

Pale INF583 – Systèmes d’exploitation

2010–2011

- L’examen dure 3 heures.
- Tous documents autorisés.
- Il est impératif de commenter les programmes et de justifier vos réponses.

Exercice 1

Question 1

Décrivez la fonction mathématique implémentée par le programme suivant, et décrivez (succinctement) le rôle de chaque séquence d’instructions ou de déclaration suivant immédiatement un commentaire de la forme // C?. Vous pourrez écrire simplement le numéro du commentaire suivi de votre explication. Par exemple :

C0 : déclaration des constantes pour le nombre de threads et la taille des vecteurs.

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

// C0
#define NUMTHRDS 4
#define VECLLEN 100

// C1
typedef struct {
    double    *a;
    double    *b;
    double    sum;
    int       veclen;
} DOTDATA;

// C2
DOTDATA dotstr;

// C3
pthread_t callThd[NUMTHRDS];

// C4
pthread_mutex_t mutexsum;

// C5
void *dotprod(void *arg) {
    int i, start, end, len;
    long offset;
    double mysum, *x, *y;
    offset = (long)arg;

// C6
    len = dotstr.veclen;
    start = offset*len;
```

```

end = start + len;
x = dotstr.a;
y = dotstr.b;

// C7
mysum = 0;
for (i=start; i<end ; i++) {
    mysum += (x[i] * y[i]);
}

// C8
pthread_mutex_lock (&mutexsum);
dotstr.sum += mysum;
pthread_mutex_unlock (&mutexsum);

pthread_exit((void*)0);
}

int main (int argc, char *argv[]) {
    long i;
    double *a, *b;
    void *status;
    pthread_attr_t attr;

    // C9
    a = (double*) malloc (NUMTHRDS*VECLEN*sizeof(double));
    b = (double*) malloc (NUMTHRDS*VECLEN*sizeof(double));

    for (i=0; i<VECLEN*NUMTHRDS; i++) {
        a[i]=1;
        b[i]=a[i];
    }

    dotstr.veclen = VECLLEN;
    dotstr.a = a;
    dotstr.b = b;
    dotstr.sum=0;

    pthread_mutex_init(&mutexsum, NULL);

    // C10
    for(i=0;i<NUMTHRDS;i++) {
        pthread_create(&callThd[i], &attr, dotprod, (void *)i);
    }

    // C11
    for(i=0;i<NUMTHRDS;i++) {
        pthread_join(callThd[i], &status);
    }

    // C12
    printf ("Sum = %f \n", dotstr.sum);

    free (a);
    free (b);

    pthread_mutex_destroy(&mutexsum);

    pthread_exit(NULL);
}

```

Question 2

3

Pourquoi l'accumulation s'effectue-t-elle dans la variable privée `mysum` et non pas dans le champ partagé entre tous les threads `dotstr.sum` ?

Question 3

Proposez deux méthodes pour améliorer les performances en éliminant l'utilisation de mutex, et expliquez succinctement les changements à apporter au programme dans les deux cas.

Exercice 2

Il s'agit d'un questionnaire à choix multiple sur le C et le système. Écrivez sur votre copie le numéro de la réponse choisie.

Notation : 1 point par bonne réponse, 0 points en l'absence de réponse, -0,5 points par réponse fautive. La note obtenue entre -2 et 4 est ensuite ramenée au nombre de points accordé à l'exercice dans le barème.

1. Que déclare la ligne de C suivante ?

```
int *p(void*);
```

- (1) un pointeur de fonction sans arguments
- (2) un pointeur de fonction avec argument pointeur quelconque
- (3) une fonction sans argument retournant un pointeur d'entiers
- (4) une fonction avec argument pointeur quelconque retournant un pointeur d'entiers

2. Que vaut `x` après l'exécution des lignes suivantes ?

