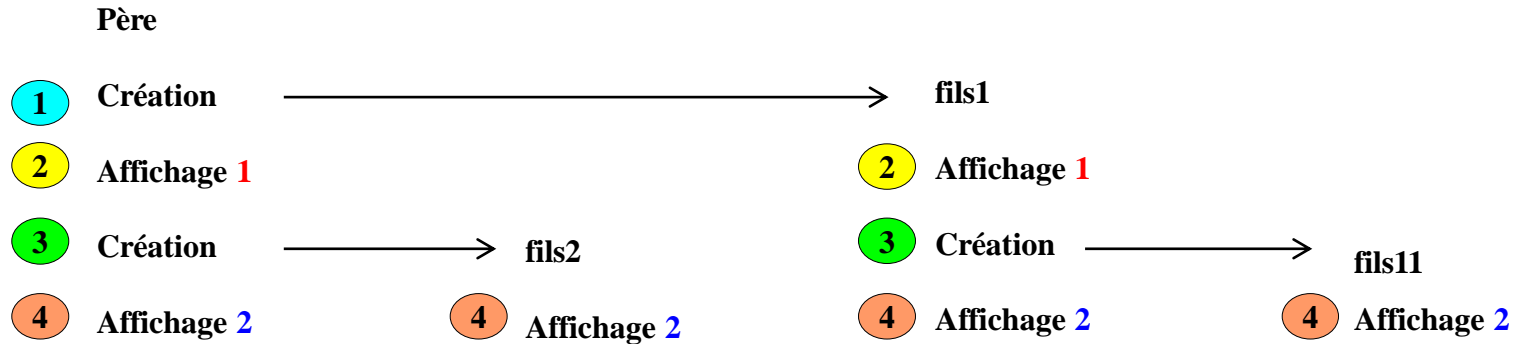
17

Père

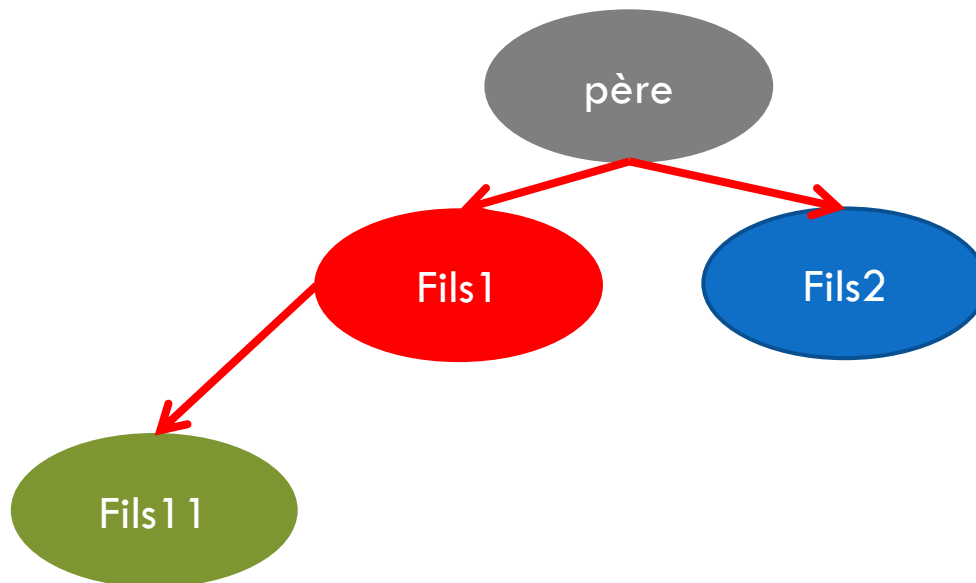


# Exercice 2

18



## L'arborescence des processus



# Exercise 2

19

```
int main()
{
    fork() ; printf(" 1 \n") ;
    fork() ; printf(" 2 \n") ;
    return(0) ;
}
```

**1- b) Donner l'output de ce programme**

# Exercise 2

20

**Output :**

1

1

2

2

2

2

# Exercice 2

21

```
int main()
{
    fork() ; printf(" 1 \n") ;
    fork() ; printf(" 2 \n") ;
    return(0) ;
}
```

**2- a)** Que se passe-t-il si l'on remplace seulement le premier `fork()` par `execlp("who", "who", NULL)` ?

# Exercise 2

22

```
int main()
{
    execlp ("who", "who", NULL) ;
    printf(" 1 \n") ;
    fork() ; printf(" 2 \n") ;
    return(0) ;
}
```

**who** : permet d'afficher des informations concernant des utilisateurs qui sont connectés.

# Exercice 2

23

## Output

user tty7 2014-09-01 11:24 (:0)

**exec** est une primitive de recouvrement qui permet de changer un processus par un autre.

Les instructions qui suivent l'exec ne sont jamais exécutées

```
int main()

{

    exelp ("who", "who", NULL) ;

    printf("fork 1 \n") ;

    fork() ; printf("fork 2 \n") ;

    return(0) ;

}
```

# Exercice 2

24

```
int main()
{
    fork() ; printf(" 1 \n") ;
    fork() ; printf(" 2 \n") ;
    return(0) ;
}
```

**3- Et si on remplace seulement  
le deuxième fork() par execlp("pwd", "pwd", NULL) ?**



# Exercise 2

25

```
int main()
```

```
{
```

```
1 → fork() ; printf(" 1 \n") ;
```

```
← 2
```

```
3 → execlp ("pwd", "pwd", NULL); printf(" 2 \n") ;
```

```
return(0) ;
```

```
}
```

**pwd** : "print working directory" permet d'afficher le chemin d'accès vers le répertoire où se situe l'utilisateur qui a entré la commande.

# Exercice 2

26

Père

① Création

② Affichage 1

③ pwd

Fils1



```
graph LR;
    A[Père] --> B[Fils1];
```

# Exercice 2

27

Père

① Création

② Affichage 1

③ pwd



Fils1

② Affichage 1

③ pwd

# Exercise 2

28

**Output :**

1

1

/home/user

/home/user

# Exercise 3

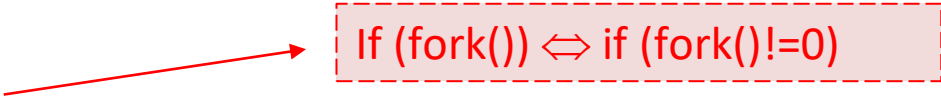
29

```
void main()
{
    int i=1;
    while (i<3)
    {
        if (fork())
            { printf("A: %d %d \n", getpid(), getppid()) ; break ; }
        else
            printf("B: %d %d \n", getpid(), getppid());
        i++;
    }
}
```

# Exercise 3

30

```
void main()
{
    int i=1;
    while (i<3)
    {
        if (fork())
            { printf("A: %d %d \n", getpid(), getppid()) ; break ; }
        else
            printf("B: %d %d \n", getpid(), getppid());
        i++;
    }
}
```



If (fork()) ⇔ if (fork() != 0)

# Exercise 3

31

```
void main()
```

```
{
```

```
    int i=1;
```

```
    ① → while (i<3)
```

```
    ② → {
```

```
    ③ → if (fork())
```

```
    ④ → { printf("A: %d %d \n", getpid(), getppid()) ; break ; }
```

```
        else
```

```
    ⑤ → printf("B: %d %d \n", getpid(), getppid());
```

```
    ⑥ → i++;
```

```
    }
```

```
}
```

