LO41 : Rapport de projet

IMPLEMENTATION DU ROUND ROBIN

CARTERET Thomas et BENIS Antoine | 06/01/2020

Mode d'emploi

En premier lieu, il faut choisir le nombre de threads qui seront utilisés lors de l'ordonnancement. Le numéro, la date de soumission, le temps d'exécution et la priorité apparaitra sous forme de liste et les valeurs de ces différentes propriétés seront aléatoires et non modifiable.

Ensuite, il faut choisir le pourcentage d'appel de chacune des priorités dans la table d'allocation. Un set de pourcentage est proposé dès le départ. On a ensuite le choix de garder ce set en appuyant sur « n » ou de le changer en appuyant sur « y ». Si on a 10 queues (10 niveaux de priorités), on peut se permettre de garder le set de base. Sinon, on doit obligatoirement rentrer manuellement le pourcentage d'appel de chaque niveau de priorité en ayant pour seule contrainte que la somme de ces pourcentages soit égale à 100.

Cas spécial : afin d'obtenir les mêmes résultats du TD sur l'ordonnancement, on choisit de prendre les valeurs du TD. S'il y a 7 threads de créés alors le numéro, la date de soumission, le temps d'exécution et la priorité sont déjà configurés et non modifiable. De plus, les pourcentages d'appel de chaque niveau de priorité dans la table d'allocation sont aussi fixés pour ce cas, cependant, on peut les modifier. (Attention : si l'on choisit ce cas il faut régler manuellement de nombre de file d'attente à 4 au lieu de 11 à l'aide la variable NUMBEROFFILESPRIORIRY dans le define en tête de programme).

Résultats obtenus :

Pour l'initialisation :

```
1   Init Queues :
2
3   6 -> 5 -> 4 -> 3 -> 2 -> END
4   Queue 0 :
5
6   0 -> END
7   Queue 1 :
8
9   1 -> END
10  Queue 2 :
11
12  END
13  Queue 3 :
14
15  END
```

Ligne 3 : Affichage de la liste d'attente : liste des threads qui ont une date de soumission non nulle.

Ligne 4 à 15 : Liste de chaque niveau de priorité (queue) avec pour chacun de ces niveaux, la liste des thread contenus.

Déroulement à un certain quantum de temps :

```
1  CURRENT QUANTUM 6
2  Queue empty
3  Thread Elected : 0
4  Waitenteringqueue is empty END
5  CURRENT QUEUE 1 CURRENT THREAD 0
6  Thread 0 se réveil
7  Thread 0 s'endort
8
9  Queue 0 :
10 END
11 Queue 1 :
12 5 -> END
13 Queue 2 :
14 0 -> 2 -> 1 -> END
15 Queue 3 :
16 4 -> END
```

Ligne 1 : Numéro du quantum de temps

Ligne 2 : Cas où la queue, initialement appelé par la table d'allocation, est vide. On lit donc la ligne suivant et ainsi de suite jusqu'à ce qu'il y ait un thread. Le nombre de « Queue empty » affiché correspond au nombre de fois où le programme à incrémenter l'appel d'une queue. Dans cet exemple, la queue 0 est vide, donc le programme va appeler le thread se trouvant dans la queue 1.

Ligne 3 : Numéro du thread élu.

Ligne 4 : Affiche la liste des threads en attente. Dans ce cas, la liste est vide.

Ligne 5 : Numéro de la queue dans laquelle le thread est élu et le numéro du thread élu.

Ligne de 9 à 16 : Liste de chaque niveau de priorité (queue) avec pour chacun de ces niveaux, la liste des thread contenus.

Résumé des structures et fonctions

threadProperties : défini la structure du thread (numéro, date de soumission, temps d'exécution, priorité et éléments nécessaires à son fonctionnement).

Element : structure qui comprend un pointeur sur la structure de l'élément suivant et la structure threadProperties

Queue : pointeur sur la structure Element

*thread : fonction qu'execute le thread, elle comprend son execution ainsi que la gestion de l'endormissement et de la mort du thread.

queryNbThread : menu qui initialise le nombre de threads utilisés

initProcess : fonction qui va associer à chaque thread, une date de soumission, un temps d'execution et une priorité aléatoire (petite variante, si 7 threads ont été créée, cette fonction va lire les valeurs qu'on a rentré manuellement)

initProcessAllocTable : fonction qui va créer la table d'allocation et permettre à l'utilisateur de rentrer les pourcentages d'appel de chaque thread manuellement (à défaut un set de données)(petite variante, si 7 threads ont été créée, cette fonction va lire un set de données)

insertElement : fonction qui insert un thread en tête de liste

getOldestElement : fonction qui va supprimé s'il en reste un thread en queue de liste et le retourner

displayQueue : fonction qui permet l'affichage des threads contenu dans un niveau de priorité (liste de thread)

executeWaitingQueue : fonction qui va gérer la file d'attente, si au départ, la date d'exécution du thread est nulle, celui-ci va directement être ajouté dans sa file de priorité correspondante, sinon il va être ajouté dans la file d'attente. Ensuite, au fur et à mesure du temps, le quantum de temps va être incrémenter et executeWaitingQueue va décrémenter la date d'exécution de tous les threads contenus dans cette liste. Dès qu'un ou plusieurs threads ont une date d'exécution nulle, ils vont être sortis de la liste, sinon ils y restent.

Difficultés rencontrées et améliorations restantes

Le programme est donc actuellement fonctionnel. Nous avons plusieurs jeux de données qui ont été vérifiée dont celui étudié en TD.

Cependant, nous avons rencontré de nombreuses difficultés notamment liées aux pointeurs de structures. En effet, le code peut être assez indigeste avec les notions et il n'était facile de s'y faire.

De plus, nous avons eu des difficultés face à des erreurs assez comprises, on peut citer notamment des erreurs liées aux fonctions qui étaient après le main et déclarée au-dessus. En effet, certaines devaient être déclarées et pas d'autres de manière assez aléatoire. Mais le fait de les positionner à la fin à corriger bien des

erreurs de compilation. Les scanf ont aussi été un assez gros problème, leur comportement variait dans l'exécution du programme. Etc..

Bilan personnel

Ce projet a été très enrichissant à plusieurs niveaux.

Premièrement, d'un point de vue de la programmation système, il nous a permis de nous familiariser et vraiment assimiler les notions vues en cours d'autant plus que nous nous sommes principalement focalisés sur les processus et ici nous avons pu travailler avec des threads.

Dans un second temps, il a été très profitable d'un point de vue d'expérience de programmation. Venant de DUT GEII et CPGE nous avions tout deux assez peu d'expérience en la matière et surtout en système. Ça a aussi été l'occasion de se familiariser un peu plus avec Linux.

Le dernier point qui est peut-être le plus important est que ça a été une très bonne occasion pour apprendre à gérer plusieurs projets puisque nous en avions tout deux deux autres en parallèle. C'était aussi une opportunité pour apprendre à trouver par soi-même des solutions en contournant et en explorant toutes les possibilités qu'offrent le C ainsi qu'en découvrir d'autres extérieurs au cours (liste chainées, expressions régulières et plein d'astuces).