Tubes, tubes nommées

La rédirection des flots

Exercice 1. Que se passe-t'il pendant l'exécution du programme suivant ?

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/wait.h>

6  int main(void)
7  {
8    int calu, tube[2];

10   pipe(tube);

12   if( fork() ) /* père */
13     {
14       close(STDOUT_FILENO); dup(tube[1]);
15       close(tube[0]); close(tube[1]);

17       while( (calu = getchar()) != '.')
18         {
19           if ( calu >= 'a' && calu <= 'z')
20             calu -= 'a' - 'A';
21           printf("%c",calu);
22         }
23       fflush(stdout);
24       close(STDOUT_FILENO);
25       wait(NULL);
26       exit(EXIT_SUCCESS);
27     }
28   else /* fils */
29     {
30       close(STDIN_FILENO); dup(tube[0]);
31       close(tube[0]); close(tube[1]);

33       while( (calu = getchar()) != EOF)
34         printf("[fils] : >%c<\n", calu);

36       printf("Fin du fils\n");
37       exit(EXIT_SUCCESS);
38     }
39 }
```

Exercice 2. Décrire précisément ce que fait le programme suivant.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>

5  int main(void)
6  {
7    int  tube[2];

9    pipe(tube);

11   if(fork()){ /* père */
12     close(STDIN_FILENO); dup(tube[0]);
13     close(tube[0]); close(tube[1]);

15     execlp("wc","Wordcount","-l", NULL);
16     /* Jaimais atteint sauf erreur */
17     printf("Erreur dans exec wc");
18   }
19   else{ /* fils */
20     close(STDOUT_FILENO); dup(tube[1]);
21     close(tube[0]);close(tube[1]);

23     execlp("ls","ls","-l",NULL);
24     /* Jaimais atteint sauf erreur */
25     printf("Erreur dans exec ls");
26   }

28   exit(EXIT_FAILURE);
29 }
```

Exercice 3. Il y a des erreurs dans le traitement des erreurs du programme précédant. Quelle sont-elles ?

Exercice 4. Écrire un programme qui prend comme paramètre une commande shell et l'exécute en remplaçant les caractères minuscules par des majuscules sur la sortie standard.

Exercice 5. Écrire un programme qui exécute la commande shell `ls -l | grep   \.c$ | wc -l`.

Tubes nommées *vs* tubes anomymes

Exercice 6. Dans le programme de l'exercice 2 remplacer la ligne 9 par

```
mkfifo("letube",0777);
```

Modifier le programme par conséquent.

Exercice 7. Les deux programmes suivants sont-ils équivalents ? Donnez une commande shell permettant de débloquer la version avec un tube nommé.

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


int main(void)
{
  int tube[2];
  char str[10];

  pipe(tube);



  write(tube[1],"bonjour",8);
  read(tube[0],str,8);
  printf("%s\n",str);

  close(tube[0]);close(tube[1]);
  exit(0);
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>

int main(void)
{
  int tube[2];
  char str[10];

  mkfifo("le_tube",0777);
  tube[0]= open("le_tube",O_RDONLY);
  tube[1] = open("le_tube",O_WRONLY);

  write(tube[1],"bonjour",8);
  read(tube[0],str,8);
  printf("%s\n",str);

  close(tube[0]);close(tube[1]);
  exit(0);
}
```

Serveur de mots

Exercice 8 : serveur de mots. Écrire un programme qui crée deux tubes nommés `question` et `answer` ; chaque fois qu'il reçoit un mot sur le tube `question`, il mélange ses lettres et les renvoie sur le tube `answer`. Hypothèse : le serveur ne s'arrête jamais de travailler.

Exercice 9 : arrêt du serveur. Modifier le programme précédent pour que le serveur s'arrête correctement lorsqu'il reçoit le signal `SIGINT`.

Exercice 10 : un client. Écrire un programme qui prend une phrase en argument, envoie tous ses mots au serveur de mots et affiche les mots rendus par le serveur.

Exercice 11 : plusieurs clients. Que se passe-t'il si l'on exécute plusieurs clients se même temps ?