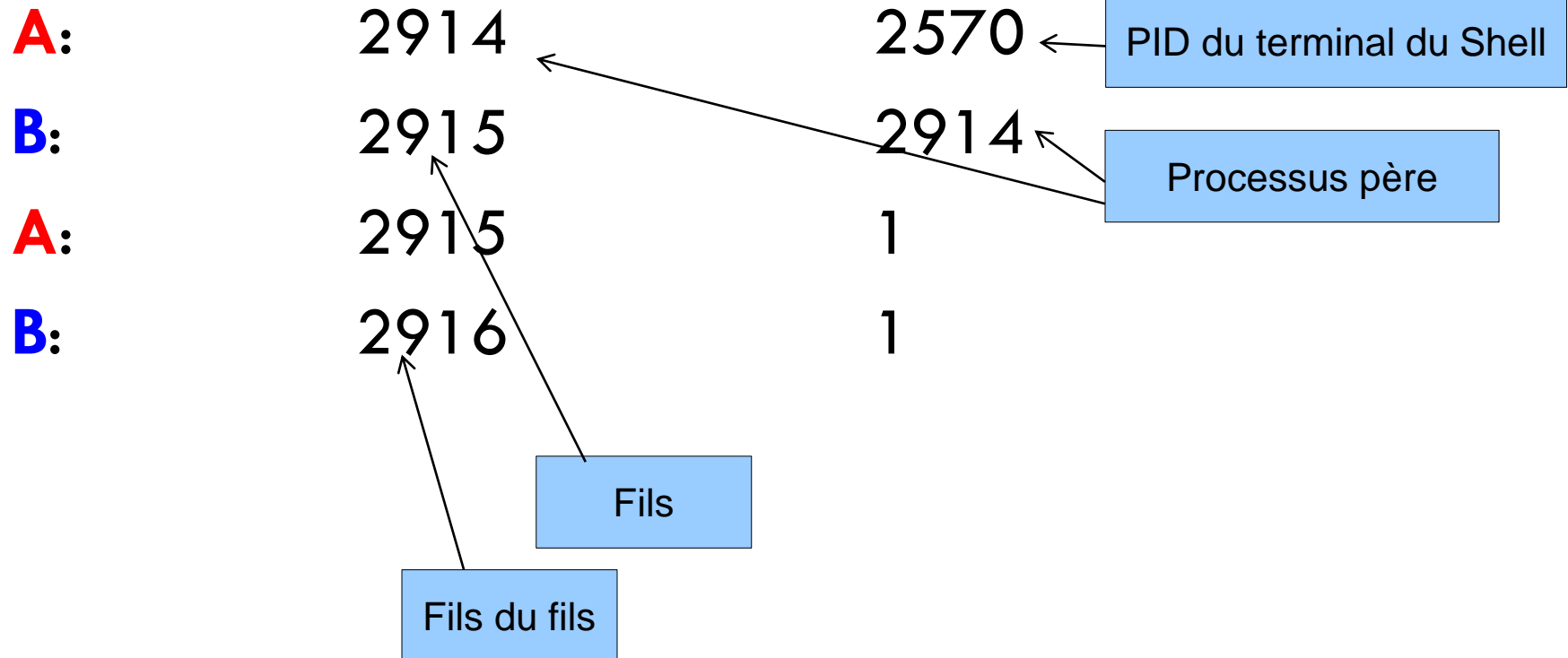
Pour indiquer  
Père ou Fils



# Exercice 3

32

Output :





# Exercice 3

Père

- 1 Vrai
- 2 Création → Fils1
- 3 Vrai
- 4 Affichage A

```
void main()
{
    int i=1;
    while (i<3)
    {
        if (fork())
            { printf("A: %d %d \n", getpid(), getppid()) ;
              break ; }
        else
            printf("B: %d %d \n", getpid(), getppid());

        i++;
    }
}
```

# Exercice 3

Père

- ① Vrai
- ② Création → Fils1
- ③ Vrai
- ④ Affichage A
- ③ Faux
- ④ sauté
- ⑤ Affichage B
- ⑥ i=2

Retour

- ① Vrai
- ② Création → Fils11
- ③ Vrai
- ④ Affichage A

```
void main()
{
    int i=1;
    while (i<3)
    {
        if (fork())
            { printf("A: %d %d \n", getpid(), getppid());
              break ; }
        else
            printf("B: %d %d \n", getpid(), getppid());
        i++;
    }
}
```

# Exercice 3

Père

- 1 Vrai
- 2 Création → Fils1
- 3 Vrai
- 3 Faux
- 4 Affichage A
- 4 sauté
- 5 Affichage B
- 6 i=2

Retour

- 1 Vrai
- 2 Création → Fils11
- 3 Vrai
- 3 Faux
- 4 Affichage A
- 4 sauté
- 5 Affichage B
- 6 i=3

Retour

- 1 Faux

```
void main()
{
    int i=1;
    while (i<3)
    {
        if (fork())
        { printf("A: %d %d \n", getpid(), getppid());
          break ; }
        else
        { printf("B: %d %d \n", getpid(), getppid());
          i++;
        }
    }
}
```

# Exercice 3

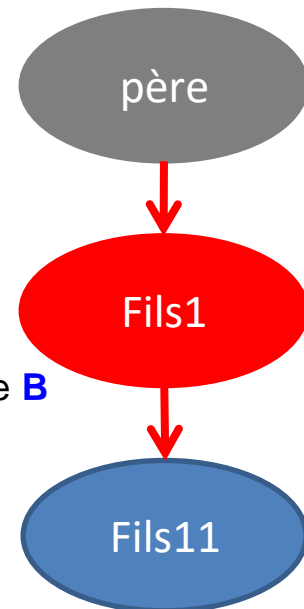
Père

- |               |               |
|---------------|---------------|
| 1 Vrai        |               |
| 2 Création    | → Fils1       |
| 3 Vrai        | 3 Faux        |
| 4 Affichage A | 4 sauté       |
|               | 5 Affichage B |
|               | 6 i=2         |
|               | Retour        |

```
void main()
{
    int i=1;
    while (i<3)
    {
        if (fork())
        { printf("A: %d %d \n", getpid(), getppid());
          break ; }
        else
        { printf("B: %d %d \n", getpid(), getppid());
          i++;
        }
    }
}
```

L'arborescence des processus

- |               |               |
|---------------|---------------|
| 1 Vrai        |               |
| 2 Création    | → Fils11      |
| 3 Vrai        | 3 Faux        |
| 4 Affichage A | 4 sauté       |
|               | 5 Affichage B |
|               | 6 i=3         |
|               | Retour        |
| 1 Faux        |               |



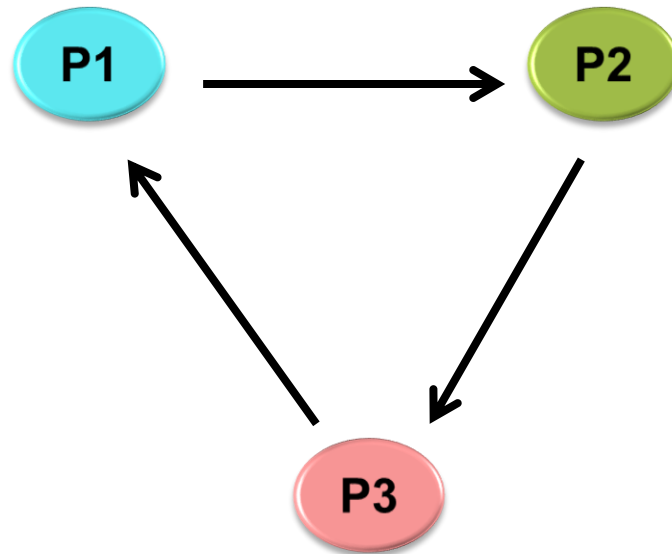
37

# Communication

# Exercice 4

38

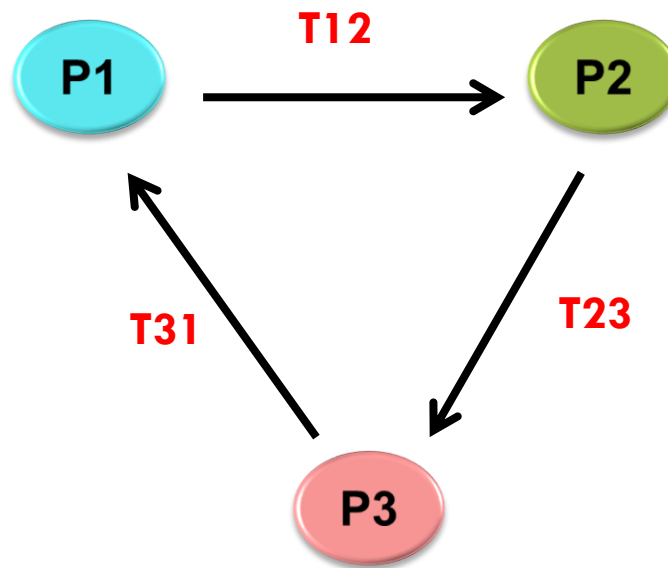
- ❖ Donner un programme C qui permet de réaliser une communication en anneau unidirectionnel avec les pipes entre trois processus.



# Exercice 4

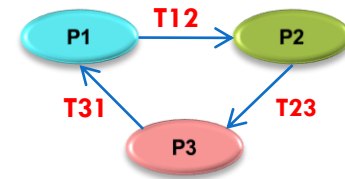
39

- ❖ Donner un programme C qui permet de réaliser une communication en anneau unidirectionnel avec les **pipes** entre trois processus.



- ❖ Pour chaque lien et chaque sens, il faut créer un pipe

# Exercice 4



40

## Anneau unidirectionnel

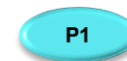
```
main () {  
    int T12[2], T23[2], T31[2];    char chaine1 [30], chaine2[30], chaine3[30];  
    pipe(T12); pipe(T23); pipe(T31);  
    if (fork())
```

```
    } else {
```

```
        // Partie du fils 1 qui est P1
```

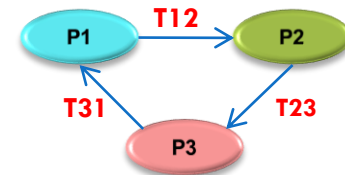
```
    }
```

```
}
```





# Exercice 4



41

## Anneau unidirectionnel

```
main () {  
    int T12[2], T23[2], T31[2];    char chaine1 [30], chaine2[30], chaine3[30];  
    pipe(T12); pipe(T23); pipe(T31);  
    if (fork())  
        if(fork())
```

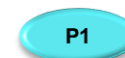
```
    } else {
```

```
        // Partie du fils 2 qui est P2
```

```
    } else {
```

```
    }
```

```
}
```



# Exercice 4

42

## Anneau unidirectionnel

```
main () {  
    int T12[2], T23[2], T31[2];    char chaine1 [30], chaine2[30], chaine3[30];  
    pipe(T12); pipe(T23); pipe(T31);  
    if (fork())  
        if(fork())  
            if(fork()) {  
                close (T12[0]); close (T12[1]);  
                close (T23[0]); close (T23[1]);  
                close (T31[0]); close (T31[1]);  
                wait(NULL); wait(NULL); wait(NULL);  
            }else {
```

