Université Ibn Zohr
Faculté des Sciences Agadir
Dépt. Informatique

2023/2024

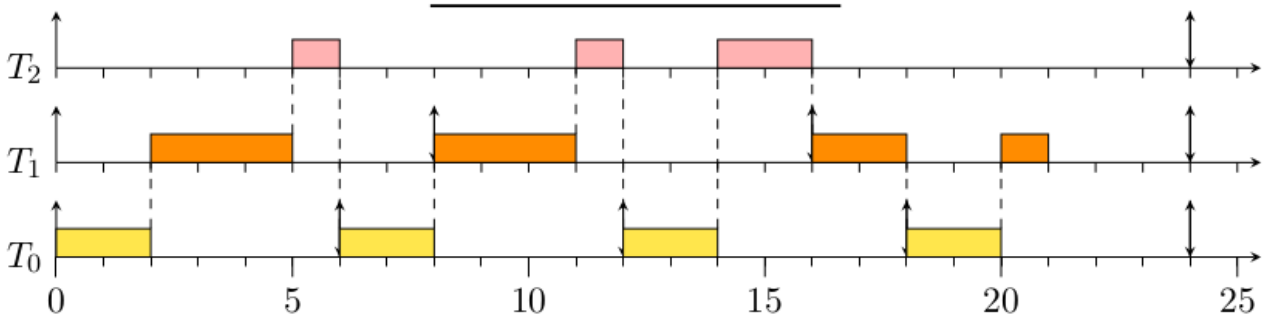
SÉRIE 1 : Programmation Temps Réel
Tâches, Thread POSIX.

Exercice 1

Soit les tâches suivantes:

Tâche	Coût	Période
T_0	2	6
T_1	3	8
T_2	4	24

le type des tache : périodique sur requête



facteur d'utilisation u=C/P
facteur de charge ch=C/D
TRi=fi-ri
Ln=D-C
D=d-r

Calculer pour chaque tâches:

1. le facteur d'utilisation du processeur.
2. le facteur de charge du processeur.
3. le temps de réponse.
4. la laxité nominal.
5. La gigue de release relative, la gigue de release absolue, la gigue de fin relative et la gigue de fin absolue.

Exercice 2

Créer un programme en langage C qui utilise la bibliothèque pthread pour exécuter un thread qui affiche un message simple à l'écran.

1. Créez une fonction **print_message** qui prend comme argument un pointeur vers une chaîne de caractères (le message à afficher) et qui affiche ce message à l'écran.

2. Dans la fonction main, créez une chaîne de caractères contenant le message que vous souhaitez afficher.
3. Utilisez la fonction **pthread_create** pour créer un thread en lui passant la fonction **print_message** et la chaîne de caractères comme argument.
4. Assurez-vous que le programme principal (main) attend la fin de l'exécution du thread en utilisant la fonction **pthread_join**.
5. Compilez votre programme avec l'option nécessaire pour inclure la bibliothèque **pthread**.
6. Exécutez le programme et observez le message affiché par le thread.

Exercice 3

Soit les deux fonctions suivantes :

```c
void *Tache1(void *arg) {

    int i=0;
    while(i<5)
    {
        printf("Execution  de Tache1\n");
        sleep(1);
        i++;
    }
    return NULL;
}
```

```c
void *Tache2(void *arg) {
    int j=0;
    while(j<3)
    {
        printf("Execution de Tache2\n");
        sleep(2);
        j++;
    }
    return NULL;
}
```

Dans le programme principale, les deux fonctions sont exécutées par deux thread (**thread1 , thread2**).

Exemple 1 :

```c
int main(int argc,char *argv[])
{

    pthread_t thread1,thread2;
    pthread_create(&thread1,NULL,Tache1,NULL);
    pthread_create(&thread2,NULL,Tache2,NULL);
    pthread_join(thread1,NULL);
    pthread_join(thread2,NULL);
    return EXIT_SUCCESS;
}
```

Exemple 2 :

```c
int main(int argc,char *argv[])
{

    pthread_t thread1,thread2;
    pthread_create(&thread1,NULL,Tache1,NULL);
    pthread_join(thread1,NULL);
    pthread_create(&thread2,NULL,Tache2,NULL);
    pthread_join(thread2,NULL);
    return EXIT_SUCCESS;
}
```

1. Exécuter le premier programme et donner le résultat de son exécution. Qu’est ce que vous remarquer ?
2. Exécuter le deuxième programme et donner le résultat de son exécution. Qu’est ce que vous remarquer ?
3. Expliquer la différence entre les deux résultats.

Exercice 4

Créer un programme en langage C qui utilise la bibliothèque pthread pour exécuter deux threads simultanément. Chaque thread aura pour tâche d'afficher un message différent.

1. Créez deux fonctions, **thread_func1** et **thread_func2**, qui seront exécutées respectivement par le premier et le deuxième thread. Ces fonctions doivent afficher un message simple, par exemple, "Thread 1: Bonjour !" et "Thread 2: Salut !".
2. Dans la fonction main, utilisez la fonction **pthread_create** pour créer les deux threads en leur attribuant les fonctions que vous avez définies.
3. Assurez-vous que le programme principal (**main**) attend la fin de l'exécution des deux threads en utilisant la fonction **pthread_join**.
4. Compilez votre programme avec l'option nécessaire pour inclure la bibliothèque **pthread**.
5. Exécutez le programme et observez les messages affichés par les deux threads.

Exercice 5

Dans cet exercice on souhaite proposer une implémentation de gestion de tâches périodiques en C. Utiliser des threads pour exécuter les tâches périodiques de manière concurrente. Chaque tâche est représentée par une structure **PeriodicTask** contenant son identifiant (**id**) et sa période d'exécution en secondes (**period**). La fonction **taskFunction** est la fonction exécutée par chaque thread de tâche périodique. Elle attend la période spécifiée (**sleep**), puis affiche un message indiquant que la tâche a été exécutée.
Créer dans la fonction principale (**main**) deux tâches périodiques (**task1** et **task2**), et lancer des threads pour les exécuter. Les threads ne se terminent pas volontairement, car les tâches sont censées s'exécuter périodiquement de manière continue.

Exercice 6

Créer un programme en langage C qui utilise la bibliothèque pthread pour diviser une tâche de calcul entre plusieurs threads. La tâche consiste à calculer la somme des éléments d'un tableau.

1. Créez une fonction **sum_partial** qui prend comme argument une structure contenant les informations nécessaires pour effectuer le calcul partiel. Cette fonction doit calculer la somme des éléments d'un sous-tableau du tableau global et stocker le résultat dans une variable partagée.
2. Dans la fonction **main**, initialisez un tableau d'entiers avec des valeurs de votre choix.
3. Déclarez une structure qui contient les informations nécessaires pour effectuer le calcul partiel, par exemple, l'indice de début et de fin du sous-tableau.
4. Divisez le tableau en plusieurs parties et créez un thread pour chaque partie. Chaque thread doit appeler la fonction **sum_partial** avec la structure correspondante.

5. Attendez la fin de l'exécution de tous les threads en utilisant la fonction **pthread_join**.
6. Calculez la somme totale à partir des résultats partiels obtenus par chaque thread.
7. Affichez la somme totale.

Assurez-vous que les threads n'accèdent pas simultanément aux mêmes parties du tableau en utilisant, par exemple, des indices différents pour chaque thread.
Utilisez des verrous (**pthread_mutex_t**) pour synchroniser l'accès à la variable partagée.