Spécification des Systèmes distribués – Master Technologie de l'Internet 1ère année

Correction TD Diffusion

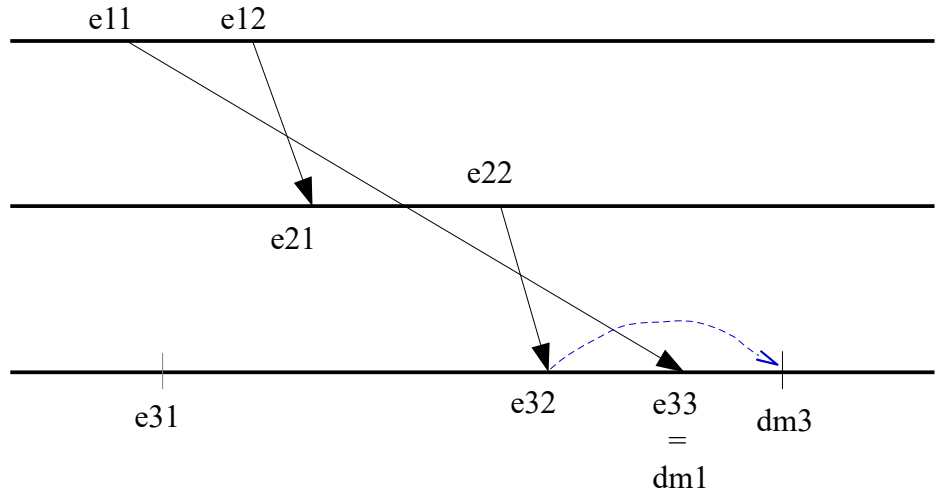
Eric Cariou

Exercice 1

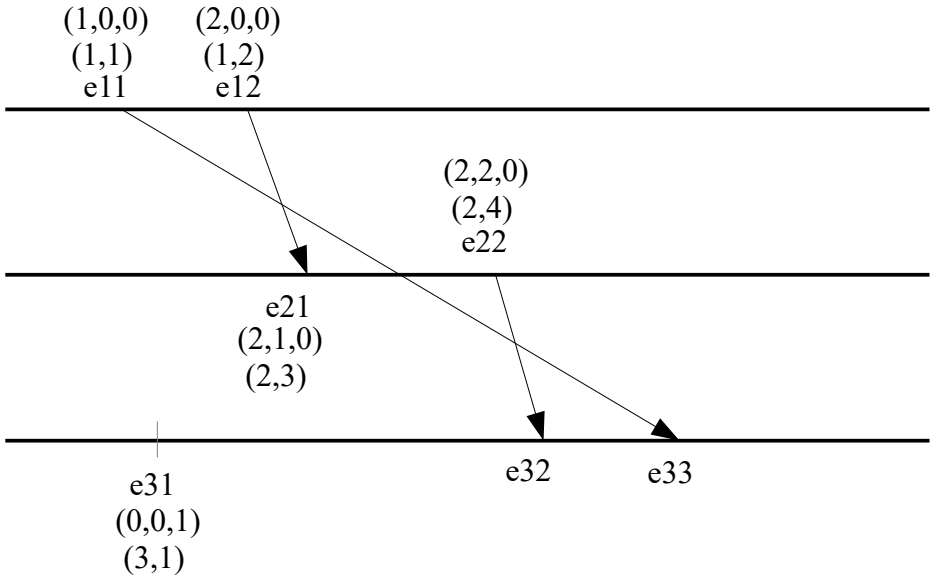
1) S'il y une dépendance causale entre l'émission de deux messages à destination du même processus, il faut que les messages soient délivrés dans le même ordre sur ce processus :

Pour tous messages m et m' reçus par un même processus : e_m → e_m' => d_m → d_m'

2) Sur P3, réception de 2 messages m1 et m3 avec e_m1 → e_m3 car e11 → e22 mais on constate que les messages sont reçus dans l'ordre inverse. Il faut donc quand on reçoit m3 sur P3, attendre de recevoir m1 pour délivrer m1 et ensuite délivrer m3 : on retarde le délivrement de m3 par rapport à sa réception en e32.



