université de BORDEAUX DEVUIP Service Scolarité	**ANNÉE UNIVERSITAIRE 2014 – 2015** **SESSION 1 D'AUTOMNE** **PARCOURS / ÉTAPE : LSTS / L3** **CODE UE : J1IN5012** **Épreuve :** Programmation système (INF 355) **Date :** 16 décembre 2014 **Heure :** 11h00 **Durée :** 1h30 Documents : non autorisés Épreuve de M./Mme : F. Pellegrini	Département DL Licence

N.B. : - Les réponses aux questions doivent être argumentées et aussi concises que possible.
- 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.
- Vos code sources doivent être commentés de façon adéquate.
- Le barème est donné à titre indicatif.

Question 1 (20 points)

Expliquez ce qu'est le BIOS, et son rôle au sein de l'ordinateur. Décrivez les principales étapes de son fonctionnement. *(15 lignes maximum)*

Question 2 (40 points)

(2.1) (20 points)

Dans les systèmes Unix, il existe une commande « `tee` », qui copie son entrée standard sur sa sortie standard, en faisant une copie dans un fichier dont le nom est passé en paramètre. Voici un exemple d'utilisation de `tee` :

```
> echo Salut a tous | tee brol
Salut a tous
> cat brol
Salut a tous
```

Écrivez le code du programme `tee`.

(2.2) (20 points)

Dans les systèmes Unix, il existe également une commande « `time` », qui indique le temps pris par la commande qui lui est passée en paramètre. Par exemple :

```
> time sleep 1
real 0m1.009s
user 0m0.000s
sys 0m0.000s
```

On veut créer une commande « `teetime` » qui combine les commandes `tee` et `time` en une seule, c'est-à dire que les deux commandes suivantes soient fonctionnellement identiques :

```
> time commande arg1 arg2 ... | tee brol
> teetime brol commande arg1 arg2 ...
```

Écrivez le code du programme `teetime`.

Question 3 (50 points)

(3.1) (10 points)

Expliquez dans quel cas il est licite d'effectuer un appel à `longjmp ()` après un appel à `setjmp ()`, et dans quel cas cela ne fonctionnera pas. Expliquez dans quel cas il est licite d'effectuer un appel à `longjmp ()` au sein d'une routine de traitement d'interruption. *(15 lignes maximum)*

(3.2) (20 points)

Pour mettre en œuvre un mécanisme ressemblant à celui des exceptions, on veut créer une routine « `int essaie (int (* f) (void))` », qui exécute la fonction `f` (et ses sous-fonctions éventuelles) jusqu'à ce que `f` termine ou qu'une fonction « `void leve (int e)` » soit appelée pour lever une exception de type `e`. Dans le premier cas, la valeur 0 sera retournée par `essaie` ; dans le deuxième cas, ce sera la valeur entière `e`.

Écrivez le code des fonctions `essaie` et `leve`. On ne s'attend pas à pouvoir utiliser ces fonctions de façon réentrante.

(3.3) (20 points)

On veut maintenant borner le temps mis par la fonction `f`. Pour cela, on imagine une fonction « `int essaie_tempo (int (* f) (void), int duree)` » telle que, si l'exécution de `f` prend plus de `duree` secondes, cette exécution est interrompue et la fonction `essaie_tempo` renvoie la valeur -1.

Écrivez le code de la fonction `essaie_tempo` et des fonctions annexes dont vous pourrez avoir besoin. On rappelle que la fonction « `alarm (t)` » déclenche l'envoi d'un signal `SIGALRM` au bout de `t` secondes.

Question 4 (90 points)

On s'intéresse à d'un module de manipulation d'images en nuances de gris, chaque valeur étant codée sur 8 bits (sous la forme d'un `unsigned char`). On s'intéresse tout particulièrement à la fonction « `calculer_histo` », qui mémorise, pour chaque nuance de gris entre 0 et 255, le nombre de pixels de l'image qui l'utilisent. Voici le code séquentiel de cette fonction :

```
#define hauteur XXXX        /* Valeurs "en dur" à la compilation */
#define largeur YYYY

unsigned char  image[hauteur][largeur];
unsigned int   histo[256];

void calculer_histo ()
{
  int i, j;

  memset (histo, 0, 256 * sizeof (unsigned int)); /* Mettre l'histogramme à 0 */

  for (i = 0; i < hauteur; i ++)
    for (j = 0; j < largeur; j ++)
      histo[image[i][j]] ++;
}
```

