TP: Thread

R305

Département Informatique

2022

Introduction

Compilation d'un programme multi-thread

L'utilisation des threads sous Linux nécessite la présence de la librairie `Posix Thread` lors de la compilation.

Pour compiler un programme multithread, nous devons informer `gcc` de cela avec le flag `pthread` :

```
gcc -pthread programme.c -o nomexecutable
```

Avec cette commande `gcc` va inclure la librairie `PThread` qui contient toutes les fonctions nécessaires à la gestion et à la synchronisation des threads.

Exo 1: calculateur multithread

Partie 1: Introduction

Le code en annexe A présente un simulateur de calcul.

Dans la version présentée, ce simulateur n'utilise qu'un seul thread pour effectuer toutes les opérations arithmétiques présentes dans un tableau.

Néanmoins, ce programme reste multi-thread car nous utilisons deux threads supplémentaires:

- un thread pour afficher une animation d'attente
- un thread pour afficher les résultats des calculs

A faire:

- Dans un premier temps, compilez et exécutez le programme.

Vous remarquez que pendant l'attente, une animation est affichée sur le terminal.

Questions:

- Comment expliquez cela ?
- Comment procède-t-on pour arrêter cette animation une fois les calculs terminés ?

Partie 2: Les calculs en parallèle

Nous souhaitons maintenant avoir un certain nombre, n, d'activités de calcul arithmétique en parallèle.

L'utilisateur devra préciser le nombre n d'activités qu'il souhaite pour effectuer tous les calculs.

Par exemple, nous pourrions lancer le programme en précisant le nombre *n* comme paramètre:

```
./exo2 4
```

Cette commande, lancera alors 4 activités de calcul en parallèle.

L'effort de traitement, tant que possible, devra être équilibré entre toutes les activités de calcul.

Le thread principal, qui sera responsable de lancer les différentes activités de calcul, devra attendre leur fin pour ensuite afficher tous les résultats.

A faire:

- Nous vous demandons de réaliser l'implémentation du programme des calculs parallèles
- En utilisant votre programme, comparez les temps de calcul en testant plusieurs valeurs pour le nombre d'activités.
- Que pouvez-vous déduire ?

Exo 2: Calculateur de fréquence de lettres

Cet exercice fera l'objet d'un rendu individuel pour le contrôle continu.

Objectif:

Nous souhaitons réaliser une application qui va calculer et afficher la fréquence d'apparition des lettres de l'alphabet dans un ensemble de fichiers textes. Nous allons considérer uniquement les lettres ASCII de `a` à `z` sans prendre en compte les accents.

L'utilisateur devra spécifier les chemins des fichiers à traiter comme des paramètres de la ligne de commande.

Voici un exemple d'utilisation de votre programme:

```
./exo3 fable1.txt fable2.txt fable3.txt
```

Vous devez alors compter toutes les occurrences des lettres dans tous ces fichiers (`fable1.txt` , `fable2.txt` et `fable3.txt`) et donner leurs fréquences en pourcentage.

Voici un affichage possible pour votre programme:

```
./exo3 fable1.txt fable2.txt fable3.txt
nb total caracteres: 1143
a 93 8.14 %
b 19 1.66 %
c 18 1.57 %
d 68 5.95 %
e 141 12.34 %
f 32 2.80 %
g 37 3.24 %
h 78 6.82 %
i 76 6.65 %
j 2 0.17 %
k 15 1.31 %
l 34 2.97 %
m 30 2.62 %
n 80 7.00 %
o 91 7.96 %
p 14 1.22 %
q 1 0.09 %
```

```
r 70 6.12 %
s 71 6.21 %
t 94 8.22 %
u 32 2.80 %
v 10 0.87 %
w 12 1.05 %
x 1 0.09 %
y 23 2.01 %
z 1 0.09 %
```

Ceci nous informe que dans les trois fichiers la lettre `a` est apparue 93 fois sur un total de 1143 caractères soit une proportion de 8.14%.

