Les entrées/sorties

Préchauffage : les E/S de la bibliothèque standard C

Exercice 1.

qat.c

```
 1 : #include <stdlib.h>
 2 : #include <stdio.h>
 3 :
 4 : int main(int argc, char *argv[])
 5 : {
 6 :   FILE *fic;
 7 :   char tampon[100];
 8 :
 9 :   /*  nombre suffisant d'arguments */
10 :   if (argc <= 1)
11 :     return EXIT_FAILURE;
12 :   printf("Le nom du fichier est : %s\n",argv[1]);
13 :
14 :   fic = fopen(argv[1],"r");
15 :   if(fic == NULL )
16 :     return EXIT_FAILURE;
17 :
18 :   while(fgets(tampon,100,fic) != NULL)
19 :     printf("%s",tampon);
20 :
21 :   fclose(fic);
22 :
23 :   return EXIT_SUCCESS;
24 : }
```

Décrire le comportement du programme `qat.c`. A quelle commande du shell le programme `qat.c` vous fait penser ?

Exercice 2. Écrire une fonction qui crée un nouveau fichier texte à partir de deux autres en entrelaçant les lignes des deux premiers fichiers.

Exercice 3. En utilisant la fonction de la librerie standard C

```
int fseek(FILE *fic, long deplacement, int origine) ;
```

écrire des fonctions `arrierefopen`, `arrieregetc`, `arrierefgets` qui se comportent exactement comme la fonctions `fopen`, `getc`, `fgets`, sauf que elles lisent un fichier de la fin vers son début.
Tester ce fonctions pour écrire un programme `arrierecat` qui lit ligne par la ligne un fichier à partir de sa fin.

Exercice 4. Modifier le programme `qat.c` pour qu'il affiche le nombre de fois où le fichier (texte) a été lu par `qat`. On pourra, par exemple, inclure dans le fichier une ligne contenant le nombre d'ouvertures. (Suggestion : ajouter cette ligne à la fin et se servir des fonctions `arrierefopen`, `arrieregetc`, `arrierefgets`).

Les appels système, les caches

Exercice 5. Analyser le programme `autreqat.c` et le comparer à `qat.c` :

autreqat.c

```
 1 : #include <stdlib.h>
 2 : #include <unistd.h>
 3 : #include <fcntl.h>
 4 :
 5 : int main(int argc, char *argv[])
 6 : {
 7 :   int desc;
 8 :   char tampon[TAILLETAMPON];
 9 :   size_t no_lu;
10 :
11 :   if (argc <= 1)
12 :     return EXIT_FAILURE;
13 :
14 :   if((desc = open(argv[1],O_RDONLY)) == -1 )
15 :     return EXIT_FAILURE;
16 :
17 :   while((no_lu = read(desc,tampon,TAILLETAMPON)) >0)
18 :     write(STDOUT_FILENO,tampon,no_lu);
19 :
20 :   close(desc);
21 :
22 :   return EXIT_SUCCESS;
23 : }
```

Que se passe-t'il pendant la suivante suite de commandes shell :

```
[utilisateur@localhost repcourant]$ ls --block-size=1 -s longfichier.ps
3158016 longfichier.ps
[utilisateur@localhost repcourant]$ for i in 1 2 4 10 50 100 1000 ;do \
echo ; \
gcc -Wall -pedantic -DTAILLETAMPON=$i autreqat.c ; \
time a.out longfichier.ps > /dev/null ;\
done
```

Commenter ce commandes en tenant compte de l'efficacité du programme `autreqat.c`.

Exercice 6. Écrire votre propre implémentations de la structure `FILE` et des fonctions `fopen`, `fprintf`, `fflush`.
À ce fin, on utilisera les appels systèmes POSIX (et non les fonctions de la librerie standard C).

Exercice 7. Dans le code source de chaque (noyau d'un) système d'exploitation on y trouve des fonctions d'entrée/sortie analogues à celle de la bibliothèque C. Par exemple, dans le noyau `linux` on y trouve une fonction appelée `printk`, analogue à `printf`.
Pour quelle raison on n'utilise pas la bibliothèque C et les fonction dont le prototype est défini dans l'en tête `stdio.h` ?

Exercice 8. Considérer le code suivant :

```
...
fptr = fopen("monfichier","w") ;
fprintf(fptr, "Bonjour le monde ! ! !") ;
fflush(fptr) ;
...
```

- Dans l'appel à la fonction de la bibliothèque standard C `fprintf`, combien de caches traverse la chaîne de caractères `"Bonjour le monde"` avant être recopiée sur le disque ?
- Immédiatement après l'appel à la fonction `fflush`, peut-on être sûr que la chaîne de caractères `"Bonjour le monde"` a été recopiée sur le disque ?