université de BORDEAUX

Collège Sciences et technologies

Année	2018-2019	Type	Examen
Licence	Informatique		
Code UE	4TIN503EX	Épreuve	Programmation Système
Date	13/12/2018	Documents	Non autorisés
Début	9h00	Durée	1h30

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. Il n'est pas demandé d'inclure les headers (stdio.h, unistd.h, ...) dans vos extraits de code.*

1 Questions de cours

Question 1 Décrivez ce que produit le programme suivant sur la sortie standard. Expliquez précisément.

```
int main ()
{
  printf ("Hello");
  fork ();
  return 0;
}
```

Question 2 Rappelez la différence entre les primitives `setjmp` et `sigsetjmp`. Dans quelles situations faut-il utiliser la seconde version ?

2 Threads

On s'intéresse au calcul du nombre d'apparitions de chaque caractère (de code ASCII compris entre 0 et 255) au sein d'un texte.

```
#define SIZE  ...

unsigned char texte [SIZE];
unsigned int occurrences [256];

void calculer_stats ()
{
  // mettre à 0 le nombre d'occurences
  memset(occurrences, 0, 256 * sizeof (int));

  for (int i = 0; i < SIZE; i++)
    occurrences [texte [i]]++;
}
```

Pour accélérer le calcul des statistiques sur une machine disposant de P cœurs, on souhaite paralléliser la boucle `for` en attribuant des blocs d'indices disjoints à chacun des P threads que l'on va créer pour l'occasion. Une fois la boucle terminée, l'exécution doit redevenir purement séquentielle.

Question 1 Donnez une nouvelle version de la fonction `calculer_stats`, dans laquelle la boucle `for` est effectuée en parallèle. Dans cette question, on ne se préoccupe pas des accès mémoire concurrents qui seront traités à la question suivante.

Question 2 L'instruction `occurrences[texte[i]]++` peut-elle poser problème si elle est exécutée par plusieurs threads simultanément ? Pourquoi ? Le cas échéant, donnez une solution **simple** permettant de corriger le problème. Indiquez simplement quelles instructions il faut ajouter, et où.

Question 3 Pourquoi ne va-t-on pas aller exactement P fois plus vite sur P cœurs avec la solution proposée à la question 2 ? Proposez une solution dans laquelle les *threads* n'accèdent pas directement au tableau `occurrences` durant les itérations, mais chacun à un tableau `occ_partiel` privé. Donnez le code complet.

3 Tubes

On suppose que l'on dispose des deux commandes suivantes :

— `reverse <file>`, qui imprime sur la sortie standard le contenu du fichier « file » en partant de la fin ;

— `diff <file1> <file2>`, qui compare le contenu de deux fichiers et dont le code de retour vaut 0 si les fichiers sont identiques. Il est également possible d'utiliser la syntaxe `diff <file> -`, auquel cas le contenu du fichier est comparé avec l'entrée standard.

En vous appuyant sur l'existence de ces deux commandes, écrivez le code d'un programme `compare <file1> <file2>` qui affiche sur la sortie standard un ou plusieurs des messages suivants :

— les fichiers sont identiques

— les fichiers sont le miroir l'un de l'autre

— les fichiers sont différents

Faites un croquis illustrant le rôle des processus et des éventuels tubes impliqués. Veillez à ne pas séquentialiser inutilement les traitements. On s'interdira d'utiliser la fonction `system()`...

4 Signaux

Voici la boucle principale d'un programme qui dessine à l'écran le tracé d'une fonction mathématique `f`.

```
float f (float x) { ... }

int main(int argc, char **argv)
{
  float x, y, prev_y = NAN; // NAN est une constante dont la valeur
  ...                       // n'est jamais retournée par la fonction f
  for(x = MIN_RANGE; x <= MAX_RANGE; x = x + DELTA) {
    y = f (x);
    if (prev_y != NAN)
      tracer_segment (x - DELTA, prev_y, x, y); // relie le point précédent
    prev_y = y;                                 // au point courant
  }
}
```

On voudrait que ce programme fonctionne même avec des fonctions qui ne seraient pas définies en tout point de l'intervalle (ex : $f = \frac{1}{x}$), c'est-à-dire qu'il soit capable de passer au point suivant lorsque le calcul de $f(x)$ échoue. On suppose que le signal `SIGFPE` est levé lorsque l'on tente d'évaluer la fonction f en un point où elle n'est pas définie. Bien sûr, on veillera à ne pas tracer de segment vers/depuis un point où la fonction n'est pas définie.

Donnez une nouvelle version de ce programme qui fonctionne comme demandé. Indiquez soigneusement quelles sont les variables additionnelles nécessaires.

Memento

```
int pipe(int fildes[2]);
int dup2(int oldfd, int newfd);
int sigsetjmp(sigjmp_buf env, int savemask);
void siglongjmp(sigjmp_buf env, int val);
int execlp(const char *file, const char *arg0, ...);
int execvp(const char *file, char *const argv[]);
pid_t waitpid(pid_t pid, int *stat_loc, int options);
int sigaction(int sig, void (*_handler)(int)); // Syntaxe simplifiée vue en cours
int sigprocmask(int how, sigset_t *set, sigset_t *oset);
int sigsuspend(const sigset_t *sigmask);
```