



Unité d'enseignement Systèmes d'exploitation centralisés

1ère année Sciences du Numérique

#### TD : Redirections et tubes

11 avril 2018

#### Thèmes traités

- redirections : principe, API Unix ;
- processus communiquant par tubes : architecture, protocole d'usage, API Unix ;
- exercice de synthèse : parallélisme, communication par tubes et signaux.

### 1 Déroulement

- duplication (dup, dup2), mise en œuvre des redirections
- présentation : planche 17 du support d'accompagnement des TD (Fichiers)
- exercice 3.4.2 (en utilisant la commande standard cat) du poly API Unix
- tubes
  - présentation, protocole d'usage : planche 18
  - exemple de réalisation du schéma producteur-consommateur : exercice 3.2.2
  - réalisation d'un pipeline sur des filtres standard : exercice 3.4.3 (who|grep|wc) (architecture et algorithmique uniquement)
  - présentation (architecture) de l'exercice de synthèse, qui sera corrigé directement et rapidement en début de TD suivant.

### 2 Testez vous

Vous devriez maintenant être en mesure de répondre clairement aux questions suivantes :

- Est-il possible de transmettre des entiers via un tube ?
- Un processus crée un tube, puis lit des données sur ce tube. Que se passe-t-il ?
- Un processus crée un tube, ferme l'entrée du tube, puis lit des données sur ce tube. Que se passe-t-il ?
- Que se passe-t-il si un processus tente d'écrire dans un tube ayant atteint le maximum de sa capacité ?

### 3 Synthèse : processus communiquant par signaux et tubes

On considère le programme suivant :

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <sys/types.h>
4 #include <signal.h>
5 #include <stdlib.h>
6
7 #define NBTESTS 5
8 static int PairImpair = -1;
9 static int Iboucle = 0;
10 static int PrintM1 = 1;
11 static int p[2];
12
13 void H1 (int sig) {
14     if (PrintM1==1) {
15         /* PrintM1 indique simplement s'il s'agit de la premiere activation de H1(_) */
16         printf ("Message\u201d1,\u201dX1=%d\u201d\u201drecu\u201dY1=%d\u201d\n", (int) getpid(), sig);
17         PrintM1 = 0;
18     }
19     else {
20         if (sig==SIGQUIT) {
21             printf ("Message\u201d2,\u201dX2=%d\u201d\u201drecu\u201dY2=%d\u201d\n", (int) getpid(), sig);
22             exit(3);
23         }
24     }
25     if (Iboucle<0) Iboucle=-Iboucle; /* Iboucle peut devenir < 0, en cas de depassement */
26     /* PairImpair=1 si Iboucle est impair et =0 sinon */
27     PairImpair=Iboucle%2;
28     printf ("Iboucle=%d\u201dPairImpair=%d\u201d\n", Iboucle, PairImpair);
29     write(p[1],&PairImpair, sizeof(int));
30     Iboucle=0;
31 }
32
33 void H2 (int sig) {
34     printf ("Message\u201d3,\u201dX3=%d\u201d\u201drecu\u201dY3=%d\u201d\n", (int) getpid(), sig);
35 }
36
37 int main (int argc, char *argv[]) {
38     int pid1, pid2, pid3, ret, i, n;
39     int q[2];
40     float nbtotal, nbpair;
41     float bilan, nbpairmoyen;
42
43     printf ("Message\u201d4,\u201dX4=%d\u201d\u201ddepere\u201dY4=%d\u201d\n", (int) getpid(), getppid());
44
45     pipe(p);
46     pipe(q);
47     signal(SIGQUIT,H1);
48     signal(SIGUSR1,H1);
49     signal(SIGUSR2,H2);
50     pid1=fork();
51     if (pid1 != 0) {
52         printf(".....\u201dpid1=%d\u201d\n", (int) pid1);
53         pid2=fork();
54         if (pid2 != 0) {
55             close(p[1]); /* Ligne 55 */
56             close(q[1]); /* Ligne 56 */
57             close(p[0]);
58     }
```

