TD 1 28/01/26

Exo 1: Donnée:

2 processus A et B reliés par un lien bidirectionnel FIFO asynchrone

A et B ne peuvent envoyer que deux messages 0 et 1

3 propriétés:

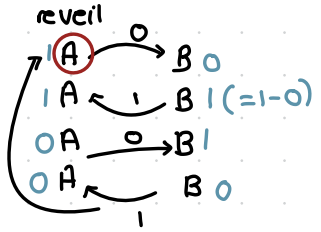
1) A et B envoient chacun une infinité de messages 0 et une infinité de messages 1;
2) sur chaque lien orienté, les suites de messages soient toujours de la forme 0^*1^* ou 1^*0^*;
3) Chaque processus est toujours en mesure d'envoyer un message.

Q1) Protocoles qui vérifient 2 des 3 propriétés

1 & 2: variables: x = message

- Lors du reveil envoyer(0)
- répéter toujours : $\emptyset$
- lors de la réception de x
  envoyer(1-x)
  si pas reveillé, faire réveil

(reveil: quand décide qu'on va lancer le protocole)
i.e. démarrer le protocole.
(→ on suppose qu'il y a au moins un reveil spontané, éventuellement plusieurs)



2 & 3:

- lors du réveil envoyer(0)
- répéter toujours: envoyer(1)
- sur réception de x
  si pas réveiller
  se réveiller
  sinon envoyer(1)

lors de décision d'envoyer: envoyer 0

1 & 3:

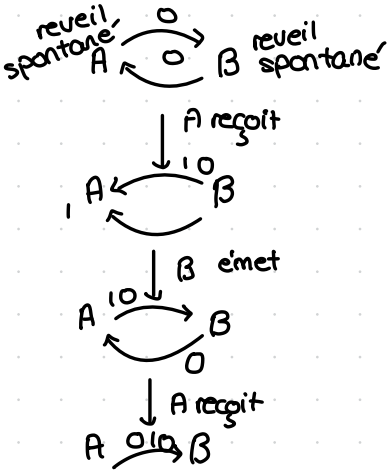
Lors du reveil envoyer(0) envoyer(1)
faire envoyer(0) envoyer(1)

1 & 3: par b = 0 // ou 1
lors de decision d'envoyer [envoyer b ; b = 7b] (alterne les 1 et les 0)

Q2) Protocole qui vérifie les 3 propriétés:

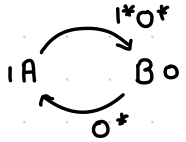
variables: valeur: entier

- lors du réveil:
  valeur ← 0
  envoyer(valeur)
- repeter toujours:
  envoyer(valeur)
- réception de x:
  si pas réveil alors faire réveil
  A: val ← x
  B: val ← 7x
  envoyer(val)



Q3) Que se passe-t-il si des liens perdent des messages?

Même si les canaux perdent des messages, "on tourne" dans l'automate
Si on suppose que c'est B qui ne reçoit jamais 1, mais A en émet une infinité, le canal ne perd pas



Q4) Interêt du protocole + m propriété avec FIFO, quitte à ce que les messages soient plus gros?

↳ peu importe si tu perds des messages, la connexion est stable entre les deux. ↳ oui, mais il faut numéroter les messages.

TD 2 04/02/26

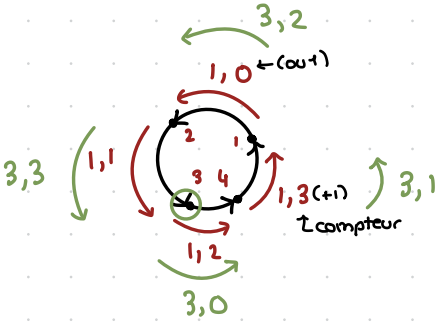
Feuille 1, exo 2

anneau asynchrone orienté

algo réparti pour que chaque processus calcule la taille de l'anneau

2 cas: • chaque processus a un identifiant unique;
• tous les processus sont identiques

