Examen 2019

Détecteur de défaillance parfait

1- Si le réseau est synchrone et nous connaissons une borne supérieure sur le temps de transfert des messages, alors il est possible de mettre en place un détecteur de défaillance parfait.

Explication :

→ Un détecteur de défaillance parfait doit pouvoir détecter de manière fiable si un processus est défaillant (ou non) en envoyant des messages et en recevant des réponses dans un délai connu.

→ Si un processus envoie un message à un autre et ne reçoit pas de réponse dans un temps fixé (qui est la borne supérieure de transfert), il peut en conclure que le processus est défaillant.

```
Processus P_i:
    envoyer "alive" à tous les autres processus P_j
    attendre réponse de P_j dans un délai Δ
    si pas de réponse dans Δ :
        marquer P_j comme défaillant
```

2- Si le réseau est synchrone mais que la borne supérieure du temps de transfert des messages est inconnue, il est impossible de mettre en place un détecteur de défaillance parfait.

Explication :

→ Sans connaître la borne supérieure du temps de transfert des messages, il est impossible de déterminer avec certitude si l'absence de réponse d'un processus est due à une défaillance ou à un retard normal du message.

→ En effet, si un processus ne répond pas à temps, cela pourrait être dû à un retard dans le réseau, et non à une défaillance réelle du processus. Par conséquent, un détecteur ne pourrait pas faire de distinction fiable.

Démonstration :

→ Supposons qu'un processus Pi envoie un message à Pj, mais Pj ne répond pas immédiatement. Si Pi ne connaît pas la borne supérieure du temps de transfert, il pourrait supposer à tort que Pj est défaillant, alors que ce n'est peut-être pas le cas. Il pourrait également considérer Pj comme défaillant à tort si le message est retardé.

Ainsi, il n'est pas possible de détecter de manière fiable une défaillance sans connaître la borne supérieure du temps de transfert.

3- Si le réseau est asynchrone (c'est-à-dire que les messages peuvent être retardés indéfiniment et que les processus n'ont pas de notion de temps absolu), il est impossible de mettre en place un détecteur de défaillance parfait.

Explication :

→ Dans un réseau asynchrone, les messages peuvent être retardés de manière imprévisible et indéfinie. Cela signifie qu'un processus ne peut jamais savoir avec certitude si un autre processus est défaillant ou si un message est simplement retardé.

→ Un processus pourrait attendre indéfiniment une réponse d'un autre processus, sans savoir si ce dernier est réellement défaillant ou si le message est en transit depuis trop longtemps.

Démonstration :

→ Supposons que Pi envoie un message à Pj et attend une réponse. Si le réseau est asynchrone, Pj pourrait être en train de répondre, mais le message de réponse pourrait être retardé indéfiniment. Sans aucune limite sur le temps de transmission des messages, Pi ne pourrait jamais être sûr que Pj est défaillant, même si Pj fonctionne parfaitement.

Ainsi, dans un réseau asynchrone, il est impossible de détecter de manière fiable les défaillances, car les messages peuvent être indéfiniment retardés.

Gestion des groupes (Séquenceur)

1- le séquenceur peut maintenir la liste des sites présents

Le séquenceur doit gérer la liste des sites actifs, en tenant compte des sites qui rejoignent, quittent ou sont en panne. Pour cela, chaque site doit notifier le séquenceur de ses actions (rejoindre ou quitter) et le séquenceur doit informer tous les autres sites du changement.

→ Algorithme pour qu'un site rejoigne le groupe :

1. Un site Pii souhaite rejoindre le groupe et envoie une demande au séquenceur.
2. Le séquenceur ajoute Pi à la liste des membres du groupe et envoie un message de confirmation à Pi.
3. Le séquenceur diffuse cette mise à jour à tous les autres sites pour les informer de l'arrivée de Pi.

→ Algorithme pour qu'un site quitte le groupe :

1. Un site Pi souhaite quitter le groupe et envoie une demande au séquenceur.

2. Le séquenceur retire Pi de la liste des membres du groupe et envoie un message de confirmation à Pi.
3. Le séquenceur diffuse cette mise à jour à tous les autres sites pour les informer du départ de Pi.

2- Algorithme de diffusion totalement ordonné

Pour garantir une diffusion totalement ordonnée, le séquenceur doit s'assurer que tous les sites reçoivent les messages dans le même ordre.

→ Algorithme de diffusion ordonnée :

1. Le séquenceur génère un numéro de séquence unique pour chaque message qu'il souhaite diffuser.
2. Le séquenceur envoie ce message avec son numéro de séquence à tous les autres sites.
3. Les sites récepteurs reçoivent les messages dans le même ordre et les traitent en respectant l'ordre de séquence.
4. Chaque site stocke un numéro de séquence qu'il a reçu, et les messages sont traités dans cet ordre.

3- Diffusion ordonnée avec perte de message

Lorsqu'un message peut être perdu, l'algorithme doit gérer la retransmission des messages et garantir que la diffusion ordonnée soit maintenue malgré les pertes.

→ Algorithme avec gestion des pertes de messages :

1. Lors de l'envoi d'un message, le séquenceur envoie une copie du message à chaque site et attend un accusé de réception pour chaque message (par exemple, une confirmation de réception).
2. Si le séquenceur ne reçoit pas l'accusé de réception d'un message dans le délai imparti, il renvoie le message.
3. Les sites reçoivent les messages dans le même ordre, et chaque site vérifie que le message qu'il reçoit a un numéro de séquence supérieur à celui qu'il a déjà traité.
4. Si un site détecte qu'il manque un message (par exemple, si un message est perdu), il demande une retransmission au séquenceur.

