info502 : Systèmes d'exploitation
TD 1 : ordonnancement et processus

Pierre Hyvernat et Florian Hatat
Pierre.Hyvernat@univ-savoie.fr
Florian.Hatat@univ-savoie.fr

Exercice 1 : Ordonnancement simple (non préemptif)

On considère les huit processus suivants :

processus	temps d'arrivée	durée	priorité
P_1	0	3	1
P_2	$1-\varepsilon$	24	2
P_3	$1-\varepsilon$	8	3
P_4	$7-\varepsilon$	5	3
P_5	$8-\varepsilon$	4	2
P_6	$10-\varepsilon$	2	5
P_7	$15-\varepsilon$	7	5
P_8	$16-\varepsilon$	2	3

Un processus qui arrive au temps $2-\varepsilon$ est un processus qui est présent juste avant le temps 2, mais qui n'était pas présent au temps 1.

Le *temps de traitement* d'un processus est la durée écoulée entre le moment où il est arrivé et le moment où il est effectivement traité.

Le *temps de réponse* d'un processus est la durée écoulée entre le moment où il est arrivé et le moment où il termine sont exécution.

Question 1. Donnez l'ordre d'exécution des processus pour la politique d'ordonnancement "FIFO".

Calculez le temps de traitement moyen et le temps de traitement maximal.

Question 2. Idem pour la politique d'ordonnancement "FIFO avec priorités" (sans réquisition).

Question 3. Idem pour la politique d'ordonnancement "plus court temps d'exécution".

Qu'en pensez-vous ?

Question 4. Discutez sur ce que serait un ordonnancement "optimal" pour ces processus.

Exercice 2 : Ordonnancement préemptif

On reprend les processus de l'exercice précédent...

Question 1. Donner l'ordonnancement des taches en suivant la politique "tourniquet" avec un quantum de temps de deux unités.

Question 2. Idem en utilisant un tourniquet avec réquisition pour les processus de priorité élevée. Quand le processeur est réquisitionné, le processus actif est remis en *tête* de file.

Quel problème est-ce que cette politique peut poser ?

Question 3. En supposant qu'on ai 4 niveaux de priorités : de 0 (priorité élevée) à 3 (priorité basse), écrivez une petite fonction en C pour faire un ordonnanceur multifiles avec feedback :

- on a une file par priorité,
- les nouveaux processus sont placés dans la file 0,
- à la fin du quantum de temps, si le processus en exécution était dans la file n :

. s'il était en exécution, il est replacé en fin de la file de priorité inférieure $(n+1)$
. s'il était bloqué pour des entrées sorties, il est replacé en fin de la file de priorité supérieure $(n-1)$

Si le processus ne peut pas changer de file, on le replace en fin de la même file.

Vous pouvez utiliser un type de données `file` avec les opérations
- `void enfile(file f, donnee d)` qui rajoute la donnée `d` dans la file `f`
- `donnee defile(f)` qui enlève et renvoie la première donnée de la file `f`
- `file nouvelle_file(void)`, `int file_vide(file f)`, `donnee premier(file)`, ...

Remarques :
- on suppose que les nouveau processus sont automatiquement mis dans la file 0,
- votre fonction `ordonnanceur` ne fait que gérer les files de processus ; ce n'est pas elle qui lance ou arrête l'exécution des processus.

Que pensez-vous de cet algorithme ?

Les 4 priorités sont des priorités *internes* (tous les processus peuvent passer par toutes les priorités). Comment peut-on rajouter des priorités *externes*, pour favoriser par exemple les processus `root` et défavoriser les processus lancés avec `nice` ?

Exercice 3 : Les états possibles d'un processus

Lors de sa vie, un processus peut se trouver dans plusieurs états :
- nouveau
- prêt ou bloqué
- en exécution
- terminé

De plus, un processus peut se trouver soit en mémoire principale, soit en mémoire secondaire ("suspendu").

Question 1. Quels sont les états qui peuvent être en mémoire principale ? Et ceux qui peuvent être en mémoire secondaire ?

À quoi correspond chaque état ? Donnez des exemples des situations où un processus se retrouve dans chaque état.

Question 2. Essayez de décrire toutes les transitions possibles entre états.

Commencez par traiter le cas où tous les processus sont en mémoire principale.