[RICM4 - AD] Février 2015
Binôme : Mammar - Barthelemy

Algorithmique Distribuée - DM

Question 1 :

Implémentation de l'algorithme :

- AskMessage.java, ReplyMessage.java : sont les deux types de message circulant sur l'anneau
- AllNode.java : est un noeud de l'anneau
- CustomGlobal.java : implémente nos deux anneaux (meilleur et pire cas) démonstratifs des propriétés justifiée aux questions ci-dessous

L'algorithme est implémenté dans la méthode handleMessages(inbox) de la classe AllNode.java

Initialement tous les noeuds sont jaunes.
Un noeud Q est rouge signifie que le programme traite un message reçu par ce noeud . Une sortie sur la console nous informe du traitement du message :

- un message de type Reply est envoyé vers l'émetteur P du message car msg(L) == 0 et que id(P) > id(Q)
- le message est relayé au noeud voisin car msg(L) > 0 et que id(P) > id(Q)
- le message est ignoré (détruit) car id(P) < id(Q)

Le noeud élu est en cyan

Question 2:

Soit P_{max} le noeud d'identifiant max.
Cet algorithme a plusieurs traits de similitude à celui du Leader optimisé vu en cours. Il consiste à élire un noeud d'identifiant le plus grand.
Similarités :

- Les hypothèses restent les mêmes (id unique, pas de perte de message, …)
- Les messages sont filtrés par le noeud élu (ici P_{max})
- Un noeud s'auto élit dès la réception de son propre message (ici d'au moins un des deux messages émis sur les deux côtés G et D).

De là découle la propriété de sûreté qui dit qu'au plus un noeud (ici P_{max}) peut être élu (recevra son propre message) car :

- les id sont uniques (sinon deux noeuds candidats de même id seraient tous deux élu)
- l'ensemble des identifiants ont une relation d'ordre total (pour déterminer le max)
- $\forall\, msg_{id' \neq idmax}$ il sera au pire éliminé par P_{max}

La propriété de vivacité qui dit qu'au moins un noeud est elu :

- Tous les initiateurs envoient leur id, donc idmax aussi
- L'initiateur dont l'id est à idmax reçoit toujours une réponse (le temps d'envoi des messages étant fini)
- P_{max} recevra son propre message en temps fini, après $log_2(n)$ phases (n nombre de noeuds de l'anneau)

Différences :

- on élit le noeud d'ID le plus grand au lieu du plus petit
- anneau bi directionel
- tous les noeuds sont candidats

Question 3:

Évaluons la propriété de sûreté :

Rappel :

Initialement tous les noeuds sont jaunes.

Un noeud Q est rouge signifie que le programme traite un message reçu par ce noeud . Une sortie sur la console nous informe du traitement du message :

- *un message de type Reply est envoyé vers l'émetteur P du message car msg(L) == 0 et que id(P) > id(Q)*
- *le message est relayé au noeud voisin car msg(L) > 0 et que id(P) > id(Q)*
- *le message est ignoré (détruit) car id(P) < id(Q)*

Test 1: (avec 15 noeuds)
On marque en bleu cyan tous les noeuds dont la variable résultat est affectée à vrai.
A l'issue de ce test on voit bien qu'un seul noeud est en cyan et c'est celui d'identifiant max (ici le noeud 14)

Test 2: (pire cas, tel que décrit en question 5)
Comme dans le test précédent on marque en bleu cyan tous les noeuds dont la variable résultat est affectée à vrai.
A l'issue de ce test un seul noeud est coloré en cyan d'où la propriété de sûreté

Question 4:

a. La variable "L" peut être assimilée à un TimeToLive du message, puisque sa valeur est décrémentée à chaque transition par un noeud. Elle croit de façon exponentielle (ligne 35 : $L \leftarrow L*2$) à chaque phase : émission_de_deux_messages_ASK -- Attente_des_acquittements (Reply)

   La variable "n" permet d'attendre la réception des deux acquittements des messages

ASK envoyés préalablement avant de renvoyer des messages ASK.