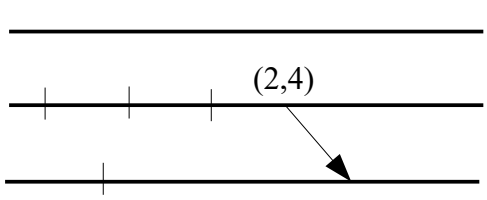
3) Horloges de Lamport et Mattern

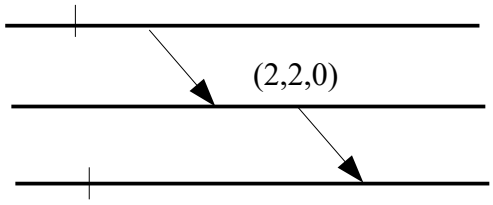


Problème à résoudre :

- Je suis P3, je ne connais que ce qu'il s'est passé chez moi et ce que j'ai reçu comme messages des autres processus
- En e32, j'ai reçu un message de P2 avec une certaine date : (2,4) ou (2,2,0)
- Localement, il s'est déroulé un événement local chez moi (e31)
- Suis-je capable, moi P3, de savoir à la réception de ce message de P2 en e32 si j'aurais du recevoir un message de P1 avant ?

Pour démontrer que les horloges de Lamport et de Mattern ne suffisent pas, il suffit de trouver un contre-exemple pour lequel P3 aura la même vision mais sans impliquer le message venant de P1 non encore reçu par P3. Pour cela, il faut qu'il reçoive de P2 un message avec une date de (2,4) ou de (2,2,0). Pour Mattern, la première case avec une valeur de 2 informe que P2 connait deux événements de P1. Cette information ne peut venir que du fait que P1 a envoyé un message à P2. Les deux contre-exemples sont les suivants (à gauche Lamport, à droite Mattern) :





4) Horloge matricielle

Chaque horloge rajoute un niveau d'information mais il en faut plus. Soit e un événement de Pi:

- Hi(e) (Lamport) : ce que Pi connait globalement du système (nb d'événements avant lui mais sans savoir sur quels processus)
- Vi(e) (Mattern) : ce que Pi connait de Pj (nb d'événements que Pi connait sur chacun des processus)
- Mi(e) (matrice) : ce que Pi connait de ce que Pj connait de Pk (Pi sait que Pj sait que Pk a envoyé des messages à d'autres processus)

Pour N processus, chaque événement est daté par une matrice N x N. Pour un événement de Pi :

- Ligne i : compteurs locaux à Pi
  - M[i,i] : nb d'événements sur Pi
  - M[i,j] avec $j \neq i$: nb de messages que Pi a envoyé à Pj
- Ligne j avec $j \neq i$: informations que l'on connait sur les autres processus
  - M[j,j] : nb d'événement que l'on connait sur Pj
  - M[j,k] avec $k \neq j$: nb de messages que l'on sait que Pj a envoyé à Pk

Mise à jour des horloges pour un événement de Pi :

- A chaque événement, on incrémente la case [i,i]
- Si l'événement est l'émission d'un message à destination de Pj, on incrémente la case [i,j]
- Si l'événement est la réception d'un message : on fait le max case par case entre la dernière date locale et la date d'émission du message sauf pour la ligne i

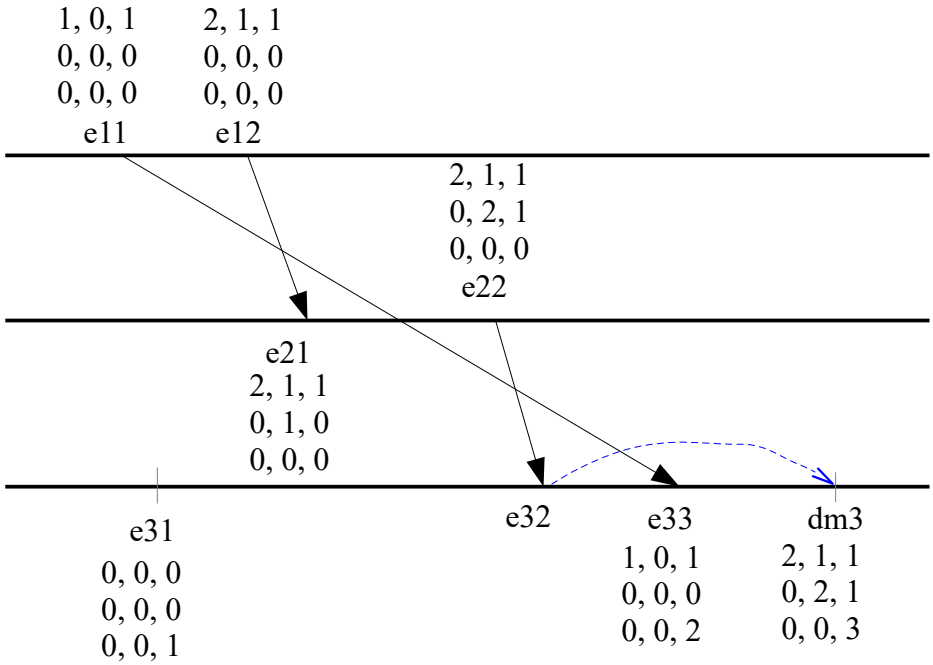
Exemple (indépendant de l'exercice) pour un événement d'un processus P2 daté par :
4, 1, 2
0, 3, 1
0, 0, 0

