Série d'exercices #6

Solution

20 février 2019

6.1 Prémption

Explique la différence entre ordonnancement prémptif et non-préemptif.

6.1 Prémption (Solution)

Explique la différence entre ordonnancement prémptif et non-préemptif.

Solution

Dans un ordonnancement préemptif, le système a toujours le contrôle et peut arrêter «à tout moment» un processus pour donner du temps à un autre processus alors que dans un système non-préemptif, il doit espérer le bon comportement des programmes qui doivent redonner le contrôle au moment qui leur semble oppurtun.
Les systèmes préemptif sont utiles dans les systêmes mult-usagers et interractifs.
Les ordonnanceurs non prémptifs se voient particulièrement utilisés dans les systêmes temps réels où les contraintes de temps de réponses ne peuvent pas être assurées par un ordonnanceur préemptif.

6.2 Simulation

Soit les tache suivante :

Tâche	Burst time	Priorité
P1	2	2
P2	1	1
P3	8	4
P4	4	2
P5	5	3

Présumer qu'elles sont toutes arrivées au temps 0, donc l'ordre indiqué.

- Illustrer dans des tableaux de Gantt l'exécution qui en découle dans le cas d'un ordonnancement de type RR, SJF, FCFS, et priorité non-préemptif.
- Pour chacun des cas, calculer le waiting time moyen et le turnaround time moyen.

6.2 Simulation (Solution)

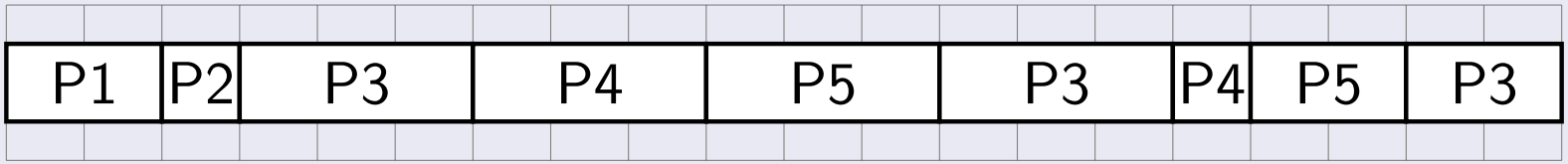
Soit les tache suivante :

Tâche	Burst time	Priorité
P1	2	2
P2	1	1
P3	8	4
P4	4	2
P5	5	3

Présumer qu'elles sont toutes arrivées au temps 0, donc l'ordre indiqué.

Solution

RR (quantum=3)



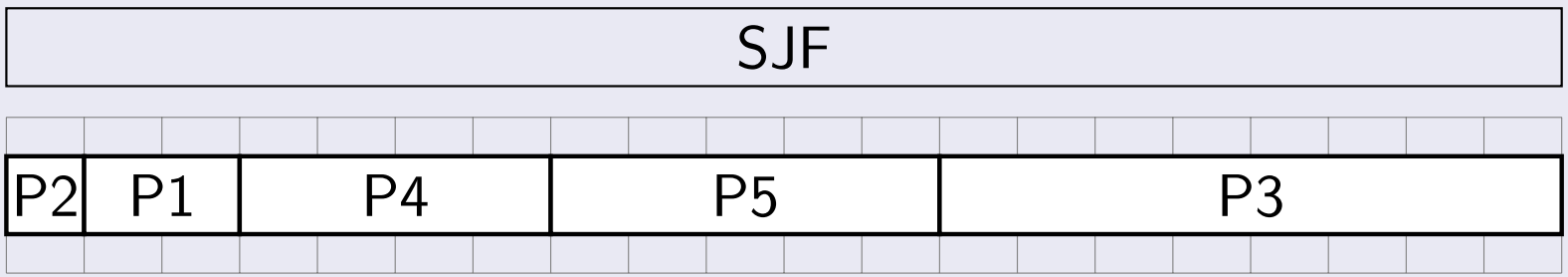
6.2 Simulation (Solution)

Soit les tache suivante :

Tâche	Burst time	Priorité
P1	2	2
P2	1	1
P3	8	4
P4	4	2
P5	5	3

Présumer qu'elles sont toutes arrivées au temps 0, donc l'ordre indiqué.