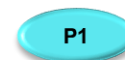
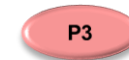
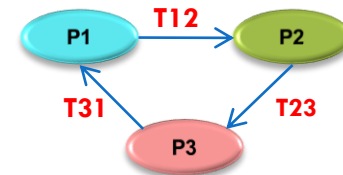
**// Partie du fils 3 qui est P3**

```
        } else {
```

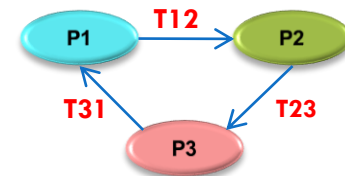
```
    } else {
```

```
    }
```

```
}
```



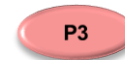
# Exercice 4



43

## Anneau unidirectionnel

```
main () {
    int T12[2], T23[2], T31[2];    char chaine1 [30], chaine2[30], chaine3[30];
    pipe(T12); pipe(T23); pipe(T31);
    if (fork())
        if(fork())
            if(fork()) {
                close (T12[0]); close (T12[1]);
                close (T23[0]); close (T23[1]);
                close (T31[0]); close (T31[1]);
                wait(NULL); wait(NULL); wait(NULL);
            } else {
                close (T23[1]);
                close (T31[0]);
                read(T23[0], chaine3, 30);
                write (T31[1], chaine3, 30);
                close (T31[1]);
                close (T23[0]);
            }
        } else {
            } else {
        }
    }
}
```

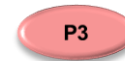
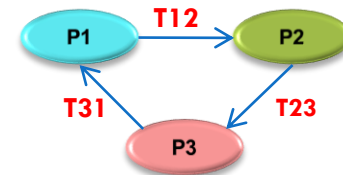


# Exercise 4

44

## Anneau unidirectionnel

```
main () {
    int T12[2], T23[2], T31[2];    char chaine1 [30], chaine2[30], chaine3[30];
    pipe(T12); pipe(T23); pipe(T31);
    if (fork())
        if(fork())
            if(fork()) {
                close (T12[0]); close (T12[1]);
                close (T23[0]); close (T23[1]);
                close (T31[0]); close (T31[1]);
                wait(NULL); wait(NULL); wait(NULL);
            } else {
                close (T23[1]);
                close (T31[0]);
                read(T23[0], chaine3, 30);
                write (T31[1], chaine3, 30);
                close (T31[1]);
                close (T23[0]);
            } else {
                close (T12[1]);
                close (T23[0]);
                read (T12[0], chaine2, 30);
                write (T23[1], chaine2, 30);
                close (T12[0]);
                close (T23[1]);
            } else {
                close (T31[1]);
                read (T31[0], chaine1, 30);
                write (T12[1], chaine1, 30);
                close (T12[1]);
                close (T31[0]);
            }
    }
}
```

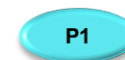
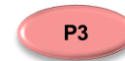
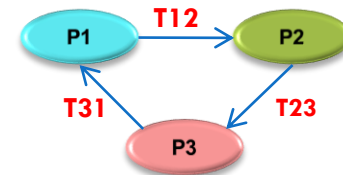


# Exercice 4

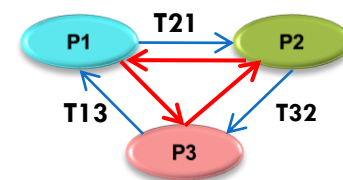
45

## Anneau unidirectionnel

```
main () {
    int T12[2], T23[2], T31[2];    char chaine1 [30], chaine2[30], chaine3[30];
    pipe(T12); pipe(T23); pipe(T31);
    if (fork())
        if(fork())
            if(fork()) {
                close (T12[0]); close (T12[1]);
                close (T23[0]); close (T23[1]);
                close (T31[0]); close (T31[1]);
                wait(NULL); wait(NULL); wait(NULL);
            } else {
                close (T23[1]);
                close (T31[0]);
                read(T23[0], chaine3, 30);
                write (T31[1], chaine3, 30);
                close (T31[1]);
                close (T23[0]);
            } else {
                close (T12[1]);
                close (T23[0]);
                read (T12[0], chaine2, 30);
                write (T23[1], chaine2, 30);
                close (T12[0]);
                close (T23[1]);
            } else {
                close (T12[0]);
                close (T31[1]);
                write (T12[1], "jeton 123\n", 30);
                read(T31[0], chaine1, 30);
                printf ("lu %s \n", chaine1);
                close(T12[1]);
                close(T31[0]);
            }
        }
    }
}
```



# Exercice 4



46

## Anneau Bidirectionnel

```

main () {
    int T12[2], T23[2], T31[2], T21[2], T32[2], T13[2];
    char chaine1 [30], chaine2[30], chaine3[30], chaine4 [30], chaine5 [30], chaine6 [30];
    pipe(T12); pipe(T23); pipe(T31); pipe(T21); pipe(T32); pipe(T13);
    if (fork())
        if(fork())
            if(fork()) {
                close (T12[0]); close (T12[1]); close (T21[0]); close (T21[1]);
                close (T23[0]); close (T23[1]); close (T32[0]); close (T32[1]);
                close (T31[0]); close (T31[1]); close (T13[0]); close (T13[1]);
                wait(NULL); wait(NULL); wait(NULL);
            }
            else {
                close (T23[1]);
                close (T31[0]);
                read(T23[0], chaine3, 30);
                write (T31[1], chaine3, 30);
                close (T31[1]);
                close (T23[0]);
            }
        }
        else {
            close (T12[1]);
            close (T23[0]);
            read (T12[0], chaine2, 30);
            write (T23[1], chaine2, 30);
            close (T12[0]);
            close (T23[1]);
        }
    }
    else {
        close (T12[0]);
        close (T31[1]);
        write (T12[1], "jeton 123\n", 30);
        read(T31[0], chaine1, 30);
        printf ("lu %s \n", chaine1);
        close(T12[1]);
        close(T31[0]);
    }
}
  
```

close (T13[1]);  
close (T32[0]);  
read(T13[0], chaine4, 30);  
write (T32[1], chaine4, 30);  
close (T32[1]);  
close (T13[0]);

close (T32[1]);  
close (T21[0]);  
read (T32[0], chaine5, 30);  
write (T21[1], chaine5, 30);  
close (T32[0]);  
close (T21[1]);

close (T13[0]);  
close (T21[1]);  
write (T13[1], "jeton inverse 132\n", 30);  
read(T21[0], chaine6, 30);  
printf ("lu2 %s \n", chaine6);  
close(T13[1]);  
close(T21[0]);

# Exercice 5

47

- ❖ Écrire un programme qui lit à l'écran le nombre de fils à créer puis les crée l'un à la suite de l'autre. Chaque fils affiche à l'écran son pid (`getpid()`) et celui de son père (`getppid()`).

# Exercice 5

48

- ❖ Écrire un programme qui lit à l'écran le nombre de fils à créer puis les crée l'un à la suite de l'autre. Chaque fils affiche à l'écran son pid (getpid()) et celui de son père (getppid()).

```
#include <stdio.h>
#include <sys/types.h>
#include <stdlib.h>
#include <unistd.h>
int main ()
{
    int i, n;
    printf(" Donner le nombre de fils à créer \n");
    scanf("%d",&n);
    for (i=1; i<=n; i++)
        if (! fork())
            {   printf("Processus fils : %d, mon père : %d\n", getpid(), getppid());
                break;
            }
    return 0;
}
```



# Exercice 6

49

- ❖ Ecrire un programme C qui permet de réaliser : **commande1 | commande2**

# Exercice 6

50

**Cas spécifique : ls -l | wc -c**

```
int main ()
{
    int fd[2];
    pipe (fd);
    if (fork())
    {
        if (fork()){
            close fd[0];    close fd [1];
            wait(NULL);    wait(NULL );

        }else {
            close (fd[1]);
            dup2(fd[0], 0);
            execlp("wc", "wc", "-c", NULL);

        }else {
            close (fd[0]);
            dup2(fd[1], 1);
            execlp("ls", "ls", "-l", NULL);

        }
    }
    return 0;
}
```

# Exercice 6

51

## Cas générique : commande1 | commande2

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main (int argc, char * argv[])
{ int tube[2];

if (argc != 3) { fprintf(stderr, "Syntaxe : %s commande_1 commande_2\n",argv[0]); exit(EXIT_FAILURE);}

if (pipe(tube) != 0) { perror("erreur du pipe"); exit(EXIT_FAILURE); }
switch (fork())
{
    case -1 :    perror("fork()");
                exit(EXIT_FAILURE);
    case 0 :    close(tube[0]);
                dup2(tube[1], STDOUT_FILENO);
                system(argv[1]);
                break;
    default :   close(tube[1]);
                dup2(tube[0], STDIN_FILENO);
                system(argv[2]);
                break;
}
return EXIT_SUCCESS;
}
```

# Exercice 6

52

## Cas générique : commande1 | commande2

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main (int argc, char * argv[])
{ int tube[2];
```