1er cas: Id unique



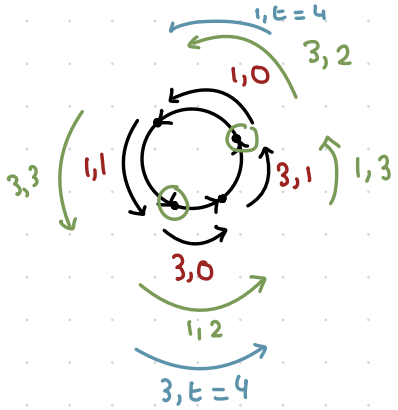
+1 reveil spontané

✓ FIFO ou pas FIFO

terminaison locale
(car asynchrone)

complexité: (selon le nb de message) n^2

pour diminuer la complexité (en mieux et en moyenne)



au mieux: un seul reveil: complexité $2n$
(calcul de la taille + diffusion du message)

au pire: $2n^2$

Algorithmes:

(1) Variables. ID : identifiant unique
taille: entier initialisé à -1
asleep: bool init à Faux

Au reveil:
asleep ← Faux
var taille = 0
envoyer (ID, 0)

Lors de la réception de (ID_reçu, x)
si asleep se reveiller
si (ID = ID_reçu) alors
taille = x+1
sinon envoyer (ID_reçu, x+1)

PID: int unique
c: canal (du successeur)
taille: int initialisé à -1
asleep...

un processus au reveil:
envoyer (exploration, PID, 0)
asleep false

un processus sur reception de (type, id, t)
si asleep (asleep=Faux)
si type = exploration alors
si id = PID alors
envoyer (diffusion, id, t+1)
taille := t+1
sinon si type = diffusion alors
si id = PID alors <terminer>
sinon
envoyer (diffusion, id, t)
taille = t

FIFO: terminaison locale si il y a autant de messages d'exploration que de messages de diffusion

↳ pour réduire la taille du message (sans compteur)

(3) sans compteur

ID : identifiant unique
taille : initialisée à 0
asleep : initialisé à True

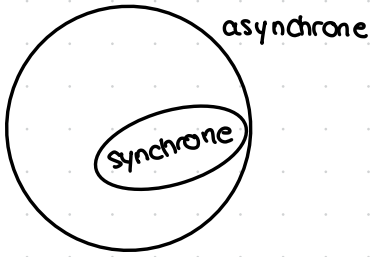
Au reveil,
  envoyer (ID)
  asleep = False
sur reception de (ID_reçu)
  si asleep, faire reveil
    taille++
  si ID_reçu ≠ ID alors
    envoyer (ID_reçu)

cas 2 : tous les processus sont identiques

algo impossible

Autres hypothèses:

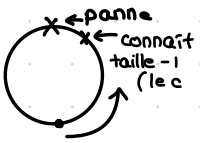
- si le système est synchrone



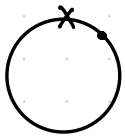
synchrone → $\tau = 1$ ⇝ à la fin $\frac{\text{temps}}{\tau}$
tick ← processus
= nb de ticks d'horloge

- panne :

→ le dernier noeud connaît taille - 1, mais ne le sait pas



- synchrone + panne



m̃ que panne, mais les noeuds savent qu'il y a une panne

- synchrone + bidirectionnel + panne

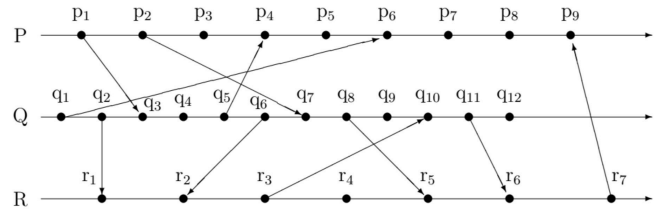
avantages : • je peux envoyer de chaque côté
• je peux détecter si mon voisin est en panne (mess + acquittement)

1) verif des voisins si pas en panne (mess check et acquittement) ⇝ 4n - 1
2) à tout voisins fiable, envoyer un message de calcul de taille.
3) quand je reçois un mess de calcul de taille, je transfère si mon voisin opposé est fonctionnel
   sinon renvoi à l'envoyeur sous forme mess
4) si je reçois mon retour, j'additionne
   si c'est mon calcul je stocke (pour savoir s'il faut additionner les 2 côtés)

$O(n^2)$: $4(n-1) - 2 + \left(2(n-3)\right).n$

TD 3 11/02/26

TD Horloges: Exo1:



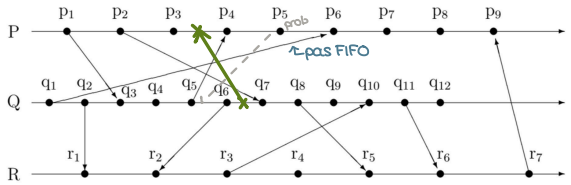
lignes de temps en local
mais les lignes ne sont pas comparables

1) que représentent p_6, q_1, p_2, r_4 ?

p_6: reception
q_1, p_2: envoi
r_4: evenement interne
} types de messages

2) Peut-on ajouter :

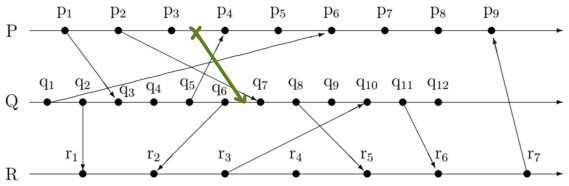
a) une émission de message juste après q_6 et sa réception juste après p_3 ?



canaux pas FIFO : possible ↳ q_1, q_5

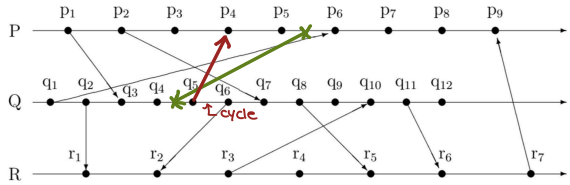
(si pas de cycle → pas de problème de causalité)

b) ——— p_3 ——— q_6 ?



possible

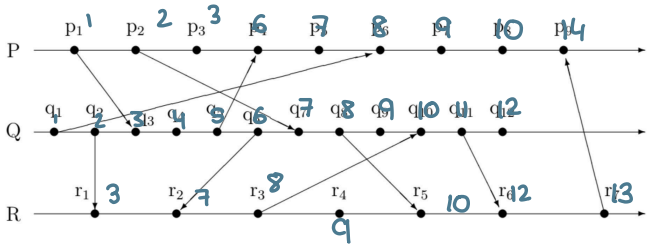
c) ——— p_5 ——— q_4 ?



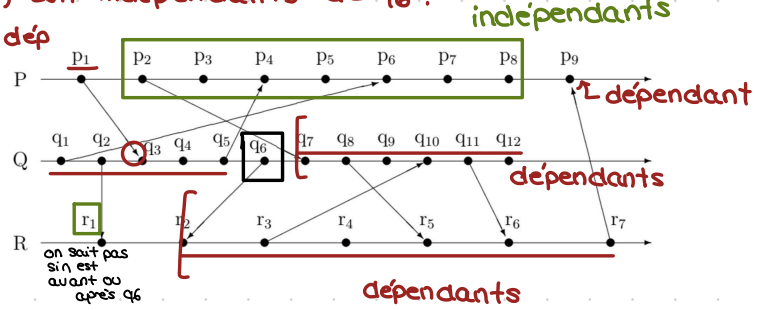
pas possible

3) Calculer les horloges de Lamport :

Evn interne / envoi : incrémentation de 1
reception : le max +1
en P après p_9 : 14
en Q après q_{12} : 12
en R après r_7 : 13



4) Evn indépendants de q_6 ?



5) Mq $L(a) < L(b) \not\Rightarrow a \longrightarrow b$

Ex: p_3 et q_4

$3 < 4$

ou q_6 et p_8 (indépendants)

6) D'après le diagramme, les voies de communication préservent-elles l'ordre des messages?