4- Initialisation d'un groupe et détermination du séquenceur unique

Mécanisme d'initialisation :

1. Un des sites peut être désigné comme candidat pour devenir le séquenceur (par exemple, le premier site à démarrer).
2. Le candidat envoie un message aux autres sites pour annoncer sa candidature en tant que séquenceur.
3. Les autres sites peuvent confirmer ou contester cette candidature. Si un site reçoit une candidature d'un autre site avec un identifiant plus bas ou une version de groupe plus ancienne, il peut accepter cette candidature.
4. Une fois qu'un site a été élu séquenceur, tous les autres sites le reconnaissent et diffusent cette information.

5- Détection des défaillance du séquenceur

Pour détecter la défaillance du séquenceur, chaque site doit vérifier si le séquenceur répond toujours aux requêtes et assure la diffusion des messages.

Détection de défaillance :

1. Chaque site envoie périodiquement une requête au séquenceur pour vérifier s'il est toujours en ligne.
2. Si un site ne reçoit pas de réponse du séquenceur dans un délai défini (par exemple, T), il peut conclure que le séquenceur est défaillant.
3. Un mécanisme de timeout est utilisé pour détecter l'absence de réponse du séquenceur.

6- Algorithme de traitement de défaillance du séquenceur

Lorsqu'un site détecte que le séquenceur est défaillant, il faut élire un nouveau séquenceur pour garantir que la gestion du groupe et la diffusion des messages continuent de manière ordonnée.

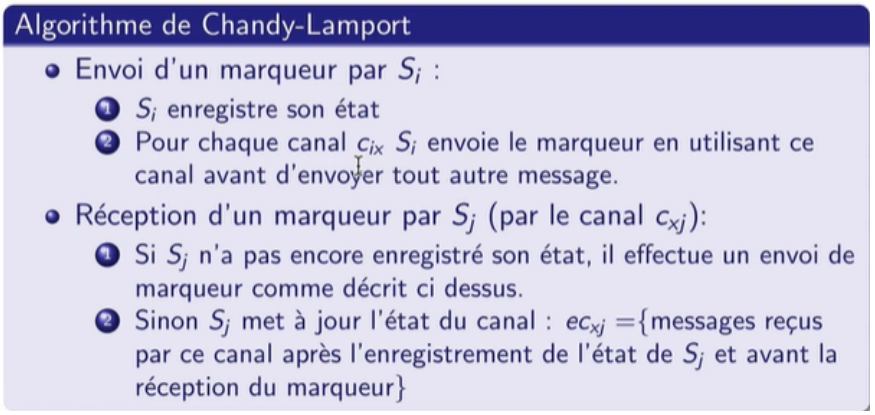
Algorithme de traitement de défaillance du séquenceur :

1. Lorsqu'un site détecte que le séquenceur est défaillant, il déclenche un mécanisme de réélection.
2. Les sites candidats s'envoient des messages pour s'identifier comme candidats à devenir le nouveau séquenceur.
3. Un nouveau séquenceur est élu selon un critère (par exemple, le site avec le plus petit identifiant).
4. Une fois un nouveau séquenceur élu, il envoie un message à tous les autres sites pour les informer de son élection et pour commencer à diffuser des messages avec un numéro de séquence valide.
5. Les autres sites mettent à jour leur variable seq pour refléter le nouveau séquenceur.

Prise de cliché

Rappel de l'algo de Chandy Lamport

1. Initiation de l'enregistrement d'état : Le processus p2 envoie un message spécial (indiqué par le marqueur) à ses voisins. Cela marque le début de l'enregistrement de l'état de tous les processus.
2. Enregistrement de l'état local : Chaque processus qui reçoit un marqueur enregistre son état local à ce moment-là. (les événement précédents le marqueur)
3. Enregistrement des messages en transit : Lorsqu'un processus reçoit un marqueur, il doit également enregistrer tous les messages qu'il a envoyés avant de recevoir ce marqueur, mais non encore reçus.



Imaginons un système avec trois processus : p1, p2, et p3.

1. Le processus p2 lance l'enregistrement d'état.
    - Il envoie un marqueur à p1 et p3.
    - Il enregistre son état local à ce moment-là.
2. Le processus p1 reçoit le marqueur de p2.
    - Lorsqu'il reçoit le marqueur, p1 enregistre son état local.
    - Ensuite, il envoie un marqueur à p2 (ou à d'autres, mais ça dépend de l'algorithme).
    - Les messages envoyés par p1 à p2 avant qu'il ait reçu le marqueur de p2 doivent être enregistrés comme étant en transit. Ce sont des messages qui ont été envoyés par p1, mais ils n'ont pas encore été reçus par p2 au moment où p2 envoie le marqueur.
3. Le processus p3 reçoit le marqueur de p2.
    - Il enregistre son état local.
    - Puis, il envoie un marqueur à p2.
    - Les messages envoyés de p3 à p2 avant qu'il n'ait reçu le marqueur de p2 doivent aussi être enregistrés comme en transit.

L'algo qui envoie le message avant de recevoir le marqueur enregistre le message.