TD3: TD Scheduling, synchronisation, communication

Pr Belkhir Abdelkader

Exercice 1 :

L'algorithme de sheduling des processus du système UNIX est de la classe des schedulers du type ROUND ROBIN avec plusieurs niveaux dépendants. Le scheduler alloue le processeur à un processus pour un quantum de temps. Il préempte le processus qui a consommé son quantum de temps et, il le met dans l'un des niveaux de priorités (file). La priorité d'un processus est fonction de l'utilisation récente (temps) du processeur central.

Quels sont les informations nécessaires pour implanter cet algorithme de scheduling?

Exercice 1 :

Soient pri la priorité d'un processus et, cpu l'utilisation récente(temps) du processeur central. Après un quantum de temps, le scheduler calcule pour chaque processus la nouvelle valeur cpu en utilisant l'équation cpu = cpu/2. Après cela, il recalcule pri en utilisant l'équation pri = (cpu/2 + k) ; où k est une constante connue par le système.

Ecrire l'algorithme de scheduling dans le cas où le système gère m niveaux de priorité et le processus élu est celui qui possède la plus basse priorité

Exercice 1 :

- La valeur de la constante k est 60, le quantum de temps est fixé à 1 seconde. L'horloge interrompe le processeur central 60 fois à la seconde. A chaque interruption, on incrémente de 1 la valeur de cpu. Après chaque quantum, on recalcule les valeurs de cpu et pri selon le processus décrit dans l'algorithme de scheduling.
- Montrer que la priorité est dynamique est majorée, trouver alors le nombre de files.

Solution Exercice 1 :

Quels sont les informations nécessaires pour implanter cet algorithme de scheduling?

priorité
Cpu
N nombre de files
Actif
Quantum
Pcb

Solution Exercice 1 :

Montrer que la priorité est dynamique est majorée, trouver alors le nombre de files.

Pour montrer que la formule de changement de priorité ne diverge pas ; il suffit de considérer un seul processus qui va occuper le processeur (la valeur limite est 89 ; grâce à la division entière).

Solution Exercice 1 :

Ecrire l'algorithme de scheduling dans le cas où le système gère m niveaux de priorité et le processus élu est celui qui possède la plus basse priorité

Solution Exercice 1 :

```
Scheduler()
Calcul()  ;        //procédure  de
calcul  à  la  fin  de  chaque
quantum,  sauf  pour  le  premier
appel
Arranger()  ;  //  remettre  chaque
processus   dans   sa   file   de
priorité correspondante
Pour i =60 à 89
Faire
        Si nonvide(Fi)
        Alors q=0 ;
                Actif=  tete(Fi)  ;
//descripteur du processus actif
                Construire     son
mot d'état ;
                Lancer actif
        Fsi
Fait
Attente() ;
```

Exercice 2 :

Le principe de l'ordonnanceur (scheduler) de partage équitable (UNIX version V) est de diviser la communauté des utilisateurs en un ensemble de groupes de partage équitable.
Cependant, le système alloue son temps d'unité centrale proportionnellement à chaque groupe, indifféremment du nombre de processus qu'il y a dans les groupes.
Soient pri la priorité d'un processus et cpu l'utilisation récente (temps) du processeur central et gpe le groupe du processus. L'horloge interrompe le processus en exécution 60 fois pendant son quantum de temps. La procédure de traitement de l'interruption horloge incrémente de 1 la valeur cpu du processus en exécution et le champ gpe de tous les processus du même groupe.
Après chaque quantum, le noyau recalcule les champs pri et gpe de chaque processus selon les formules suivantes:

cpu = cpu/nombre de groupes
gpe = gpe/nombre de groupes
pri = priorité de base + cpu/nombre de groupes + gpe/nombre de groupes

Exercice 2 :

Considérons les trois processus A,B, C de priorité initiale 60. Supposons que le processus A et B soient dans un groupe et que les processus C est dans un autre groupe. La priorité de base est 60 (la plus haute priorité) et le quantum de temps est égal à 1 seconde.

a) Appliquer l'algorithme de partage équitable pour les processus A,B, C pour une période de 6 secondes.

b) Comment peut-on implémenter le champ gpe afin de procéder à une incrémentation efficace de son incrémentation pour tous les processus qui appartiennent au même groupe.

