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 interruption, 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 et sans recyclage, seule l'arrivée d'une tâche 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.

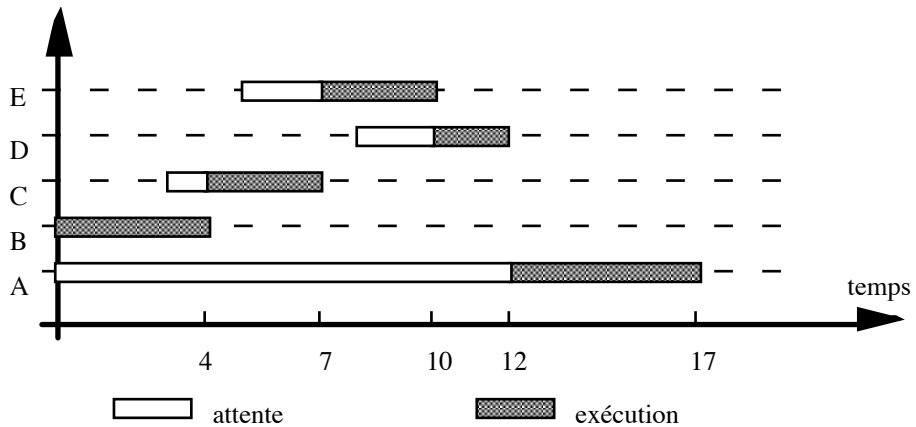
2. ORDONNANCEMENT EN MODE BATCH

On considère l'ensemble de tâches suivant :

tâche	A	B	C	D	E
date dispo.	0	0	3	8	5
durée	5	4	3	2	3

2.1.

Le processeur est géré selon une stratégie SJN (Shortest Job Next). 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 ?



temps réponse = date fin - date arrivée

A : 17 B : 4 C : 4 D : 4 E : 5

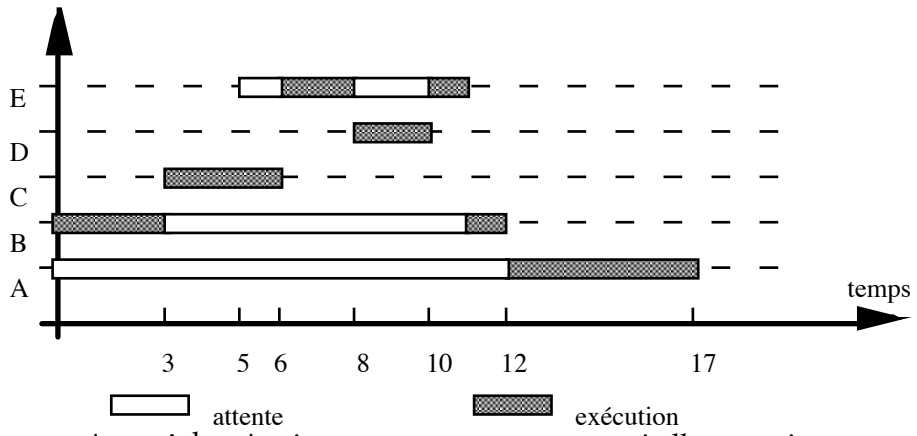
NB : le temps de réponse n'est pas forcément une bonne mesure de performance car il ne tient pas compte du service demandé. Ici B, C et D ont le même temps de réponse, mais pas le même "niveau de satisfaction" puisqu'elles ne demandaient pas le même service.

taux de pénalisation = temps réponse / temps service.
A : 3.4 B : 1 C : 1.33 D : 2 E : 1.66

Pour SJN comme pour PSJN, il y a risque de famine si le processeur est occupé à 100%. En effet, si des tâches courtes arrivent régulièrement, une tâche longue ne s'exécutera jamais. Cette situation est cependant très improbable dans la pratique.

2.2.

Même question avec une stratégie PSJN (Preemptive SJN).



On suppose qu'une tâche n'en interrompt une autre que si elle est strictement plus prioritaire (donc ici, plus courte).

temps de réponse :	A : 17	B : 12	C : 3	D : 2	E : 6
taux de pénalisation :	A : 3.4	B : 3	C : 1	D : 1	E : 2

3. ORDONNANCEMENT EN TEMPS PARTAGE

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 travaillant en parallèle avec la CPU. On considère les tâches suivantes :

	Temps CPU	E/S	Durée E/S
T1	300 ms	aucune	
T2	30 ms	toutes les 10 ms	250 ms
T3	200 ms	aucune	
T4	40 ms	toutes les 20 ms	180 ms

A t = 30 (resp. t = 40) la tâche T2 (resp. T4) 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 qu'elles sont initialement prêtes et rangées dans l'ordre de leur numéro. Quels sont les avantages et les inconvénients de cette technique?