```

58     printf(".....pid2=%d\n", (int) pid2);
59     for (i=0; i<NBTESTS; i++)
60     {
61         sleep(1);
62         kill(pid1, SIGUSR1);
63     }
64     kill(pid2, SIGUSR2);           /* Ligne 64 */
65     ret=wait(&n);               /* Ligne 65 */
66     if (WIFSIGNALED(n))
67     { printf("Message 5, X5=%d et Y5=%d\n", ret, WTERMSIG(n)); }
68     else
69     { printf("Message 6, X6=%d et Y6=%d\n", ret, WEXITSTATUS(n)); }
70     kill(pid1, SIGINT);          /* Ligne 70 */
71     ret=wait(&n);               /* Ligne 71 */
72     if (WIFSIGNALED(n))
73     { printf("Message 7, X7=%d et Y7=%d\n", ret, WTERMSIG(n)); }
74     else
75     { printf("Message 8, X8=%d et Y8=%d\n", ret, WEXITSTATUS(n)); }
76     bilan=-999;
77     ret=read(q[0], &bilan, sizeof(float));
78     if (ret>0) {bilan = bilan * 2;}
79     printf ("bilan=%f\n", bilan);
80 }
81 else /* pid2 == 0 */           /* Ligne 82 */
82     close(p[1]);
83     close(q[1]);
84     printf ("Message 9, X9=%d de pere Y9=%d\n", (int) getpid(), getppid());
85     pause();
86     execvp("ps", "ps", NULL);
87     ret=wait(&n);             /* Ligne 87*/
88 }
89 }
90 else /* pid1 == 0 */
91     pid3=fork();
92     if (pid3 != 0) {
93         close(q[1]);           /* Ligne 93 */
94         printf(".....pid3=%d\n", (int) pid3);
95         printf ("Message 10, X10=%d de pere Y10=%d\n", (int) getpid(), getppid());
96         for (;;)                /* boucle infinie, Ligne 96*/ {
97             Iboucle++;
98         }
99         /* Ligne 99 */
100    }
101 else /* pid3 == 0 */
102 {
103     close(p[1]);
104     close(q[0]);
105     nbpair = 0; nbtotal=0; nbpairmoyen = -1; nbttotal=0;
106     while ( read (p[0], &i, sizeof(int)) > 0 ) /* Ligne 106*/
107     {
108         nbttotal = nbttotal + 1;
109         if (i==0) { nbpair = nbpair+1; }
110     }
111     if (nbttotal!=0) nbpairmoyen = nbpair / nbttotal;
112     printf("nbpair=%f\n", nbpair);
113     printf("nbpairmoyen=%f\n", nbpairmoyen);
114     write (q[1], &nbpairmoyen, sizeof(float));
115 }
116 }
117
118 return 0;
119 }

```

## Questions

Une exécution du code exam avec NBTESTS=5 donne la sortie suivante (notez que certaines valeurs ont volontairement été remplacées par des ????)

```
Message 4, X4=???? de pere Y4=????  
..... pid1 = 2504  
..... pid2 = 2505  
..... pid3 = 2506  
Message 10, X10=???? de pere Y10=????  
Message 9, X9=???? de pere Y9=????  
Message 1, X1=? a recu Y1=10  
Iboucle=241534614 PairImpair=0  
Iboucle=242681100 PairImpair=0  
Iboucle=485825588 PairImpair=0  
Iboucle=242802438 PairImpair=0  
Message 3, X3=? a recu Y3=?  
Iboucle=485986617 PairImpair=1  
PID TTY TIME CMD  
2130 pts/0 00:00:00 bash  
2503 pts/0 00:00:00 exam  
2504 pts/0 00:00:04 exam  
2505 pts/0 00:00:00 ps  
2506 pts/0 00:00:00 exam  
Message 6, X6=? et Y6=?  
nbpair=4.000000  
nbpairmoyen=0.800000  
Message 7, X7=? et Y7=?  
bilan=1.600000
```

1. Décrire l'architecture de communication entre processus et indiquer/représenter les descripteurs ouverts par processus.
2. Quels sont les descripteurs connectés au processus qui effectue la boucle infinie (Ligne 96). Comment ce processus sort-il de cette boucle ?
3. Compléter et expliquer l'affichage des valeurs  $X_i$  et  $Y_i$  pour  $i=1,10$ . Expliquer en particulier pourquoi les messages 2, 5 et 8 ne sont pas affichés.
4. Expliquer pourquoi dans la sortie
  - (a) on trouve 3 lignes dont le motif est

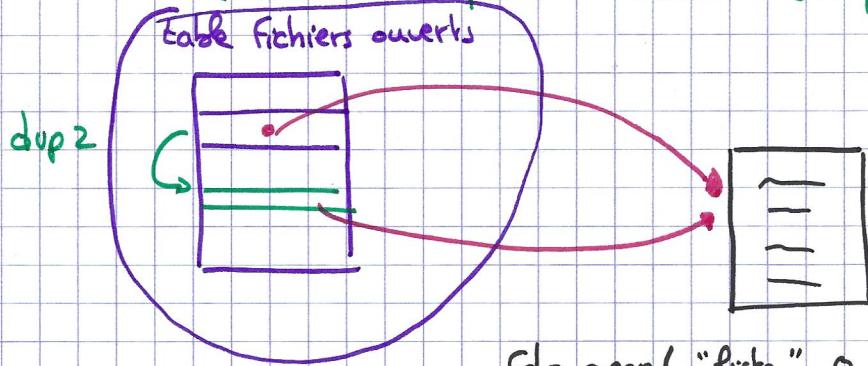
```
.... pts/0 ..... exam ?
```
  - (b) Quel que soit l'ordre de séquencement des processus par le système trouvera-t-on toujours toujours exactement 3 lignes avec

