3 - ORDONNANCEMENT

1. RAPPELS

Un algorithme d'ordonnancement (scheduling) permet de choisir parmi un ensemble de tâches prêtes celle qui va occuper le processeur. Cet algorithme peut ou non utiliser des priorités affectées aux tâches.

Avec un algorithme sans réquisition, la tâche élue conserve le processeur jusqu'à sa terminaison : elle ne peut être interrompue ni par l'arrivée d'une autre tâche, ni par une interruption horloge (ex : FCFS, SJN). Avec un algorithme avec réquisition (appelée aussi Préemption), l'arrivée d'une tâche prête plus prioritaire peut interrompre la tâche élue (ex : PSJN). La tâche élue est alors remise en tête de la file des tâches prêtes.

Dans un système en temps partagé, la tâche élue l'est pour (au plus) un quantum de temps.

Un bon algorithme d'ordonnancement doit à tout prix éviter les problèmes de **famine**. Celle-ci survient lorsque les critères utilisés par l'algorithme pour déterminer la tâche élue sont tels que l'une des tâches n'est jamais choisie et ne peut donc pas terminer son exécution.

2. ORDONNANCEMENT EN MODE BATCH

On considère l'ensemble de tâches suivant :

tâche	A	B	C	D	E
date dispo.	0	0	1	5	5
durée	5	4	2	1	2

2.1.

Le processeur est géré selon une stratégie SJN (Shortest Job Next) qui choisit la tâche prête ayant les plus petit d'exécution restant. Représentez l'exécution des tâches sous forme de diagramme. Donnez pour chacune son temps de réponse et son taux de pénalisation. Y a-t-il un risque de famine ?

2.2.

Même question avec une stratégie PSJN (Preemptive SJN).

3. ORDONNANCEMENT EN TEMPS PARTAGÉ

L'algorithme Round-Robin

Dans l'algorithme d'ordonnancement circulaire ou **round-robin**, les tâches sont rangées dans une file unique. Le processeur est donné à la **première** tâche **prête** de la file. La tâche perd le processeur en cas d'entrée/sortie ou quand elle a épuisé son quantum de temps. Elle est alors mise en fin de la file d'attente des tâches. Si une tâche arrive au début de la file dans l'état "bloquée" (attente d'une E/S), elle reste en début de file et on parcourt la file pour trouver une tâche prête. Toute nouvelle tâche est mise à la fin de la file.

On dispose d'un ordinateur ayant une unité d'échange (UE) travaillant en parallèle avec la CPU.

On suppose que l'UE gère plusieurs disques, et que les E/S des tâches se font sur des disques différents.

On considère les tâches suivantes :

	Temps CPU	E/S	Durée E/S
T1	300 ms	aucune	
T2	20 ms	toutes les 10 ms	250 ms
T3	200 ms	aucune	

A t = 20, la tâche T2 fait une entrée/sortie avant de se terminer.

3.1.

Décrire précisément l'évolution du système : *instant*, *nature de l'événement* (commutation, demande d'E/S, fin d'E/S), *tâche élue*, *état de la file*. Donner pour chacune des tâches l'instant où elle termine son exécution. On prendra un quantum de 100 ms, et on supposera que les tâches sont initialement prêtes et rangées dans l'ordre de leur numéro.

3.2.

Quels sont les avantages et les inconvénients de ce mécanisme d'ordonnancement ?

Y a-t-il un risque de famine ? Pourquoi ?

L'algorithme round-robin ne permet pas de prendre en compte le fait que certaines tâches sont plus urgentes que d'autres. On introduit donc des algorithmes d'ordonnancement qui gèrent des priorités éventuelles entre les tâches.

Ordonnancement avec priorités statiques

Dans ce type d'algorithme, chaque tâche dispose à sa création d'une priorité qui conserve la même valeur tout au long de son exécution.

3.3.

Donner un algorithme de gestion des tâches qui utilise ces priorités. En supposant qu'il existe 4 niveaux de priorité numérotés de 0 à 3 (0 étant la plus faible priorité), reprendre la question 3.1 pour cet algorithme en considérant les tâches suivantes :

	Temps CPU	E/S	durée E/S	Priorité
T1	300 ms	aucune		0
T2	20 ms	toutes les 10 ms	250 ms	3
T3	200 ms	aucune		1
T4	40 ms	toutes les 20 ms	180 ms	3
T5	200 ms	aucune		1

NB : on suppose que l'UE gère deux disques, et que les E/S des tâches 2 et 4 se font sur des disques différents.

3.4.

Quels sont les avantages et les inconvénients de ce mécanisme d'ordonnancement ?

Y a-t-il un risque de famine ? Pourquoi ?

Ordonnancement avec priorités dynamiques

Dans ce type d'algorithme, les priorités affectées aux tâches changent en cours d'exécution. Plusieurs algorithmes sont possibles suivant le critère de changement de priorité.

3.5.

Trouvez un algorithme où l'évolution des priorités défavorise les tâches les plus longues.

3.6.

Décrivez en détail le début de l'exécution (t < 900 ms) de cet algorithme pour le modèle de tâches suivant :

	Temps CPU	périodicité
T1	15 ms	toutes les 150 ms
T2	200 ms	toutes les 300 ms
T3	1000 ms	-

On supposera qu'à l'instant 0 la file est T1, T2, T3 et qu'aux multiples de 300 ms, les tâches de type T1 arrivent avant les tâches de type T2. On prendra un quantum de 100 ms.

3.7.

La tâche T3 s'exécutera-t-elle entièrement ? Est-ce toujours le cas ? Donnez un exemple.

3.8.

Trouvez un algorithme où les tâches n'utilisant pas tout leur quantum de temps sont favorisées. Vérifiez que votre algorithme ne crée pas de problème de famine.