Université de Bordeaux — U.F. Informatique
Licence Informatique 3ème année — Année 2014 – 2015

PROGRAMMATION SYSTÈME

Devoir Surveillé

1 heure 20 minutes
Sans documents

N.B. : - Indiquez de façon visible votre numéro de groupe sur votre copie.
- Lisez l'intégralité du sujet avant de commencer à répondre aux questions. Certaines peuvent être traitées même si vous n'avez pas répondu aux précédentes.
- À la fin du sujet, vous trouverez un memento vous rappelant la syntaxe de quelques appels système utiles.
- Les réponses aux questions doivent être argumentées et aussi concises que possible.
- Le barème est donné à titre indicatif.

Question 1 (5 points)

(1.1) (2 points)
Quels sont les trois types de « *bufférisation* » pouvant être associés par le système à un pointeur de flot (« `FILE *` ») ouvert ? Expliquez en quoi ils consistent.
(15 lignes maximum)

(1.2) (1 point)
Quel est le résultat affiché dans la console lorsqu'on lance le `programme` ci-dessous au moyen de la commande « `./programme` » ? Faites bien attention aux caractères de mise en page !

```
1 main ()
2 {
3   printf ("S1");
4   fprintf (stderr, "E1");
5   printf ("S2\n");
6   fprintf (stderr, "E2");
7   printf ("S3");
8   *((int *) NULL) = 42;   /* SEGFAULT */
9 }
```

(1.3) (1 point)
Quel est le contenu du fichier `/tmp/brol.txt` lorsqu'on lance ce programme au moyen de la commande « `./programme > /tmp/brol.txt` » ?

(1.4) (1 point)
On modifie le programme ci-dessus en intercalant la ligne « `fflush (stdout);` » entre les lignes 5 et 6 du programme précédent. Refaites les deux questions précédentes sur la base de ce programme modifié.

Question 2 (8 points)

On dispose des deux fonctions suivantes, issues d'une bibliothèque tierce fournie sous forme objet, et qu'on ne peut donc modifier :

— `producteur (FILE * sortie)` écrit des données sur le flot de sortie qui lui est passé en paramètre ;

— `consommateur (FILE * entrée)` utilise les données qu'elle lit sur le flot d'entrée qui lui est passé en paramètre.

On veut combiner ensemble ces deux fonctions, en un ou plusieurs exemplaires.

(2.1) (1 point)

Une solution triviale pour « brancher » les deux fonctions ensemble consiste à utiliser un tube anonyme, de la façon suivante (la gestion des erreurs a été omise par souci de lisibilité) :

```
1  main ()
2  {
3    int     fd[2];
4    FILE *  fp[2];
5
6    pipe (fd);
7    fp[0] = fdopen (fd[0], "r");
8    fp[1] = fdopen (fd[1], "w");
9    producteur (fp[1]);
10   consommateur (fp[0]);
11   return (EXIT_SUCCESS);
12 }
```

En quoi cette solution est-elle erronée ?

(2.2) (2 points)

Écrivez le code C d'un programme modifié, basé sur l'emploi d'un processus fils, pour remédier au problème précédent.

(2.3) (1 point)

Qui, du producteur ou du consommateur, doit être le processus fils, afin que l'invite de commande n'apparaisse que lorsque toutes les données issues du producteur ont bien été consommées ? Justifiez votre réponse.

(2.4) (3 points)

La production des données est un processus coûteux en temps. Sur un système multiprocesseurs, on veut donc pouvoir lancer n processus producteurs, écrivant sur le même tube anonyme, qui sera lu par un consommateur unique.

Écrivez le code C d'un programme modifié, lancé au moyen de la commande « `./programme` n », qui génère n processus producteurs. Faites bien attention au nombre de processus producteurs lancés par votre code !

(2.5) (1 point)

Dans quel état se trouvent les processus producteurs une fois qu'ils ont terminé, et avant que le processus consommateur n'ait lui-même terminé ? Justifiez votre réponse.

Question 3 (4 points)

On veut créer un programme `verserreur`, qui redirige la sortie standard du programme qui lui est passé en argument vers la sortie d'erreur. Plusieurs versions sont envisagées.

(3.1) (3 points)

Créez une version de `verserreur` qui ne prenne comme seul argument que le nom du programme : « `./verserreur programme` ».

(3.2) (1 point)

Modifiez ce programme afin que le programme qu'il lance puisse disposer d'autant d'arguments qu'il le souhaite : « `./verserreur programme arg1 arg2 ...` ».

Question 4 (3 points)

Écrivez le code C d'un programme `index` qui affiche sur sa sortie standard, en mode texte, l'ensemble des positions (*offset*) auxquelles se trouvent les instances d'un certain caractère au sein d'un fichier. On lance ce programme au moyen de la commande « `./index caractère fichier` ».

Memento

```
int dup2 (int oldfd, int newfd);
int execlp (const char * file, const char * arg, ...);
int execvp (const char * file, char * const argv[]);
FILE * fdopen (int fd, const char * mode);
int fflush (FILE *);
pid_t fork (void);
int fprintf (FILE * stream, const char * format, ...);
size_t fread (void * ptr, size_t size, size_t nitems, FILE * stream);
int fscanf (FILE * stream, char * format, ...);
size_t fwrite (void * ptr, size_t size, size_t nitems, FILE * stream);
off_t lseek (int fd, off_t offset, int whence);
  /* whence = SEEK_SET ou SEEK_CUR ou SEEK_END */
int open (const char * pathname, int flags, mode_t mode);
int pipe (int pipefd[2]);
int printf (const char * format, ...);
ssize_t read (int fd, void * buf, size_t count);
ssize_t write (int fd, const void * buf, size_t count);
```