SDR - Mock TE1 solution

Q1 - Détecteur de pannes parfait, avec pannes récupérables

Nous avons parlé en cours de détecteurs de pannes parfaits dans le contexte de pannes permanentes uniquement.

Quelles garanties du détecteur changeraient (s'il y en a qui changent), si l'on autorisait également les pannes récupérables.

> ⓘ **Solution**
>
> Il perdrait sa garantie de précision, puisqu'il risquerait de donner des faux positifs, c'est à dire de détecter un processus qui est effectivement correct, mais qui est tombé en panne récupérable une seule fois et plus jamais ensuite.

Q2 - Tolérer uniquement les pannes arbitraires

Je viens de finir d'implémenter un algorithme capable de gérer des pannes arbitraires, et j'en suis très fier. Par contre, je réalise seulement maintenant que j'ai complètement oublié de considérer les autres types de pannes : permanentes et récupérables. Une connaissance me dit que ce n'est pas grave, et que je n'ai pas besoin de m'en soucier.

A-t-elle raison ou tort ? Expliquez pourquoi.

> ⓘ **Solution**
>
> Elle a raison ! Une panne arbitraire est une panne qui fait diverger le comportement d'un processus de l'algorithme. Le fait de disparaitre, de manière permanente ou non, représente une divergence de l'algorithme, et donc une panne arbitraire.

Q3 - Horloge de Lamport - Déduction

Un processus `A` reçoit un message d'un processus `B`, marqué d'une estampille 33.

Juste ensuite, `A` envoit un message marqué d'une estampille 35.

Quelle était la valeur de l'estampille de `A`, juste avant réception du message de `B` ?

> ⓘ **Solution**

- [ ] Exactement 35
- [ ] Exactement 34
- [ ] Exactement 33
- [x] 33 ou moins
- [ ] Strictement moins de 33

Q4 - Lamport - Ordre des événements

Dans un système distribué, étant donnés deux événements arbitraires, sélectionnez la ou les affirmations correctes.

Dans cette question,

- quand on dit que `E1` arrive avant `E2`, on entend que `E1` était l'envoi et `E2` la réception d'un même message, ou bien que `E1` et `E2` sont arrivés dans cet ordre sur le même processus ;
- l'heure logique d'un événement est la valeur de l'échantillon (timestamp) telle que décrite par le protocole de Lamport, sans départition des égalités.

Solution

- [x] Il est possible que E1 ne soit pas arrivé avant E2, et E2 ne soit pas arrivé avant E1 non plus.
- [ ] Si l'heure logique de E1 est strictement inférieure à celle de E2, alors E1 est arrivé avant E2.
- [x] Si E1 est arrivé avant E2, alors l'heure logique de E1 est inférieure à celle de E2.
- [x] Il est possible que l'heure logique de E1 soit égale à celle de E2.

Q5 - Lamport - Variantes de timestamps

Voulant me sentir unique, je propose de modifier la fonction de mise à jour de l'estampille de Lamport, dans l'algorithme de Mutex de Lamport, comme suit.

Le timestamp local est dénoté `ts`.

Lors de la réception d'un message marqué d'un timestamp `ts_i`, quelle nouvelle valeur peut-on donner à `ts` sans casser l'algorithme ?

On suppose que les timestamps commencent à 1.

Solution

- ○ max(ts_i, ts)
- ✓ max(ts_i, ts)*2
- ✓ max(ts_i+1, ts+1)
- ✓ max(2 * ts_i, 2 * ts)
- ✓ ts_i + ts

Q6 - Lamport - Un REQ trop lent

Une connaissance, en train de réviser pour le TE de SDR, vous présente le scénario suivant, à propos de l'algorithme de Mutex de Lamport :

> "Imaginons deux processus, A et B :
>
> - À l'instant T=1, A envoit un REQ avec timestamp 2 à B. Ce REQ prend du temps à transiter, et arrivera chez B à T=5.
> - À l'instant T=2, B envoit un REQ avec timestamp 3 à A.
> - Le REQ de A vers B étant lent, A a le temps de recevoir le REQ de B à l'instant T=3, et d'y répondre avec ACK.
> - B reçoit ce ACK à l'instant T=4, et peut donc entrer en SC, alors même qu'il n'a pas la requête la plus ancienne !"

- Si cette connaissance a raison, expliquez pourquoi l'algorithme de Lamport autorise B à entrer en Section Critique même sans avoir la requête la plus ancienne, sans que cela ne brise les propriétés garanties par l'algorithme.
- Si elle a tort, où se trompe-t-elle, en quelques phrases au plus.

Solution

B ne peut pas recevoir le ACK de A avant de recevoir le REQ de A, car le réseau ne réordonne pas les messages. C'est une supposition sur les garanties du réseau qui est nécessaire au bon fonctionnement de l'algorithme.

Q7 - Ricart et Agrawala - Renvoyer REQ après OK

Une différence entre l'algorithme de Ricart et Agrawala, et celui de Carvalho et Roucairol est que ce dernier doit parfois renvoyer son message de type `REQ` après un message `OK`.

Pour quelle raison l'algorithme de Ricart et Agrawala ne nécessite jamais, lui, de renvoyer un `REQ` après un `OK`.

Solution

Car dans Ricart et Agrawala, la requête a déjà été envoyée par le passé, l'autre processus l'a stockée et est déjà au courant. Dans Carvalho et Roucairol, on n'envoit pas la requête si on a déjà le jeton, donc l'autre n'est peut-être pas encore au courant que je veux le jeton.

Q8 - Carvalho et Roucairol - Avec un lien en moins

Considérons un réseau de taille N, dans lequel chaque processus connait l'existence de tous les N-1 autres.

Supposons que, à cause d'une erreur de configuration, il existe deux processus X et Y qui n'ont pas conscience d'être voisins. Cela signifie que X pense avoir N-2 voisins dont Y ne fait pas partie, et similairement pour Y.

L'algorithme de Carvalho et Roucairol fonctionne-t-il correctement sur ce réseau ?

- Si oui, expliquez pourquoi.
- Si non, donnez un exemple de situation problématique.

Solution

L'agorithme ne fonctionne plus. En effet, si X et Y font une demande simultanée, et qu'aucun voisin n'est en Section Critique, alors leurs N-2 voisins leur fourniront leur jeton. Ils auront donc tous les deux tous les jetons qu'ils attendent, et ils entreront tous les deux en Section Critique.