L'exécution se fait comme suit :

```
./prog6 "commande1" "commande2"
```

```
if (argc != 3) { fprintf(stderr, "Syntaxe : %s commande_1 commande_2\n",argv[0]); exit(EXIT_FAILURE);}
```

```
if (pipe(tube) != 0) { perror("erreur du pipe"); exit(EXIT_FAILURE); }
```

```
switch (fork())
```

```
{
```

```
    case -1 :    perror("fork()");
                 exit(EXIT_FAILURE);
```

```
    case 0 :    close(tube[0]);
                 dup2(tube[1], STDOUT_FILENO);
                 system(argv[1]);
                 break;
```

```
    default :    close(tube[1]);
                  dup2(tube[0], STDIN_FILENO);
                  system(argv[2]);
                  break;
```

```
}
```

```
return EXIT_SUCCESS;
```

```
}
```

# Exercice 7

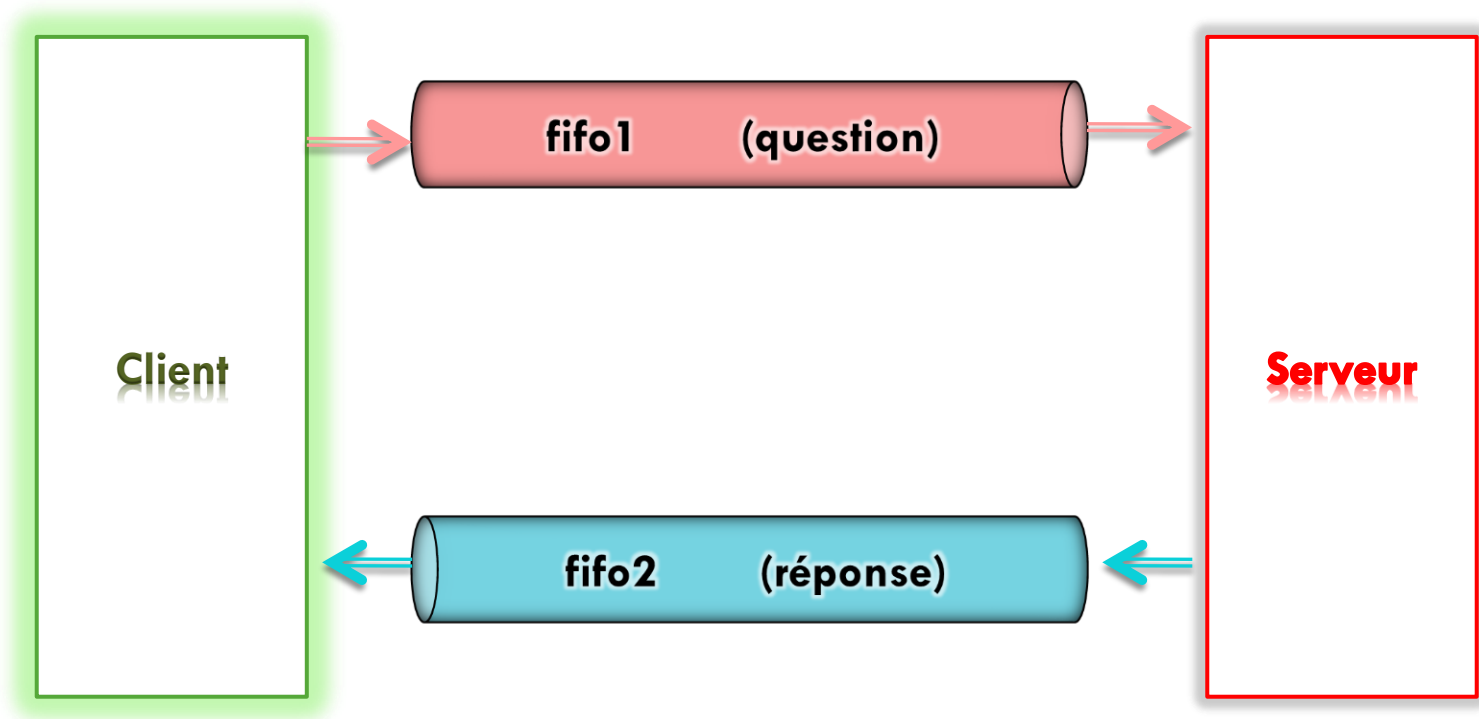
53

- ❖ Donner un programme C qui permet de réaliser une communication entre un client et un serveur en utilisant les **prises**.

# Exercice 7

54

- ❖ Rappelons le modèle de communication via une **pipe** :



# Exercice 7

55

## Client.c

```
int main () {  
    char chaine1 [1024], chaine2 [1024];  
    char * Question = "/tmp/client";  
    char * Reponse = "/tmp/serveur";  
  
    int d_E_Clt, d_L_Clt;  
    mkfifo (Question, 0644);  
  
    d_E_Clt = open (Question, O_WRONLY);  
    d_L_Clt = open (Reponse, O_RDONLY);  
  
    printf("donner la question" );  
    scanf (" %s", chaine1);  
    write (d_E_Clt, chaine1, 30);  
  
    read (d_L_Clt, chaine2, sizeof(chaine2));  
    printf("la réponse est : %s\n" , chaine2);  
  
    return 0; }
```

## Serveur.c

```
int main ()  
{  
    char chaine1 [1024], chaine2 [1024];  
    char * Question = "/tmp/client";  
    char * Reponse = "/tmp/serveur";  
  
    int d_E_Serv, d_L_Serv;  
    mkfifo (Reponse, 0644);  
  
    d_L_Serv = open (Question, O_RDONLY);  
    d_E_Serv = open (Reponse, O_WRONLY);  
  
    read (d_L_Serv, chaine1, sizeof(chaine1));  
    printf("la question est : %s\n" , chaine1);  
  
    printf("donner la réponse " );  
    scanf (" %s", chaine2);  
    write (d_E_Serv, chaine2, 30);  
  
    return 0; }
```

# Exercice 8

56

- ❖ Créer deux programmes qui traitent une même chaîne (**en mémoire partagée**). Le premier propose à l'utilisateur d'entrer une chaîne de caractères puis la place en mémoire partagée. Le second prend cette chaîne et l'inverse.



# Exercice 8

57

- ❖ Créer deux programmes qui traitent une même chaîne (**en mémoire partagée**). Le premier propose à l'utilisateur d'entrer une chaîne de caractères puis la place en mémoire partagée. Le second prend cette chaîne et l'inverse.
- ❖ Correction durant les séances du TP

# Exercice 9

58

Soit le programme suivant :

```
int main ( )
{   int fd;
    if (fork())
    {
        mkfifo("Fich1",0644);
        fd=open("Fich1", O_WRONLY);
        printf ("vraiment");
        dup2(fd,1) ;
        close (1);
        printf ("très");
        execlp("echo", "echo", "Bonjour", NULL) ;
        printf ("bien\n");
    }
    return 0; }
```

1. Quel est le moyen de communication employé
2. Quels sont les droits d'accès à ce moyen de communication
3. Donner l'output de ce programme

# Exercice 9

59

## 1. Moyen de communication :

Prise

```
int main ( )
{   int fd;
    if (fork())
    {
        mkfifo("Fich1",0644);
        fd=open("Fich1", O_WRONLY);
        printf ("vraiment");
        dup2(fd,1) ;
        close (1);
        printf ("très");
        execlp("echo", "echo", "Bonjour", NULL) ;
        printf ("bien\n");
    }
    return 0; }
```

# Exercice 9

60

## 2. Droits d'accès :

Propriétaire :

lecture et écriture

Groupe :

lecture

Autres :

lecture

```
int main ( )
{   int fd;
    if (fork())
    {
        mkfifo("Fich1",0644);
        fd=open("Fich1", O_WRONLY);
        printf ("vraiment");
        dup2(fd,1) ;
        close (1);
        printf ("très");
        execlp("echo", "echo", "Bonjour", NULL) ;
        printf ("bien\n");
    }
    return 0; }
```

# Exercice 9

61

3. Output de ce programme :

```
int main ( )
{   int fd;
    if (fork())
    {
        mkfifo("Fich1",0644);
        fd=open("Fich1", O_WRONLY);
        printf ("vraiment");
        dup2(fd,1) ;
        close (1);
        printf ("très");
        execlp("echo", "echo", "Bonjour", NULL) ;
        printf ("bien\n");
    }
    return 0; }
```

# Exercice 9

62

## 3. Output de ce programme :

```
int main ( )
{   int fd;
    if (fork())
    {
        mkfifo("Fich1",0644);
        fd=open("Fich1", O_WRONLY);
        printf ("vraiment");
        dup2(fd,1) ;
        close (1);
        printf ("très");
        execlp("echo", "echo", "Bonjour", NULL) ;
        printf ("bien\n");
    }
    return 0; }
```

Le programme n'affiche rien puisqu'il sera bloqué à cause de l'absence de consommateur (il n'y pas un programme qui fait appel à `fd=open("Fich1", O_RDONLY) ;`);

# Exercice 9

63

En mode non bloquant

```
int main ( )
{   int fd;
    if (fork())
    {
        mkfifo("Fich1",0644);
        fd=open("Fich1", O_WRONLY|O_NONBLOCK);
        printf ("vraiment");
        dup2(fd,1) ;
        close (1);
        printf ("très");
        execlp("echo", "echo", "Bonjour", NULL) ;
        printf ("bien\n");
    }
    return 0; }
```

Output ?