b. Un processus P d'identifiant idP reçoit ses deux acquittements si, de gauche comme de droite, son identifiant est le plus grand sur un segment de noeuds de taille L.
En effet, il suffirait qu'un noeud Q du flanc gauche ou droit de P ait un identifiant supérieur à idP pour que la variable *résultat* de Q passe à false, et qu'aucun paquet Reply ne soit émis vers P.
P resterait ainsi en attente d'un acquittement qui n'aura jamais été émis, et serait donc considéré comme éliminé.

c. Le programme termine pour une valeur de i égale à l'arrondi supérieur de $log_2(n)$.
Avec n = nombre de noeuds de l'anneau.

d. Justifions les propriétés de sûreté et vivacité :
Sûreté :
Supposons que deux noeuds id1 et id2 soient élus avec id1>id2. Cela implique que le noeud id1 a reçu le message <ASK,id2,x> et a renvoyé soit <REPLY,id2> en acquittement soit <ASK,id2,x-1> à la suite de l'anneau.
Cependant, ces messages ne peuvent être envoyés que si id2>id1. Nous avons donc une contradiction: deux messages ne peuvent pas être tous deux élus.

Vivacité :
Prenons le cas du noeud idmax. Pour chaque message <ASK,idmax,L-1> envoyé, il reçoit en temps fini un acquittement <REPLY,idmax> lorsque L<n. Cela se vérifie par le fait que pour chaque noeud recevant <ASK,idmax,x> le message <ASK,idmax,x-1> est transité si x>0, sinon <REPLY,idmax> est renvoyé en temps fini vers le noeud idmax.
Lorsque les deux réponses sont arrivées, la phase est terminée, et $L \leftarrow L*2$. Le processus est réitéré jusqu'à la phase i où $2^i \geq n$, alors le message <ASK,idmax,2^i-1> est envoyé des deux cotés de l'anneau, et le noeud idmax reçoit en temps fini deux messages <ASK,idmax,x> et la variable résultat du noeud idmax est placée à vrai

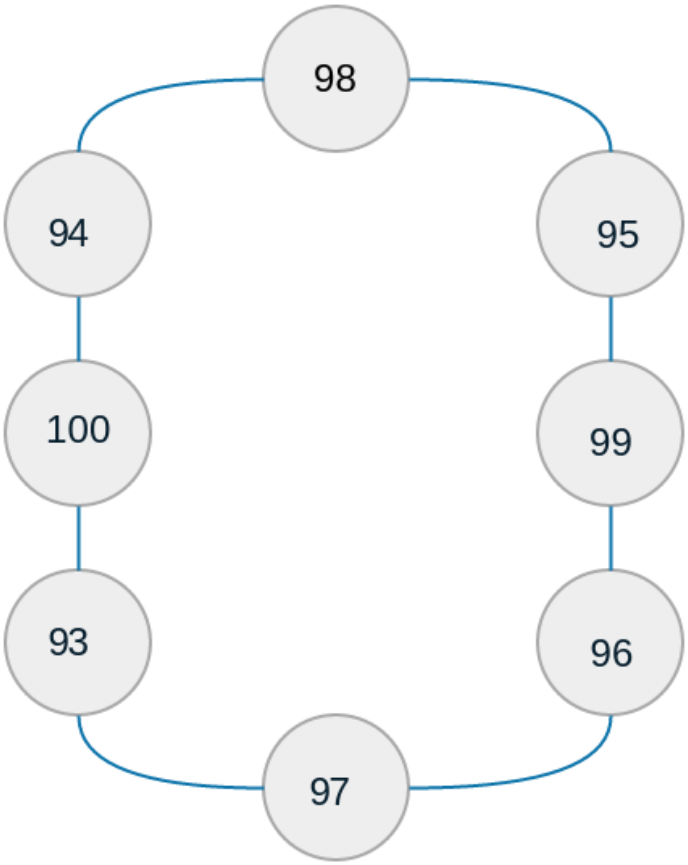
Question 5 :

Soit n le nombre de noeuds.
En nombre de messages, l'algorithme est en complexité *O(n log(n))* :

