

★ **Exercice 1:** On considère le programme ci-contre :

▷ **Question 1:** Qu'affiche-t-il quand `toto.txt` et `titi.txt` existent tous les deux ?

▷ **Question 2:** Quelle différence si `toto.txt` n'existe pas ? Et si `titi.txt` n'existe pas ?

```

1 #include <stdio.h>
2 #include <fcntl.h> /* O_RDONLY */
3 #include <errno.h>
4 int main(){
5     int fd1, fd2;
6     fd1 = open("toto.txt", O_RDONLY, 0);
7     close(fd1);
8     fd2= open("titi.txt", O_RDONLY, 0);
9     printf("fd2 = %d (errno=%d)\n", fd2, errno);
10    return 0;
11 }

```

★ **Exercice 2:** On suppose que le fichier `toto.txt` contient les six caractères "LAMBDA".

On considère les trois programmes ci-après.

▷ **Question 1:** Qu'affichent chacun de ces programmes ?

Si vous ne le savez pas (ce n'est pas dans le cours), proposez des hypothèses en les justifiant.

Deux open

```

1 #include <stdio.h>
2 #include <fcntl.h>
3 int main() {
4     int fd1, fd2; char c;
5     fd1=open("toto.txt", O_RDONLY, 0);
6     fd2=open("toto.txt", O_RDONLY, 0);
7     read(fd1, &c, 1);
8     read(fd2, &c, 1);
9     printf("c = %c\n", c);
10    return 0;
11 }

```

Après fork

```

1 #include <stdio.h>
2 #include <fcntl.h> /* O_RDONLY */
3 int main(){
4     int fd; char c;
5     fd=open("toto.txt", O_RDONLY, 0);
6     if (fork() == 0) {
7         read(fd, &c, 1);
8         return 0;
9     }
10    wait(NULL);
11    read(fd, &c, 1);
12    printf("c = %c\n", c);
13    return 0;
14 }

```

Après dup

```

1 #include <stdio.h>
2 #include <fcntl.h> /* O_RDONLY */
3 int main(){
4     int fd1, fd2; char c;
5     fd1=open("toto.txt", O_RDONLY, 0);
6     read(fd1, &c, 1);
7     fd2 = dup(fd1);
8     read(fd2, &c, 1);
9     printf("c = %c\n", c);
10    return 0;
11 }

```

★ **Exercice 3:** Redirection et shell.

▷ **Question 1:** Quels sont les appels systèmes faits par le shell lorsque l'on tape `cat < toto` ? (question de cours)

▷ **Question 2:** Si le shell suit bien votre hypothèse de la question précédente, que devrait-il se passer si l'on fait `./mycat < toto` ? (*i.e.*, on remplace `cat` par le programme ci-contre)

▷ **Question 3:** Comment expliquez-vous que l'on obtienne l'erreur « Mauvais descripteur de fichier » à la place ? Modifiez votre réponse à la première question si besoin pour l'expliquer.

mycat.c

```

1 #include <errno.h>
2 int main() {
3     char BUFF[1024];
4     int lu;
5
6     while ((lu=read(3,&BUFF,1024))>0) {
7         int a_ecrire=lu;
8         char *pos=BUFF;
9
10        while (a_ecrire) {
11            int ecrit=write(1,pos,a_ecrire);
12            a_ecrire -= ecrit;
13            pos += ecrit;
14        }
15    }
16    if (lu<0)
17        perror("probleme de lecture");
18 }

```

★ **Exercice 4:** Réimplémenter `popen`.

La fonction `popen` permet de lire la sortie standard d'une commande (ou d'écrire sur son entrée standard) comme s'il s'agissait d'un fichier. Exemple d'ouverture en lecture : `fich = popen("ls -lR", "r");`

Exemple d'ouverture en écriture : `fich = popen("ssh neptune", "w");`

▷ **Question 1:** Implémentez une fonction permettant de lire la sortie d'une commande (ie, en supposant que le second argument est "r").

REMARQUE : `popen()` retourne un `FILE*` utilisable avec `fscanf`. Votre version devra retourner un descripteur pour `read` (c'est plus simple ainsi).

▷ **Question 2:** Que se passe-t-il si l'on ferme les descripteurs retournés par votre fonction avec `close` ? Proposez une solution à ce problème dans une fonction `mypclose`.

★ **Exercice 5: Processus et tubes** Considérez le programme suivant. Il crée une série de processus ouvrant des tubes entre eux.

▷ **Question 1:** Dessinez l'organisation des processus et tubes telle qu'elle est en ligne 26. Il ne vous est pas demandé de dessiner l'historique menant à cette situation (comme à l'exercice précédent), mais bien la situation à l'instant où la ligne 26 est exécutée (en supposant que tous les processus exécutent cette ligne en même temps). Pensez à indiquer quel processus est créé à quel ligne sur votre schéma.

▷ **Question 2:** Décrivez en une phrase ce que font les processus créés par ce programme.

```
7 int main(int argc, char *argv[ ]) {
8   int tubA[2], tubB[2], tubC[2];
9   pipe(tubA); pipe(tubB); pipe(tubC);
10  int n1=0, n2=0;
11  close(0); close(1);
12
13  if ((n1=fork())) {
14    dup2(tubA[0],0);   dup2(tubB[1],1);
15  } else if ((n2=fork())) {
16    dup2(tubB[0],0);   dup2(tubC[1],1);
17  } else {
18    dup2(tubC[0],0);   dup2(tubA[1],1);
19    printf("A");
20    fflush(stdout);
21  }
22  close(tubA[0]); close(tubA[1]);
23  close(tubB[0]); close(tubB[1]);
24  close(tubC[0]); close(tubC[1]);
25
26  /* Dessinez l'état des lieux à ce point */
27
28  char c;
29  while ((read(0,&c,1))>0) { /* Cette boucle ne s'arrête jamais */
30    write(1,&c,1);
31  }
32 }
```