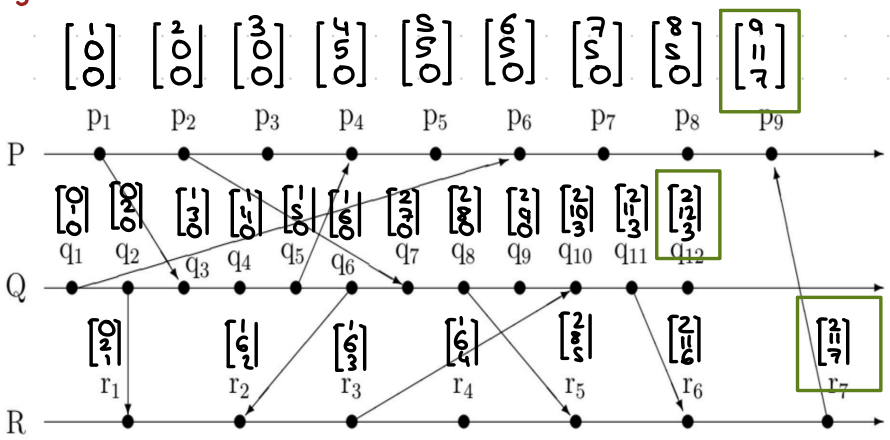
NON pas FIFO

5) Z_{min}: min temps d'envoie de message (on néglige les evnts locaux)

algo: calculer la durée min de l'algo décrit par la relation de causalité?

(Lamport en ignorant les évènements internes et les récetions, → prende la valeur max et la multiplier par Z_{min}.)

8) Horloge de Matter



vecteurs : une entrée par processus

↳ un elt de b > elt de a

9) Mq $M(a) < M(b) \Leftrightarrow a \longrightarrow b$

preuve:

($\Leftarrow$) par définition

(l'evnt qui est causalement après va augmenter l'une des composantes du vecteur)

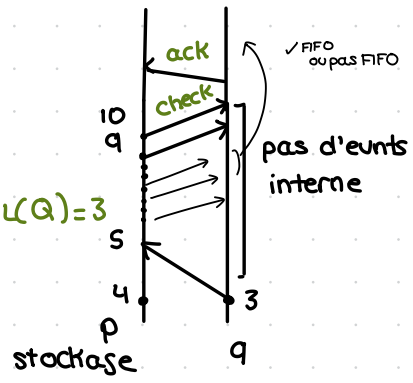
($\Rightarrow$) On suppose $M(a) < M(b)$

$M(a)|_p$ < $M(b)|_{P_a}$

un événement de b est plus grand pour le process p.

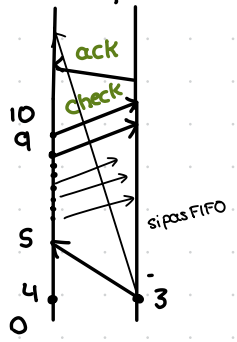
Exo2:

1) algo reparti à deux processus pour borner la valeur absolue de l'écart entre leurs deux horloges



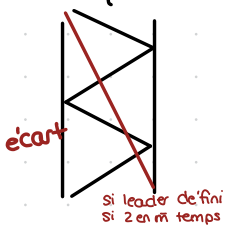
2) L'ecart entre la valeur sur un processus et sur un message est-il aussi borné?

Oui (P n'envoi pas de message tant qu'il a pas reçu l'aquittement



→ sol possible

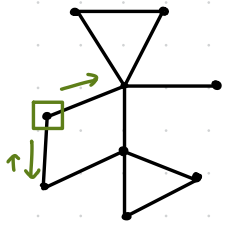
on acquitte à chaque fois



3)
pour généraliser ?
relations 2 à 2 à chaque voisin

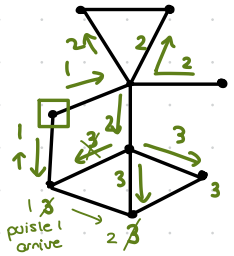
4) Pourquoi cela peut être utile ?
diminuer la taille des messages
+ reset au bout d'un moment
limite le nb de bits qu'on envoi

TD : Algo création d'arbre



synchrone : ✓ arbre en largeur (fixe)
asynchrone : fil
(pas forcément un arbre en largeur)
↳ pour les diffusions d'information

↳ on rajoute une notion de "distance"



↝ profondeur 3

→ pour la terminaison
diffuser
↝ puis répondre oui
(pas l'inverse)

complexité :
au pire : nb arcs x nb sommets

ou bien : algorithme "par vague"
racine : envoi à d=1 , aqu
d=2 , aqu
d=3 , aqu
↳
pour ne pas utiliser de compteur
on introduit un booléen "participant"
fin ?
(renvoyer des messages à la racine

→ accroche ?

