

Les tubes :

les communications entre processus peuvent avoir lieu en utilisant différents mécanismes :

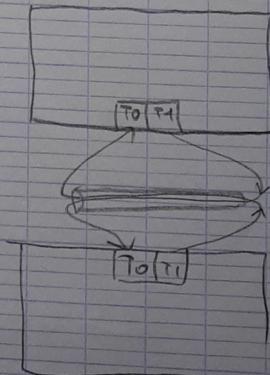
- argument sur ligne de commande
- parenté
- fichiers
- signaux
- tubes

l'appel système qui permet de créer un tube :

```
int pipe(int T[2])
```

→ pipe crée un tube de communication entre les descripteurs ~~T[0] et T[1]~~
T[0] et T[1] les données écrites sur T[1] sont accessibles en lecture sur T[0]

```
int main (...){  
    int T[2];  
    pipe (&T[0]);  
    if (fork ()==0) {  
        close (T[1]);  
        exit (0);  
    } else {  
        close (T[0]);  
        exit (1);  
    }
```



Exo:

Ecrire un programme constitué de 2 processus et d'un tube. Le processus père envoie un entier au processus fils. Le processus fils affiche l'entier reçus

```

int main (...) {
    int T[2];
    int a, rep;
    pipe (&T[0]);
    param[0] = "Entrez une val";
    scanf ("%d", &a);
    if (fork() == 0) {
        close (T[1]);
        read (T[0], &rep, sizeof (int));
        printf ("%d\n", rep);
        exit (0);
    } else {
        close (T[0]);
        scanf ("%d", &a);
        write (*T[1], &a, sizeof (int));
        wait (0);
        close (T[1]);
    }
    exit (0);
}

```

▷ faux fermement
du tube avant
que le processus
fils lire => le
faire après + wait

Exo:

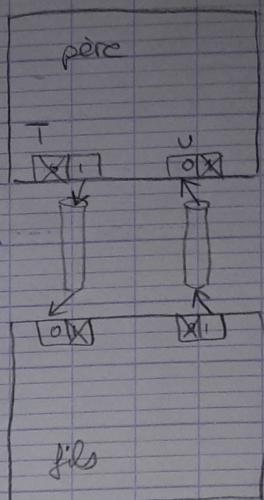
Pareil mais le processus fils multiplie par 2 l'entier et le renvoie au père

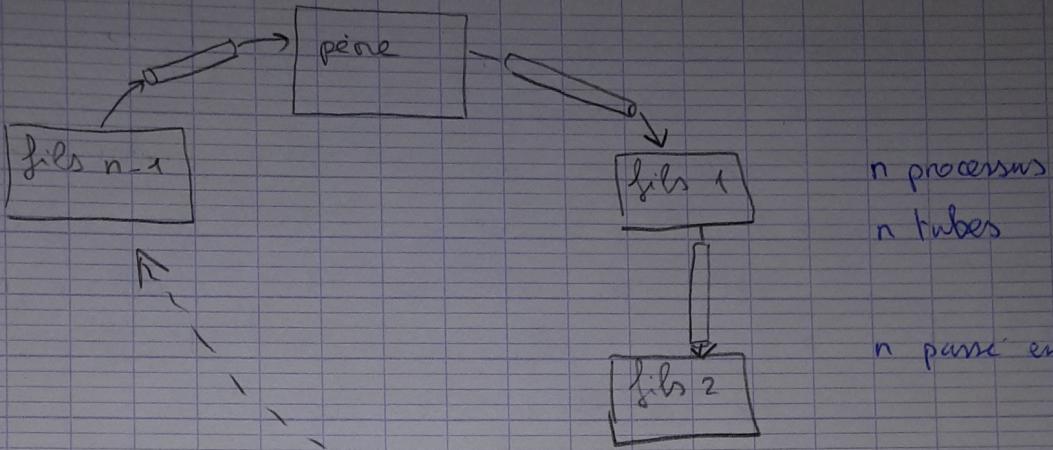
▷ il faut 2 tubes

```

int main (...) {
    int a, rep;
    int T[2], U[2];
    pipe (&T[0]); pipe (&U[0]);
    if (fork() == 0) {
        close (T[1]);
        close (U[0]);
        read (T[0], &rep, sizeof(int));
        rep *= 2;
        write (U[1], &rep, sizeof(int));
        close (T[0]); close (U[1]);
        exit(0);
    } else {
        close (T[0]);
        scanf ("y.d", &a);
        write (T[1], &a, sizeof(int));
        writ
        read (U[0], &a, sizeof(int));
        printf (...);
        wait();
        close (T[1]); close (U[0]);
    }
    exit(0);
}

