Université Ibn Zohr
Faculté des Sciences Agadir
Dépt. Informatique

2023/2024

SÉRIE 2 : Programmation Temps Réel
Ordonnancement et synchronisation.

Exercice 1

Considérons un ensemble de tâches devant être exécutées sur un processeur. Chaque tâche est représentée par une paire (temps d'arrivée, temps d'exécution). L'objectif est de déterminer l'ordonnancement optimal des tâches pour minimiser le temps total d'exécution.

Tâches :

- Tâche A : (0, 2)
- Tâche B : (1, 4)
- Tâche C : (3, 3)
- Tâche D : (5, 2)

A. Utilisez l'algorithme de la file d'attente (FIFO - First-In, First-Out) pour ordonner les tâches.

B. Utilisez l'algorithme SJF (Shortest Job First) pour ordonner les tâches.

C. Utilisez l'algorithme de tourniquet (Round Robin) avec une quantum de 2 unités de temps pour ordonner les tâches.

Exercice 2

Considérons le système temps réel suivant :

Tâche	Temps de réveil (r)	Durée d'exécution (C)	Délai critique (D)	Période (P)
T1	0	3	7	20
T2	0	2	4	5
T3	0	2	9	10

1. Calculer la période d'étude de ce système.

2. Vérifier l'acceptabilité d'ordonnancement de ce système et tracer le diagramme de Gantt sur une période d'étude en utilisant RM puis DM.
3. Vérifier l'ordonnançabilité de ce système et tracer le diagramme de Gantt correspondant en utilisant EDF (D3 = 8 au lieu de 9).
4. Étudier et tracer l'ordonnancement généré par LLF (D3 = 8).

Exercice 3

Soit un système temps réel avec 3 tâches ayant les paramètres suivants :

Tâche	Temps de réveil (r)	Durée d'exécution (C)	Délai critique (D)	Période (P)
T1	0	2	4	6
T2	0	**X**	8	8
T3	0	1	3	4

1. Déterminez sous quelle condition sur X ce système est ordonnançable selon RM (least upper bound seulement) puis EDF.
2. Nous souhaitons maintenant introduire m copies de la tâche T2 en parallèle : Déterminez la condition obtenue sur X, en fonction de m, pour que ce nouveau système soit ordonnançable selon EDF ?
3. Pour X=3, tracer le diagramme de Gantt avec l'ordonnanceur RR (Q = 2) ?

Exercice 4

Les trois processus P1, P2 et P3 sont exécutés sur un processeur partagé. Ils peuvent coordonner leur exécution via des sémaphores partagés qui répondent aux procédures standards secure(S) et release(S). L'objectif est d'afficher le mot HELLO. Supposons que l'exécution puisse basculer entre l'un des trois processus à tout moment.

P1:
Afficher("L")

P2:
Afficher("L")
Afficher("O")

P3:
Afficher("H")
Afficher("E")

1. En supposant qu'aucun sémaphore n'est utilisé, pour chacune des séquences de caractères suivantes, spécifiez si le système peut ou non produire cette sortie. (répondez par oui ou non)

LHELO : ……..	LLOHE : ……..	LLOHE : ……..	EHOLL : ……..

Vous souhaitez vous assurer que la séquence HELLO sera imprimée correctement.

2. Précisez le nombre de sémaphores nécessaire pour assurer l'affichage correcte de la séquence. Justifiez votre réponse.
3. Précisez les valeurs initiales pour chacun de vos sémaphores
4. Donnez en pseudo-code l'implémentation des deux primitives Secure(S) et Release(S).
5. Ajoutez les appels Release(S) et Secure(S) manquants au codes des trois processus.

Exercice 5

Considérez un système avec deux threads (Thread 0 et Thread 1) et une variable partagée ***counter***. Les deux threads tentent d'incrémenter ***counter*** simultanément dans une section critique. Utilisez l'algorithme de Peterson pour assurer une exécution sans accrocs de la section critique.

1. Déclarez une variable globale ***counter*** et initialisez-la à zéro.
2. Implémentez les fonctions ***enter_critical_section*** et ***leave_critical_section*** en utilisant l'algorithme de Peterson pour gérer l'accès à la variable ***counter***.
3. Créez deux threads qui s'alternent pour incrémenter la variable ***counter***. Chaque thread doit entrer dans la section critique, incrémenter la variable, puis quitter la section critique.
4. Assurez-vous que l'accès à la variable ***counter*** est correctement synchronisé en utilisant l'algorithme de Peterson.
5. Affichez la valeur finale de la variable ***counter*** après que les deux threads ont terminé leur exécution.
6. Documentez votre code pour expliquer la logique derrière l'utilisation de l'algorithme de Peterson.

Remarque : N'utilisez pas de mécanismes de synchronisation autres que l'algorithme de Peterson.

Exercice 6

Réalisez le même programme que Exercice 5 mais en utilisant l'algorithme de Dekker.

Exercice 7

Donner l'implémentation de l'exercice 4 en langage C utilisant les threads POSIX et les semaphores.

- Assurez-vous que la séquence HELLO est Affichée correctement..
- Le programme principal doit attendre la fin de l'exécution des deux threads avant de se terminer.

Utiliser les threads **POSIX** et les fonctions **pthread_create**, **pthread_join**, **sem_init, sem_destroy, sem_wait** et **sem_post**.