Meilleur cas:

Tous les noeuds sauf le noeud élu sont éliminés dès la première phase; les id des noeuds sont en ordonnés (croissant ou décroissant). Les noeuds éliminés reçoivent une seule réponse.
Nombre de messages de la phase 1:
$2n$ (nombre de messages initiaux) $+n$ (Une réponse par noeud)
Nous avons donc $3n$ messages dans cette première phase.
Pour chaque phase i non terminale, nous avons 2^{i+2} messages transités (2^{i+1} messages à l'aller (ask) et au retour (reply), et ce pour les deux directions d'où au final 2^{i+2})
Pour la phase terminale, $2n$ messages transitent.
Nous avons de l'ordre de $log_2(n)$ phases: à chaque phase, $L \leftarrow L*2$ et lorsque $L>n$, il s'agit de la dernière phase
Si on considère 2^{i+2} de l'ordre de n, nous avons donc de l'ordre de $O(n*log(n))$ messages envoyés (les messages de début et de fin étant de l'ordre de n, $n*log(n)$ englobe le tout)

Pire cas:
Chaque phase divise par deux le nombre de noeuds non éliminés. Le nombre de messages envoyé à chaque phase est alors de $3n$. Nous avons toujours $log(n)$ phases, ainsi, la complexité est de l'ordre de $O(n*log(n))$



A chaque phase non terminale le nombre de messages envoyé est 3n

Phase 1 : L = 1

2*n : 2 * n * L msg ASK emis par chaque noeud sur un segment L
n : 2 * (n/2) * L msg REPLY car la topologie révèle que la moitié est éliminée

Phase 2 : L = 2

2*n : 2 * (n/2) * L msg ASK sur les n/2 noeuds restants
n : 2 * (n/4) * L msg REPLY sur les n/4 noeuds non éliminés

Phase 3 : L = 4
2*n : 2 * (n/4) * L msg ASK sur les n/4 noeuds restants
n : 2 * (n/8) * L msg REPLY sur la moitié non éliminée

Phase 4 (finale) : L = 8>=n
2*n : 2 * 1 * n msg ASK: le noeud élu envoie 2 messages parcourant chaque noeud

Question 6 :

Voici le déroulement des rondes permettant de déterminer la complexité en temps de l'algorithme:
0->1: Tous les processus sont prêts à s'exécuter au début font un pas de calcul entre 0 et 1
1->2: Tous les processus prêts à s'exécuter en 1 font un pas entre 1 et 2
Ensuite, nous constatons qu'au fil des rondes seul le processus élu est celui qui reste actif le plus longtemps. Ainsi, la complexité en temps est associée au nombre de rondes accordées à ce processus.
Chaque phase i (avec i initialisé à 1) requiert 2^{i+1} rondes.
L'algorithme requiert de l'ordre de $log_2(n)$ rondes, ainsi, le nombre de rondes réalisé au total est équivalent à $2^{log_2(n)+1}-1$ = $2*n-1$. Nous avons donc un nombre de rondes de l'ordre de $O(n)$.

Question 7 :

Dans le cadre du cours, nous avons vu deux algorithmes d'élection sur anneau. Sur chacun d'entre eux, la complexité en nombre de messages était de l'ordre de $O(n^2)$. Ici, dans le pire des cas, nous avons une complexité de l'ordre de $O(n*log(n))$, ce qui est une optimisation non négligeable.
En parallèle, le coût mémoire de cet algorithme est moindre: chaque noeud contient 3 variables entières et un booléen, aucune liste n'est requise.
Nous pouvons également comparer cet algorithme à celui de Peterson vu en TD. Dans celui-ci, la complexité en nombre de messages est le même, $O(n*log(n))$, pour chaque phase $2n$ messages sont envoyés dans le pire cas.
L'algorithme présent envoie $3n$ messages dans le pire cas, ce qui est donc du même ordre.
Nous avons ainsi un algorithme optimisé en terme de mémoire et de nombre de messages, ce qui rend donc cet algorithme au moins aussi intéressant que ceux vus précédemment.