

TP1 : Compte Rendu

- Réalisé par : Walid NOUBIR & Zakaria EZZHAR
- Filière : Ingénierie Informatique et Systèmes Embarqués

Exercice 1 :

1- Le facteur d'utilisation du processus : $U = C / P$

→ Pour T0 : $U_0 = 2/6 = 1/3$

→ Pour T1 : $U_1 = 3/8$

→ Pour T2 : $U_2 = 4/24 = 1/6$

2- Le facteur de charge du processus : $ch = C / D$

- puis que $D = P$ alors :

→ Pour T0 : $ch_0 = U_0 = 1/3$

→ Pour T1 : $ch_1 = U_1 = 3/8$

→ Pour T2 : $ch_2 = U_2 = 1/6$

3- Le temps de réponse : $TR = f - r$

→ Pour T0 : Block 1 : $2 - 0 = 2$; Block 2 : $8 - 6 = 2$; Block 3 : $14 - 12 = 2$; Block 4 : $20 - 18 = 2$;

→ Pour T1 : Block 1 : $5 - 0 = 5$; Block 2 : $11 - 8 = 3$; Block 3 : $21 - 16 = 5$;

→ Pour T2 : Block 1 : $16 - 0 = 16$;

4- La laxité nominale : $L = D - C$

→ Pour T0 : $L_0 = 6 - 2 = 4$

→ Pour T1 : $L1 = 8 - 3 = 5$

→ Pour T2 : $L2 = 24 - 4 = 20$

5- Les giques :

Exercice 2 :

```
[*] TP1_EX2.c TP1_EX4.c TP1_EX5.c TP1_EX3.c
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <pthread.h> // La biblio. des treads
4  #include <string.h>
5
6  // Fonction exécutée par le thread
7  void *print_message(void *arg){
8      printf("Salut Nous sommes ZAKARIA et WALID! \n"); // Affiche un message
9      return NULL;
10 }
11
12 int main(int arg, char *argv[]){
13     pthread_t msg; // Déclaration de la variable pour stocker l'identifiant du thread
14     pthread_create(&msg, NULL, print_message, NULL); // Création d'un nouveau thread
15     pthread_join(msg, NULL); // Attente de la fin du thread créé
16     return EXIT_SUCCESS; // Fin du programme principal
17 }
18
```

Exécution du code :

```
C:\Users\dell\OneDrive\Documents\FS LE\S6\Temps Reel\TP\TP1_EX2.exe
Salut Nous sommes ZAKARIA et WALID!
-----
Process exited after 0.1055 seconds with return value 0
Appuyez sur une touche pour continuer...
```

Exercice 3 :

```
[*] TP1_EX2.c TP1_EX4.c TP1_EX5.c [*] TP1_EX3.c
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <pthread.h>
4  #include <string.h>
5
6  void *Tache1(void *arg){
7      int i = 0;
8      while(i<5){
9          printf("Execution de Tache1\n");
10         sleep(1); // Pause d'une seconde
11         i++;
12     }
13     return NULL;
14 }
15
16 void *Tache2(void *arg){
17     int j = 0;
18     while(j<3){
19         printf("Execution de Tache2\n");
20         sleep(2); // Pause de deux secondes
21         j++;
22     }
23     return NULL;
24 }
25
26 int main(int arg, char *argv[]){
27     pthread_t thread1, thread2;
28     pthread_create(&thread1, NULL, Tache1, NULL);
29     pthread_create(&thread2, NULL, Tache2, NULL);
30     pthread_join(thread1, NULL);
31     pthread_join(thread2, NULL);
32     return EXIT_SUCCESS;
33 }
34 /* Ce code peut entraîner un affichage parallèle des messages
35 "Execution de Tache1" et "Execution de Tache2" dans la sortie.*/
36
37
38 int main(int arg, char *argv[]){
39     pthread_t thread1, thread2;
40     pthread_create(&thread1, NULL, Tache1, NULL);
41     pthread_join(thread1, NULL);
42     pthread_create(&thread2, NULL, Tache2, NULL);
43     pthread_join(thread2, NULL);
44     return EXIT_SUCCESS;
45 }
46 /* Dans cette version de la fonction main, Le thread 2 ne commence
47 son exécution qu'après que le thread 1 se soit terminé, en raison
48 de l'appel à pthread_join(thread1, NULL) avant la création du thread 2.*/
49
50
```