Ligne 1 : on sait qu'il y a eu 4 événements sur P1, que P1 a envoyé 1 message à P2 et 2 à P3.
Ligne 2 : on a envoyé aucun message à P1 et 1 à P3 et il y a eu 3 événements sur P2
Ligne 3 : on ne sait rien de ce que P3 a fait

Pour assurer l'ordre causal, à la réception d'un message sur Pi venant de Pj, on compare la colonne i de la date locale et de la date d'émission du message (sauf la case [i,i]). Tout est bon si :

- Locale[j,i] +1 = Message[j,i] : différence de 1, c'est le message qu'on vient de recevoir de Pj
- Locale[k,i] >= Message[k,i] : on a délivré tous les messages de Pk que Pj sait que Pk a envoyé à Pi

Application sur le chronogramme de l'exercice avec un point important : <u>on ne date que les événements de délivrement de message, pas de réception</u>. Le principe étant que la date locale permet de savoir à la fois les messages que l'on a envoyé à chaque processus ainsi que les messages que l'on a délivré venant de chaque processus.



Date de e11, première ligne : 1, 0, 1. Le premier 1 est pour indiquer qu'il y a eu un événement sur P1 et le dernier pour indiquer que P1 a envoyé un message à P3.

Date de e21 : la première ligne est mise à jour avec la date d'émission du message venant de P1. La deuxième ligne, en case 1 indique que P2 a délivré un message P1 (celui qui vient d'être reçu) et la case 2 qu'il y a eu 1 événement sur P2. La troisième ligne reste à 0 car ne contient que 0 localement ou pour l'émission du message.

En e32, dernière date locale :
0, 0, 0
0, 0, 0
0, 0, 1

Date du message reçu en e32 venant de P2 :
2, 1, 1
0, 2, 1
0, 0, 0

On est sur P3, on compare les colonnes 3 (sauf la case [3,3]), soient les cases en bleu :

- Ligne 1, en locale = 0, du message = 1 : P3 a délivré 0 message venant de P1 mais P2 sait que P1 a envoyé 1 message à P3. Si P2 sait cela, c'est que l'émission du message qu'il vient

d'envoyer à P3 est dépendante causalement de l'émission de ce message là. On sait qu'on doit retarder le délivrement du message de P2 en attendant de recevoir ce message de P1.

- Ligne 2, en locale = 0, du message = 1 : une différence de 1 mais pas de problème ici car il s'agit du message venant de P2 non encore délivré.

Grâce aux dates, on sait en e32 qu'on doit retarder le délivrement du message venant de P2 tant qu'on a pas reçu celui venant de P1. Il sera reçu en e33 et là les dates seront cohérentes, on peut directement délivrer ce message de P1 puis ensuite on délivre le message de P2 qui avait été mis de coté.

La dernière colonne de la date de dm3 précise que sur P3, 1 message de P1 et 1 message de P3 ont été délivrés.

Exercice 2

1) Pour tous messages m et m' et pour tous processus i : $e\_m \rightarrow e\_m' \Rightarrow d^i\_m \rightarrow d^i\_m'$

2) Au niveau des dépendances causales entre les diffusions de message quand on regarde le chronogramme, on trouve :

- em1 → em2
- em1 → em3
- em1 → em4
- em2 → em4
- em2 || em3
- em3 → em4

Sur les processus, les messages sont reçus dans cet ordre :

- P1 : m1 m3 m4 m2
- P2 : m1 m2 m3 m4
- P3 : m1 m3 m2 m4

Le problème est sur P1 où m4 est reçu avant m2 alors que le chrogramme montre une dépendance causale en émission dans l'autre sens.

3) La question n'a pas d'utilité ici : il s'agit de démontrer, comme dans l'exercice précédent, que les horloges de Mattern ne permettent pas de différencier le cas du chronogramme avec un contre-exemple mais dans le cas précis de l'exercice, il n'existe pas de contre-exemple. Néanmoins, pour les mêmes raisons que pour l'exercice 1, on manquera de manière plus générale d'informations pour assurer le délivrement causal.

4) On pourrait utiliser les horloges matricielles comme dans l'exercice 1 mais on peut faire plus simple vu que l'on ne communique que par diffusion : il n'est pas utile de savoir combien de messages a envoyé un processus à tel autre processus puisque par principe, un message diffusé est reçu par tous les processus.

La version restreinte des horloges de Mattern consiste alors à ne compter que les messages diffusés par chacun des processus. Par exemple sur P1, une date (2, 1, 3) indiquera que P1 a diffusé 2 messages, P1 a reçu (et délivré) 1 message de P2 et 3 de P3.