```





```

int main ( int argc, char * argv [ ] ) {
    int n = atoi ( argv [ 1 ] );
    int in, out, c = 0;
    int * tube = ( int * ) malloc ( 2 * n * sizeof ( int ) );
    for ( int i = 0; i < n; i++ ) {
        pipe ( & tube [ 2 * i ] );
    }
}

```

```

for ( i = 0; i < n; i++ ) {
    if ( fork () == 0 ) {
        in = tube [ 2 * i ];
        out = tube [ 2 * i + 1 ];
        read ( in, &c, sizeof ( int ) );
        write ( out, &c, sizeof ( int ) );
        exit ( 0 );
    }
}

```

```

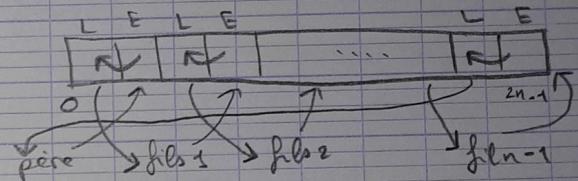
} else {
    in = tube [ 2 * n - 2 ];
    out = tube [ n ];
    write ( out, &c, sizeof ( int ) );
    read ( in, &c, sizeof ( int ) );
}

```

```

}
exit ( 0 );
}

```



fait
fork +
exec +
pipe
a l'intérieur
il est conseillé

utilisation du tube vers des programmes est :

- File * popen (const char * cmd, const char * t)
- ouvrir un tube avec le programme cmd
- t = "t" la sortie standard du programme appelle est redirigée sur descripteur retourné
- t = "w" l'entrée standard du programme est le descripteur renvoyé

Exo :

Line le nb de caractères que retourne un ls

```
File * p = popen ("ls", "r");
while (!feof (cp)) {
    fscanf (p, "%c", &p);
    i++;
}
printf ("%d", i);
```

Puis on peut créer des tubes dont les processus l'utilisant n'ont aucun lieu de parenté connu.

On utilise des fichiers spéciaux qu'on appelle Fifo

On crée une entrée dans le système de fichier avec la commande :

```
int mkfifo (const char * path, mode_t mode)
```

- ↳ crée un fichier type tube à l'emplacement path avec les droits mode
- ↳ pour accéder à ce tube, on utilise open, close, read, write.

Exemple: écrire 3 processus :

- 1- crée le tube
- 2- écrire 2 entiers sur le tube
- 3- lire et afficher les 2 entiers sur le tube.



```
1- int main (int argc, char * argv [3]) {
    if (mkfifo (argv[1], 700) == -1) {
        printf ("erreur à la création du tube\n");
    }
    exit(0);
}
```

```
2- int main (int argc, char * argv [3]) {
    int tube;
    int a, b;
    tube.open(argv[1], O_WRONLY, 0);
    scanf ("%d", &a);
    scanf ("%d", &b);
    write (tube, &a, sizeof(int));
    write (tube, &b, sizeof(int));
    close (tube);
    exit(0);
}
```

```
3- int main (int argc, char * argv [3]) {
    int z, x, tube;
    tube (argv[1], O_RDONLY, 0);
    =open
    read (tube, &z, sizeof(int));
    read (tube, &x, sizeof(int));
    printf (... );
    close (tube);
    unlink(argv[1]);
    exit(0);
}
```