→ 3 / Dans la première version, les deux threads démarrent simultanément, permettant un affichage des messages dans un ordre non déterministe. Dans la deuxième version, le démarrage du deuxième thread est retardé jusqu'à ce que le premier thread se termine, garantissant un affichage séquentiel des messages.

Exécution du thread1 :

```
C:\Users\dell\OneDrive\Documents\F5 LE\S6\Temps Reel\TP\TP2\TP1_EX3.exe
Execution de Tache1
Execution de Tache2
Execution de Tache1
Execution de Tache1
Execution de Tache2
Execution de Tache1
Execution de Tache2
Execution de Tache1

-----
Process exited after 6.146 seconds with return value 0
Appuyez sur une touche pour continuer...
```

Exécution du thread2 :

```
C:\Users\dell\OneDrive\Documents\F5 LE\S6\Temps Reel\TP\TP2\TP1_EX3.exe
Execution de Tache1
Execution de Tache1
Execution de Tache1
Execution de Tache1
Execution de Tache1
Execution de Tache2
Execution de Tache2
Execution de Tache2

-----
Process exited after 11.22 seconds with return value 0
Appuyez sur une touche pour continuer...
```

Exercice 4 :

```

1 #include <stdio.h>
2 #include <pthread.h>
3
4 // Fonction exécutée par Le premier thread
5 void *thread_func1(void *arg) {
6     printf("Thread 1: Bonjour !\n");
7     return NULL;
8 }
9
10 // Fonction exécutée par Le deuxième thread
11 void *thread_func2(void *arg) {
12     printf("Thread 2: Salut !\n");
13     return NULL;
14 }
15
16 int main() {
17     pthread_t tid1, tid2;
18
19     // Création des deux threads
20     pthread_create(&tid1, NULL, thread_func1, NULL);
21     pthread_create(&tid2, NULL, thread_func2, NULL);
22
23     // Attente de la fin de l'exécution des deux threads créés
24     pthread_join(tid1, NULL);
25     pthread_join(tid2, NULL);
26
27     // Terminaison du thread principal
28     pthread_exit(NULL);
29 }

```

Exécution du code :

```

C:\Users\LENOVO\AppData\Local\Microsoft\Windows\INetCache\IE\DL6K9MYN\EXO4_Temps_reel[1].exe
Thread 1: Bonjour !
Thread 2: Salut !

-----
Process exited after 0.101 seconds with return value 0
Appuyez sur une touche pour continuer...

```

Exercice 5 :

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

// Structure représentant une tâche périodique
typedef struct {
    int id;
    int period;
} PeriodicTask;

// Fonction exécutée par chaque thread de tâche périodique
void *taskFunction(void *arg) {
    PeriodicTask *task = (PeriodicTask *)arg;

    while (1) {
        sleep(task->period);
        printf("la tâche %d est exécutée\n", task->id);
    }
    // Le thread termine son exécution
    pthread_exit(NULL);
}

int main() {
    // Création des tâches périodiques
    PeriodicTask task1 = {1, 2}; // Tâche 1 avec période de 2 secondes
    PeriodicTask task2 = {2, 3}; // Tâche 2 avec période de 3 secondes

    // Création des threads pour exécuter les tâches
    pthread_t thread1, thread2;
    pthread_create(&thread1, NULL, taskFunction, (void *)&task1);
    pthread_create(&thread2, NULL, taskFunction, (void *)&task2);

    // Attente des threads (les tâches s'exécutent en arrière-plan)
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    return 0;
}
```


Exécution du code :

C:\Users\LENOVO\Desktop\To prepare Language C\Exercice_5.exe