Solution



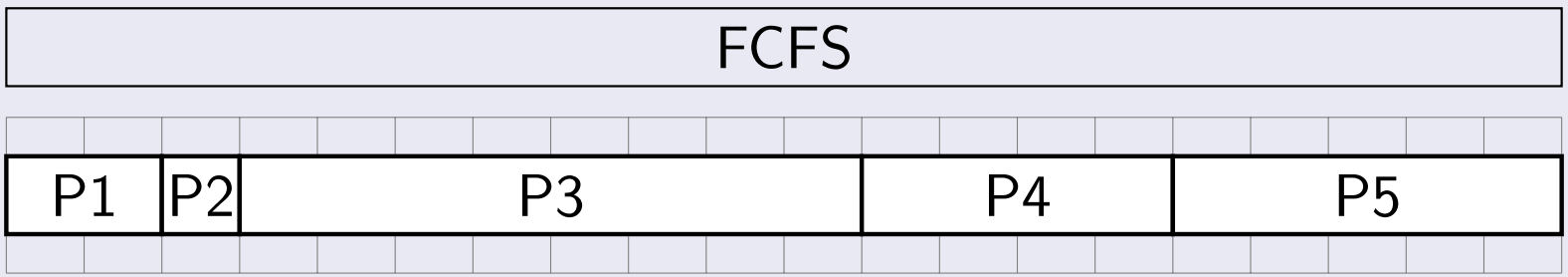
6.2 Simulation (Solution)

Soit les tache suivante :

Tâche	Burst time	Priorité
P1	2	2
P2	1	1
P3	8	4
P4	4	2
P5	5	3

Présumer qu'elles sont toutes arrivées au temps 0, donc l'ordre indiqué.

Solution



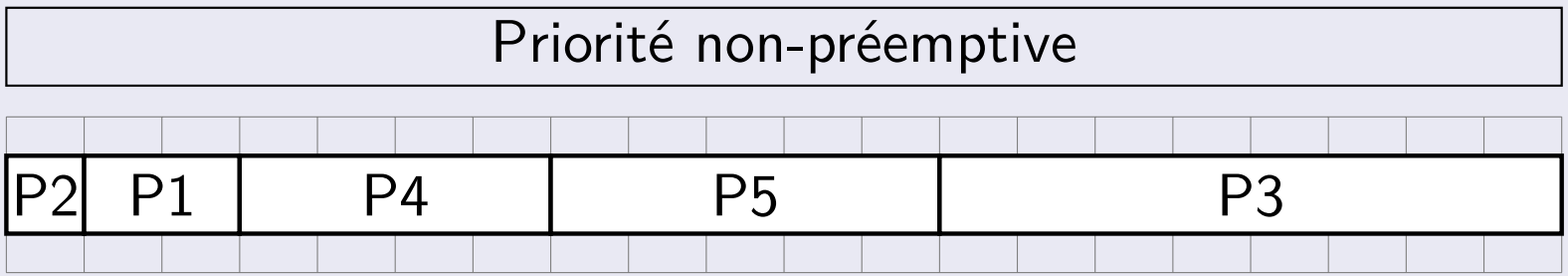
6.2 Simulation (Solution)

Soit les tache suivante :

Tâche	Burst time	Priorité
P1	2	2
P2	1	1
P3	8	4
P4	4	2
P5	5	3

Présumer qu'elles sont toutes arrivées au temps 0, donc l'ordre indiqué.

Solution



6.3 Durée du quantum

Donner les avantages et inconvénients d'avoir un quantum plus court ou plus long. Expliquer quel est l'avantage d'avoir des quantums différents pour les différentes queues d'un ordonnanceur à plusieurs niveaux.

6.3 Durée du quantum (Solution)

Donner les avantages et inconvénients d'avoir un quantum plus court ou plus long. Expliquer quel est l'avantage d'avoir des quantums différents pour les différentes queues d'un ordonnanceur à plusieurs niveaux.