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.

Les hypothèses sur les E/S sont nécessaires pour éviter que l'E/S de la tâche 4 ne commence effectivement qu'après la fin de l'E/S de la tâche 2.

Dans le tableau suivant, 1p = tâche 1 prête et 2b = tâche 2 bloquée.

Instant	Evénement	Tâche Elue	Etat de la file	Commentaire
0	Chargement T1	T1	1p 2p 3p 4p	
100	Commutation	T2	2p 3p 4p 1p	
110	E/S T2, Commutation	T3	3p 4p 1p 2b	
210	Commutation	T4	4p 1p 2b 3p	
230	E/S T4, Commutation	T1	1p 2b 3p 4b	
330	Commutation	T3	2b 3p 4b 1p	
360	fin E/S T2	T3	2p 3p 4b 1p	
410	fin E/S T4	T3	2p 3p 4p 1p	
430	Commutation	T2	2p 4p 1p	fin T3
440	E/S T2, Commutation	T4	4p 1p 2b	
460	E/S T4, Commutation	T1	1p 2b 4b	fin T1
560	Commutation	aucune	2b 4b	
640	fin E/S T4	aucune	2b	fin T4
690	fin E/S T2	T2	2p	
700	E/S T2, Commutation	aucune	2b	
950	fin E/S T2	aucune	vide	fin T2

Avantage : simplicité.

Inconvénient :. ne permet pas de prendre en compte le fait que certaines tâches sont plus urgentes (prioritaires) que d'autres.

3.2.

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

Non, dès qu'une tâche est dans la file des tâches prêtes, elle accède au processeur au bout d'un temps fini. En effet, seules les tâches qui sont devant elle à cet instant s'exécuteront avant elle et elles sont en nombre fini.

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 4.1 pour cet algorithme en considérant les tâches suivantes:

	Temps CPU	E/S	durée E/S	Priorité
T1	300 ms	aucune		0
T2	30 ms	toutes les 10 ms	250 ms	3
T3	200 ms	aucune		1
T4	40 ms	toutes les 20 ms	180 ms	3
T5	300 ms	aucune		1

Pour chaque priorité, il existe une file d'attente où sont rangées les tâches de cette priorité en attente du processeur. On applique l'ordonnancement circulaire à la file non vide de plus haute priorité. Dans le tableau ci-dessous les / séparent les files d'attente de priorités différentes.

Instant	Evénement	Tâche Elue	Etat de la file	Commentaire
0	Chargement T2	T2	2p 4p / / 3p 5p / 1p	
10	E/S T2, Commutation	T4	4p 2b / / 3p 5p / 1p	
30	E/S T4, Commutation	T3	2b 4b / / 3p 5p / 1p	
130	Commutation	T5	2b 4b / / 5p 3p / 1p	
210	fin E/S T4	T5	2b 4p / / 5p 3p / 1p	
230	Commutation	T4	4p 2b / / 3p 5p / 1p	
250	E/S T4, Commutation	T3	2b 4b / / 3p 5p / 1p	
260	fin E/S T2	T3	2p 4b / / 3p 5p / 1p	
350	Commutation	T2	2p 4b / / 5p 3p / 1p	fin T3
360	E/S T2, Commutation	T5	4b 2b / / 5p / 1p	
430	fin E/S T4	T5	4p 2b / / 5p / 1p	fin T4
460	Commutation	T5	2b / / 5p / 1p	
560	Commutation	T1	2b / / / 1p	fin T5
610	fin E/S T2	T1	2p 4b / / 5p / 1p	
660	Commutation	T2	2p / / / 1p	
670	E/S T2, Commutation	T1	2b / / / 1p	
770	Commutation	T1	2b / / / 1p	
870	Commutation	aucune	2b / / /	fin T1
920	fin E/S T2	aucune	vide	fin T2

Avantage : résout le problème posé par round-robin.

Inconvénient : le résout mal puisque cet algorithme crée un problème de famine (cf. question suivante).

3.4.

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

Oui, évidemment. Une tâche de priorité non maximum qui voit arriver régulièrement des tâches de priorité supérieure risque de ne jamais être exécutée.