Exercice 2 :

a) Appliquer l'algorithme de partage équitable pour les processus A,B, C pour une période de 6 secondes.

Exercice 2 :

temps	A			B			C		
	Pri	Cpu	gpe	Pri	Cpu	gpe	Pri	Cpu	gpe
0	60	0 1 2 ¦ 60	0 1 2 ¦ 60	60	0	0 1 2 ¦ 60	60	0	0
1	90	30	30	75	0	30	60	0 1 2 ¦ 60	0 1 2 ¦ 60
2	74	15	15 16 17 ¦ 75	67	0 1 2 ¦ 60	15 16 17 ¦ 75	90	30	30
3	81	7	37	93	30	37	75	15 16 17 ¦ 75	15 16 17 ¦ 75
4	70	3 4 5 ¦ 63	18 19 20 ¦ 78	76	15	18 19 20 ¦ 78	96	37	37
5	104	31	39	82	7	39	63	18 19 20 ¦ 78	18 19 20 ¦ 78

Corrigé Exercice 2 :

b) Comment peut-on implémenter le champ gpe afin de procéder à une incrémentation efficace de son incrémentation pour tous les processus qui appartiennent au même groupe.

Gpe pointeur sur une variable partagée appartenant à l'utilisateur propriétaire des processus

Exercice 5 :

Synchronisation des processus

Deux villes A et B sont reliés par une seule voie de chemin de fer. Les trains peuvent circuler dans le même sens de A vers B ou de B vers A. Mais, ils ne peuvent pas circuler dans les sens opposés. On considère deux classes de processus : les trains allant de A vers B (Train AversB) et les trains allant de B vers A (Train BversA). Ces processus se décrivent comme suit :

Train AversB :

Demande d'accès à la voie par A ;

Circulation sur la voie de A vers B;

Sortie de la voie par B;

Train BversA :

Demande d'accès à la voie par B ;

Circulation sur la voie de B vers A;

Sortie de la voie par A;

Exercice 5 :

Ecrire sous forme de commentaires en utilisant les sémaphores, les opérations P et V, les codes de demandes d’accès et de sorties, de façon à ce que les processus respectent les règles de circulation sur la voie unique.

Corrigé Exercice 5 :

Demande d'accès par un train AversB

P(mutex)
Si NbAB =0 alors P(autorisation)
NbAB=NbAB+1 ;
V(mutex) ;

Sortie de la voie par B

P(mutex)
Si NbAB =1 alors V(autorisation)
NbAB=NbAB-1 ;
V(mutex) ;

Corrigé Exercice 5 :

Demande d'accès par un train BversA

P(mutex)
Si NbBA =0 alors P(autorisation)
NbBA=NbBA+1 ;
V(mutex) ;

Sortie de la voie par A

P(mutex)
Si NbBA =1 alors V(autorisation)
NbBA=NbBA-1 ;
V(mutex) ;

Exercice 6 :

Algorithme

Pour réaliser une barrière de synchronisation, il faut disposer de deux sémaphores et d'une variables

- Un sémaphore Mutex (initialise à 1) protégeant la variable
- Un sémaphore Attente (initialisé 0) permettant de mettre en attente les tâches
- Une variable Nb_Attente permettant de compter le nombre de tâches déjà arrivées à la barrière avant de l'ouvrir

Solution Exercice 6 :

```
P(Mutex)
Nb_Attente++
Si Nb_Attente = N
Alors pour i =1 à N-1
        faire V(Attente)
        fait
        Nb_Attente=0
        V(Mutex)
Sinon
       V(Mutex)
        P(Attente)
fsi
```