Spécifications

Vous devez prendre en compte les points suivants pour réaliser votre application:

- Afin d'accélérer les calculs nous vous demandons de dédier pour chaque fichier un thread de traitement.
- Le thread principale devra attendre la fin de tous les threads de traitement avant d'afficher les fréquences des lettres.
- Il est possible que différents threads essaient de mettre à jour les données du même caractère au même moment. Vous devez identifier toutes les sections critiques et les protéger avec des mutexes.
- Attention les lettres dans les textes peuvent être en majuscule, il faudra alors convertir toutes les lettres en miniscule pour uniformiser la représentation.

Aide:

L’annexe B présente le code d’une application monothread pour compter les fréquences des lettres. Vous pouvez utiliser de ce code comme point de départ pour construire votre version multithread.

Tests

Un jeu de test est fourni. Il s’agit des fables de Jean de La Fontaine traduites en anglais.

Comparez vos résultats obtenus avec le jeu de test avec la fréquence des lettres dans la langue anglaise tirée du Concise Oxford Dictionary (9th edition, 1995):

Lettre	**Frequence**	**Lettre**	**Frequence**
E	11.1607%	M	3.0129%
A	8.4966%	H	3.0034%
R	7.5809%	G	2.4705%
I	7.5448%	B	2.0720%
O	7.1635%	F	1.8121%
T	6.9509%	Y	1.7779%
N	6.6544%	W	1.2899%
S	5.7351%	K	1.1016%
L	5.4893%	V	1.0074%

C	4.5388%	X	0.2902%
U	3.6308%	Z	0.2722%
D	3.3844%	J	0.1965%
P	3.1671%	Q	0.1962%
———	———	———	———

Annexe A

```c
#include <sys/types.h>
#include <unistd.h>
#include <stdint.h>
#include <time.h>
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>

// taille du tableau de requêtes
#define TAILLE_FILE 20

// une structure pour la requete et le resultat
typedef struct
{
    float operand1;
    float operand2;
    float resultat;
    char operation; // '+' '*' '/' '-'
} requete_t;

// simulateur du calculateur
float simulateur_calcul(float op1, float op2, char operation)
```

```
{
    float result;
    switch (operation)
    {
    case '+':
        sleep(1);
        result = op1 + op2;
        break;
    case '-':
        sleep(1);
        result = op1 - op2;
        break;
    case '*':
        sleep(2);
        result = op1 * op2;
        break;
    case '/':
        sleep(3);
        result = op1 / op2;
        break;
    default:
        sleep(1);
        result = 0;
        break;
    }
    return result;
}

// les opérations arithmétiques
#define NUMBER_OPS 4
char operations[NUMBER_OPS] = {
    '+',
    '-',
    '*',
    '/',
};
// tableau des commandes à faire
requete_t requetes_tab[TAILLE_FILE];
```

```c
void initialisation()
{
    srand((unsigned int)time(NULL));
}

// retourne un nombre float random entre 0 et a
float random_float(float a)
{
    return ((float)rand() / (float)(RAND_MAX)) * a;
}
// fonction pour remplir le tableau des opérations
int remplir_les_commandes()
{
    for (int i = 0; i < TAILLE_FILE; i++)
    {
        requetes_tab[i].operation = operations[rand() % NUMBER_OPS];
        requetes_tab[i].operand1 = random_float(1.0);
        requetes_tab[i].operand2 = random_float(1.0);
    }
}


//========Les activités=========================

//structure pour les parametres d une activite de calcul
typedef struct argument_calcul_t  {
    int index_debut ;
    int index_fin ;
    int step ;
}  argument_calcul_t;

//activite de calcul
void* activite_calculateur(void* arg)
{
    printf("thread[%ld] lance\n",pthread_self());
    argument_calcul_t* data = (argument_calcul_t*) arg;

    for (int i = data->index_debut; i < data->index_fin ; i = i +
        data->step)
    {
```

```
            requetes_tab[i].resultat =
            simulateur_calcul(requetes_tab[i].operand1,

            requetes_tab[i].operand2,

            requetes_tab[i].operation);
        }
        printf("thread[%ld] fin\n",pthread_self());
}


//affichage des resultats
void* activite_affichage(void* arg)
{
    for (int i = 0; i < TAILLE_FILE ; i = i + 1)
    {
                printf("%f %c %f = %f\n", requetes_tab[i].operand1,
                requetes_tab[i].operation,
                requetes_tab[i].operand2,
                requetes_tab[i].resultat);
    }
    fflush(stdout);
}
//affichage d une animation
void* activite_attente(void* arg)
{
    char anim[4] = {'/','-','\\','|'};
    int i = 0;
    while(1)
    {
        printf("\033[1D"); //demande au terminal de deplacer le
        curseur une case a gauche
        printf("%c",anim[i]);
        i = (i+1)%4;
        fflush(stdout);
        sleep(1);
    }
}
//=============================================
int main(int argc, char **argv)
{
    initialisation();
```

```
    remplir_les_commandes();
    // traitement

    pthread_t thread_calcul_id;

    //calcul de 0 à TAILLE par pas de 1
    argument_calcul_t parametre_calcul;
    parametre_calcul.index_debut = 0;
    parametre_calcul.index_fin = TAILLE_FILE;
    parametre_calcul.step = 1;
    pthread_create(&thread_calcul_id,NULL,
        activite_calculateur,&parametre_calcul);

    //une activite pour animer l'attente de l'utilisateur
    pthread_t thread_attente_id;
    pthread_create(&thread_attente_id,NULL, activite_attente,NULL);

    pthread_join(thread_calcul_id,NULL);
    //le calcul est pret nous devons annuler l affichage de l
        attente
    pthread_cancel(thread_attente_id);



    // affichage des résultats
    pthread_t thread_affichage_id;

    pthread_create(&thread_affichage_id,NULL,
        activite_affichage,NULL);
    pthread_join(thread_affichage_id,NULL);


    exit(0);
}
```

