ARA

Projet Peersim

Robin Blottiere-Mayo

Pierre Oumeddour

Professeur : Jonathan Lejeune

25 janvier 2019

Exercice 1

1.1 Question 1 :

L'algorithme de verrouillage utilisé dans la classe Application est un algorithme dérivé de celui de Naimi-Tréhel utilisant la notion de jeton.

Ainsi tous les sites se partagent le même jeton et seul le site possédant le jeton peut entrer en section critique. Dans la classe Application, les constantes initial_owner et nil servent à discriminer le site possédant le jeton des autres.

Chaque noeud maintient sa propre liste de noeuds en attente de section critique (variable next) et son compteur global de sections critiques (variable global_counter). Il possède également l'adresse du noeud dont il pense qu'il possède le jeton (variable last) ainsi que le nombre de sections critiques qu'il a lui-même effectuées (variable nb_cs). Si un noeud veut une section critique, il envoie une demande pour avoir le jeton puis il se met en attente.

Quand un noeud sort de section critique, il envoie le jeton au dernier noeud de la liste des noeuds demandant la section critique avec les informations qu'il possède du nombre de sections critiques réalisées et des noeuds en attente de section critique.

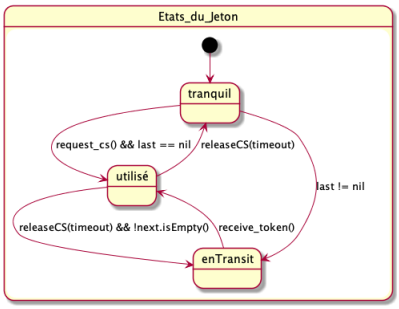
Lorsqu'un noeud reçoit le jeton, il met à jour son compteur de sections critiques avec la valeur envoyée par le dernier noeud ayant été en section critique. Il compare également sa liste de noeuds en attente avec celle qu'il a reçue. Si des noeuds se trouvent dans les deux listes mais pas dans le même ordre, l'ordre envoyé par le dernier noeud en section critique est maintenu. Si des demandes sont absentes de la liste reçue, elles sont ajoutées en fin de liste.

1.2 Question 2 :

On considère trois états du jeton :

1. Un noeud possède le jeton mais ne s'en sert pas (état tranquille).
2. Un noeud possède le jeton et est en section critique (état utilisé).
3. Le jeton est en transit (état enTransit).

tranquille → utilisé : la section critique est demandée et personne ne requiert le jeton.
tranquille → enTransit : un noeud requiert le jeton.
utilisé → tranquille : la section critique se termine.
utilisé → enTransit : releaseCS(timeout) & !next.isEmpty()
enTransit → utilisé : receive_token()



1.3 Question 3 :

α étant le temps moyen qu'un processus passe en section critique et β le temps passé entre les sections critiques.

Le ratio $\rho = \alpha \ / \ \beta$ représente la charge du réseau.

Plus ce ratio est élevé plus le réseau est ralenti.

1.4 Question 4 :

1.5 Question 5 :

Le fait que le temps de transmission moyen soit supérieur au temps moyen passé en section critique augmente la charge du réseau. Le temps de transport augmentant, le temps ou le jeton est possédé par un noeud est plus faible.

Cela ne change pas fondamentalement les résultats en dehors de rendre le système plus lent et de diminuer le nombre de section critique pour un temps d'exécution égale.

Dans le détail, on voit que les premiers instants du fonctionnement du système le noeud ayant le jeton peut exécuter plusieurs fois sa section critique avant de recevoir une requête et que la file des noeuds demandant l'accès à la section critique mettra plus de temps à se remplir. Cependant après que tous les noeuds aient exécuté au moins une fois leur section critique, le temps passé en section critique et le temps de transmission du message s'additionnant rendront la suite de l'exécution semblable à ce qu'elle aurait été avec un temps de transmission moyen plus faible que le temps passé en section critique.

Exercice 2

+ Inline
Avantages :
- Permet d'économiser le coût d'un appel de fonction.
- Permet des optimisations impossible avec un appel.
Inconvénient :
- Augmente la taille du code, donc les cache miss.
- Utilisations accrue des registres pour les paramètres.

Propre à gcc mais pas POSIX :
likely(), unlikely(). Fait deux sauts dans le cas le moins probable et 0 dans le cas le plus probable.

asmlinkage : Précède tous les appel système. Force le passage par la pile système.

le struct hack (c'est vraiment très important !).

char oui[4] est une constante symbolique. Il n'y a pas de case d'adresse. Un pointeur de fonction est aussi une constante symbolique.
=> &oui == oui c'est un identificateur de vecteur.

+ La fonction vmalloc() peut être utilisée pour obtenir des zones mémoire continues dans l'espace d'adresse virtuel même si les pages ne sont pas en continues en mémoire physique.

+ La fonction ioremap() ne fait pas d'allocation, mais fait correspondre le segment donné en mémoire physique dans l'espace d'adressage virtuel.

Exercice 3