L’algorithme d’exclusion mutuelle de Lamport

L’avantage de cet algorithme d’exclusion mutuelle est qu’en cas d’absence de demande de section critique, le système finit par atteindre un état dans lequel plus aucun message ne circule.

Hypothèses :

- Processus et canaux asynchrones.
- Canaux FIFO.
- Processus avec identit´e.
- Pas de faute.
- Topologie en clique. (Au moins deux nœuds.)
- Multi-initiateurs

Principe : L’algorithme est défini par les cinq règles suivantes. Chacune de ces règles sera considérée comme un unique évènement.

1. Pour demander la section critique, un processus p envoie le message <demande, t_m> à tous les autres processus où t_m est la valeur de l’horloge de p au moment de l’envoi et met ce message dans sa liste Requête.
2. Lorsqu’un processus p reçoit un message <demande, tm> du processus q, il met ce message dans sa liste Requête et envoie un accusé réception Ack à q tagué avec sa valeur d’horloge.
3. A la fin de sa section critique, le processus p envoie un message <sortie, tm> à tous les autres processus où tm est la valeur de l’horloge de p au moment de l’envoi.
4. Lorsqu’un processus p reçoit un message <sortie, tmi> du processus q, il supprime toute requête de q présente dans sa liste Requête.
5. Un processus p accède à sa section critique quand il y a une requête (tm, p) dans sa liste Requête telle que :
    a. (tm, p) est plus petite que toutes les autres requêtes de Requête suivant l’ordre.
    b. p a reçu un message tagué avec une valeur d’horloge supérieure à tm de chacun des autres processus.

Idées de solutions

Les règles précédentes sont implantées dans l’algorithme 5. Dans cet algorithme, « _ » dénote une valeur quelconque.

Preuve Dans l’algorithme 5, les cinq évènements définis précédemment implantent les règles d’incrémentation de l’horloge. Donc, les conditions découlant de l’horloge sont vérifiées par les horloges locales Horloge[p]p de chacun des processus. D’où la condition CH est vérifiée. Dans la suite, nous identifions chaque demande de section critique d par le couple (d.ID, d.H) où d.ID est le processus demandeur et d.H est la valeur de Horloge[d.ID] dans la m´mémoire locale de d.ID au moment de l’arrivée de la demande.

Lemme 12. Seul un nombre fini d’autres demandes peut accéder à la section critique avant qu’une demande particulière y accède.

Preuve. Soit d une demande de section critique. Lorsque d.ID reçoit cette demande, il met d dans sa liste Requêtes, envoie <demande, d.H> à tous les autres processus. Les messages demande arrivent en un temps fini, à la suite de quoi, chaque processus $q \neq p$ insère d dans sa liste Requêtes et affecte son horloge à une valeur strictement supérieure à d.H. Ensuite, tout processus retire d de sa liste Requêtes uniquement après que p ait accédé à la section critique. Puisque les horloges sont

strictement croissantes, toutes les nouvelles demandes ne pourront accéder à la section critique qu'un fois que d aura été satisfaite. Ainsi, les seules demandes déjà présentes dans la liste Requêtes au moment de l'insertion de d (au plus n − 1) peuvent être satisfaite avant d.

Lemme 13. S'il existe des demandes non satisfaites, une demande finit par être satisfaite.

Preuve. Les demandes sont totalement ordonnées par les principes de causalité. Soit d l'unique demande non-satisfaite minimum. Lorsque d.ID a reçu cette demande, il a mis d dans sa liste Requêtes, et a envoyé <demande, d.H> `a tous les autres processus. Toutes les demandes satisfaites finissent par être supprimées de la liste Requêtes de d.ID car leur processus demandeur envoie un message de sortie `a d.ID `a la sortie de la section critique. Ainsi, d finit par être la demande la plus petite dans la liste Requêtes de d.ID. Enfin, tous les autres processus accusent réception du message <demande, d.H> avec un message Ack tagué avec une valeur strictement supérieure à d.H. Par suite, d.ID finit par accéder à la section critique.

D'après les deux lemmes précédents, nous avons le corollaire suivant.

Corollaire 3 (Vivacité). Tout processus demandeur accède à la section critique en temps fini.

Lemme 14 (Sureté). Jamais plus d'un processus n'est en section critique.

Preuve. (Par contradiction) Supposons que deux demandes d et d'accèdent à la section critique concurremment, c'est-à-dire, d (resp. d') entre en section critique avant que d' (resp. d) en sorte.

Les demandes sont totalement ordonnée par les estampilles. Sans perte de généralité supposons que d précède d'. Lorsque d' accède à la section critique, d'.ID a reçu, en particulier, un message de d.ID tagué avec une valeur tm > d'.H. Or, quand d.ID est devenu demandeur, il a envoyé, en particulier, un message <demande, d.H> à d'.ID. Puisque, d.H ≤ d'.H < Tm, et que les liens sont FIFO, on peut déduire que d 0 .ID a reçu <demande, d.H> avant le message tagué avec tm. Donc, d'.ID a inséré la demande d dans sa liste Requêtes avant d'accéder à la section critique. Puisque le demande d'accès en section critique d' est autorisée avant que d ne soit terminé. La demande d est toujours dans la liste Requêtes de d'.ID au moment où la demande d 0 est autorisée à accéder à la section critique, contradiction.

D'après le lemme et le corollaire précèdent, nous avons :

Théorème 5. L'algorithme 5 résout l'exclusion mutuelle dans une clique.

Complexité. Nombre de messages par demande : 3(n − 1)

Algorithme 5 Exclusion mutuelle de Lamport pour tout processus p

Initialisation

1. Requêtes ← ∅
2. Pour tout processus q faire
    a. Horloge[q] ← 0
3. Fin Pour

Sur demande

1. Requêtes ← Requêtes ∪ {(p, Horloge[p])}
2. Demandeur ← vrai
3. Envoyer <Demande, Horloge[p]>à tous les autres processus
4. Acces()

5. Horloge[p] ← Horloge[p] + 1

Réception de <Demande, T> de q

1. Horloge[q] ← T
2. Horloge[p] ← max(Horloge[p], Horloge[q] + 1)
3. Requêtes ← Requêtes ∪ {(q, T)}
4. Envoyer <Ack, Horloge[p]> à q
5. Acces()
6. Horloge[p] ← Horloge[p] + 1

Réception de <Ack, T> de q

1. Horloge[q] ← T
2. Horloge[p] ← max(Horloge[p], Horloge[q] + 1)
3. Acces()
4. Horloge[p] ← Horloge[p] + 1

Réception de <Sortie, T> de q

1. Horloge[q] ← T
2. Horloge[p] ← max(Horloge[p], Horloge[q] + 1)
3. Requêtes ← Requêtes \ {(q, _)}
4. Acces()
5. Horloge[p] ← Horloge[p] + 1

Fonction Acces()

1. Si Demandeur alors
    a. Si (p, T) = min(Requêtes) ∧ pour tout processus q ≠ p, Horloge[q] > T alors
        i. SC
        ii. Requêtes ← Requêtes \ {(p, _)}
        iii. Demandeur ← faux
        iv. Envoyer <Sortie, Horloge[p]> à tous les autres processus
    b. Fin Si
2. Fin Si