TD1 : Ordonnancement des processus

Système I

Deuxième année

N.B.

Dans tous les exercices on suppose qu'une valeur faible correspond à une priorité plus élevée. On suppose qu'il n'y a que les processus indiqués dans le système et un seul processeur. Pour départager 2 processus, on fera l'ordonnancement par ordre alphabétique.

★ **Exercice 1:**

5 travaux A, B, C, D et E de durées respectives 10, 6, 2, 4 et 8 secondes, sont soumis à un calculateur dans cet ordre, mais quasi simultanément. Ils ne font pas d'entrées-sorties.

Q 1: Compléter le tableau ci-dessous où DR est le délai de rotation, TA est le temps d'attente et TR est le temps de réponse. Pour le cas avec priorités, on donne $P(A) = 3$, $P(B) = 5$, $P(C) = 2$, $P(D) = 1$, $P(E) = 4$. Pour le Round-Robin, on suppose un quantum de 2 secondes (sans priorités).

Algorithme	FIFO			SJF			Priorities			Round-Robin		
Schedule												
Métriques	DR	TR	TA	DR	TR	TA	DR	TR	TA	DR	TR	TA
A												
B												
C												
D												
E												
Moyenne												

★ **Exercice 2:**

Q 2: Pour les processus du tableau ci-dessous, donnez le schéma d'ordonnancement par priorités dans les 2 cas : non préemptif et préemptif. La date d'arrivée et la durée sont exprimées en cycles.

Processus	Arrivée à	Durée	Priorité
A	0	4	6
B	1	3	5
C	2	3	3
D	3	5	4

★ **Exercice 3:** Sous Linux, soit un processus P qui fait des calculs pendant 60 ms crée 5 processus A, B, C, D et E pratiquement en même temps. Ensuite, il attend la fin de tous ses fils avant de faire 30 ms de calcul et terminer.
Il met A, C et E dans la classe SCHED_RR avec les priorités 4, 2 et 2 respectivement. Il met B et D dans la classe SCHED_FIFO avec les priorités 3 et 1 respectivement.

Q 3: Donner le schéma d'ordonnancement correspondant sachant que :

- A : calcul(100)
- B : calcul(90), usleep(30), calcul(10)
- C : calcul(40), fait sched_setscheduler() pour passer en SCHED_NORMAL, calcul(30)
- D : calcul(70)
- E : calcul(30)

Le quantum de la classe SCHED_RR est 10. L'unité de temps utilisée est le milliseconde. Le schéma d'ordonnancement est à donner selon le format suivant : $X(T_X)$-$Y(T_Y)$- ...où X est le nom du processus et T_X est le temps alloué à ce processus.

★ **Exercice 4:** On considère une machine Linux (1 CPU). Soient 4 processus A, B, C et D crées en même temps avec les valeurs de nice 0, -4, -8 et 4 respectivement. Tous appartiennent à la classe SCHED_NORMAL (CFS) et ont le comportement suivant :

- **A :** 2×(1 ms calcul, 8 ms E/S)
- **B :** 2×(2 ms calcul, 5 ms E/S)
- **C :** 20 ms de calcul
- **D :** 10 ms de calcul

Q 4: Sur quoi CFS se base-t-il pour choisir le prochain processus à exécuter ?

Q 5: Donner le poids attribué à chaque processus et calculer la durée CPU (prorata des processus "runnable") initiale correspondante

no.	Durée	CPU	Ready-Q	I/O-Queue	Justifications/Remarques
0	0		ABCD		
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					

CSF - Completely Fair Scheduler

Le CFS est l'ordonnanceur best-effort de Linux pour la classe SCHED_NORMAL. Il vise à réaliser un partage équitable du processeur, pondéré par des priorités, sans quantum fixe et sans prédiction du futur.

- Chaque processus reçoit une part du processeur proportionnelle à son poids. L'ordonnancement repose uniquement sur le temps CPU déjà consommé, corrigé par un facteur de priorité. Une valeur `nice` plus faible correspond à une priorité plus élevée.
- À chaque valeur `nice` est associé un poids w via la table `prio_to_weight` (listing 1).
- À un instant donné, pour un ensemble de processus *runnables*, la part de processeur reçue par un processus i est :
$$\text{part}_i = \frac{w_i}{\sum_{j \in \text{runnable}} w_j}.$$
- Le CFS introduit une période de référence L, appelée *scheduler latency*. Sur cette période, la slice cible d'un processus i est :
$$\text{slice}_i = L \times \frac{w_i}{\sum_{j \in \text{runnable}} w_j}.$$
Cette slice est recalculée à chaque décision d'ordonnancement.
- Pour éviter des passages processeur trop courts, une granularité minimale est imposée min_granularity
- Le `vruntime` est calculé comme suit :
$$\texttt{vruntime}_i = t_i \times \frac{1024}{w_i}.$$
où t_i est la durée CPU obtenue (cumulative).

Listing 1 – Correspondance nice-weight

```
const int sched_prio_to_weight[40] = {
 /* -20 */     88761,     71755,     56483,     46273,     36291,
 /* -15 */     29154,     23254,     18705,     14949,     11916,
 /* -10 */      9548,      7620,      6100,      4904,      3906,
 /*  -5 */      3121,      2501,      1991,      1586,      1277,
 /*   0 */      1024,       820,       655,       526,       423,
 /*   5 */       335,       272,       215,       172,       137,
 /*  10 */       110,        87,        70,        56,        45,
 /*  15 */        36,        29,        23,        18,        15,
};
```