R. Namyst Jeudi 15 novembre 2018

Licence informatique — Programmation Système (4TIN503U)
Devoir Surveillé

Durée : 1h30 — Sans document

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.*

1 Questions de cours (échauffement)

Question 1 Décrivez le comportement par défaut adapté par un processus lors de la délivrance des signaux suivants : `SIGINT`, `SIGCHLD`, `SIGKILL` et `SIGSEGV`.

Question 2 Les entrées-sorties effectuées au moyen des primitives de la bibliothèque standard (e.g. `fread`, `fwrite`) sont-elles généralement plus performantes que les appels systèmes correspondants (e.g. `read`, `write`), ou est-ce le contraire ? Que cache principalement le type `FILE` ?

2 Fichiers

On souhaite disposer d'un programme `indexer` qui construit, à partir d'un fichier texte, un fichier d'index qui contient les positions de chaque début de ligne du fichier texte. Ces positions sont écrites en format brut dans le fichier. Le programme a donc deux arguments : `indexer <fichier texte> <fichier index>`

Prenons l'exemple d'un fichier texte `exemple.txt` qui contient 10 caractères (dont `'\n'` après `'c'` et après `'g'`) :

```
abc
defg
h
```

L'exécution de `indexer exemple.txt index` doit créer un fichier `index` contenant 3 entiers : 0, 4 et 9 stockés sous format brut (i.e. 12 octets au total).

Question 1 Donnez le code du programme `indexer`.

Question 2 Donnez le code de programme `reverse`, qui prend en arguments le fichier texte et le fichier d'index, et qui imprime sur la sortie standard le contenu du fichier en inversant l'ordre des lignes :

```
[toto@machine-du-cremi] reverse exemple.txt index
h
defg
abc
[toto@machine-du-cremi]
```

L'idée est de se servir du fichier d'index pour réaliser efficacement ce traitement.

3 Tubes

On souhaite disposer d'une fonction `int read_piped_command (char *cmd, char **argv)` qui crée un processus chargé d'exécuter la commande `cmd` avec les arguments `argv`, et renvoie un descripteur de fichier permettant de récupérer (en lecture) la sortie standard de la commande.

Exemple d'appel pour récupérer le résultat de la commande `ls -l` :

```
char * argv[] = { "ls", "-l", NULL };
int fd = read_piped_command ("ls", argv);
```

4 Signaux

On souhaite réaliser une animation à l'écran en rafraichissant la représentation graphique (*sprite*) d'un objet périodiquement. On dispose d'une fonction `display_sprite (int i)` permettant d'afficher la ième image de l'objet. On a écrit une première version du code permettant d'appeler périodiquement cette fonction ci-dessous :

```
#define NB_IMAGES 10
volatile int i = 0;

void handler (int sig)
{
  i = (i + 1) % NB_IMAGES;
  alarm (2);
}

int main ()
{
  sigaction(SIGALRM, handler); // syntaxe volontairement simplifiée

  alarm (2);
  for (;;) {
    display_sprite (i);
    pause ();
  }
}
```

Question 1 Décrivez le comportement de ce programme. Affiche-t-il immédiatement l'image 0, puis les images successives (1, 2, etc.) à raison d'une toute les deux secondes, ou affiche-t-il l'image 0 seulement au bout de deux secondes ?

Question 2 Expliquez précisément ce qui va se produire si l'exécution de la fonction `display_sprite` dure systématiquement un peu plus de deux secondes. À quelle fréquence les images sont-elles affichées ? Le sont-elles toutes ?

Question 3 Modifiez le code précédent pour que, lorsque la durée d'affichage de l'image i excède deux secondes, l'exécution soit interrompue pour démarrer celle de l'image $i + 1$ à temps, c'est-à-dire deux secondes après le début de l'affichage de la précédente.

Memento

```
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
int open(const char *pathname, int flags, mode_t mode);
off_t lseek(int fd, off_t offset, int whence);
/* whence = SEEK_SET ou SEEK_CUR ou SEEK_END */
int sigsetjmp(sigjmp_buf env, int savemask);
```

```
int dup2(int oldfd, int newfd);
int execvp(const char *file, char *const argv[]);
int pipe(int fildes[2]);
int sigprocmask(int how, sigset_t *set, sigset_t *oset);
int sigsuspend(const sigset_t *sigmask);
void siglongjmp(sigjmp_buf env, int val);
```