Solution

Plus court : Pratique pour les applications interactive qui n'utilise généralement pas complètement leur quantum (petit burst). Entraine beaucoup de changement de contexte.
Plus long : Pour les application demandant beaucoup de temps processeur sans interruption. Un quantum plus long permet d'éviter des changements de contexte. Tend vers FCFS pour les processus qui prennent moins de temps que le quantum.

6.4 RR multiple

Soit une variante de RR dans laquelle les entrées dans la queue des processus prêts à l'exécution sont des pointeurs vers des PCBs :

1. Quel serait l'effet de placer deux pointers vers le même PCB dans la queue ?
2. Quels seraient les avantages et les incovénients de cet usage ?
3. Comment modifier l'algorithme du RR de base de manière à obtenir le même résultat sans avoir à placer plusieurs copie du même pointeur.

6.4 RR multiple (Solution)

Soit une variante de RR dans laquelle les entrées dans la queue des processus prêts à l'exécution sont des pointeurs vers des PCBs :

1. Quel serait l'effet de placer deux pointers vers le même PCB dans la queue ?
   On double l'exécution du processus, ce qui revient à augmenter la priorité du processus.
2. Quels seraient les avantages et les incovénients de cet usage ?
3. Comment modifier l'algorithme du RR de base de manière à obtenir le même résultat sans avoir à placer plusieurs copie du même pointeur.

6.4 RR multiple (Solution)

Soit une variante de RR dans laquelle les entrées dans la queue des processus prêts à l'exécution sont des pointeurs vers des PCBs :

1. Quel serait l'effet de placer deux pointers vers le même PCB dans la queue ?
   On double l'exécution du processus, ce qui revient à augmenter la priorité du processus.
2. Quels seraient les avantages et les incovénients de cet usage ?
   Certains processus peuvent alors obtenir plus de temps selon leur priorité. Affecte négativement les processus moins prioritaire. L'ordonnanceur n'est plus strictement «fair» (Mais bon, c'est peut-être l'effet désiré).
3. Comment modifier l'algorithme du RR de base de manière à obtenir le même résultat sans avoir à placer plusieurs copie du même pointeur.

6.4 RR multiple (Solution)

Soit une variante de RR dans laquelle les entrées dans la queue des processus prêts à l'exécution sont des pointeurs vers des PCBs :

1. Quel serait l'effet de placer deux pointers vers le même PCB dans la queue ?
   On double l'exécution du processus, ce qui revient à augmenter la priorité du processus.
2. Quels seraient les avantages et les incovénients de cet usage ?
   Certains processus peuvent alors obtenir plus de temps selon leur priorité. Affecte négativement les processus moins prioritaire. L'ordonnanceur n'est plus strictement «fair» (Mais bon, c'est peut-être l'effet désiré).
3. Comment modifier l'algorithme du RR de base de manière à obtenir le même résultat sans avoir à placer plusieurs copie du même pointeur.
   Allouer un temps de quantum en fonction de la priorité et donc permettre plusieurs temps de quantum pour un RR.

6.5 Quantums

Soit un système avec 10 tâches *I/O-bound* et une tâche qui n'utilise pas le périphériques (et donc *CPU-bound*). Chaque tâche *I/O-bound* lance une opération d'entrée sortie après chaque milliseconde de calcul qui prend 10ms pour se terminer. Soit un coût de 0.1ms pour chaque pour chaque changement de contexte. Si on présume que toutes les tâches s'exécutent indéfiniment, décrire l'utilisation du CPU dans le cas d'un ordonnancement de type *round-robin* :

1. Avec un quantum de 1ms.
2. Avec un quantum de 10ms.

6.5 Quantums (Solution)

Soit un système avec 10 tâches *I/O-bound* et une tâche qui n'utilise pas le périphériques (et donc *CPU-bound*). Chaque tâche *I/O-bound* lance une opération d'entrée sortie après chaque milliseconde de calcul qui prend 10ms pour se terminer. Soit un coût de 0.1ms pour chaque pour chaque changement de contexte. Si on présume que toutes les tâches s'exécutent indéfiniment, décrire l'utilisation du CPU dans le cas d'un ordonnancement de type *round-robin* :