```
.... pts/0 ..... exam ?
```
5. Expliquer comment le processus qui effectue le while en Ligne 106 sort de cette boucle.
6. Si on supprime le close(p[1]) en ligne 82, que se passe-t-il ?
7. Si on supprime le close(q[1]) en ligne 93, que se passe-t-il ?
8. Que se passe t-il si on supprime le close(p[1]) en Ligne 55 ?
9. Que se passe t-il si on déplace close(q[1]) en Ligne 56 après le wait en Ligne 65 ?
10. Si on supprime l'instruction kill(pid2, SIGUSR2) en Ligne 64, que se passe-t-il ?
11. Dans quel cas atteint-on la ligne 87 ?
12. Si on ajoute l'instruction signal(SIGUSR1,SIG\_IGN) avant la boucle infinie ligne 96 que se passe-t-il ? Indiquer la valeur de bilan.
13. Que se passe-t-il si on supprime la ligne 70 (kill(pid1,SIG\_INT)) ?  
<sub>4</sub>
14. Que se passe-t-il si on remplace kill(pid1,SIGINT) en ligne 70 par kill(pid1,SIGQUIT) ? Indiquer notamment l'effet sur l'affichage du message 7.
15. Pour de grandes valeurs de NBTESTS, vers quelle valeur doit tendre la variable bilan ?

TD<sub>4.2</sub> SEC : Redirection et Duplication

`int dup2 (int descriptor_copié, int descriptor_affecté)`

→ affectation explicite entre descripteurs.



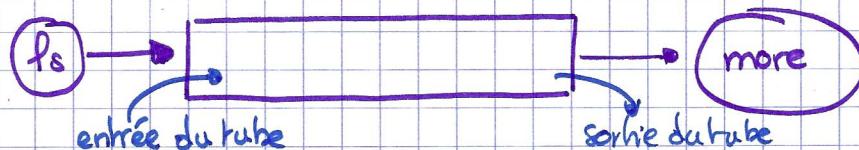
```
fd = open("liste", O_WRONLY)
dup2(fd, 1)
close(fd)
execvp("ls", "ls", "-P", NULL).
```

### Exercice 3.4.2

```
fd1 = open(source, O_RDONLY)
fd2 = open(destination, O_WRONLY)
cat fd1 > fd2
close(fd1)
```