```
int x=1, y=2; int *p=&y; int **q=&p;
*q=&x; **q+=*p;
```

- (1) 1
- (2) 2
- (3) 3
- (4) 4

3. Que fait le code suivant :

```
#include <stdio.h>
int main (int argc, char *argv[]) {
    char buf[100];
    FILE *fd=fopen ("toto", "r");
    read (fileno(fd), buf, 100);
}
```

- (1) il ouvre un fichier `toto` en lecture seule et lit 100 octets
 - (2) il ouvre un fichier `toto` en lecture/écriture et lit 100 octets
 - (3) rien, la syntaxe est incorrecte
 - (4) n'importe quoi, les types des arguments sont incorrects
4. Lors qu'un processus utilisant plus que la moitié de la mémoire disponible effectue un appel système `fork()`, quel résultat n'est *pas* susceptible de se produire ?
- (1) le processus est tué immédiatement pour libérer de la mémoire
 - (2) un autre processus du système est tué immédiatement
 - (3) l'appel système renvoie -1 et `errno` vaut `ENOMEM` (plus de mémoire)
 - (4) `fork` retourne normalement

Exercice 3

On souhaite réaliser un système de messagerie instantanée local. Un dossier contient plusieurs fichiers texte, chacun contenant les conversations de l'un des salons hébergés par le serveur de messagerie.

Un "client" inscrit au salon est un processus permettant d'écrire et de lire la conversation dans le fichier correspondant à ce salon. Dans cet exercice, on suppose qu'il n'y a qu'un seul salon actif, et que le fichier associé est appelé "salon.txt".

Un client écrit dans "salon.txt" lorsque l'utilisateur veut envoyer du texte dans la conversation. Le client teste en boucle si "salon.txt" n'a pas été modifié par d'autres clients :

- si non, il ne fait rien ;
- si oui, il lit les derniers octets ajoutés au fichier et les affiche sur sa sortie standard.

Question 4

Ce mode de fonctionnement impose de résoudre un problème de concurrence. Lequel ? Comment le résoudre ?

Question 5

Le client test en boucle l'état du fichier "salon.txt". Écrivez une fonction C qui implémente cette boucle d'attente, avec l'appel système approprié.

Question 6

Expliquez en quelques phrases (maximum) ce qu'est l'attente active, et donnez un exemple différent de celui de la boucle d'attente précédente.

Question 7

On souhaite mettre en place un système d'attente passive.

On choisit de mettre en place un système de signaux pour détecter quand un message a été posté dans le salon. On se restreint à l'utilisation du signal SIGUSR1. La réception par le client d'un signal SIGUSR1 signifie que "salon.txt" a été modifié par un autre client.

Quelle fonction utiliser pour mettre en place la gestion du signal SIGUSR1 dans le client ? Faut-il faire un programme multithreadé ? Pourquoi ?

Question 8

Modifiez la fonction et la boucle d'attente précédentes pour mettre en place ce mécanisme de détection du signal SIGUSR1.

Question 9

Quelles précautions faut-il prendre dans le client lorsque celui-ci lit sur son entrée standard ce que l'utilisateur tape au clavier ?

Question 10

On s'intéresse désormais à l'émission du signal SIGUSR1.

Qui est responsable de l'envoi de SIGUSR1 pour signaler la modification de "salon.txt" ? Quel est la principale difficulté lors de l'utilisation du signal SIGUSR1 pour signaler une modification de "salon.txt" ?

Question 11

Comment chaque client peut obtenir et mettre à jour la liste des clients à qui envoyer SIGUSR1 ? Détaillez le mécanisme (inutile d'écrire un programme C).

Question 12

Le client est incapable d'afficher les messages pendant qu'il est en attente sur l'entrée standard (lorsque l'utilisateur saisit un nouveau message). Proposez une solution à plusieurs processus qui permette l'affichage des messages entrants pendant la saisie interactive. Décrivez votre solution sans programme C (texte, dessin).

Question 13

Pour améliorer la solution précédente, vous ferez en sorte que le ou les processus fils se terminent proprement lorsque le processus client principal se termine par un appel à `exit()` ou qu'il est tué par un signal rattrapable/masquable (comme SIGINT, associé à Ctrl-C par exemple).

Question 14

Expliquer comment plusieurs utilisateurs connectés à internet peuvent utiliser le chat de manière sécurisée pour discuter entre eux sachant que le programme n'utilise aucune socket ?

Question 15

Comment faire pour "commencer à sécuriser" le système de messagerie instantanée, i.e., autoriser seulement certains utilisateurs à accéder et/ou participer à certains salons ? Donner un exemple. Quelles sont les limites de ce système ?