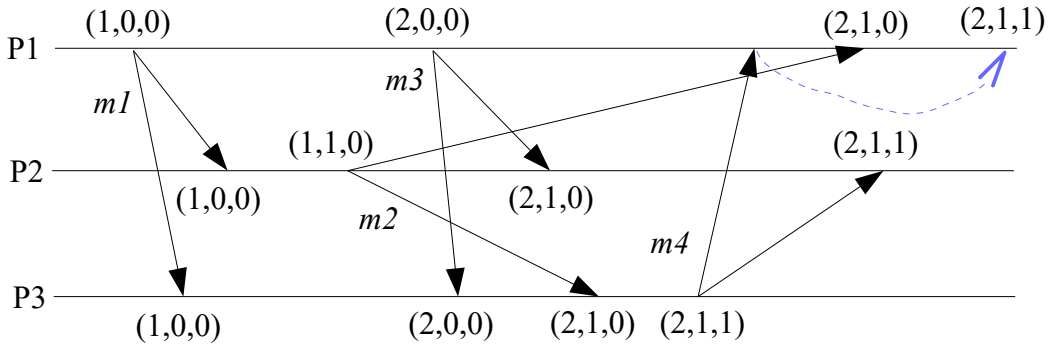
A chaque fois que l'on diffuse un message, on incrémente sa case du vecteur et on envoie la date avec le message. A la réception d'un message, si on le délivre, on fait le max case par case entre la date locale et la date du message (il n'y a pas de compteur d'événements locaux à tenir, on ne comptabilise que les émissions et délivrements de messages).

Quand on reçoit sur Pi un message de Pj, on compare les cases entre la date locale et la date du message :

- locale[j] < message[j] + 1 : s'il y a une différence de plus d'1, c'est qu'il y a un autre message de Pj qui n'a pas été reçu en plus de celui qu'on vient de recevoir.
- locale[k] < message[k] avec k ≠ i : Pj connait plus de diffusions faites par Pk que Pi. C'est donc que ces diffusions auraient du arriver sur Pi avant qu'on l'on reçoive celle de Pj.

Ces comparaisons permettent d'assurer l'ordre causal. L'ordre FIFO consiste simplement à comparer les dates pour les messages venant du même processus, pas de n'importe lesquels.

En simplifiant le chronogramme pour ne pas nommer les événéments et ne pas préciser la diffusion à soi-même (où on fera par principe le délivrement immédiatement), voici ce que donne la datation avec ces horloges :



A l'émission de m1, P1 met son horloge à (1,0,0) pour signifier qu'il a diffusé un message. A la réception du message sur P2 et P3, les horloges locales (qui étaient à 0) sont mises à jour avec cette valeur de (1,0,0).

Sur P1, à la réception de m4 venant de P3, la date locale est (2,0,0) et la date du message est (2,1,1). Au niveau de P3, tout va bien, il y a une différence de 1 entre la date locale et celle du message : c'est le message qu'on vient de recevoir. Par contre il y a aussi une différence de 1 au niveau de P2 : P3 connait 1 diffusion de P2 alors que P1 n'en connait aucune. Il y a donc une diffusion de P2 que P1 aurait du recevoir. On peut alors retarder le délivrement de m4 jusqu'à ce que ce message arrive.

5) Il n'est pas possible d'assurer un ordre total avec des horloges. Les messages m2 et m3 sont diffusés sans dépendance causale entre eux. Au niveau des dates, il n'y a pas d'ordre déterminable ni de problèmes détectables. On peut le voir sur P3 où m3 est d'abord reçu puis m2, ce qui donne comme dates sur P3 : (1,0,0) puis (2,0,0) puis (2,1,0). Si on avait d'abord m2 puis m3, on aurait eu : (1,0,0) puis (1,1,0) puis (2,1,0). Dans les deux cas, il n'y a pas de problème de cohérence.

L'indépendance causale, par principe, ne pose pas de problème d'ordonnancement donc les horloges utilisées pour gérer ce type de problème ne peuvent pas être utiles. De toute façon, un ordre total n'impose pas forcément un ordre causal. On peut décider de délivrer tous les messages dans le

même ordre sur tous les processus mais sans que cela ne respecte les dépendances causales entre les émissions des messages. On peut bien sur avoir un ordre causal en plus d'un ordre total mais ça n'est pas une obligation.

Une solution serait que les processus se mettent d'accord entre eux pour décider dans quel ordre délivrer les messages. Cela peut se faire avec un algorithme de consensus comme on le verra dans le reste du cours ou par un algorithme d'exclusion mutuelle où la "ressource partagée" est ici le droit de diffuser et tant qu'une diffusion n'est pas finie (tous les récepteurs ont acquitté le message diffusé), on ne peut pas en commencer une autre.