```
fd = open(argv[2], O_WRONLY);
dup2(argv[2], 1);
close(fd)
execvp("cat", "cat", argv[1], NULL)
```

Tube : canal de communication entre processus ayant un même ancêtre.



Accès au tube : 2 descripteurs de fichiers  
entrée → p[1] sortie → p[0].

- 1) Crée tampon
- 2) Crée 2 descripteurs
- 3) Copier les indices des descripteurs dans p.

ls 1 more

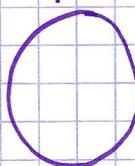


### Exercice 3.2.2

pipe(p)

fork()

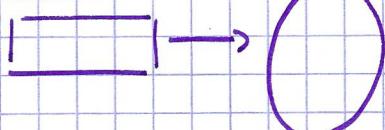
p1



fork() → création père + fils

? pipe(p) → chacun fait un tube

p2



lire et somme → afficher le résultat.  
les entiers lus

main(args, argv[]) {

pid\_t pidF;

int p[2];

pipe(p);

pidF = fork();  
if (pidF == 0) {

close(p[1]);

int somme = 0;

int tampon;

while (read(p[0], &tampon, sizeof(int))) {

somme = somme + tampon;

? printf("%d", somme);

} else {

close(p[0]);

int i;

for (i=0; i< atoi(argv[0]); i++) {

write(p[1], i, sizeof(int));

}

? exit(0); } ..



ls | grep toto | wc -P

créer tube

créer fili

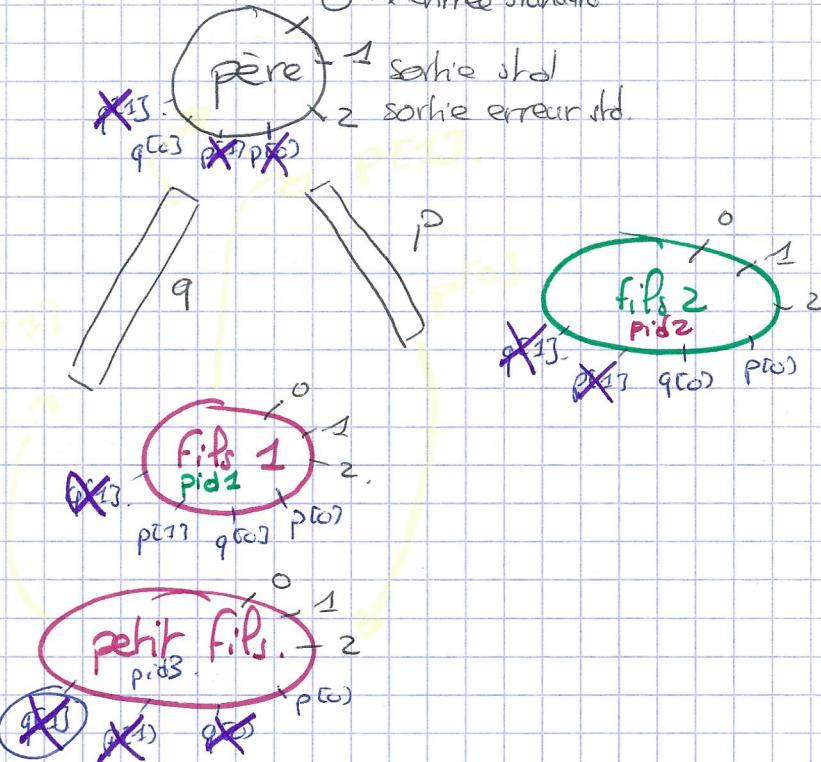
rediriger

fermer les descripteurs inutiles.

Synthèses : processus communiquant par signaux et tubes

$\circ$  → entrée standard

1 /



pid1 := 0 → code du père

2. / C'est le fils 1 qui exécute la boucle inf.

Donc on a 0, 1, 2, p[0], q[0] - p[1].

Sortie bête quand reçoit signal signal

3. / 2 → reçoit jamais SIGQUIT.

5 →

8 → pas affiché car il saute car fils arrêté par SIGINT.

6 a Normal car on a 1 process qui exécute exam. mais un qui exécute ps donc s'appelle ps.

5 read doit renvoyer 0 ou -1. (-1) erreur param

0 → tube fermé en lecture

plus fils sort bête qd père finit écriture. → (si on n't).

6 Rien ne se passe spécial car p[1] fermé à la fin du fils 2

7 p[1] fils écrira son bilan qd fils sera fermé. de qd q[1] sera fermé automatiquement.

8 Pg ne se formera pas . car l'abre p ne fera pas éclat  
il restera le père comme écrit vu la . Donc blocage.

9 → skippee.

10 Père va rester bloqué car le père a besoin de la  
fermation du fils 2 pour avancer.

11 Jamais car exec remplace le reste du code ap lai

12 →

13 Tr le monde se bloque.

14 →

15 Bilan va rendre ver  $2 \times p = 2 \times 0,5 = 1$ .