CP 1

ASR2

1 open/fopen

Écrivez deux programmes en C qui écrivent « coucou » dans un fichier « test.txt ». Le premier se servira de `fopen` pour ouvrir le fichier, et le second de `open`

Quelle sont les différence(s) entre la famille de fonctions `fopen`, `fread`, `fwrite`, `fclose` et la famille `open`, `read`, `write`, `close` ?

Peut-on mixer les fonctions des deux familles ? Quand préférer plutôt l'une que l'autre ?

2 Duplication de descripteurs

Y a-t-il une différence entre les codes suivants ? (Expérimentez si besoin.) Expliquez pourquoi.

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

int main() {
  int a=open("test.txt",O_CREAT|O_WRONLY,0644);
  int b=open("test.txt",O_CREAT|O_WRONLY,0644);
  write(a,"coucou\n",7);
  write(b,"coucou\n",7);
  close(a);
  close(b);
  return 0;
}
```

Et :

```
int main() {
  int a=open("test.txt",O_CREAT|O_WRONLY,0644);
  int b=dup(a);
  write(a,"coucou\n",7);
  write(b,"coucou\n",7);
  close(a);
  close(b);
  return 0;
}
```

3 Processus

— Écrivez un programme qui se remplace par « xeyes ».

— Quelle est la sortie du code suivant ? (Devinez, et vérifiez.) Que se passe-t-il si on fait l'`open` après les `fork` ?

```
int main() {
  int a=open("test.txt",O_CREAT|O_WRONLY,0644);
  fork();
  fork();
  write(a,"coucou\n",7);
  close(a);
  return 0;
}
```

— Expliquez ce que fait le code suivant. Quelle est son utilité ?

```
int main() {
  int status;
  if (fork())
    wait(&status);
  else
    if (!fork())
      execlp("xeyes","xeyes",NULL);
  return 0;
}
```

4 Bonus

J'aime bien lancer mes application (même les graphiques) depuis un terminal/shell. Cela permet plus facilement de trouver les fichiers qui seront argument des mes commandes. Exemple :

```
mrao@meshuggah:~$ cd Documents
mrao@meshuggah:~/Documents$ ls
baxter_67_dimer.pdf  baxter_f_model_triangular.pdf  B.pdf  C.pdf
    penta_gardner.pdf  samuel80.pdf
mrao@meshuggah:~/Documents$ evince baxter_67_dimer.pdf
```

Malheureusement, quand on lance l'application, le shell reste (et doit rester) vivant, alors qu'il n'est pas utilisable/utile. Écrivez un programme `launch` en C, qui prend en arguments un exécutable avec une liste d'arguments, puis qui lance l'exécutable avec ses arguments, et rend la main au terminal. Le shell/terminal pourra alors être tué, sans que le nouveau processus ne soit tué à son tour. Dans l'exemple précédent, cela devient :

```
mrao@meshuggah:~/Documents$ launch evince baxter_67_dimer.pdf
mrao@meshuggah:~$
```

Puis, faites que `launch` termine (tue) le shell qui l'a lancé, juste après avoir lancé l'exécutable.