Annexe B

```
#include <sys/types.h>
#include <unistd.h>
```

```c
#include <stdint.h>
#include <time.h>
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>

FILE *ouvreFichierLecture(const char *path)
{
    return fopen(path, "r");
}

char lireCaractere(FILE *file)
{
    return fgetc(file);
}

int index_caractere(char c)
{
    return isalpha(c) ? tolower(c) - 'a' : -1;
}

typedef struct
{
    char *path;
} info_fichier_t;

typedef struct
{
    char caractere;
    unsigned int nb_occurence;
} info_caractere_t;

typedef struct
{
    unsigned int nb_caractere_total;
    info_caractere_t occurence_info[26];
} info_frequence_t;

info_frequence_t info_frequence;
```

```c
void traitementFichier(info_fichier_t *data)
{
    FILE *f = ouvreFichierLecture(data->path);
    if (f)
    {

        char c;
        while ((c = lireCaractere(f)) != EOF)
        {
            int i = index_caractere(c);
            if (i >= 0 && i < 26)
            {
                info_frequence.nb_caractere_total++;
                info_frequence.occurence_info[i].nb_occurence++;
            }
        }
    }
}

void init()
{
    info_frequence.nb_caractere_total = 0;
    for (char i = 'a'; i <= 'z'; i++)
    {
        info_frequence.occurence_info[i - 'a'].caractere = i;
        info_frequence.occurence_info[i - 'a'].nb_occurence = 0;
    }
}

void affiche()
{
    printf("nb total caracteres: %d\n",
         info_frequence.nb_caractere_total);
    for (char i = 'a'; i <= 'z'; i++)
    {
        int occurrence = info_frequence.occurence_info[i -
         'a'].nb_occurence;
        float freq = ((float)occurrence /
         info_frequence.nb_caractere_total) * 100;
```

```
        printf("%c %d %.2f %%\n", info_frequence.occurence_info[i -
         'a'].caractere, occurrence, freq);
    }
}


int main(int argc, char **argv)
{
    if (argc < 2)
    {
        printf("%s fichier1 fichier2 fichier3 ...", argv[0]);
        exit(0);
    }
    init();
    int nbfichiers = argc - 1;
    info_fichier_t *fichiers = malloc(sizeof(info_fichier_t) *
        (nbfichiers));


    for (int i = 0; i < nbfichiers; i++)
    {
        /* code */
        fichiers[i].path = argv[i + 1];
        traitementFichier(&fichiers[i]); // lancer le thread
    }
    affiche();
}
```