# Exercice 9

64

En mode non bloquant

```
int main ( )
{   int fd;
    if (fork())
    {
        mkfifo("Fich1",0644);
        fd=open("Fich1", O_WRONLY|O_NONBLOCK);
        printf ("vraiment");
        dup2(fd,1) ;
        close (1);
        printf ("très");
        execlp("echo", "echo", "Bonjour", NULL) ;
        printf ("bien\n");
    }
    return 0; }
```

Output ?

vraiment

echo: Erreur d'écriture : Mauvais descripteur de fichier



# Exercice 10

65

- ❖ En utilisant les sockets, créer un client et un serveur qui communiquent entre eux via un service donnée.

# Exercice 10

66

- ❖ En utilisant les sockets, créer un client et un serveur qui communiquent entre eux via un service donnée.
- ❖ Fait au niveau du cours et du Projet

# EXERCICES DE RÉVISION

**Programmation Système et Réseaux**

# Exercice 1: wait/exit

68

Exécuter le programme suivant et tirer les enseignements utiles.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
int main ()
{ int valeur, ret_fils, etat ;
  valeur=fork() ;
  switch (valeur)
  { case 0 : printf ("\t Proc fils de pid %d et du Pere de pid %d \n", getpid(), getppid() ) ;
             printf(" \t Je vais dormir 30 secondes ...\n") ;
             sleep (30) ;
             printf("\t Je me reveille et je termine mon execution par un EXIT(7)\n") ;
             exit (7) ;

    case -1 : printf ("Le fork a echoue") ;          exit(2) ;
    default : printf ("\t Proc pere de pid %d et Fils de pid %d \n", getpid(), valeur) ;
              printf ("\t J'attends la fin de mon fils...\n") ;
              ret_fils = wait (&etat) ;
              printf ("\t Mon fils de pid %d a termine,\n Son etat etait : %0x\n", ret_fils, etat) ;

  }
  return 0;}
```

# Exercice 1: wait/exit

69

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
int main ()
{ int valeur, ret_fils, etat ;
  valeur=fork() ;
  switch (valeur)
  { case 0 : printf ("\t Proc fils de pid %d et du Pere de pid %d \n", getpid(), getppid() ) ;
             printf(" \t Je vais dormir 30 secondes ...\n") ;
             sleep (30) ;
             printf("\t Je me reveille et je termine mon execution par un EXIT(7)\n") ;
             exit (7) ;
    case -1 : printf ("Le fork a echoue") ;          exit(2) ;
    default : printf ("\t Proc pere de pid %d et Fils de pid %d \n", getpid(), valeur) ;
             printf ("\t J'attends la fin de mon fils...\n") ;
             ret_fils = wait (&etat) ;
             printf ("\t Mon fils de pid %d a termine,\n Son etat etait : %0x\n", ret_fils, etat) ;
  }
  return 0;}
```

# Exercice 1: wait/exit

70

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
int main ()
{ int valeur, ret_fils, etat ;
  valeur=fork() ;
  switch (valeur)
  { case 0 : printf ("\t Proc fils de pid %d et du Pere de pid %d \n", getpid(), getppid() ) ;
             printf(" \t Je vais dormir 30 secondes ...\n") ;
             sleep (30) ;
             printf("\t Je me reveille et je termine mon execution par un EXIT(7)\n") ;
             exit (7) ;
    case -1 : printf ("Le fork a echoue") ;          exit(2) ;
    default : printf ("\t Proc pere de pid %d et Fils de pid %d \n", getpid(), valeur) ;
              printf ("\t J'attends la fin de mon fils...\n") ;
              ret_fils = wait (&etat) ;
              printf ("\t Mon fils de pid %d a termine,\n Son etat etait : %0x\n", ret_fils, etat) ;
  }
  return 0;}
```

Proc fils de pid 5052 et du Pere de pid 5051  
Je vais dormir 30 secondes ...

# Exercice 1: wait/exit

71

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
int main ()
{ int valeur, ret_fils, etat ;
  valeur=fork() ;
  switch (valeur)
  { case 0 : printf ("\t Proc fils de pid %d et du Pere de pid %d \n", getpid(), getppid() ) ;
             printf(" \t Je vais dormir 30 secondes ...\n") ;
             sleep (30) ;
             printf("\t Je me reveille et je termine mon execution par un EXIT(7)\n") ;
             exit (7) ;
    case -1 : printf ("Le fork a echoue") ;          exit(2) ;
    default : printf ("\t Proc pere de pid %d et Fils de pid %d \n", getpid(), valeur) ;
             printf ("\t J'attends la fin de mon fils...\n") ;
             ret_fils = wait (&etat) ;
             printf ("\t Mon fils de pid %d a termine,\n Son etat etait : %0x\n", ret_fils, etat) ;
  }
  return 0;}
```

Proc fils de pid 5052 et du Pere de pid 5051  
Je vais dormir 30 secondes ...  
Proc pere de pid 5051 et Fils de pid 5052  
J'attends la fin de mon fils...

# Exercice 1: wait/exit

72

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
int main ()
{ int valeur, ret_fils, etat ;
  valeur=fork() ;
  switch (valeur)
  { case 0 : printf ("\t Proc fils de pid %d et du Pere de pid %d \n", getpid(), getppid() ) ;
             printf(" \t Je vais dormir 30 secondes ...\n") ;
             sleep (30) ;
             printf("\t Je me reveille et je termine mon execution par un EXIT(7)\n") ;
             exit (7) ;

    case -1 : printf ("Le fork a echoue") ;          exit(2) ;
    default : printf ("\t Proc pere de pid %d et Fils de pid %d \n", getpid(), valeur) ;
              printf ("\t J'attends la fin de mon fils...\n") ;
              ret_fils = wait (&etat) ;
              printf ("\t Mon fils de pid %d a termine,\n Son etat etait : %0x\n", ret_fils, etat) ;

  }
  return 0;}
```

Proc fils de pid 5052 et du Pere de pid 5051

Je vais dormir 30 secondes ...

Proc pere de pid 5051 et Fils de pid 5052

J'attends la fin de mon fils...

Je me reveille et je termine mon execution par un EXIT(7)



# Exercice 1: wait/exit

73

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
int main ()
{ int valeur, ret_fils, etat ;
  valeur=fork() ;
  switch (valeur)
  { case 0 : printf ("\t Proc fils de pid %d et du Pere de pid %d \n", getpid(), getppid() ) ;
             printf(" \t Je vais dormir 30 secondes ...\n") ;
             sleep (30) ;
             printf("\t Je me reveille et je termine mon execution par un EXIT(7)\n") ;
             exit (7) ;

    case -1 : printf ("Le fork a echoue") ;          exit(2) ;
    default : printf ("\t Proc pere de pid %d et Fils de pid %d \n", getpid(), valeur) ;
              printf ("\t J'attends la fin de mon fils...\n") ;
              ret_fils = wait (&etat) ;
              printf ("\t Mon fils de pid %d a termine,\n Son etat etait : %0x\n", ret_fils, etat) ;

  }
  return 0;}
```

Proc fils de pid 5052 et du Pere de pid 5051

Je vais dormir 30 secondes ...

Proc pere de pid 5051 et Fils de pid 5052

J'attends la fin de mon fils...

Je me reveille et je termine mon execution par un EXIT(7)

Mon fils de pid 5052 a termine,

Son etat etait : 700

# Exercice 1: wait/exit

74

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
int main ()
{ int valeur, ret_fils, etat ;
  valeur=fork() ;
  switch (valeur)
  { case 0 : printf ("\t Proc fils de pid %d et du Pere de pid %d \n", getpid(), getppid() ) ;
             printf(" \t Je vais dormir 30 secondes ...\n") ;
             sleep (30) ;
             printf("\t Je me reveille et je termine mon execution par un EXIT(7)\n") ;
             exit (7) ;

    case -1 : printf ("Le fork a echoue") ;          exit(2) ;
    default : printf ("\t Proc pere de pid %d et Fils de pid %d \n", getpid(), valeur) ;
              printf ("\t J'attends la fin de mon fils...\n") ;
              ret_fils = wait (&etat) ;
              printf ("\t Mon fils de pid %d a termine,\n Son etat etait : %0x\n", ret_fils, etat) ;

  }
  return 0;}
```

Proc fils de pid 5052 et du Pere de pid 5051

Je vais dormir 30 secondes ...

Proc pere de pid 5051 et Fils de pid 5052

J'attends la fin de mon fils...

Je me reveille et je termine mon execution par un EXIT(7)

Mon fils de pid 5052 a termine,

Son etat etait : 700

La valeur de retour de  
wait est le PID du fils  
qui a terminé son  
exécution

# Exercice 1: wait/exit

75

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
int main ()
{ int valeur, ret_fils, etat ;
  valeur=fork() ;
  switch (valeur)
  { case 0 : printf ("\t Proc fils de pid %d et du Pere de pid %d \n", getpid(), getppid() ) ;
             printf(" \t Je vais dormir 30 secondes ...\n") ;
             sleep (30) ;
             printf("\t Je me reveille et je termine mon execution par un EXIT(7)\n") ;
             exit (7) ;
    case -1 : printf ("Le fork a echoue") ;          exit(2) ;
    default : printf ("\t Proc pere de pid %d et Fils de pid %d \n", getpid(), valeur) ;
             printf ("\t J'attends la fin de mon fils...\n") ;
             ret_fils = wait (&etat) ;
             printf ("\t Mon fils de pid %d a termine,\n Son etat etait : %0x\n", ret_fils, etat) ;
  }
  return 0;}
```

Proc fils de pid 5052 et du Pere de pid 5051

Je vais dormir 30 secondes ...