(4.1) (30 points)

Pour exécuter en parallèle n occurrences d'une fonction f au sein d'un processus, on peut imaginer une fonction de lancement « `void parallele (int nbre, void (* f) (void * info))` », telle que `nbre` est le nombre d'occurrences à exécuter en parallèle et `f` est le pointeur sur la fonction à exécuter, celle-ci recevant en fait, en tant que valeur d'argument « `void *` », un pointeur vers une structure « `Info` ». Cette structure, dont un exemplaire différent doit donc être fourni à chaque *thread*, est définie par :

```
typedef struct Info_ {
  int       nbre;     /* Nombre de threads lancés                    */
  int       rang;     /* Rang du thread pointant sur cette structure */
} Info;
```

Ainsi, grâce à cette structure sur laquelle il reçoit un pointeur, chaque *thread* lancé, tout comme le *thread* initial, peut connaître le nombre total de threads et son propre rang (égal à 0 pour le *thread* initial, et compris entre 1 et (`nbre` $-$ 1) pour les autres).

Codez la fonction `parallele` au moyen des *threads* POSIX, sachant que le *thread* appelant doit lui aussi exécuter la fonction `f`, et que tous les *threads* lancés doivent avoir terminé avant que la fonction `parallele` ne retourne. Faites attention à bien gérer la mémoire des tableaux de structures que vous devez créer.

(4.2) (30 points)

Au moyen de cette fonction `parallele`, on veut réécrire la fonction `calculer_histo`, pour l'exécuter sur un nombre donné de *threads*. Le nouveau prototype sera donc : « `calculer_histo (int nbre)` ». Elle lancera en parallèle une fonction « `calculer_histo2 (void * info)` », qui travaillera sur les lignes de l'image. Seule la boucle en `i` sera donc parallélisée. La fonction de rang `rang` traitera les lignes à partir de la ligne de numéro `rang`, le numéro de ligne étant incrémenté de `nbre` à chaque fois. Écrivez le code des fonctions `calculer_histo` et `calculer_histo2`. Dans cette question, on ne se préoccupe pas des accès mémoire concurrents, qui seront traités à la question suivante.

(4.3) (30 points)

En quoi l'instruction « `histo[image[i][j]] ++;` » peut-elle poser problème si elle est exécutée simultanément par plusieurs *threads* ? En quoi le résultat de l'exécution pourrait-il différer du résultat attendu ? Réécrivez le code de la question précédente afin de corriger le problème.

Memento

```
int dup2 (int oldfd, int newfd);
int fclose (FILE * fp);
FILE * fdopen (int fd, const char * mode);
int fflush (FILE *);
FILE * fopen (const char * path, const char * mode);
  /* mode = "r", "r+", "w", "w+", "a", "a+" */
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);

int execlp (const char * file, const char * arg, ...);
int execvp (const char * file, char * const argv[]);
pid_t fork (void);

unsigned int alarm (unsigned int seconds);
sighandler_t signal (int signum, sighandler_t handler);
  /* typedef void (* sighandler_t) (int); */
int sigaction (int signum, struct sigaction * act, struct sigaction * oldact);
  /* struct sigaction {
       void     (*sa_handler)(int);
       sigset_t sa_mask;
       int      sa_flags;
       ...
     }; */

void longjmp (jmp_buf env, int val);
int setjmp (jmp_buf env);
void siglongjmp (sigjmp_buf env, int val);
int sigsetjmp (sigjmp_buf env, int savesigs);

int pthread_create (pthread_t * thread, const pthread_attr_t * attr,
                    void * (*start_routine) (void *), void *arg);
void pthread_exit (void * retval);
int pthread_join (pthread_t thread, void ** retval);

int pthread_mutex_init (pthread_mutex_t * mutex,
                        const pthread_mutexattr_t * attr);
int pthread_mutex_destroy (pthread_mutex_t * mutex);
int pthread_mutex_lock (pthread_mutex_t * mutex);
int pthread_mutex_unlock (pthread_mutex_t * mutex);
```