TD 4 18/02/26

(Feuille de TD1)

Exo 3:

réseaux liens bidirectionnels + processus distingué

1) protocole de diffusion d'informations par le processus distingué

booléen deja_vu (sinon on ne s'arrête jamais)
plusieurs info:
hash / idée

pour tout processus i:

variable déjaVu : bool à faux

fct Reveil ( ): si je suis le leader
deja Vu ← vrai
envoi (Info) à tout mes voisins

lors de reveil Reveil ( )

lors de réception de Info sur c:
si ¬ dejavu alors
dejaVu ← Vrai
envoyer (Info) à tous les voisins (sauf c)

complexité:

2x. le nb d'arêtes (borne supérieure)

2) avec les acquittements

pour tout processus i:

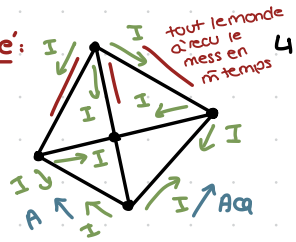
variable père : canal
déjaVu : bool à faux
acqVoisins : entier (liste ou nb de voisins ↑ messages frauduleux)

lors de reveil si je suis le leader
dejaVu ← vrai
envoi (Info) à tout mes voisins
acqVoisins ← monDegré
sinon (nb d'acq que je dois recevoir) ← monDegré - 1 (pas nécessaire)
père ← c
si acqVoisins == 0 alors envoyer (Acq) à père

lors de réception de Info sur c:
si ¬ dejavu alors
dejaVu ← Vrai
envoyer (Info) à tous les voisins sauf c
acqVoisin ← monDegré - 1
sinon
envoyer (Acq) à c

Lors de réception de (Acq) sur c:
acqVoisin - -
si acqVoisins == 0 alors si non Leader envoyer (Acq) à père
sinon <terminer>

3) Complexité :

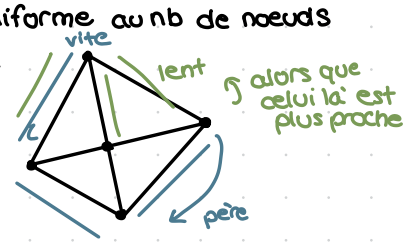


4 * nb arrêtes messages
- 2 * nb arrete dans l'arbre
= nb noeud - 1

4) Complexité en temps

temps fixe Z : = 2 * le rayon centré au leader x Z

transmission borné par Z : au pire : arbre filiforme au nb de noeuds
= 2 * nb noeuds x Z



→ Création d'un arbre :

algo distance, algo vague

algo distance :

sans accusé de réception :

```
var: pere : canal
     dist : entier à ∞
Au reveil :
     si proc distinguer alors
        envoyer(distance (1)) à tous les voisins
        dist = 0
sur reception de distance(x) sur c :
   si x < dist alors
      pere ← c
      distance ← x
      envoyer distance(x+1) à tout les voisins sauf c
```

avec accusé de réception :

```
var: pere : canal
     dist : entier à ∞
     AcqVoisins : entier à 0
Au reveil :
     si proc distinguer alors
        envoyer(distance (1)) à tous les voisins
        dist = 0
        AcqVoisin = monDegré
```

```
sur reception de distance(x) sur c :
   si x < dist alors
      pere ← c
      si pere ≠ Null alors
         envoyer (Acq)
      pere ← c
      distance ← x
      envoyer distance(x+1) à tout les voisins sauf c
      AcqVoisins += monDegre - 1   (~ les prochains doivent avoir la bonne distance)
      si AcqVoisins == 0 alors envoyer Acq à père
sinon // si x ⩾ dist
   envoyer (Acq) à c
```

```
sur reception Acq de c
   AcqVoisins --
   si AcqVoisins = 0 alors
      si proc distingué <terminé>
      sinon
         envoyer Acq à père
```

complexité:

au pire: 2x nb de sommets x nb d'arêtes (nb messages)

$2.\ |S| \cdot |A|$

$|V| \cdot |E|$

$O(|S| \cdot |A|)$

en temps: $O(rayon)$

algo des vagues: (voir cours
version avec acquittement)

complexité:

en temps: $O(rayon^2)$ 1 2 3 somme de 1 à rayon

en message:

$O(rayon \times |S| + A)$

(écrire l'algorithme + complexité)
+ ex 4.
entrainnement.