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.

Quand une tâche est créée, elle a la plus forte priorité (ou sa priorité statique). A chacune de ses exécutions elle descend d'un niveau de priorité. Le nombre de niveaux de priorité étant potentiellement illimité, on peut utiliser une liste dynamique de files d'attente classée suivant les niveaux de priorité. Amélioration : Au niveau le plus haut elle exécute un quantum, au niveau en dessous, deux quanta, 4, 8, 16, ...

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.

Instant	Evénement	Tâche Elue	Etat de la file
0	Chargement 1.0	1.0	1.0 2.0 3.0
15	Commutation	2.0	2.0 3.0
115	Commutation	3.0	3.0 / 2.0
150	new (1.1)	3.0	3.0 1.1 / 2.0
215	Commutation	1.1	1.1 / 2.0 3.0
230	Commutation	2.0	/ 2.0 3.0
300	new (1.2 ; 2.1)	2.0	1.2 2.1 / 2.0 3.0
330	Commutation	1.2	1.2 2.1 / 3.0
345	Commutation	2.1	2.1 / 3.0
445	Commutation	3.0	/ 3.0 2.1
450	new (1.3)	3.0	1.3 / 3.0 2.1
545	Commutation	1.3	1.3 / 2.1 / 3.0
560	Commutation	2.1	/ 2.1 / 3.0
600	new (1.4 ; 2.2)	2.1	1.4 2.2 / 2.1 / 3.0
660	Commutation	1.4	1.4 2.2 / / 3.0
675	Commutation	2.2	2.2 / / 3.0
750	new (1.5)	2.2	2.2 1.5 / / 3.0
775	Commutation	1.5	1.5 / 2.2 / 3.0
790	Commutation	2.2	/ 2.2 / 3.0
890	Commutation	3.0	/ / 3.0

3.7.

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

La tâche T3 est toujours prête, le processeur n'est donc jamais oisif. Sur une période de 600 ms, il est occupé 60 ms par T1 et 400 ms par T2. Il reste donc 140 ms soit plus que la durée nécessaire à l'exécution du quantum de T3.

T3 s'exécutera donc entièrement, et on sait qu'elle sera exécutée au moins une fois par tranche de 600 ms, donc que sa date de fin sera bornée par 10 X600 ms = 6 s.

A partir du moment où T3 est descendue au 3ème niveau de priorité, elle passe toujours après T1 (qui reste toujours au 1er niveau) et T2 (qui ne descend jamais au delà du 2ème). Donc s'il y a toujours des tâches T1 ou T2 prêtes, T3 ne sera plus jamais exécutée.

C'est le cas par exemple si les tâches T1 ont une durée de 50 ms.

3.8.

Trouvez un algorithme où les tâches n'utilisant pas tout leur quantum de temps sont favorisées. Proposez un modèle de tâches qui illustre cet algorithme.

- On peut reprendre l'algorithme précédent en faisant baisser la priorité d'une tâche seulement quand elle a épuisé un quantum et pas à chacune de ses exécutions. Problème : à partir du moment où une tâche a vu sa priorité diminuer, si on n'utilise que la priorité comme critère d'ordonnancement, on a toutes chances de créer un problème de famine.

- Si on crée un tâche avec une priorité basse et que l'on augmente la priorité si elle n'utilise pas tout son quantum, on ne crée pas de problème de famine, mais on risque de défavoriser les tâches courtes qui utilisent tout un quantum : elles doivent attendre la terminaison des tâches qui n'ont utilisé qu'une partie du quantum, même si celles-ci sont longues.

- Idée : faire monter et descendre la priorité. Par exemple :

on ajoute dans le descripteur de tâche une variable q.rest qui mémorise le temps restant dans le quantum.

Une tâche est créée avec q.rest = 0.

Lorsqu'elle perd le processeur,

si elle a utilisé tout son quantum, alors q.rest = 0

sinon q.rest = temps non consommé.

Lors d'une commutation, on choisit la tâche prête avec q.rest maximum.

si q.rest = 0 alors on initialise RH avec le quantum

sinon on l'initialise avec q.rest.

Cet algorithme reste assez simple et ne crée pas de problème de famine.

NB : dans tous les cas, l'algorithme doit rester simple, pour ne pas créer un overhead trop important.

L'exemple d'application est un bon exercice de réflexion pour les étudiants : leur rappeler qu'un bon exemple doit mettre en valeur les qualités et les défauts de l'algorithme.