Proc pere de pid 5051 et Fils de pid 5052

J'attends la fin de mon fils...

Je me reveille et je termine mon execution par un EXIT(7)

Mon fils de pid 5052 a termine,

Son etat etait : 700

La valeur de retour du  
fils qui a terminé son  
exécution est stockée  
dans l'argument du  
wait

# Exercice 2: fork

76

**Cas 1**

Cas 2

Cas 3

Cas 4

Cas 5

Cas 6

Déterminer l'arborescence et l'output du programme suivant :

Programme	Arborescence	Output
<pre>int main( ) { if (fork())     if (fork())         if (fork())      ; //         else             printf("Fils 3\n");     else         printf("Fils 2\n"); else     printf("Fils 1\n"); return (0); }</pre>		

# Exercise 2: fork

77

Cas 1

Cas 2

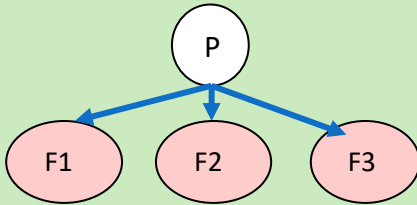
Cas 3

Cas 4

Cas 5

Cas 6

Arborescence

Programme	Arborescence	Output
<pre>int main( ) { if (fork())     if (fork())         if (fork())      ; //         else             printf("Fils 3\n");     else         printf("Fils 2\n"); else     printf("Fils 1\n"); return (0); }</pre>	 <pre>graph TD     P((P)) --&gt; F1((F1))     P --&gt; F2((F2))     P --&gt; F3((F3))</pre>	

# Exercise 2: fork

78

Cas 1

Cas 2

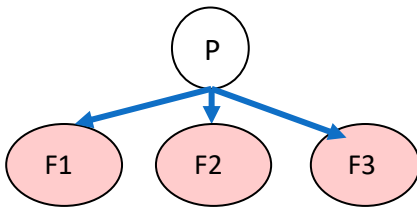
Cas 3

Cas 4

Cas 5

Cas 6

Output

Programme	Arborescence	Output
<pre>int main( ) { if (fork())     if (fork())         if (fork())      ; //         else             printf("Fils 3\n");     else         printf("Fils 2\n"); else     printf("Fils 1\n"); return (0); }</pre>	 <pre>graph TD     P((P)) --&gt; F1((F1))     P --&gt; F2((F2))     P --&gt; F3((F3))</pre>	<p>Fils 1 Fils 2 Fils 3</p>

# Exercice 2: fork

79

Cas 1

**Cas 2**

Cas 3

Cas 4

Cas 5

Cas 6

Déterminer l'arborescence et l'output du programme suivant :

Programme	Arborescence	Output
<pre>#include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main () { printf("Bonjour\n"); if (fork() &amp;&amp; fork())     printf("Monsieur\n"); else     printf("Madame\n");  return 0; }</pre>		

# Exercise 2: fork

80

Cas 1

**Cas 2**

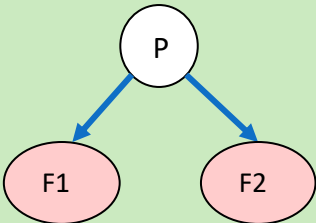
Cas 3

Cas 4

Cas 5

Cas 6

Arborescence

Programme	Arborescence	Output
<pre>#include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main () { printf("Bonjour\n"); if (fork() &amp;&amp; fork())     printf("Monsieur\n"); else     printf("Madame\n");  return 0; }</pre>	 <pre>graph TD     P((P)) --&gt; F1([F1])     P --&gt; F2([F2])</pre>	



# Exercise 2: fork

81

Cas 1

**Cas 2**

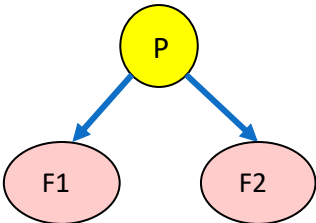
Cas 3

Cas 4

Cas 5

Cas 6

Output

Programme	Arborescence	Output
<pre>#include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main () { printf("Bonjour\n"); if (fork() &amp;&amp; fork())     printf("Monsieur\n"); else     printf("Madame\n");  return 0; }</pre>	 <pre>graph TD     P((P)) --&gt; F1([F1])     P --&gt; F2([F2])</pre>	<p>Bonjour Monsieur</p>

# Exercise 2: fork

82

Cas 1

**Cas 2**

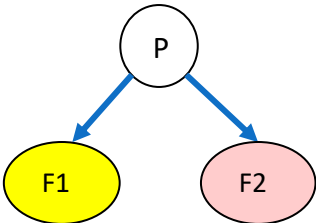
Cas 3

Cas 4

Cas 5

Cas 6

Output

Programme	Arborescence	Output
<pre>#include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main () { printf("Bonjour\n"); if (fork()&amp;&amp;fork())     printf("Monsieur\n"); else     printf("Madame\n");  return 0; }</pre>	 <pre>graph TD     P((P)) --&gt; F1((F1))     P --&gt; F2((F2))</pre>	Bonjour Monsieur Madame

# Exercise 2: fork

83

Cas 1

**Cas 2**

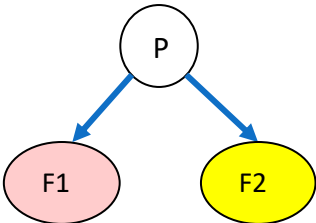
Cas 3

Cas 4

Cas 5

Cas 6

Output

Programme	Arborescence	Output
<pre>#include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main () { printf("Bonjour\n"); if (fork() &amp;&amp; fork())     printf("Monsieur\n"); else     printf("Madame\n");  return 0; }</pre>	 <pre>graph TD     P((P)) --&gt; F1((F1))     P --&gt; F2((F2))</pre>	<p>Bonjour Monsieur Madame <b>Madame</b></p>

# Exercice 2: fork

84

Cas 1

Cas 2

**Cas 3**

Cas 4

Cas 5

Cas 6

Déterminer l'arborescence et l'output du programme suivant :

Programme	Arborescence	Output
<pre>#include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main () { int i;  for (i=1;i&lt;4;i++) if (!fork()) printf("Prog Sys:%d\n", i); return 0; }</pre>		

# Exercise 2: fork

85

Cas 1

Cas 2

Cas 3

Cas 4

Cas 5

Cas 6

## Arborescence

Programme	Arborescence	Output
<pre>#include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main () { int i;  for (i=1;i&lt;4;i++) if (!fork()) printf("Prog Sys:%d\n", i); return 0; }</pre>		

# Exercise 2: fork

86

Cas 1

Cas 2

Cas 3

Cas 4

Cas 5

Cas 6

Output

Programme	Arborescence	Output
<pre>#include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main () { int i;  for (i=1;i&lt;4;i++) if (!fork()) printf("Prog Sys:%d\n", i); return 0; }</pre>		<p>Prog Sys :1</p>

# Exercise 2: fork

87

Cas 1

Cas 2

Cas 3

Cas 4

Cas 5

Cas 6

Output

Programme	Arborescence	Output
<pre>#include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main () { int i;  for (i=1;i&lt;4;i++) if (!fork()) printf("Prog Sys:%d\n", i); return 0; }</pre>		<p>Prog Sys :1 Prog Sys :2</p>

# Exercise 2: fork

88

Cas 1

Cas 2

**Cas 3**

Cas 4

Cas 5

Cas 6

Output

Programme	Arborescence	Output
<pre>#include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main () { int i;  for (i=1;i&lt;4;i++) if (!fork()) printf("Prog Sys:%d\n", i); return 0; }</pre>		<pre>Prog Sys :1 Prog Sys :2 Prog Sys :3</pre>



# Exercise 2: fork

89

Cas 1

Cas 2

**Cas 3**

Cas 4

Cas 5

Cas 6

Output

Programme	Arborescence	Output
<pre>#include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main () { int i;  for (i=1;i&lt;4;i++) if (!fork()) printf("Prog Sys:%d\n", i); return 0; }</pre>		<pre>Prog Sys :1 Prog Sys :2 Prog Sys :3 Prog Sys :2</pre>

# Exercise 2: fork

90

Cas 1

Cas 2

Cas 3

Cas 4

Cas 5

Cas 6

## Output

Programme	Arborescence	Output
<pre>#include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main () { int i;  for (i=1;i&lt;4;i++) if (!fork()) printf("Prog Sys:%d\n", i); return 0; }</pre>		<pre>Prog Sys :1 Prog Sys :2 Prog Sys :3 Prog Sys :2 Prog Sys :3</pre>

# Exercise 2: fork

91

Cas 1

Cas 2

**Cas 3**

Cas 4

Cas 5

Cas 6

## Output

Programme	Arborescence	Output
<pre>#include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main () { int i;  for (i=1;i&lt;4;i++) if (!fork()) printf("Prog Sys:%d\n", i); return 0; }</pre>	<pre>graph TD     P((P)) --&gt; F1([F1])     P --&gt; F2([F2])     P --&gt; F3([F3])     F1 --&gt; F12([F12])     F1 --&gt; F13([F13])     F12 --&gt; F123([F123])     F2 --&gt; F23([F23])     F3 --&gt; None</pre>	<pre>Prog Sys :1 Prog Sys :2 Prog Sys :3 Prog Sys :2 Prog Sys :3 <b>Prog Sys :3</b></pre>

