TD 5 - LIFPCA Programmation Concurrente et Administration Système

—

Ordonnancement

Sylvain Brandel, Yves Caniou, Guillaume Damiand, Meriem Ghali, Laurent Lefèvre, Thibaut Modrzyk, Grégoire Pichon, Alec Sadler, Florence Zara, Jerry Lacmou Zeutouo

Printemps 2024

I Ordonancement

Nous allons utiliser l'implantation POSIX 1003b sur Linux. Il existe 100 niveaux de priorité :

- Le niveau 0 est réservé à SCHED_OTHER et les niveaux de priorité 1 à 99 aux politiques SCHED_FIFO et SCHED_RR.
- Les tâches de priorité 99 sont les tâches de plus forte priorité.
- SCHED_OTHER est dédié à l'ordonnanceur temps partagé.
- Le quantum utilisé par la politique SCHED_RR est de 2 unités de temps.
- Pour SCHED_FIFO et SCHED_RR, lorsqu'une nouvelle tâche arrive, elle est placée en queue (i.e. les tâches sont exécutées dans l'ordre d'arrivée).
- L'ordonnancement est préemptif.

Soit le jeu de tâches apériodiques suivant :

Tâche	Date d'arrivée	Priorité	Durée	Politique
o1	0	0	5	SCHED_OTHER
rr1	7	5	2	SCHED_RR
rr2	10	10	6	SCHED_RR
rr3	6	10	7	SCHED_RR
fifo1	1	10	4	SCHED_FIFO
fifo2	3	10	2	SCHED_FIFO

Q.I.1) - On suppose qu'une fois arrivées, les tâches sont toujours prêtes. Dessinez de l'instant 0 à l'instant 26, l'ordonnancement généré par l'ordonnanceur.

Q.I.2) - Donner le temps réponse de chaque tâche.

II Choix d'ordonancement

Sur le noyau Linux (après le 2.4.4) une option est apparue pour l'ordonnanceur : il s'agit de `child_run_first`. Cette option concerne l'ordonnancement lorsqu'un processus fait un `fork`. Comme son nom l'indique, cette option signifie qu'à la suite du `fork`, c'est le processus fils qui prend la main.

Q.II.1) - Rappelez le comportement des `fork` vis-à-vis de la mémoire et la façon dont le système le gère.

Q.II.2) - Expliquez l'intérêt de l'option `child_run_first` en considérant le programme suivant.

```
1         int pid_t pid_com;
2         int nb_fils = 0;
3         ...
4         pid_com = fork();
5         if (pid_com < 0) {
6           perror("Il y a eu un problème durant le fork()");
7           exit(1);
8         }
9         if (pid_com == 0) {
10          /* Dans le fils */
11          execvp(arg_com1[0], arg_com1);
12          exit(2);
13        } else {
14          nb_fils++;
15          printf("Fils %d de pid %d lancé\n", nb_fils, pid_com);
16          ...
17        }
```

Q.II.3) - Pourtant depuis le noyau Linux 2.6.32 cette option est désactivée par défaut. Proposez une raison.

III Problème de mutex (si le temps le permet)

Un utilisateur exécute la fonction suivante plusieurs fois depuis plusieurs threads différents :

```
1     std::mutex m;
2     int get_tache(lt *l)
3     {
4       m.lock();
5       if (l->debut == l->nbtaches) {
6         return -1;
7       }
8       int res = l->tab[l->debut];
9       l->debut++;
10
11      m.unlock();
12      return res;
13    }
```

Après un certain temps, le programme se bloque. Expliquez le problème. Comment le corriger ?