1. Avec un quantum de 1ms. Il y a 11 tâches qui vont tous maximiser un quantum de temps.

$$1/1.1 * 100 = 91$$

2. Avec un quantum de 10ms.

6.5 Quantums (Solution)

Soit un système avec 10 tâches *I/O-bound* et une tâche qui n'utilise pas le périphériques (et donc *CPU-bound*). Chaque tâche *I/O-bound* lance une opération d'entrée sortie après chaque milliseconde de calcul qui prend 10ms pour se terminer. Soit un coût de 0.1ms pour chaque pour chaque changement de contexte. Si on présume que toutes les tâches s'exécutent indéfiniment, décrire l'utilisation du CPU dans le cas d'un ordonnancement de type *round-robin* :

1. Avec un quantum de 1ms. Il y a 11 tâches qui vont tous maximiser un quantum de temps.

$$1/1.1 * 100 = 91$$

2. Avec un quantum de 10ms. Il y a 10 *I/O-bound* qui arrêtent après 1 ms et une tâches *CPU-bound* qui utilise tout le quantum de temps.

$$\frac{10 * 1 + 10}{10 * 1.1 + 10.1} * 100 = 94$$

6.6 Multiple niveaux

Soit un algorithme de queues à niveaux multiples, où un processus passe à la queue de priorié supérieure s'il relâche le CPU avant la fin de sont *quantum* et passe dans la queue de priorité inférieure dans le cas inverse. Quels genre de comportement peut-on attendre d'un tel algorithme, en présence de processus variés, certains utilise beaucoup le CPU d'autres seulement les périphériques, d'autres un mélange des deux ?

6.6 Multiple niveaux (Solution)

Soit un algorithme de queues à niveaux multiples, où un processus passe à la queue de priorié supérieure s'il relâche le CPU avant la fin de sont *quantum* et passe dans la queue de priorité inférieure dans le cas inverse. Quels genre de comportement peut-on attendre d'un tel algorithme, en présence de processus variés, certains utilise beaucoup le CPU d'autres seulement les périphériques, d'autres un mélange des deux ?

Solution

Les processus *CPU-bound* auront tendances à se retrouver dans les files de basse priorité alors que ceux I/O-bound se retrouverons dans les files de priorité élevée. Les autres processus se retrouveront dans les files de priorité moyenne.

6.7 Priorités dynamiques

Soit un algorithme d'ordonnancement préemptif basé sur des priorités dynamiques. Quand un processus est en attente du CPU, sa priorité augmente à un rythme α ; quand il est en cours d'exécution, sa priorité augmente à un rythme β. Tous les processus commencent avec une priorité de 0. Les paramètres α et β peuvent être choisis pour obtenir différents résultats.

1. Quel algorithme obtient-on si $\beta > \alpha > 0$?
2. Quel algorithme obtient-on si $\alpha < \beta < 0$?

6.7 Priorités dynamiques (Solution)

Soit un algorithme d'ordonnancement préemptif basé sur des priorités dynamiques. Quand un processus est en attente du CPU, sa priorité augmente à un rythme α ; quand il est en cours d'exécution, sa priorité augmente à un rythme β. Tous les processus commencent avec une priorité de 0. Les paramètres α et β peuvent être choisis pour obtenir différents résultats.

1. Quel algorithme obtient-on si $\beta > \alpha > 0$? FCFS
2. Quel algorithme obtient-on si $\alpha < \beta < 0$?

6.7 Priorités dynamiques (Solution)

Soit un algorithme d'ordonnancement préemptif basé sur des priorités dynamiques. Quand un processus est en attente du CPU, sa priorité augmente à un rythme α ; quand il est en cours d'exécution, sa priorité augmente à un rythme β. Tous les processus commencent avec une priorité de 0. Les paramètres α et β peuvent être choisis pour obtenir différents résultats.

1. Quel algorithme obtient-on si $\beta > \alpha > 0$? FCFS
2. Quel algorithme obtient-on si $\alpha < \beta < 0$? LIFO