# Exercise 2: fork

92

Cas 1

Cas 2

Cas 3

Cas 4

Cas 5

Cas 6

## Output

Programme	Arborescence	Output
<pre>#include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main () { int i;  for (i=1;i&lt;4;i++) if (!fork()) printf("Prog Sys:%d\n", i); return 0; }</pre>	<pre>graph TD     P((P)) --&gt; F1((F1))     P --&gt; F2((F2))     P --&gt; F3((F3))     F1 --&gt; F12((F12))     F1 --&gt; F13((F13))     F2 --&gt; F23((F23))     F12 --&gt; F123((F123))</pre>	<pre>Prog Sys :1 Prog Sys :2 Prog Sys :3 Prog Sys :2 Prog Sys :3 Prog Sys :3 Prog Sys :3</pre>

# Exercice 2: fork

93

Cas 1

Cas 2

Cas 3

Cas 4

Cas 5

Cas 6

Déterminer l'arborescence et l'output du programme suivant :

Programme	Arborescence	Output
<pre>#include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main () { int i=1; while (i&lt;4) { if (fork()) printf("Step : %d \n", i); i++; } return 0; }</pre>		

# Exercise 2: fork

94

Cas 1

Cas 2

Cas 3

Cas 4

Cas 5

Cas 6

## Arborescence

Programme	Arborescence	Output
<pre>#include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main () { int i=1; while (i&lt;4) { if (fork()) printf("Step : %d \n", i); i++; } return 0; }</pre>	<pre>graph TD     P((P)) --&gt; F1((F1))     P --&gt; F2((F2))     P --&gt; F3((F3))     F1 --&gt; F12((F12))     F1 --&gt; F13((F13))     F2 --&gt; F23((F23))     F12 --&gt; F123((F123))</pre>	

# Exercise 2: fork

95

Cas 1

Cas 2

Cas 3

Cas 4

Cas 5

Cas 6

Output

Programme	Arborescence	Output
<pre>#include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main () { int i=1; while (i&lt;4) { if (fork()) printf("Step : %d \n", i); i++; } return 0; }</pre>		<p>Step : 1 Step : 2 Step : 3</p>

# Exercise 2: fork

96

Cas 1

Cas 2

Cas 3

**Cas 4**

Cas 5

Cas 6

Output

Programme	Arborescence	Output
<pre>#include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main () { int i=1; while (i&lt;4) { if (fork()) printf("Step : %d \n", i); i++; } return 0; }</pre>		Step : 1 Step : 2 Step : 3 <b>Step : 2</b> <b>Step : 3</b>



# Exercise 2: fork

97

Cas 1

Cas 2

Cas 3

Cas 4

Cas 5

Cas 6

Output

Programme	Arborescence	Output
<pre>#include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main () { int i=1; while (i&lt;4) { if (fork()) printf("Step : %d \n", i); i++; } return 0; }</pre>		Step : 1 Step : 2 Step : 3 Step : 2 Step : 3 <b>Step : 3</b>

# Exercise 2: fork

98

Cas 1

Cas 2

Cas 3

**Cas 4**

Cas 5

Cas 6

Output

Programme	Arborescence	Output
<pre>#include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main () { int i=1; while (i&lt;4) { if (fork()) printf("Step : %d \n", i); i++; } return 0; }</pre>	<pre>graph TD     P((P)) --&gt; F1((F1))     P --&gt; F2((F2))     P --&gt; F3((F3))     F1 --&gt; F12((F12))     F1 --&gt; F13((F13))     F2 --&gt; F23((F23))     F12 --&gt; F123((F123))</pre>	Step : 1 Step : 2 Step : 3 Step : 2 Step : 3 Step : 3 <b>Step : 3</b>

# Exercice 2: fork

99

Cas 1

Cas 2

Cas 3

Cas 4

Cas 5

Cas 6

Déterminer l'arborescence et l'output du programme suivant :

Programme	Arborescence	Output
<pre>#include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main( ) { int i, p;  for (i=1; i&lt;=4; i++) { p = fork( );   if (p) printf("%d\n", i); } return (0); }</pre>		

# Exercise 2: fork

100

Cas 1

Cas 2

Cas 3

Cas 4

Cas 5

Cas 6

## Arborescence

Programme	Arborescence	Output
<pre>#include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main( ) { int i, p;  for (i=1; i&lt;=4; i++) { p = fork( );   if (p) printf("%d\n", i); } return (0); }</pre>	<pre>graph TD     P((P)) --&gt; F1((F1))     P --&gt; F2((F2))     P --&gt; F3((F3))     P --&gt; F4((F4))     F1 --&gt; F12((F12))     F1 --&gt; F13((F13))     F1 --&gt; F14((F14))     F2 --&gt; F23((F23))     F2 --&gt; F24((F24))     F3 --&gt; F34((F34))     F4 --&gt;      F12 --&gt; F123((F123))     F12 --&gt; F124((F124))     F13 --&gt; F134((F134))     F14 --&gt;      F123 --&gt; F1234((F1234))     F23 --&gt; F234((F234))     F24 --&gt;      F34 --&gt;      style P fill:#fff,stroke:#000     style F1 fill:#f99,stroke:#000     style F2 fill:#f99,stroke:#000     style F3 fill:#f99,stroke:#000     style F4 fill:#f99,stroke:#000     style F12 fill:#9cf,stroke:#000     style F13 fill:#9cf,stroke:#000     style F14 fill:#9cf,stroke:#000     style F23 fill:#9cf,stroke:#000     style F24 fill:#9cf,stroke:#000     style F34 fill:#9cf,stroke:#000     style F123 fill:#f9c,stroke:#000     style F124 fill:#f9c,stroke:#000     style F134 fill:#f9c,stroke:#000     style F234 fill:#f9c,stroke:#000     style F1234 fill:#fff,stroke:#000</pre>	

# Exercise 2: fork

101

Cas 1

Cas 2

Cas 3

Cas 4

Cas 5

Cas 6

Output

Programme	Arborescence	Output
<pre>#include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main( ) {   int i, p;    for (i=1; i&lt;=4; i++)   { p = fork( );     if (p) printf("%d\n", i);   }   return (0); }</pre>	<pre>graph TD     P((P)) --&gt; F1((F1))     P --&gt; F2((F2))     P --&gt; F4((F4))     F1 --&gt; F12((F12))     F1 --&gt; F13((F13))     F1 --&gt; F14((F14))     F12 --&gt; F123((F123))     F12 --&gt; F124((F124))     F123 --&gt; F1234((F1234))     F2 --&gt; F23((F23))     F2 --&gt; F24((F24))     F23 --&gt; F234((F234))     F3((F3)) --&gt; F34((F34))     F4 --&gt; None</pre>	<div>1</div> <div>2</div> <div>3</div> <div>4</div>

# Exercise 2: fork

102

Cas 1

Cas 2

Cas 3

Cas 4

Cas 5

Cas 6

Output

Programme	Arborescence	Output
<pre>#include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main( ) {   int i, p;    for (i=1; i&lt;=4; i++)   { p = fork( );     if (p) printf("%d\n", i);   }   return (0); }</pre>	<pre>graph TD   P((P)) --&gt; F1((F1))   P --&gt; F2((F2))   P --&gt; F3((F3))   F1 --&gt; F12((F12))   F1 --&gt; F13((F13))   F1 --&gt; F14((F14))   F2 --&gt; F23((F23))   F2 --&gt; F24((F24))   F3 --&gt; F34((F34))   F12 --&gt; F123((F123))   F12 --&gt; F124((F124))   F13 --&gt; F134((F134))   F23 --&gt; F234((F234))   F123 --&gt; F1234((F1234))   F14 --&gt;    F24 --&gt;    F34 --&gt;    F234 --&gt;    F1234 --&gt; </pre>	<pre>1 2 3 4 2 3 4</pre>

# Exercise 2: fork

103

Cas 1

Cas 2

Cas 3

Cas 4

**Cas 5**

Cas 6

Output

Programme	Arborescence	Output
<pre> #include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main( ) { int i, p;  for (i=1; i&lt;=4; i++) { p = fork( );   if (p) printf("%d\n", i); } return (0); } </pre>		<pre> 1 2 3 4 2 3 4 3 4 </pre>

# Exercise 2: fork

104

Cas 1

Cas 2

Cas 3

Cas 4

**Cas 5**

Cas 6

Output

Programme	Arborescence	Output
<pre> #include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main( ) { int i, p;  for (i=1; i&lt;=4; i++) { p = fork( );   if (p) printf("%d\n", i); } return (0); } </pre>		<pre> 1 2 3 4 2 3 4 3 4 4 4 4 </pre>



# Exercise 2: fork

105

Cas 1

Cas 2

Cas 3

Cas 4

**Cas 5**

Cas 6

Output

Programme	Arborescence	Output
<pre> #include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main( ) { int i, p;  for (i=1; i&lt;=4; i++) { p = fork( );   if (p) printf("%d\n", i); } return (0); } </pre>	<pre> graph TD     P((P)) --&gt; F1((F1))     P --&gt; F2((F2))     P --&gt; F3((F3))     P --&gt; F4((F4))     F1 --&gt; F12((F12))     F1 --&gt; F13((F13))     F1 --&gt; F14((F14))     F12 --&gt; F123((F123))     F12 --&gt; F124((F124))     F123 --&gt; F1234((F1234))     F2 --&gt; F23((F23))     F2 --&gt; F24((F24))     F23 --&gt; F234((F234))     F3 --&gt; F34((F34))     F4 --&gt;  </pre>	<pre> 1 2 3 4 2 3 4 3 4 4 3 4 </pre>