```
la tache 1 est executee
la tache 2 est executee
la tache 1 est executee
la tache 2 est executee
la tache 1 est executee
la tache 1 est executee
la tache 2 est executee
la tache 1 est executee
la tache 2 est executee
la tache 1 est executee
la tache 1 est executee
la tache 2 est executee
la tache 1 est executee
la tache 2 est executee
la tache 1 est executee
la tache 1 est executee
la tache 2 est executee
la tache 1 est executee
la tache 2 est executee
la tache 1 est executee
la tache 1 est executee
la tache 2 est executee
```

Exercice 6 :

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

#define ARRAY_SIZE 8 // Taille du tableau d'entiers
#define NUM_THREADS 4 // Nombre de threads à utiliser
#define PART_SIZE (ARRAY_SIZE / NUM_THREADS) // Taille des parties du tableau

int array[ARRAY_SIZE]; // Tableau d'entiers
int total_sum = 0; // Somme totale des éléments du tableau
pthread_mutex_t mutex; // Verrou pour synchroniser l'accès à total_sum

// Structure contenant les informations nécessaires pour effectuer le calcul partiel
typedef struct {
    int start_index; // Indice de début du sous-tableau
    int end_index; // Indice de fin du sous-tableau
} PartialSumInfo;

// Fonction pour calculer la somme des éléments d'un sous-tableau du tableau global
void *sum_partial(void *arg) {
    PartialSumInfo *info = (PartialSumInfo *)arg;
    int partial_sum = 0;

    for (int i = info->start_index; i < info->end_index; i++) {
        partial_sum += array[i];
    }

    // Verrouillage pour éviter les accès concurrents à total_sum
    pthread_mutex_lock(&mutex);
    total_sum += partial_sum;
    pthread_mutex_unlock(&mutex);
}
```



```
int main() {
    pthread_t threads[NUM_THREADS]; // Tableau de threads
    PartialSumInfo infos[NUM_THREADS]; // Tableau de structures d'informations pour chaque thread

    // Initialisation du tableau d'entiers avec des valeurs de 1 à ARRAY_SIZE
    for (int i = 0; i < ARRAY_SIZE; i++) {
        array[i] = i + 1;
    }

    // Initialisation du verrou
    pthread_mutex_init(&mutex, NULL);

    // Création des threads et division du tableau en parties
    for (int i = 0; i < NUM_THREADS; i++) {
        infos[i].start_index = i * PART_SIZE;
        infos[i].end_index = (i + 1) * PART_SIZE;
        pthread_create(&threads[i], NULL, sum_partial, (void *)&infos[i]);
    }

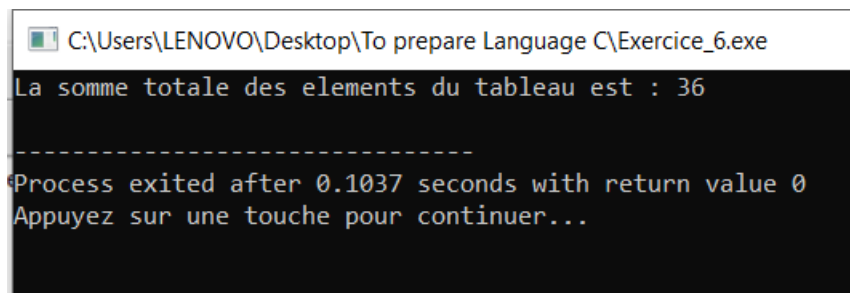
    // Attente de la fin de l'exécution de tous les threads
    for (int i = 0; i < NUM_THREADS; i++) {
        pthread_join(threads[i], NULL);
    }

    // Affichage de la somme totale
    printf("La somme totale des elements du tableau est : %d\n", total_sum);

    // Destruction du verrou
    pthread_mutex_destroy(&mutex);

    return 0;
}
```

Exécution du code :



```
C:\Users\LENOVO\Desktop\To prepare Language C\Exercice_6.exe
La somme totale des elements du tableau est : 36
-----
Process exited after 0.1037 seconds with return value 0
Appuyez sur une touche pour continuer...
```