# Exercise 2: fork

106

Cas 1

Cas 2

Cas 3

Cas 4

**Cas 5**

Cas 6

Output

Programme	Arborescence	Output
<pre> #include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main( ) { int i, p;  for (i=1; i&lt;=4; i++) { p = fork( );   if (p) printf("%d\n", i); } return (0); } </pre>		<pre> 1 2 3 4 2 3 4 3 4 4 3 4 4 </pre>

# Exercise 2: fork

107

Cas 1

Cas 2

Cas 3

Cas 4

**Cas 5**

Cas 6

Output

Programme	Arborescence	Output
<pre> #include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main( ) { int i, p;  for (i=1; i&lt;=4; i++) { p = fork( );   if (p) printf("%d\n", i); } return (0); } </pre>	<pre> graph TD     P((P)) --&gt; F1((F1))     P --&gt; F2((F2))     P --&gt; F4((F4))     F1 --&gt; F12((F12))     F1 --&gt; F13((F13))     F1 --&gt; F14((F14))     F12 --&gt; F123((F123))     F12 --&gt; F124((F124))     F13 --&gt; F134((F134))     F123 --&gt; F1234((F1234))     F2 --&gt; F23((F23))     F2 --&gt; F24((F24))     F23 --&gt; F234((F234))     F3((F3)) --&gt; F34((F34))     F4 --&gt;  </pre>	<pre> 1 2 3 4 2 3 4 3 4 4 3 4 4 4 4 </pre>

# Exercise 2: fork

108

Cas 1

Cas 2

Cas 3

Cas 4

**Cas 5**

Cas 6

Output

Programme	Arborescence	Output
<pre> #include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main( ) { int i, p;  for (i=1; i&lt;=4; i++) { p = fork( );   if (p) printf("%d\n", i); } return (0); } </pre>	<pre> graph TD     P((P)) --&gt; F1((F1))     P --&gt; F2((F2))     P --&gt; F3((F3))     P --&gt; F4((F4))     F1 --&gt; F12((F12))     F1 --&gt; F13((F13))     F1 --&gt; F14((F14))     F12 --&gt; F123((F123))     F12 --&gt; F124((F124))     F13 --&gt; F134((F134))     F2 --&gt; F23((F23))     F2 --&gt; F24((F24))     F23 --&gt; F234((F234))     F3 --&gt; F34((F34))     F4 --&gt;      F123 --&gt; F1234((F1234)) </pre>	<pre> 1 2 3 4 2 3 4 3 4 4 3 4 4 4 4 4 </pre>

# Exercice 2: fork

109

Cas 1

Cas 2

Cas 3

Cas 4

Cas 5

Cas 6

Déterminer l'arborescence et l'output du programme suivant :

Programme	Arborescence	Output
<pre>#include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main ( ) { fork(); printf ("Prog Sys \n") ; fork(); execlp("ls","ls","-l",NULL); fork(); printf ("Réseaux\n") ; return (0); }</pre>		

# Exercise 2: fork

110

Cas 1

Cas 2

Cas 3

Cas 4

Cas 5

Cas 6

## Arborescence

Programme	Arborescence	Output
<pre>#include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main ( ) { fork(); printf ("Prog Sys \n") ; fork(); execlp("ls","ls","-l",NULL); fork(); printf ("Réseaux\n") ; return (0); }</pre>	<pre>graph TD     P((P)) --&gt; F1((F1))     P --&gt; F2((F2))     F1 --&gt; F12((F12))</pre>	

# Exercice 2: fork

111

Cas 1

Cas 2

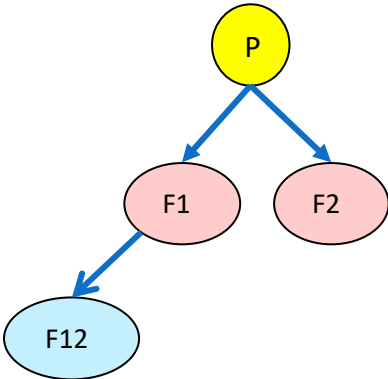
Cas 3

Cas 4

Cas 5

Cas 6

Output

Programme	Arborescence	Output
<pre>#include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main ( ) { fork(); printf ("Prog Sys \n") ; fork(); execlp("ls","ls","-l",NULL); fork(); printf ("Réseaux\n") ; return (0); }</pre>	 <pre>graph TD     P((P)) --&gt; F1([F1])     P --&gt; F2([F2])     F1 --&gt; F12([F12])</pre>	<pre>Prog Sys // exécution de ls</pre>

# Exercice 2: fork

112

Cas 1

Cas 2

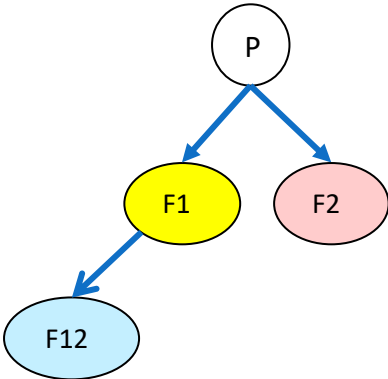
Cas 3

Cas 4

Cas 5

Cas 6

## Output

Programme	Arborescence	Output
<pre>#include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main ( ) { fork(); printf ("Prog Sys \n") ; fork(); execlp("ls","ls","-l",NULL); fork(); printf ("Réseaux\n") ; return (0); }</pre>	 <pre>graph TD     P((P)) --&gt; F1((F1))     P --&gt; F2((F2))     F1 --&gt; F12((F12))</pre>	<pre>Prog Sys // exécution de ls Prog Sys // exécution de ls</pre>



# Exercice 2: fork

113

Cas 1

Cas 2

Cas 3

Cas 4

Cas 5

Cas 6

## Output

Programme	Arborescence	Output
<pre>#include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main ( ) { fork(); printf ("Prog Sys \n") ; fork(); execlp("ls","ls","-l",NULL); fork(); printf ("Réseaux\n") ; return (0); }</pre>	<pre>graph TD     P((P)) --&gt; F1((F1))     P --&gt; F2((F2))     F1 --&gt; F12((F12))</pre>	<pre>Prog Sys // exécution de ls Prog Sys // exécution de ls // exécution de ls</pre>

# Exercice 2: fork

114

Cas 1

Cas 2

Cas 3

Cas 4

Cas 5

Cas 6

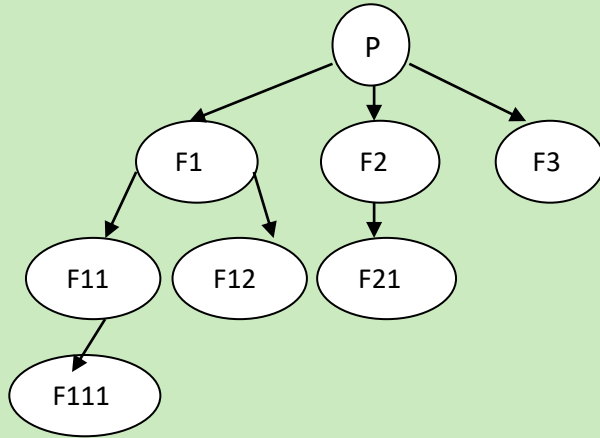
## Output

Programme	Arborescence	Output
<pre>#include &lt;stdio.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; int main ( ) { fork(); printf ("Prog Sys \n") ; fork(); execlp("ls","ls","-l",NULL); fork(); printf ("Réseaux\n") ; return (0); }</pre>	<pre>graph TD     P((P)) --&gt; F1((F1))     P --&gt; F2((F2))     F1 --&gt; F12((F12))</pre>	<pre>Prog Sys // exécution de ls Prog Sys // exécution de ls // exécution de ls // exécution de ls</pre>

## Exercise 3: fork

115

Donner le code qui permet de créer cette arborescence :



# Programme

```
int main ()
```

 $\{$ 

.....

.....

.....

.....

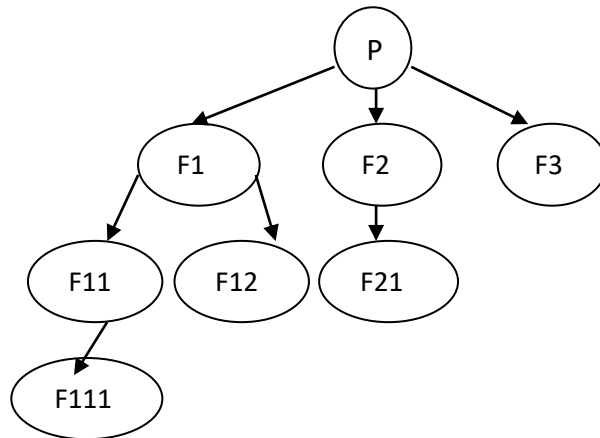
---

```
return 0;
```

}

## 116

# Arborescence



```
int main ( )
```

 $\{$ 

```
for (i=1; i<=3; i++)
    fork();
```

```
return 0;
}
```